INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

ANÁLISE DE *FRAMEWORKS* DE DESENVOLVIMENTO *WEB* PARA A CONVERSÃO DE UMA APLICAÇÃO *DESKTOP*

José António Vieira Lacerda

Escola Superior de Tecnologia e Gestão

INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

José António Vieira Lacerda

Análise de *frameworks* de desenvolvimento *web* para a conversão de uma aplicação *desktop*

Mestrado em
Tecnologia e Gestão de Sistemas de Informação

Trabalho de Projeto efetuado sob a orientação de
Doutor António Miguel Cruz e Doutor Pedro Miguel Castro

Dezembro de 2013

AGRADECIMENTOS

Aos meus orientadores, Doutor Miguel Cruz e Doutor Pedro Castro, pela sua grande disponibilidade, boa vontade, orientação e pelos sábios conselhos que me deram ao longo do trabalho.

Ao João Cruz e ao Jorge Silva, administradores da Macwin, S. A., por terem permitido e facilitado a conciliação do meu tempo de trabalho com as minhas atividades académicas e por terem colocado à minha disposição a infraestrutura da empresa para a realização dos testes.

Por fim, e ao mesmo tempo o mais sentido, um agradecimento muito especial ao meu filho Daniel e à minha esposa Sílvia pelo enorme e constante apoio prestado e pela infindável compreensão para com os meus últimos tempos de ausência.

RESUMO

As aplicações *web* têm vindo a ganhar uma preponderância muito grande, especialmente na área do *software* de gestão. Durante vários anos, este tipo de *software* apenas era disponibilizado aos utilizadores na plataforma *desktop* mas esta tendência tem vindo a ser contrariada. Atualmente, os grandes produtores de *software* de gestão nacionais já disponibilizam aos seus clientes soluções de gestão totalmente baseadas na *web* como uma alternativa viável à sua linha de produtos para o *desktop*. Este processo tem sido lento e difícil atendendo às caraterísticas do *software* e também a alguma resistência por parte dos utilizadores que temem perder a experiência e rapidez de utilização a que estavam habituados.

O processo de criação de uma aplicação *web* pode revelar-se bastante complexo logo na fase de seleção da plataforma de desenvolvimento uma vez que existem várias opções disponíveis (p. ex. .NET, PHP, *Python*, *Ruby*, *Java* e *Perl*) e, para cada uma, identificam-se diversos *frameworks*.

Com este trabalho efetua-se uma análise a algumas das plataformas (.NET, *Java* e PHP) e *frameworks* de desenvolvimento de aplicações *web* e seleciona-se aquela que melhor se adequará para a conversão de um *software* que, no presente, apenas está disponível na plataforma *desktop*. Neste processo, também são analisados *frameworks* mais vocacionados para o desenvolvimento das interfaces do utilizador, com o objetivo de selecionar aquele que permitirá uma melhor experiência de utilização. Esta análise passa pelo estudo das caraterísticas e funcionalidades de cada um dos *frameworks* e pela realização de testes de desempenho às plataformas.

ABSTRACT

Web applications have been gaining great importance, particularly in the development of management software. For several years, this type of software was only available in the desktop but this trend has been inverted. Currently, the largest Portuguese producers of management software already offer to their clients fully web based management solutions as a viable alternative to its desktop product line. This process has been slow and difficult given the characteristics of the software and also due to some resistance from users who fear losing the experience and speed of use to which they were accustomed.

Creating a web application can be a complex task early in the selection phase of the platform development since there are several available options (e.g. .NET, PHP, Python, Ruby, Java and Perl) and, for each, there are several frameworks.

In this project we made an analysis of some platforms (.NET, Java and PHP) and frameworks used to develop web applications and have selected the framework that will more adequate to convert a software which, at present, is only available for desktop. In this process, we also analysed some frameworks that are more oriented to the user interface in order to select the one that will provide a better user experience. In this analysis we made a study of the characteristics and functionalities of each of the frameworks and executed platforms performance testing.

CONTEÚDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ACRÓNIMOS

ACL	Access Control List
ADO	ActiveX Database Objects
AJAX	Asynchronous Javascript and XML
AJAJ	Asynchronous Javascript and JSON
AOP	Aspect Oriented Programming
API	Application Programming Interface
ASP	Ative Server Pages
BMP	Bean Managed Persistent
CAPTCHA	Completely Automated Public Turing Test to Tell Computers and Humans Apart
CGI	Common Gateway Interface
CLR	Common Language Runtime
CLS	Common Language Specification
CMP	Container-Managed Persistence
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update, Delete
CSS	Cascade Style Sheets
CSV	Character Separated Values
CTS	Common Type System
DGCI	Direção Geral de Contribuições e Impostos
DI	Dependency Injection
DOM	Document Object Model
EF	Entity Framework
EJB	Enterprise JavaBean
ERP	Enterprise Resource Planning
ESA	Enterprise Service Abstraction
FAQ	Frequently Asked Questions
FTP	File Transfer Protocol
GPL	General Public License
GUID	Globally Unique Identifier
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTML5	HyperText Markup Language – Version 5
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IoC	Inversion of Control
IVA	Imposto Sobre o Valor Acrescentado
JAAS	Java Authentication and Authorization Service
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JAX-RS	Java API for RESTful Web Services
JAX-WS	Java API for XML Web Services
JCA	Java Cryptography Architecture
JCP	Java Card Platform
JDBC	Java Database Connectivity
JDK	Java Development Kit

JDO	Java Data Objects
JMS	Java Message Service
JPA	Java Persistence API
JRE	Java Runtime Environment
JSF	JavaServer Faces
JSNI	JavaScript Native Interface
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JSR	Java Specification Requests
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LINQ	Language-Integrated Query
LGPL	Lesser General Public License
MRP	Manufacturing Resource Planning
MSIL	Microsoft Intermediate Language
MVC	Model - View - Controller
OGNL	Object Graph Navigation Language
OOP	Object Oriented Programming
ORM	Object-relational mapping
OSGi	Open Services Gateway Initiative
PDA	personal Digital Assistant
PHP	PHP: Hypertext Preprocessor
POJO	Plain Old Java Object
POS	Point of Sales
RAD	Rapid Application development
RBAC	Role-Based Access Control
RIA	Rich Internet Application
RMI	Remot Method Invocation
RSS	Rich Site Summary/Really Simple Syndication
SAML	Security Assertion Markup Language
SF	Spring Framework
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SpEL	Spring Expression Language
SSL	Secure Sockets Layer
STS	Spring Tool Suite
SVG	Scalable Vector Graphics
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WAF	Web Application Framework
WHATWG	Web Hypertext Application Technology Working Group
WML	Wireless Markup Language
WMS	Warehouse Management System
XAML	Extensible Application Markup Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
XUL	XML User Interface Language
ZF2	Zend Framework 2

1. INTRODUÇÃO

O processo de conversão de uma aplicação *desktop* numa aplicação *web* pode ser bastante complexo logo na fase de seleção da plataforma/*framework* e das ferramentas de desenvolvimento. Se, ao nível das linguagens de programação, as soluções são praticamente as mesmas, atendendo a que a maioria das linguagens existentes para *desktop* também permitem o desenvolvimento de aplicações *web*, o mesmo não se verifica quanto aos *frameworks* de desenvolvimento e às técnicas a utilizar. Uma breve pesquisa permite identificar centenas de *frameworks*, cada um deles com as suas caraterísticas e propondo abordagens totalmente diferentes ao desenvolvimento. Outro fator que adiciona complexidade no momento de seleção das tecnologias a utilizar é a necessidade de se utilizarem várias ferramentas, linguagens de programação e técnicas para se desenvolver uma aplicação *web*. Enquanto que uma aplicação *desktop* é desenvolvida, na maioria das vezes, utilizando apenas uma linguagem e um só ambiente de desenvolvimento, uma aplicação *web* envolve, tipicamente, a utilização de diversas linguagens. Com efeito, a implementação de soluções *web* envolve, regra geral, a escrita de código HTML, código a ser executado no cliente e código a ser executado no servidor, sendo que, para cada tipo de código a desenvolver dispomos de várias possibilidades. A todas estas alternativas ainda é possível adicionar mais algumas relacionadas com o desenvolvimento da parte visual das aplicações. A interface do utilizador pode ser desenvolvida utilizando, apenas, código HTML dinâmico, utilizando extensões (*plugins*) existentes para os navegadores ou, ainda, utilizando uma das inúmeras bibliotecas gráficas existentes para o desenvolvimento de aplicações ricas.

O principal objetivo deste trabalho é selecionar o *framework* e o conjunto de tecnologias a utilizar num processo de conversão para a plataforma *web* de um *software* atualmente existente para a plataforma *desktop*. Neste sentido efetuamos uma análise a vários *frameworks* de desenvolvimento de aplicações *web*, cujos critérios de seleção foram, essencialmente, os seguintes:

- Selecionar *frameworks* sustentados em plataformas com uma grande base de instalação e utilização;
- Para cada plataforma, selecionar os *frameworks* que utilizam abordagens diferentes para o desenvolvimento das aplicações.

Depois de ser feita uma análise inicial, pretendemos indicar, para cada plataforma, qual o *framework* mais apropriado para o desenvolvimento do novo *software*. Este processo de seleção será baseado, fundamentalmente, num conjunto de parâmetros como, por exemplo, funcionalidades disponibilizadas, desempenho e suporte. Além disso, ainda pretendemos realizar testes ao desempenho e comportamento de cada uma das plataformas com o intuito de identificar aquela que melhor se adequa às caraterísticas do *software* e à realidade da empresa.

O presente projeto encontra-se dividido nos seguintes capítulos:

- Capítulo 1: Capítulo atual onde é feita uma introdução/apresentação do projeto;
- Capitulo 2: Neste capítulo será feita uma apresentação do *software* GM *Macwin*, o qual se pretende que seja, futuramente, desenvolvido como uma aplicação *web*. Faremos uma apresentação das caraterísticas genéricas da aplicação e, de seguida, descreveremos as principais funcionalidades de cada um dos módulos que compõem a aplicação atual;
- Capítulo 3: Apresentação de conceitos relacionados com o desenvolvimento de aplicações *web*. Tratam-se de conceitos que serão referidos ao longo do relatório de projeto e aos quais, dada a sua importância, é feita uma análise mais aprofundada;
- Capítulo 4: Análise às plataformas selecionadas (.NET, *Java* e PHP) e a alguns dos *frameworks* disponíveis em cada uma delas;
- Capítulo 5: O objetivo deste capítulo é indicar, para cada plataforma, qual será o *framework* que melhor se adequa ao desenvolvimento da aplicação descrita no capítulo 2. Dentro de cada plataforma serão comparadas e valorizadas as caraterísticas de cada um dos *frameworks* descritos no capítulo anterior;
- Capítulo 6: Neste capítulo serão efetuados testes de desempenho às plataformas selecionadas. Os testes incidirão, principalmente, sobre a interação das plataformas com uma base de dados *Microsoft SQL Server* e sobre o processamento de diversos tipos de listas;
- Capítulo 7: O objetivo deste capítulo é identificar qual a tecnologia mais adequada para desenvolver a componente visual da aplicação e, dentro da tecnologia recomendada, analisar alguns *frameworks* e selecionar o que aparentar ser o mais adequado;
- Capítulo 8: Este será o último capítulo do relatório de projeto e apresentará as conclusões finais sobre o mesmo.

2. APRESENTAÇÃO DO *SOFTWARE* GM *MACWIN*

O *software* GM *Macwin* é um sistema integrado de gestão, vocacionado para médias e grandes empresas, que se carateriza por ser extremamente modular, escalável e adaptável às caraterísticas das organizações. Atualmente é utilizado por três tipos distintos de empresas:

- Indústria têxtil e de vestuário;
- Fabricação de armas de caça e de defesa;
- Comércio eletrónico.

O sistema é transversal a todas as áreas da empresa, disponibilizando módulos para, entre outros, a gestão da produção, recursos humanos, gestão da qualidade, contabilidade, imobilizado e tesouraria. Na tabela 2.1 apresentam-se os módulos que constituem o *software*:

Módulos comuns	Recursos Humanos (inclui submódulos para gestão da formação e avaliação do desempenho) Contabilidade Tesouraria Imobilizado Faturação *Stocks* Manutenção de Equipamentos
Módulos da área têxtil	Gestão da Produção Laboratório Tinturaria Tecelagem MRP POS
Módulos da área da fabricação de armas	Gestão da Produção Qualidade Assistência Após Venda Gestão de Pedidos de Compra Gestão da Expedição Controlo de Gestão
Módulos da área do comércio eletrónico	Gestão de traduções Gestão de rotulagens Processamento de encomendas Apoio ao cliente Marketing (inclui submódulos de gestão de conteúdos, inquéritos e *newsletters*) WMS (*Warehouse Management System*)

Tabela 2.1 – Módulos da aplicação GM *Macwin*

O processo de conversão do *software* para a plataforma *web* decorrerá em duas fases. Na primeira fase serão convertidos os módulos comuns (com exceção do módulo de Gestão da Manutenção dos Equipamentos) e os módulos da área do comércio eletrónico. Numa fase posterior, quando a primeira fase estiver concluída e a aplicação *web* estiver consolidada, procederemos à conversão dos restantes módulos.

O ERP *Macwin*, para além dos módulos apresentados na tabela 2.1, possui um vasto conjunto de caraterísticas disponíveis em todos os módulos, nomeadamente:

- Multiempresa: O sistema suporta a utilização de múltiplas empresas, de forma totalmente independente e isolada;
- Multidivisa: Todos os documentos, lançamentos ou outros tipos de transações podem ser registados utilizando qualquer uma das divisas que estejam registadas na aplicação;
- Multi-Idioma: Nas principais tabelas da aplicação é possível definir diversos campos em qualquer um dos idiomas definidos na aplicação. Por exemplo, para cada um dos registos de artigos, o utilizador tem a possibilidade de definir as respetivas descrições em vários idiomas;
- Multiutilizador: A aplicação permite o registo e controlo dos acessos por utilizador;
- Configuração de acesso a módulos, menus, janelas e campos por utilizador ou perfil. É possível definir, por utilizador ou perfil de utilizadores, a que opções, janelas ou campos é que tem acesso;
- Configuração de permissões de inserção, alteração e eliminação em todas as tabelas por utilizador ou perfil;
- Criação de filtros personalizados para consulta das listas em todas as tabelas;
- Integração com o *Microsoft Excel*. Todas as listagens da aplicação são exportáveis em formato *Excel*;
- Filtros de pesquisa personalizados: Cada utilizador poderá definir os seus critérios de filtragem das listas que são apresentadas na aplicação;
- Histórico de operações nos registos: O sistema regista todas as operações realizadas pelos utilizadores em todas as tabelas. Em áreas mais sensíveis (por exemplo, fichas de recursos humanos, documentos e lançamentos contabilísticos) é possível registar qual foi a informação alterada em cada operação;
- Sistema de atualização automática: O *software* possui um sistema de atualização que permite que o mesmo esteja sempre atualizado com as versões mais recentes. Todas estas operações processam-se sem a intervenção dos utilizadores;
- Sistema *TypeAhead* que permite, em campos associados a tabela auxiliares, facilitar o trabalho através do preenchimento ou sugestão de valores;
- *Status* das tabelas: Todas as tabelas possuem um campo onde se poderá definir o estado dos registos. Desta forma, é possível definir, por exemplo, que o registo X da tabela Y não estará disponível para utilização a partir do dia Z;
- Painel de controlo: Trata-se de um sistema interno de comunicação entre os utilizadores da aplicação. Permite enviar ou gerar mensagens para os utilizadores e registar, de forma manual ou automática, qualquer informação relevante para a organização;

- Desenvolvido utilizando as linguagens de programação *Omnis Studio*[1], *Microsoft Visual C++* e *Java*;
- Utilização da base de dados *Microsoft SQL Server* 2000, 2005, 2008 ou 2012 (algumas das funcionalidades da área do comércio eletrónico exigem, pelo menos, a versão 2008);
- *Software* certificado pela DGCI com o n.º 690[2].

Todos estes módulos e funcionalidades estão disponíveis para utilização imediata por todas as organizações que optem por utilizar este sistema. Adicionalmente, a *Macwin, S. A.* desenvolve soluções à medida, assentes na plataforma atrás descrita, com o objetivo de adequar ainda mais as caraterísticas do *software* às reais necessidades dos utilizadores.

Nas secções seguintes procederemos a uma análise mais detalhada dos diversos módulos que serão objeto de conversão para a plataforma *web*, apresentando as caraterísticas mais importantes.

2.1. RECURSOS HUMANOS

O GM Recursos Humanos é a solução desenvolvida pela *Macwin* para a gestão do capital humano das empresas. As funcionalidades deste módulo podem ser agrupadas nos seguintes submódulos:

- Gestão contratual;
- Gestão de presenças;
- Processamento de vencimentos;
- Gestão da formação;
- Sistema de gestão e avaliação do desempenho;
- Mapas legais e de gestão.

Nas opções da gestão contratual, os utilizadores podem gerir todos os aspetos relacionados com os contratos dos colaboradores com as organizações. As principais opções deste módulo são as seguintes:

- Ficha de funcionário para registo de todos os dados associados ao funcionário;
- Ficha de contrato para registo de todos os dados relevantes para o contrato;
- Cadastro de alterações. Regista todas as alterações efetuadas nas fichas de funcionários e contratos possibilitando a reconstituição das fichas a uma determinada data. Esta funcionalidade é extremamente importante uma vez que

[1] http://www.tigerlogic.com/tigerlogic/omnis/products/studio/index.jsp, Março 2013
[2] http://www.portaldasfinancas.gov.pt/pt/Out/consultaProgCertificadosM24.action, Março 2013

muitos dos mapas legais da área dos recursos humanos são baseados em dados históricos;
- Alertas. Permite emitir alertas para, por exemplo, finais de contratos, finais de períodos experimentais, aniversários, datas de mudanças de categorias ou de regimes de segurança social.

A gestão de presenças engloba diversas opções relacionadas com o registo e controlo de presenças, faltas e trabalho extraordinário, nomeadamente:

- Definição e atribuição de horários;
- Importação de marcas reais registadas em relógios de ponto. À medida que os funcionários vão registando as entradas e saídas nos relógios de ponto, estes vão guardando esta informação internamente. Posteriormente, o sistema *Macwin* acede ao relógio para obter os dados das picagens;
- Processamento de marcas reais importadas para criação de faltas e de horas extraordinárias;
- Validação de faltas e horas extra geradas no processamento. Também permite a inserção manual de faltas e horas extra para os casos em que o processamento automático não tenha gerado os registos devidamente ou para as organizações que não possuem sistemas de controlo de entradas e saídas;
- Gestão de férias. Permite a marcação de dias de férias e o registo do gozo das mesmas.

O grupo do processamento de vencimentos engloba um conjunto de funcionalidades relacionadas com o processamento, pagamento e contabilização dos salários aos colaboradores. Neste grupo identificam-se as seguintes funcionalidades:

- Processamento de vencimentos de forma manual ou automática. Inclusão automática das faltas e horas extra geradas no módulo da gestão de presenças. Processamento de subsídios de férias e de natal em duodécimos ou integral. Tratamento de subsídios de turno, abonos para falhas e isenção de horários;
- Processamento de retroativos para acerto de valores previamente processados e encerrados;
- Impressão e envio de recibos de vencimento;
- Pagamento dos recibos (por numerário, cheque ou por transferência bancaria, com geração do respetivo ficheiro de ordens de transferência);
- Cálculo das previsões de subsídios de férias e de período de férias para o ano seguinte;
- Imputação automática dos processamentos e dos cálculos de previsões.

O submódulo da gestão da formação permite controlar todos os aspetos relacionados com a formação profissional contínua que as organizações são obrigadas a ministrar aos seus colaboradores. Identificam-se as seguintes funcionalidades:

- Tratamento das necessidades individuais de formação. Tratam-se de inquéritos feitos aos funcionários para avaliar as necessidades de formação que estes sentem;
- Registo de cursos e áreas de formação. Nestas opções registam-se todas as informações relacionadas com os cursos e áreas de formação ministradas nas empresas. Esta informação será utilizada, numa fase posterior, para o preenchimento de formulários legais;
- Planeamento da formação. Esta é uma das principais funcionalidades do módulo de gestão da formação. Aqui os utilizadores poderão calendarizar as diversas ações de formação de cada curso, indicar os participantes assim como os formadores. Também é aqui que se definem outras caraterísticas das ações, tais como, se concedem direito a certificado de frequência ou de aproveitamento, os horários em que ocorrerão e os custos;
- Registo das execuções das ações de formação. À medida que as ações de formação vão decorrendo, é necessário registar estas evoluções e quem a elas faltou. É nesta opção que se procede ao registo desta informação;
- Emissão de certificados. Após a conclusão das ações de formação, normalmente, procede-se à emissão de certificados (que podem ser dos mais variados tipos);
- Controlo dos créditos de formação. Esta opção é extremamente importante uma vez que a formação contínua dos funcionários é regulada por lei e obedece a um conjunto rígido de princípios e regras. A título de exemplo, cada funcionário tem direito a receber, pelo menos, 35 horas de formação por ano e caso esta quantidade de formação não seja ministrada num ano, transitará e acumulará para o ano seguinte. De forma semelhante, se um funcionário, num determinado período de tempo, receber mais do que 35 horas, o excedente pode ser deduzido às horas do ano seguinte. As opções de controlo dos créditos de formação permitem gerir estas regras de forma simples e segura.

O sistema de gestão e avaliação do desempenho é o submódulo mais recente da área dos recursos humanos. Nesta área, os utilizadores poderão gerir todos os aspetos relacionados com a avaliação do desempenho e das competências dos funcionários. Destacam-se as seguintes opções:

- Definição dos *clusters* de competências e dos pesos por famílias de funções, para posterior avaliação de cada um dos funcionários sujeitos a avaliação;
- Planeamento e definição dos objetivos. Trata-se de uma operação que deve ser feita de forma individual para cada funcionário;
- Registo das avaliações e autoavaliações periódicas;
- Emissão dos documentos de avaliação.

O último grupo de funcionalidades dos recursos humanos está relacionado com as obrigações legais a que as organizações estão sujeitas, nomeadamente ao nível da prestação de informação às entidades legais. Os principais mapas que a aplicação permite emitir são os seguintes:

- Quadro de pessoal;
- Balanço social;
- Relatório único (anexos relacionados com a gestão de recursos humanos);
- Mapa de seguros;
- Mapa de sindicatos;
- Registo de trabalho extraordinário;
- Declaração mensal de remunerações;
- Modelo 10;
- Declaração de rendimentos para IRS;
- Diversos inquéritos legais (p. ex., índice do custo do trabalho e inquérito complementar à estrutura dos ganhos).

2.2. CONTABILIDADE

Este foi o primeiro módulo a ser desenvolvido pela *Macwin*. Trata-se de um sistema que permite gerir, de forma simples e eficaz, todos os requisitos contabilísticos e legais das empresas. Ao estar integrada na mesma plataforma que os restantes módulos, permite uma elevada integração e automatismo de processos. Por exemplo, sempre que se emita uma fatura ou se calculem as amortizações do imobilizado, é possível refletir, de forma automática, essas operações na contabilidade através da geração dos lançamentos contabilísticos adequados. Para além de dar resposta a todos os requisitos legais a que as organizações estão obrigadas, o módulo GM Contabilidade também permite uma exploração e controlo mais aprofundados disponibilizando funcionalidades nas áreas da contabilidade de gestão e orçamental. Na lista seguinte apresentam-se as principais funcionalidades desse módulo:

- Plano anual de contabilidade financeira e analítica. Os planos legais (e também o analítico) são definidos numa base anual. É possível ter, em simultâneo, planos abertos para anos distintos e efetuar lançamentos em anos diferentes;
- Planos anuais alternativos. Para além dos planos contabilísticos legais definidos pelo Sistema de Normalização Contabilística, a aplicação também permite que se criem planos alternativos que se adequem melhor às caraterísticas das empresas. Poderão ser criados tantos planos alternativos quanto os que forem necessários;
- Planos orçamentais. Os planos orçamentais permitem fazer previsões para cada uma das contas do plano oficial e, posteriormente, efetuar análises comparativas entre o previsto e o real;
- Geração manual de lançamentos contabilísticos. Esta é a janela principal do módulo da contabilidade. Aqui, o utilizador pode criar, alterar ou eliminar os lançamentos que estão na base de todo o sistema de contabilidade. Para facilitar o trabalho, a janela possui diversas funcionalidades (das quais se destacam as máscaras de lançamentos e a definição de *layouts* por tipo de lançamento) que permitem um elevado desempenho na sua utilização;

- Reflexões na contabilidade analítica. Os lançamentos na contabilidade analítica podem ser feitos manualmente ou serem gerados, automaticamente, pela aplicação quando o utilizador grava um lançamento na contabilidade financeira. Também existe uma outra opção para refletir todos ou um conjunto de lançamentos na analítica. Esta última opção é útil quando, por qualquer motivo, é necessário alterar a forma como as contas da contabilidade geral refletem na analítica. Com a funcionalidade de recálculo das reflexões, é possível efetuar uma nova reflexão na analítica já com as novas configurações;
- Reflexões nos planos alternativos. Os lançamentos nos planos alternativos seguem os mesmos princípios dos lançamentos na analítica. Podem ser gerados automaticamente, criados manualmente e recalculados através de uma opção disponibilizada para o efeito;
- Apuramentos de IVA. A aplicação permite calcular, de forma automática, os lançamentos de apuramento do IVA e auxiliar no preenchimento da respetiva declaração periódica;
- Apuramentos de resultados. Nesta opção procede-se à geração dos lançamentos para apuramento do resultado líquido do exercício e dos diferentes tipos de resultados intermédios;
- Consolidação de contas. Esta é a funcionalidade mais recente do módulo de contabilidade. Com ela, é possível agrupar as contabilidades de diversas empresas e tratá-las como se se tratassem de uma só empresa;
- Mapas legais e de gestão. O GM Contabilidade permite dar resposta às obrigações legais de *reporting* financeiro emitindo praticamente todos os mapas legais e uma quantidade considerável de mapas de gestão. Com estes mapas poderemos fazer inúmeras análises numa perspetiva de gestão bem como dar resposta às obrigações legais a que as empresas estão obrigadas.

2.3. TESOURARIA

O módulo de tesouraria, embora sendo identificado de forma isolada, está intimamente relacionado com o módulo da contabilidade. Aqui tratam-se todos os aspetos relacionados com os fluxos financeiros das organizações. As principais funcionalidades ao dispor do utilizador são as seguintes:

- Emissão de recebimentos. É no módulo de tesouraria que os utilizadores podem registar os valores recebidos de entidades terceiras a título de recebimento/liquidação de pendentes;
- Planeamento e emissão de pagamentos. Os pagamentos a terceiros podem ser feitos de duas formas: individualmente ou em lote. Os pagamentos em lote, para além de permitirem uma grande celeridade na seleção dos pendentes a liquidar, também permitem planear os pagamentos a efetuar num determinado período de tempo;

- Gestão de pendentes titulados. Tanto ao nível dos pagamentos como ao nível dos recebimentos, o sistema permite efetuar uma gestão de pendentes titulados (que, na sua maior parte, se tratam das chamadas "Letras"), contemplando as opções de criação de pendentes titulados, pagamento e reforma;
- Reconciliação bancária. Esta funcionalidade possibilita a conferência dos movimentos bancários registados no *software* com os movimentos constantes nos extratos bancários;
- Mapas de gestão. Dos diversos mapas de gestão que estão disponíveis no módulo de tesouraria, destacam-se os mapas de previsão de pagamentos, onde se pode analisar os fluxos financeiros de saída em intervalos regulares de tempo e o mapa da idade de saldos, que apresenta os valores pendentes agrupados pela idade.

2.4. IMOBILIZADO

O GM Imobilizado permite gerir todos processos relacionados com os ativos fixos das organizações, independentemente da sua natureza. Das diversas funcionalidades disponíveis neste módulo e que serão apresentadas a seguir, uma das mais importantes e, provavelmente, a mais utilizada é o processamento das depreciações (ou amortizações segundo a nomenclatura anteriormente em vigor). Embora este módulo possa ser utilizado em todo o tipo de organizações, apresenta algumas funcionalidades e caraterísticas que o tornam especialmente vocacionado para empresas que pretendam efetuar um controlo numa base mensal.

- Ficha de bens. Permite gerir os registos dos bens da empresa que fazem parte do imobilizado. Nesta janela, o utilizador tem a possibilidade de consultar toda a informação que seja relevante para cada um dos bens;
- Decomposição de bens. Por vezes as empresas necessitam de, a partir de um bem, dar origem a diversos bens diferentes, num processo a que se chama de Decomposição. Por exemplo, decompor um computador pessoal num monitor, uma CPU e um teclado/rato. Nesta opção, o utilizador pode decompor um bem em vários, decompondo, também, toda a informação económico-financeira a ele associado;
- Saídas de bens. Remove um bem da lista dos ativos fixos da organização. As saídas podem ser resultantes, por exemplo, de roubos, extravios, doação, venda, obsolescência ou cisão da empresa, sendo que cada um dos motivos apontados tem efeitos diferentes ao nível da contabilidade;
- Grandes reparações. Quando um bem é objeto de uma grande reparação que lhe altere o valor real, essa operação deve ser registada no sistema. É nesta opção que se regista este tipo de intervenção;
- Processamento de depreciações. Esta será, por ventura, uma das opções mais importantes do módulo do imobilizado, já que tudo o que se faz nas outras opções terá reflexo no processamento das depreciações. Este cálculo pode ser feito numa

base mensal ou anual e, caso se opte pelo cálculo mensal, o sistema fará os cálculos numa base de 11 e 12 meses;

- Reavaliações dos ativos fixos. Permite registar as reavaliações a que os bens foram sujeitos, quer se tratem de reavaliações livres ou legais. Este registo permite, ainda, que o sistema processe de forma adequada os valores resultantes desta operação;
- Processamentos de imparidades, reversões e revalorizações. A adoção das novas normas contabilísticas impostas pelo Sistema de Normalização Contabilística introduziu novos mecanismos que permitem ajustar os valores contabilísticos dos bens aos seus reais valores. As imparidades, reversões e reavaliações são formas de registar estas diferenças;
- Cálculos de previsões de depreciações. Permite calcular previsões de amortizações para um conjunto de anos e simular diversos cenários de custos, alterando taxas das depreciações;
- Mapas legais e de gestão. Tal como acontece nos outros módulos, também aqui os utilizadores têm à disposição um conjunto muito vasto de mapas de gestão, bem como mapas que dão resposta às obrigações legais.

2.5. FATURAÇÃO

No módulo de faturação são tratados todos os aspetos relacionados com os processos de vendas, que vão desde a emissão de documentos até ao preenchimento das diversas obrigações legais.

- Ficha de entidades. Gere toda a informação relevante para se manter um ficheiro de entidades. Trata-se de informação de todos os tipos de entidades com quem a organização se relaciona;
- Geração de documentos de diversos tipos. Emissão de todo o tipo de documentos que uma empresa necessita para desenvolver a sua atividade. Estes documentos podem ser, por exemplo, orçamentos, encomendas, guias de remessa, faturas, notas de débito e crédito. É possível converter documentos de um tipo noutro tipo diferente (por exemplo, converter encomendas em guias de remessa) poupando-se, desta forma, muito tempo na elaboração dos documentos;
- Faturação automática de guias de remessa. Nesta opção convertem-se, de forma automática, guias de remessa em faturas;
- Processamento de avenças. Opção vocacionada para empresas que processam, de forma sistemática e periódica, sempre o mesmo tipo de documento e com o mesmo valor;
- Gestão de vendedores e de comissões. Opção útil para organizações que pretendam controlar a faturação por vendedor e gerir todos os aspetos relacionados com as comissões. Estas podem ser calculadas com base num valor faturado ou com base no valor recebido;

- Gestão de preços. A aplicação permite definir preços dos produtos e mercadorias por mercado, zona geográfica, canal de distribuição ou ainda por período de tempo;
- Mapas legais e de gestão. Nos mapas de gestão destaca-se uma grande variedade de mapas de venda, com vários critérios de filtragem e agrupamentos. Ao nível dos mapas legais, destaca-se a geração automática do ficheiro SAF-T PT.

2.6. *STOCKS*

O módulo de *stocks* permite gerir e controlar todos os aspetos relacionados com a gestão dos ativos circulantes da empresa. Através das opções deste menu é possível emitir todo o tipo de documentos assim como analisar os *stocks* das organizações sob as mais diversas perspetivas (p. ex. quantidades disponíveis, localizações, valorizações ou prazos de validade). As opções mais importantes da gestão de *stocks* são apresentadas na lista seguinte:

- Ficha de artigo. Opção onde se efetua a manutenção dos registos dos artigos com os quais a empresa trabalha. Aqui consta toda a informação relevante sobre os artigos;
- Análise de *stocks* para sugestão de compras. Através desta opção os utilizadores obtêm uma lista com sugestões de quantidades a encomendar aos fornecedores. Estas quantidades são calculadas com base na evolução das vendas dos últimos tempos (normalmente 90, 120 ou 180 dias);
- Geração automática de encomendas. Utilizando a informação obtida na análise de *stocks* para sugestões de compras é possível gerar, de forma automática, as encomendas aos fornecedores. Estes documentos têm em conta diversos fatores, tais como os prazos de entrega e os preços praticados pelos fornecedores;
- Gestão de artigos à consignação. O *software* possibilita uma gestão dos artigos que, eventualmente, estejam na empresa à consignação. É possível saber, a qualquer momento, quais os artigos que se encontram à consignação assim como saber as quantidades vendidas destes artigos num determinado período de tempo;
- Mapas de gestão. Os mapas de gestão de *stocks* possibilitam uma análise e controlo total sobre os artigos. Os utilizadores têm a possibilidade de analisar toda a informação referente aos *stocks* sob as mais variadas perspetivas (p. ex. quantidades disponíveis, valorizações ou quebras).

2.7. GESTÃO DE TRADUÇÕES

A gestão de traduções é um módulo específico da área do *e-commerce*. Uma vez que os *sites* podem ser acedidos de qualquer parte do mundo, é importante que as informações sejam

apresentadas no maior número possível de idiomas. Neste módulo faz-se a gestão de todos os aspetos relacionados com a tradução dos textos a apresentar nos *sites* e documentos enviados aos clientes.

- Definição de tradutores e revisores;
- Pedidos de tradução;
- Revisão de traduções efetuadas.

2.8. GESTÃO DE ROTULAGENS

Aqui geram-se os processos e a informação relacionada com a rotulagem dos produtos vendidos nos *sites*. Alguns produtos, pela sua natureza ou composição, necessitam de uma rotulagem especial ou de um registo de autorização de comercialização. Tudo isto é tratado nas opções apresentadas a seguir:

- Manutenção dos ingredientes e de todas as suas caraterísticas (descrições detalhadas, instruções de utilização, efeitos positivos e negativos, possibilidades de combinação com outros ingredientes, composição, FAQ ou referências, entre outras);
- Emissão de registos de comercialização de produtos;
- Definição e impressão de etiquetas.

2.9. PROCESSAMENTO DE ENCOMENDAS

A gestão de encomendas (de clientes e a fornecedores) é um processo que pode ser relativamente complexo, especialmente em empresas de venda de produtos *on-line*. O módulo de processamento de encomenda contempla diversas opções que acompanham as diferentes fases de um processo de encomenda e receção de mercadorias.

- Emissão e controlo de encomendas a fornecedores. Esta opção é usada para a emissão de encomendas a fornecedores, para o acompanhamento dos respetivos níveis de liquidação e para a emissão de documentos com erros (é frequente que as mercadorias recebidas não coincidam com o que foi encomendado);
- Receção de encomendas e colocação dos artigos no armazém. Opções relacionadas com a receção dos artigos encomendados e os processos de colocação dos mesmos nas localizações dos armazéns. Para cada artigo rececionado, o sistema pode recomendar uma localização onde deve ser colocado;
- Validação de encomendas de clientes. As encomendas que os clientes fazem nos *sites* necessitam de uma validação antes de serem processadas. Este processo de validação consiste na verificação de alguns requisitos que, quando aprovados,

permitem que a encomenda seja assinalada para processamento pelo armazém (p. ex., uma encomenda apenas pode ser processada caso a morada seja válida). As opções de validação de encomendas tratam de todos os aspetos relacionados com este processo;

- Processamento de encomendas de clientes (*picking*, embalagem, pesagem e envio). O processamento de encomendas de clientes é um processo que envolve várias atividades que vão desde a recolha dos artigos no armazém (processo usualmente chamado de *picking*) até à expedição das encomendas pelas transportadoras. Cada uma das atividades tem as suas opções e janelas específicas;
- Tratamento de trocas, devoluções e não entregas. Este item agrupa as funcionalidades relacionadas com o tratamento das encomendas que não conseguiram ser entregues pelas transportadoras e, também, das devoluções e trocas efetuadas pelos clientes, após terem recebido os produtos encomendados.

2.10. APOIO AO CLIENTE

O módulo do apoio ao cliente engloba, essencialmente, funcionalidades relacionadas com os contactos estabelecidos entre os clientes e as empresas, seja qual for o meio utilizado para o efeito. Com este sistema, as empresas conseguem saber toda a informação relevante para a área do apoio ao cliente. Por exemplo, sabe-se quantos pedidos tratou cada funcionário e o n.º de registos abertos por país, artigo ou data.

- Ficha de utilizadores. Nesta janela definem-se todos os funcionários que trabalham na área do apoio ao cliente e para os quais poderão ser encaminhados os contactos iniciais dos clientes;
- Registo dos contactos dos clientes. Engloba um conjunto de opções onde os utilizadores registam todas a iterações com os clientes. Normalmente, este processo inicia-se com um *e-mail* enviado pelo cliente, o qual dará origem a um registo na tabela de contactos. A partir daí, esse contacto será encaminhado para o funcionário adequado, o qual tratará do assunto em causa.

2.11. MARKETING

Este módulo contém uma grande quantidade de funcionalidades, todas elas relacionadas com a área do marketing e promoção de vendas. A lista apresentada a seguir é um resumo das funcionalidades disponíveis. Como se trata de uma área muito dinâmica e na qual estão constantemente a surgir novas ideias e iniciativas, o módulo do marketing está, permanentemente, em desenvolvimento e a receber novas funcionalidades. Na prática, cada nova ação de marketing de uma empresa exige sempre um desenvolvimento à

medida ou um ajuste a algo já existente. Por este motivo, a lista apresentada a seguir poderá variar muito de empresa para empresa.

- Sistema de gestão de cupões de desconto: Permite gerir todos os aspetos relacionados com os cupões de desconto. Os cupões de desconto são *vouchers* que são oferecidos aos clientes, pelos mais variados motivos e que, quando utilizados numa encomenda, traduzem-se num benefício para estes;
- Gestão de conteúdos: Esta opção é constituída por várias janelas nas quais é possível definir os conteúdos que serão apresentados nos *sites*. Toda a informação apresentada nos *sites* é gerida pelo sistema *Macwin* sendo que os *sites* (que poderão ser desenvolvidos em qualquer plataforma) tratam-se apenas de uma "camada" para apresentação dos dados;
- Gestão de inquéritos: É frequente que as empresas de vendas *on-line* façam inquéritos aos seus clientes. Neste submódulo é possível definir as questões a colocar aos clientes, o período em que estas questões estarão visíveis e, ainda, fazer estatísticas das respostas obtidas;
- Gestão de *newsletters*: As *newsletters* são um meio muito eficiente de potenciar as vendas e enviar informações aos clientes. Com as funcionalidades da gestão de *newsletters* é possível controlar todo este processo podendo-se definir, por exemplo, os conteúdos, datas de envio ou grupos de clientes abrangidos;
- Sistema de *Refer-a-friend (*RaF*)*: O sistema RaF está muito relacionado com o advento das redes sociais e permite que os clientes obtenham benefícios monetários em função de determinado tipo de ações que vão tomando ao longo do tempo. Estas ações, geralmente, são convites que os clientes fazem a outras pessoas para que se tornem "amigos" das páginas das empresas. Por cada convite que dê origem a um "amigo" ou que se torne cliente, quem fez o convite receberá um determinado valor monetário. Se, por sua vez, esse convite para se tornar "amigo" der origem a uma venda, então a compensação recebida será maior. As funcionalidades relacionadas com o RaF controlam todas estas atividades e relacionamentos;
- Gestão de ofertas: Estas funcionalidades permitem definir e controlar a atribuição de ofertas aos clientes mediante o cumprimento de um determinado número de requisitos. Por exemplo, é possível definir que todos os clientes que façam compras superiores a 50€, do país IT receberão o produto Y (enquanto houver quantidade em armazém).

2.12. WMS (*WAREHOUSE MANAGEMENT SYSTEM*)

O módulo de WMS trabalha, de forma integrada com os módulos de gestão de *stocks* e de processamento de encomendas. Sempre que um utilizador necessitar de uma localização para lá colocar artigos ou necessitar de recolher os artigos de um conjunto de encomendas, o WMS fornecerá toda a informação necessária. Grande parte das

funcionalidades relacionadas com o WMS pode ser feita em PC ou através de PDA. Na lista seguinte apresentam-se mais algumas funcionalidades relevantes do WMS:

- Manutenção de armazéns e localizações. Permite definir quais os armazéns da empresa e, dentro de cada um deles, quais as respetivas localizações;
- Otimização de localizações. Esta funcionalidade analisa todas as localizações dos armazéns e sugere alterações nas colocações dos artigos com o objetivo de otimizar o nível de ocupação das localizações e diminuir o tempo de *picking*;
- Mapas de gestão. Emissão dos mais variados mapas relacionados com a gestão dos armazéns. São exemplos destes mapas, diversos tipos de inventários, com agrupamentos por vários campos e mapas de ocupação de localizações.

3. CONCEITOS DE DESENVOLVIMENTO DE APLICAÇÕES *WEB*

3.1. *WEB DEVELOPMENT*

O *Web Development* é um termo utilizado para referir as diversas atividades relacionadas com a criação e manutenção de *web sites*. As atividades realizadas no âmbito do desenvolvimento *web* são, essencialmente, o desenho, a programação e a publicação. O termo *Web Development* pode ser aplicado tanto para a criação de uma simples página estática como para a criação de uma complexa aplicação que será acedida por milhares de utilizadores em todo o mundo. Normalmente, o desenvolvimento *web* pode ser dividido em dois grandes grupos:

- *Client Side*
- *Server Side*

O *Client Side* corresponde à parte dos *sites* que é executado do lado do cliente (geralmente um *browser*). Enquadram-se neste grupo as tarefas mais ligadas com a apresentação das páginas, sendo que a maioria do desenvolvimento nesta área é feito com recurso à linguagem *JavaScript* e sendo também frequente a utilização de ferramentas de desenvolvimento para o desenho da interface gráfica e posterior geração do respetivo código HTML. Também se enquadram nesta categoria outras técnicas de apresentação, salientando-se o *Adobe Flex*, *Microsoft Silverlight* e *JavaFX*.

O *Server Side*, pelo contrário, engloba todas as tarefas relacionadas com a parte do *site* que é executada no lado do servidor. Trata-se, fundamentalmente, do desenvolvimento de código que será executado para dar resposta a um pedido realizado por um cliente. ASP, ASP.NET, PHP, *Java*, *Python* e *Perl*, são algumas das linguagens de programação e plataformas utilizadas no desenvolvimento do código que será executado do lado do servidor.

O termo *Web Development* surge, muitas vezes, associado ao conceito de *Web Design*. Embora se tratem de conceitos relacionados (sendo frequente, em empresas de menor dimensão, a mesma pessoa desempenhar funções em ambas as áreas), na prática são conceitos totalmente diferentes. O *Web Design* reporta-se, exclusivamente, aos aspetos visuais da aplicação, não tendo, portanto, qualquer responsabilidade ao nível da programação. A principal responsabilidade de um *Web Designer* é definir a forma como um *Web Site* será apresentado ao utilizador e como este interage com a aplicação, não tendo qualquer responsabilidade quanto à tecnologia que suportará o projeto. Isto significa que, do lado do cliente, poderemos ter a intervenção de um *Web Designer* e de um *Web Developer*, o primeiro tratando dos aspetos visuais e o segundo desenvolvendo o código que os suportarão e que será executado no lado do cliente ou do servidor. Este código pode realizar tarefas tão díspares como a apresentação ou ocultação de elementos das páginas, a interação com os sistemas de bases de dados para obtenção de informação ou para efetuar a sua manutenção, a validação e filtragem dos dados inseridos pelos

utilizadores, o tratamento de questões relacionadas com a internacionalização e localização das aplicações, entre outras.

3.2. *SERVER-SIDE SCRIPTING*

O *Server Side Scripting* (ou *Server Side Coding*) é uma tecnologia que permite a geração de código HTML específico para um determinado pedido. Normalmente, quando um *browser* solicita uma página ao servidor *web*, este envia-a sem qualquer modificação. No entanto, se a página em causa possuir alguma indicação de que deve ser processada antes de ser enviada (indicação feita através de *tags* pré-definidas que contêm o código a ser executado), o servidor processará a página em causa, o que dará origem a um código HTML customizado e específico para o pedido. Existem várias tecnologias e linguagens de programação que podem ser utilizadas, sendo que as mais conhecidas e utilizadas na atualidade são o ASP.NET, PHP e *Java*, tal como se pode verificar na figura 3.1:

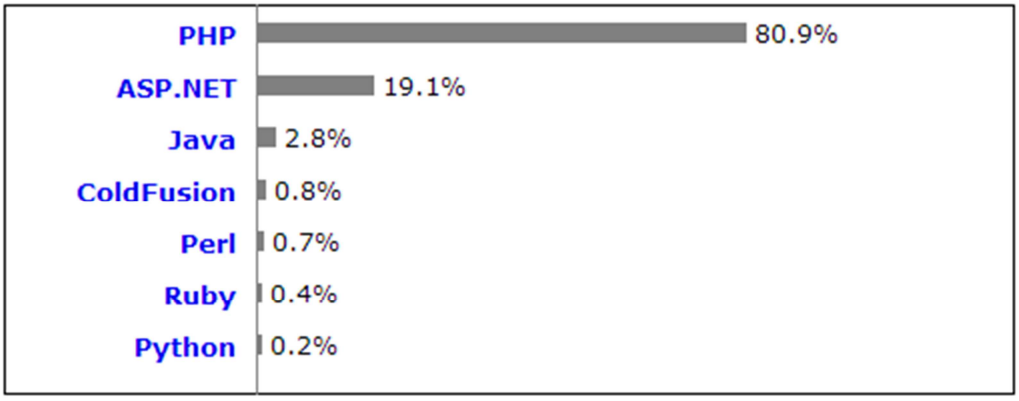


Figura 3.1 – Distribuição da utilização de linguagens *Server-Side* [3]

Apesar de haver uma grande proliferação de tecnologias e linguagens, para os *browsers* trata-se de uma tecnologia transparente, uma vez que o resultado do processamento de um *script* será sempre um código entendível pelos *browsers* (p. ex. HTML, *JavaScript*, CSS). Na lista seguinte indicam-se algumas tarefas que podem ser desempenhadas utilizando o *Server Side Scripting*:

- Inserir, alterar e eliminar dados de um sistema de base de dados;
- Alterar o conteúdo de uma página de acordo com os requisitos de um pedido, tornando-a mais apropriada à necessidade dos utilizadores;
- Processar os dados fornecidos pelo utilizador (dados estes que, usualmente, são provenientes de um formulário HTML);

[3] http://w3techs.com/technologies/overview/programming_language/all, Agosto 2013

- Aumentar os níveis de segurança, uma vez que os *browsers* não conseguem aceder ao código que está no servidor. Também permite limitar o acesso dos clientes às origens de dados utilizadas para criar as páginas;

Os *scripts* a serem executados pelo servidor podem ser embutidos diretamente no código HTML ou, como sucede em aplicações de maior dimensão, serem colocados em diversos ficheiros para uma melhor organização do código-fonte.

3.3. *CLIENT-SIDE SCRIPTING*

O *Client Side Scripting* (ou *Client Side Coding*) refere-se ao código que acompanha os ficheiros HTML e que é sempre executado do lado do cliente. Tal como sucede com o *Server Side Scripting*, o código-fonte executado pelos *browsers* pode estar embutido no próprio HTML ou organizado em diversos ficheiros externos. Estes ficheiros externos serão enviados pelo servidor à medida que forem sendo necessários. Existem, fundamentalmente, dois tipos de *Client Side Scripts*:

- *Scripts* que são executados, de forma automática e apenas uma vez, quando a página é carregada pelo *browser*;
- *Scripts* que são executados como reação a um evento ou condição. Ou seja, trata-se de código que será executado sempre que uma determinada ação ou condição se verifique.

Os *Client Side Scripts* possibilitam a implementação de muitas funcionalidades, destacando-se as seguintes:

- Alteração das páginas a apresentar, sendo um componente extremamente importante para a implementação do conceito de páginas dinâmicas;
- Validação prévia dos dados inseridos pelos utilizadores, sem haver necessidade de comunicar com o servidor (embora isto não signifique que a validação no lado do cliente possa substituir totalmente a validação do lado do servidor);
- Responder, automaticamente, a vários tipos de eventos, tais como o carregamento da página, a movimentação do rato ou a sua passagem em determinadas áreas ou a entrada ou saída de campos de texto;
- Emissão de mensagens e de avisos diversos ao utilizador.

Existem diversas linguagens de programação que podem ser utilizadas para criar este tipo de código (p. ex. *JavaScript*, *Jscript* ou *Dart*) mas aquela que, atualmente, é mais utilizada é o *JavaScript*, estando presente na esmagadora maioria dos *sites*, como se pode verificar pela análise da figura 3.2.

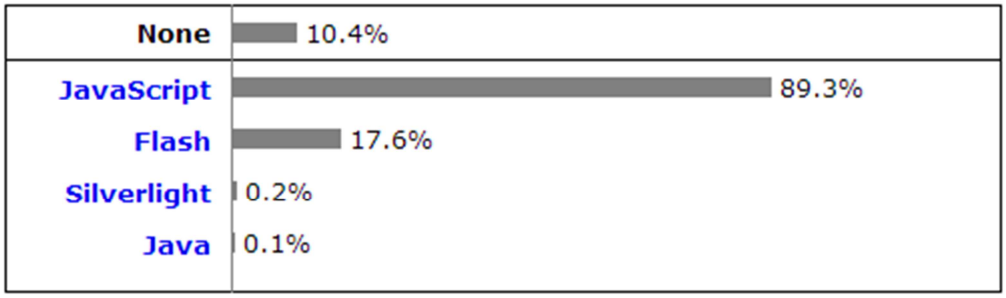


Figura 3.2 – Distribuição da utilização de linguagens *Client-Side* [4]

Por questões de segurança, o código-fonte que é executado pelos clientes tem algumas limitações e caraterísticas a que os programadores devem prestar muita atenção. Por exemplo, os *Client Side Scripts* não podem aceder a qualquer recurso ou dispositivo externo ao *browser* onde são executados. O espaço de atuação deste tipo de código resume-se ao *browser* e é necessário ter em atenção que, por vezes, diferentes *browsers* podem ter comportamentos distintos para o mesmo código-fonte. Esta última caraterística representa um dos maiores desafios e transtornos para quem desenvolve código nesta área, uma vez que é necessário ter em conta as caraterísticas de cada *browser* e é sempre conveniente testar o código no maior número possível de *browsers*. Um outro aspeto a ter em conta neste tipo de desenvolvimento é que a execução deste código pode ser desativada nos *browsers*, o que poderá impossibilitar totalmente a utilização das aplicações. Apesar destas limitações, o *Client Side Scripting* tem um papel fundamental na Internet sendo um dos componentes fundamentais no desenvolvimento de aplicações *web*.

3.4. *WEB APPLICATION*

Uma aplicação *web* (*web application*) é um *software* que é executado no contexto de um *browser* e que é suportada pelos *standards* e tecnologias relacionadas com a Internet [Shklar03]. Normalmente, a aplicação reside num servidor que a disponibiliza e envia quando for solicitada por um *browser*. Embora este termo esteja muito ligado à Internet, na realidade, uma aplicação *web* não necessita da infraestrutura da Internet para ser executada, podendo ser disponibilizada numa rede local ou apenas num só computador (neste último caso, tanto o servidor como o *browser* residem na mesma máquina). As aplicações *web* têm apresentado um grande crescimento nos últimos tempos, podendo-se apontar as seguintes vantagens:

- Do lado do cliente, as aplicações *web* apenas necessitam de um *browser* para funcionar, não sendo necessária a instalação de qualquer aplicação adicional. Existem, contudo, algumas plataformas que requerem a instalação de um *plugin* para que a aplicação possa ser executada. No entanto, trata-se de instalações

[4] http://w3techs.com/technologies/overview/client_side_language/all, Agosto 2013

simples que, na maioria das vezes, apenas requerem uma confirmação do utilizador;
- Este tipo de aplicações não necessita de grandes espaços em disco. O pouco espaço que eventualmente possam necessitar fica mais a dever-se ao *browser* do que à aplicação;
- Salvo algumas exceções (por exemplo, o *Microsoft Silverlight*) as aplicações *web* funcionam em qualquer dispositivo ou sistema operativo. Pode-se afirmar que as aplicações *web* funcionam em qualquer dispositivo que possua um *browser*;
- Os procedimentos de atualização das aplicações *web* são muito simples, uma vez que basta atualizar uma só aplicação que reside no servidor. Atualizando a aplicação no servidor, todos os *browsers* que acedam a ela estarão, automaticamente, a executar a versão atualizada, sem qualquer intervenção adicional por parte do utilizador.

Quanto aos aspetos negativos que se apontam às aplicações *web*, apresentam-se os seguintes:

- Na eventualidade de ocorrer uma quebra no acesso à Internet, as aplicações *web* deixam de funcionar. Esta questão tende a ser minimizada com a adoção do sistema *Offline Web Application* do HTML5[5]. Contudo, esta limitação aplica-se apenas na hipótese da aplicação *web* ser acedida através da Internet. Se se tratar de uma aplicação *web* que está instalada numa rede local ou num só computador, este problema das quebras de ligação não se coloca;
- Ainda num cenário em que a aplicação está instalada num servidor externo com acesso via Internet, quem utiliza a aplicação estará muito dependente de terceiros para a poder utilizar. Se a empresa onde está instalada a aplicação encerrar ou tiver problemas de excesso de tráfego, poderá afetar largamente os utilizadores;
- O desempenho e a fluidez de uma aplicação *web* está muito dependente da largura de banda de acesso à Internet assim como ao número de utilizadores que estão a utilizar a referida ligação (independentemente de estarem a usar a aplicação ou não);
- Apesar dos avanços ocorridos, uma aplicação *web* ainda não tem uma fluidez e experiência de utilização comparável às apresentadas pelas aplicações *desktop*. Na maioria das situações, utilizar uma aplicação *web* para determinado tipo de tarefas (por exemplo, para a inserção de lançamentos contabilísticos) continua a ser muito mais lento e difícil do que fazer o mesmo trabalho numa aplicação *desktop*;
- As aplicações *web* estão muito dependentes dos *browsers* e das funcionalidades que estes implementam ou deixam de implementar. Pode-se argumentar que este problema também se verifica nas aplicações *desktop*, uma vez que é usual surgirem problemas sempre que se atualiza um sistema operativo. No entanto, é mais frequente fazer-se uma atualização de um *browser* (havendo casos em que este se atualiza automaticamente sem qualquer intervenção do utilizador e sem que este se aperceba da operação) do que uma atualização de um sistema operativo.

[5] https://developer.mozilla.org/en/docs/HTML/Using_the_application_cache, Agosto 2013

O conceito *Aplicação Web* surge, amiúde, relacionado com o termo *Website* (ou *Web Site*). Embora relacionados, trata-se de situações que apresentam algumas diferenças. Sendo certo que ambos são executados num *browser*, utilizam os *standards* da Internet e podem residir num servidor que os disponibiliza quando são solicitados, os *websites* são, geralmente, percecionados como sendo mais estáticos e mais utilizados para fins informativos. Nos *websites*, a postura dos utilizadores é mais passiva e, principalmente, de obtenção de informação. Pelo contrário, uma aplicação *web* é mais dinâmica, onde é exigida uma maior intervenção por parte do utilizador. Ao nível técnico, uma aplicação *web* tenderá a ser mais complexa porque, quase sempre, tem que lidar com aspetos muito variados, como, por exemplo, a autenticação e segurança, interação com bases de dados, ou processamento de grandes volumes de informação. Isto não significa que um *website* não tenha que levar em consideração os mesmos aspetos que as aplicações *web* mas, regra geral, são caraterísticas que surgem mais associadas a estas últimas.

3.5. *WEB APPLICATION FRAMEWORK*

Um *web appplication framework* (WAF) é uma plataforma de desenvolvimento de *software* que facilita a criação de aplicações *web* [IBMnewto]. A dimensão e complexidade de um WAF pode variar muito, identificando-se *frameworks* simples vocacionadas, apenas, para uma determinada área de uma aplicação (por exemplo, *frameworks* para desenvolvimento de interfaces do utilizador como o *Microsoft Silverlight*) até *frameworks* mais extensos e complexos que abarcam todas as áreas de desenvolvimento de uma aplicação empresarial. Normalmente, um WAF define um conjunto de padrões e regras, na grande maioria o padrão *Model-View-Controller* [Shan06] ou uma variação deste, que devem ser seguidas pelos programadores e disponibilizam componentes de *software* que podem ser usados (e reutilizados) no desenvolvimento das aplicações. Estes componentes podem ser visuais (p. ex., botões, caixas de texto, separadores ou listas, e usados para a criação de interfaces com o utilizador, mas também podem ser unidades de código fonte que implementam determinado tipo de funcionalidades que podem passar pela gestão de acessos, envio de correio eletrónico, consumo de *web services*, interação com bases de dados, entre outros.

Os WAF apresentam vantagens (que se tornam mais evidentes em projetos de maior dimensão) e desvantagens que os tornam desaconselháveis em determinadas situações. Na lista seguinte apresentam-se os aspetos positivos dos *Web Application Frameworks* [Brown08;Merkel10;Porebski11]:

- Os WAF são indicados no desenvolvimento de aplicações que interajam muito com uma base de dados (por exemplo, aplicações de gestão ou que sejam utilizadas para o registo de qualquer tipo de informação). Como quase todos os *frameworks* disponibilizam componentes para interação com bases de dados, a adoção de um WAF pode revelar-se bastante vantajosa. Os WAF também se revelam muito adequados para o desenvolvimento de aplicações modulares em que os módulos

podem ser adicionados ou removidos dinamicamente sem afetar o resto das aplicações;

- Sendo bem desenvolvido, um WAF pode favorecer a implementação de boas práticas de programação que, de outra forma, poderiam nunca ser colocadas em prática pelos programadores. Por exemplo, como os WAF tendem a implementar o padrão *Model-View-Controller* (MVC), os programadores são forçados a desenvolver as aplicações com base neste princípio. Se a mesma aplicação fosse desenvolvida sem se recorrer a um WAF, a probabilidade de não se utilizar este padrão seria muito elevada. Isto não significa, contudo, que o facto de não se utilizar o padrão MVC dê origem a um código de menor qualidade;
- Se uma aplicação for desenvolvida com base num *framework*, torna mais fácil a entrada de novos elementos na equipa do projeto, uma vez que a forma como a aplicação está estruturada já é conhecida;
- A utilização de um WAF facilita muito o desenvolvimento de aplicações *web* uma vez que abstrai o programador dos aspetos mais difíceis e morosos do desenvolvimento *web* (p. ex., a interação com bases de dados, desenho de formulários, autenticação e autorização), ao mesmo tempo que disponibiliza componentes que podem ser utilizados no *software* (o que também pode reduzir drasticamente os tempos de desenvolvimento).

Apesar das vantagens e benefícios que se obtêm com a utilização de WAF, estes também apresentam aspetos negativos e existem situações em que a sua utilização é desadequada [Merkel10; Porebski11]:

- A utilização de um WAF torna-se desadequada para a criação de aplicações simples ou muito pequenas. Neste tipo de aplicações a adoção de um *framework* pode aumentar, desnecessariamente, a complexidade. Embora o objetivo de um WAF seja o de simplificar o desenvolvimento, esta vantagem apenas se revela em projetos de alguma dimensão. Em projetos que requeiram elevados níveis de desempenho, um WAF também pode ser contraproducente uma vez que a facilidade e simplicidade que os *frameworks* aportam às aplicações são conseguidas à custa do desempenho. A utilização de WAF também se revela desvantajosa no desenvolvimento de aplicações muito específicas que não se enquadram na linha de orientação dos *frameworks*. Nestes casos, como um WAF pode estar vocacionado para o desenvolvimento de um determinado tipo de aplicações, tentar desenvolver algo diferente pode revelar-se muito difícil;
- As vantagens de um *framework* são quase sempre obtidas, pelo menos, à custa de uma redução no desempenho da aplicação. A adoção de um *framework*, seja ele qual for e em que plataforma for, representa normalmente uma perda no desempenho. Por isso, cabe à equipa de projeto aferir se essa perda é compensada pelos benefícios;
- Os *frameworks* ocultam as especificidades do desenvolvimento com o objetivo de o tornar mais simples e rápido. Esta camada de abstração poderá ser prejudicial uma vez que caso não se conheçam as especificidades do *framework* corre-se o risco tomar decisões erradas no desenvolvimento ou criar soluções menos eficientes;

- O desenvolvimento inicial com um *framework* pode ser lento uma vez que, por mais simples que os *frameworks* possam ser, há sempre uma curva de aprendizagem e demora sempre algum tempo até que um programador ajuste o seu trabalho e raciocínio à forma de trabalhar proposta (ou imposta) pelo WAF.

3.6. TIPOS DE *WEB APPLICATION FRAMEWORKS*

Tal como foi referido na secção 3.5, a maioria dos *frameworks* de desenvolvimento de aplicações *web* implementam o padrão MVC. Contudo, a forma como os componentes interagem entre si pode variar, sendo identificáveis vários tipos, dos quais destacamos os seguintes [Shan06]:

- *Action-based* (ou *Request-based*);
- *Component-based*.

Um *framework action-based* recebe os pedidos provenientes dos clientes e processa-os através de um *controller*, o qual também é responsável por selecionar e invocar a ação (ou ações) necessária para satisfazer o pedido. Todo o trabalho relacionado com a interpretação e tratamento do pedido (p. ex. análise da URL, extração e tratamento de parâmetros ou seleção e invocação da ação) é feito pelo *controller*, o que permite um maior controlo sobre o que está a ser feito e uma grande independência entre os diferentes componentes. Adicionalmente, o *controller* também é responsável pelo envio da resposta ao cliente que efetuou o pedido.

Nos *frameworks component-based* toda a lógica de interpretação e processamento dos pedidos está encapsulada em componentes, tornando este processo mais simples. Neste tipo de *framework* o trabalho do programador está centrado na criação das ações e dos componentes que farão parte do *model*, ficando a cargo do *framework* a utilização dos mesmos. Ao contrário do que sucede com os *frameworks action-based*, nos *component-based* existe uma menor independência entre o *view* e o *controller*. Aqui, a lógica não está centrada no princípio *request-response* mas sim na disponibilização e utilização de componentes de código e visuais.

3.7. HTTP/HTTPS

O HTTP (*Hypertext Transfer Protocol*) é o protocolo usado para transferir recursos entre dois dispositivos, sendo o protocolo de referência na Internet. Trata-se de um protocolo que segue o princípio do *Request/Response* entre um cliente (habitualmente um navegador *web*) e um servidor [RFC2616]. Uma outra caraterística deste protocolo é que não guarda

o estado entre duas petições, ou seja, cada petição que é feita a um servidor é independente e não terá qualquer relação com outras petições que previamente tenham sido feitas. Sempre que um cliente pretender aceder a um recurso localizado num servidor fará um pedido HTTP a esse servidor. Ao receber esse pedido, o servidor fará os processamentos que forem necessários e enviará o recurso solicitado, após o que terminará a sessão. Caso o mesmo cliente solicite ao mesmo servidor outro recurso será estabelecida uma nova comunicação que nada terá a ver com o pedido anterior. O facto de se tratar de um protocolo sem estado permite-lhe elevados níveis de escalabilidade e, provavelmente, terá sido um dos principais fatores do seu sucesso. No entanto, esta caraterística também é uma das principais dificuldades a contornar para se desenvolverem aplicações *web*. Uma petição HTTP tem sempre o seguinte formato:

```
<Método> <URL> <Versão HTTP>
[<Nome1>:<Valor1>,[<Valor2>,…,<ValorN>]]
[<Nome2>:<Valor1>,[<Valor2>,…,<ValorN>]]
…
[<NomeN>:<Valor1>,[<Valor2>,…,<ValorN>]]

[<Corpo da Mensagem>]
```

Na primeira linha, entre cada conjunto de <> tem que haver um espaço e tudo o que aparece entre parêntesis retos é de preenchimento facultativo. De seguida, apresenta-se um exemplo de uma petição efetuada por um *browser*:

```
GET /?modulo=contactos HTTP/1.1
Host: www.macwin.pt
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:23.0) Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: pt-pt,pt;q=0.8,en;q=0.5,en-us;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

O primeiro elemento da primeira linha é o comando que se envia ao servidor. No exemplo anterior usa-se o comando GET, o qual serve para pedir ao servidor um determinado recurso. Para além do comando GET, o protocolo HTTP prevê a utilização dos seguintes comandos [Thomas01]:

- **HEAD**: Pede ao servidor o envio de uma resposta mas contendo apenas o respetivo cabeçalho. Equivale ao pedido GET em que se pretende, apenas, obter o cabeçalho da resposta;
- **POST**: Método usado para enviar dados ao servidor para que sejam processados por este. Os dados a enviar serão colocados no corpo do pedido;
- **PUT**: Método usado para enviar um recurso ao servidor. A diferença entre este método e o POST está na forma como o servidor interpreta o endereço indicado na mensagem. No POST, o endereço identifica um componente que irá receber os

recursos e processá-los. No PUT, o endereço representa o local onde o servidor deverá colocar os recursos

- **DELETE**: Elimina um recurso no servidor;
- **TRACE**: Método usado para pedir ao servidor uma mensagem de resposta. Pode ser útil para o diagnóstico de possíveis problemas;
- **OPTIONS**: Método usado para determinar quais os métodos suportados pelo servidor;
- **CONNECT**: Este método deverá ser usado para efetuar pedidos em situações em que seja necessário usar um *proxy* e pedidos que requeiram o protocolo SSL;
- **PATCH**: Método a utilizar para atualizar um recurso localizado num servidor.

No exemplo anterior, por se tratar de um método GET não existe a secção correspondente ao corpo da mensagem. Apresenta-se, de seguida, um exemplo de uma petição POST com dados de envio no corpo da mensagem:

```
POST / HTTP/1.1
Host: www.macwin.pt
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:23.0) Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: pt-pt,pt;q=0.8,en;q=0.5,en-us;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive

modulo=contactos
```

Esta petição é praticamente igual à anterior sendo que a única diferença está no método. Quando se trata de um GET, a informação a enviar será colocada no endereço, logo a seguir ao nome do método. Num POST a informação será colocada no corpo da mensagem.

Após receber a petição, o servidor efetuará as operações necessárias e enviará a resposta. Uma resposta de um servidor respeita o seguinte formato:

```
<Versão http> <Código de Retorno> <Mensagem>
[<Nome1>:<Valor1>,[<Valor2>,…,<ValorN>]]
[<Nome2>:<Valor1>,[<Valor2>,…,<ValorN>]]
…
[<NomeN>:<Valor1>,[<Valor2>,…,<ValorN>]]

[<Corpo da Mensagem>]
```

Também aqui os elementos da primeira linha são separados por espaços e o número de elementos `Nome:Valor` poderá variar. Apresenta-se, de seguida, um exemplo de uma resposta de um servidor *web* a um pedido efetuado por um agente.

```
HTTP/1.1 200 OK
Date: Mon, 26 Aug 2013 13:09:27 GMT
Server: Apache
X-Powered-By: PHP/5.3.27
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
  <title>Macwin - Sistemas Informaticos, S. A.</title>
</head>
<body>
…
</body>
</html>
```

No exemplo anterior o código de retorno foi o 200, o que significa que não ocorreu nenhum erro. A especificação do protocolo define diversos códigos de resposta, os quais são constituídos por números de 3 algarismos e servem para indicar, de forma normalizada, o que se passou com o pedido. Os códigos são agrupados em categorias da seguinte forma [RFC2616]:

- **1XX**: Indicam que a petição foi recebida e que está em processamento;
- **2XX**: Códigos que indicam que a petição foi aceite e processada sem erros;
- **3XX**: Códigos indicativos de redireccionamento e que deverão ser tomadas as ações para a conclusão do pedido;
- **4XX**: Indicam que a petição não foi satisfeita porque ocorreu algum erro imputável ao cliente (p. ex. endereço errado ou mensagem em formato inválido);
- **5XX**: Indicam que a petição não foi satisfeita por um motivo imputável ao servidor.

Existe uma variante do protocolo HTTP e que se chama HTTPS. Trata-se de um protocolo em tudo semelhante ao HTTP mas que utiliza o protocolo de segurança SSL para encriptar e autenticar o tráfego entre um cliente e o servidor.

3.8. *RICH INTERNET APPLICATION*

Uma *Rich Internet Application* (RIA) é uma aplicação *web* desenvolvida com o objetivo de disponibilizar as mesmas funcionalidades e experiência de utilização que uma aplicação *desktop*. Ao desenvolver uma aplicação RIA tenta-se combinar as vantagens de uma aplicação *desktop* (mais concretamente os aspetos relacionados com a interface do utilizador) com as vantagens de uma aplicação *web*. Embora tentando simular a experiência de utilização de uma aplicação *desktop*, uma RIA continua a ser executada num *browser* e contém, igualmente, código que é executado no lado do servidor e no lado do

cliente. Apesar de ser apenas necessário um navegador para se executar uma RIA, existem aplicações que requerem a instalação de um *plugin* no navegador para que possam funcionar. São exemplos deste tipo de plataformas o *Adobe Flex*, *JavaFX*/*Java Applets* ou o *Microsoft Silverlight*. Para que seja possível executar uma RIA desenvolvida com qualquer uma destas plataformas é obrigatória a presença de um *plugin* no *browser*. Durante muito tempo esta era a principal técnica para se desenvolverem RIA mas, com o surgimento do HTML5 e com uma utilização intensiva da linguagem *JavaScript*, passou a ser possível desenvolverem-se aplicações RIA com a mesma fluidez e experiência de utilização que as aplicações que requerem um *plugin*. Aliás, a opção pelo HTML5 e por sofisticadas bibliotecas de *JavaScript* tem ganho preponderância em relação à opção por plataformas com *plugins*, sendo cada vez mais as aplicações desenvolvidas segundo esta técnica [Smeets08].

Ao nível das vantagens, uma RIA apresenta os mesmos aspetos positivos que são apontados às aplicações *web* tradicionais e, para além dessas, podem-se referir, adicionalmente, as seguintes [Smeets08]:

- **Desempenho melhorado:** Uma vez que nem todas as ações dos utilizadores implicam uma interação com o servidor, o desempenho de uma RIA tende a ser melhor. Contudo, este desempenho também depende muito do *browser* que se estiver a utilizar e das características do equipamento onde é executada;
- **Mais escalável:** Como grande parte do processamento de uma RIA pode ser efetuado do lado do cliente, a escalabilidade da aplicação será maior uma vez que o servidor não terá tanta carga de trabalho e este será sempre repartido pelos novos utilizadores;

Quanto às desvantagens específicas das RIA, destacam-se as seguintes [Smeets08]:

- **Dependência do *JavaScript* ou de *plugins*:** Como as RIA fazem uma utilização muito intensiva do *JavaScript*, se este estiver desativado no *browser*, a aplicação poderá ter funcionalidades indisponíveis ou, no limite, não funcionar na totalidade. A exigência de um *plugin* também pode ser considerado como um aspeto negativo uma vez que nem todos os utilizadores têm permissões para instalarem *software* adicional nos seus equipamentos;
- **Dificuldades nas pesquisas:** Em virtude da sua arquitetura, as RIA não são muito bem indexadas pelos motores de busca. Se considerarmos que, geralmente, este tipo de aplicações correm em ambientes fechados e restritos, esta limitação dos motores de busca não se revela muito negativa;
- **Volume de dados a transferir:** Numa primeira utilização ou sempre que ocorrerem alterações no código-fonte de uma aplicação RIA, será necessário transferir para o equipamento onde está a correr o *browser* toda ou parte da aplicação, o que pode originar um acréscimo de tempo no arranque da aplicação.

3.9. *WEB SERVER*

O termo *Web Server* pode ser utilizado para referir dois conceitos diferentes, embora relacionados. Um *Web Server* pode ser um computador que está *online* e que alberga os *web sites* que poderão ser acedidos pelos utilizadores. Contudo, o termo também é utilizado para se referir o *software* que, usando um modelo cliente/servidor, responde às petições feitas pelos clientes, disponibilizando os recursos armazenados nos computadores [Shklar03;Mateu10]. Temos, portanto, que o termo *Web Server* aplica-se tanto ao *hardware* como ao *software* e que desempenham um papel fundamental na Internet.

Centrando a nossa análise na definição referente ao *software*, podemos afirmar que cada computador que esteja *online* e que contenha páginas *web* com o objetivo de as disponibilizar a quem o solicitar, terá que ter instalado, obrigatoriamente, um *software web server*. Existem diversas aplicações que podem ser usadas como servidores *web*, sendo o *Apache Web Server* e o *Microsoft Internet Information Services* (IIS) duas das mais conhecidas e, simultaneamente, das mais utilizadas[6]. Tal como já se referiu, um servidor *web* é uma aplicação que é usada para disponibilizar, mediante uma prévia petição, os recursos armazenados num computador. Este processo de disponibilização de recursos inicia-se, normalmente, com um utilizador a digitar um endereço num *browser* ou a clicar numa ligação para uma determinada página. Após a ação do utilizador, o navegador enviará um pedido HTTP para o endereço especificado. No endereço em causa estará um *web server* que, mal receba o pedido, iniciará os procedimentos necessários para o envio da correspondente resposta.

Para além da disponibilização dos recursos localizados num sistema, também é frequente encontrarmos *web servers* a funcionarem como interface de acesso a diversos tipos de dispositivos, tais como impressoras, *routers* ou câmaras.

3.10. PÁGINAS *WEB* ESTÁTICAS/DINÂMICAS (*DYNAMIC/STATIC WEB PAGES*)

Uma página *web* pode ser estática ou dinâmica. Uma página considera-se estática quando o seu conteúdo não varia a cada petição que é feita. O conteúdo de uma página estática nunca varia a não ser que seja alterada, diretamente, no local onde está alojada. Por este motivo será sempre disponibilizada a mesma página, independentemente do contexto em que seja visualizada e não será possível qualquer ação por parte do utilizador no sentido de alterar o conteúdo apresentado [IBMnewto]. Normalmente, este tipo de páginas apenas é usado para a apresentação de informação, não sendo possível qualquer tipo de interação. Esta é a forma mais simples de se criarem páginas *web* embora também se revele uma solução com muitas limitações. Apesar das suas limitações, as páginas estáticas apresentam as seguintes vantagens:

[6] http://w3techs.com/technologies/overview/web_server/all, Agosto, 2013

- As páginas estáticas, por não exigirem grandes cargas de processamento aos servidores, são a forma mais rápida de disponibilizar conteúdos;
- Podem funcionar sem recurso a complexas técnicas de programação e sem recurso a bases de dados;
- Permitem uma otimização dos sistemas de *caching* uma vez que o conteúdo disponibilizado é sempre o mesmo.

Quanto às desvantagens, destacam-se as seguintes:

- A utilização de páginas estáticas torna-se muito complicada e pouco eficiente no desenvolvimento de *sites* grandes e aplicações *web* (podendo mesmo inviabilizar o desenvolvimento deste tipo de aplicações);
- Não permitem adequar os conteúdos a disponibilizar ao contexto (p. ex. ao utilizador ou ao país);
- Uma pequena alteração numa página implica a alteração do respetivo ficheiro no servidor onde se encontra alojado;
- Não permitem o desenvolvimento de aplicações centradas nos dados (ou baseadas numa base de dados).

A alternativa às páginas estáticas e às suas limitações reside nas páginas dinâmicas, as quais permitem ajustar o conteúdo ao contexto no qual são apresentadas. Neste tipo de páginas o seu conteúdo é gerado em tempo real e quase sempre incluem código que é executado no servidor antes de serem disponibilizadas [IBMnewto]. Em virtude desta caraterística, torna-se muito mais simples e fácil alterar o conteúdo das páginas. Ao contrário das páginas estáticas, as dinâmicas já permitem receber dados do utilizador, os quais têm a possibilidade de alterar os conteúdos ou fornecerem dados. As páginas dinâmicas apresentam o seguinte conjunto de vantagens:

- Permitem adequar o conteúdo a apresentar ao utilizador, local ou navegador, entre outros, e facilitam o desenvolvimento de aplicações *web*;
- São a solução mais adequada para a criação de páginas que apresentam informação que varia muito uma vez que é possível atualizar o conteúdo sem ser necessário alterar o código HTML das páginas;
- Possibilitam a implementação de medidas de autenticação, permissão, registo de acessos, acesso aos recursos do servidor físico onde estão instaladas e interagir com um sistema de bases de dados para registo/obtenção de informação;
- Possibilitam o desenvolvimento de aplicações empresariais, tipicamente centradas nos dados.

Embora sejam apresentadas poucas vantagens, estas são fundamentais para a construção da Internet tal como a conhecemos hoje. Na lista seguinte apresentam-se os aspetos negativos apontados às páginas dinâmicas:

- Em comparação com páginas estáticas, as páginas dinâmicas requerem uma maior capacidade de processamento e uma maior quantidade de recursos. Em princípio, o número de páginas estáticas disponibilizadas por segundo será sempre superior ao número de páginas dinâmicas disponibilizadas no mesmo período de tempo;
- Como as páginas dinâmicas contêm código que é executado no servidor e que pode aceder aos recursos do sistema e às bases de dados, uma falha na segurança terá efeitos maiores neste tipo de páginas;
- O desenvolvimento deste tipo de páginas poderá ser mais complexo uma vez que exige mais conhecimentos por parte de quem está a desenvolver a página.

3.11. *MODEL-VIEW-CONTROLLER PATTERN*

O *Model-View-Controller* (MVC) é um padrão de arquitetura de *software* desenvolvido por Trygve Reenskaug para a plataforma *Smalltalk*. O padrão identifica três componentes distintos (o *Model*, o *View* e o *Controller*) cada um deles desempenhando um papel específico [Fowler02]:

- ***Model*:** Composto por um ou vários objetos não visuais que armazenam os dados da aplicação, contendo, também, as regras de negócio;
- ***View*:** Componente responsável pela apresentação dos dados ao utilizador, não tendo qualquer outra função para além desta;
- ***Controller*:** Responsável pela gestão das ações dos utilizadores, pela atualização dos dados do *Model* e do *View* (quando ocorrem alterações no *Model*).

Na figura 3.3 apresenta-se um esquema simplificado do padrão MVC.

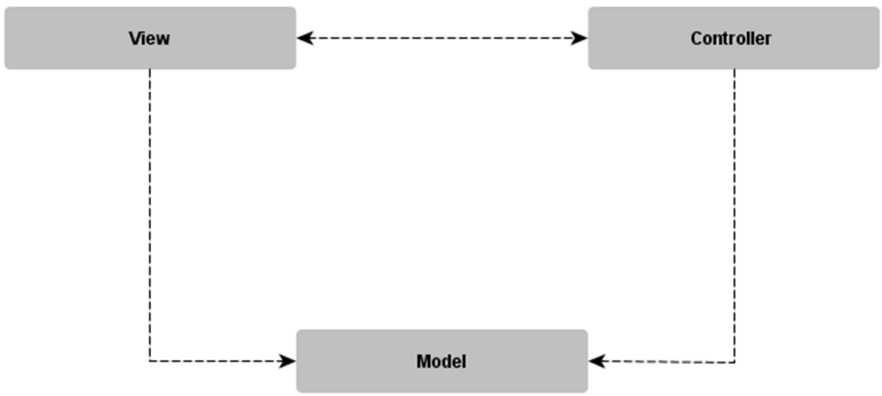


Figura 3.3 – Padrão *Model, View, Controller* [7]

[7] Adaptado de [Fowler02]

As ações executadas pelos utilizadores sobre o *View* são encaminhadas para o *Controller* para processamento. Este processamento pode dar origem a alterações no *Model* ou no próprio *View* (por exemplo, a apresentação ou ocultação de um componente visual). Neste padrão, o *View* e o *Model* são totalmente independentes, o que possibilita que um *Model* possa ser utilizado por diferentes *Views*. Embora sejam totalmente independentes, o *View* tem que ter conhecimento do *Model* mas este último pode, e deve, ser desenvolvido sem haver qualquer referência aos *Views* que, eventualmente, o possam vir a utilizar. Esta separação de conceitos é a principal característica e vantagem do padrão MVC.

No âmbito da plataforma *Java* é frequente utilizar-se o termo *Model 2* associado ao conceito MVC. Este termo foi definido numa das primeiras especificações do *JavaServer Pages*[8] (JSP) e representa uma forma alternativa (e uma evolução) de como devem ser estruturadas as aplicações *JavaEE*. Na primeira forma (*Model 1*) os *browsers* enviam os pedidos diretamente a uma página JSP a qual acederá ao componente *Model* da aplicação e, como resposta, reenviará um novo *View* (nova página JSP) para o *browser*. Como se pode verificar, nesta arquitetura os componentes *Controller* e *View* são implementados pelas páginas JSP. No *Model 2*, os pedidos dos *browsers* são recebidos e processados por *Servlets* que desempenham o papel do *Controller*. O *Servlet*/*Controller* irá interagir com o *Model* e quando esta fase for concluída, selecionará e enviará o *View* adequado como resposta. Na prática, o *Model 2* é igual ao padrão MVC, implementando todos os componentes desta arquitetura.

3.12. *FRONT CONTROLLER PATTERN*

O *Front Controller* é um padrão de desenvolvimento cuja principal caraterística é a existência de um ponto único para a interação entre a aplicação e o exterior. No âmbito de uma aplicação *web* isto significa que todos os pedidos efetuados pelos clientes são encaminhados sempre para o mesmo objeto (chamado de *Front Controller*) [Fowler02]. Com o encaminhamento de todos os pedidos para o mesmo local, torna-se mais fácil e simples a implementação de diversas funcionalidades, merecendo destaque a validação e controlo de acessos ou o registo de operações, evitando que este tipo de código esteja espalhado ao longo da aplicação.

Quando um *Front Controller* recebe um pedido HTTP, a primeira tarefa que realiza é a seleção do comando que deve ser executado. Estes comandos podem ser outros *controllers* ou métodos/objetos. O processo de seleção do comando é realizado em função do pedido e pode variar muito de implementação para implementação. A segunda tarefa a realizar pelo *Front Controller* é a invocação do comando selecionado.

Neste padrão, o trabalho do *Front Controller* resume-se, apenas, a ser o ponto único de entrada de pedidos e o seu reencaminhamento dentro das aplicações. A partir do momento em que um pedido é reencaminhado a participação do *Front Controller* termina. Revela-se uma arquitetura bastante adequada para situações em que se pretenda aplicar a

[8] http://www.kirkdorffer.com/jspspecs/jsp092.html, Agosto 2013

mesma lógica para um conjunto diversificado de pedidos, apresentando, também, a grande vantagem de evitar a duplicação de código ao longo de vários *controllers*.

3.13. *PAGE CONTROLLER PATTERN*

No padrão *Page Controller* implementa-se um *controller* específico para cada página que constitui uma aplicação *web*. Os *controllers* específicos podem fazer parte da página ou, alternativamente, estarem colocados em ficheiros/pastas separados [Fowler02]. Com esta abordagem, consegue-se bastante simplicidade uma vez que cada *controller* apenas tem que lidar com as questões de uma só página. Neste padrão as funções desempenhadas pelos *page controllers* são as seguintes:

- Retirar das URL, caso exista, toda a informação necessária para completar o pedido;
- Reencaminhar os dados para o modelo, tendo em vista o seu processamento, evitando, desta forma, que exista alguma relação entre o modelo e o pedido HTML;
- Redirecionar a informação produzida pelos modelos para a página/vista associada.

Este padrão de desenvolvimento adequa-se em situações em que a lógica da aplicação é simples, por oposição ao padrão *Front Controller* que se adequa melhor em aplicações mais complexas. Também é frequente encontrarem-se aplicações que implementam os dois padrões de desenvolvimento, especialmente em processos de conversão.

3.14. *ACTIVE RECORD PATTERN*

O *Active Record* é um padrão de desenvolvimento de *software* que encapsula num objeto um registo de uma tabela de uma base de dados e que, simultaneamente, define as regras de negócio associadas à tabela [Fowler02]. O objetivo deste padrão é que cada um dos objetos representem, da forma mais fiável possível, a estrutura das tabelas associadas sendo, também, responsáveis pela leitura e gravação dos dados. Desta forma, um objeto deverá conter, para além dos dados, os seguintes métodos:

- Método para inserção, alteração e eliminação de registos;
- Métodos para leitura e preenchimento dos campos dos objetos;
- Métodos de validação dos dados e de imposição das regras de negócio;
- Métodos para construção de objetos com base numa chave primária ou com base num valor devolvido por uma consulta;
- Métodos para localização de um registo numa base de dados e que devolvem uma instância do objeto.

O padrão *Active Record* revela-se bastante adequado em cenários em que as regras de negócio não sejam muito complicadas. Para situações em que as regras de negócio sejam complexas e impliquem muitas interações com outros objetos, o padrão *Active Record* apresenta alguns constrangimentos, sendo preferível optar por outros padrões.

3.15. *OBSERVER PATTERN*

O padrão *Observer* define uma relação entre um objeto (chamado de *Subject*) e um ou vários objetos (chamados de *Observers*), na qual uma alteração no *Subject* será comunicada ao/aos objetos *Observers* que estejam associados. Este tipo de relacionamento entre os objetos é usual chamar-se de Publicação/Subscrição (*Publish/Subscribe*) e caracteriza-se, também, pelo facto dos objetos envolvidos no relacionamento serem totalmente independentes. Outra caraterística muito importante deste padrão é que para cada *Subject* podem existir tantos *Observers* quanto os que forem necessários [Gamma95]. O padrão *Observer* revela-se especialmente apropriado em cenários em que uma alteração no estado de um objeto implica alterações nos estados de outros, sendo também aplicável mesmo em situações em que os objetos, apesar de estarem relacionados, são totalmente independentes uns dos outros. O *Subject* sabe que tem que notificar outros objetos sobre eventuais alterações no seu próprio estado mas nada mais sabe de concreto sobre os objetos a notificar.

Uma vez que este padrão se baseia num conjunto de objetos desacoplados tem a vantagem de poder ser facilmente implementável em sistemas cuja arquitetura é baseada em camadas. Ou seja, o *Subject* pode estar situado numa determinada camada e notificar objetos que estejam localizados em camadas diferentes. Contudo, a implementação deste padrão pode, quando mal efetuada, dar origem a notificações/atualizações em cadeia, o que pode causar uma sobrecarga de trabalho e, no limite, bloquear totalmente uma aplicação. Por exemplo, um *Subject* A notifica um *Observer* B que, simultaneamente, é um *Subject* de outro objeto C. O objeto B, ao receber a notificação do A e ao atualizar o seu estado irá, segundo o mesmo princípio, notificar o objeto C. Se, por sua vez, este objeto C for *Subject* do A, teremos uma chamada referência circular que poderá, caso não se tenham tomado a devidas precauções, entrar num ciclo infinito de atualizações.

3.16. URI/URL/URN

Um *Uniform Resource Identifier* (URI) é um conjunto de carateres que identifica um recurso (físico ou abstrato) de forma simples, única e uniforme [RFC3986]. Neste contexto, considera-se como sendo um recurso todo e qualquer conjunto de informação (p. ex. um documento de texto, uma imagem, uma fonte de informação ou um serviço) que esteja à

disposição de quem dele necessitar. Adicionalmente, um recurso não tem que estar, obrigatoriamente, disponível na Internet, podendo estar acessível em locais físicos. A uniformidade dos URI garante que recursos totalmente diferentes possam ser referenciados seguindo o mesmo padrão. A constituição de um URI pode conter vários elementos da lista que a seguir se apresenta:

- *Scheme*
- *Authority*
- *Path*
- *Query*
- *Fragment*

Nem todos os elementos são obrigatórios para a construção de um URI embora a presença do *Scheme* desempenhe um papel muito importante uma vez que estipula a forma e a sequência como será constituído o restante URI. Assim, um URI poderá ter a seguinte forma (incluindo todos os elementos):

```
xxxx://www.macwin.pt:8080/products?codigo=abcd#detalhes
```

No URI anterior, identificam-se os seguintes elementos:

Elemento	Valor
Scheme	xxxx
Authority	www.macwin.pt:8080
Path	/products
Query	codigo=abcd
Fragment	detalhes

Tabela 3.1 – Elementos constituintes de um URI

Os exemplos seguintes também se tratam de URI mas, nestes casos, não está contemplada a presença de todos os elementos:

```
ftp://ftp.macwin.pt/docs/manuais/imobilizado.pdf
mailto:abcd@macwin.pt
news:comp.macwin.info.pt
urn:isbn: 978-0321563842
```

Um URI pode, ainda, ser classificado com um localizador, um nome ou ambos. O localizador, chamado de *Uniform Resource Locator* (URL) corresponde à parte do URI que possibilita a localização do recurso em causa. O nome, chamado de *Uniform Resource Name* (URN) possibilita a identificação do recurso. Ou seja, o URN permite que se identifique um recurso e o URL permite que o mesmo possa ser localizado, sendo que um URI poderá conter apenas um dos dois ou ambos. Um exemplo típico da combinação de URL com URI

pode ser dado utilizando um livro. Este objeto pode ser identificado usando a notação ISBN

```
urn:isbn: 978-0321563842
```

que o identificará, univocamente, como sendo o livro *The C++ Programming Language, 4th Edition*, o qual poderia estar disponível para download na localização:

```
http://books.com/programming/the_cpp_programming_lang_4th_ed.pdf
```

No primeiro exemplo estamos perante um URN e no segundo perante um URL sendo que em ambos os casos se tratam de URI.

3.17. HTML

O *HyperText Markup Language* (HTML) é uma linguagem usada na Internet que permite publicar documentos, os quais podem conter, por exemplo, texto, imagens, vídeos, sons, listas, formulários e ligações a outras páginas. Um documento HTML é constituído por um conjunto de etiquetas (utilizando-se com mais frequência o termo *tags*) as quais, na sua maioria, utilizam-se em pares (uma para abrir a secção e outra para a encerrar). Apesar desta regra, existem algumas *tags* que não necessitam de ser utilizadas nestas condições [HTML401]. No código seguinte apresenta-se um pequeno exemplo de um documento HTML:

```
<!DOCTYPE html>
<html>
	<head>
		<title>Página experimental</title>
	</head>
	<body>
		<h2>Corpo da página</h2>
		<br/>
		<p>Outra linha</p>
	</body>
</html>
```

A linguagem HTML foi desenvolvida por Tim Berners-Lee no início da década de 90 enquanto colaborador do CERN. Esta primeira versão da linguagem foi sendo aperfeiçoada e foram surgindo diversos rascunhos ao longo do tempo até que em Novembro de 1995 surgiu a primeira versão oficial, chamada de HTML 2.0 [RFC1866]. A partir desta data foram sendo lançados diversos documentos, por várias entidades, que adicionaram novas funcionalidades à linguagem. Em Janeiro de 1997 surge uma nova versão oficial da linguagem, o HTML 3.2, desta vez sob o controlo e coordenação do W3C. A principal vantagem desta nova versão foi permitir normalizar e definir um padrão que foi aceite

pelas entidades que tinham vindo a participar no processo de evolução da linguagem. Em Dezembro de 1997 é lançada a versão 4.0 do HTML que expande bastante a versão anterior, introduzindo mecanismos melhorados de formatação e estilo, tabelas mais funcionais e poderosas e a possibilidade de se adicionarem *scripts*. Em Dezembro de 1999 foi lançada uma pequena atualização, com a versão 4.01, sendo esta, atualmente, a versão mais recente da linguagem.

Apesar da versão 4.01 ser a mais recente, o trabalho para a especificação de uma nova versão foi iniciado há bastante tempo. Com efeito, em 2004 o WHATWG iniciou os trabalhos de especificação de uma nova versão para o HTML, chamado de HTML5. Em Janeiro de 2008 foi publicado o primeiro rascunho da especificação, que tem vindo a ser evoluída e alvo de consolidação desde essa data até aos dias de hoje. Esta nova versão do HTML, para além de adicionar muitas novas funcionalidades à versão atual vem, também, reforçar a separação muito clara entre a definição do conteúdo e a forma como o mesmo é apresentado. Neste sentido, o HTML5 vem tornar obsoletas e desaconselhar/proibir a utilização de *tags* relacionadas especificamente com os aspetos visuais e de formatação. Prevê-se que a versão final da especificação seja apresentada durante o ano de 2014[9].

3.18. XML

O *Extensible Markup Language* (XML) é uma linguagem simples e flexível que permite a criação de documentos facilmente interpretados e processados pelas aplicações, independentemente da sua linguagem ou plataforma [XML10;Mateu10]. Ao contrário do HTML, cujo principal objetivo é a apresentação de dados, o XML tem como principal finalidade o transporte e a sua disponibilização. Apesar de ambas as linguagens apresentarem bastantes semelhanças (por exemplo, a existência de *tags*) as suas finalidades não são as mesmas. Outro aspeto que também diferencia as duas linguagens é que no XML não existe num número limitado e definido de *tags* podendo cada documento definir as tags que forem necessárias para descrever os dados nele contidos.

Atualmente existem duas versões do XML. A primeira versão surgiu em 1998 com a publicação de uma recomendação da W3C. Desde então têm sido publicadas pequenas revisões (que não alteraram a versão) sendo a última edição a 5.ª, publicada em 2008. A versão 1.1 do XML foi publicada em 2004 tendo surgido, posteriormente, uma pequena revisão em 2006 [XML11]. O lançamento desta nova versão do XML não significa que a versão 1.0 deixou de poder ser utilizada. Trata-se, apenas, de uma forma alternativa de gerar um documento XML respeitando regras ligeiramente diferentes.

Em qualquer uma das versões, os objetivos que estiveram na origem da criação da linguagem foram os seguintes:

- O XML deveria ser compatível com os padrões da SGML, embora muito mais simples e rápido de usar;

[9] http://dev.w3.org/html5/decision-policy/html5-2014-plan.html, Agosto 2013

- Deveria ser usado e suportado por todo o tipo de aplicações assim como ser usado livremente na Internet;
- Deveria ser fácil de criar e interpretar e as ferramentas de manuseamento do XML também deveriam ser simples;
- Um documento XML deverá ser facilmente lido e entendido por um ser humano;
- A criação de um documento XML deverá ser uma operação simples, fácil e rápida;
- A estrutura de um documento XML deverá ser formal e concisa.

Como se poderá verificar através do exemplo seguinte, pelo menos os objetivos da simplicidade, facilidade de leitura e rapidez de criação foram alcançados:

```
<?xml version="1.0" encoding="UTF-8"?>
<cursos>
  <curso>
    <descricao>MTGSI</nome>
    <anoletivo>2012/2013</anoletivo>
    <alunos>
      <aluno>
        <nome>Nome do aluno</nome>
      </aluno>
      <aluno>
        <nome>Nome de outro aluno</nome>
      </aluno>
    </alunos>
  </curso>
</cursos>
```

Devido à sua grande flexibilidade e simplicidade, bem como à sua elevada extensibilidade, o XML é usado em inúmeros cenários alguns dos quais estão apresentados na lista seguinte:

- Configurações. Ficheiros de configuração usados nos mais variados tipo de *software*;
- *Web Services*. Usado tanto para a definição e publicação dos serviços como no seu consumo;
- Bases de dados. O XML é usado tanto no armazenamento como na geração de informação;
- Integração de sistemas. A comunicação entre sistemas (muitas vezes tratando-se de sistemas totalmente diferentes) é uma área onde o XML desempenha um papel fundamental;
- Conteúdos *web*. Como as páginas *web* podem ser criadas usando a linguagem XHTML (a qual está relacionada com o XML), pode-se afirmar que este está, também, muito disseminado havendo, também, muito conteúdo que é disponibilizado através de ficheiros XML.

3.19. XHTML

O *Extensible Hypertext Markup Language* (XHTML) é uma linguagem que expande o HTML, modelando-o às regras do XML por forma a tornar a sua utilização mais padronizada [XHTML10]. As vantagens da utilização do XHTML em detrimento do HTML estão resumidas na lista seguinte:

- Como um documento no formato XHTML é criado conforme os princípios do XML, poderá ser lido e processado por ferramentas de tratamento de ficheiros XML;
- O facto de o XHTML ser baseado XML também facilita a interação com outros padrões e ambientes de utilização;
- Usando o XHTML torna-se muito mais fácil a introdução de novos elementos ou atributos, dando uma rápida resposta às constantes alterações que vão surgindo no dia-a-dia.

O primeiro rascunho da especificação do XHTML foi publicado em Dezembro de 1998 mas a versão 1.0 da especificação apenas surgiu em Janeiro de 2000. A versão 1.1 do XHTML foi lançada em Maio de 2001 [XHTML11] e, desde essa data, esta tem sido a versão corrente da linguagem. Entretanto, foram surgindo rascunhos para novas versões da linguagem (versões 1.2 e 2.0) mas estas nunca chegaram a ser efetivamente confirmadas como oficiais. Não obstante, alguns dos melhoramentos e acrescentos dos rascunhos acabaram por ser incluídos na versão 1.1 através da emissão pelo W3C de documentos de correção.

Comparando a linguagem XHTML com o HTML encontram-se diversas diferenças, destacando-se as seguintes:

- A presença do elemento `DOCTYPE` num documento XHTML é obrigatória, bem como a presença dos elementos `html`, `head`, `title` e `body`;
- O elemento `html` tem que conter um atributo para a definição do *namespace*;
- Ao contrário do que se passa no HTML, todos os elementos usados no XHTML têm que ser devidamente encerrados;
- O XHTML diferencia os carateres maiúsculos dos minúsculos, tanto nos elementos como nos atributos;
- A linguagem XHTML é baseada no XML enquanto que o HTML é baseado no SGML;
- No XHTML todos os valores dos atributos têm que estar delimitados por aspas, sendo que esta obrigação não existe no HTML (embora a não utilização de aspas seja desaconselhada).

De uma forma geral, um documento escrito usando a linguagem XHTML obedece a um conjunto de regras mais rígidas, as quais, caso não sejam observadas, darão origem à interrupção do processamento do documento.

3.20. DHTML

O *Dynamic* HTML (DHTML), ao contrário do HTML e do XHTML, não é uma linguagem para apresentação de conteúdos mas sim um conjunto de técnicas (HTML, *JavaScript*, CSS e uma API DOM) que, quando usadas em conjunto, possibilitam a criação de páginas *web* dinâmicas [MDNDHTML;Morris11]. A grande vantagem do DHTML é que permite o dinamismo das páginas sem recurso a *plugins* adicionais, tal como sucede com outras tecnologias. O dinamismo das páginas é alcançado através da alteração dos aspetos visuais dos elementos apresentados, inserindo novos elementos ou removendo-os, tudo isto com o recurso à linguagem de programação *JavaScript*. Neste sentido, o dinamismo pode ser definido como sendo a capacidade que os *browsers* têm de alterar o aspeto das páginas após estas terem sido carregadas e apresentadas ao utilizador.

O conceito de DHTML poderá partilhar algumas semelhanças com o conceito de páginas *web* dinâmicas ou com o AJAX, especialmente na capacidade de gerar páginas diferentes em função de diversas variáveis. Contudo, as semelhanças terminam neste ponto atendendo a que o dinamismo das páginas *web* dinâmicas ocorre sempre do lado do servidor e antes de serem apresentadas. Quanto ao AJAX, a maior diferença está no facto deste implicar sempre uma interação entre o cliente e o servidor o que não tem que ocorrer, necessariamente, com o DHTML, onde todo o processamento ocorre no lado do cliente.

3.21. CSS

O *Cascading Style Sheets* (CSS) é uma linguagem usada para definir os aspetos relacionados com a apresentação de documentos estruturados (geralmente documentos HTML ou XHTML) [RFC2318]. A informação dos CSS pode ser colocada em ficheiros diferentes daqueles que formatam ou como parte integrante do ficheiro a formatar. As caraterísticas que são passíveis de configuração variam muito, podendo tratar-se de dimensões, cores do corpo ou de fundo, limites, sombreados, tipo e tamanho das fontes, posicionamento dos elementos, entre outras.

A primeira versão do CSS (chamada de CSS *Level 1*) foi lançada em 1996 sob o controlo do W3C, tendo o CSS2 (CSS *Level 2*) surgido em Maio de 1998. Esta nova versão era baseada na versão anterior à qual foram adicionadas novas funcionalidades. Em 2002 iniciaram-se os trabalhos para uma pequena revisão do CSS2 chamada de CSS *Level 2.1* (ou CSS2.1). Esta versão corrigiu diversos erros da versão anterior e também removeu da especificação funcionalidades que ainda não estavam implementadas pelos *browsers* ou em que a implementação não era a mais adequada. A conclusão desta nova especificação demorou cerca de 9 anos, tendo atingido o estatuto definitivo apenas em 2011. Em 1999 foi lançado um rascunho inicial para o *Level 3* do CSS. Esta nova versão introduziu uma alteração substancial chamada de *Modules*. Ao contrário das versões anteriores em que todas as definições estavam agrupadas numa só especificação, no CSS3 a especificação foi dividida em módulos independentes que evoluem de forma isolada e a velocidades distintas,

havendo, por este motivo, módulos em níveis de evolução diferentes. Outro aspeto relevante em relação aos módulos do CSS3 é que poderão ser adicionados novos módulos à especificação sempre que tal seja necessário. Na lista seguinte apresentam-se alguns dos módulos atualmente associados à especificação do CSS3 [CSS3MOD]:

- CSS *Color Level 3*
- *Selectors Level 3*
- CSS *Namespaces*
- CSS *Style Attributes*
- CSS *Backgrounds and Borders Level 3*
- CSS *Conditional Rules Level 3*
- CSS *Flexible Box Layout*
- *Media Queries*
- CSS *Transitions*

Apesar de nem todos os módulos do CSS3 estarem concluídos e com o estatuto definitivo, já existem módulos para o *Level 4*, tais como o *Selectors Level 4* e o CSS *Backgrounds and Borders Level 4*. Com a criação dos módulos a evolução do CSS passou a processar-se de forma algo diferente, sendo possível existirem módulos do *Level 4* já com o estatuto definitivo havendo módulos do *Level 3* ainda numa fase de trabalho.

A sintaxe do CSS divide-se em dois itens: o *selector* e a *rule*, sendo que em cada secção podem existir um ou vários itens. O *selector* é usado para identificar os elementos aos quais serão aplicadas as regras definidas na secção *rule*. No exemplo seguinte apresenta-se um esquema genérico de um código CSS:

```
Selector 1 [, Selector 2, …, Selector N]
{
      Regra 1: Valor da Regra 1;
      [
      …
      Regra N: Valor da regra N;
      ]
}
```

Para se definir, por exemplo, que todos os elementos `h1` de um documento HTML passarão a ter a cor vermelha, deveria escrever-se o seguinte código:

```
h1 {color: red;}
```

A utilização de CSS apresenta muitas vantagens, destacando-se a efetiva separação entre o conteúdo e as regras de apresentação o que permite uma melhor organização do código assim como um maior aproveitamento e coerência das regras de apresentação. Por outro lado, como é possível aproveitar os aspetos da apresentação e definir regras para toda a

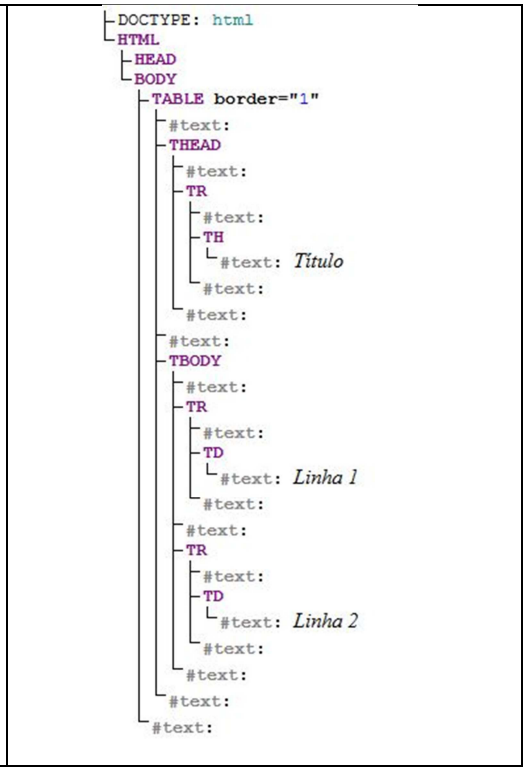
aplicação, isso permite uma maior consistência na apresentação. A possibilidade de se poder definir os aspetos de uma família ou grupo de elementos também trará vantagens ao nível do trafego uma vez que deixa de ser necessário especificar em cada elemento qual será o seu aspeto.

3.22. DOM

O *Document Object Model* (DOM) é uma API multiplataforma e multilinguagem que possibilita o acesso, a manipulação e a definição de documentos HTML e XML. Através desta API todo e qualquer elemento de um documento poderá ser acedido, modificado ou eliminado sendo dada também a possibilidade de se inserirem novos elementos [DOM]. A primeira versão do DOM (DOM *Level 1*) foi publicada em 1998 através de uma recomendação da W3C. A versão 2 (DOM *Level 2*) foi introduzida em 2000. A última versão do DOM (Dom *Level 3*) foi lançada em Abril de 2004 sendo, atualmente, a versão corrente da API. A versão 4 do DOM já se encontra em desenvolvimento tendo sido publicado um rascunho em Dezembro de 2012 [DOMTR].

No DOM, um documento é representado numa estrutura hierárquica em que cada elemento (juntamente com as suas propriedades e caraterísticas) surge como um nó (ou como um objeto). No exemplo seguinte apresenta-se um pequeno documento HTML e o respetivo DOM.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <table border="1">
      <thead>
        <tr>
          <th>Título</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Linha 1</td>
        </tr>
        <tr>
          <td>Linha 2</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```



Código HTML e sua representação no DOM

No DOM, um documento poderá ter um nó *doctype* (opcional) mas terá sempre que ter um nó representando o documento e que servirá de raiz para os restantes nós que representam a página. É importante salientar que a especificação do DOM não define a forma como uma estrutura deverá ser representada. O DOM é um modelo lógico de um documento e poderá ser representado usando o método que for mais adequado ao momento. No exemplo seguinte e na figura 3.4 apresenta-se um código HTML e a sua representação usando uma estrutura em árvore, possibilitando uma visualização diferente.

```
<html>
  <head>
    <title>Exemplo</title>
  </head>
  <body>
    <h1>Documento</h1>
    <p>Isto é um <b>exemplo</b> de um documento.
  </body>
</html>
```

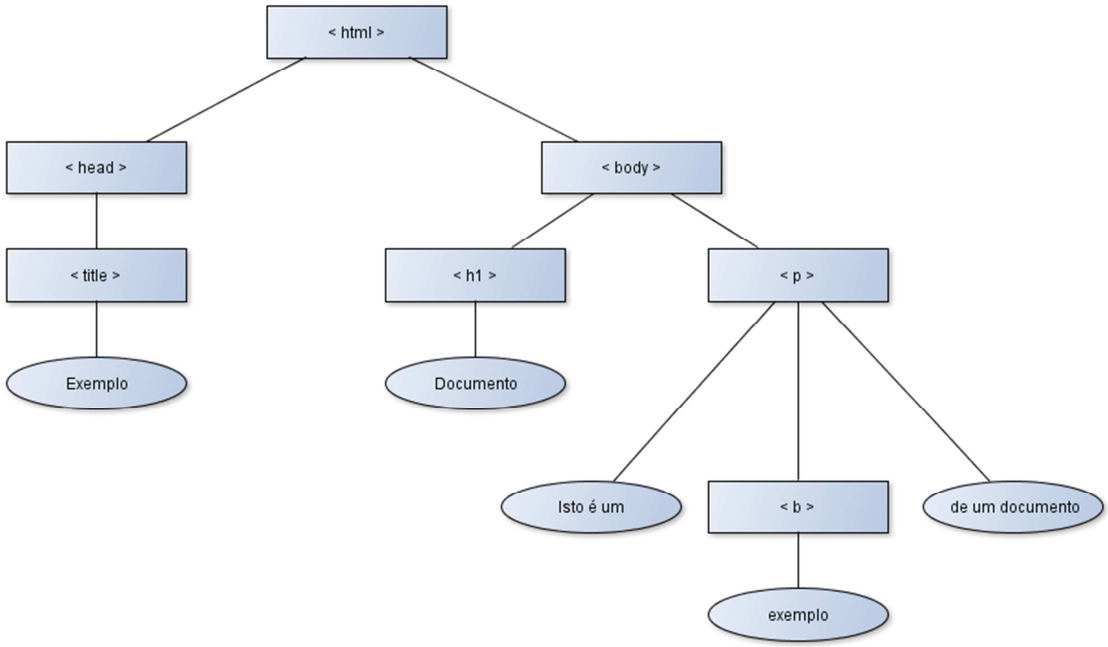


Figura 3.4 – Representação de uma página usando uma estrutura em árvore.

Como num documento existem vários tipos de elementos a sua representação no DOM também é feita usando diferentes tipos de nós. Assim, a especificação prevê a existência de diversos tipos de nós, cada um deles definindo e disponibilizando funcionalidades distintas. Além disso, um determinado tipo de nó pode conter subtipos. Por exemplo, o tipo *Document* contém os subtipos *HTMLDocument* e *XMLDocument*. Os principais tipos de nós que existem estão representados na lista seguinte:

- *Document*
- *DocumentType*
- *Comment*
- *Text*
- *Attr*
- *DocumentFragment*
- *Element*
- *CDATASection*
- *Entity*
- *EntityReference*
- *ProcessingInstruction*
- *Notation*

Alguns destes tipos podem, por sua vez, conter subtipos como, por exemplo, o tipo *Document* (já referido anteriormente) e o tipo *Element* que contém, entre outros, os subtipos *HTMLHeadElement*, *HTMLBodyElement*, *HTMLTitleElement* e *HTMLInputElement*.

Embora o DOM seja referido como sendo uma API, a especificação prevê a existência de vários módulos e API que, no seu conjunto, constituem a DOM API. Na figura 3.5 apresentam-se todos os módulos definidos na especificação.

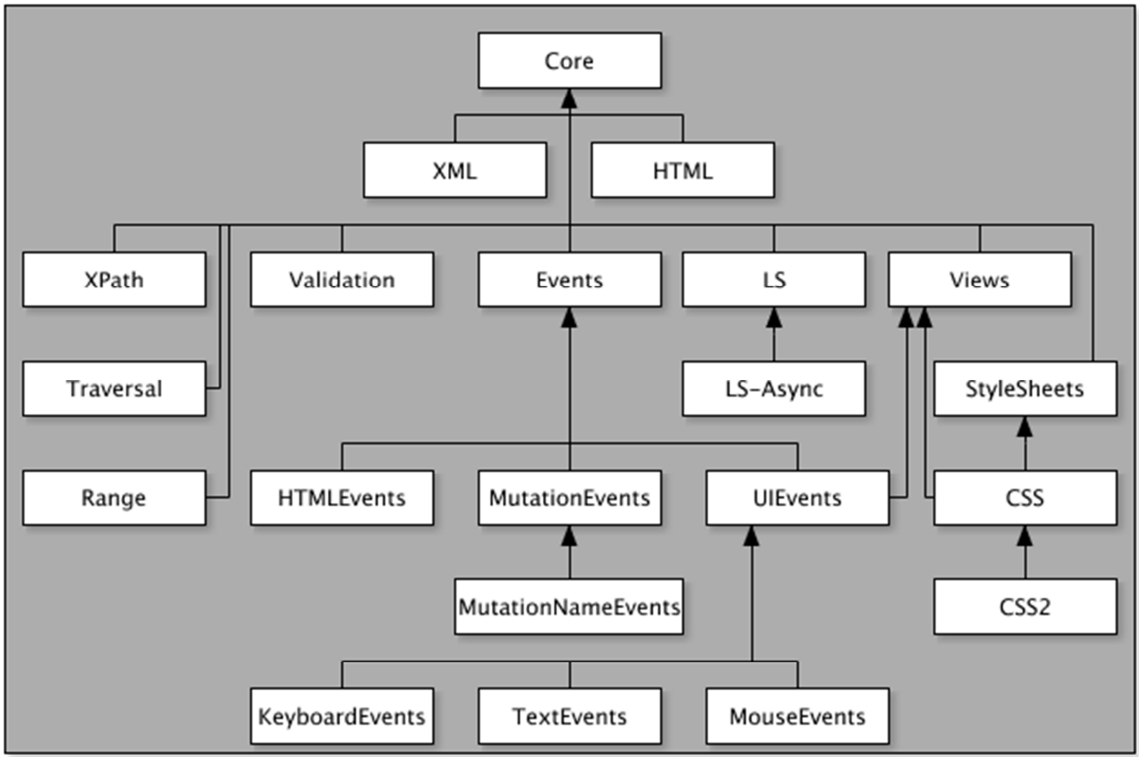


Figura 3.5 – Módulos da DOM API[10]

[10] Adaptado de [DOM]

Cada implementação do DOM (*software* responsável por analisar os documentos e disponibilizá-los via interfaces) poderá implementar apenas uma parte dos módulos ou todos, sendo que quanto mais módulos implementar maior será o grau de conformidade com a especificação.

3.23. JSON

O *JavaScript Object Notation* (JSON) é um formato estruturado, simples e flexível para representação de dados que é facilmente entendível pelos humanos e que facilita o intercâmbio de dados entre aplicações e sistemas. Apesar do JSON ser derivado do *JavaScript* (sintaticamente, um documento JSON pode ser considerado como sendo código *JavaScript*) o formato é multiplataforma e multilinguagem existindo várias linguagens de programação que disponibilizam módulos e API para a leitura e manuseamento de ficheiros JSON [11]. A especificação do JSON foi definida em 2006 através do RFC4627 e, mais recentemente, do ECMA-404 [RFC4627].

A utilização do JSON é feita em inúmeras situações destacando-se, cada vez mais, a sua utilização nas comunicações entre os *browsers* e os servidores em detrimento do XML, dando origem ao acrónimo AJAJ. Uma possível explicação para a crescente importância do JSON poderá estar no facto de se tratar de um formato mais compacto e reduzido do que o XML. No código seguinte apresenta-se um pequeno exemplo de um ficheiro JSON.

```
{
   "titulo": "The C++ Programming Language",
   "paginas": 1250,
   "disponivel": true,
   "localcompra": null,
   "localizacao":
   {
      "estante": "E1",
      "altura": "A1",
      "posicao": 4
   },
   "capitulos":
   [
      {"nome": "Capítulo 1"},
      {"nome": "Capítulo 2"},
      {"nome": "Capítulo 3"}
   ]
}
```

Analisando o exemplo anterior destacam-se algumas diferenças em relação ao XML sendo que as mais importantes são as seguintes:

[11] http://www.json.org/, Agosto 2013

- Possibilidade de definição de *arrays* (coleções de valores) usando a notação `[{Elemento_1}, {Elemento 2}, …, {Elemento_N}]`;
- Apresentação dos dados em função do seu tipo. As cadeias de carateres (*Strings*) estão delimitadas por aspas. Os números, quer sejam inteiros ou com casas decimais, são representados sem qualquer delimitador. Os valores booleanos são representados pelas constantes `true/false`. As datas e horas são consideradas como sendo do tipo *string*;
- A utilização da expressão `null` para indicar a ausência de valor atribuído.

Tal como acontece no XML é possível definir elementos compostos que, no âmbito do JSON, são chamados de objetos. No exemplo anterior, o elemento `localizacao` corresponde a um objeto composto pelos elementos `estante`, `altura` e `posicao`.

3.24. AJAX

A expressão AJAX foi criada por Jesse James Garret, em 2005, para se referir a um conjunto de tecnologias que, quando utilizadas em conjunto, permitem criar páginas e aplicações *web* com uma experiência de utilização que se aproxima à de uma aplicação *desktop* [Garret05].

O termo AJAX significa *Asynchronous JavaScript and XML* e abrange as seguintes tecnologias [McPeak07]:

- HTML/XHTML e CSS utilizados na apresentação dos conteúdos;
- DOM para permitir a adição de dinamismo às páginas;
- XML/XSLT para o tratamento dos dados;
- XMLHttpRequest usado nas comunicações assíncronas entre o cliente e o servidor;
- *JavaScript* usado como agregador e coordenador das tecnologias referenciadas nos pontos anteriores.

Com base no atrás exposto, o AJAX não deve ser visto como uma tecnologia mas sim como uma expressão que representa um conjunto de tecnologias, as quais já existiam e eram utilizadas antes do conceito AJAX ter sido apresentado pela primeira vez.

O aspeto provavelmente mais diferenciador e caraterístico do AJAX é o facto de permitir comunicações assíncronas entre o cliente e o servidor, identificado pelo termo *Asynchronous*. Com esta caraterística é possível que os *browsers* comuniquem com o servidor e não fiquem bloqueados à espera da resposta deste. Desta forma, a fluidez da aplicação não sofre pausas indesejáveis podendo o utilizador continuar a usar a aplicação, enquanto esta, em segundo plano, comunica com o servidor. Por outras palavras, a comunicação entre o cliente e o servidor é feita em segundo plano e quando o *browser*

receber a resposta, esta será tratada normalmente e sem quebras ou interrupções na aplicação.

Nas páginas e aplicações *web* tradicionais (nas quais não existe comunicação assíncrona) as ações dos utilizadores, de uma forma geral, resultam na necessidade de se efetuar uma comunicação com o servidor o qual, após a receção do pedido e o subsequente processamento, devolverá uma nova página que será apresentada ao utilizador. Durante este período de tempo o utilizador não verá qualquer atividade por parte do *browser* nem poderá interagir com ele (na prática, o utilizador pode continuar a interagir com o *browser* desde que abra uma nova página). Na figura 3.6 apresenta-se um esquema do procedimento atrás descrito.

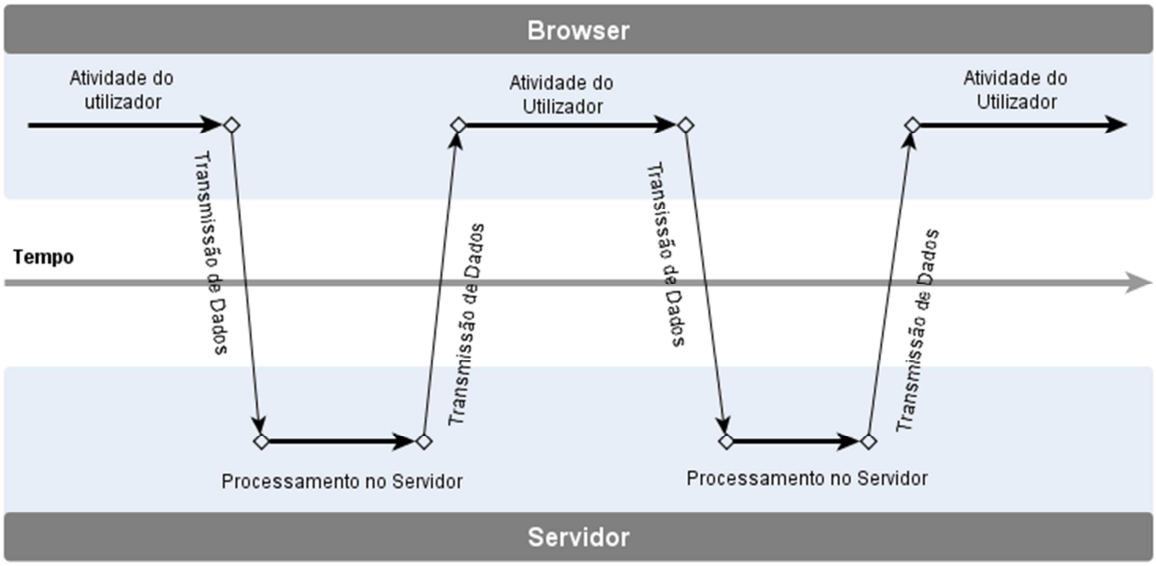


Figura 3.6 – Comunicação síncrona entre o cliente e o servidor[12]

Utilizando o modelo AJAX as ações dos utilizadores são encaminhadas para o AJAX *Engine* (que, na prática, se trata de código *JavaScript* que é executado sempre que seja necessário efetuar um pedido ao servidor) o qual fará o pedido assíncrono ao servidor e permitirá que o utilizador continue a operar com o *browser*. Logo que o AJAX *Engine* receba a resposta ao pedido efetuado, produzirá as alterações necessárias na página, não sendo necessário que a página seja totalmente recarregada.

Na figura 3.7 apresenta-se o esquema de uma aplicação *web* usando o modelo AJAX, na qual se identificam as interações entre o utilizador e o AJAX *Engine* e entre este último e o servidor.

[12] Adaptada de [Garret05]

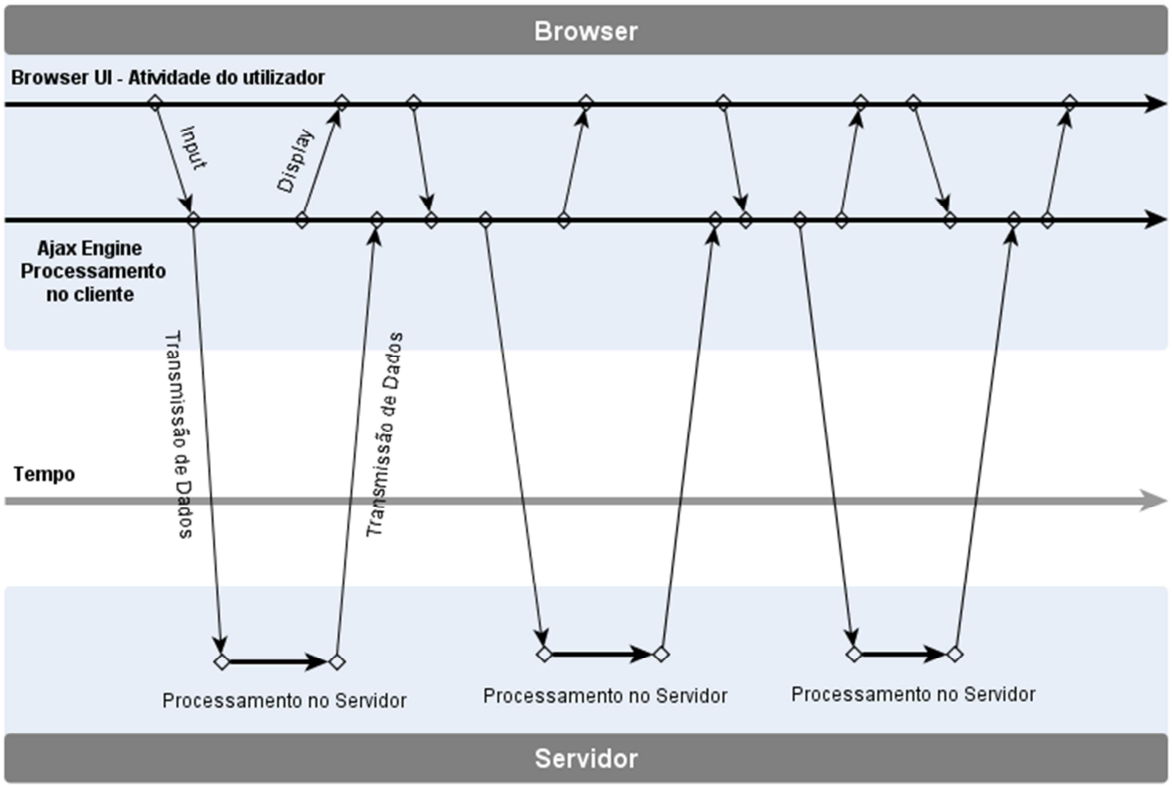


Figura 3.7 – Comunicação assíncrona entre o cliente e o servidor[13]

Comparativamente com o modelo tradicional de aplicações e páginas *web*, o AJAX introduz uma camada adicional na arquitetura que possibilita todo o sistema de comunicações assíncronas. A figura 3.8 apresenta uma comparação entre os dois modelos:

A outra tecnologia do AJAX que também desempenha uma tarefa de vital importância é o *JavaScript* uma vez que é com o recurso a esta linguagem que os processos descritos na secção anterior são possíveis. É através do *JavaScript* que são feitas as ligações com o servidor e é o *JavaScript* que processa e dá seguimento às respostas recebidas do servidor. Para que as comunicações assíncronas entre o *browser* e o servidor sejam efetuadas é necessário que o *JavaScript* utilize o objeto especial *XMLHttpRequest*. Sendo através deste objeto que se consegue a assincronia e sendo esta a caraterística mais marcante do AJAX, pode-se afirmar que o objeto *XMLHttpRequest* é o núcleo do AJAX.

Quando o termo AJAX foi proposto, o XML desempenhava um papel muito importante em todo o processo, uma vez que, juntamente com o XSLT, era utilizado para a comunicação dos dados. Contudo, o XML não é a única forma de tratamento dos dados podendo ser utilizados outros formatos, destacando-se, neste campo, a utilização do formato JSON.

Por cima destas tecnologias já apresentadas trabalham o HTML/XHTML, o CSS e o DOM para que as páginas sejam atualizadas e alteradas sem ser necessário que as mesmas sejam totalmente reconstruídas pelo servidor ou pelo *browser*.

[13] Adaptada de [Garret05]

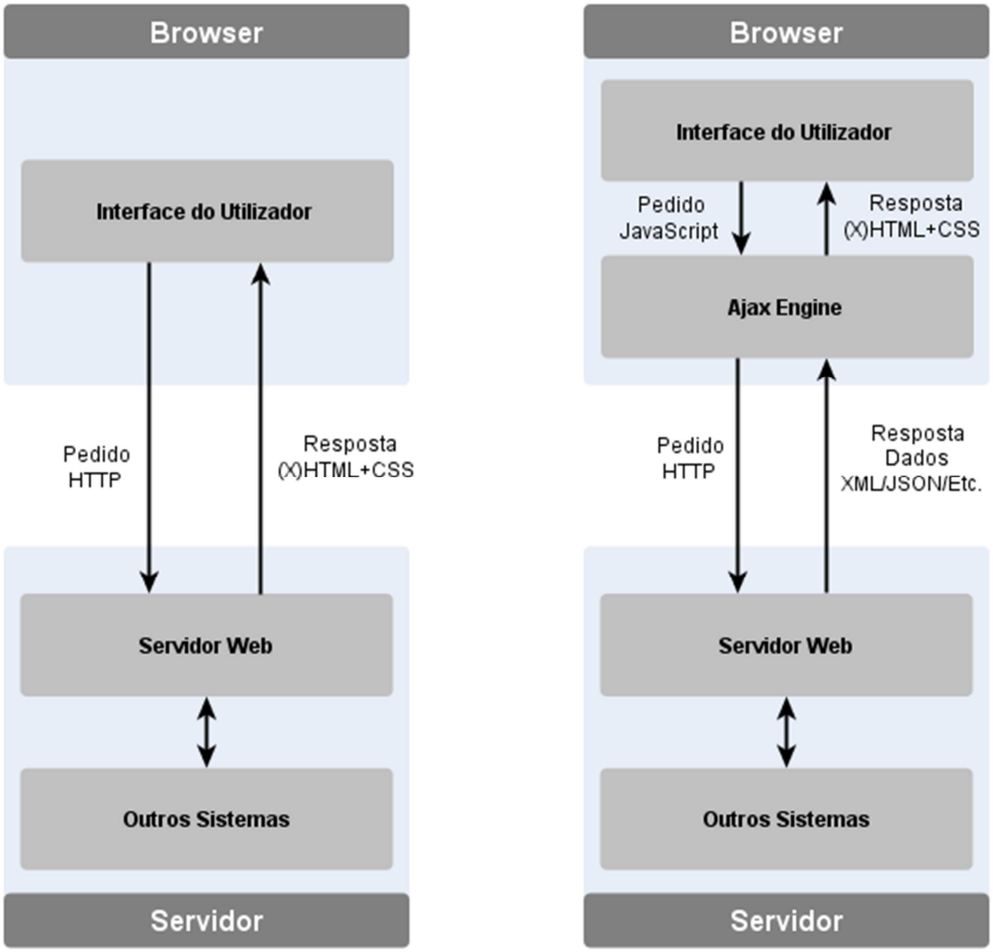


Figura 3.8 – Comparação da arquitetura de uma página no modelo tradicional e usando o AJAX[14]

A adoção do modelo AJAX no desenvolvimento de aplicações *web* tem tido um grande impacto neste tipo de aplicações assim como no seu desenvolvimento. O facto de se passar a poder distribuir aplicações *web* com uma experiência de utilização muito próxima da experiência de utilização das aplicações *desktop* pode significar uma transferência ou disponibilização destas últimas para a plataforma *web*. O AJAX acaba por ser mais um fator a influenciar a migração das aplicações *desktop* para aplicações *web*. Por outro lado, o AJAX veio reforçar a importância da linguagem *JavaScript*, passando a ser a principal linguagem a ser executada do lado do cliente. Adicionalmente, o AJAX introduz profundas alterações ao nível da arquitetura e do desenvolvimento. Nas aplicações *web* tradicionais é normal os utilizadores premirem o botão Anterior ou Refrescar para voltarem para a página anterior à atual ou para que a atual seja novamente carregada. Numa aplicação AJAX este tipo de operação torna-se muito mais complicado ou impossível, atendendo à natureza deste tipo de aplicações e ao facto de muitas vezes ainda estarem a decorrer pedidos ao servidor quando se primem os referidos botões. A adoção do modelo AJAX também significa, muitas vezes, que uma aplicação *web* será executada sempre no mesmo URL, ocorrendo as atualizações sempre dentro da mesma página. Este paradigma representa uma mudança radical relativamente às aplicações *web* tradicionais. As aplicações AJAX também

[14] Adaptada de [Garret05]

representam um grande desafio para os programadores uma vez que passam a ter à sua disposição um conjunto de tecnologias que lhes permitem desenvolver aplicações mais ricas, complexas e interativas.

Na lista seguinte apresentam-se alguns aspetos positivos e vantagens associadas ao AJAX [Teague06]:

- O AJAX permite criar aplicações *web* mais ricas e fluídas, proporcionando uma melhor experiência de utilização;
- Permite uma aproximação efetiva entre as aplicações *desktop* e *web*, agrupando as vantagens de ambas;
- Ao não ser necessário reconstruir e recarregar toda a página como resposta às ações dos utilizadores, obtém-se um melhor aproveitamento da largura de banda tornando as aplicações mais rápidas. Como apenas circula uma parte da informação necessária para atualizar as páginas, a quantidade de informação a circular será muito menor;
- Como resultado do ponto anterior, o AJAX permite uma maior e melhor interatividade entre os utilizadores e as aplicações;
- O AJAX é usado pelas maiores empresas a operar na Internet, o que pode ser sinónimo de que estamos perante um modelo com muitas vantagens, eficiente e com muito futuro.

Apesar de apresentar um conjunto de vantagens grande, diversificado e importante, o AJAX também tem aspetos negativos, os quais devem ser tidos em conta pelos programadores. Assim, as principais desvantagens do AJAX são as seguintes [Teague06]:

- Um dos principais aspetos negativos apontados ao AJAX é a dificuldade que os *browsers* apresentam no momento em que se pretende voltar para páginas anteriores. Como as ações dos utilizadores provocam alterações nos estados das páginas e como o botão Anterior é usado para voltar para a página que estava a ser exibida antes da atual (e não para voltar para o estado anterior), cria-se uma situação complicada de ultrapassar. No entanto, existem diversas formas de se ultrapassar esta limitação e a especificação do HTML5 já contempla o retrocesso em páginas ou aplicações *web* que sofram alterações no seu estado;
- Embora, de uma maneira geral, a utilização das tecnologias do AJAX aumente a velocidade de uma aplicação, este tipo de *software* poderá sofrer mais com eventuais falhas nas comunicações, uma vez que uma aplicação AJAX, em virtude da sua natureza mais interativa, tende a estabelecer um maior número de comunicações com o servidor;
- Tal como acontece com qualquer aplicação *web* dinâmica, os motores de busca não conseguem lidar devidamente com as aplicações AJAX. Como este projeto incide sobre uma aplicação interna a uma organização, isto não deverá ser um problema. Contudo, para páginas públicas será conveniente criarem-se mecanismos alternativos que facilitem a pesquisa e a indexação de páginas;

- Para que uma aplicação AJAX possa funcionar, é obrigatório que os *browsers* suportem as versões mais recentes das tecnologias e que tenham a funcionalidade de execução do *JavaScript* ativada;
- A utilização do AJAX em aplicações *web* tende a aumentar consideravelmente a complexidade das aplicações, podendo refletir-se nos tempos de desenvolvimento e de manutenção;
- O conceito *Same Origin Policy* que impede as linguagens que correm no lado do cliente (quase sempre *JavaScript*) de comunicarem com domínios, protocolos ou portos diferentes do da página atual também pode causar problemas e limitações na criação de páginas AJAX. No entanto, existem formas de se ultrapassar esta limitação sendo uma delas a utilização do JSONP.

Apesar de apresentar alguns aspetos negativos que podem criar algum transtorno, acreditamos que o AJAX é um paradigma fundamental no desenvolvimento de aplicações *web*, principalmente quando se pretende que estas se aproximem da experiência de utilização proporcionada pelas aplicações *desktop*.

3.25. *JAVASCRIPT*

O *JavaScript* é uma linguagem de programação que se carateriza por ser interpretada, orientada a objetos, baseada em protótipos, com ênfase nas funções e de tipagem dinâmica. Embora seja executada, maioritariamente, em *browsers*, também pode correr noutros tipos de ambientes [Reid13].

A linguagem começou a ser desenvolvida em 1995 por um programador da *Netscape* chamado Brendan Eich tendo sido oficialmente apresentada nesse mesmo ano juntamente com o *browser Netscape Navigator 2.0*. Devido ao enorme sucesso que a linguagem alcançou, a *Microsoft* apresentou uma implementação semelhante ao *JavaScript* (à qual chamou *JScript*) juntamente com o navegador *Internet Explorer 3.0*. Com este lançamento por parte da *Microsoft* passaram a existir duas implementações para a mesma linguagem e como não havia nenhuma entidade independente que regulasse a linguagem, corria-se o risco de que as duas implementações seguissem caminhos totalmente distintos, o que poderia tornar-se numa situação bastante complicada tanto para os programadores como para a própria linguagem. Com o objetivo de evitar este inconveniente, o *JavaScript* foi submetido à ECMA em 1996 para que as especificações e caraterísticas da linguagem passassem a ser definidas por uma entidade independente [Ecma262;Zakas12]. Após esta submissão, a primeira edição do *standard* foi publicada em 1997 com o nome de ECMA-262, a qual definiu uma nova linguagem chamada de *ECMAScript*. Esta nova linguagem passou, a partir dessa data, a ser a especificação e referência, passando o *JavaScript* e o *JScript* a ser uma implementação do *ECMAScript*. Desde o lançamento da primeira edição do ECMA-262 até aos dias de hoje já foram publicadas 4 atualizações, sendo a versão mais recente a ECMA-262 *Edition 5.1* que foi publicada em Junho de 2011. Entretanto,

prosseguem os trabalhos tendo em vista a publicação de uma nova edição da especificação que se chama *ECMAScript Harmony*. Assim, embora muitas vezes o termo *JavaScript* seja usado como referência, na realidade a referência é o ECMAScript sendo o *JavaScript* uma das implementações da especificação.

O *JavaScript* possui diversas caraterísticas que o tornam numa linguagem simultaneamente fácil de aprender mas difícil de dominar. Na lista seguinte apresenta-se uma descrição mais detalhada das caraterísticas mais importantes do *JavaScript* [Reid13]:

- **Tipagem dinâmica:** No *JavaScript* uma variável pode, num dado momento, representar uma cadeia de carateres e, na linha de código seguinte ser associada a um valor numérico. Isto significa que o tipo das variáveis é definido/calculado no momento em que lhe é atribuído um valor, ou seja, se uma variável referir um valor numérico, então o seu tipo será numérico mas se, noutra zona do código fonte a mesma variável já referir uma cadeia de carateres, então o seu tipo passará a ser *string*;
- **Linguagem dinâmica:** O dinamismo da linguagem está associado às variáveis (ponto anterior) e à possibilidade de se poder executar código *JavaScript* mediante a utilização da função `eval()`, a qual aceita como parâmetros uma cadeia de carateres contendo uma expressão *JavaScript*. Isto significa que é possível usando *JavaScript*, gerar um trecho de código *JavaScript* que poderá ser executado logo após a sua criação;
- **Linguagem de *Scripting*:** Uma aplicação escrita em *JavaScript* pode ser considerada como um conjunto de *scripts* que serão interpretados e executados por um interpretador (ou *engine*). Apesar de esta abordagem apresentar algumas desvantagens quando comparada, por exemplo, com linguagens compiladas, tem a vantagem de tornar o código *JavaScript* altamente portável, podendo ser interpretado/executado em qualquer dispositivo que possua um interpretador;
- **Orientada a objetos:** Embora possa ser considerada uma linguagem orientada a objetos, a forma como implementa esta caraterística difere bastante da forma como outras linguagens o fazem. Ao contrário da utilização de conceitos de herança, no *JavaScript* usa-se uma técnica diferente chamada de *prototyping* (ou *Prototype-Based* ou *Prototypal Inheritance*);
- **Funções de primeira classe:** No *JavaScript* as funções são elementos de primeira classe, o que significa que podem ter as suas propriedades, conter outras funções, ser passadas como parâmetros ou devolvidas de outras funções;
- **Sintaxe:** Tem uma sintaxe semelhante à das linguagens da família C/C++ sendo, tal como elas, uma implementação de uma especificação.

Tal como foi referido na secção anterior, o *JavaScript* é uma linguagem interpretada, o que significa que, para que possa ser executada, é necessária a existência de um interpretador (*JavaScript Engine*) que tem a seu cargo a interpretação e execução do código. Atualmente existem diversos *JavaScript Engines* (p. ex. *V8*, *Rhino*, *SpiderMonkey* ou *JavaScriptCore*) os quais apresentam diferentes níveis de conformidade com a especificação ECMA-262. Se um *JavaScript Engine* implementar o ECMA-262 *Edition 3* significa que, pelo menos, está conforme todas as características definidas na referida especificação podendo, além disso,

implementar outra especificações posteriores ou, ainda, extensões próprias. Como a última edição do ECMA-262 data de Junho de 2011, regra geral, os interpretadores associados aos principais *browsers* implementem a edição 5.1 do ECMA-262.

O facto de existirem vários interpretadores do *JavaScript* pode, à primeira vista, parecer uma vantagem dado que a existência de concorrência promove, frequentemente, uma evolução das aplicações a um ritmo mais elevado. Em contrapartida, criam-se situações em que o mesmo código fonte pode gerar resultados diferentes em diferentes interpretadores. Um exemplo típico destas situações são as interfaces DOM que, por não fazerem parte do ECMA-262 podem ser implementadas de forma diferente de interpretador para interpretador e, por consequência, de *browser* para *browser*. Uma forma de se ultrapassar esta dificuldade passa pela utilização de uma das diversas bibliotecas *JavaScript* existentes (p. ex. *jQuery*, *Prototype*, *script.aculo.us*, *MooTools*) as quais permitem libertar os programadores das preocupações relacionadas com as especificidades dos interpretadores *JavaScript*.

Usado no âmbito de uma aplicação *web*, o *JavaScript* tem inúmeras utilidades, desempenhando um papel fundamental na implementação do dinamismo das páginas. Com o recurso ao *JavaScript* é possível realizar, entre outras, as seguintes tarefas [Wilton10; Morris11; Goodman07]:

- Emissão de mensagens de alerta;
- Animação de elementos visuais;
- Efetuar validações prévias dos dados, antes destes serem enviados para o servidor;
- Definir ações a executar quando uma página está totalmente carregada;
- Estabelecer comunicações com o servidor;
- Responder às ações dos utilizadores e executar código que, por exemplo, manipula os estados dos elementos de uma página;
- Melhorar as interfaces geradas com HTML tornando-as mais dinâmicas e fáceis de usar.

Apesar do *JavaScript* apresentar uma lista considerável de vantagens e utilidades, também existem algumas limitações e aspetos a ter em conta quando se desenvolve código em *JavaScript*. Contudo, é importante ressalvar que algumas das limitações foram criadas com o propósito de proteger os utilizadores de ataques e acessos indesejáveis. Assim, o *JavaScript* apresenta o seguinte conjunto de limitações/desvantagens [Goodman07;Suehring13]:

- Usando exclusivamente *JavaScript*, não é possível ler ou gravar ficheiros no disco dos computadores dos clientes;
- Também não é permitido enviar *e-mails* de forma anónima ou iniciar a execução de uma aplicação no computador do cliente;
- Se a execução do *JavaScript* estiver desativada no *browser*, o código nunca será executado;

- O *JavaScript* não deve ser utilizado como a única forma de implementação de medidas de segurança uma vez que o código fonte pode ser facilmente acedido e analisado;
- Como o código é executado do lado do cliente, e apesar de todas as medidas de segurança, torna-se um alvo apetecível para ataques e acessos não autorizados;
- O facto de diferentes interpretadores poderem gerar resultados diferentes deve ser visto como um aspeto bastante negativo e que merece muita atenção por parte dos programadores;
- Alguns aspetos da linguagem podem ser de difícil compreensão para programadores que venham de outras linguagens. A herança por prototipagem, os *closures* ou contextos de execução são conceitos que podem ser difíceis de dominar e fonte de problemas e atrasos no desenvolvimento.

Apesar das desvantagens, o *JavaScript* tem desempenhado um papel fundamental, tendo-se tornado, ao longo dos tempos, na linguagem de *scripting* mais utilizada na Internet.

3.26. *XMLHttpRequest*

O *XMLHttpRequest* (também conhecido como XHR) é um objeto que é disponibilizado pelos principais *browsers* e que permite o intercâmbio de dados com o servidor [XHR]. Com este objeto é possível efetuar, entre outras, as seguintes operações [Ullman07]:

- Efetuar um pedido assíncrono ao servidor;
- Receber dados de um servidor como resposta a um pedido ou envio de dados;
- Enviar dados para o servidor.

Sempre que um *browser* necessitar de estabelecer um canal de comunicação com um servidor terá que utilizar uma instância deste objeto. Embora o nome sugira que o objeto apenas trabalha com dados no formato XML, na realidade, a informação processada pelo objeto poderá estar noutro formato diferente como, por exemplo, o formato JSON.

O *XMLHttpRequest* desempenha um papel fundamental no âmbito de uma aplicação *AJAX* uma vez que é com o recurso a ele, através da sua implementação em *JavaScript*, que as comunicações assíncronas são estabelecidas. Para que seja possível trabalhar com o objeto, este disponibiliza diversos métodos (`open`, `send`, `abort`) eventos (`onreadystatechange`) e propriedades (p. ex. `status`, `statustext`, `responseXML`) [Ullman07].

O conceito associado a este objeto já existe desde os finais da década de 90 (tendo surgido, ao longo do tempo diversas implementações) mas só em Abril de 2006 é que foi publicado pelo W3C o primeiro rascunho da especificação. O objetivo deste documento era o de

normalizar as características dos objetos até aí existentes. Desde essa data já foram publicados diversos documentos do W3C relacionados com o *XMLHttpRequest* sendo o último datado de Dezembro de 2012. Apesar das várias publicações do W3C relativas ao objeto, o processo de estandardização ainda não está totalmente concluído uma vez que ainda não foi disponibilizada a versão final da especificação. Contudo, os objetivos iniciais do primeiro rascunho estão alcançados atendendo a que já existe uma grande padronização do objeto nos principais *browsers*.

4. ANÁLISE DOS *FRAMEWORKS*

Apesar da quantidade de plataformas disponíveis para o desenvolvimento do código que é executado no lado do servidor ser bastante extensa (p. ex. *Ruby*, *Java*, PHP, .NET, *Python* ou *Perl*) apenas serão analisadas as plataformas .NET, *Java* e PHP uma vez que são as mais utilizadas[15].

4.1. .NET

A plataforma .NET é um *framework* criado pela *Microsoft* que permite o desenvolvimento de aplicações (*web*, *desktop*, *mobile*/micro) e serviços[16]. A primeira versão da plataforma foi disponibilizada em 2002 e, desde então, têm surgido várias atualizações, sendo a versão mais recente a 4.5 disponibilizada em 2012[17]. O *framework* é distribuído sob uma licença proprietária da *Microsoft* embora possa ser usado livremente sem ser necessário adquirir qualquer tipo de licença comercial. O desenvolvimento da plataforma teve, na sua génese, o seguinte conjunto de princípio e objetivos [Troelsen12]:

- Permitir elevados níveis de interoperabilidade entre a plataforma e o código desenvolvido para ela com o *software* já existente e desenvolvido ao abrigo de outras tecnologias;
- Disponibilizar uma plataforma de desenvolvimento que possibilitasse a utilização de diversas linguagens de programação e que permitisse que o código desenvolvido numa determinada linguagem pudesse ser utilizado, sem qualquer restrição, por código desenvolvido noutras linguagens;
- Disponibilizar uma plataforma de desenvolvimento que reduzisse os problemas relacionados com a distribuição de aplicações e com os conflitos de versões;
- Disponibilizar uma extensa biblioteca que facilitasse o trabalho dos programadores, protegendo-os das complexidades e especificidades relacionadas com a programação de baixo nível e a interação com diversos tipos de dispositivos. Simultaneamente, a biblioteca deveria permitir um modelo de utilização consistente, independentemente do ambiente em que o *software* operasse (*web*, *desktop* ou *mobile*.);

A base da plataforma .NET é constituída, fundamentalmente, por dois grandes componentes [Troelsen12]:

- *Common Language Runtime* (CLR)
- .NET *Framework Class Library*

---

15 http://w3techs.com/technologies/overview/programming_language/all, Fevereiro 2013
http://trends.builtwith.com/framework, Fevereiro 2013

16 http://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.aspx, Fevereiro 2013

17 http://msdn.microsoft.com/en-us/library/bb822049%28v=vs.110%29.aspx, Fevereiro 2013

O CLR é uma máquina virtual que constitui a base da plataforma .NET. É este componente que faz a gestão da memória, coordenação de *threads* e outras tarefas de baixo nível, assim como também é responsável pela compilação, verificação e execução do código. O CLR contém, ainda, dois elementos que desempenham um papel muito importante: O *Common Type System* (CTS) e o *Common Language Specification* (CLS). Uma das funções do CTS é garantir que o código que é executado pelo CLR está conforme as regras impostas pela plataforma. Por outro lado, o CTS também especifica a forma como os tipos de dados devem ser definidos, declarados, usados e geridos bem como a forma como os tipos podem interagir entre eles. O CLS, por seu lado, é uma especificação que define um conjunto de funcionalidades base que são suportadas pelo CLR e que as linguagens deverão implementar. Se uma aplicação ou API respeitar as regras impostas pelo CLS é garantido que poderá ser usada por outras aplicações ou API na plataforma .NET, independentemente da linguagem em que tiverem sido desenvolvidas.

O .NET *Framework Class Library* é um conjunto de classes que estão disponíveis para todas as linguagens da plataforma .NET e serve de base e suporte para o desenvolvimento do *software* na plataforma. Através desta biblioteca os programadores têm à sua disposição uma enorme quantidade de funcionalidades como, por exemplo, a interação com o sistema de ficheiros, gestão de *threads*, classes para tratamento de coleções e listas de valores, interação com bases de dados ou interação com dispositivos externos. Para além das classes, o .NET *Framework Class Library* também define interfaces e constantes que podem ser usadas para desenvolver qualquer tipo de *software*. Na figura 4.1 apresenta-se um esquema simplificado da plataforma .NET.

Identificam-se o CLS, CTS e o CLR assim como a *.NET Framework Base Class Library*. Sobre esta estrutura são desenvolvidas as restantes funcionalidades da plataforma. Estas funcionalidades poderão ser utilizadas pelas linguagens de programação permitidas na plataforma .NET.

Como foi referido anteriormente, uma das funções do CLR é a execução do código que tenha sido desenvolvido e compilado na plataforma .NET. Ao contrário de outros sistemas, o processo de compilação de um código escrito numa das linguagens da plataforma .NET não resulta na geração de código nativo do sistema operativo (situação que ocorre, por exemplo, quando se compila uma aplicação em C++ ou *Delphi*). No .NET o processo de compilação gera um código intermédio chamado de *Microsoft Intermediate Language* (MSIL)[18]. Este código intermédio é totalmente independente da arquitetura e do sistema operativo, podendo ser executado em qualquer dispositivo desde que disponha de uma versão do .NET. Quando o CLR tem que executar um determinado código, não o executa diretamente, procedendo, pelo contrário, a uma compilação do código intermédio em código nativo do sistema operativo onde será executado. O componente responsável por esta compilação *on-the-fly* é o *Microsoft .NET Framework Just in Time Compiler*. Com este processo obtém-se um desempenho muito elevado uma vez que o código intermédio nunca será interpretado nem executado, mas sim compilado para código nativo, sendo este o código que, efetivamente, será executado.

[18] O MSIL também pode ser chamado de *Intermediate Language* (IL) ou *Common Intermediate Language* (CIL)

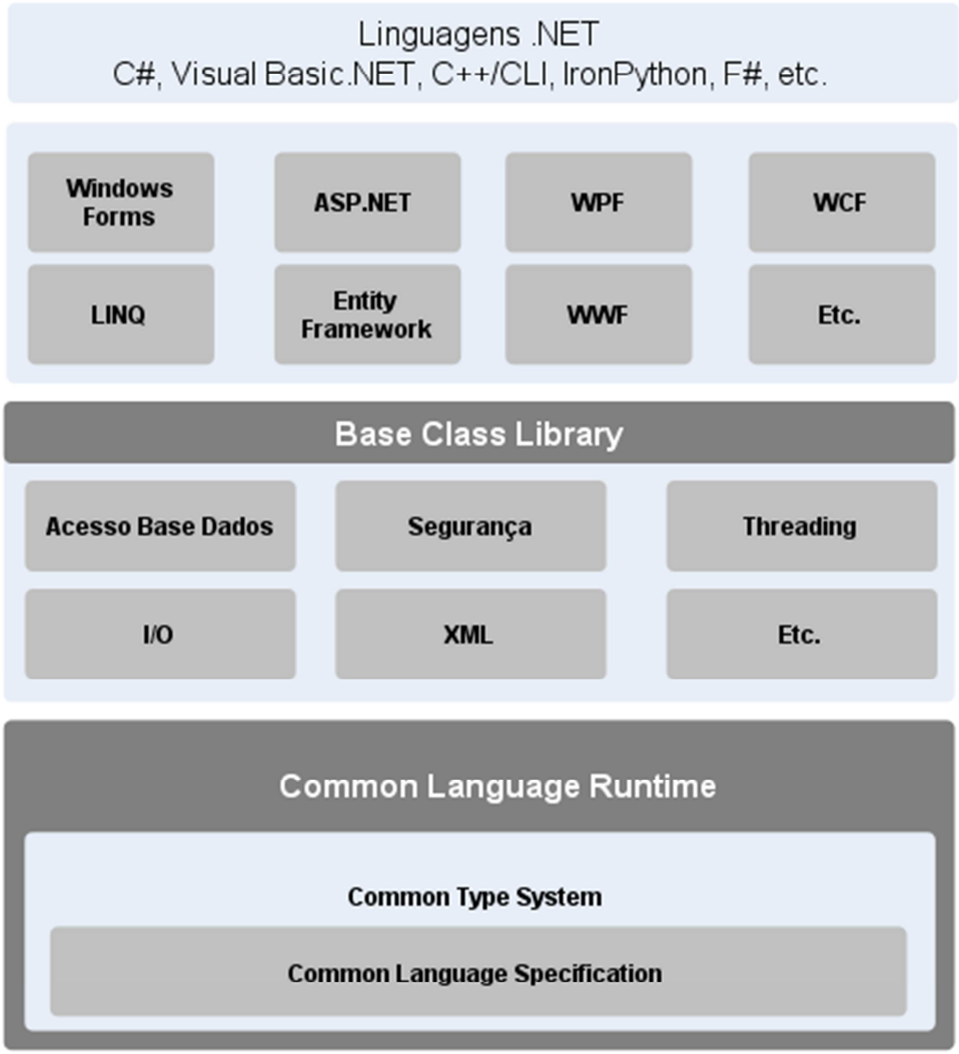


Figura 4.1 – Esquema simplificado da plataforma .NET[19]

Na plataforma .NET os programadores podem desenvolver *software* utilizando várias linguagens (p. ex. C#, *Visual Basic.NET*, F# ou C++/CLI) e para diversas plataformas (*desktop*, *web*, *mobile*, micro). Na lista seguinte apresentam-se alguns dos tipos de aplicações que podem ser desenvolvidos usando a plataforma .NET e os seus componentes:

- Aplicações de linha de comandos;
- Aplicações de *desktop* com interface gráfica de utilizador. Este tipo de aplicações pode ser desenvolvido usando as tecnologias *Windows Forms*, *Windows Presentation Foundation* ou, ainda, usando a nova tecnologia *Windows Runtime* (WinRT) para o desenvolvimento de aplicações específicas para o *Microsoft Windows 8*;

[19] Adaptado de [Troelsen12]

- Aplicações *Web*. Estas podem ser desenvolvidas usando o ASP.NET *Web Forms*, ASP.NET MVC ou ASP.NET *Web Pages*;
- Serviços usando a nova biblioteca *Web* API ou o *Windows Comunications Foundation* ou, ainda, o *Windows Service Applications*;
- Aplicações para dispositivos móveis ou de reduzida capacidade de processamento usando, para o efeito, o *.NET MicroFramework* ou o *.NET Compact Framework* (que, embora não sejam a plataforma .NET, são versões mais reduzidas e simples);
- Aplicações para serem alojadas e executadas a partir da *cloud*, mais concretamente, usando a infraestrutura *Windows Azure*.

Na lista seguinte apresentam-se as principais vantagens que são reconhecidas à plataforma .NET [Nagel12]:

- Disponibiliza uma plataforma e uma extensa biblioteca que facilitam o desenvolvimento e abstraem o programador das funções de mais baixo nível e das suas complexidades;
- Permite o desenvolvimento de *software* utilizando diferentes linguagens de programação. Apesar do código intermédio gerado ser igual, independentemente da linguagem utilizada, cada programador tem a possibilidade de selecionar a linguagem em que sente mais à vontade;
- Numa só plataforma é possível desenvolver diversos tipos de *software*;
- As ferramentas de desenvolvimento associadas à plataforma (*Visual Studio*) são de elevada qualidade e muito produtivas;
- Permite o aproveitamento do código desenvolvido usando outras tecnologias da *Microsoft*;
- O desenvolvimento de aplicações usando a plataforma .NET tende a ser mais rápido do que utilizando outras plataformas;
- Existe uma grande integração com o sistema operativo *Microsoft Windows* bem como com outras aplicações desenvolvidas pela empresa (destacando-se, fundamentalmente, o *Microsoft SQL Server* e o *Microsoft Office*).

Relativamente às desvantagens podemos apontar as seguintes:

- A plataforma .NET, ao estar intimamente relacionada com o *Microsoft Windows* dificulta ou impossibilita a utilização noutros sistemas. Apesar de existirem implementações do .NET para *Linux* e *Apple* (Projeto *Mono*) estas não estão tão evoluídas quanto a implementação de referência da *Microsoft*;
- Custos de licenciamento. Apesar de existirem versões *Express* das ferramentas de desenvolvimento, estas não estão preparadas para o desenvolvimento por equipas, o que obriga à aquisição das versões pagas;
- Embora seja de utilização livre, a plataforma .NET é propriedade da *Microsoft* e é de código fechado;
- O desempenho de uma aplicação que corra na plataforma .NET poderá não ser tão rápido quanto o desempenho de uma aplicação nativa.

Como este projeto se centra no desenvolvimento para a plataforma *web*, importa destacar, de entre as diversas tecnologias disponibilizadas pelo .NET, o ASP.NET uma vez que é esta que permite o desenvolvimento deste tipo de aplicações. O ASP.NET é uma tecnologia *server-side* proposta pela *Microsoft* para o desenvolvimento de aplicações *web*, assente na plataforma .NET e que representa uma evolução relativamente à tecnologia anterior (ASP – *Active Server Pages*). A primeira versão do ASP.NET surgiu no início de 2002 e a versão mais recente é a 4.5, tendo sido lançada em Agosto de 2012, juntamente com a plataforma .NET 4.5. O ASP.NET permite várias abordagens para o desenvolvimento de aplicações *web*, destacando-se as seguintes [Hanselman13]:

- ***Web Pages*** **-** Permite a criação de aplicações *web* simples e dinâmicas, utilizando o modelo *Single Page*;
- ***Web Forms*** **-** Utiliza o modelo *Event Driven*, permitindo uma elevada produtividade, através da reutilização de controlos e da facilitação de inclusão de dados nas páginas;
- **MVC –** Metodologia baseada no padrão *Model-View-Controller*, o que permite uma efetiva separação dos componentes.

Embora se tratem de três metodologias diferentes para construção de páginas e aplicações *web*, todas elas se baseiam na tecnologia ASP.NET. Atendendo à natureza da aplicação que pretendemos converter para a *web*, apenas iremos analisar mais detalhadamente as metodologias *Web Forms* e MVC. Na figura 4.2 apresenta-se a forma como estas três metodologias se enquadram no âmbito do ASP.NET.

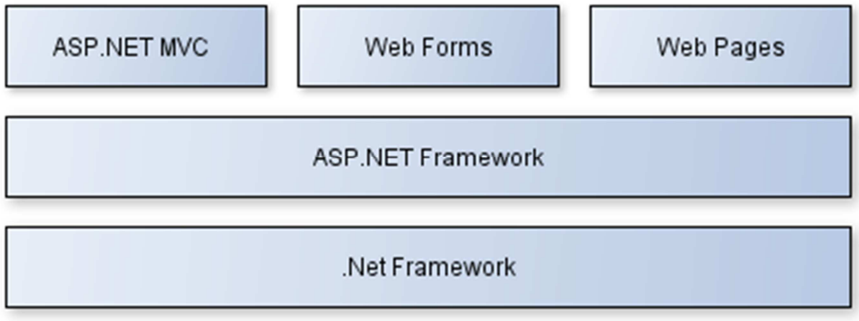


Figura 4.2 – Plataforma ASP.NET[20]

Qualquer uma destas tecnologias são compatíveis com qualquer linguagem que seja suportada pela plataforma .NET e o código desenvolvido poderá utilizar todas as funcionalidades disponibilizadas.

[20] Adaptado de [Hanselman13]

4.1.1. ASP.NET WEB FORMS

A tecnologia *Web Forms* foi a primeira das três a ser desenvolvida e, quando surgiu, permitiu um conjunto vasto de melhoramentos em relação à tecnologia anterior da *Microsoft*. Carateriza-se por implementar o modelo *Event Driven* (à semelhança do desenvolvimento para a plataforma *desktop*, em que a execução da aplicação é determinada por eventos e/ou mensagens). A base dos *Web Forms* são componentes que, à semelhança dos *Windows Forms* para o desenvolvimento para *desktop* podem ser "arrastados" para as páginas e permitem a definição de propriedades e a associação de eventos[21]. Estes componentes da UI, numa fase posterior, serão traduzidos em código HTML que será enviado para os clientes, tendo em conta as respetivas especificidades e caraterísticas.

Atendendo às semelhanças com o desenvolvimento para *desktop*, esta metodologia poderá apresentar muitos benefícios caso seja utilizada por programadores que prefiram um desenvolvimento mais declarativo e baseado em controlos de UI, sendo bastante popular como ferramenta de desenvolvimento RAD. Ao disponibilizar um ambiente de desenvolvimento baseado em conceitos *drag and drop*, pode ser considerado como sendo um *framework* muito adequado para programadores que pretendam dar os primeiros passos no desenvolvimento *web* ou que pretendam desenvolver rapidamente uma aplicação.

As principais caraterísticas e vantagens dos *Web Forms* estão resumidas na lista seguinte[22] [Esposito11;Esposito09]:

- Os *Web Forms* são compilados e executados no servidor, o que gera o código HTML que será apresentado do lado do cliente, levando em consideração as características e especificidades;
- Permitem uma separação entre o código HTML da *user interface* (UI) e a lógica da aplicação. Geralmente, os aspetos visuais são colocados num ficheiro com a extensão ASPX e a lógica num ficheiro separado (por exemplo, VB ou CS caso seja desenvolvido utilizando o *Visual Basic.NET* ou C#, respetivamente);
- São constituídos por um conjunto vasto de elementos (ou controlos) visuais, que encapsulam o adequado HTML, *Javascript* e CSS. Estes controlos respondem a vários tipos de eventos, facilitando a sua integração e utilização;
- Permitem uma significativa redução das linhas de código a desenvolver através da utilização de técnicas de ligação entre os controlos visuais e os dados que deverão apresentar ou processar (*Data Binding*);
- Disponibiliza um modelo *Event-Based* que é muito familiar aos programadores de aplicações para *desktop*, o que facilita, em grande medida, a sua entrada no desenvolvimento *web*;
- Possibilita o desenvolvimento de controlos adicionais por outras empresas, o que aumenta a quantidade de controlos disponíveis para utilização assim como as funcionalidades dos já existentes;

[21] http://www.asp.net/web-forms, Fevereiro 2013
[22] http://msdn.microsoft.com/en-us/library/ms973868.aspx, Fevereiro 2013

- Implementa diversos padrões de desenvolvimento, destacando-se um padrão para a manutenção de estados utilizando o protocolo HTTP (o que favorece o desenvolvimento de aplicações *web* vocacionadas para o negócio), o *Page Controller* e o *View State*;
- Adequada para pequenas equipas de desenvolvimento e para pequenos projetos que possam tirar proveito dos conceitos RAD;
- Em projetos de reduzida dimensão, uma aplicação desenvolvida utilizando os *Web Forms*, tende a ser mais simples e com menos linhas de código do que aplicações desenvolvidas noutras plataformas ou utilizando outras metodologias.

O ASP.NET *Web Forms*, apesar de apresentar diversas vantagens também tem alguns aspetos menos positivos que deverão ser levados em consideração. Os principais aspetos negativos a apontar estão resumidos na lista seguinte [Esposito10;Esposito09]:

- O mecanismo de manutenção do estado entre pedidos ao servidor (chamado de *View State*) pode representar um aumento considerável do tráfego entre o cliente e o servidor e com tendência para aumentar à medida que se vai utilizando a aplicação. Em ligações rápidas (como as que existem, atualmente) isto pode não ser problemático mas em ligações mais lentas poderá ter um impacto bastante negativo na fluidez da aplicação;
- A ligação dos eventos que ocorrem no lado do cliente ao código que os deve processar e que está localizado no servidor é um processo complicado e que está muito sujeito a erros. De uma forma genérica, muitas das facilidades que os *Web Forms* possibilitam são obtidas à custa de técnicas complexas e que aumentam, substancialmente, a possibilidade de ocorrência de erros e dificulta o trabalho dos programadores quando estes pretendem analisar melhor o código;
- Ao contrário de outros *frameworks*, o ASP.NET *Web Forms* não favorece a estruturação da aplicação em diversas camadas. Apesar do *framework* permitir uma clara separação entre o código HTML e o restante código da aplicação, neste último nível a estruturação em camadas não é muito facilitada;
- O programador tem um controlo muito reduzido sobre o HTML gerado pelo servidor. Pode acontecer que o HTML gerado não seja compatível com os *standards* da *web* ou que não utilize o CSS da melhor forma (embora a versão 4 do ASP.NET tenha reduzido significativamente esta situação) e, nestes casos, a possibilidade de intervenção do programador é muito reduzida. O HTML gerado, para além de poder não ser o mais eficiente, também pode complicar a utilização de código *Javascript*. Todas estas situações relacionadas com o HTML gerado podem ser difíceis de resolver ou contornar em virtude do reduzido controlo que os *Web Forms* disponibilizam ao programador;
- Com o objetivo de simplificar o desenvolvimento do *software*, o ASP.NET *Web Forms* afasta dos programadores muitas questões técnicas relacionadas com o desenvolvimento *web*. Se, por um lado, esta abstração é vantajosa, poderá tornar-se num entrave muito grande caso se pretenda alterar o comportamento de determinados componentes ou ter um maior controlo sobre o comportamento da aplicação. Para ultrapassar estas situações, por vezes é necessário implementar soluções complexas ou que permitam contornar/alterar as regras do *framework*,

com os inconvenientes que daí poderão advir (como, por exemplo, dificuldades na manutenção do código no futuro ou as soluções deixarem de funcionar após uma atualização do *framework*). Os elevados níveis de abstração dos *Web Forms* também podem ser considerados como um entrave para o conhecimento mais aprofundado da plataforma ASP.NET;

- A arquitetura dos *Web Forms* não é muito adequada para a definição de testes unitários e de integração. Quando surgiram as primeiras versões do ASP.NET *Web Forms*, os conceitos de testes unitários e de integração não estavam muito entrosados e, por isso, não foram considerados na definição da arquitetura.

4.1.2. ASP.NET MVC

O ASP.NET MVC é um *framework* para desenvolvimento de aplicações *web* que segue os princípios do padrão *Model–View–Controller* [Galloway12]. Trata-se de um *framework* adequado para projetos em que se pretende separar, muito claramente, a camada das regras do negócio da camada de apresentação e que se baseia em princípios de *Test Driven Development*, *Inversion of Control* e *Dependency Injection*[23]. A primeira versão do ASP.NET MVC surgiu em 2009 e, desde essa data, têm sido lançadas diversas atualizações, sendo a mais recente, a versão 4 que se baseia nas versões 4.0 e 4.5 da plataforma .NET.

O ASP.NET MVC pode ser considerado como sendo uma alternativa ao ASP.NET *Web Forms* uma vez que apresenta uma abordagem totalmente diferente na forma como se constroem as aplicações *web*. Os programadores, em vez de se concentrarem na implementação de páginas ASPX, eventos ou controlos, têm de se focar em controladores, ações e vistas. Embora estejamos perante uma abordagem totalmente diferente, a plataforma ASP.NET serve de base aos dois *frameworks*, sendo possível utilizar as duas metodologias na mesma aplicação.

Os princípios definidos pelo padrão MVC têm-se revelado, ao longo do tempo, bastante adequados no desenvolvimento de aplicações, quer se tratem de aplicações para *desktop* ou aplicações *web*, e, por este motivo, serviram de base ao *framework* em análise. Separando a camada da apresentação das restantes camadas permite-se a implementação de alterações nas diversas camadas de forma isolada e facilita-se a participação nos projetos de técnicos com valências distintas. Desta forma, um *Web Designer* pode implementar mais facilmente a parte visual da aplicação sem interferir com o trabalho do programador que desenvolve as regras do negócio e a camada de acesso aos dados. Por outro lado, separando a camada visual das restantes camadas facilita-se uma eventual mudança tecnológica nesta área, como, por exemplo, a introdução de código HTML5 em páginas.

O *framework* ASP.NET MVC apresenta um conjunto de caraterísticas e vantagens que são apresentadas a seguir [Esposito10;Esposito09;Palermo12;Freeman11]:

[23] http://msdn.microsoft.com/en-us/library/dd381412%28v=vs.108%29.aspx, Fevereiro 2013

- **Arquitetura MVC –** Esta arquitetura possui algumas caraterísticas que se adequam ao desenvolvimento *web*. A interação dos utilizadores com uma aplicação MVC segue um fluxo natural: o utilizador efetua uma ação e, como resposta, a aplicação altera a camada de dados, atualizando a camada de apresentação para refletir essa alteração. Este ciclo repete-se para cada ação do utilizador e adequa-se perfeitamente às aplicações *web*. Normalmente, uma aplicação *web* utiliza diversas tecnologias, as quais serão mais bem organizadas e utilizadas se estiverem agrupadas numa arquitetura em camadas.
- **Extensibilidade –** O *framework* ASP.NET MVC é constituído por componentes independentes, o que facilita a sua alteração ou substituição por outros mais adequados. Isto só é possível porque os componentes comunicam entre si utilizando interfaces padronizados baseados na plataforma .NET. Para cada um dos componentes será possível realizar uma das seguintes operações: 1) utilizar os componentes tal como são disponibilizados pelo *framework*; 2) expandir a funcionalidade através da criação de subclasses (seguindo os princípios da programação orientada aos objetos); 3) substituir totalmente um componente por outro diferente desenvolvido especificamente para o efeito.
- **Maior controlo do código HTML da aplicação –** Com este *framework* consegue-se um código HTML mais simples e conciso do que o gerado pelo *framework* ASP.NET *Web Forms*. Além disso, o ASP.NET MVC permite um maior controlo sobre o HTML. Apesar desta caraterística, o ASP.NET MVC facilita a utilização de elementos visuais (que, regra geral, são complexos) das diversas bibliotecas de componentes visuais que atualmente existem. O ASP.NET MVC também permite um controlo muito grande e efetivo dos pedidos e respostas trocados entre o cliente e o servidor.
- **Testes unitários e manutenção –** A utilização deste *framework* favorece a implementação de testes unitários e facilita a manutenção das aplicações numa fase posterior. Isto é alcançado através do isolamento das diversas áreas da aplicação em componentes independentes. Os testes unitários poderão ser definidos e realizados utilizando ferramentas *open source* (tais como o *NUnit* ou o *xUnit*) ou utilizando o *framework* de testes da *Microsoft* (*MSTest*).
- **Utilização de URL simplificadas –** O ASP.NET MVC facilita a utilização de URL simplificadas através de um conjunto de classes definidas na plataforma .NET. Estas classes permitem controlar as URL da aplicação e criar um padrão de URL que seja simples e coerente ao longo da aplicação. As URL simplificadas apresentam várias vantagens que vão desde benefícios nos motores de busca até à possibilidade de se ocultarem detalhes técnicos das aplicações.
- **Plataforma .NET –** Trata-se de uma metodologia desenvolvida sobre a plataforma .NET e que utiliza o que de melhor ela tem. Uma das vantagens mais evidentes desta caraterística é a possibilidade de se poder escrever o código em qualquer uma das linguagens de programação suportadas pela plataforma .NET. Isto permite ao programador aceder a quase todos os recursos e funcionalidades disponibilizados pela plataforma .NET. *Master Pages*, *Forms Authentication*, *Membership* e *Internationalization* são algumas das funcionalidades disponibilizadas pelo ASP.NET e que permitem uma redução significativa da quantidade de código necessária para desenvolver e manter uma aplicação (por se tratarem de funcionalidades da plataforma .NET, a maior parte delas também estão

disponíveis no ASP.NET *Web Forms* pelo que esta vantagem também é aplicável ao referido *framework*);

- **Ligação com ASP.NET *Web Forms* -** O *framework* ASP.NET MVC também permite a utilização dos componentes *Web Forms*, possibilitando o aproveitamento dos inúmeros controlos existentes. No entanto, isto apenas é possível se os componentes não utilizarem técnicas que sejam específicas dos *Web Forms*, tais como, por exemplo, o *View State*. Ao estar baseada na plataforma .NET, todas as evoluções que esta tem sofrido ao longo do tempo passaram a estar disponíveis para utilização no ASP.NET MVC. São exemplo destas evoluções, mas não se limitando a esta lista, as expressões *Lambda*, LINQ ou os tipos de dados anónimos e dinâmicos.
- ***Open Source* –** Uma das características que mais diferenciam o ASP.NET MVC no seio da plataforma .NET é o facto de este ser um *framework open source*. Os programadores são livres de fazerem o *download* do código-fonte e de o analisarem e modificarem conforme as suas necessidades. Esta caraterística é de extrema importância uma vez que pode facilitar muito o trabalho a quem, por exemplo, pretender utilizar o *framework* a um nível mais aprofundado. No entanto, as eventuais alterações que venham a ser introduzidas não podem ser enviadas para o repositório oficial da *Microsoft*. Isto significa que sempre que seja lançada uma nova versão ou atualização do ASP.NET MVC, as alterações efetuadas por terceiros terão que ser aplicadas, novamente, na versão disponibilizada. O ASP.NET MVC é distribuído sobre a *Apache License 2.0*.

Apesar da quantidade considerável de vantagens que apresenta, relativamente a outros *frameworks* da plataforma .NET, o ASP.NET MVC também possui alguns aspetos negativos que se apresentam de seguida:

- Atendendo a que se trata de um *framework* que possibilita um grande controlo dos aspetos relacionados com o desenvolvimento *web*, o processo de aprendizagem pode ser mais demorado e exigir mais dos programadores. Por outro lado, por se tratar de uma abordagem de mais baixo nível ao desenvolvimento de *software*, a construção de uma aplicação *web* tenderá a ser mais demorada. No entanto, se atendermos a que neste processo de construção são respeitadas as boas práticas de programação, o tempo investido nesta fase poderá ser amplamente recuperado na fase da manutenção das aplicações [Esposito09]. Normalmente, quando um *framework* facilita muito a tarefa do desenvolvimento de uma aplicação, isso é obtido à custa do tempo que será gasto na manutenção;
- Comparativamente ao *framework Web Forms*, o ASP.NET MVC é uma técnica mais recente e, por este motivo, mais propensa ao surgimento de erros e à inexistência de determinadas funcionalidades. Isto implica uma maior cadência no lançamento de novas versões, o que pode trazer alguns inconvenientes [Esposito09].

4.1.3. SILVERLIGHT

O *Microsoft Silverlight* é um *framework* de desenvolvimento de aplicações RIA baseado na plataforma .NET. Geralmente, uma aplicação *Silverlight* correrá dentro de um *browser* mas também é possível desenvolver aplicações que funcionem fora dos *browsers* como se se tratassem de aplicações para *desktop*. Este *framework* utiliza código XAML para desenho dos aspetos visuais e C# ou *Visual Basic.NET* para o desenvolvimento da lógica aplicacional [Lair12]. A primeira versão do *Silverlight* surgiu em 2007, tendo sido lançadas mais 4 versões ao longo dos últimos anos. Atualmente, a versão estável do *Silverlight* é a 5.0, tendo surgido em finais de 2011 [Brown12]. As principais caraterísticas do *Silverlight* estão resumidas na lista seguinte [Lair12]:

- Trata-se de uma tecnologia *multi-browser*, que funciona na maioria dos *browsers* atualmente existentes (mediante a instalação de um *runtime*), e que está disponível para os sistemas operativos *Windows* e *Apple Mac OSX*. Existe, também, uma implementação para *Linux* (chamada *Moonlight*) que não se encontra, atualmente, em desenvolvimento, em virtude do fraco nível de aceitação que obteve;
- Para além de funcionar dentro dos *browsers* e no *desktop*, também permite a criação de aplicações para o sistema operativo *Windows Phone*;
- Muito vocacionada para aplicações multimédia, permitindo, facilmente, o processamento de vídeo, áudio e imagens;
- Permite uma clara separação entre a parte visual e a lógica da aplicação, disponibilizando ferramentas de desenvolvimento específicas para as duas áreas.

Para se poder desenvolver aplicações *Silverlight* será conveniente utilizar as ferramentas disponibilizadas pela *Microsoft*. Os programadores terão à sua disposição as diversas versões do *Microsoft Visual Studio* (ao qual deve ser adicionado o *Silverlight Tools for Visual Studio* para facilitar o tratamento dos projetos) assim como o *Microsoft Expression Blend* para o desenho dos aspetos visuais. Embora seja disponibilizada uma ferramenta específica para o tratamento da UI, este pode ser feito, também, dentro do *Visual Studio*, embora, neste caso, será mais complicado tirar partido de todas as capacidades do *framework*, no que ao aspeto gráfico diz respeito [MacDonald12].

Embora seja necessário um runtime para que uma aplicação *Silverlight* funcione num determinado equipamento, a *Microsoft* desenvolveu grandes esforços com o objetivo de tornar o processo de *download* e instalação do *runtime* muito simples e rápido. Se, ao iniciar uma aplicação, o computador não tiver o *runtime* instalado, serão iniciados os procedimentos para o *download* e instalação do mesmo[24]. O instalador atual do *runtime* tem um tamanho de 6 MB na versão 32 bits e cerca de 12 MB para a versão 64 bits.

Comparando o *Silverlight* com o ASP.NET (*Web Forms* e MVC) poderemos enunciar os seguintes aspetos [Anderson12]:

[24] http://msdn.microsoft.com/pt-pt/library/bb404700, Fevereiro 2013

- O ASP.NET requer que toda a plataforma .NET esteja instalada na máquina onde corre o servidor *web* (normalmente, o *Microsoft Internet Information Services*), enquanto que para se correr uma aplicação *Silverlight* basta que esteja instalado o respetivo *runtime* na máquina do cliente. Contudo, esta característica também se pode traduzir numa vantagem para o ASP.NET, uma vez que terá ao dispor todas as funcionalidades da plataforma .NET;
- Uma aplicação ASP.NET será executada em praticamente todos os dispositivos que possuam uma ligação à Internet, não requerendo qualquer instalação adicional de qualquer tipo de *software*, para além de um *browser*;
- A tecnologia *Silverlight* é mais recente não tendo, por este motivo, o mesmo grau de maturidade que o ASP.NET. Esta existe desde que surgiu a plataforma .NET;
- Uma aplicação ASP.NET requer uma maior intervenção do programador no código que é executado no *browser*, exigindo mais conhecimentos de *Javascript* e HTML. Isto pode dar origem a que uma aplicação tenha comportamentos e aspetos diferentes de *browser* para browser, algo que, com uma aplicação *Silverlight* será mais difícil de acontecer;
- A interação das aplicações com o sistema operativo anfitrião não é possível com aplicações ASP.NET. Embora de forma limitada, esta funcionalidade está disponível em aplicações *Silverlight*;
- O desenvolvimento de uma aplicação RIA é mais facilitado utilizando o *Silverlight*. Tecnicamente, é possível desenvolver uma aplicação semelhante usando o ASP.NET embora seja necessário a utilização de bibliotecas *Javascript* e técnicas AJAX;

Desde o seu surgimento, o *Silverlight* tem sido alvo de muitas críticas e por diversas vezes tem sido anunciada a sua descontinuidade. Apesar destas críticas e rumores, a *Microsoft* continua a lançar novas atualizações, tendo surgido 5 novas versões no curto espaço de 4 anos. Com o surgimento e a adoção cada vez maior do HTML5 como padrão *web*, a posição do *Silverlight* volta a ser posta em causa, uma vez que, atendendo às suas caraterísticas, poderemos considerar que o *Silverlight* e o HTML5 são tecnologias concorrentes e que ocupam o mesmo espaço na área do desenvolvimento de aplicações *web*.

Uma das grandes vantagens do HTML5 sobre o *Silverlight* é o facto de se tratar de uma tecnologia que está e estará disponível numa quantidade muito grande de dispositivos e plataformas. Teoricamente, o HTML5 estará disponível e correrá em qualquer dispositivo que o possa interpretar, independentemente do sistema operativo em que esteja a correr e das caraterísticas do referido dispositivo. Neste campo, o HTML5 leva uma clara vantagem sobre o *Silverlight*, embora se possa afirmar que quanto maior for a abrangência de uma tecnologia, menos proveito ela poderá retirar das especificidades de cada um dos dispositivos e sistemas operativos em que corra. Uma outra caraterística em que o HTML5 leva vantagem sobre o *Silverlight* é a ausência da necessidade de *plugin* para ser executado, o que poderá ser vantajoso em situações em que a capacidade de processamento seja limitada. Pelo contrário, isto também pode ser visto como uma desvantagem, atendendo a que um mesmo código HTML poderá ser apresentado de forma diferente pelos *browsers* e também não é garantido que todos os *browsers* existentes sejam totalmente compatíveis com o HTML5. Para um programador, será sempre mais difícil e

demorado trabalhar com o HTML5 do que com o *Silverlight*. Ao nível da comunidade de desenvolvimento, o HTML5, por se tratar de um standard não proprietário, possui uma base muito maior do que o *Silverlight* e com tendência para crescer, à medida que o HTML5 se vai implementado. Apesar destes números, normalmente, o *Silverlight* evolui de forma mais rápida do que o HTML5. Um aspeto que pode ser de grande desvantagem para o HTML5 é o facto de se tratar de uma tecnologia recente e imatura, o que, geralmente, se traduz numa menor produtividade dos programadores e que poderá ser agravado pela reduzida oferta de ferramentas de desenvolvimento. A quantidade de código-fonte disponível também é menor nestas fases iniciais de implementação de uma nova tecnologia.

De uma forma geral, o HTML5 deve ser considerado em projetos que pretendem ser utilizados no maior número possível de plataformas e dispositivos. Pela sua natureza mais restrita e proprietária, o *Silverlight* apresenta-se como a solução mais adequada para situações em que se pretenda tirar maior proveito das especificidades das plataformas onde possa ser executado.

4.2. *JAVA*

As origens da plataforma *Java* remontam ao início da década de 1990 quando um grupo de programadores da *Sun*, do qual fazia parte James Gosling, se reuniu com o objetivo de desenvolver aplicações para dispositivos de eletrónica de consumo[25]. Numa fase inicial, o projeto estava a ser desenvolvido em C++ mas acabaram por verificar que esta não era a linguagem de programação mais adequada e, por este motivo, optaram por iniciar o desenvolvimento de uma linguagem própria. Em setembro de 1992 a equipa apresentou um protótipo de um dispositivo semelhante aos atuais PDA, totalmente desenvolvido pela equipa e num curto espaço de tempo. Em cerca de 18 meses o grupo de trabalho tinha desenvolvido um sistema operativo, uma linguagem de programação e uma API. Apesar do aparente sucesso e das inúmeras possibilidades que o dispositivo permitia, o mercado ainda não estava preparado para dar este passo, o que levou a que o projeto fosse abandonado e que os seus elementos fossem incorporados noutras áreas da empresa. No início de 1993, surgiu a possibilidade da nova plataforma ser utilizada numa *set-top-box* para um sistema de televisão interativa mas, apesar dos esforços da *Sun* e dos vários contactos estabelecidos junto das grandes empresas do ramo, não foi possível avançar com o projeto. Em 1994, com o advento da Internet, surgiu uma nova possibilidade de afirmação da plataforma que, finalmente, encontrou uma área em franco crescimento e que se adequava, perfeitamente, às suas caraterísticas. Muitos dos princípios que serviram de base para a criação do *Java* - segurança, independência da plataforma e robustez - eram requisitos e caraterísticas necessárias para a *World Wide Web*. Em 1995, na *Sun World Conference*, a linguagem *Java* foi anunciada oficialmente [Martins09].

[25] http://ei.cs.vt.edu/book/chap1/java_hist.html, Março 2013

Uma das principais caraterísticas da tecnologia *Java* é a possibilidade de se escrever código uma vez e este ser executado em qualquer sistema. Para que isto seja possível existe uma camada adicional de *software* entre o código *Java* e o *hardware*, que permite "ocultar" as especificidades de cada sistema. Esta camada intermédia é a chamada *Java Virtual Machine* (JVM). Assim, quando se compila código *Java*, não se está a compilar diretamente para o código nativo das máquinas mas sim para um código intermédio que será, posteriormente, interpretado/executado pela JVM específica do sistema onde correr (o que significa que terão que existir JVM específicas para cada sistema). O código intermédio é chamado de *byte-code* e é igual, independentemente do dispositivo onde será executado. O que varia em função do dispositivo será sempre a JVM [Knudsen05].

Ao contrário de outras tecnologias, o termo "*Java*" é utilizado para identificar a plataforma de base assim como a linguagem de programação. Apesar de partilharem o mesmo nome tratam-se de dois conceitos totalmente distintos, embora estejam relacionados. A linguagem de programação *Java* é uma linguagem orientada a objetos com uma sintaxe semelhante ao C++, que tenta colmatar as principais falhas e limitações das linguagens C/C++ e, ao mesmo tempo, tornar os programadores mais produtivos. Isto é conseguido através da simplificação de alguns conceitos (por exemplo, não é permitido o conceito de herança múltipla) e de um sistema muito eficiente de gestão de memória (chamado de *Garbage Collector*). A plataforma *Java*, por outro lado, abrange um conceito muito mais vasto e pode ser definida como sendo o conjunto de todas as classes que estão à disposição do programador e nas quais se baseia todo o *software* escrito na linguagem *Java*. Estas classes são a base de todo o *software* desenvolvido em *Java*, e por serem em número muito elevado (a versão 7 engloba mais de 4000 classes) surgem agrupadas em função da sua natureza, criando o conceito de *package*. Existem *packages* para tratamento de listas, entrada/saída de dados, processamento de *strings*, operações matemáticas, entre outras. Atualmente existem três tipos de plataformas *Java* [Pordel08]:

- *Java Standard Edition* (*Java* SE)
- *Java Enterprise Edition* (*Java* EE)
- *Java Micro Edition* (*Java* ME)

A plataforma *Java* SE está vocacionada para o desenvolvimento de aplicações *desktop* que podem funcionar em PC ou servidores. Também tem como objetivo disponibilizar as funcionalidades básicas de qualquer tipo de aplicação (p. ex. I/O, manipulação de strings, manipulação de listas e acesso a base de dados) assim como as funcionalidades específicas para o desenvolvimento de aplicações *desktop* (essencialmente funcionalidades relacionados com a interface do utilizador). Por isso, podemos considerar a *Java* SE como sendo a base de todos os restantes tipos de plataformas, uma vez que é com base nela que foram criadas as outras. É composta pela JVM, pelas classes que constituem a *Java* SE *API* e por vários utilitários de desenvolvimento, os quais constituem o *Java Standard Development Kit* (JDK). Esta é a plataforma mais comum que se pode encontrar nos computadores.

Analisando a figura 4.3, verifica-se que a base das plataformas *Java* é o *Java Runtime Environment* (JRE), o qual já contém a JVM. Este elemento é o mínimo indispensável para

se executar uma aplicação *Java*. O JDK surge como uma extensão ao JRE e apenas será necessário se pretendermos desenvolver aplicações na linguagem de programação *Java*. O JDK é constituído, principalmente, por um conjunto de utilitários necessários ao desenvolvimento, como, por exemplo, um compilador (*javac*), um depurador (*jdb*), um gerador de documentação (*javadoc*) ou um gerador de arquivos *jar*.

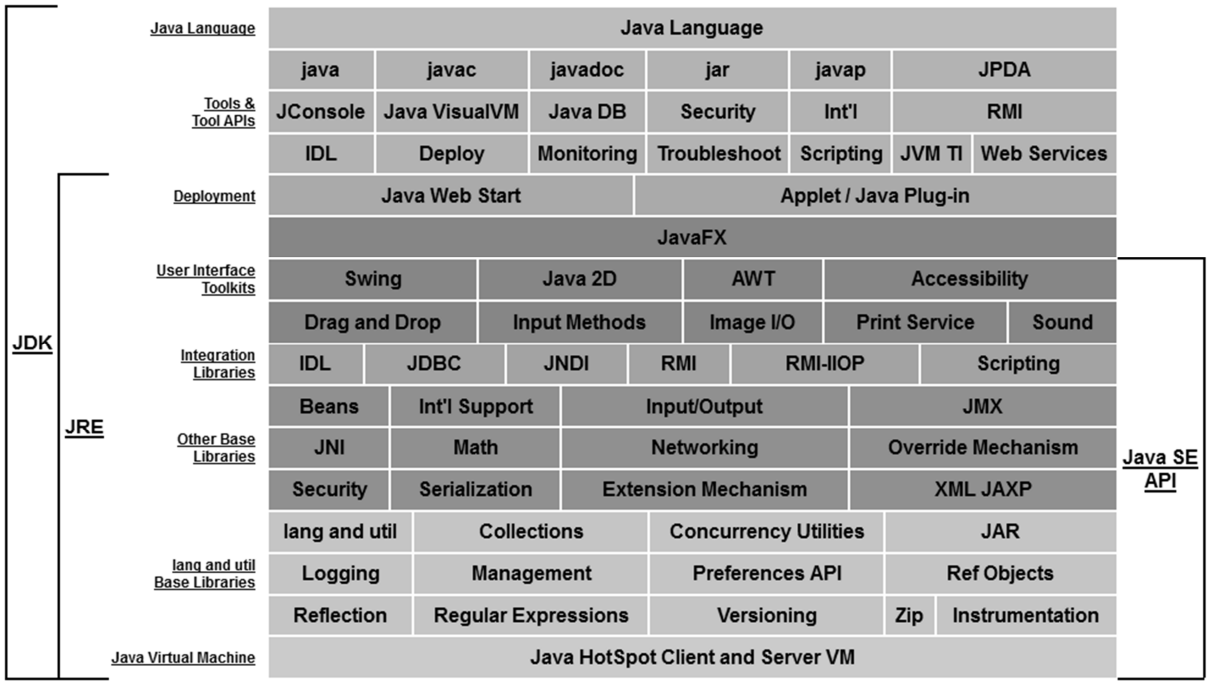


Figura 4.3 – Plataforma *Java* SE[26]

A plataforma *Java* EE destina-se ao desenvolvimento de aplicações distribuídas através da definição de diversas camadas aplicacionais. Uma das principais diferenças da plataforma *Java* EE para a *Java* SE é que a primeira disponibiliza serviços que, normalmente, são utilizados em aplicações empresariais. Numa aplicação *Java* EE, a camada de apresentação, que pode ser uma página *web*, um *applet* ou uma aplicação *Java* SE, está separada das restantes camadas (em geral, as camadas das regras de negócio e de acesso aos dados) [JavaEETut12]. Esta separação é alcançada através da utilização de várias tecnologias, destacando-se os *JavaServer Pages* (JSP), *Java Servlets*, *Facelets* ou *Enterprise Java Beans*. A plataforma *Java* EE também disponibiliza várias API orientadas para a interação entre as camadas aplicacionais.

A plataforma *Java ME* é utilizada no desenvolvimento de aplicações para dispositivos móveis ou com pouca capacidade de processamento, disponibilizando classes específicas para este tipo de dispositivos. Dentro da *Java* ME pode-se identificar, ainda, o *Java Card Platform* (JCP), vocacionado para o desenvolvimento de *software* para cartões com *chip*.

[26] http://docs.oracle.com/javase/7/docs, Abril 2013

A tecnologia *Java* apresenta vários benefícios, destacando-se os seguintes:

- ***Write Once, Run Anywhere***. Um dos princípios base da plataforma que permite que se escreva o código uma só vez e que este possa ser executado em vários sistemas distintos sem ser necessária qualquer alteração. Embora esta caraterística possa ser vista como uma vantagem, apresenta, também, algumas limitações, principalmente no desenvolvimento de aplicações com uma interface para os utilizadores, uma vez que uma aplicação *desktop* escrita em *Java* nunca tem um aspeto 100% nativo;
- **Ferramentas de desenvolvimento**. Existem diversas ferramentas de desenvolvimento disponíveis para *Java*, sendo que a maioria delas são gratuitas e de utilização livre de quaisquer encargos (a título de exemplo, referimos os IDE *Netbeans* e *Eclipse*, dois dos mais utilizados na plataforma). Para além destas ferramentas de desenvolvimento existem milhares de bibliotecas disponibilizadas por terceiros que facilitam o desenvolvimento para a plataforma *Java* (como exemplo, podemos referir as bibliotecas *JFreechart*, para o desenvolvimento de gráficos e a *JasperReports* utilizada na criação de listagens);
- **Visão da rede como um todo**. A plataforma *Java* foi criada tendo em mente que os recursos podem estar disponíveis através de uma rede e em qualquer local desta e que, por este motivo, o programador deverá ter acesso a eles de forma rápida e simples;
- **Dinamismo e flexibilidade**. Todo o código fonte de uma aplicação *Java* está organizado em classes, sendo cada uma delas colocada no seu respetivo ficheiro. Estas classes apenas são processadas pela JVM quando são, efetivamente, necessárias. Esta caraterística, para além de diminuir o tempo de arranque das aplicações também permite que uma aplicação aumente as suas funcionalidades de forma dinâmica, através da adição de classes à medida que vão sendo necessárias;
- **Internacionalização**. Ao contrário de outras plataformas, em que as questões da internacionalização foram sendo adicionadas ao longo do tempo, na plataforma *Java* as mesmas questões foram consideradas desde o início, o que possibilita um maior desempenho e integração em todas as bibliotecas;
- **Desempenho**. Uma aplicação *Java*, embora não sendo código nativo, apresenta um desempenho bastante elevado, sendo mais rápido que a maioria das linguagens interpretadas que atualmente existem. Isto é possível graças à compilação intermédia para *byte-code*. No entanto, apesar deste elevado desempenho, regra geral, uma aplicação *Java* continuará a ser mais lenta que uma aplicação desenvolvida em C/C++. Ao longo do tempo o desempenho da plataforma *Java* tem sido consideravelmente incrementado através da introdução de diversas técnicas e otimizações. Uma das técnicas que causou maior impacto foi a compilação *Just In Time* que permite converter o *byte-code* em código nativo, mesmo antes de ele ser executado.

Quanto às desvantagens, apresentam-se as seguintes:

- **Desatualização e incompatibilidades das JVM.** É frequente encontrarem-se versões muito antigas da JVM nos computadores dos utilizadores. Isto pode ser um

problema sério tanto ao nível do desempenho como da segurança. Apesar de, hoje em dia, o *software* ser bastante eficiente na gestão das versões e das atualizações, há sempre um risco elevado em estarmos a utilizar uma JVM inadequada. Por outro lado, atendendo a que as máquinas virtuais podem ser desenvolvidas por qualquer empresa, também existe a possibilidade de se estar a utilizar uma JVM que não seja a mais adequada;

- **Desempenho.** Embora este aspeto tenha sido apresentado como uma vantagem, principalmente em relação a outras linguagens interpretadas ou a outras plataformas, na realidade, existem situações em que o desempenho da plataforma poderá ser insuficiente, sendo preferível optar por linguagens nativas. Por outro lado, atendendo à plataforma necessária para executar uma aplicação *Java*, o consumo de memória e de recursos será sempre superior em relação a aplicações compiladas para código nativo;
- **Evolução lenta da linguagem e da plataforma.** Embora este aspeto também possa ser visto como uma vantagem, atendendo a que ao evoluir mais lentamente há mais tempo para ponderar as opções e efetuar mais testes, a realidade é que, comparativamente a outras plataformas (por exemplo, com a plataforma .NET) a evolução do *Java* é muito mais lenta, o que se pode traduzir numa maior desatualização e num menor acompanhamento das novas tendências e tecnologias;
- **Aspeto das interfaces de utilizador.** O aspeto das aplicações *Java* desenvolvidas utilizando a biblioteca *Swing* apresenta diferenças relativamente às aplicações nativas. Estas diferenças advêm do facto das aplicações poderem ser executadas em qualquer sistema. No entanto, existem bibliotecas alternativas (por exemplo, a *Standard Widget Toolkit*) que permitem obter um aspeto mais aproximado, ou mesmo igual, embora estas bibliotecas não façam parte da plataforma base;
- **Proliferação de bibliotecas e *frameworks*.** A existência de um grande número de *frameworks* e de bibliotecas desenvolvidas por terceiros pode fragmentar a comunidade de programadores e aumenta o risco de se optar por uma metodologia ou biblioteca que possam ser descontinuadas.

4.2.1. JAVA ENTERPRISE EDITION

A plataforma *Java* EE é constituída por um conjunto de especificações que permitem o desenvolvimento de aplicações empresariais, podendo ser vista como uma extensão do *Java* SE. O seu principal objetivo é fornecer ao programador diversas API que facilitem o desenvolvimento de aplicações robustas, seguras, escaláveis e rápidas, ocultando os aspetos mais complexos e conseguindo tudo isto no mais curto espaço de tempo possível [JavaEETut12]. O desenvolvimento das especificações da plataforma é feito pelo *Java Comunity Process* (JCP). O JCP é uma organização criada em 1998 e é composto por representantes de empresas, universidades, organizações das mais diversas áreas e pessoas singulares. Sempre que surge a necessidade de se criar uma nova API, gera-se uma *Java Specification Request* (JSR) a qual será tratada pelos membros do JCP. Assim que a JSR esteja totalmente definida, será aprovada por um comité executivo e, após este passo,

passará a fazer parte do *Java* EE. Assim, a plataforma *Java* EE pode ser vista como um conjunto de JSR que, posteriormente, serão implementadas por terceiros.

A plataforma *Java* EE surgiu, pela primeira vez, em 1999. Foi lançada com o nome de J2EE 1.2 e continha 10 especificações. Em 2001 surgiu a versão 1.3 da plataforma, a qual continha 13 especificações. A versão seguinte, J2EE 1.4, lançada em 2003 apresentava 20 JSR. A partir deste momento, a nomenclatura da plataforma sofreu alterações, passando a identificar-se como *Java* EE *X*, em que o "*X*" representa a versão. Assim, em 2006 surgiu a plataforma *Java* EE *5* contendo 23 especificações. A versão atual da plataforma, *Java* EE 6, foi lançada no final de 2009 e especificava 28 JSR. Uma das diferenças desta última versão relativamente às versões anteriores é que contempla o conceito de *pruning* que consiste na marcação de algumas JSR como estando obsoletas e com possibilidade de serem removidas em futuras versões da plataforma. A tabela 4.1 apresenta a evolução da plataforma ao longo do tempo, bem como uma previsão para a próxima versão do *Java* EE [Goncalves10]:

Versão	**Data**	**N.º Especificações**	**Principais Funcionalidades**
J2EE 1.2	Dez./1999	10	*Servlets,* JSP, EJB, JMS, RMI/IIOP
J2EE 1.3	Set./2001	13	EJB CMP, JCA, JAXP, JAAS*, Java Mail Api*
J2EE 1.4	Nov./2003	20	*Web Services, Management Deployment*
Java EE 5	Mai./2006	23	*Anotations,* JPA, EJB 3.0, JAX-WS, JAXB, JSF
Java EE 6	Dez./2009	28	*Pruning,* JAX-RS*, Servlet 3.0,* EJB 3.1 *Lite, Context & Dependency Injection*
Java EE 7	2013 (Prev.)	33	JMS 2.0*, WebSocket,* JSON, JAX-RS 2.0*, JCache*

Tabela 4.1 – Evolução da plataforma *Java* EE

A plataforma *Java* EE baseia-se num modelo distribuído para o desenvolvimento de aplicações empresariais. Neste modelo, a lógica das aplicações é dividida em função da sua natureza e colocada em diferentes camadas. Estas camadas podem residir em máquinas diferentes ou estarem todas na mesma máquina. Geralmente, identificam-se as seguintes camadas numa aplicação *Java* EE:

- Camada de apresentação (*Client Tier*) que, normalmente, reside e é executada na máquina do cliente;
- Camada *Web* (*Web Tier*) que corre num servidor *Java* EE;
- Camada de Negócios (*Business Tier*) que, tal como a camada anterior, também é executada num servidor *Java* EE;
- Camada da base de dados (*Enterprise Information System Tier*) que pode estar situada na mesma máquina que o servidor *Java* EE ou numa máquina diferente.

Apesar de, em termos teóricos, se identificarem 4 camadas, na prática consideram-se que as aplicações são constituídas por 3 camadas, uma vez que, regra geral, são distribuídas por 3 máquinas diferentes: Cliente, Servidor *Java* EE e Base de Dados. A figura 4.4 apresenta a estrutura tipo de uma aplicação *Java* EE.

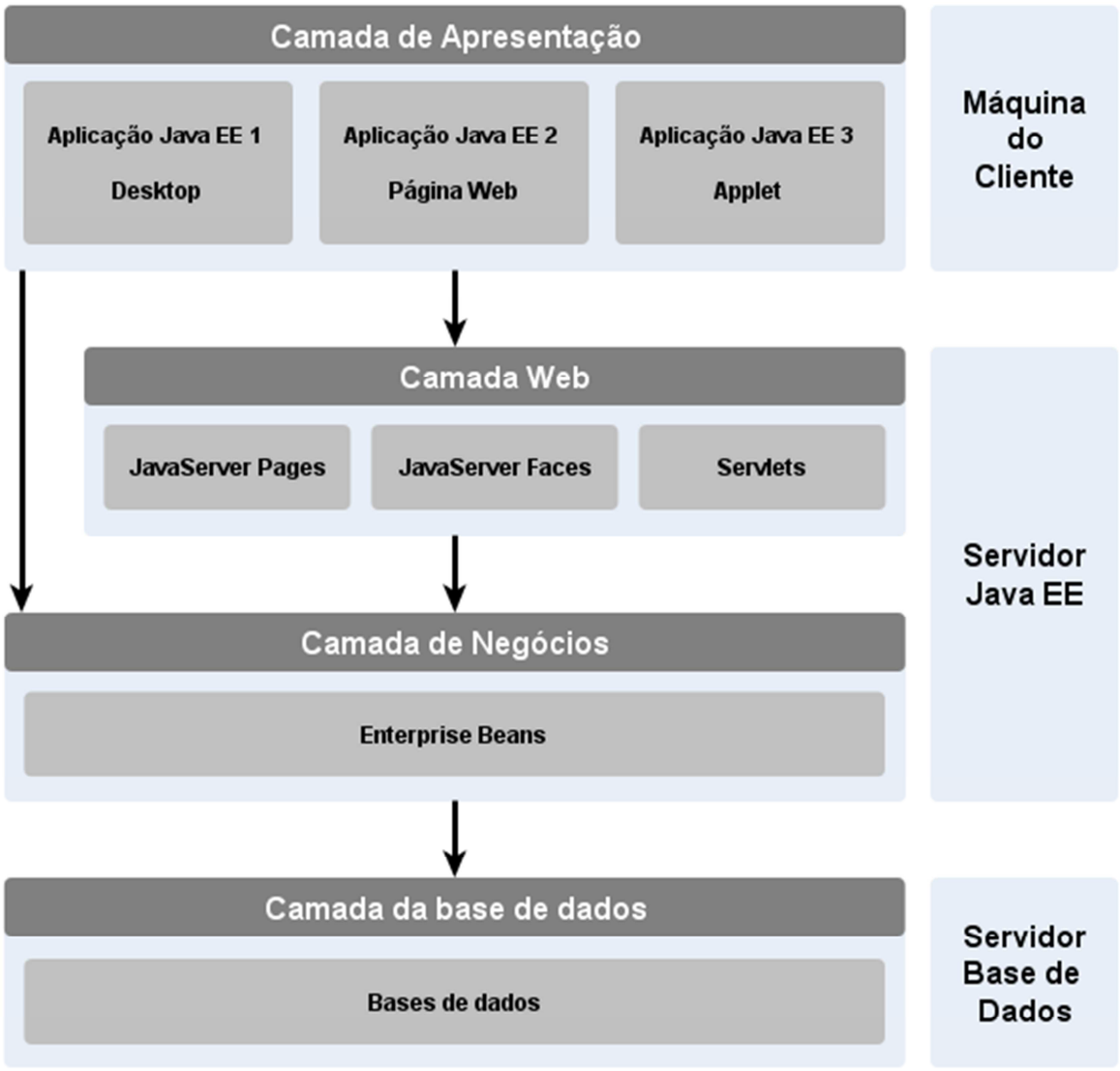


Figura 4.4 – Aplicação *Java* EE[27]

Na figura 4.4 é possível identificar os diversos componentes que fazem parte de uma aplicação *Java* EE. Um componente pode ser definido como uma unidade funcional e independente de *software* que, em conjunto com outros componentes, dão origem a uma aplicação *Java* EE. Podemos identificar os seguintes tipos de componentes [Goncalves10]:

- ***Java Applets*:** Pequenas aplicações *Java*, com uma interface de utilizador rica, que são executadas dentro de um *browser*. Normalmente, a parte visual é desenvolvida

[27] Adaptado de [JavaEETut12]

utilizando a biblioteca *Swing* da *Java* SE. Para que um *Applet* possa ser executado é necessário que seja instalado um *plugin* no *browser*, o que pode levantar alguns problemas de segurança, levando mesmo a situações em que não é permitido ao utilizador instalar este tipo de extras. Por este motivo, a utilização de *Applets* deverá ser substituída por aplicações *web* que serão apresentadas nos pontos seguintes;

- **Aplicações cliente (*Application Clients*):** Este tipo de aplicações é executado nas máquinas dos clientes e tratam-se de aplicações *desktop* com interfaces de utilizador ricas desenvolvidas utilizando bibliotecas disponibilizadas pela *Java* SE ou, menos frequente, com uma interface simples em modo de texto. As aplicações *desktop* comunicam, quase sempre, com a camada de negócios enquanto que as aplicações *web* comunicam, preferencialmente, com os componentes localizados na camada *web*;
- **Aplicações *Web (Web Applications)*:** Este tipo de componentes é constituído por *Java Servlets*, *JavaServer Faces* ou *JavaServer Pages* e são executados na camada *web*, como resposta a pedidos feitos por *Web Clients*. Os *Java Servlets* são classes desenvolvidas na linguagem *Java* que processam os pedidos HTTP recebidos e constroem respostas através da geração dinâmica de código HTML. Um exemplo tipo do funcionamento de um *Java Servlet* pode ser dado da seguinte forma: um *browser* efetua um pedido a um servidor *web*, solicitando uma lista de artigos disponíveis em armazém. O servidor *web* reencaminha o pedido para o *servlet* adequado e este, utilizando a API JDBC, ou comunicando com um *Enterprise Java Bean*, obtém a lista pretendida. O passo seguinte será a construção de uma tabela em HTML contendo a lista de artigos e enviá-la como resposta ao *browser*. O *JavaServer Pages* (JSP) é uma especificação que define a forma como se podem inserir elementos dinâmicos numa página HTML, que trabalha, de forma muito próxima, com os *Java Servlets*. Finalmente, o *JavaServer Faces* é uma especificação mais recente e está mais relacionada com a parte visual das aplicações. Baseada nos *Java Servlets* e nos JSP, permitem a criação de interfaces para o utilizador através da utilização de componentes já existentes, seguindo um princípio semelhante ao da biblioteca *Swing* para a criação de interfaces de utilizador para aplicações *desktop*;
- **Componentes Empresariais:** São componentes executados na camada de negócios podendo ser identificados, entre outros, os *Enterprise Java Beans*, *Java Transaction API* e o *Java Message Service*. Estes componentes, geralmente, estão relacionados com a definição e implementação das regras de negócio das organizações, podendo ser vistos como unidades de lógica ou dados do negócio. A colocação das regras do negócio numa camada isolada e independente apresenta várias vantagens, especialmente ao nível da robustez e da escalabilidade. Numa utilização padrão, estes componentes recebem dados para validação e gravação ou recebem pedidos para obtenção de informação armazenada na camada da base de dados.

4.2.2. JAVASERVER FACES

O *JavaServer Faces* (JSF) é um *framework* para o desenvolvimento de interfaces de utilizador, baseado na utilização de componentes visuais e desenvolvido sobre a plataforma *Java* EE [Jacobi06]. A primeira versão do JSF surgiu em 2004, tendo sido proposta pela JSR 127 em 2001. Ainda em 2004 surgiu a versão 1.1 tratando-se, apenas, de uma versão de correção de erros. Em 2006 surgiu uma nova versão que apresentava diversos melhoramentos e novas funcionalidades [Mann05]. Tratava-se da versão 1.2 que foi introduzida na plataforma *Java* EE *5* através da JSR 252. A versão 2.0 do JSF foi apresentada em 2009 tendo sido proposta pela JSR 314. A versão atual do *framework* é a 2.1 e foi disponibilizada em Outubro de 2010 tratando-se de uma pequena atualização relativamente à versão anterior. Para breve, juntamente com o lançamento do *Java* EE *7*, está prevista a disponibilização da versão 2.2 através da JSR 344 [Goncalves10].

Ao contrário de outros *frameworks*, este é definido pelo JCP fazendo, por isso, parte da plataforma *Java* EE. O principal objetivo que esteve na origem do JSF foi tornar o desenvolvimento *web* mais fácil, rápido e semelhante aos princípios associados ao desenvolvimento de aplicação para *desktop*. Isto representa uma alteração significativa nos modelos de desenvolvimento *web* uma vez que, ao utilizarem o *framework* JSF, os programadores poderão passar a raciocinar mais em termos de componentes visuais e nos eventos a eles associados. Com efeito, o desenvolvimento de uma aplicação usando JSF assemelha-se muito ao desenvolvimento de aplicações para *desktop*, existindo, inclusive, IDE que permitem o desenho da camada de apresentação fazendo *drag and drop* dos componentes a apresentar. Utilizando o *framework* JSF, um programador poderá realizar, de forma simples e rápida, o seguinte conjunto de tarefas:

- Criar páginas *web* arrastando e largando componentes visuais;
- Estabelecer ligações entre os componentes de uma página e os objetos situados no lado do servidor;
- Associar eventos gerados nos componentes visuais a métodos do lado do servidor;
- Manter o estado das aplicações para além do ciclo de vida de um pedido HTTP;
- Reutilizar e expandir as funcionalidades dos componentes visuais existentes.

Na figura 4.5 apresenta-se um esquema de uma aplicação JSF, onde estão representados os principais componentes[28]:

[28] Numa apresentação do JSF o termo *componente* pode ser utilizado em dois sentidos distintos: um componente (unidade de *software*) que faz parte da aplicação (p. ex. um *servlet*, página JSP, classe *Java*) ou um componente visual que será apresentado na camada de apresentação (p. ex. botões, caixas de introdução de texto ou grelhas). No âmbito deste capítulo, utilizaremos o termo componente de forma isolada para nos referirmos a uma unidade de *software* que constitui a aplicação e usaremos a expressão componente visual para nos referirmos a um objeto que será apresentado ao utilizador.

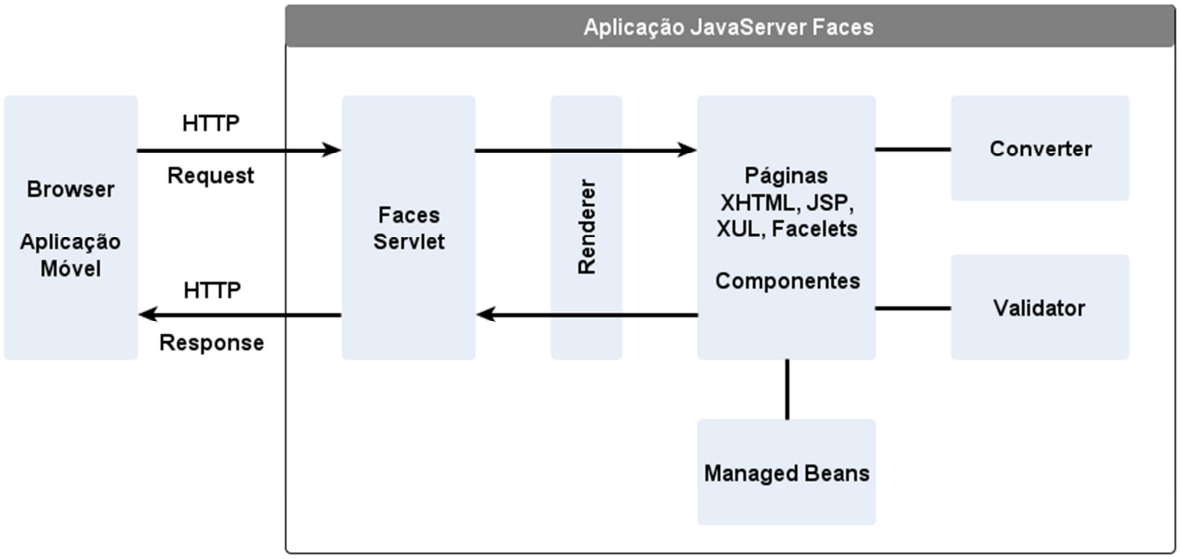


Figura 4.5 – Aplicação *JavaServer Faces*[29]

As aplicações JSF são aplicações *web* normais que intercetam pedidos HTTP através de um componente chamado *Faces Servlet* e geram código HTML como resposta. Uma das principais caraterísticas do JSF (e também uma das suas vantagens) é que não está dependente de nenhuma tecnologia específica para a geração da camada de apresentação, podendo a mesma ser gerada usando as mais diversas tecnologias e também para os mais diversos tipos de dispositivos. O mais usual é que a camada seja gerada utilizando código HTML e seja apresentada num *browser*.

O JSF está implementado seguindo o padrão *Model 2*, em que o *Faces Servlet* desempenha o papel do *Controller*. Todos os pedidos passam por este componente que os analisa e os reencaminha para os restantes componentes da aplicação.

As páginas e os componentes são usados para construir o HTML que será enviado como resposta. A arquitetura JSF permite que sejam utilizados diversos tipos de tecnologias para a definição da camada visual. Atualmente, as tecnologias mais utilizadas são os *JavaServer Pages* e os *Facelets*, sendo que, destas duas, o mais recomendado seja a utilização dos *Facelets*, uma vez que foram desenvolvidos após a criação da especificação dos JSF e dão uma melhor resposta. Os JSP, pelo contrário, são mais antigos que os JSF e, por isso, apresentam algumas limitações, especialmente quando se utiliza uma versão mais recente do JSF. Tanto os JSP como os *Facelets* definem um conjunto de componentes visuais aos quais se dá o nome de *Tree of Components* e que permitem que os utilizadores interajam com a aplicação. O JSF disponibiliza vários componentes visuais e também permite que os programadores possam criar outros que se adequem melhor às caraterísticas e necessidades das aplicações.

O código de apresentação dos componentes visuais poderá ser gerado de duas formas distintas: pelos próprios componentes visuais ou através de um novo componente

[29] Adaptado de [Goncalves10]

chamado de *Renderer*. A utilização de *renderers* permite que os componentes visuais se tornem independentes da tecnologia e dos dispositivos nos quais serão apresentados. Regra geral, cada tipo ou família de componente visual tem um respetivo *renderer* e estes podem ser organizados em *kits* tendo por base o dispositivo para o qual estão mais vocacionados. Assim, poderão existir *Render Kits* para, por exemplo, geração de código HTML, WML (*Wireless Markup Language*) ou SVG (*Scalable Vector Graphics*).

Os *Converters* são componentes que possibilitam a conversão e a formatação dos valores que estão do lado do servidor com o objetivo de serem apresentados aos clientes. Também possibilitam a operação inversa, ou seja, converterem os textos fornecidos pelos clientes e armazená-los nas estruturas de dados do lado do servidor. Por exemplo, uma data será sempre apresentada no dispositivo cliente como sendo uma cadeia de carateres mas, internamente, deverá ser trabalhada como um objeto *Date* ou *DateTime*. A função dos *converters* é ler e gravar adequadamente os valores nos objetos internos, existindo vários componentes pré-definidos para tratamento dos tipos de valores mais usuais.

Associados à função dos *Renderers* podem surgir os *Validators*, tratando-se de componentes que validam os dados antes de estes serem processados. Com esta função, garantem que os dados inseridos pelos utilizadores são válidos.

O último componente identificado na figura 4.5 são os *Managed Beans*. Tratam-se de simples classes escritas em *Java* (POJO) que permitem a sincronização dos componentes visuais com os valores, processos e lógica de negócio. Esta sincronização e associação são conseguidas através da utilização de uma notação chamada de *Expression Language*, a qual associa um componente visual a uma propriedade ou método de um *Managed Bean*. Podemos, por isso, afirmar que os *Managed Beans* são a representação dos componentes visuais no lado do servidor, guardando toda a informação relevante para o normal funcionamento da aplicação. Os *Managed Beans* desempenham mais funções no âmbito de uma aplicação JSF, sendo utilizados para controlar a navegação pelas páginas da aplicação e, também, para implementar as regras do negócio (através da utilização de *Enterprise Java Beans*).

Na lista seguinte apresentam-se as principais vantagens do *framework*:

- A tecnologia JSF favorece a separação de conceitos, nomeadamente uma efetiva separação entre a apresentação e o comportamento. Esta separação de conceitos possibilita uma clara definição dos papéis e responsabilidades dos diversos intervenientes no desenvolvimento de uma aplicação, assim como facilita o desenvolvimento e manutenção da própria aplicação;
- A possibilidade de se reutilizarem e expandirem os componentes visuais pode significar um aumento da produtividade dos programadores e da consistência visual ao longo da aplicação;
- Faz parte do conjunto de especificação da plataforma *Java* EE, o que garante estabilidade e evolução ao longo do tempo. Algumas das grandes empresas mundiais na área da informática (por exemplo, *Oracle*, *JBoss* e IBM) apresentam implementações do *framework JavaServer Faces*, o que garante elevados níveis de qualidade e de suporte. *JBoss RichFaces*, *PrimeFaces*, *ICEFaces*, *Oracle ADF Faces Rich Client*, *Apache MyFaces* são algumas das implementações da plataforma JSF;

- Disponibiliza um ambiente que facilita a manutenção do estado dos componentes ao longo da aplicação, a validação e processamento dos dados e gestão dos eventos;
- Existência de vários *Render Kits* que permitem a apresentação dos componentes para diversos tipos de dispositivos;
- Como faz uma abordagem ao desenvolvimento *web* seguindo alguns conceitos relacionados com o desenvolvimento para *desktop*, poderá ser um bom ponto de entrada para os programadores de aplicações *desktop*.

Ao nível das desvantagens, destacam-se as seguintes:

- Ao contrário de outros *frameworks* (por exemplo, o *Spring Framework*) não contempla todas as áreas de uma aplicação (desde a camada da base de dados até à camada de apresentação), deixando a possibilidade dos programadores selecionarem uma tecnologia que não se adeque corretamente;
- A utilização de ferramentas para desenho das páginas, embora represente um ganho substancial de tempo numa fase inicial, oculta muita informação e código, o que poderá traduzir-se numa aplicação mais pesada e menos eficiente;
- Uma vez que a gestão dos estados é feita de forma automática pelo *framework*, este pode não responder adequadamente em cenários de utilização intensiva.

4.2.3. APACHE STRUTS 2

O *Apache Struts 2* é um *framework open source* para o desenvolvimento de aplicações *web* utilizando a linguagem *Java* e que implementa o padrão MVC. A primeira versão do *framework* começou a ser desenvolvida em Maio de 2000 tendo sido oficialmente disponibilizada em Julho de 2001. Posteriormente, em 2002, o projeto passou para a alçada da *Apache Software Foundation*, tendo-se tornado num projeto principal em 2005. Desde o início, o *framework* teve uma grande aceitação da parte da comunidade, tornando-se num dos *standards* do desenvolvimento *web* na plataforma *Java*. Apesar deste sucesso, a evolução da plataforma foi-se tornando lenta e apresentando muitos problemas, o que deu origem ao surgimento de um novo *framework* chamado *Apache Struts 2*. Este *framework* não deve ser visto como uma atualização ao *Struts 1* mas sim como uma plataforma totalmente nova, baseada no trabalho desenvolvido pela empresa *OpenSymphony* no *framework WebWork*. Com efeito, foi a partir do *WebWork* que surgiu o *Apache Struts 2* com o objetivo de ultrapassar as limitações do *Struts 1* e de dar resposta às crescentes solicitações das aplicações *web*, através da implementação das melhores práticas de desenvolvimento [Brown08]. O *Apache Struts 2* apresenta diversas funcionalidades novas em relação ao *Struts 1*, sendo de destacar as seguintes [Roughley07]:

- ***Interceptors*** **–** Código que pode ser executado antes ou depois de determinada ação com o objetivo de separar conceitos genéricos dos conceitos especificamente

relacionados com a lógica do negócio (permitem uma programação orientada a aspetos);

- **Anotações *Java* –** As anotações *Java* foram introduzidas com o objetivo de reduzir substancialmente a dimensão e a quantidade de ficheiros XML;
- ***Object Graph Navigation Language* (OGNL) –** Linguagem usada internamente no *framework* para obtenção de dados de alguns componentes;
- ***UI Tag API* –** Conjunto de *tags* (*taglib*) para suporte de componentes gráficos.

Atualmente, o *Apache Struts 2* é disponibilizado segundo a *Apache License 2.0*, sendo a versão estável mais recente a 2.3.15.1. Os requisitos mínimos exigidos para se poder utilizar o *framework* são o *Java 5* ou superior, *Servlet* API *2.4* e JSP API 2.0[30]. Na figura 4.6 apresentam-se os principais componentes do *framework* e a sequência do processamento de um pedido.

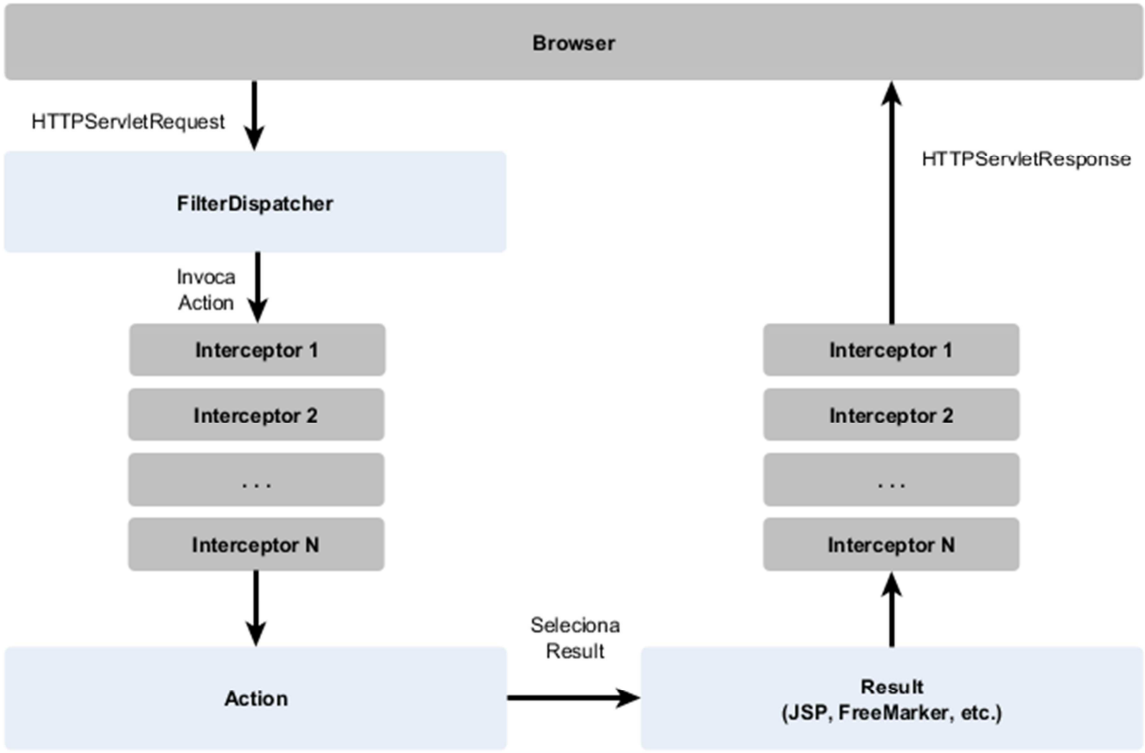


Figura 4.6 – Estrutura do *Apache Struts 2*[31]

O trabalho do *FilterDispatcher* é efetuar o mapeamento entre os pedidos recebidos do cliente e as *Actions*. Este objeto é da máxima importância dentro de uma aplicação *Apache Struts 2* uma vez que é por ele que passam todos os pedidos HTTP. Todo este trabalho de seleção das *Actions* correspondentes aos pedidos é gerido pelo *framework*, ficando o programador com a missão de indicar a relação entre os pedidos e as *Actions* e o

[30] http://struts.apache.org/development/2.x/, Março 2013
[31] http://struts.apache.org/development/2.x/docs/big-picture.html, Março 2013 (Adaptado)

desenvolvimento das últimas. Esta associação pode ser feita através de um ficheiro XML ou utilizando as anotações do *Java*. Atendendo a que existe apenas um ponto de entrada dos pedidos para a aplicação pode-se afirmar que o *Apache Struts 2*, para além de estar baseado na arquitetura MVC, também implementa o padrão *Front Controller* (o *FilterDispatcher* é o primeiro componente a entrar em ação como resposta a um pedido).

Depois de ter sido feita a seleção da *Action* a executar, será criada uma instância da mesma, passando para o objeto recém-criado o processamento do pedido. É na *Action* que será executado todo o código relacionado com a implementação e validação das regras do negócio, podendo, também, ser instanciados outros objetos para, por exemplo, interagir com bases de dados ou realizar outro tipo de operações necessárias à satisfação do pedido. Uma outra caraterística muito importante deste componente é que pode ser implementado utilizando apenas POJO não estando, por este motivo, fortemente associado a especificações rígidas. O único aspeto a ter em conta nas *Actions* é que os objetos têm que implementar um método chamado *execute* (no entanto, até esta caraterística pode ser alterada, desde que essa informação conste nas configurações da aplicação).

Quando a *Action* terminar o seu trabalho será altura de gerar a vista como resposta ao pedido efetuado pelo cliente. Nesta fase, surge o componente *Result* que tem como finalidade gerar e apresentar ao utilizador o estado da aplicação. Embora, de uma maneira geral, o *Result* esteja relacionado com a tecnologia *JavaServer Pages*, a realidade é que a camada de apresentação do *Apache Struts 2* permite a utilização de várias tecnologias para a apresentação do estado. Por exemplo, em substituição dos *JavaServer Pages* podem ser utilizadas as seguintes tecnologias: *Apache Velocity Template*, *FreeMaker Template* ou XSLT *Transformations*. Também é possível que o *Result*, em vez de gerar uma página, redirecione o *output* para outra *Action*, para o *JasperReports* ou outras tecnologias. Atendendo à grande variedade de tecnologias disponíveis para a camada de apresentação, às quais se podem juntar, ainda, as tecnologias associadas às RIA, torna-se extremamente importante esta separação de conceitos proporcionada pela arquitetura do *Apache Struts 2*.

Para além dos componentes referidos nos parágrafos anteriores, o *Apache Struts 2* disponibiliza outros componentes que também desempenham um papel muito importante no processamento dos pedidos. Destacam-se os seguintes [Roughley07]:

- *Interceptors*
- *ValueStack*
- *Object Graph Navigation Language (OGNL)*
- *Tag API*

Os *Interceptors* são componentes que se executam, de forma automática, antes e após o processamento de cada pedido. Cada *Action* pode estar associada a vários *Interceptors* e estes podem ser executados antes e/ou depois da *Action* realizar o seu trabalho. Normalmente, os *Interceptors* são usados para a realização de tarefas que, não fazendo parte das regras de negócio, são fundamentais para que estas sejam postas em prática. As principais atividades realizadas pelos *Interceptors* são, entre outras, as validações, tratamentos de exceções, registo (*logging*) de operações e disponibilizar um ponto de

execução de código pré e pós execução da *Action*. Tratam-se, essencialmente, de operações que são transversais a toda a aplicação e que tornam os *Interceptors* num dos componentes mais importantes do *framework*. Os programadores têm à sua disposição vários *Interceptors* pré-definidos pela plataforma que poderão utilizar nas suas aplicações. Para além deste tipo de *Interceptors* já existentes e disponíveis para utilização, é possível ao programador desenvolver *Interceptors* específicos para as suas aplicações. Fazendo uma comparação com o *Spring Framework*, podemos afirmar que os *Interceptors* estão para o *Apache Struts 2* tal como o AOP está para o *Spring Framework*.

O *ValueStack* pode ser definido como sendo um repositório central dos dados que são utilizados no processamento dos pedidos. Em vez dos dados serem passados de objeto para objeto durante o processamento, ficam armazenados numa área central da aplicação, acessível a todos os componentes. Os tipos de dados que o *ValueStack* contém podem ser agrupados em 4 categorias [Roughley07]:

- ***Temporary Objects*** **–** Objetos temporários que são criados durante o processamento dos pedidos;
- ***Model Object*** **–** Quando a *Action* implementa o interface *ModelDriven*, o objeto será colocado no ValueStack;
- ***Action Object*** **–** A *Action* que, atualmente, está a ser executada;
- ***Named Objects*** **–** Objetos internos do *framework* que caracterizam e fornecem informações sobre o processo que está a ser executado e sobre o próprio *framework*.

A leitura e gravação de dados no *ValueStack* devem ser feitas utilizando a OGNL, a qual pode ser utilizada pelos programadores e internamente pelo *framework*.

A *Tag API* é constituída por um conjunto de *tags* que são utilizadas para a criação dinâmica de páginas. A utilização destas *tags* permite uma ligação estreita entre o processo de criação das páginas e as *Actions* assim como a obtenção dos dados armazenados no *ValueStack*. Grande parte dos *frameworks* possuem uma biblioteca de *tags* para a construção das páginas mas a API do *Apache Struts 2* permite uma integração maior com o *framework* o que garante um aproveitamento maior das suas caraterísticas (por exemplo, a possibilidade de recorrerem ao *ValueStack* para a obtenção de informação). Identificam-se quatro categorias de *tags* [Roughley07]:

- ***Data Tags*** **-** As *tags* desta categoria destinam-se, essencialmente, ao manuseamento de informação resultante das *Actions* que foram executadas, bem como à obtenção e escrita de dados no *ValueStack*;
- ***Control Tags*** **-** As *Control Tags* destinam-se a alterar, de forma condicional, o processo de construção das páginas e a definir qual a informação que será apresentada;
- ***UI Tags*** **-** Tratam-se de *tags* que encapsulam as *Form Tags* do HTML assim como outros componentes visuais adicionais que possam ser utilizados nas páginas;
- ***Miscelaneous Tags*** **-** Engloba as *tags* que não se enquadram nas restantes categorias.

Atendendo à enorme volatilidade que existe na área do desenvolvimento de aplicações *web*, o *Apache Struts 2* disponibiliza um eficiente mecanismo de *plugins*. Os *plugins* poderão ser utilizados para adicionar novas funcionalidades ao *framework* ou para alterar a forma como são realizadas determinadas operações. Na lista seguinte apresentam-se as principais vantagens associadas à utilização do *Apache Struts 2*:

- Arquitetura simples: A arquitetura do *Apache Struts 2* carateriza-se por ser bastante simples quando comparada com outros *frameworks*. Esta simplicidade pode reduzir o tempo e aprendizagem da plataforma e, também, a probabilidade de ocorrência de erros;
- Expansibilidade: Apesar de se tratar de uma arquitetura simples tem uma grande capacidade de expansão através de um eficiente mecanismo de *plugins*. Esta caraterística tem ainda a vantagem de não sobrecarregar o *framework* com funcionalidades que os utilizadores não necessitem. As funcionalidades podem ser adicionadas ao *framework* à medida que vão sendo necessárias;
- Facilidade na integração com AJAX e RIA: Existem *plugins* que possibilitam a integração do *framework* com as bibliotecas *Dojo Toolkit* e *jQuery UI* para o desenvolvimento de aplicação ricas e aproveitamento das técnicas AJAX;
- Simplicidade nos processos de configuração: Sempre que possível, utiliza o princípio *Convention over Configuration*, o que permite uma substancial redução de ficheiros de configuração. A utilização de anotações *Java* também ajuda neste processo;
- Comunidade muito extensa: A comunidade de programadores e utilizadores do *Apache Struts 2* é muito grande, o que pode ser benéfico para a obtenção de ajuda e apoio;
- Testabilidade: A arquitetura do *framework* e a utilização preferencial de POJO favorece a realização de testes às aplicações.

Quanto aos aspetos negativos do *Apache Struts 2*, apresentamos os seguintes:

- Apesar do *Struts 2* permitir a utilização de anotações *Java* para efeitos de configuração, continua-se a utilizar muitos ficheiros XML. Não se tratando de um defeito da plataforma, acaba por ser um facto muito visível que transitou do *Struts 1*;
- Embora a comunidade relacionada com o *Struts* seja muito grande, a documentação existente não é muito extensa e está mal organizada. Pesquisas efetuadas nos motores de busca devolvem informação maioritariamente relacionada com o *Struts 1*;
- Nos últimos tempos o *framework* tem perdido alguma preponderância e visibilidade em relação a outros mais recentes ou que disponibilizem abordagens diferentes;
- Como o *Apache Struts 2* é muito diferente do *Struts 1*, os processos de conversão das aplicações tendem a ser demorados. Este fator também pode contribuir para

que muitos programadores optem por outros *frameworks* para o desenvolvimento de novas aplicações.

4.2.4. GOOGLE WEB TOOLKIT

O *Google Web Toolkit* (GWT) é um *framework* baseado na linguagem *Java* que permite a criação de aplicações *web* ricas de uma forma simples e rápida. Utilizando este *framework*, os programadores desenvolvem o código na linguagem *Java*, o qual, posteriormente, será compilado para linguagem *JavaScript* para que possa ser executado no lado do cliente [Smeets08]. Embora se trate de uma plataforma muito vocacionada para o desenvolvimento de interfaces gráficas para o utilizador que serão sempre executadas no lado do cliente, também permite o desenvolvimento de código que é executado do lado do servidor. A primeira versão do GWT surgiu em Maio de 2006 tendo a versão 2.0 sido lançada em finais de 2009. Durante este período de tempo foram sendo lançadas novas versões intermédias, cada uma delas apresentando diversas melhorias e novas funcionalidades. A versão atual é a 2.5.1, tendo sido lançada em Março de 2013[32]. O *framework* GWT pode ser dividido em 3 partes distintas [Smeets08]:

- *Java to JavaScript Compiler*
- JRE *Emulation Library*
- UI *Building Library*

O *Java to JavaScript Compiler* é o componente central de todo o *framework*, sendo, como se depreende a partir do nome, um compilador que converte o código *Java* escrito pelo programador em código *JavaScript*, tal como um compilador *Java* converte o código escrito pelo programador em *byte-code*. No entanto, o compilador do GWT apresenta uma caraterística muito importante que é o facto de apenas converter para *JavaScript* o código que é efetivamente utilizado pela aplicação, conseguindo-se, desta forma, um código *JavaScript* mais pequeno e eficiente. O inconveniente desta técnica é que torna o processo de compilação mais lento e obriga a que esteja disponível todo o código fonte das bibliotecas utilizadas pela aplicação. Outro aspeto muito importante neste processo é o output gerado; o compilador criará código *JavaScript* específico para cada *browser* de destino, sendo criados, por este motivo, vários ficheiros, cada um contendo código específico para um determinado *browser*. Isto significa que os programadores não precisam de se preocupar com a forma como cada um dos *browsers* processa o código *JavaScript*, aumentado consideravelmente a velocidade de desenvolvimento do *software*. Todos estes aspetos são tratados de forma automática pelo compilador *Java to JavaScript* sem qualquer intervenção por parte dos programadores. Apesar do código ser escrito em *Java* e, posteriormente, ser convertido para *JavaScript*, o programador pode inserir código

32
http://www.gwtproject.org/versions.html, Abril 2013
https://docs.google.com/presentation/d/1pC9WK80-fzIs2iMQOO3Jsvfmqv2erI9xucuF3lHOE7Q/edit?pli=1#slide=id.g11d0accd_0_8, Abril 2013

JavaScript no código *Java* utilizando a API *JavaScript Native Interface* (JSNI). Esta API pode ser útil quando é necessário efetuar uma chamada direta ao código *JavaScript* a partir do código *Java*. O compilador do GWT pode gerar código *JavaScript* de três modos diferentes [Hanson13]:

- ***Obfuscated* –** Neste modo, o código *JavaScript* é compactado ao máximo, tornando a sua leitura extremamente difícil. Todos os carateres não essenciais são removidos e todos os objetos, variáveis e funções terão nomes compostos pelo menor número possível de carateres;
- ***Pretty* –** O código *JavaScript* gerado por este modo já é de mais fácil leitura, sendo utilizados nomes mas descritivos para os objetos, funções e variáveis. Aqui também são apresentados os espaços que os programadores possam ter usado para efeitos de formatação;
- ***Detailed* –** Este modo é semelhante ao *Pretty* sendo que os nomes dos elementos também passam a indicar o nome da classe à qual pertencem.

O JRE *Emulator Library* também desempenha um papel fundamental no âmbito do GWT. Como já foi referido, o processo de compilação requer que o código fonte *Java* das bibliotecas utilizadas esteja disponível. Esta necessidade também se aplica às classes que constituem o JRE. É neste ponto que o JRE *Emulation Library* desempenha o seu papel, fornecendo um mapeamento das classes que constituem o JRE e as suas correspondentes em *JavaScript* e disponibilizando o código fonte necessário para o processo de compilação. Ao desenvolver código *Java* que será compilado para *JavaScript* apenas se poderão utilizar as classes emuladas pelo JRE *Emulator Library*. A título de exemplo, não é possível utilizar classes de gravação de ficheiros nem classes de interação com bases de dados SQL. Este tipo de código terá que ficar do lado do servidor uma vez que aí não há qualquer limitação à utilização das classes do JRE. As limitações impostas pelo JRE *Emulator Library* apenas se aplicam ao código que será compilado para *JavaScript*. Todo o restante código poderá utilizar as classes *Java* sem qualquer restrição ou condição especial.

Finalmente, o último elemento do GWT é o UI *Building Library*, sendo constituído por classes relacionadas com a parte visual da aplicação e com a construção de componentes específicos. Os componentes visuais podem ser divididos em dois grandes grupos: *widgets* e *panels*. Os primeiros são componentes visuais a apresentar nas páginas e que serão usados para interagir com os utilizadores (por exemplo, botões ou caixas de texto). Os *panels*, por sua vez, são utilizados para organizar o aspeto das aplicações e para conterem outros componentes.

Uma vez que o código é desenvolvido na linguagem *Java*, favorece-se a utilização das melhores práticas definidas na plataforma e pode-se aproveitar as várias ferramentas de desenvolvimento disponíveis. Contudo, para que se possa tirar proveito das referidas ferramentas e da linguagem *Java* é necessário que o código seja executado em ambientes específicos para as linguagens *Java* e *JavaScript*. Assim, pode-se identificar dois modos distintos para a execução de aplicações GWT [Smeets08]:

- ***Hosted Mode* –** Neste modo, as aplicações são executadas sempre na linguagem *Java* o que permite que as mesmas sejam depuradas e tratadas como se se

tratassem de aplicações *Java* normais. Neste modo a aplicação é executada numa versão específica de um *browser*;
- ***Web Mode* –** Neste modo, o código *Java* é compilado para *JavaScript* e é executado normalmente dentro de um *browser*.

Numa aplicação desenvolvida usando o *framework* GWT identificam-se 4 grandes elementos, dos quais apenas um é opcional [Smeets08]:

- ***Module Descriptor* –** Trata-se de um ficheiro que descreve e define as configurações da uma aplicação GWT. Uma informação muito importante fornecida por este ficheiro é o nome da classe que contém o ponto de entrada da aplicação;
- ***Public Resources* –** Constituído por todos os ficheiros que poderão ser apresentados durante a execução da aplicação. Podemos indicar, por exemplo, imagens, páginas estáticas ou a *Host Page* (página principal da aplicação que contém o ponto de arranque da aplicação GWT);
- ***Client-Side Code* –** Este é o código que o compilador irá converter para *JavaScript* e que, posteriormente, será executado num *browser*;
- ***Server-Side Code* –** Esse é o elemento opcional da aplicação uma vez que nem todas as aplicações GWT necessitam de comunicar com um servidor. Trata-se de código *Java* que é executado no lado do servidor e que, ao contrário do *Client-Side Code*, não está limitado nem será convertido para *JavaScript*. A comunicação entre o código que corre no servidor e o código que é executado no lado do cliente pode ser feita utilizando duas técnicas diferentes disponibilizadas pelo GWT: utilizando a classe *RequestBuilder*, que permite que uma aplicação GWT comunique com qualquer tipo de servidor (os quais até podem ter sido escritos noutras linguagens e nada terem a ver com o GWT); a comunicação também pode ser feita usando o GWT-RPC que é uma técnica mais específica do GWT e que só permite a comunicação entre aplicações Java.

Ao trabalhar com o *framework* GWT o programador fica afastado das principais dificuldades que o desenvolvimento na linguagem *JavaScript* apresenta, criando-se uma camada de abstração que facilita o desenvolvimento das aplicações. Apesar desta camada de abstração, o programador poderá intercalar código *Java* com código *JavaScript*. Nas listas seguintes apresentam-se as vantagens e desvantagens da utilização do GWT. Relativamente às vantagens, salientam-se as seguintes:

- O desenvolvimento das aplicações utilizando a linguagem *Java* e seguindo princípios de programação associados ao desenvolvimento para *desktop* permite que os programadores *Java* sejam mais eficientes e desenvolvam aplicações de forma mais rápida (também se pode aplicar aos programadores de outras linguagens e que apenas tenham experiência no desenvolvimento de aplicação para *desktop*). Adicionalmente, possibilita que se continuem a utilizar as ferramentas de desenvolvimento associadas à linguagem *Java*;

- O código *JavaScript* gerado pelo GWT tende a ser muito mais rápido e eficiente do que o código gerado por outras vias;
- O GWT permite uma clara separação de conceitos havendo identificação muito clara do código que será executado do lado do cliente e do código de lado do servidor;
- Algumas ferramentas de desenvolvimento permitem que se desenhem as interfaces do utilizador o que facilita o trabalho e reduz substancialmente o tempo de desenvolvimento;
- O *framework* disponibiliza vários componentes visuais, os quais podem ser expandidos para melhor se adequarem às especificidades de cada aplicação. Na eventualidade dos componentes disponibilizados pelo GWT não serem suficientes existem várias bibliotecas de terceiros que fornecem mais componentes (por exemplo, *Vaadin*, *Sencha GXT* e *Smart GWT*).

Quanto às desvantagens, destacam-se as seguintes:

- As aplicações GWT não são facilmente indexáveis pelos motores de busca (à semelhança do que acontece com outros *frameworks* RIA);
- Se o utilizador, por questões relacionadas com políticas de segurança, tiver a execução do *JavaScript* desativada no *browser*, as aplicações não serão executadas, aparecendo, apenas, a *host page*. Em situações extremas, isto pode implicar que seja desenvolvida, em paralelo, uma versão específica para as situações em que a execução do *JavaScript* esteja desativada;
- Embora haja uma clara separação entre o código que será executado no servidor e no cliente, não existe uma separação efetiva entre este último e o código HTML nem é muito facilitada a participação de *web designers* uma vez que toda a apresentação é gerada através da linguagem Java. A exceção a esta regra é a *host page* uma vez que se trata de uma página HTML simples. No entanto, a partir do momento em que se inicia a aplicação, toda a parte visual será tratada de forma diferente;
- Apesar de permitir alguma interferência na geração do código *JavaScript*, este processo tende a ser muito automatizado e pouco controlável pelo programador.

4.2.5. SPRING FRAMEWORK

O *Spring Framework* é um *framework* para desenvolvimento de aplicações empresariais que se baseia em técnicas de *Dependency Injection* (DI) e *Aspect Oriented Programming* (AOP) e cujo objetivo é tornar o desenvolvimento de aplicações *Java* EE mais simples e robusto [Walls11]. O *framework* surgiu pela primeira vez com a publicação do livro *Expert one-on-one* J2EE *Design and Development* em 2002. Neste livro, o autor (Rod Johnson) apresentou a sua experiência enquanto programador na plataforma *Java* EE indicando um conjunto de boas práticas que facilitavam o desenvolvimento e como obter sucesso no desenvolvimento e implementação deste tipo de projetos. Adicionalmente, foi

disponibilizado para *download* um exemplo do *framework*, o que muito contribuiu para a sua implementação e para demonstrar o seu enorme potencial. Após a publicação do livro e da distribuição do protótipo do *framework*, a primeira versão oficial surgiu em Março de 2004. Depois do lançamento da primeira versão, foram sendo lançadas diversas atualizações, sendo de destacar a versão 2.0 saída em Outubro de 2006 e a versão 3.0 em Dezembro de 2009. A versão atual do *framework* é a 3.2.4 tendo sido lançada em Agosto de 2013 sob a licença *Apache License 2.0*. Embora se trate de um *framework* de código aberto, o seu desenvolvimento e manutenção é feito por uma sociedade comercial chamada *SpringSource*. O desenvolvimento do *Spring Framework* baseou-se num conjunto de princípios que a seguir se apresentam [Johnson05]:

- **Simples mas poderoso –** Embora o principal objetivo do *framework* seja a simplificação do desenvolvimento de aplicações *Java* EE, esta simplificação não é conseguida à custa de perda de funcionalidades. Por exemplo, embora seja possível interagir com uma base de dados sem ser necessário prestar atenção aos detalhes do JDBC, o programador terá sempre a possibilidade de utilizar diretamente a API do JDBC. O *Spring Framework* cria uma camada de abstração mas esta poderá ser utilizada ou não pelos programadores;
- **Flexibilidade –** O *Spring Framework* é constituído por vários módulos totalmente independentes uns dos outros. Este nível de independência possibilita que apenas sejam utilizados os módulos necessários, tornando o *framework* muito mais adaptável às necessidades de cada projeto;
- **Possibilidade de escolha –** O *framework* não impõe nenhuma tecnologia ou abordagem específica, antes deixando ao programador a possibilidade de escolher a que melhor resposta dá às suas necessidades. Por exemplo, na área da persistência das aplicações, os programadores podem optar pelo JDBC, *Hibernate* ou JPA, entre outros. O mesmo se pode referir noutras áreas, tais como nos *frameworks* de aplicações *web* onde é possível utilizar, por exemplo, o *JavaServer Faces* ou o *Apache Struts 2* em substituição do *Spring* MVC;
- **Independência do código relativamente ao *framework* –** Ao contrário de outros *frameworks* que impõem que as classes utilizadas respeitem e implementem as regras e interfaces próprios, o *Spring* minimiza essa dependência do código, permitindo, sempre que possível, que seja totalmente independente. A maior parte das classes de uma aplicação *Spring* não obedece a nenhuma interface específica nem expande nenhuma classe. Com isto consegue-se que o código seja facilmente transferido para novas versões do *Spring* ou até que seja utilizado fora do contexto de uma aplicação *Spring*. Um outro aspeto positivo desta independência é que facilita o aproveitamento de código antigo que não tenha sido desenvolvido para o *Spring Framework*;
- **Facilita a realização de testes –** Ao promover a criação de classes isoladas e independentes, facilita-se muito a realização de testes às aplicações, aumentando, assim, a qualidade geral do *software*.

Para que estes princípios pudessem ser colocados em prática foi necessário criar uma arquitetura baseada em objetos simples e independentes. Na plataforma *Java* estes objetos, normalmente, são chamados de *Plain Old Java Objects* (POJO). Para que seja

possível desenvolver uma aplicação empresarial baseada em POJO, o *Spring Framework* socorre-se de três técnicas diferentes [Wolff08]:

- *Dependency Injection* (DI);
- *Aspect Oriented Programming* (AOP);
- *Enterprise Service Abstraction* (ESA).

A DI é a tecnologia que mais identifica o *framework* e é esta técnica que permite a independência entre os objetos, uma vez que esta apenas é definida durante a execução da aplicação, sendo da responsabilidade do *Spring* a sua definição e o relacionamento dos objetos entre si.

O *Spring Framework* também facilita a implementação de técnicas de AOP, o que permite uma separação do código relacionado com as regras de negócio do código relacionado com os serviços do sistema. O exemplo típico da utilização do AOP são os serviços de gestão das transações das bases de dados, segurança e registo de atividades. Este tipo de código deve estar isolado das regras de negócio e o *Spring* favorece este tipo de programação. Por exemplo, o código de gestão das transações de uma base de dados não deve estar colocado num POJO que gere a informação de uma entidade. Devem existir mecanismos que iniciem e concluam automaticamente uma transação sempre que se gravar alguma informação na base de dados.

Por último, o ESA permite que se utilizem as API do *Java* de uma forma muito mais simples, escrevendo menos código e com menor probabilidade de ocorrência de erros. Desta forma, o *Spring Framework* permite que o programador se concentre mais na implementação das funcionalidades relacionadas com o negócio, deixando que as complexidades das API sejam tratadas pelo *framework*.

Apesar de simplificar o desenvolvimento de aplicações *Java* EE com base num conjunto de princípios, o *Spring Framework* abrange todas as camadas das aplicações empresariais disponibilizando soluções adequadas e independentes em cada uma delas. O *Spring Framework* é composto por cerca de 20 módulos que se encontram agrupados em 6 categorias diferentes, tal como apresentado na figura 4.7.

Estes módulos disponibilizam todas as funcionalidades essenciais para o desenvolvimento de aplicações empresariais. No entanto, o programador terá sempre a possibilidade de selecionar quais os módulos que utilizará no seu trabalho. As caraterísticas mais relevantes das categorias dos módulos do *Spring Framework* são [Johnson13]:

- ***Core Spring Container* –** Este *container* é o núcleo do todo o *framework*. É aqui que são geridos os objetos usados pela aplicação. A criação dos objetos (geralmente chamados de *Beans*) é feita no módulo *Beans* (que também pode ser chamado de *Beans Factory*), assim como a sua eliminação e a definição das dependências entre eles. Para além da gestão dos objetos, também é neste *container* que se disponibilizam as diversas formas de acesso às configurações das aplicações (através dos chamados *Application Context*) e são disponibilizados outros serviços empresariais, tais como os serviços de correio eletrónico e de

integração com os *Enterprise Java Beans*. O módulo *Expression* disponibiliza a *Spring Expression Language* (SpEL) que é uma poderosa linguagem que permite efetuar consultas e manipular uma coleção de objetos. Embora seja semelhante a outras linguagens do género, a SpEL disponibiliza um conjunto mais vasto de funcionalidades (como, por exemplo, a possibilidade de se invocarem métodos dos objetos) e permite que se utilize uma linguagem uniforme em todo o ambiente *Spring*;

- ***AOP & Aspects*** **–** Os módulos deste *container* servem de base para o desenvolvimento do código segundo os princípios do AOP permitindo implementar conceitos que são transversais a várias áreas ou objetos. Esta implementação processa-se de forma totalmente independente e transparente, sem afetar ou alterar as áreas ou objetos sobre os quais incide. O módulo *Aspects* é utilizado quando se pretende fazer integração com o *AspectJ*;
- ***Instrumentation*** **–** Este módulo permite medir e analisar o desempenho das aplicações, efetuar o diagnóstico de erros e registar informação que seja considerada pertinente. Estas operações são realizadas de forma não intrusiva através de técnicas de AOP. Pelas suas caraterísticas e pelo tipo de informação que pode prestar, trata-se de um módulo bastante importante no âmbito do *Spring*;
- ***Data Access & Integration*** **–** Com os módulos deste *container*, os programadores têm a possibilidade de desenvolver código que interage com bases de dados sem terem que se preocupar com os aspetos mais difíceis da API JDBC nem com as especificidades de cada base de dados. Caso se opte pela utilização de um *Object-relational mapping* (ORM), os programadores podem trabalhar com um dos vários ORM suportados, através do módulo ORM. O *Spring Framework* não possui um ORM próprio mas permite que sejam utilizados outros. Ainda neste *container*, existe um módulo que permite uma abstração sobre o *Java Message Service* (JMS) disponibilizando um serviço de integração assíncrona entre aplicações. Também são disponibilizados módulos para o mapeamento Objetos/XML. O último módulo deste *container* é o sistema de gestão de transações nas bases de dados, o qual, fruto dos seus níveis de abstração, pode ser utilizado juntamente com as mais variadas API;
- ***Web & Remoting*** **–** Neste *container* estão colocados diversos módulos relacionados com o desenvolvimento da componente *web* de uma aplicação empresarial. O módulo *web* é utilizado para realizar diversas operações de baixo nível relacionadas com este tipo de aplicações. Apesar do *Spring* permitir a integração com diversos *frameworks* de aplicações *web* (por exemplo, o *Apache Struts 2* e o *JavaServer Faces*) também disponibiliza um *framework* próprio chamado de *Spring Model-View-Controller* (MVC). A implementação deste *framework* é feita nos módulos *Servlet* e *Portlet*. O módulo *Struts*, tal como o nome indica, contém um conjunto de classes para integração com o *framework Apache Struts 2*, embora, neste caso, estejamos perante um módulo que se encontra obsoleto. Para além das funcionalidades referidas, os módulos deste *container* também disponibilizam várias funcionalidades que visam a interação com outras aplicações. São exemplo destas funcionalidades os *Remote Method Invocation* (RMI), *Hessian* e JAX-WS;
- ***Testing*** **–** Reconhecendo a importância que os testes têm no desenvolvimento de aplicações robustas e fiáveis, o *Spring Framework* disponibiliza um módulo

vocacionado para esta atividade, utilizando o *JUnit* ou o *TestNG*. Poderão ser realizados testes unitários ou testes de integração sendo disponibilizados, para cada caso, as funcionalidades adequadas.

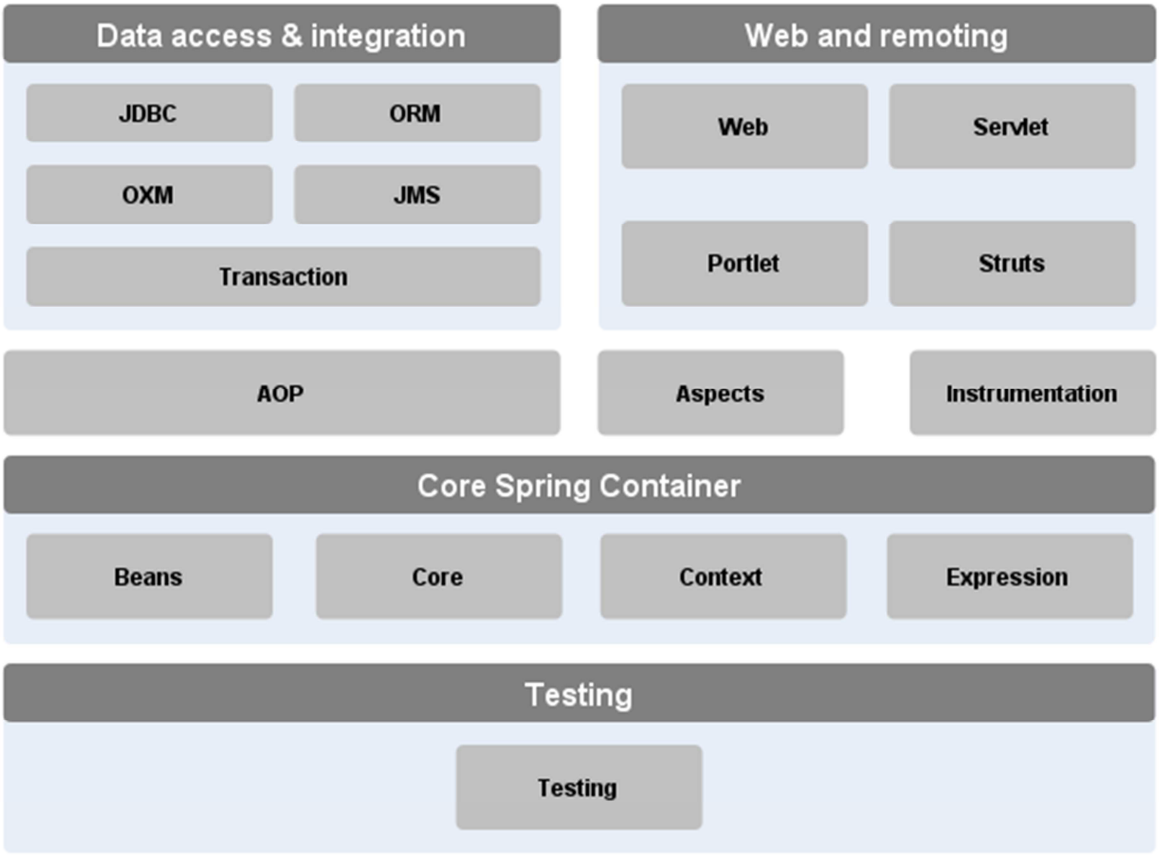


Figura 4.7 – Módulos do *Spring Framework*[33]

Na secção anterior fizemos uma apresentação dos módulos que compõem o *Spring Framework*. Contudo, esta lista não inclui todas as ferramentas, tecnologias e projetos relacionadas com o *Spring*. Este conjunto adicional e opcional de ferramentas, que podem ser descarregadas de forma individual, denomina-se de *Spring Portfolio* e é constituído, fundamentalmente, pelo seguinte [Wolff08;Walls11]:

- ***Spring Web Flow* –** *Framework* criado com base no *Spring MVC*, promovendo o desenvolvimento de aplicações *web flow-based*, em que os utilizadores são guiados ao longo da aplicação tendo em vista um determinado objetivo (o exemplo mais comum deste tipo de aplicações são os *wizards* e os passos para finalização de um carrinho de compras numa loja *on-line*);
- ***Spring Web Services* –** Este módulo centra-se na criação de *web services* do tipo *document-driven*. Embora o *Spring Framework* também permita a criação de *web services*, estes serão sempre baseados em interfaces e código já existente (*contract-*

[33] Adaptado de [Johnson13]

last) algo que, com o Spring WS deixa de ser obrigatório, uma vez que privilegia o desenvolvimento de *web services* segundo o modelo *contract-first* (o código será desenvolvido com base num WSDL já existente);

- ***Spring Security* –** Este módulo, desenvolvido usando o módulo *Spring AOP*, disponibiliza diversas funcionalidades relacionadas com a segurança das aplicações, sem influenciar as áreas e o código relacionado com as regras de negócio. Sendo utilizado, maioritariamente, para serviços de autenticação, registo e controlo de acessos, também disponibiliza a integração com outras tecnologias relacionadas com a segurança, tais como certificados, JAAS e LDAP;
- ***Spring Integration* –** Atendendo à importância que a integração de sistemas tem nos ambientes empresariais, o *Spring Framework* disponibiliza um módulo que contém vários padrões de desenvolvimento para integração de sistemas;
- ***Spring Batch* –** Esta biblioteca disponibiliza funcionalidades que facilitam o desenvolvimento de aplicações para processamento de dados em lote. Este tipo de *software* apresenta algumas dificuldades e cuidados a ter, como, por exemplo, gestão de erros e controlo de tentativas. O *Spring Batch* apresenta modelos de desenvolvimento que levam em consideração este tipo de desafios;
- ***Spring Social* –** O *Spring Social* é uma extensão do *Spring Framework* que simplifica a integração das aplicações com as diversas plataformas de redes sociais atualmente existentes;
- ***Spring Mobile* –** Tratando-se de um dos módulos mais recentes da plataforma, o *Spring Mobile* facilita o desenvolvimento de aplicações *web* vocacionadas para as plataformas móveis. Continuando no âmbito do desenvolvimento para dispositivos móveis, pode-se, ainda, identificar o *Spring Android*;
- ***Spring .NET* –** Este projeto tem como objetivo transportar para a plataforma .NET todas as caraterísticas e funcionalidades associadas ao *Spring Framework*. Adicionalmente, é possível utilizar tecnologias específicas da plataforma .NET tais como o ADO.NET, ASP.NET ou *NHibernate*;
- ***Spring Flex* –** O principal objetivo deste projeto é facilitar o desenvolvimento de aplicações *web* ricas utilizando o *Adobe Flex* como camada de apresentação. Com o *Spring Flex* é possível estabelecer comunicações entre o *Adobe Flex* e o código *Java* localizado no servidor utilizando uma tecnologia de integração chamada *Spring BlazeDS*;
- ***Spring LDAP* –** Trata-se de uma biblioteca que facilita e simplifica a realização de operações relacionadas com o LDAP (*Lightweight Directory Access Protocol*). O princípio associado a esta biblioteca é semelhante ao utilizado como JDBC: simplificar ao máximo a utilização de bibliotecas através do encapsulamento dos aspetos mais difíceis;
- ***Spring Dynamic Modules* –** O *Spring Dinamic Modules* permite associar as características da injeção de dependência do *Spring Framework* à modularidade dinâmica do OSGi, sendo possível criar *software* constituído por módulos totalmente independentes e de acordo com as caraterísticas de um *framework* OSGi. Este projeto teve um enorme impacto na área do OSGi ao ponto de ter sido incluído na especificação deste *framework* como o OSGi *Blueprint Container*. Em 2009, a *SpringSource* transferiu o *Spring Dinamic Modules* para o projeto *Eclipse*, passando a chamar-se *Gemini Blueprint*;

- ***Spring Rich Client* –** Projeto desenvolvido com o objetivo de disponibilizar um modelo de desenvolvimento de aplicações ricas para *desktop* (usando a biblioteca *Java Swing*) tendo por base e seguindo as boas práticas do *Spring Framework*. Com este projeto, os programadores poderão criar aplicações ricas, altamente configuráveis e bem estruturadas;
- ***Spring ROO* –** O *Spring ROO* é uma ferramenta de desenvolvimento rápido de aplicações *Spring*, permitindo, em poucos minutos, criar uma aplicação que segue as melhores práticas de desenvolvimento. Esta ferramenta caracteriza-se por gerar código *Java* facilmente compreensível e alterável e tem como principal objetivo aumentar o desempenho e a produtividade dos programadores *Java*, sem, no entanto, colocar em causa a robustez e fiabilidade dos sistemas.
- ***Spring Extensions* –** O *Spring Extensions* é um projeto desenvolvido por uma comunidade muito ativa de programadores e tem como objetivo expandir o *Spring Framework* a outras áreas ou tecnologias e adicionar novas funcionalidades aos projetos existentes. Como exemplo, referem-se uma implementação do *Spring Framework* para a linguagem de programação *Python*, a integração do *Spring* com as bases de dados *db4o* e *CouchDB* e a extensão do *Spring Security* para o protocolo de autenticação *Kerberos* e para o *standard* XML *Security Assertion Markup Language* (SAML);
- ***Spring Payment Services* –** Este é um dos projetos mais recentes da plataforma e tem como objetivo a integração no *Spring Framework* de um vasto conjunto de serviços de pagamentos *on-line*. O desenvolvimento deste projeto conta com o apoio da *Visa* mas, por ainda se encontrar numa fase de desenvolvimento, não poderá ser utilizado em ambientes de produção;
- ***Spring Tool Suite* –** O *Spring Tool Suite* (STS) é um IDE (*Integrated Development Environment*) baseado no IDE *Eclipse*. Para além de todas as funcionalidades disponibilizadas pelo *Eclipse*, o STS apresenta, ainda, várias ferramentas vocacionadas para o desenvolvimento de aplicações *Spring* (p. ex. exemplos de códigos-fonte e acesso à base de conhecimento da *SpringSource*)

Ao longo desta análise foram sendo referidos diversos aspetos positivos e princípios que justificam a utilização do *Spring Framework*. Na lista seguinte, apresentam-se mais algumas vantagens relevantes [Johnson05]:

- O *Spring Framework* disponibiliza camadas de abstração sobre as mais diversas tecnologias, libertando o programador das complexidades e especificidades das API, o que permite que se dedique e concentre mais no desenvolvimento das regras de negócio das aplicações. Apesar desta abstração, o programador tem sempre a possibilidade de trabalhar diretamente com a API, caso seja necessário;
- "Não reinventa a roda". Apesar do *Spring* ser um *framework* que abrange todas as áreas das aplicações empresariais, não tenta apresentar soluções próprias para problemas para os quais já existem soluções comprovadas. Por exemplo, é possível utilizar qualquer um dos ORM mais conhecidos do mercado sem qualquer limitação. Ao nível da camada de apresentação, é possível utilizar o *Spring MVC* ou utilizar outras tecnologias disponíveis (p. ex. *JavaServer Faces*, *Apache Struts 2*, *Sencha Ext JS*);

- Ao ser desenvolvido com base em módulos, permite que sejam utilizados, apenas, os módulos necessários. A utilização de POJO, por outro lado, permite uma maior facilidade na reutilização de *software* que não tenha sido escrito para o *Spring* e, no sentido inverso, a utilização de código desenvolvido para o *Spring* noutras plataformas;
- Embora seja distribuído sob a *Apache License*, o *Spring Framework* é desenvolvido por uma grande empresa, o que garante estabilidade, manutenção e crescimento. Atualmente, a *SpringSource* pertence à *VMWare* que é um dos gigantes da área da informática;
- O *Spring Framework* foi desenvolvido desde o início com o objetivo de facilitar a realização de testes às aplicações, o que muito favorece a qualidade e robustez das aplicações.

Apesar das inúmeras vantagens, este *framework* não está isento de aspetos negativos (embora estes sejam em muito menor número). Destacam-se os seguintes:

- Embora seja constituído por módulos independentes e isolados, o *Spring* é uma plataforma muito extensa e complexa (o sistema de ajuda contém milhares de classes[34]). Esta situação pode aumentar consideravelmente a dimensão das aplicações. A dimensão do *framework* também pode ser um problema ao nível da aprendizagem, uma vez que demorará mais tempo até que os programadores tenham um entendimento mais aprofundado sobre a plataforma;
- Nas primeiras versões do *Spring*, as configurações eram todas feitas através de ficheiros XML, o que aumentava o tamanho destes e a sua quantidade. Nas versões mais recentes as configurações passaram a poder ser feitas utilizando as anotações do Java mas, mesmo assim, continua a haver uma grande proliferação de ficheiros XML;
- A introdução de camadas de abstração para as API pode dar origem a que os programadores não as conheçam de forma adequada. Este desconhecimento pode, em determinadas situações, ser um entrave a que os programadores utilizem as API de forma adequada ou que não as consigam utilizar para dar resposta a situações mais específicas em que seja necessário utilizá-las diretamente.

Como se pode verificar pela análise efetuada nesta secção, o *Spring Framework* é constituído por uma grande quantidade de projetos, módulos e bibliotecas, os quais abrangem todas as áreas de uma aplicação empresarial. Por este motivo, o *Spring Framework* pode ser visto não como um só *framework* mas sim como um conjunto de *frameworks* que poderão ser utilizadas à medida das necessidades de cada organização.

[34] http://docs.spring.io/spring/docs/3.2.4.RELEASE/javadoc-api/, Agosto 2013

4.3. PHP

O PHP é uma linguagem de programação criada por Rasmus Lerdorf e vocacionada para o desenvolvimento de aplicações *web*, que pode ser colocada dentro de uma página HTML ou usada de forma independente. Embora se trate de uma linguagem maioritariamente utilizada no lado do servidor (*Server Side Scripting*) também é utilizada na criação de *scripts* que podem ser executados numa linha de comandos ou na criação de interfaces de utilizador através da utilização da extensão PHP-GTK [Lerdorf06].

A primeira versão do PHP surgiu em 1994 com o nome de *PHP Tools* (*Personal Home Page Tools*) e consistia num conjunto de módulos CGI desenvolvidos em C. Esta versão inicial apenas permitia o registo dos acessos a uma determinada página mas, ao longo do tempo, foram sendo adicionadas novas funcionalidades ao projeto. Em Junho de 1995 o código fonte foi disponibilizado à comunidade com o objetivo de aumentar o envolvimento de novos programadores nas tarefas de correção de erros e de melhoramento geral. Em Novembro de 1997 surgiu a versão 2.0, tendo, na altura, sido chamada da PHP/FI (*Personal Home Page/ Forms Interpreter*). Esta nova versão já se aproximava mais daquilo que é, hoje em dia, o PHP deixando de ser, apenas, um conjunto de ferramentas para auxílio na construção de páginas *web* e dando os primeiros passos na criação de uma linguagem de programação.

A versão 3.0 foi lançada em Junho de 1998 e representou um ponto de viragem muito importante no projeto, que passou a ser desenvolvido por um grupo de programadores (Rasmus Lerdorf, Andi Gutmans e Zeev Suraski). Nesta nova versão, o interpretador foi totalmente reescrito para permitir a criação de uma nova linguagem de programação. Para além desta importante funcionalidade, o PHP 3.0 disponibilizava uma interface para comunicação com várias bases de dados e diversas API de uso genérico. Também permitia um elevado nível de extensibilidade, o que atraiu muitos programadores para o projeto e o consequente surgimento de novos módulos que expandiam as funcionalidades da plataforma. A partir daqui, a expressão PHP deixou de significar *Personal Home Page Tools* passando a significar *PHP: Hypertext Preprocessor*. A partir desta nova versão, assistiu-se a um enorme aumento da popularidade da linguagem e do número de servidores onde estava instalada.

Logo após o lançamento do PHP 3.0, começou o trabalho na versão 4.0 com o objetivo de melhorar o desempenho em aplicações complexas e aumentar os níveis de modularidade. Este trabalho culminou com o lançamento da versão 4.0 em Maio de 2000. Os objetivos inicialmente propostos foram alcançados e esta nova versão apresentava um novo interpretador chamado de *Zend Engine* [Lerdorf06]. Para além das melhorias referidas, a versão 4.0 apresentava novas funcionalidades que alavancaram a sua utilização em ambientes empresariais. Uma gestão melhorada dos recursos, diversos melhoramentos ao nível da programação orientada aos objetos, API de encriptação e gestão de sessões HTTP, entre outros, são alguns exemplos das novas funcionalidades disponibilizadas pelo PHP 4.0.

A versão 5.0 foi lançada em Julho de 2004, juntamente com a versão 2.0 do *Zend Engine*. Esta nova versão, para além de melhorar significativamente as bibliotecas/extensões existentes, apresentou novas funcionalidades, destacando-se as relacionadas com o

suporte ao XML e *Web Services*, bem como o suporte nativo para a base de dados *SQLite*. A versão mais recente do PHP é a 5.5.1 tendo sido lançada em Julho de 2013[35]. Desde o lançamento da versão 5.0 até à atual foram implementadas inúmeras funcionalidades (suporte Nativo ao JSON, Expressões Regulares, *Lambdas* e *Closures*) e diversos melhoramentos [Gilmore10].

O PHP é distribuído sobre uma licença própria chamada de *PHP License*. Esta licença é uma das mais permissivas colocando apenas limitações quanto à utilização do nome PHP.

Embora o PHP seja uma linguagem interpretada (sendo o *Zend Engine* o interpretador) existem diversos compiladores que permitem desassociar a linguagem do interpretador e obter um maior desempenho na execução do código. Existem vários compiladores para PHP sendo de destacar os seguintes:

- ***Phalanger*** - Compilador PHP para .NET. Permite converter código escrito em PHP em *assemblies* do .NET[36];
- ***HipHop for* PHP** – Compilador de PHP para C++ desenvolvido dentro do *Facebook* que permitia aumentar em 6 vezes o desempenho de uma aplicação[37];
- **PHC (*The Open Source* PHP *Compiler*)** – Compilador de PHP que permite a geração de executáveis nativos[38].

Outro projeto que merece algum destaque no âmbito do PHP é o *HipHop VM for* PHP[39] o qual está a ser desenvolvido pelo *Facebook*. Apesar do sucesso obtido com o compilador *HipHop for* PHP, este apresentava algumas limitações que foram ultrapassadas com esta nova abordagem. O *HipHop VM for* PHP trata-se de uma máquina virtual que permite a execução de forma muito mais veloz de código desenvolvido em PHP. Num processo semelhante ao que ocorre na plataforma *Java* e .NET, o *HipHop VM for* PHP converte o código PHP num código otimizado e intermédio que, numa fase posterior, será executado pela máquina virtual.

Nos pontos seguintes apresentam-se as principais vantagens reconhecidas ao PHP [Suehring09;MacIntyre10]:

- **Multiplataforma e compatibilidade**. O PHP pode ser executado em praticamente todos os sistemas operativos da atualidade (*Unix*, *Linux*, *Windows, Mac OS, ou OpenBSD*), sendo compatível com os servidores *web* mais utilizados (*Apache HTTP Server* e *Microsoft Internet Information Services*). À lista dos servidores *web*

[35] Apesar da versão mais recente do PHP ser a 5.5.1 encontra-se, com muita frequência, referências ao PHP 6.0 [Suehring09]. Esta nova versão do PHP estava planeada para, entre outras funcionalidades, disponibilizar suporte nativo ao Unicode. Contudo, em Março de 2010, e em virtude do surgimento de dificuldades no processo de desenvolvimento, os programadores decidiram focar a sua atenção no desenvolvimento das versões 5.x tendo muitas das funcionalidades que estavam previstas para a versão 6.0 sido incluídas nas versões do ramo 5.x. Por este motivo, e até indicações em contrário por parte da equipa de desenvolvimento do PHP, a versão 6.0 ainda não está disponível para utilização.
[36] *Phalanger – the PHP compiler for .NET*, http://www.php-compiler.net/, Junho 2013
[37] *HipHop for PHP*, http://www.hiphop-for-php.com/wiki/Main_Page, Junho 2013
[38] *The Open Source PHP Compiler*, http://phpcompiler.org/, Junho 2013
[39] *HipHop VM for PHP*, https://github.com/facebook/hiphop-php#readme, Junho 2013

compatíveis pode-se adicionar, ainda, outros servidores menos conhecidos tais como o *Netscape Enterprise Server* ou o *AOL Server*;

- **Licenciamento e custos**. O PHP é disponibilizado livre de qualquer custo e sem qualquer limitação quanto à sua utilização, modificação ou redistribuição (havendo apenas uma pequena ressalva quanto ao nome a atribuir a produtos relacionados com o PHP). Mesmo ao nível das ferramentas de desenvolvimento, a grande maioria é disponibilizada sem qualquer custo ou restrição. Comparativamente a outras plataformas ou linguagens de programação, o PHP apresenta custos muito inferiores (ou inexistência de custos) tanto ao nível da linguagem, servidor e ferramentas de desenvolvimento;
- **Bibliotecas e extensões**. O PHP disponibiliza mais de 200 bibliotecas que estão à disposição dos programadores e que permitem expandir as funcionalidades da plataforma. A estas bibliotecas podem juntar-se muitas extensões desenvolvidas por terceiros que adicionam ainda mais funcionalidades. Interfaces de ligação às bases de dados mais utilizadas, suporte aos principais protocolos e *standards*, geração de ficheiros PDF, tratamento de expressões regulares, suporte às tecnologias COM e CORBA, interfaces de ligação aos maiores sistemas de pagamento eletrónico são alguns exemplos das funcionalidades disponibilizadas pelo PHP. O PHP *Extension and Application Repository* e o PHP *Extension Community Library* são repositórios que armazenam grande parte das extensões e do código desenvolvido por terceiros, os quais estão disponíveis para utilização;
- **Facilidade de utilização e de aprendizagem**. O PHP torna extremamente fácil e rápido o desenvolvimento de aplicações *web*. Ao contrário de outras plataformas, não é necessário possuir um conhecimento muito aprofundado da tecnologia para se poder trabalhar com ela. Para além da simplicidade intrínseca, a plataforma disponibiliza muito código pré-definido que aumenta substancialmente a facilidade de utilização e a produtividade dos programadores;
- **Estabilidade**. O conceito de estabilidade do PHP pode ser utilizado em contextos distintos: o sistema não necessitar de ser reiniciado com frequência; a plataforma responde de forma eficaz e eficiente às cargas de trabalho a que está sujeita, não bloqueando frequentemente; o *software* não introduz alterações drásticas em cada nova versão, as quais podem gerar erros e quebras no funcionamento. Acresce à estabilidade do PHP a estabilidade do servidor *web Apache* e do sistema operativo *Linux*. Como a configuração mais usual da utilização do PHP é num cenário PHP/*Apache Web Server*/*Linux*, estamos perante uma das plataformas mais estáveis e robustas que se podem encontrar;
- **Popularidade**. A popularidade do PHP pode ser vista tanto ao nível da utilização como ao nível da comunidade de programadores. Tanto num caso como no outro estamos perante níveis de popularidade muito elevados que atestam o sucesso da plataforma. Quanto à utilização, diversos sistemas de medição reportam que o PHP é a linguagem mais utilizada na Internet[40]. Relativamente à comunidade de desenvolvimento em redor do PHP, a sua dimensão e envolvimento são um garante da continuidade da evolução da linguagem. Embora o número de

[40] http://w3techs.com/technologies/overview/programming_language/all, Agosto 2013
http://trends.builtwith.com/framework, Agosto 2013

programadores que trabalham no núcleo seja relativamente pequeno (uma consulta ao site *php.net* permite identificar 10 programadores), a lista de participantes nas restantes áreas é extremamente longa, o que pode ser confirmado pela quantidade de projetos relacionados com o PHP;

- ***Software* não proprietário**. O PHP não está associado a nenhuma empresa em particular, o que lhe confere uma grande flexibilidade e o liberta dos aspetos negativos e restrições associadas ao *software* e *standards* proprietários. Ao contrário do ASP.NET que está intimamente relacionado com um sistema operativo ou da plataforma *Java* que está muito ligada à *Oracle*, o PHP não está sujeito a estas restrições típicas dos sistemas proprietários. Esta independência do PHP, que também se estende a outras áreas como, por exemplo, *browsers* ou bases de dados, funcionou no passado e funciona atualmente como uma alavanca para o seu crescimento e evolução.

Apesar das vantagens apresentadas na secção anterior, o PHP é alvo de bastantes críticas e são-lhe apontadas, entre outras, as seguintes desvantagens [MacIntyre10]:

- **Linguagem dinamicamente tipada**. Embora as linguagens de tipagem dinâmica (ou fracamente tipadas) possam simplificar a programação e ser úteis em situações específicas, as desvantagens desta caraterística ultrapassam largamente os aspetos positivos. O facto de não se indicar o tipo de dados que uma variável irá armazenar pode ser visto como uma situação que pode aumentar a probabilidade de ocorrência de *bugs*, embora a realização de testes unitários possa minimizar estes efeitos. É frequente encontrarem-se argumentos no sentido contrário, defendendo que a tipagem dinâmica deve ser vista como uma vantagem;
- **Pouca coerência nos nomes e na ordem dos parâmetros**. Como se trata de um projeto colaborativo no qual participam centenas de programadores, existem várias inconsistências nos nomes atribuídos a funções ou objetos. É frequente encontrarem-se nomes de funções que seguem convenções totalmente diferentes (por exemplo, `find_value` e `findvalue`). Outro exemplo muito comum de inconsistência verifica-se nas funções `strstr($haystack, $needle, …)` e `array_search($needle, $haystack, …)`. Para além da utilização de convenções diferentes na definição do nome, os parâmetros que estas funções recebem não estão na mesma ordem. Apesar dos esforços feitos no sentido de normalizar as nomenclaturas, ainda subsistem muitas incoerências no PHP;
- **Integração do código PHP com código HTML**. Numa primeira análise, a possibilidade de se integrar o código PHP com o HTML (através da utilização de *tags* específicas) aparenta ser um aspeto positivo. No entanto, este processo tende a dificultar a leitura e a manutenção do código, especialmente em projetos de grande dimensão, para além de impossibilitar a implementação de boas práticas do desenvolvimento de *software*;
- **Ausência de conceitos para organização do código-fonte (*namespaces* ou *packages*)**. A ausência de *namespaces* ou *packages* dificulta a organização do código, principalmente em aplicações grandes.

- **Grande propensão a ataques por *SQL Injection***. As bibliotecas de interação com as bases de dados estão muito sujeitas a ataques de *SQL Injection*. Apesar deste tipo de ataques também afetar outras plataformas, a forma como estão desenvolvidas as API do PHP tende a aumentar este risco.

As desvantagens acima referidas fazem-se sentir, com maior intensidade, em projetos de grande dimensão (apesar do PHP ser muito utilizado em projetos pequenos, não significa que não seja utilizado em aplicações de maior dimensão). Para minimizar estas desvantagens e facilitar o desenvolvimento das aplicações, têm surgido vários *frameworks* para a linguagem PHP. Ao contrário do que acontece noutras plataformas, em que cada *framework* apresenta uma abordagem totalmente diferente, os *frameworks* PHP caraterizam-se por não apresentarem diferenças significativas entre si. As principais caraterísticas diferenciadoras dos *frameworks* que a seguir serão apresentados resumem-se, essencialmente, à comunidade que os sustenta e às bibliotecas/extensões que disponibilizam. Ao nível da arquitetura, todas elas apresentam soluções bastante semelhantes.

4.3.1. ZEND FRAMEWORK 2

O *Zend Framework 2* (ZF2) é um *framework* de código aberto vocacionado para o desenvolvimento de aplicações *web* utilizando a linguagem PHP. É desenvolvido pela *Zend Technologies*, em conjunto com programadores independentes, e disponibilizado sob a licença *New* BSD *License* [Allen09]. A primeira versão (*Zend Framework* 1.0.0) foi oficialmente disponibilizada em Junho de 2007, tendo sido, desde essa data, lançadas diversas atualizações. O *Zend Framework 2* foi lançado em Setembro de 2012, sendo que, atualmente, a versão mais recente disponível para *download* é a 2.2.1 (disponibilizada em Junho de 2013).[41]

A arquitetura do ZF2 baseia-se no padrão *Model-View-Controller* sendo também implementado o padrão *Front Controller* uma vez que todos os pedidos efetuados pelos clientes são canalizados para um único componente de entrada. Na figura 4.8 apresenta-se a arquitetura simplificada do ZF2.

Na arquitetura do *Zend Framework 2* identificam-se cinco áreas distintas que, a seguir, serão analisadas: o *Router* e o *Dispatcher* (que implementam o padrão *Front Controller*) e o *Model*, *View* e *Controller* (que implementam o padrão MVC). O *Router* e o *Dispatcher* trabalham em conjunto, sendo o ponto de entrada centralizado dos pedidos. Ao receberem um pedido do cliente, o *Router* é responsável por identificar qual a ação que deve ser executada e, de seguida, o *Dispatcher* inicia a execução da ação previamente selecionada e de outras que, eventualmente, venham a ser necessárias. Este processo de seleção/execução das ações é chamado de *Routing/Dispatching*.

[41] http://framework.zend.com/downloads/archives, Junho 2013

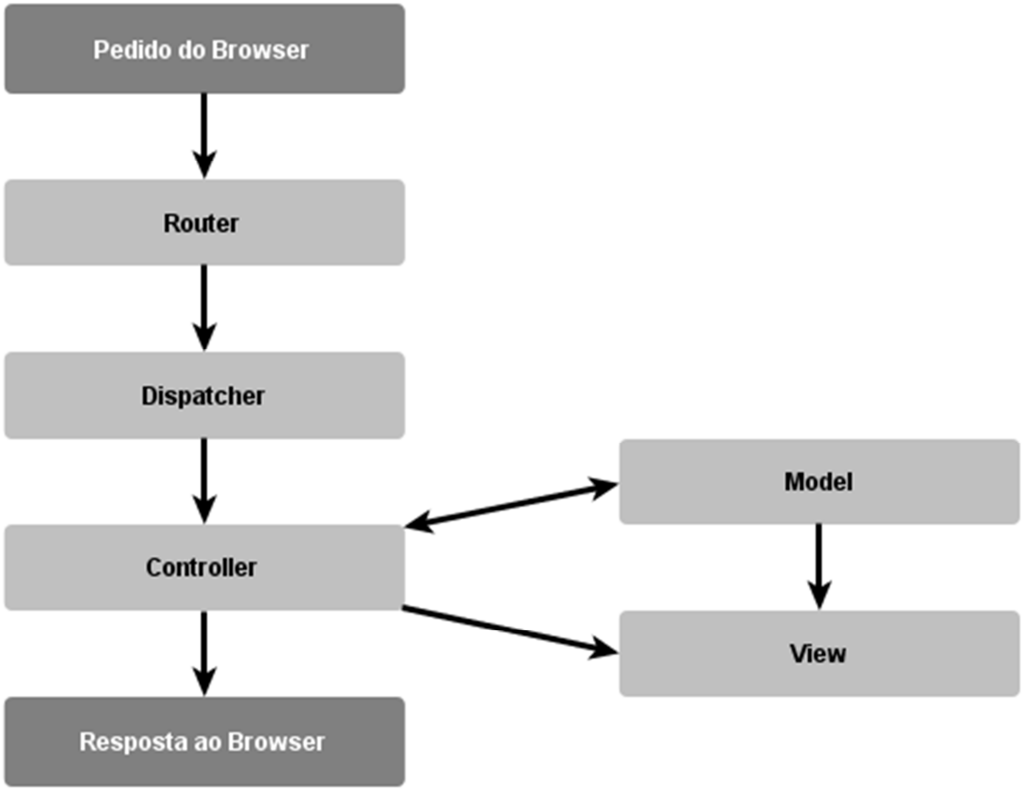


Figura 4.8 – Arquitetura do *Zend Framework 2*[42]

O *Model* é a área que contém todo o código que implementa as regras de negócio da aplicação assim como o código de interação com a base de dados.

O *View* é a área responsável pela apresentação da lógica da aplicação. Geralmente, a lógica da apresentação, no âmbito de uma aplicação *web*, consiste em código HTML mas pode incluir outras operações como, por exemplo, a geração de um ficheiro XML ou CSV para exportação são funções que se enquadram nas responsabilidades do *View*. Os ficheiros associados ao *View* costumam conter código que permite a apresentação da informação gerada pelo *Model*, sendo chamados de *Templates* ou *Scripts*. Em aplicações mais complexas esse código pode ser deslocado para funções (*View Helpers*), tendo em vista uma melhor organização e o aumento da reutilização do código.

Finalmente, o *Controller* pode ser visto como a área que coordena as restantes áreas em resposta aos pedidos efetuados pelos clientes. O sistema de controlo do ZF2 baseia-se no padrão *Front Controller* onde todos os pedidos passam por um componente único o qual invocará as ações adequadas para preparar a resposta.

Cada uma das áreas referidas nos parágrafos anteriores é implementada por vários componentes. São estes componentes que constituem a base do ZF2 e que disponibilizam as funcionalidades mais comuns e necessárias para o desenvolvimento de aplicações *web* empresariais. Apesar do número elevado de componentes, o ZF2 foi construído de forma flexível o que permite que apenas sejam utilizados os componentes que efetivamente

[42] Adaptado de [Allen09]

sejam necessários. Cada um dos componentes apresentados na tabela 4.2 é implementado por uma ou várias classes. Por exemplo, o componente *Zend_Config* contém, entre outras, as classes *Zend_Config_XML* (para ler configurações armazenadas em ficheiros XML) e *Zend_Config_Ini* (para ler configurações contidas em ficheiros INI). Os principais componentes do *framework* podem ser agrupados em 6 categorias, as quais são apresentadas na tabela 4.2.

MVC	**Authentication and Access**
Zend_Controller Zend_Layout Zend_View	Zend_Acl Zend_Auth Zend_Session Zend_OpenId Zend_InfoCard
Internationalization	**Interapplication Communication**
Zend_Currency Zend_Date Zend_Locale Zend_View Zend_Translate	Zend_Http_Client Zend_Json Zend_Ldap Zend_RRest Zend_TimeSync Zend_XmlRpc
Web Services	**Core**
Zend_Feed Zend_GData Zend_Service_*	Zend_Db Zend_Cache Zend_Config Zend_Filter Zend_Form Zend_Log Zend_Mail Zend_Memory Zend_Pdf Zend_Registry Zend_Search Zend_Uri Zend_Validate

Tabela 4.2 – Principais componentes do *Zend Framework 2*[43]

Os componentes MVC permitem a implementação de parte do padrão MVC no *framework*. O *Controller* é implementado pelas classes do componente *Zend_Controller* enquanto que as funções de visualização são implementadas pelas classes dos componentes *Zend_Layout* e *Zend_View*. O *Model* é implementado pelas classes que compõem os componentes *Zend_Db* e *Zend_Service*. As classes do *Zend_Controller* implementam o padrão *Front Controller* permitindo uma centralização do processamento dos pedidos. Apesar desta centralização e automatismo, o programador pode interferir no processo alterando-o para dar resposta às especificidades da aplicação. O componente *Zend_Db* juntamente com o

[43] [Allen09]

componente *Zend_Services* formam a base do *Model*, permitindo a interação com as principais bases de dados atualmente disponíveis e o consumo de *Web Services*. Os componentes MVC, trabalhando em conjunto com outros componentes *Core* (por exemplo, o componente *Zend_Filter* para filtragem, controlo e transformação da informação e o *Zend_Config* para leitura e gravação de informações de configuração) formam o núcleo de uma aplicação desenvolvida no *Zend Framework 2.*

Os *Authentication and Access Components* são utilizados no processo de autenticação e de controlo de acesso dos utilizadores à aplicação. Por um lado, permitem controlar quem pode aceder à aplicação e, uma vez o acesso dado, quais são as opções para as quais tem acesso e quais as operações que podem ser realizadas. Estes componentes disponibilizam diversas formas de realização do processo de autenticação e controlo (p. ex. HTTP *Digest*, *OpenID*, LDAP ou registo em base de dados) mas caso nenhum deles seja satisfatório, o utilizador poderá sempre desenvolver novos mecanismos de autenticação e controlo.

O ZF2 disponibiliza vários componentes para tratamento das questões relacionadas com a internacionalização das aplicações. Questões com o símbolo da unidade monetária, o formato da data e a seleção do idioma adequado são tratadas de forma muito simples pelos *Internationalization Components*.

No *grupo Interapplication Communication* constam os componentes que facilitam o desenvolvimento de funcionalidades de comunicação com outras aplicações, quer sejam aplicações de *desktop*, *web* ou serviços. Os componentes deste grupo suportam os mais variados protocolos e formatos para comunicação inter-aplicação, tais como o JSON e XML-RPC.

Os componentes da área *Web Services Components* estão relacionados com o consumo e disponibilização de *web services* de uma forma extremamente simples e normalizada. Adicionalmente, também disponibilizam o acesso às mais variadas API de empresas como a *Google*, *Amazon* ou *Yahoo*. Também é nesta área que se encontram os componentes para a gestão dos RSS.

Na área dos *Core Components* foram colocados todos os restantes componentes que não se enquadram nas outras áreas. Componentes para envio de correio eletrónico, geração de documentos no formato PDF, pesquisas e *cache* de dados, são alguns exemplos das funcionalidades disponibilizadas pelos componentes desta secção.

O ZF2 possui um elevado número de componentes que possibilitam a implementação, de forma rápida e simples, das funcionalidades exigíveis a uma aplicação *web*. Embora extensa, a lista de componentes está em constante evolução, tanto ao nível da qualidade e fiabilidade como ao nível da adição de novos componentes.

Na lista seguinte apresentam-se as vantagens mais relevantes do ZF2:

- **Componentes independentes.** O *Zend Framework 2* é composto por vários componentes que cobrem a grande maioria dos elementos necessários para desenvolver uma aplicação *web* empresarial. Estes componentes caraterizam-se por serem independentes uns dos outros. Com esta independência, os programadores apenas precisam de utilizar os componentes que, efetivamente,

necessitam, tornando a aplicação mais simples e flexível. Embora o ZF2 seja constituído por muitos componentes, também permite que sejam utilizadas outras bibliotecas independentes, apresentando, ainda, elevados níveis de integração com outros *frameworks* PHP;

- **Desenho e implementação moderna.** O ZF2 foi desenvolvido usando a versão 5 do PHP e utilizando diversos conceitos e boas práticas de programação. Esta arquitetura permite uma elevada flexibilidade no *framework* e nas aplicações desenvolvidas;
- **Fácil de aprender.** Apesar de se tratar de um *framework* muito extenso, a sua modularidade facilita muito a aprendizagem. Os programadores não têm que conhecer todo o *framework* para poderem começar a trabalhar com ele de forma eficiente. É possível conhecer e utilizar os componentes de forma individual e ir expandindo o conhecimento à medida que vai sendo necessário utilizar outros componentes;
- **Documentação.** Esta vantagem refere-se não só à documentação existente sobre o próprio *framework* (que é abundante e está atualizada) mas também à facilidade com que os programadores podem criar documentação dos projetos usando marcadores especiais no próprio código;
- **Desenvolvimento simples e rápido.** O ZF2 simplifica muito o desenvolvimento de aplicações *web* deixando ao programador apenas as tarefas relacionadas com a implementação das regras de negócio. Todo o código que não seja específico da aplicação fica a cargo do ZF2, sendo este código muito robusto e eficiente. Por outro lado, a forma como o *framework* está desenvolvido favorece a realização de testes, a manutenção e a adição de novas funcionalidades às aplicações;
- **Comunidade de desenvolvimento.** O desenvolvimento do *framework* é feito por uma empresa chamada *Zend Technologies* em conjunto com uma grande comunidade de programadores externos. A *Zend Technologies* foi fundada por dois dos mais importantes e ativos programadores do PHP: Andi Gutmans e Zeev Suraski. Numa perspetiva, o ZF2 tem o apoio e suporte de uma empresa com grande *know-how* técnico. Por outro lado também envolve uma comunidade grande de programadores, o que garante não só a continuidade do projeto como a disponibilidade e intervenção de um elevado conhecimento técnico;
- ***Zend Studio* e *Zend Server*.** A *Zend Technologies* disponibiliza uma ferramenta de desenvolvimento chamada *Zend Studio*. Este IDE está otimizado para o desenvolvimento de aplicações PHP e, em especial, aplicações PHP baseadas no *Zend Framework 2*. Contudo, isto não significa que não seja possível, ou que seja muito difícil, desenvolver aplicações com outros IDE. Trata-se, apenas, de uma ferramenta de desenvolvimento que facilita, ainda mais, o trabalho dos programadores. A *Zend Technologies* também disponibiliza o *Zend Server* que se trata de um servidor de aplicações *web* especialmente vocacionado para as aplicações desenvolvidas em PHP (preferencialmente usando o ZF2) e que necessitem de altos níveis de segurança, performance e fiabilidade. O servidor pode ser executado em *Windows*, *Linux* e *Mac OS X* sendo disponibilizado em várias edições (*Free Edition*, *Small Business Edition*, *Professional Edition* e *Enterprise Edition*). Com este conjunto (*Zend Server*, PHP e *Zend Framework*), normalmente chamado de *Zend Stack*, os administradores de sistemas passam a dispor de um

conjunto de ferramentas que garantem um excelente nível de desempenho, segurança, fiabilidade e escalabilidade.

Apesar das diversas vantagens que o tornam o *framework* num dos mais utilizados na linguagem PHP, o ZF2 também apresenta algumas desvantagens que são listadas a seguir:

- Não está otimizado para o desenvolvimento de pequenas aplicações. Apesar do ZF2 apresentar claras vantagens ao nível do tempo de desenvolvimento, estas apenas se fazem sentir em projetos de grande dimensão. Para projetos pequenos será preferível optar por outros ou pelo desenvolvimento sem o auxílio de qualquer *framework*;
- Ao contrário de outros *frameworks* PHP não inclui nenhum ORM nativo. Esta caraterística nem sempre poderá ser vista como um aspeto negativo uma vez que é sempre possível utilizar um dos vários ORM disponíveis no mercado;
- O ZF2 implementa padrões e práticas de programação que podem dificultar o desenvolvimento caso se opte por fazer algo de forma diferente. O desconhecimento de conceitos de OOP ou dos padrões MVC e *Front Controller* podem ser um entrave a uma correta utilização do *framework*. Não se tratando, com efeito, de uma desvantagem direta do ZF2, a forma como está arquitetado pode potenciar as lacunas dos programadores;
- Apesar de ser possível excluir os componentes não utilizados, a experiência revela que o mais comum é os programadores não excluírem os componentes não utilizados, aumentando, sem qualquer necessidade, a dimensão das aplicações.

4.3.2. CAKEPHP

O *CakePHP* é um *framework* PHP de código livre para desenvolvimento rápido de aplicações, sendo disponibilizado sob a licença MIT *License*[44] [Bari08]. A primeira versão do *framework* foi lançada em Maio de 2006, tendo a versão 2.0.0 surgido em Outubro de 2011. A versão mais recente é a 2.3.7, disponibilizada em Julho de 2013. A estrutura do *CakePHP* foi implementada seguindo o padrão MVC, embora também seja identificado o padrão *Front Controller*. Na figura 4.9 apresenta-se um esquema da arquitetura do *CakePHP*.

O cliente efetua um pedido, o qual é recebido pelo *Dispatcher*. Este, analisando a estrutura do URL, determina qual o *Controller* que deverá ser executado, reencaminhando-lhe o pedido juntamente com mais alguma informação que, eventualmente, possa estar associada. Caso seja necessário o *Controller* obter dados adicionais para processar o pedido, estes terão que ser solicitados ao *Model*. É da responsabilidade do *Model* a interação com as bases de dados para obtenção dos dados necessários ao processamento do pedido. Depois do *Model* concluir o seu trabalho, o controlo da operação é devolvido ao

[44] *CakePHP: the rapid development php framework*, http://cakephp.org/, Junho 2013

Controller, o qual, mediante o resultado obtido, selecionará o *View* adequado ou poderá voltar a interagir com o *Model* para a obtenção de mais dados. O *View* gerará a vista apropriada à informação recebida e enviá-la-á para o *browser* como resposta ao pedido efetuado.

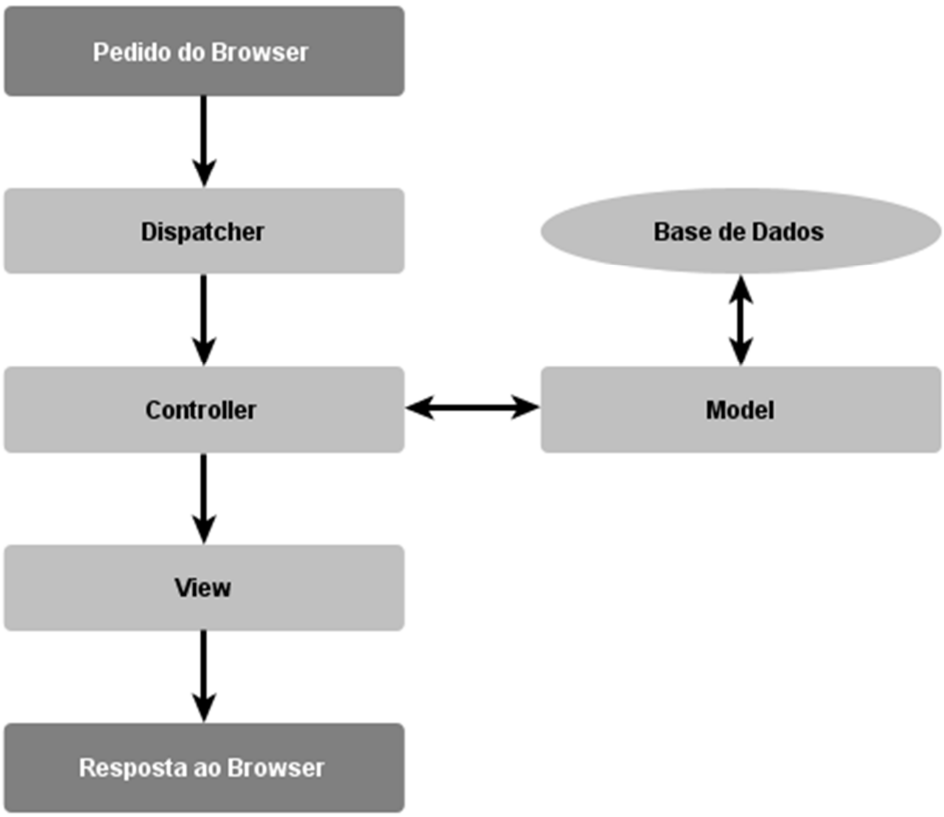


Figura 4.9 – Arquitetura do *CakePHP*[45]

No *framework CakePHP* os *Controllers* têm como função controlar o fluxo da aplicação. Com base num pedido que recebem, passam o controlo da aplicação para o *Model* e, quando o ciclo de interação com este componente estiver concluído, passarão o controlo para o *View*. Ao nível do *Model*, o *CakePHP* apresenta uma diferença substancial em relação a outras implementações do padrão MVC uma vez que neste *framework* um objeto do *Model* representa, apenas, uma tabela na base de dados. Assim, cada tabela da base de dados que seja usada na aplicação terá o seu respetivo objeto no *Model* onde constará todo o código PHP usado para inserir, alterar, eliminar e consultar registos, bem como o código para definição e verificação da integridade referencial e de validação dos conteúdos. Segundo as regras definidas pelo padrão MVC, é no *Model* que constam todas as regras de negócio da aplicação. Finalmente, o *View* é o componente responsável pela elaboração da resposta ao pedido efetuado pelo cliente. A resposta pode ser em código HTML – o mais usual – mas também pode ser gerada noutro formato qualquer. Na tabela 4.3 apresentam-se as principais classes disponibilizadas pelo *framework*.

[45] Adaptado de [Golding08]

General Purpose	Behaviors
Global Constants and Functions App Class Events System Collections	ACL Containable Translate Tree
Components	**Utilities**
Access Control Lists Authentication Cookie EmailComponent Request Handling Pagination Security Sessions	Caching CakeEmail CakeNumber CakeTime Data Sanitization Folder & File HttpSocket Inflector Internationalization & Localization Logging Router Security Hash Set String Xml
Helpers	
CacheHelper FormHelper HtmlHelper JsHelper NumberHelper Paginator RSS SessionHelper TextHelper TimeHelper	

Tabela 4.3 – Classes e *helpers* do *CakePHP*[46]

As bibliotecas *General Purpose* são bibliotecas que podem ser usadas ao longo de toda a aplicação e que não se enquadram numa das restantes classificações. No *Global Constants and Function* agrupam-se todas as funções genéricas e valores constantes. O *Events System* desempenha um papel muito importante no *CakePHP* porque se trata de uma forma simples de implementar o padrão *Observer*. Com esta biblioteca é possível emitir notificações (eventos) numa área da aplicação e processá-los noutra diferente sem haver uma ligação direta entre quem emite o evento e quem o processa.

As bibliotecas *Behaviors* são usadas, geralmente, na camada *Model* das aplicações, permitindo a separação, reutilização e expansão dos *models*. O *CakePHP* disponibiliza classes *Behaviors* para tratamento de estruturas em árvore, *Access Control Lists*, listas de elementos e traduções. Para além dos *behaviors* disponibilizados pelo *CakePHP* a todo o momento são criados novos pela comunidade. Os *Components* podem ser vistos como funcionalidades que estão disponíveis para serem utilizados pelos *controllers*. A utilização dos *Components* ajuda a manter o código dos *Controllers* mais simples, e facilita a sua reutilização. O *CakePHP* fornece um grande conjunto de *Components* destacando-se os relacionados com os *Emails*, *Pagination*, *Cookies* e *Sessions*.

[46] http://book.cakephp.org/2.0/en/core-libraries.html, Junho 2013

Os *Helpers* são componentes que podem ser utilizados ao longo de toda a aplicação, embora, na prática, surjam mais vezes relacionados com os aspetos visuais. Com os *Helpers*, os programadores poderão desenvolver mais fácil e rapidamente as camadas visuais das aplicações uma vez que estes componentes têm um papel importante na criação do código HTML. Por exemplo, o componente *FormHelper* é muito utilizado na criação de formulários porque permite que os formulários sejam criados de uma forma muito rápida, através da automatização das questões relacionadas com a validação, posicionamento dos campos, obtenção dos valores, entre outros.

Para além dos componentes referidos nos parágrafos anteriores, o *CakePHP* coloca à disposição dos programadores diversas classes utilitárias que facilitam as tarefas de desenvolvimento do *software*. Enquadram-se nesta categoria, entre outras, classes para tratamento das questões relacionadas com o *caching* (p. ex. *caching* de dados e páginas), registo de operações, internacionalização e localização, envio de *e-mails* e manutenção do sistema de ficheiros.

Na lista seguinte são apresentadas as principais caraterísticas e vantagens que, normalmente, estão associadas ao *CakePHP* [Golding08; Bari08]:

- **Desenvolvimento rápido e simples.** O *CakePHP* facilita o desenvolvimento dos projetos, especialmente na fase inicial, através da disponibilização de uma estrutura-base de código já devidamente organizada. A inclusão de vários padrões de desenvolvimento resolve os problemas mais usuais no desenvolvimento de aplicações *web*, permitindo que os programadores se concentrem mais no código da aplicação propriamente dita, reduzindo os tempos de desenvolvimento. A possibilidade de se poderem definir, rapidamente, os fluxos ou sequências das aplicações, principalmente nos aspetos visuais, melhora os níveis de comunicação com os futuros utilizadores o que, regra geral, se traduz em menores tempos de desenvolvimento;
- **Convenção em detrimento da configuração.** Embora sejam muito importantes para o funcionamento de um sistema, as configurações também podem tornar-se numa fonte de problemas e complicações. No *CakePHP* as configurações são reduzidas ao mínimo indispensável ao ponto de, na maior parte dos casos, apenas ser necessário indicar qual a base de dados a utilizar. O *CakePHP* baseia-se mais em convenções, ou seja, caso determinados requisitos sejam observados, o sistema estará automaticamente configurado. Por exemplo, se os nomes das tabelas e campos respeitarem as convenções do *CakePHP*, estes serão automaticamente utilizados. O mesmo também se aplica aos nomes dos *Controllers*, *Models* e *Views*. Se respeitarem uma nomenclatura definida pelo *CakePHP* serão utilizados sem qualquer configuração adicional;
- ***Application Scaffolding.*** Esta funcionalidade permite criar, sem qualquer intervenção por parte do programador, toda a parte visual da aplicação com base na estrutura de uma base de dados. O *framework* analisa a estrutura da base de dados e tenta gerar o código HTML que melhor se lhe adequa. Embora o código HTML gerado não deva ser usado de forma direta nas aplicações, esta técnica permite que se testem, numa fase inicial, diversos aspetos das aplicações sem ser necessário investir muito tempo na construção das *views*;

- **Geração automática de código.** O *CakePHP* disponibiliza um gerador automático de código, o qual pode ser executado a partir da linha de comandos, chamado de *Bake Script*. Para se usufruir desta funcionalidade apenas será necessário indicar quais as tabelas para as quais se pretende gerar o código. Após a indicação das tabelas, o *Bake Script* gerará todo o código necessário para inserir, alterar, eliminar e consultar registos;
- **Funcionalidades "prontas a utilizar".** O *CakePHP* disponibiliza, de raiz, diversas funcionalidades que visam facilitar o trabalho dos programadores. A lista é muito extensa destacando-se, no entanto, as seguintes: um mecanismo simplificado e automático para validação dos dados introduzidos pelos utilizadores. O programador apenas terá que indicar o tipo de dados e o *framework* fará toda a gestão e validação. Com o *Access Control List* torna-se extremamente fácil controlar o acesso a cada uma das opções das aplicações. Em função do tipo de utilizador que aceder à aplicação poderemos indicar, facilmente, a que funcionalidades terá acesso e quais as que lhe estarão vedadas. O *Custom Page Layout* facilita a definição de um *layout* uniforme ao longo de toda a aplicação. Colocando a definição do *layout* na pasta apropriada, essa formatação será definida para todas as páginas da aplicação. Com o *Ajax Helper* torna-se muito simples adicionar funcionalidades AJAX à aplicação. Para além das funcionalidades referidas, existem muitas mais que poderão ser usadas e à quais se juntam outras que, constantemente, estão a ser adicionadas ao *framework*.

Quanto às desvantagens apontadas ao *CakePHP* destacam-se as seguintes:

- **Muito focado em aplicações CRUD**. O *CakePHP* foi desenvolvido tendo em vista as aplicações CRUD (aplicações cujas principais funcionalidades são a criação, consulta, alteração e eliminação de registos) estando perfeitamente adaptado para este tipo de aplicação e facilitando ao máximo o seu desenvolvimento. O aspeto negativo desta abordagem é que caso se pretenda desenvolver outro tipo de aplicações, o *CakePHP* poderá não ser a solução mais adequada;
- **Dificuldade em implementar soluções que não sigam os princípios do *framework*.** No desenvolvimento do *framework* implementaram-se diversos padrões de desenho de *software*, devidamente comprovados e testados. A forma como o *CakePHP* foi desenvolvido, assim como a sua arquitetura, auxiliam muito o programador no desenvolvimento do código com qualidade e bem estruturado. No entanto, também são um entrave caso se pretenda desenvolver uma funcionalidade sem seguir as regras e os padrões do *CakePHP*;
- **Documentação em qualidade e quantidade inferior**. Comparativamente com outros *frameworks* PHP, a documentação do *CakePHP* não é muito abundante nem está muito bem organizada;
- **Poucos programadores participam no projeto**. Embora a comunidade ligada ao *CakePHP* seja bastante extensa e ativa (uma consulta ao fórum permite confirmar este facto) o núcleo de programadores que participam no desenvolvimento do *framework* é bastante reduzido e só quem pertence a esse grupo é que pode participar no desenvolvimento do *CakePHP*. Isto traduz-se numa evolução mais

lenta, o que o pode colocar em desvantagem relativamente a outros *frameworks* onde a evolução se processa a um ritmo mais elevado.

4.3.3. CODEIGNITER

O *CodeIgniter* é um *framework* PHP, desenvolvido pela empresa *EllisLab, Inc*., vocacionado para o desenvolvimento rápido de aplicações *web*, de código aberto e disponibilizado sob uma licença adaptada da *Apache*/BSD (a qual permite uma utilização e modificação do *framework*, praticamente, sem qualquer tipo de restrição) [Upton07]. A primeira versão surgiu em Fevereiro de 2006 sendo a versão mais recente a 2.1.4 lançada em Julho de 2013[47]. Desde o seu lançamento inicial já foram disponibilizadas mais de 30 atualizações, merecendo especial destaque a versão 2.0.0 (em Janeiro de 2011) a qual deixou de suportar a versão 4.0 do PHP. O *CodeIgniter* carateriza-se por ser um *framework* muito rápido, simples e que exige poucos recursos do sistema, permitindo que os programadores desenvolvam aplicações num curto espaço de tempo. Na base desta simplicidade e desempenho está um conjunto muito pequeno de bibliotecas, que formam o núcleo do *framework*, e às quais se poderão adicionar outras bibliotecas, à medida que forem sendo necessárias. O desenvolvimento de aplicações no *CodeIgniter* torna-se bastante rápido porque são disponibilizadas muitas bibliotecas que tratam dos aspetos mais comuns das aplicações *web* deixando para o programador o desenvolvimento das funcionalidades específicas dos projetos [Griffiths10].

A arquitetura do *CodeIgniter* baseia-se no padrão MVC embora, ao contrário dos restantes *frameworks* PHP, a implementação do *Model* seja opcional. Se, num determinado projeto, a equipa de desenvolvimento entender que a utilização do *Model* acarreta mais desvantagens do que benefícios poderá optar por não o utilizar sem que esta decisão represente qualquer tipo de problema ou limitação. Com efeito, o *framework* disponibiliza uma biblioteca chamada *Database* que pode ser utilizada no *Controller*, possibilitando a não utilização do *Model* sem comprometer a organização do código. O *CodeIgniter* foi desenvolvido de forma flexível e, por este motivo, permite que seja utilizado código que não tenha sido desenvolvido especificamente para o *framework*. Desta forma, permite-se que o programador trabalhe da forma que lhe parecer mais conveniente e não seja limitado por restrições impostas pelo *framework*. O *CodeIgniter* foi desenvolvido com o objetivo de se disponibilizar um *framework* extremamente rápido, com muitas funcionalidades, flexível e, em simultâneo, disponível num formato pequeno e exigindo poucos recursos. Estes objetivos foram alcançados com base nos seguintes princípios:[48]

- **Instanciação dinâmica.** Os componentes são carregados na memória e executados apenas quando são, efetivamente, necessários. Nenhum componente será previamente inicializado no pressuposto de que possa (ou não) vir a ser utilizado no futuro. Com esta regra, consegue-se um sistema muito mais eficiente e menos exigente em termos de recursos;

[47] http://ellislab.com/codeigniter/user-guide/changelog.html, Julho 2013
[48] http://ellislab.com/codeigniter/user-guide/overview/goals.html, Julho 2013

- **Independência dos componentes.** A instanciação dinâmica será tanto mais eficiente quanto maior for a independência dos objetos entre si. A flexibilidade e a capacidade de reutilização do código também é positivamente afetada pela independência dos componentes;
- **Função única.** Cada componente apenas deve ter uma função ou, caso esta regra não seja possível de implementar, deverá ter o menor número possível. Quanto mais restrito for o número de funcionalidades que cada componente desempenha, mais fácil se torna expandir o *framework* e mais flexível este se torna.

Na figura 4.10 apresenta-se um esquema simplificado da arquitetura do *CodeIgniter*.

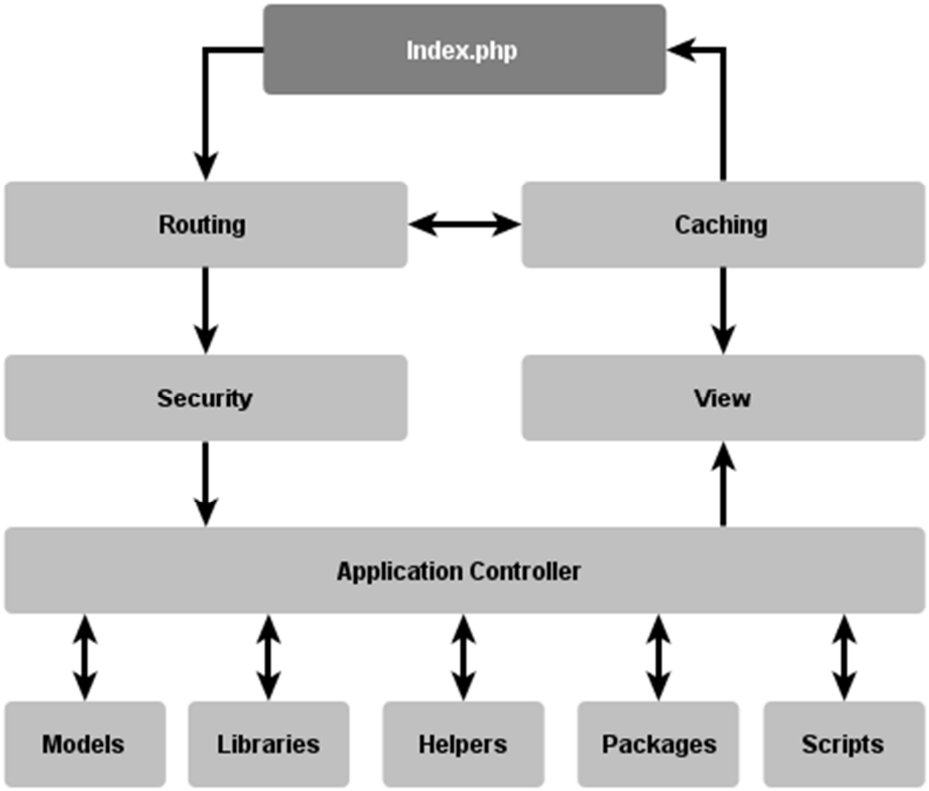


Figura 4.10 – Arquitetura do *CodeIgniter*[49]

O componente *Index.php* funciona como o *Controller* principal (*Front Controller*) sendo responsável pela inicialização de todos os recursos necessários para a execução da aplicação. A principal função do *Routing* é examinar os pedidos HTTP e encaminhá-los de forma adequada. No entanto, se no *Caching* existir informação suficiente para satisfazer o pedido, a resposta será enviada automaticamente sem que o pedido avance mais na execução. Por questões de segurança, antes que o pedido chegue ao *Application Controller*, terá que passar pelo componente *Security* onde será filtrado e validado contra os

[49] http://ellislab.com/codeigniter/user-guide/overview/appflow.html, Julho 2013

principais tipos de ataques a que uma aplicação *web* possa estar sujeita. O *Application Controller* é o componente responsável pela coordenação das operações que visam satisfazer o pedido. Para dar resposta a um pedido terá que interagir com o *Model* (caso exista), utilizar bibliotecas do núcleo do *framework* ou com outros tipos de componentes. Quando o sistema possuir toda a informação necessária e o trabalho estiver concluído, será gerada uma *View* como resposta. Esta *View* será enviada ao cliente mas, caso o sistema de *Caching* esteja ativado, poderá ser aí guardada para que um pedido igual possa ser satisfeito de forma mais rápida e sem sobrecarregar a aplicação principal.

O *CodeIgniter* contém muitas bibliotecas e *helpers* que auxiliam os programadores no desenvolvimento das aplicações. A lista é muito extensa e, por isso, na tabela 4.4 apresentam-se, apenas, as principais bibliotecas e *helpers* disponibilizadas.

Bibliotecas	**Helpers**
Benchmarking	Array
Calendaring	Cookie
Config	Date
Database	Directory
Email	Download
Encryption	File
File Uploading	Form
FTP	HTML
HTML Table	Inflector
Image Manipulation	Security
Input and Security	Smiley
Language	String
Loader	Text
Output	Typography
Pagination	URL
Session	XML
Template Parser	
Trackback	
Unit Testing	
URI Class	
User Agent	
Validation	
XML – RPC	
Zip Encoding	

Tabela 4.4 – Principais bibliotecas e *helpers* do *CodeIgniter*[50]

Da lista de bibliotecas merecem destaque a *Benchmarking*, *Database*, *Email*, *Encryption*, *HTML Table*, *Pagination*, *Unit Testing* e *Session*. A *Benchmarking* permite ao programador avaliar o desempenho de uma secção do código fonte assim como o consumo de memória; a *Database* é usada em todo o *framework* para interagir com uma base de dados (consultas, inserções, alterações e eliminações); a biblioteca *Email* disponibiliza um conjunto de ferramentas utilizadas em operações relacionadas com o envio e receção de

[50] http://ellislab.com/codeigniter/user-guide/overview/features.html, Julho 2013

mails; a *Encryption* trata-se de uma biblioteca usada para a encriptação de informação; a biblioteca HTML *Table* permite a geração de tabelas de uma forma muito simples e versátil; a *Pagination* pode ser utilizada para efetuar a paginação automática dos resultados de uma consulta a uma base de dados, poupando, desta forma, muito trabalho ao programador; a *Unit Testing* poderá ser utilizada para a realização de testes unitários ao código produzido; finalmente, a biblioteca *Session* é utilizada para manter a informação sobre os utilizadores e as sessões estabelecidas.

Ao nível dos helpers disponibilizados pelo CodeIgniter, são de destacar os seguintes: *Cookie*, HTML, *Security*, *Typography* e URL. O *helper Cookie* contém, tal como o nome indica, funções relacionadas com o tratamento de *cookies* (criação, leitura e eliminação); no HTML estão disponíveis funções que facilitam a criação de blocos de código HTML; as funções relacionadas com a segurança (por exemplo, funções para filtragem de tentativas de ataque de *cross site scripting*) estão agrupadas no *helper Security*; o *Typography* contém métodos para formatação de textos (por exemplo, métodos para substituição de quebras de linhas pela respetiva *tag* HTML); finalmente, o *helper* URL contém funções que auxiliam o programador a trabalhar com URL.

Na lista seguinte apresentam-se os principais aspetos positivos proporcionados pelo *framework CodeIgniter*:

- **Funcionalidades.** Apesar de se tratar de um *framework* bastante ligeiro e com pouca necessidade de recursos, disponibiliza ao programador um elevado número de funcionalidades que facilitam muito o trabalho. Das muitas funcionalidades destacam-se as seguintes: validação de formulários, gestão de sessões, bibliotecas de manipulação de imagens, sistemas de gestão de localizações e internacionalização, classes para testes unitários, sistema de *caching* de páginas, encriptação e compactação de dados, ORM integrado com suporte ao padrão *Active Record* e classes para envio de correio eletrónico suportando diversos protocolos. A lista de funcionalidades apresentada trata-se de um pequeno exemplo, atendendo a que a quantidade de funcionalidades do *CodeIgniter* é muito superior e está em constante crescimento;
- **Convenção em detrimento da configuração.** Tal como sucede com outros *frameworks* PHP, no *CodeIgniter* os processos de configuração podem ser substancialmente reduzidos caso se respeitem as regras convencionadas pelo *framework*;
- **Adaptável a qualquer tipo de aplicação.** Tratando-se de um *framework* bastante leve e flexível, adapta-se com muita facilidade a todo o tipo de aplicações *web* e de qualquer dimensão. O facto de não obrigar a que o código obedeça a regras muito rígidas reforça o grau de adaptabilidade;
- **Documentação e comunidade.** A documentação do *CodeIgniter* é abundante e está muito bem organizada. Também se verifica que a comunidade é extensa e bastante participativa;
- **Facilidade de aprendizagem.** Quando comparado com outros *frameworks*, o processo de aprendizagem do *CodeIgniter* é muito reduzido, o que torna o programador produtivo num curto espaço de tempo. A sua simplicidade, coerência do código-fonte e documentação muito contribuem para uma curva de aprendizagem curta.

Relativamente às desvantagens, apontam-se as seguintes:

- O *CodeIgniter* é desenvolvido, em exclusivo, por uma empresa, o que acaba sempre por ser um ponto em desfavor. Embora a comunidade seja muito grande, quem desenvolve o *framework* é um grupo muito restrito de programadores;
- O *CodeIgniter* tem uma abordagem minimalista. Para se poder trabalhar com o *framework* será necessário instalar vários *plugins* adicionais. Praticamente todas as funcionalidades do *framework* requerem a instalação de um *plugin*, o que pode causar algumas dificuldades e demoras no processo de instalação e de arranque da codificação;
- O facto de o *CodeIgniter* não impor regras muito rígidas quanto à forma e tipo do código-fonte e permitir a integração de código externo, pode dar origem a problemas de estabilidade do código, especialmente em aplicações de grande dimensão.

4.3.4. SYMFONY

O *Symfony* é um *framework* PHP para a criação de aplicações *web* sendo o seu desenvolvimento feito pela empresa francesa *SensioLabs* [Zaninotto07]. Apesar de ser um produto desenvolvido por uma empresa, trata-se de um *framework* de código livre que é disponibilizado sob a MIT *License*[51]. A primeira versão do *software* foi lançada em 2005 pelo fundador do projeto, Fabien Potencier, tendo, desde essa data, surgido diversas atualizações. Atualmente, a versão disponibilizada é a 2.3.1, tendo sido lançada em Junho de 2013.[52]

O desenvolvimento do projeto *Symfony* tinha, como principal objetivo, a satisfação dos seguintes requisitos [Zaninotto07]:

- Ser um *framework* multiplataforma e fácil de instalar em qualquer sistema operativo;
- Totalmente independente do sistema de base de dados que possa servir de suporte às aplicações;
- Baseado no princípio da convenção em detrimento da configuração;
- Ser simples e fácil de utilizar, permitindo uma adaptação ao maior número possível de situações;
- Permitir a criação de código que seja fácil de manter e de expandir.

A arquitetura do *Symfony* baseia-se, à semelhança dos restantes *frameworks* PHP analisados, no padrão MVC. Na figura 4.11 apresenta-se uma versão simplificada da

[51] http://symfony.com/doc/current/contributing/code/license.html, Junho 2013
[52] http://symfony.com/blog/symfony-2-3-1-released, Junho 2013

arquitetura. Analisando a referida figura identificam-se as usuais três camadas do padrão MVC. Por sua vez, cada uma das camadas também apresentam diversos componentes.

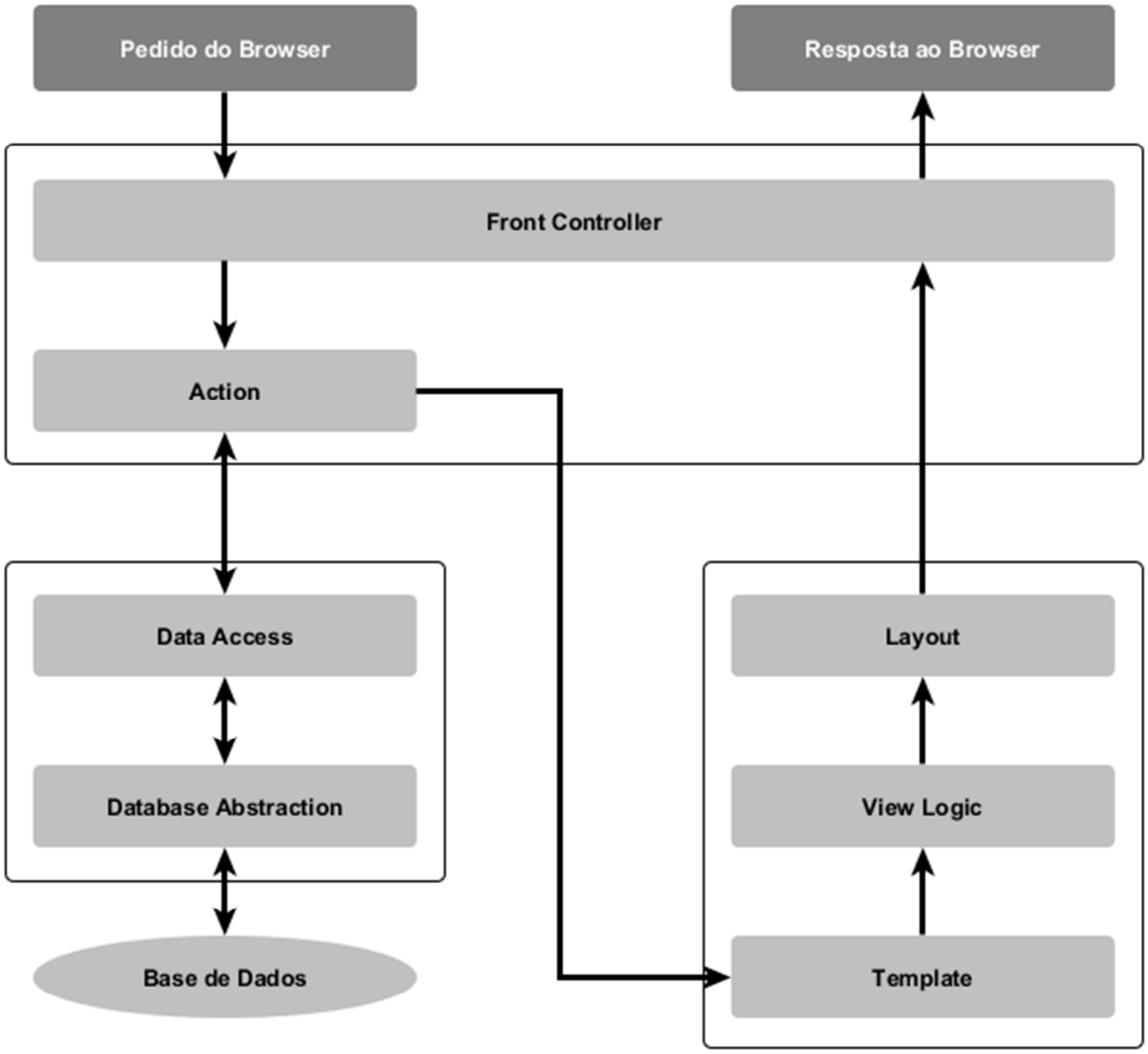


Figura 4.11 – Arquitetura do *Symfony*[53]

Na camada *Model* identificam-se dois componentes: *Data Access* e *Database Abstraction*. Com esta divisão, as funções e métodos de acesso aos dados não têm que conter código específico de cada base de dados. Este tipo de código será sempre colocado no elemento *Database Abstraction* e, caso se mude de base de dados, apenas este elemento terá que ser alterado.

A camada *View* também beneficia da separação de código. Como, na maioria dos casos, as páginas de uma aplicação têm um esquema que não varia, o *Symfony* disponibiliza os elementos *Layout* e *Template*. O *Layout*, normalmente, é comum a toda a aplicação e permite definir elementos constantes nas páginas (por exemplo, os cabeçalhos e os

[53] Adaptado de [Zaninotto07]

rodapés). O *Template* é responsável por apresentar a informação fornecida pelo *Controller*. A coordenação entre estes dois elementos é feita pelo *View Logic*.

Na camada *Controller* identificam-se dois tipos distintos de elementos: o *Front Controller* e os *Action Controllers*. O primeiro é um *controller* único para toda a aplicação, fornecendo um ponto de entrada único para todos os pedidos. Ao nível dos *Action Controllers*, poderá existir um para cada página da aplicação. Apesar de serem identificados 7 componentes diferentes na arquitetura (*Database Abstraction*, *Data Access*, *View Logic*, *Template Layout*, *Front Controller* e *Action Controllers*), não significa que, para se desenvolver uma aplicação, seja necessário escrever todo este código para cada página. O *Front Controller* e o *Layout* são comuns a toda a aplicação pelo que terão que ser escritos apenas uma vez (e no caso do *Front Controller*, o código é gerado automaticamente pelo *Symfony*). O código da camada *Model* também pode ser gerado, de forma automática, pelo *framework*, com base na estrutura da base de dados. O *Symfony* contém duas bibliotecas (*Propel* e *Creole*) que tratam da geração do código dos modelos. Os únicos elementos que requerem a intervenção do programador são os *Action Controllers*, os *Templates* e os *Views Logic*. Tudo o resto é desenvolvido e coordenado pelo *framework*.

Na tabela 4.5 apresentam-se as principais bibliotecas do *Symfony*.

Componente	**Componente**
BrowserKit	Form
ClassLoader	HttpFoundation
Config	HttpKernel
Console	Locale
CssSelector	Intl
Debug	Icu
DependencyInjection	OptionsResolver
DomCrawler	Process
EventDispatcher	PropertyAccess
Filesystem	Routing
Finder	Security
Translation	Serializer
Validator	Stopwatch
Yaml	Templating

Tabela 4.5 – Principais componentes do *Symfony*[54]

Da lista apresentada merecem destaque as seguintes: a biblioteca *Config* é usada para trabalhar valores referentes às configurações que, geralmente, são armazenadas em ficheiros externos. A *EventDispatcher* implementa o padrão *Observer* para a emissão e captação de eventos entre os componentes de uma aplicação. A biblioteca *DependencyInjection* usa-se, à semelhança do que é feito no *Spring Framework*, para implementar o princípio da injeção de dependências e para normalizar e centralizar a criação de objetos. A *Serialize* é uma das classes mais utilizadas porque permite a criação e leitura de diversos formatos de dados (p. ex. XML, JSON e YAML). A depuração das

[54] http://symfony.com/doc/current/components/index.html, Junho 2013

aplicações fica a cargo do componente *Debug*, o qual facilita muito a análise e depuração do código PHP, juntamente com o *Stopwatch* que permite a medição do desempenho de um fragmento de código. O componente *DomCrawler* também se reveste de alguma importância uma vez que é usado para a navegação na árvore DOM de documentos HTML ou XML. O componente *Security* disponibiliza diversos métodos usados para a encriptação de dados e autenticação. O *Filesystem* contém diversos métodos para a manutenção e controlo do sistema de ficheiros assim como o componente *Finder*.

Para além dos componentes apresentados no parágrafo anterior, o *Symfony* disponibiliza ainda mais 3 classes de extrema importância: a *sfController* que se trata da classe que implementa o *Front Controller*, a *sfRequest* que armazena toda a informação referente aos pedidos e a *sfResponse* que contém todos os dados referentes à resposta que se está a processar.

Apresentam-se, de seguida, os principais aspetos positivos que, normalmente, são apontados ao *Symfony*:

- **Ferramentas.** O *Symfony* é distribuído com muitas ferramentas que auxiliam muito os programadores no desenvolvimento das aplicações. Por exemplo, as classes que pertencem ao *Model* podem ser geradas usando as ferramentas apropriadas. Também são disponibilizadas ferramentas adicionais para *logging*, administração ou depuração, entre outros;
- **Crescimento e utilização.** Segundo os autores do projeto, o *Symfony* é um dos *frameworks* PHP mais utilizados, tendo o número de utilizadores e *downloads* vindo a crescer a um ritmo constante ao longo do tempo. Os programadores do *Symfony* são, também eles, utilizadores do *framework* no desenvolvimento de outras aplicações, o que permite uma perceção diferente do que é necessário e uma melhor adequação do *framework* às reais necessidades dos programadores e das aplicações desenvolvidas;
- ***Plugins*.** Existem muitos *plugins* disponíveis para serem utilizados livremente pelos programadores. Estes *plugins* estão centralizados num único local e devidamente catalogados para uma fácil localização e seleção;
- **Funcionalidades.** O *framework* disponibiliza avançadas funcionalidades aos programadores, destacando-se as seguintes: biblioteca para tratamento das questões relacionadas com a internacionalização, testes unitários, validação e filtragem de dados, ORM *Doctrine* integrado, autenticação, *caching* e sistema de gestão de eventos;
- **Simplicidade e flexibilidade.** O *Symfony* carateriza-se por ser extremamente simples de utilizar, bastando que o programador possua conhecimentos de PHP para, rapidamente, se tornar produtivo com a sua utilização. A flexibilidade também é uma caraterística a ter em conta uma vez que se trata de um *framework* muito adaptável a todo o tipo e dimensão de aplicações e que não impõe grandes restrições quanto à forma de programar. Assim, pode ser utilizado para a criação de grandes aplicações com um elevado número de funcionalidades, aplicações modulares em que as funcionalidades vão sendo adicionadas à medida que são necessárias ou para o desenvolvimento de pequenas aplicações que, inclusive, podem ser enquadradas em projetos maiores.

Quanto aos aspetos negativos do *Symfony*, apresentam-se os seguintes:

- Apesar de existirem muitos *plugins* disponíveis, que aumentam consideravelmente as funcionalidades, muitos deles foram desenvolvidos para versões antigas do *framework*, o que pode dificultar ou impossibilitar a sua utilização na versão mais recente do *Symfony*;
- Comparado com outros *frameworks* PHP, a documentação do *Symfony* é disponibilizada em menor quantidade e qualidade, o que não significa que seja má;
- Algumas ferramentas podem ser de difícil utilização, destacando-se, neste ponto, as ferramentas de depuração;

4.3.5. Yii

O *Yii* é um *framework* PHP, *component-based*, de código aberto, vocacionado para o desenvolvimento de aplicações *web* de forma rápida e fácil. O projeto teve início em Janeiro de 2008 por iniciativa do programador Qiang Xue, tendo a primeira versão sido lançada em Dezembro do mesmo ano [Winesett12]. A versão atual do *framework* é a 1.1.13, lançada em Dezembro de 2012[55], sendo distribuída sob a *New* BSD *License*, a qual permite a utilização do *framework* praticamente sem qualquer limitação.[56]

A implementação MVC do *framework Yii* segue o usual do padrão. Numa aplicação *Yii* os pedidos recebidos do cliente são sempre analisados pelo *Application Router*, o qual decidirá, com base no formato do URL, para que componente da aplicação deverá ser enviado. Na maioria dos casos, o *Router* selecionará um método específico dentro do *Controller*. Após receber o pedido, o método selecionado efetuará as operações necessárias para o satisfazer. Estas operações podem envolver uma ou várias interações com o *Model*. Depois de concluir as operações necessárias, o *Controller* terá os dados necessários para a resposta ao pedido, os quais serão enviados para o *View*. É da responsabilidade deste último componente preparar a resposta adequada.

No *Yii* as classes do *Model* são sempre instâncias da classe *CModel* ou de uma sua subclasse. Contudo, embora todas as classes do *Model* sejam dependentes do *CModel*, o *Yii* contempla a criação de dois tipos de classes diferentes: *CFormModel* e *CActiveRecord*. O *CFormModel* representa os dados obtidos a partir de um formulário HTML e encapsula toda a lógica relacionada com a validação dos campos dos formulários. A classe *CActiveRecord* está relacionada com o padrão *Active Record* e permite representar um registo de uma tabela da base de dados e contém, também, toda a lógica de negócio associada ao registo em causa.

[55] http://www.yiiframework.com/news/65/yii-php-framework-1-1-13-is-released/, Junho 2013
[56] http://www.yiiframework.com/license/, Junho 2013

O *View*, tal como é usual no padrão MVC, é o responsável pela geração visual da resposta ao pedido, sendo que esta resposta, na maior parte das situações, é gerada com base nos dados obtidos do *Model*. Trata-se, normalmente, de código HTML embora também possa conter código PHP (o qual deverá ser muito simples).

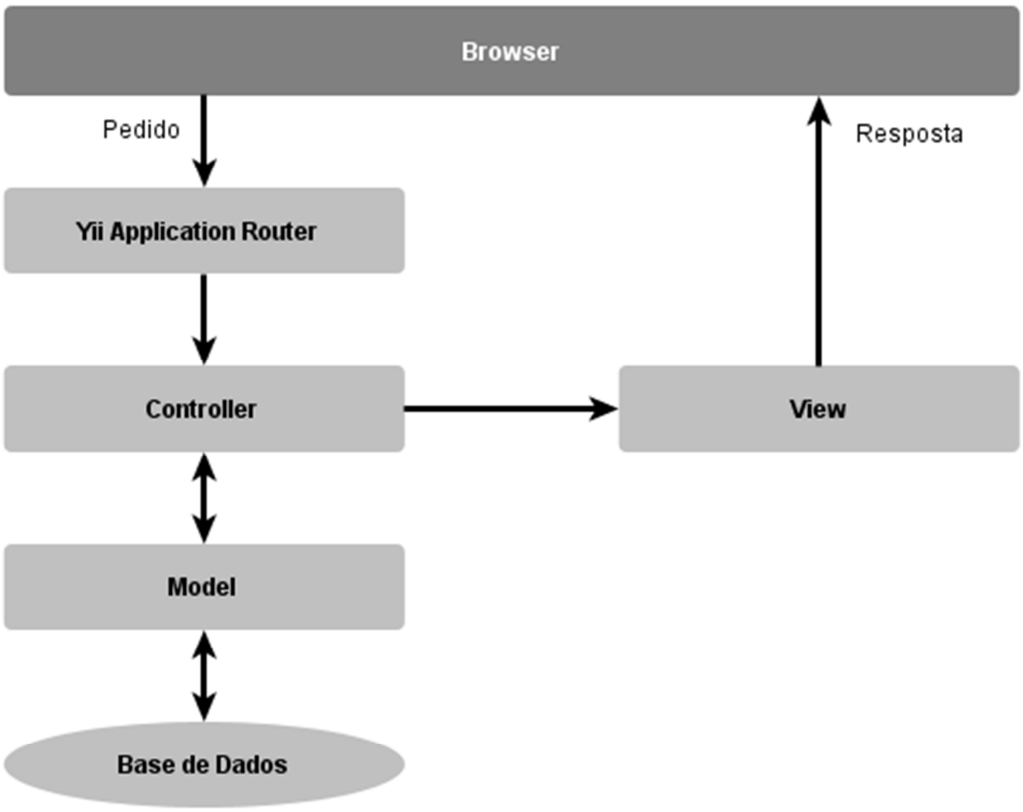


Figura 4.12 – Arquitetura do *framework* PHP *Yii*[57]

A coordenação de toda a aplicação é feita pelo *Controller*, o qual, após receber o pedido do *browser*, iniciará as ações necessárias para devolver a resposta adequada. No *framework Yii*, o *Controller* será sempre uma instância da classe *CController* ou de uma classe herdada. Neste *framework*, o *Controller* e o *View* trabalham de forma muito próxima. Os componentes do *View* pertencem sempre ao *Controller*, o que permite que, a partir do *View*, seja possível aceder facilmente ao *Controller*. Adicionalmente, o *Controller* também pode ativar eventos pré e pós processamento, gerir a paginação dos dados a apresentar, controlar os acessos a determinadas opções e desempenhar muitas outras tarefas que libertam o programador para as tarefas específicas da aplicação que está a desenvolver.

Uma análise à lista das classes da API do *framework Yii* permite identificar várias centenas de classes que estão à disposição dos programadores. Por este motivo, na tabela 4.6 apresentam-se apenas alguns componentes que, geralmente, são utilizados no desenvolvimento de aplicações empresariais.

[57] Adaptado de [Winesett12]

zii.widgets	system.web.widgets	system.db
CBaseListView	CActiveForm	CDbCommand
CDetailView	CAutoComplete	CDbConnection
CListView	CClipWidget	CDbDataReader
CMenu	CContentDecorator	CDbException
CPortlet	CFilterWidget	CDbMigration
CButtonColumn	CFlexWidget	CDbTransaction
CCheckBoxColumn	CHtmlPurifier	CActiveFinder
CDataColumn	CInputWidget	CActiveRecord
CGridColumn	CMarkdown	CActiveRecordBehavior
CGridView	CMaskedTextField	CActiveRecordMetaData
CLinkColumn	CMultiFileUpload	CActiveRelation
CJuiAccordion	COutputCache	CBaseActiveRelation
CJuiAutoComplete	COutputProcessor	CBelongsToRelation
CJuiButton	CStarRating	CHasManyRelation
CJuiDatePicker	CTabView	CHasOneRelation
CJuiDialog	CTextHighlighter	CJoinElement
CJuiDraggable	CTreeView	CJoinQuery
CJuiDroppable	CWidget	CManyManyRelation
CJuiInputWidget	CCaptcha	CDbColumnSchema
CJuiProgressBar	CCaptchaAction	CDbCommandBuilder
CJuiSlider	CBasePager	CDbCriteria
CJuiSortable		CDbExpression
CJuiTabs		CDbSchema
CJuiWidget		CDbTableSchema
system.web.helpers	**system.web.auth**	**system.caching**
CGoogleApi	CAccessControlFilter	CApcCache
CHtml	CAccessRule	CCache
CJSON	CAuthAssignment	CDbCache
CJavaScript	CAuthItem	CDummyCache
CJavaScriptExpression	CAuthManager	CEAcceleratorCache
	CBaseUserIdentity	CFileCache
	CDbAuthManager	CMemCache
	CPhpAuthManager	CMemCacheServerConfig.
	CUserIdentity	CWinCache
	CWebUser	CZendDataCache
Outros		
CApplication	CLogger	CCaptchaValidator
CErrorEvent	CDateTimeParser	CCompareValidator
CErrorHandler	CFileHelper	CDateValidator
CEvent	CFormatter	CDefaultValueValidator
CException	CPropertyValue	CEmailValidator
CConfiguration	CTimestamp	CFileValidator
CList	CHttpCookie	CFilterValidator
CMap	CHttpRequest	CNumberValidator
CStack	CHttpSession	CRangeValidator

Tabela 4.6 – Classes e componentes do *Yii*[58]

As classes dos *packages zii.widgets* e *system.web.widgets* estão relacionadas com a implementação dos aspetos visuais das aplicações, mais concretamente com os componentes visuais que são apresentados aos utilizadores e que permitem que estes interajam com a aplicação. São exemplos destes componentes os campos para inserção de

[58] http://www.yiiframework.com/doc/api/, Junho 2013

valores (*CAutoComplete*, *CInputWidget*, *CMaskedTextField*, *CJuiInputWidget* ou o *CJuiDatePicker*), as grelhas e listas (*CGridView*, *CTabView*, *CTreeView* e *CListView*), os componentes para organização e apresentação de outros componentes (*CJuiAccordion*, *CJuiDialog* e *CJuiTabs*) ou, ainda, componentes genéricos como menus, barras de progressão, botões e CAPTCHA.

Os componentes do *package system.db* estão todos relacionados com a interação da aplicação com as bases de dados. Identificam-se, por exemplo, classes para estabelecimento de uma ligação com uma base de dados (*CDbConnection*), para envio de um comando de leitura ou alteração dos dados (*CDbCommand*), para obtenção sequencial dos dados provenientes de uma consulta (*CDbDataReader*), para início, fim ou cancelamento de uma transação (*CDbTransaction*). Neste *package* também se identificam classes relacionadas com a estrutura e organização da base de dados. Existem classes para representação das tabelas (*CDbSchema*), campos (*CDbColumnSchema*), relacionamentos (*CHasManyRelation*, *CHasOneRelation* ou *CManyManyRelation*). Embora não estejam referenciados na tabela 4.6, o *package system.db* está dividido em várias unidades, contendo cada uma delas classes específicas das principais bases de dados (*MySQL*, *MS SQL Server*, *PostgreSQL* e *SQLite*). Isto não significa que o *Yii* apenas permita trabalhar com estas bases de dados mas sim que existem classes que aproveitam as características únicas de cada uma das bases de dados. Na prática, o *Yii* permite trabalhar com quase todo o tipo de bases de dados existentes.

No *package system.web.helpers* estão agrupados componentes de utilidade diversa. Por exemplo, o *CGoogleApi* disponibiliza métodos de acesso à API do *Google*. Também contém classes para trabalhar com dados no formato JSON, manuseamento de código HTML e para tratamento de questões relacionadas com o *JavaScript*.

As questões relacionadas com a autenticação e permissão estão agrupadas no package *system.web.auth*. Uma das classes mais importantes deste módulo é a *CAuthManager*, que serve de base para as restantes classes de gestão de acessos (*CDbAuthManager*, *CPhpAuthManager*), e que implementa o *Role-Based Access Control* (RBAC). Também se destaca a classe *CWebUser*, que permite guardar o estado de um utilizador de uma aplicação.

Todo o sistema de *caching* do *Yii* reside nas classes do package *system.caching*. Estas classes tratam de todas as questões relacionadas com os diversos tipos de *cache* que são possíveis de implementar. A classe base é a *CCache*, a partir da qual derivam todas as classes que implementam mecanismos de *cache* (p. ex. *xcache*, *WinCache – cache* específica para aplicações que correm na plataforma *Windows – memcache* ou *eAccelerator*).

No último grupo de classes, apresentam-se componentes que pertencem a vários *packages*. A classe *CApplication* desempenha um papel muito importante uma vez que se trata da classe que contém toda a informação relevante sobre a aplicação que se está a executar e que disponibiliza métodos relativos a toda a aplicação (por exemplo, gestão de erros e exceções, mensagens ou pedidos). As classes *CEvent* e *CException* encapsulam os aspetos relacionados com a gestão de eventos e exceções, contendo todo o código necessário para estas tarefas. As classes *CConfiguration*, *CList, CMap, CQueue* e *CStack* pertencem todas ao mesmo *package* (*system.collections*) e implementam diversos tipos de

listas de objetos. A classe *CLogger* é utilizada para registar em memórias a atividade da aplicação, fazendo parte do *package system.loggin*, o qual contém mais componentes relacionados com a manutenção do registo das operações. As classes *CHttpCookie*, *CHttpRequest* e *CHttpSession* encapsulam informações e fornecem métodos para tratamento de *cookies*, do pedido e da sessão. Na lista das classes genéricas merecem, ainda, referência as classes de validação de dados (p. ex. *CCaptchaValidator*, *CDateValidator*, *CEmailValidator*, *CFileValidator*, *CFilterValidator*, e *CNumberValidator*) que, como o próprio nome indica, possibilitam a verificação dos dados trabalhados pela aplicação, quer sejam provenientes do utilizador ou de outra fonte qualquer.

Na lista seguinte apresentam-se os aspetos positivos do *Yii*:

- **Evolução muito rápida e elevado potencial.** Quando comparado com os *frameworks* PHP analisados, o *Yii* é o que apresenta os maiores níveis de crescimento na utilização e na comunidade. A leitura dos grupos de discussão e de artigos relacionados com o PHP demonstram claramente que se trata de um *framework* que recolhe uma excelente opinião junto da comunidade;
- **Desempenho.** Diversos testes têm demonstrado que o *Yii* é o *framework* PHP que apresenta o melhor desempenho. Para esta elevada performance muito contribui a técnica de *Lazy Loading* em que os objetos apenas são criados ou lidos à medida que vão sendo utilizados pela aplicação[59];
- **Ferramentas e funcionalidades.** O *Yii* é disponibilizado com muitas ferramentas que prestam um grande auxílio aos programadores (por exemplo, utilitários para geração automática de código). Para além das ferramentas, o conjunto de funcionalidades disponibilizadas é elevado. A título de exemplo, apresentam-se, a seguir, algumas funcionalidades do *framework*: classes de tratamento das questões relacionadas com a internacionalização e localização; classes e *helpers* para validação e filtragem dos dados inseridos pelo utilizador nos formulários; diversos níveis de *caching* de dados (p. ex. *caching* de dados, páginas e respostas); possibilidade de realização de testes unitários e testes funcionais; diversas classes relacionadas com a autenticação, suportando, entre outros, a autorização RBAC) e segurança, para tratamento e prevenção dos principais tipos de ataques conhecidos. Ao elevado número de funções do *Yii*, deve-se adicionar ainda mais funcionalidades e componentes criados pela comunidade. Na página onde estão catalogados os componentes desenvolvidos por terceiros contam-se mais de mil unidades disponíveis para *download*;
- **Integração automática com a biblioteca jQuery.** O *Yii* já contempla um conjunto grande de componentes visuais perfeitamente integrados com a biblioteca *Javascript jQuery*. Esta integração resulta num código mais eficiente e fácil de manter;
- **Excelente suporte para interação com as bases de dados.** O *framework* engloba um sofisticado ORM, que é implementado com base no padrão *Active Record*. O ORM integra-se totalmente com todos os componentes do *Yii*, permitindo muitos automatismos na transferência e validação de dados.

[59] http://www.yiiframework.com/performance/, Julho 2013

Quanto aos pontos negativos, salientam-se os seguintes:

- O *Yii* pode ser mais difícil de aprender do que outros *frameworks* PHP, caso os programadores tenham pouca experiência no desenvolvimento na linguagem PHP ou na utilização de *frameworks*;
- Tratando-se de um *framework* relativamente recente, a documentação existente é em menor quantidade quando comparada com a de outros *frameworks*. No entanto, esta situação tende a ser ultrapassada à medida que a comunidade vai crescendo e o *framework* vai evoluindo. Por outro lado, a pouca maturidade do projeto pode significar um maior número de erros uma vez que o código não está tão maduro e testado quando o código de outros *frameworks*.

5. SELEÇÃO DOS POTENCIAIS *FRAMEWORKS*

No presente capítulo selecionaremos, dentro de cada plataforma, qual o *framework* que será o mais apropriado para o desenvolvimento do *software* GM *Macwin*. Esta seleção será feita com base nas análises do capítulo anterior e, numa primeira fase, passará pela verificação da existência, em cada *framework*, do seguinte conjunto de funcionalidades:

- **Acesso BD/ORM –** Indica se o *framework* inclui, nativamente, um ORM ou código que simplifique e facilite a interação da aplicação com bases de dados;
- **Autenticação –** Significa que o *framework* disponibiliza código relacionado com as funções de autenticação de utilizadores;
- **Validações –** Este item significa que o *framework* possui funcionalidades de validação dos dados inseridos pelos utilizadores;
- **Cache –** Esta funcionalidade indica se o *framework* disponibiliza código para a criação e manutenção de sistemas de *caching*, com o objetivo de aumentar o desempenho da aplicação;
- **AJAX –** O *framework* contempla código que facilita o desenvolvimento de aplicações *web* AJAX, mais concretamente, que facilite as comunicações assíncronas entre o cliente e o servidor;
- **Internacionalização –** O *framework* possui código para tratamento das questões relacionadas com a internacionalização, principalmente a possibilidade de definir e gerir vários idiomas;
- **Geração de código –** Existem aplicativos que, mediante determinados pressupostos ou configurações, podem gerar código-fonte de forma automática;
- **Testes unitários –** Indica se o *framework* contempla a possibilidade de se criarem e executarem testes unitários.

Depois de se identificarem as funcionalidades acima referidas, avançaremos para a próxima fase na qual faremos uma valorização das caraterísticas apresentadas na lista seguinte:

- **Funcionalidades/Dimensão –** Esta caraterística refere-se à quantidade de funcionalidades disponibilizadas pelos *frameworks*. Quanto mais funcionalidades disponibilizar maior será a pontuação;
- **Desempenho -** O desempenho apenas terá alguma relevância e algum peso nos *frameworks* PHP uma vez que estes apresentam muitas semelhanças entre si;
- **Custo –** Nesta caraterística avalia-se o custo de aquisição das ferramentas de desenvolvimento. Não se consideram, contudo, os eventuais custos de licenciamento do *software* que corre no lado do servidor porque todos os clientes da *Macwin* já possuem licenças para este tipo de *software*;
- **Suporte –** Existência de suporte técnico para o *framework*. Neste item será valorizada a disponibilização de apoio técnico pela organização que desenvolve o *framework* ou por outra que possua certificação e competências para o efeito;

- **Conhecimento atual –** Este item representa o conhecimento técnico que os funcionários da *Macwin* possuem sobre o *framework*;
- **Curva de aprendizagem –** Representa o grau de dificuldade e o tempo necessário para que um programador domine ou seja produtivo na utilização do *framework*. Aqui, a escala aplica-se de forma inversa, ou seja, quanto maior for o valor apresentado menor será a curva de aprendizagem;
- **Escalabilidade –** Neste item avalia-se a capacidade que o *framework* apresenta para o desenvolvimento de uma aplicação que está em constante evolução, à qual são adicionadas, com muita frequência, novos módulos e funcionalidades e em que os requisitos estão em constante mutação;
- **Ferramentas de desenvolvimento –** Avalia-se a qualidade (e, numa escala menor, a quantidade) das ferramentas de desenvolvimento disponíveis;
- **Integração com *SQL Server* –** Valoriza a facilidade com que o *framework* interage com a base de dados *Microsoft SQL Server*, uma vez que este é o sistema de gestão de bases de dados atualmente utilizado pela *Macwin*;
- **Tempo de desenvolvimento –** Nesta caraterística tenta-se aferir o tempo que os programadores demoram a desenvolver uma aplicação;
- **Multiplataforma –** Esta caraterística valoriza a possibilidade do código que corre no lado do cliente estar disponível em vários sistemas operativos e dispositivos.

Cada uma destas caraterísticas será valorizada numa escala de 1 (um) a 5 (cinco) em que o 1 representa o pior resultado e o 5 o melhor. No final, para cada plataforma, será selecionado o *framework* que apresentar a melhor pontuação.

5.1. .NET

Os *frameworks* que foram analisados nesta plataforma (ver secção 4.1) apresentam 3 abordagens distintas para o desenvolvimento de aplicações *web*, embora nenhum deles abranja todas as áreas de desenvolvimento de uma aplicação empresarial. Pode-se afirmar que os três *frameworks* analisados (ASP.NET *Web Forms*, ASP.NET MVC e *Silverlight*) se focam mais nos aspetos visuais das aplicações relegando para a plataforma .NET e para o ASP.NET todas as restantes questões.

O *Silverlight* trata-se de um *framework* muito vocacionado para o desenvolvimento da parte visual das aplicações (tornando o desenvolvimento de aplicações *web* muito parecido com o desenvolvimento de aplicações para a plataforma *desktop*), não apresentando qualquer solução para o código que é executado noutras áreas/camadas das aplicações. Dos três *frameworks*, é o único que obriga à instalação de *software* adicional no dispositivo onde é executado o cliente e que, por este motivo, não está disponível em todas as plataformas.

O ASP.NET *Web Forms* apresenta uma abordagem totalmente diferente embora também tente aproximar o desenvolvimento *web* ao desenvolvimento para o *desktop*, criando uma camada de abstração que emula as caraterísticas deste último. Ao contrário do *Silverlight*, não requer qualquer *plugin* do lado do cliente, sendo o processamento realizado, em grande medida, do lado do servidor. O principal objetivo do ASP.NET *Web Forms* é o desenvolvimento rápido de aplicações e a facilidade da mudança da plataforma *desktop* para a *web*.

O ASP.NET MVC, pelo contrário, apresenta-se como um *framework* mais profissional, no sentido em que dá ao programador a possibilidade de controlar todos os aspetos das aplicações *web* e facilita a implementação de várias boas práticas de programação. A abordagem proposta pelo *framework* é a de dar ao programador o controlo total sobre o que está a ser feito. Ao contrário do que se passa com o ASP.NET *Web Forms*, onde muito do trabalho é feito de forma automática pelo *framework*, no ASP.NET MVC é exigida uma maior participação por parte do programador. Com isto, consegue-se um maior controlo sobre o que se está a passar, tornando o desenvolvimento mais flexível e poderoso. No entanto, esta flexibilidade e controlo são obtidos à custa da facilidade e da velocidade com que se desenvolve uma aplicação.

	Silverlight	**ASP.NET *Web Forms***	**ASP.NET MVC**
Licença	*Freeware*/ Proprietária	*Freeware*/ Proprietária	*Apache License 2.0*/Proprietária
Versão atual	5.1	4.5	4.0
Data da versão atual	Julho/2013	Agosto/2012	Agosto/2012
Data de início do projeto	Setembro/2007	Janeiro/2002	Março/2009
Programador	Microsoft	Microsoft	Microsoft
Padrão			MVC *Front Controller*
Acesso BD/ORM[60]	X	X	X
Autenticação			X
Validações	X	X	X
Cache	X	X	X
AJAX	X	X	X
Internacionalização	X	X	X
Geração de código			X
Testes unitários			X

Tabela 5.1 – Comparativo das funcionalidades dos *frameworks* ASP.NET

[60] As funcionalidades de acesso BD/ORM, *Cache* e Internacionalização são disponibilizadas pela plataforma .NET estando, por este motivo, disponíveis para utilização em todos os *frameworks*.

As licenças dos *frameworks* apresentam dois tipos diferentes porque as aplicações podem ser desenvolvidas usando as ferramentas *Express* (gratuitas, embora apresentem algumas limitações na utilização) ou as *Standard* (pagas). É possível desenvolver uma aplicação *web* utilizando apenas as versões *freeware* das ferramentas de desenvolvimento e da plataforma mas, para projetos de maior dimensão, estas ferramentas apresentam algumas limitações que impossibilitam a sua utilização. Por exemplo, o *Visual Studio Express* não permite a instalação de *plugins*, o que impossibilita a integração com qualquer controlador de versões do código fonte como, por exemplo, o *Apache Subversion*.

Apesar de nenhum dos *frameworks* incluir, de raiz, um ORM, existem na plataforma .NET vários projetos que podem ser utilizados para esta finalidade. De entre todos eles destacam-se o *Entity Framework* (desenvolvido pela própria *Microsoft* sendo o ORM padrão) e o *NHibernate* podendo, qualquer um deles, ser utilizado nos *frameworks* analisados. Ao contrário do que acontece nas plataformas *Java* e PHP, os *frameworks* da plataforma .NET não incluem muitas funcionalidades porque estas estão disponíveis na própria plataforma não sendo, por isso, necessário que os *frameworks* as implementem. Assim todas as funcionalidades listadas na tabela 5.1 estão disponíveis para utilização por qualquer um dos *frameworks*, embora, de uma maneira geral, o ASP.NET *Web Forms* tente simplificar um pouco mais a sua utilização. A exceção a esta regra é a implementação de testes unitários.

Na tabela 5.2 apresenta-se uma avaliação dos fatores de avaliação dos *frameworks* .NET.

	Silverlight	**ASP.NET *Web Forms***	**ASP.NET MVC**
Funcionalidades/Dimensão	3	5	4
Desempenho	4	4	5
Custo	3	3	3
Suporte	5	5	5
Conhecimento atual	1	1	1
Curva de aprendizagem	5	4	3
Escalabilidade	3	3	5
Ferramentas desenvolvimento	5	5	5
Integração com *SQL Server*	5	5	5
Tempo de desenvolvimento	4	5	4
Multiplataforma	2	5	5
Total	**40**	**45**	**45**

Tabela 5.2 – Avaliação dos fatores de seleção

A valorização dos fatores de seleção relativos à plataforma .NET permite destacar o ASP.NET *Web Forms* e o ASP.NET MVC em detrimento do *Silverlight*. O principal motivo pelo qual estes dois *frameworks* se destacaram foi a questão da multiplataforma. Num

cenário em que os sistemas operativos da *Apple* e *Linux* poderão ganhar cada vez mais preponderância em relação aos sistemas operativos *Microsoft*, optar-se por uma tecnologia que limita o funcionamento das aplicações aos sistemas operativos *Windows* vai contra o objetivo inicial do processo de conversão das aplicações para a plataforma *web* e que é a sua universalidade e independência de sistemas operativos, plataformas e dispositivos. Por este motivo, a escolha do *framework* será feita, apenas, entre o ASP.NET *Web Forms* e o ASP.NET MVC. De uma forma resumida, os aspetos que recomendam a utilização do ASP.NET *Web Forms* são os seguintes:

- *Framework* baseado em eventos que simplifica e acelera o desenvolvimento;
- Permite uma gestão automatizada da manutenção do estado entre pedidos HTTP;
- Mais vocacionado para equipas de desenvolvimento pequenas.

Quanto ao ASP.NET MVC os aspetos a considerar são os seguintes:

- Utiliza o padrão MVC o que possibilita uma efetiva separação de conceitos e uma melhor organização do código;
- Permite um controlo total sobre todos os aspetos da aplicação;
- Facilita a implementação de testes unitários, a flexibilidade e a expansibilidade;
- Mais vocacionado para equipas de desenvolvimento grandes onde há uma divisão entre *web developers* e *web designers*.

Analisando os aspetos mais relevantes de cada um dos *frameworks*, conclui-se que o mais adequado para o desenvolvimento da aplicação GM *Macwin* será o ASP.NET MVC. Apesar do ASP.NET *Web Forms* apresentar como principal argumento a facilidade e rapidez no desenvolvimento, a experiência na *Macwin* tem demonstrado que, com este tipo de abordagem, o tempo que se ganha na fase do desenvolvimento perde-se totalmente na fase da manutenção. A ferramenta de desenvolvimento que a empresa utiliza atualmente segue um princípio parecido com o do ASP.NET *Web Forms*, onde se privilegia a facilidade e a rapidez de desenvolvimento. Com isto, ganha-se, efetivamente, muito tempo na fase inicial (rapidamente se desenvolve uma aplicação ou um módulo) mas, regra geral, o código desenvolvido é pouco flexível e difícil de manter. Por este motivo, sempre que é necessário efetuar alguma alteração, perde-se muito mais tempo e o custo é superior. O ASP.NET MVC, pelo contrário, foi desenvolvido com o objetivo de favorecer a qualidade e flexibilidade do código. Desenvolver uma aplicação usando o ASP.NET MVC demorará mais tempo mas, regra geral, o código estará mais bem estruturado e preparado para futuras alterações e/ou correções. Como a aplicação a converter está em constante evolução (quase sempre implicando a alteração do código já existente), o aspeto do tempo de manutenção/alteração reveste-se de uma importância extrema e é fundamental no processo de seleção do *framework*.

Existe ainda outro fator que também contribuiu para a opção pelo ASP.NET MVC e que é o facto de se verificar que este *framework* está a ser uma grande aposta da *Microsoft*. Embora não exista nada escrito que, de forma explicita, refira que a *Microsoft* aposta mais

no MVC do que no *Web Forms*, pela leitura de vários artigos e páginas da própria *Microsoft* pode-se concluir que há uma aposta clara da empresa no ASP.NET MVC.

5.2. *JAVA*

Os *frameworks* analisados na plataforma *Java* (ver secção 4.2) apresentam abordagens diferentes para o desenvolvimento de aplicações assim como dimensões e áreas de atuação bastante diferentes. Se, por um lado, temos o *Google Web Toolkit*, mais reduzido e muito vocacionado para a camada visual das aplicações, do lado oposto temos o *Spring Framework* que abarca praticamente todas as áreas do desenvolvimento de uma aplicação empresarial sendo o *framework* mais extenso e completo de todos os analisados ao longo deste trabalho. Entre estas duas propostas, temos o *JavaServer Faces* e o *Apache Struts 2* que, não sendo tão completos quanto o *Spring*, apresentam-se como soluções mais ligeiras: o primeiro mais focado nos aspetos visuais e o segundo como uma solução mais global.

O *Google Web Toolkit* é um *framework* bastante diferente dos restantes uma vez que, na prática, trata-se de um conversor de código *Java* em *JavaScript* que será executado no lado do cliente. Para além deste processo de compilação, o *framework* também disponibiliza componentes para uma eficiente comunicação entre o cliente e o servidor. Quanto ao código que corre no lado do servidor, o *Google Web Toolkit* não apresenta qualquer solução específica sendo da responsabilidade do programador selecionar as ferramentas e bibliotecas mais adequadas. O principal elemento diferenciador deste *framework* é a possibilidade de se desenvolver uma aplicação relativamente complexa sem ter qualquer conhecimento das principais tecnologias relacionadas com a *web* (fundamentalmente HTML, CSS e *JavaScript*) tornando a passagem de um ambiente *desktop* para um ambiente *web* mais fácil e rápida.

O *JavaServer Faces*, embora também estando muito vocacionado para a parte visual das aplicações, apresenta um funcionamento substancialmente diferente do *Google Web Toolkit*. Neste caso, é exigido ao programador mais conhecimentos para poder desenvolver uma aplicação. No entanto, esta exigência acaba por ser vantajosa uma vez que conhecendo os aspetos internos das aplicações e das tecnologias que a suportam acabará sempre por beneficiar os programadores. O *JavaServer Faces* é um *framework* que pertence à especificação *Java* EE estando, por este motivo, muito integrado na plataforma *Java*. Por ser baseado numa especificação, existem várias implementações, quase todas de utilização gratuita ou mista, que poderão ser testadas e utilizadas pelos programadores. Grandes empresas da área informática (entre elas a *Oracle*, IBM e *Jboss*) disponibilizam diversas implementações do *JavaServer Faces*.

O *Apache Struts 2*, ao contrário dos dois anteriores, disponibiliza um conjunto mais vasto de funcionalidades, não se centrando, apenas, nos aspetos visuais das aplicações. Foi um dos primeiros *frameworks Java* para o desenvolvimento de aplicações *web* e durante muito tempo foi um dos mais utilizados. O *Apache Struts 2* disponibiliza vários componentes que facilitam o trabalho de desenvolvimento das aplicações o que, aliados a uma grande

simplicidade e a uma elevada elasticidade, o tornam num *framework* bastante capaz para o desenvolvimento de aplicações *web*.

Por último, o *Spring Framework* surge como a plataforma mais completa para o desenvolvimento de aplicações *web*. Tendo sido desenvolvido, desde o início, com base num conjunto de princípios de programação sólidos apresenta-se, simultaneamente, como um *framework* robusto e simples. A lista de organizações que utilizam *software* desenvolvido usando este *framework*, e que pode ser consultada no site da *SpringSource*, atesta bem as suas potencialidades e robustez.

	JavaServer Faces	***Google Web Toolkit***	***Spring Framework***	***Apache Struts 2***
Licença	Várias	*Apache License 2.0*	*Apache License 2.0*	*Apache License 2.0*
Versão atual	2.2	2.5.1	3.2.4	2.3.15.1
Data da versão atual	Abril/2013	Março/2013	Agosto/2013	Julho/2013
Data de início do projeto	Março/2004	Maio/2006	Março/2004	Julho/2001
Programador	Vários	Vários	*SpringSource*	*Apache Software Foundation*
Padrão			MVC/Outros	MVC
Acesso BD/ORM			X	
Autenticação			X	
Validações	X	X	X	X
Cache			X	
AJAX	X	X	X	X
Internacionalização	X	X	X	X
Geração de código		X		
Testes unitários			X	X

Tabela 5.3 – Comparativo das funcionalidades dos *frameworks Java*

Embora estejamos perante *frameworks* livres, disponibilizados sob licenças bastante permissivas, nem todos permitem uma participação livre no seu desenvolvimento. Tanto o *JavaServer Faces* como o *Spring Framework* são desenvolvidos por empresas, que também regulam e controlam a sua evolução, sendo disponibilizados gratuitamente à comunidade. Nos restantes *frameworks*, o desenvolvimento pode ser feito pela comunidade, embora a coordenação do projeto seja efetuada por organizações (a *Apache Software Foundation* no caso do *Apache Struts 2* e o GWT *Steering Committee* no caso do *Google Web Toolkit*).

À semelhança do que se verificou na plataforma .NET, nenhum dos *frameworks* inclui, de raiz, um ORM próprio, deixando a possibilidade de escolha aos utilizadores. No caso da plataforma *Java*, existem vários, como, por exemplo, o *Hibernate*, JPA, JDO, *iBatis* ou *Oracle*

TopLink. O facto de alguns *frameworks* não apresentarem determinadas funcionalidades não significa que não seja possível implementá-las no *framework* em causa. Por exemplo, o facto do *JavaServer Faces* não apresentar nenhuma solução de *caching* (nem faz muito sentido que apresente atendendo a que se trata de um *framework* muito vocacionado para a interface do utilizador) não significa que não seja possível de implementar um sistema de caching numa aplicação ou que o mesmo não exista na plataforma *Java*. (*Java Caching System*, *EHCache*, *Jboss Cache* ou *Cache4j*, só para citarmos alguns).

Na tabela 5.4 apresentam-se os fatores de seleção dos *frameworks* e a respetiva avaliação:

	JavaServer Faces	***Google Web Toolkit***	***Spring Framework***	***Apache Struts 2***
Funcionalidades/Dimensão	2	2	5	3
Desempenho	5	5	4	5
Custo	5	5	5	5
Suporte	5	4	5	4
Conhecimento atual	1	1	4	1
Curva de aprendizagem	4	5	4	4
Escalabilidade	3	2	5	5
Ferramentas desenvolvimento	5	5	5	5
Integração com *SQL Server*	5	5	5	5
Tempo de desenvolvimento	2	2	4	3
Multiplataforma	5	5	5	5
Total	**42**	**41**	**51**	**45**

Tabela 5.4 – Avaliação dos fatores de seleção dos *frameworks Java*

A tabela 5.4 permite identificar quatro itens que influenciam substancialmente a pontuação obtida por cada um dos *frameworks*: Funcionalidades/Dimensão, Conhecimento Atual, Escalabilidade e Tempo de Desenvolvimento.

Ao nível das Funcionalidades/Dimensão, a análise aos *frameworks* permitiu aferir que o *Spring Framework* é o mais completo e extenso de todos, contendo soluções para praticamente todas as áreas de uma aplicação. Adicionalmente, facilita a integração com outros *frameworks*, incluindo o JSF e GWT para a camada de apresentação. Isto significa que é perfeitamente possível desenvolver uma aplicação usando o *Spring Framework* e em que na camada visual seja utilizado o *JavaServer Faces*.

Relativamente ao conhecimento atual, também existe uma vantagem clara do *Spring Framework* uma vez que há, na empresa *Macwin*, funcionários com bastante experiência no desenvolvimento de aplicações usando o *framework*. Pelo contrário, para os restantes

frameworks não há existem programadores que possuam alguma experiência de utilização que possa ser considerada relevante.

Por causa da sua dimensão, o *Spring Framework* tem alguma desvantagem no item da curva de aprendizagem. Tratando-se de uma plataforma muito extensa é normal que o tempo necessário para a entender e ser produtivo na sua utilização seja superior. No entanto, atendendo a que existem programadores na *Macwin* com muito conhecimento e experiência na sua utilização, este aspeto negativo é bastante minimizado.

No item do Tempo de Desenvolvimento também se evidencia uma clara vantagem do *Spring Framework*. Ao disponibilizar bibliotecas e classes que abrangem praticamente todas as situações, o desenvolvimento torna-se mais rápido. Mesmo nas situações em que o *Spring* possa, eventualmente, não apresentar uma solução, a facilidade de integração e interação com outras bibliotecas e *frameworks* poderá ajudar e acelerar o processo de desenvolvimento.

Finalmente, quanto ao suporte prestado, todos os *frameworks* contam com uma grande comunidade, e nos respetivos fóruns de discussão a maioria das questões colocadas obtém resposta. Contudo, tanto o *Spring Framework* como o *JavaServer Faces* obtêm uma pequena vantagem uma vez que também contam com o suporte comercial prestado pelas empresas que os desenvolvem ou apoiam,

Face aos aspetos referidos nos parágrafos anteriores, conclui-se que o *framework* mais adequado para o desenvolvimento da aplicação é o *Spring Framework*. O *Apache Struts* também obteve uma boa pontuação e perfilava-se como uma alternativa. No entanto, o facto de não ser tão completo quanto o *Spring* e de não haver, dentro da empresa, programadores que tenham alguma experiência na sua utilização levou a que o *Spring* obtivesse uma melhor classificação.

5.3. PHP

A análise efetuada aos *frameworks Zend*, *CakePHP*, *CodeIgniter*, *Symfony* e *Yii* (ver secção 4.3) permite concluir que há muitas caraterísticas comuns entre eles. Todos implementam o padrão MVC - embora no caso do *Yii* e do *Symfony* existam algumas especificidades - todos disponibilizam as funcionalidades essenciais para o desenvolvimento de aplicações empresariais, todos têm uma comunidade muito vasta e ativa e todos são disponibilizados de forma livre sob licenças bastante permissivas (podendo, como é o caso do *Zend*, optar-se por licenças pagas, o que garantirá algumas vantagens adicionais).

Os fatores onde se verificaram as maiores diferenças e que, por esse motivo, afetarão mais a seleção do *framework* a utilizar, estão relacionados com o suporte prestado, desempenho e expansibilidade. Na tabela 5.5 apresenta-se um resumo das caraterísticas analisadas.

	Zend	***CakePHP***	***CodeIgniter***	***Symfony***	***Yii***
Licença	*New* BSD	MIT	*Apache*/BSD	MIT	*New* BSD
Versão atual	2.2.1	2.3.7	2.1.4	2.3.1	1.1.13
Data da versão atual	Junho/2013	Julho/2013	Julho/2013	Junho/2013	Dezembro/ 2012
Data de início do projeto	Junho/2007	Maio/2007	Fevereiro/2006	Outubro/2005	Janeiro/2008
Programador	*Zend Technologies*	*Cake Software Foundation*	*EllisLab*	*Sensio Labs*	*Yii Software LLC*
Padrão	MVC *Front Controller*	MVC	MVC	MVC	MVC *Active Record*
Acesso BD/ORM	X	X	X	X	X
Autenticação	X	X	X	X	X
Validações	X	X	X	X	X
Cache	X	X	X	X	X
AJAX	X	X	X	X	X
Internacionalização	X	X	X	X	X
Geração de código		X		X	X
Testes unitários	X	X	X	X	X

Tabela 5.5 – Comparativo das funcionalidades dos *frameworks* PHP

Analisando os dados do *Google Trends* (Figura 5.1), verifica-se que o *Yii* tem vindo a obter cada vez mais popularidade tendo um crescimento constante praticamente desde o início do projeto. O mesmo também pode ser aplicado ao *CodeIgniter* que também tem crescido de forma constante ao longo do tempo. Quanto aos restantes *frameworks* verifica-se uma estabilização nos últimos anos, provavelmente em virtude do seu nível de maturidade e também devido ao crescimento de outros *frameworks*.

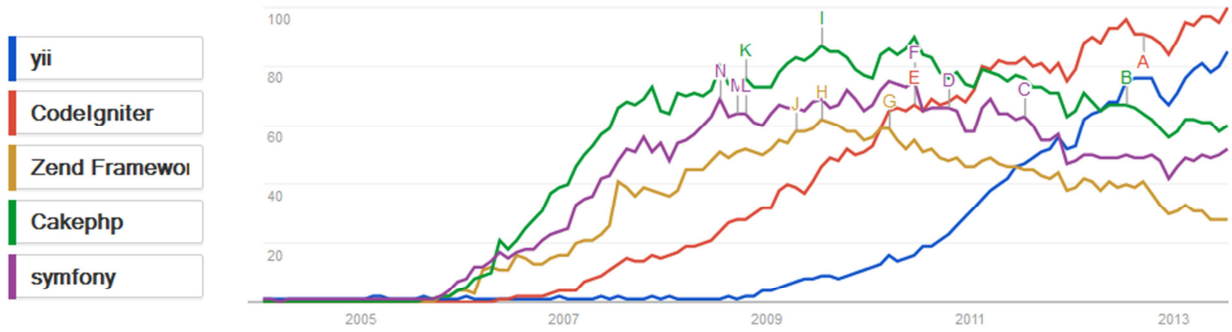


Figura 5.1 – Evolução da popularidade dos *frameworks* PHP [61]

[61]http://www.google.com/trends/explore#q=yii%2C%20CodeIgniter%2C%20Zend%20Framework%2C%20Cakephp%2C%20symfony&cmpt=q, Julho 2013

Ao nível do suporte técnico, o *Zend Framework 2* é o que apresenta maiores vantagens, uma vez que a empresa que efetua o seu desenvolvimento teve, e tem, um papel muito importante no desenvolvimento e evolução do próprio PHP.

Quanto ao desempenho, o *framework* que, de uma maneira geral, apresenta os melhores resultados é o *Yii*[62] (embora os testes efetuados utilizem *frameworks* antigos como concorrentes). Neste aspeto, os *frameworks Zend* e *Symfony* apresentam uma clara desvantagem em relação aos restantes, embora isto possa ser entendido como um custo a pagar pelo facto de se tratarem de *frameworks* maiores e mais complexos. Contudo, a lentidão de um *framework* poderá ser sempre compensada com a aquisição de *hardware* mais rápido e mais recente. Não se tratando da melhor solução será sempre uma forma de minimizar este aspeto negativo.

Finalmente, no que diz respeito à expansibilidade, todos os *frameworks* apresentados demonstram ter essa caraterística, embora tanto o *Zend* como o *Symfony*, por se vocacionarem mais para os ambientes empresariais, aparentem facilitar mais a expansibilidade (tanto ao nível da dimensão como das funcionalidades).

Os 5 *frameworks* podem ser organizados em dois grupos, em que um deles é constituído pelo *Zend Framework 2* e o *Symfony*, que se apresentam como sendo *frameworks* nitidamente vocacionados para o desenvolvimento de aplicações empresariais. Privilegiam as funcionalidades, o suporte e a expansibilidade em detrimento de outras caraterísticas como a facilidade de utilização, o automatismo ou a velocidade no desenvolvimento. No outro grupo temos o *CodeIgniter* e o *CakePHP* que apresentam princípios diferentes. São *frameworks* em que se privilegia a facilidade de utilização e a rapidez no desenvolvimento. Entre estas duas abordagens surge o *Yii* que apresenta caraterísticas que o aproximam da perspetiva empresarial sem, no entanto, prescindir dos aspetos relacionados com a facilidade de utilização e o desenvolvimento rápido.

Na tabela 5.6 apresenta-se uma avaliação dos fatores de seleção usados para escolher o *framework* PHP. Desta tabela destacam-se os *frameworks Zend* e *Yii*. Embora tenham obtido a mesma pontuação esse valor distribui-se de forma diferente pelos diversos itens valorizados. O *Zend* favorece uma perspetiva mais empresarial em detrimento do desempenho e da facilidade de utilização. Em alternativa, o *Yii* é um *framework* mais simples e menos exigente em termos de recursos, que privilegia o desempenho e a facilidade de utilização sem abdicar das caraterísticas inerentes a um *framework* empresarial.

62
http://www.yiiframework.com/performance/, Julho 2013
http://systemsarchitect.net/performance-benchmark-of-popular-php-frameworks/, Julho 2013

	Zend	***CakePHP***	***CodeIgniter***	***Symfony***	***Yii***
Funcionalidades/Dimensão	5	4	4	4	4
Desempenho	2	4	4	3	5
Custo	3	5	5	5	5
Suporte	5	3	3	4	4
Conhecimento atual	5	2	2	2	2
Curva de aprendizagem	3	5	5	3	4
Escalabilidade	5	3	3	5	4
Ferramentas desenvolvimento	5	4	4	4	4
Integração com *SQL Server*	3	3	3	3	3
Tempo de desenvolvimento	3	5	5	3	4
Multiplataforma	5	5	5	5	5
Total	**44**	**43**	**43**	**41**	**44**

Tabela 5.6 – Avaliação dos fatores de seleção

Perante o exposto atrás, recomenda-se a opção pelo *framework Yii* para a realização dos testes de desempenho com base nos seguintes motivos:

- Embora tenha tido a mesma pontuação que o *Zend*, os aspetos em que obteve vantagem são mais importantes para a empresa, principalmente, as questões relacionadas com o desempenho;
- O *Yii* está muito vocacionado para o desenvolvimento de aplicação CRUD, o que o torna mais adequado para o desenvolvimento do *software GM Macwin*;
- A grande integração do *Yii* com o *jQuery* e os componentes visuais facilita muito a criação das interfaces do utilizador;
- O elevado desempenho do *Yii* tende a minimizar este ponto fraco do PHP em relação ao .NET e ao *Java*. Como o PHP, geralmente, é mais lento que as aplicações das outras plataformas, a opção por um *framework* mais rápido não aumenta esta diferença.

6. TESTES DE DESEMPENHO DAS PLATAFORMAS

Nesta secção são reportados alguns testes de desempenho às plataformas analisadas nos capítulos anteriores. O tipo de testes realizados tenta simular as operações que, geralmente, são realizadas em aplicações de gestão. Por este motivo, a maior parte dos testes incidirá sobre a interação entre a plataforma e a base de dados. Os testes a realizar são os seguintes:

- Leitura de 300.000 registos da base de dados e colocação dos mesmos numa lista/vetor/*array*;
- Construção de uma estrutura *map* a partir de uma lista/vetor/*array*;
- Inserção/leitura de registos;
- Ler um ficheiro de texto muito extenso e colocá-lo num vetor/lista;
- Criação de uma lista/vetor/*array* com muitos elementos e em cada elemento efetuar diversos cálculos matemáticos;
- Geração de uma página HTML com muita informação (grelha com muitas linhas/colunas);

Estes testes, embora não simulem todo o tipo de operações que são realizadas por um *software* de gestão, permitem reproduzir, pelo menos, o tipo de operações que costumam ser mais demoradas e exigentes. Cada um dos testes foi realizado 7 vezes em cada plataforma tendo-se excluído o melhor e pior tempos verificados. Todos os valores temporais apresentados são em segundos.

O *hardware* utilizado foi constituído por um servidor, no qual estava instalada a base de dados, e um PC no qual estava instalado todo o restante *software* (Servidores *Web*, IDE, *browsers* e o código fonte). O servidor físico é um *Dell PowerEdge R710*, equipado com um processador *Intel Xeon X5647*, a 2.93 GHz e 32 GB de memória RAM. Nesta máquina existem duas máquinas virtuais a correrem sob o *software Hypervisor VMWare ESX*. A máquina virtual usada nos testes tem as seguintes caraterísticas:

- *Microsoft Windows Server 2003, 64 bits*
- 16 GB RAM
- Disco Rígido de 500GB
- *Microsoft SQL Server 2008 R2 Enterprise Edition, 64bits*

Quanto ao PC onde está instalado o restante *software* é um *Dell Optiplex 7010* com as seguintes caraterísticas:

- *Intel Core i7-3770*, 3.4GHz
- 8GB RAM
- Disco Rígido de 500GB
- *Microsoft Windows 8 Pro, 64bits*

Para os testes da plataforma .NET, o *software* utilizado foi o seguinte:

- *.Net Framework 4.5.1*
- *Visual Studio 2013 Express*
- *C# 5.0*
- *Internet Information Services Express 8.0*
- *ADO.NET provider for SQL Server*

Quanto à plataforma Java, utilizou-se o seguinte:

- *Java 1.7.0_45, 64 bits*
- *Netbeans IDE 7.4*
- *Apache Tomcat 7.0.41*
- *Microsoft JDBC Driver 4.0 for SQL Server*

Para a plataforma PHP optou-se por utilizar o pacote de desenvolvimento XAMPP 1.8.3 o qual junta, no mesmo instalador, o servidor *Apache* e o PHP:

- *PHP 5.5.4*
- *Notepad++ 6.4.1*
- *Apache Server 2.4.4*
- *Microsoft Drivers 3.0 for PHP for SQL Server*[63]

6.1. TESTE DE LEITURA DE REGISTOS

Este teste consiste na leitura de 300.000 registos da base de dados e a sua colocação numa lista em memória. Cada registo é constituído por 22 campos dos seguintes tipos: inteiros, decimais, verdadeiro/falso, texto de tamanho fixo (*char*/*varchar*), texto de tamanho variável (*text/memo*), data/hora e GUID. O código desenvolvido em cada uma das linguagens tem a seguinte sequência:

```
Estabelecer conexão com a base de dados
Executar query

Enquanto existirem registos
      Ler registo
      Colocar registo na lista

Terminar ligação à base de dados
```

[63] Como se utilizou a versão 5.5 do PHP e atendendo a que a Microsoft ainda não disponibilizou uma versão oficial do seu *driver* que funcione com o PHP 5.5, uilizou-se uma versão alterada do *driver* e que está disponível em: http://www.hmelihkara.com/files/php_sqlsrv_55.rar

Na tabela 6.1 apresentam-se os resultados obtidos:

.NET	*Java*	PHP
2,52	3,53	19,10
2,53	3,69	18,95
2,53	3,59	19,15
2,45	3,70	19,13
2,50	3,70	19,16

Tabela 6.1 – Teste de leitura de registos

Neste teste a plataforma .NET obteve o melhor desempenho tendo a plataforma *Java* demorado, em média, mais 40% a concluir a operação e tendo a plataforma PHP demorado mais cerca de 770%. A explicação para esta discrepância nos valores, principalmente nos valores do PHP, pode estar relacionada com o facto de tanto a plataforma .NET como o *SQL Server* serem desenvolvidos pela mesma empresa, o que permitirá uma maior integração e otimização dos recursos. Pelo contrário, o *Java* e o PHP têm que utilizar *drivers* desenvolvidos por terceiros (que poderão ou não estar otimizados) os quais não serão tão eficientes a interagir com o *SQL Server*. Ao longo dos tempos tem sido notório o investimento da *Microsoft* na tecnologia ADO.NET para a interação com a sua base de dados em detrimento de outras tecnologias como o ODBC/JDBC. Outro aspeto que também tem uma grande influência negativa no desempenho do *Java* e do PHP são os campos do tipo data/hora, GUID e texto de tamanho variável.

6.2. TESTE DE CONSTRUÇÃO DE UMA ESTRUTURA *MAP*

Neste teste constrói-se uma estrutura *map* (*Map*, *Dictionary* ou *Array* conforme estejamos na plataforma *Java*, .Net ou PHP, respetivamente) a partir de uma lista contendo 322.814 registos de vendas de artigos. O mapa a construir ficará com 2.538 linhas contendo os totais das quantidades vendidas e o respetivo valor de vendas para cada artigo. A sequência do teste é a seguinte:

```
Estabelecer conexão com a base de dados
Executar query

Enquanto existirem registos
      Ler registo
      Colocar registo na lista

Terminar ligação à base de dados

Para cada elemento na lista
      Verificar se existe no mapa
      Se não existir no mapa
            Insere um novo elemento
      Se existir
            Atualiza a quantidade vendida e o valor de vendas
```

Os valores da duração da operação descrita na secção anterior estão resumidos na tabela 6.2. Em cada linha da grelha são apresentados dois valores: o primeiro corresponde ao tempo gasto com a criação da lista a partir da base de dados e o segundo valor corresponde ao tempo gasto para criar o mapa a partir dos valores existentes na lista:

.NET	***Java***	**PHP**
2,15 0,047	2,52 0,078	4,38 0,531
2,20 0,062	2,47 0,063	4,39 0,540
2,17 0,078	2,60 0,079	4,68 0,533
2,22 0,078	2,52 0,078	4,66 0,543
2,20 0,063	2,47 0,047	4,44 0,535

Tabela 6.2 – Teste de criação de uma estrutura *Map*

Neste segundo teste os valores já se aproximam mais, embora o PHP ainda continue a apresentar resultados piores. Comparando os valores da leitura dos registos com os valores obtidos no teste anterior poderemos, à primeira vista, estar perante uma inconsistência; estamos a ler mais de 300.000 registos e os tempos são substancialmente inferiores, principalmente no *Java* e no PHP. Esta redução significativa dos tempos de leitura dos registos está relacionada com a quantidade e o tipo dos campos. Se no teste anterior tínhamos mais de 20 campos para ler e dos mais variados tipos, no teste atual temos que ler apenas 4 campos sendo dois do tipo texto fixo e os restantes do tipo numérico. Daqui podemos concluir que parte do desempenho inferior do *Java* e PHP na leitura dos registos fica a dever-se aos campos do tipo data/hora, GUID e texto livre. Se utilizarmos campos de texto de tamanho fixo e numéricos a diferença entre a plataforma .NET e as restantes será inferior, embora ainda com vantagem para a primeira.

Em relação aos valores de construção dos mapas, o *Java* e o .NET apresentam valores muito aproximados, enquanto que o PHP continua a apresentar um desempenho substancialmente inferior. Uma possível explicação para esta diferença entre o PHP e as restantes pode estar no objeto utilizado para criar o mapa. Enquanto que na plataforma .NET e *Java* usamos objetos próprios para a construção de mapas de valores (*Dictionary* e *Map* respetivamente) e que, por isso, estarão extremamente otimizados neste processo, na plataforma PHP usamos um *Array* normal em que o índice é o código do artigo.

6.3. TESTE DE LEITURA E INSERÇÃO DE REGISTOS

Neste teste tenta-se simular uma operação em que é necessário inserir um elevado número de registos (10.000) numa tabela, através de um processo de leitura/gravação. Numa primeira fase são lidos 10.000 registos da base de dados contendo cada registo o código de um documento. Na fase seguinte, percorre-se a lista e, para cada código aí armazenado, serão lidos os restantes campos do documento. Depois de se ler toda a informação de um documento, esta será gravada numa outra tabela. A tabela de destino dos registos não tem qualquer índice, chave primária ou chave de integridade referencial. Desta forma tenta-se minimizar o impacto que a atualização destes elementos teria no processo de gravação dos registos. O código desenvolvido para as três plataformas tem o seguinte princípio:

```
Estabelecer conexão com a base de dados
Executar query de obtenção dos códigos dos documentos

Enquanto existirem registos
      Ler registo
      Colocar registo na lista

Para cada elemento na lista
      Ler todos os campos do documento
      Inserir documento

Terminar ligação à base de dados
```

Na tabela 6.3 apresentam-se os tempos de execução desta operação nas diferentes plataformas:

.NET	***Java***	**PHP**
15,85	22,72	17,57
16,20	21,72	18,25
15,90	22,30	18,52
16,05	22,19	17,44
16,25	22,14	17,54

Tabela 6.3 – Teste de leitura e inserção de registos

Ao contrário do que se tem verificado nos testes anteriores, o desempenho do PHP foi muito bom, tendo ultrapassado os valores obtidos na plataforma *Java* e aproximando-se bastante dos valores conseguidos pela plataforma .NET. Uma vez que nos testes anteriores o PHP demonstrou ser muito mais lento na interação com a base de dados e com o manuseamento de listas, uma possível explicação para este desempenho poderá estar na forma como o PHP trabalha com os parâmetros das *queries*. Como se pode constatar pela descrição do teste, numa primeira fase lêem-se os códigos dos documentos da base de dados e colocam-se numa lista. Na segunda fase do teste, percorre-se a lista e para cada código armazenado serão lidos todos os campos do documento e inseridos numa outra

tabela. O processo de inserção dos dados é feito com recurso a uma *query* parametrizada e poderá ser neste passo que o PHP ganha vantagem. No .NET e *Java* os parâmetros são especificados um por um de forma individual mas no PHP os mesmos parâmetros são disponibilizados à *query* num só objeto. Nos dois exemplos seguintes apresenta-se a forma como o C# e o PHP disponibilizam os valores para o preenchimento dos parâmetros da *query* de inserção de um registo (na plataforma *Java* segue-se uma abordagem semelhante à do C#). Como se pode verificar, no C# cada um dos campos a gravar é preenchido de forma individual;

```
paramCodigoGravacao.Value = dr.GetValue(0);
paramTipo.Value = dr.GetValue(1);
paramNumero.Value = dr.GetValue(2);
paramEntidade.Value = dr.GetValue(3);
paramData.Value = dr.GetValue(4);
paramMoeda.Value = dr.GetValue(5);
paramCambio.Value = dr.GetValue(6);
…
…
paramCodigoEnvio.Value = dr.GetValue(33);
paramStatusSincronizacao.Value = dr.GetValue(34);
paramCampoGUID.Value = dr.GetValue(35);

cmdGravacao.ExecuteNonQuery();
```

Pelo contrário, no PHP, os mesmos parâmetros são preenchidos com os valores disponibilizados através do objeto `$reg_doc` e poderá ser aqui que reside o incremento do desempenho do PHP.

```
$stmt_ins_doc = sqlsrv_query($conn, $sql_ins_doc, $reg_doc);
```

Também se verificou, através da utilização do SQL *Server Profiler*, que a base de dados processa os pedidos de forma diferente sempre que estes têm origem no PHP. Tanto para o *Java* como para o C#, o motor da base de dados utiliza internamente uma *stored procedure* chamada `sp_prepexec` mas, para a realização das operações solicitadas pelo PHP utiliza outra *stored procedure* chamada `sp_executesql`. A utilização de procedimentos diferentes para o PHP também poderá ajudar a explicar os resultados obtidos pelo PHP, atendendo a que as duas *stored procedures* referidas têm comportamentos diferenciados [64]. No entanto, importa ressalvar que as duas explicações atrás apresentadas necessitam de mais testes e análises para que possam ser consideradas como válidas.

64
http://technet.microsoft.com/en-us/library/ff848812.aspx, Agosto 2013
http://technet.microsoft.com/en-us/library/ms188001.aspx, Agosto 2013

6.4. TESTE DE LEITURA DE FICHEIROS

Neste teste pretende-se aferir o desempenho das linguagens na leitura e processamento de um ficheiro de texto extenso. Cada linha lida do ficheiro será objeto de tratamento (conversão de texto em valores numéricos e data/hora) e será colocada numa lista/vetor. Esta operação será executada num ficheiro com 400.000 linhas, contendo cada linha um comprimento de 32 carateres. Cada linha representa um registo de entrada ou saída de um funcionário numa organização e contém dados referentes ao relógio, funcionário, sentido (entrada ou saída) e a data/hora da picagem. A operação será executada da seguinte forma:

```
Abrir o ficheiro

Enquanto existirem linhas no ficheiro
      Ler linha
      Processar linha
      Colocar linha na lista

Encerrar o ficheiro
```

Os tempos em segundos desta operação estão refletidos na tabela 6.4:

.NET	*Java*	PHP
0,432	0,823	2,737
0,490	0,814	2,714
0,449	0,820	2,700
0,432	0,814	2,703
0,450	0,813	2,700

Tabela 6.4 – Teste de leitura de ficheiros

Apesar de neste teste não existir qualquer interação com a base de dados, a tendência dos resultados verificada nos testes anteriores mantém-se, tendo o .NET obtido o melhor resultado, o *Java* ficado em segundo e o PHP em último.

6.5. TESTE DE CRIAÇÃO DE LISTA

Este teste consiste na criação de uma lista com 2.000.000 de elementos sendo que cada elemento se trata de um objeto que contém 12 campos numéricos e um campo de texto. Cada objeto, antes de ser adicionado à lista, irá guardar o resultado de diversas operações matemáticas e de conversão de valores numéricos em cadeias de carateres. Assim, o código a executar terá a seguinte sequência:

```
Para N igual a 1 até 2000000
      Calcular log(N)
      Calcular log10(N)
      Calcular exp(N)
      Calcular sqrt(N)
      Calcular cos(N)
      Calcular acos(N)
      Calcular sin(N)
      Calcular asin(N)
      Calcular tan(N)
      Calcular atan(N)
      Calcular N^0.2
      Calcular multiplicação de todos os valores anteriores
      Converte valor anterior numa string
      Adicionar objeto à lista
```

Os resultados obtidos neste teste foram os seguintes:

.NET	*Java*	PHP
3,37	3,49	8,03
3,35	3,49	7,96
3,37	3,49	7,92
3,37	3,49	7,92
3,37	3,48	7,97

Tabela 6.5 – Teste de criação de lista

Os resultados deste teste seguem o mesmo padrão que os resultados do teste anterior embora a diferença entre o .NET e o *Java* seja, agora, muito reduzida e praticamente irrelevante.

6.6. TESTE DE CRIAÇÃO DE TABELA EM HTML

O objetivo deste teste é analisar o desempenho das plataformas na criação/geração de código HTML. Para isso, será criada uma tabela HTML que terá 10.000 linhas e 23 colunas (230.000 células). Os valores apresentados foram obtidos com a utilização do *browser Internet Explorer 10* mas os resultados obtidos com outros navegadores (*Firefox* e *Chrome*) foram idênticos. Esta semelhança de resultados deve-se ao facto de apenas estarmos a medir o tempo gasto na criação da grelha (trabalho efetuado no lado do servidor) e não o tempo que um navegador demora a apresentar uma grelha desta dimensão. Para a criação da grelha desenvolvemos um código com o seguinte padrão:

```
Estabelecer conexão com a base de dados
```

```
Executar query

Enquanto existirem registos
      Ler registo
      Colocar registo na lista

Terminar ligação à base de dados

Gerar código HTML para cabeçalho

Para cada elemento na lista
      Gerar código HTML para uma linha

Gerar código HTML para rodapé
```

O que se está a medir neste teste é apenas a geração da tabela HTML não sendo contabilizado o tempo gasto na interação com a base de dados. Os valores obtidos foram os seguintes:

.NET	***Java***	**PHP**
0,047	0,062	0,081
0,047	0,062	0,080
0,031	0,047	0,081
0,031	0,047	0,082
0,047	0,047	0,083

Tabela 6.6 – Teste de criação de tabela em HTML

Mantém-se o padrão registado em praticamente todos os testes mas agora a diferença de valores é muito menor tanto para o *Java* como para o PHP. Atendendo a que se trata da geração de uma tabela enorme podemos afirmar que as diferenças de tempo verificadas neste teste são insignificantes e totalmente impercetíveis pelo utilizador.

Apesar dos resultados dos testes terem demonstrado que existem diferenças de desempenho entre as plataformas, estas diferenças apenas se manifestaram porque estávamos a trabalhar com grandes volumes de dados. A título de exemplo, para o teste da leitura e processamento de um ficheiro, usamos um documento com 400.000 linhas. Na prática, esta situação nunca se verificará uma vez que nenhuma das organizações que utilizam o *software* GM *Macwin* alguma vez processaram um ficheiro com tal dimensão. Como o maior ficheiro processado pelo *software* até à data não tinha mais do que 5.000 linhas isto significa que as diferenças verificadas entre as plataformas serão impercetíveis e insignificantes. Se o processamento de um ficheiro com 400.000 linhas demorou 0,4s, 0,8s e 2.7s para o .NET, *Java* e PHP respetivamente, o processamento de um ficheiro com 5.000 linhas apresentará valores menores e muito mais aproximados entre si. Este princípio também poderá ser aplicado nos restantes testes uma vez que foram usados volumes de informação grandes com o único propósito de vincar as diferenças de desempenho. A exceção a este princípio reside no teste de leitura de 300.000 registos. Não se tratando de uma operação que seja feita a todo o momento, pode-se referir que estamos

perante uma situação que se verifica com alguma regularidade. Qualquer utilizador pode gerar uma listagem com um volume de dados da mesma grandeza e existem operações que, inclusive, processam volumes de informação maiores. No entanto, este tipo de processamento está agendado para decorrer em horários noturnos por forma a evitar sobrecargas no sistema.

Estes testes permitiram aferir que a plataforma .NET é a que apresenta o melhor desempenho para o tipo de operações selecionadas, ficando o *Java* em segundo lugar e o PHP em terceiro. No entanto, as diferenças verificadas serão esbatidas caso se processem volumes de informação menores. Para volumes de informação menores, as diferenças passarão a ser menores e, provavelmente, impercetíveis aos utilizadores numa utilização diária. Também é importante ressalvar que os resultados obtidos dizem respeito a uma determinada configuração (*Windows*/*SQL Server*), podendo ser diferentes caso os testes sejam realizados em sistemas operativos diferentes ou utilizando outros sistemas de bases de dados.

7. ANÁLISE E SELEÇÃO DOS *FRAMEWORKS* PARA A INTERFACE DO UTILIZADOR

Depois de termos analisado diversas plataformas e *frameworks* que operam, principalmente, no lado do servidor, iremos proceder, neste capítulo, à apresentação de alguns *frameworks* relacionados com o desenvolvimento *web* do lado do cliente e, na parte final, à seleção do que se afigurar mais adequado. Como a aplicação que se pretende desenvolver deverá ser o mais próxima possível de uma aplicação *desktop*, a seleção deverá recair sobre um *framework* para o desenvolvimento de RIA e, preferencialmente, que seja independente da tecnologia usada do lado do servidor. Nesta área temos disponíveis, essencialmente, três metodologias diferentes:

- HTML5
- Utilização de *plugins*
- *Frameworks JavaScript*

A utilização de *plugins* apresenta a vantagem de normalizar as especificidades dos *browsers* mas, ao longo do tempo tem vindo a perder preponderância, em parte por causa dos problemas de segurança verificados em algumas plataformas. Do lado oposto, surge o HTML5 que se trata de uma nova especificação para a linguagem HTML mas que, devido ao facto de ainda nem todos os *browsers* implementarem a totalidade da especificação, não apresenta a maturidade suficiente para que possa ser utilizada em pleno no desenvolvimento de uma aplicação *web*. Os *frameworks JavaScript* já estão disponíveis há vários anos (não podendo, por isso, ser considerada uma tecnologia imatura) e têm vindo, ao longo do tempo, a ganhar um espaço cada vez maior no desenvolvimento de RIA. Adicionalmente, a utilização de um *framework JavaScript* permite uma efetiva separação de conceitos entre o código que corre no servidor e o código que é executado no lado do cliente. Com efeito, como numa arquitetura típica o *framework JavaScript* apenas comunica com o servidor através do intercâmbio de mensagens no formato JSON ou XML, não existe uma dependência entre estes dois níveis da aplicação, o que favorece grandemente a manutenção e a qualidade do código. Do lado do servidor, independentemente da plataforma ou *framework* que for utilizado, apenas terá que ter a capacidade de interpretar os dados que lhe forem fornecidos e de os disponibilizar num formato entendível pelo *framework JavaScript*. Outro aspeto relevante e positivo destes *frameworks* é a existência de um grande leque de possíveis escolhas, com diferentes tipos de funcionalidades, licenciamentos, preços e arquiteturas.

Face ao exposto anteriormente, a camada de apresentação da aplicação será desenvolvida com recurso a um *framework JavaScript* que será selecionado após a análise a ser efetuada nos subcapítulos seguintes.

7.1. *SENCHA EXT JS*

O *Sencha Ext JS*[65] é um *framework JavaScript* que permite a construção de aplicações ricas para a Internet. O *framework* é desenvolvido pela empresa *Sencha* (a qual também desenvolve outros produtos relacionados) e disponibilizado num modelo de duplo licenciamento: uma licença baseada no GPL *License* v3 ao abrigo da qual é possível desenvolver aplicações *Open Source* (uma aplicação desenvolvida com o *Ext JS* licenciado com a GPL v3 terá que ser disponibilizada aos utilizadores juntamente com o respetivo código fonte); e outra licença comercial que permite o desenvolvimento de aplicações sem que seja necessário disponibilizar o código desenvolvido. Para além de se poder distribuir aplicações de código fechado, a aquisição de uma licença comercial garante mais algumas vantagens adicionais destacando-se o acesso a versões intermédias, acesso privilegiado ao suporte, incluindo o suporte telefónico.

A versão mais recente do *Ext JS* é a 4.2.1, tendo sido disponibilizada ao público em geral em Maio de 2013. Para os detentores de uma licença comercial, a *Sencha* disponibilizou, em Outubro de 2013, a versão 4.2.2.

Na lista seguinte apresentam-se as principais caraterísticas e funcionalidades do *framework Sencha Ext JS*:

- As aplicações desenvolvidas com o *Ext JS* podem ser executadas na maioria dos *browsers*, estando assegurada a compatibilidade total com os *browsers* mais utilizados (*Internet Explorer*, *Firefox*, *Chrome*, *Safari* e *Opera*);
- Arquitetura MVC. Embora uma aplicação *Ext JS* possa ser desenvolvida sem respeitar um padrão particular, este *framework* facilita e encoraja o desenvolvimento de aplicações usando o padrão MVC;
- Componentes prontos a utilizar. O *Ext JS* disponibiliza centenas de componentes para que possam ser utilizados pelos programadores. Para além deste elevado número de componentes, os programadores podem expandi-los ou criar novos componentes que satisfaçam e se adequem melhor às suas necessidades. De entre os componentes disponíveis merecem especial destaque as tabelas e grelhas que permitem, de uma forma simples e fácil, a edição na própria grelha, ordenação, agrupamento, formatação de colunas, entre outras caraterísticas. Juntamente com os componentes, são disponibilizados mais de 10 gestores de *layout* que possibilitam a colocação e organização de componentes nas páginas e formulários com base em diversos critérios e regras;
- Gráficos. O *framework* permite a utilização de diversos tipos de gráficos para consulta e apresentação de dados;
- Módulo de gestão de dados. O *Data Package* do *Ext JS* é um poderoso módulo que permite gerir todos os dados utilizados pela aplicação. Este módulo permite simular no *Ext JS* as tabelas, relacionamentos e validações que, normalmente, existem numa base de dados Esta arquitetura também possibilita que com poucas linhas de código sejam lidos e gravados registo numa ou em várias tabelas;

[65] http://www.sencha.com/products/extjs/, Outubro 2013

- *Drag and Drop*. Praticamente todos os componentes do *Ext JS* permitem o controlo de eventos relacionados com a operação de arrastar e largar dados de um objeto para outro;
- Outros aspetos mais genéricos do desenvolvimento de *software*, tais como a localização, testes unitários, acessibilidade e gestão de temas também estão disponíveis dentro deste *framework*;
- Documentação abundante e bem estruturada. A documentação do *Sencha Ext* JS é muito extensa e clara, existindo também diversos artigos e vídeos que auxiliam os programadores. Quanto a livros publicados, estes ultrapassam a dezena de títulos.

Apesar do *Ext JS* ser um produto individual, também pode ser visto como fazendo parte de um pacote de *software* disponibilizado pela *Sencha*. Neste conjunto de *software* destaca-se o *Sencha Architect* (ferramenta de criação interfaces de utilizador tanto para a plataforma *web* como para a plataforma *mobile*), o *Sencha Touch* (*framework* para desenvolvimento de aplicações *web* móveis), o *Sencha Eclipse Plugin*, o *Sencha Touch Charts* e o *Sencha Touch Grids* (gráficos e grelhas interativas, vocacionadas para a utilização em dispositivos móveis).

7.2. *SMARTCLIENT*

O *SmartClient*[66] é um *framework JavaScript* que é desenvolvido e comercializado pela empresa *Isomorphic Software*. Tal como o *Ext JS*, este *framework* permite a criação de RIA, embora também apresente as suas caraterísticas próprias. Quanto ao licenciamento são disponibilizados dois regimes: LGPL ou licença comercial. Caso uma empresa opte pela licença LGPL poderá desenvolver e comercializar livremente *software* que utilize o *framework* mas as funcionalidades disponibilizadas neste regime são muito menores. Caso se opte pela aquisição de uma versão paga, os utilizadores poderão escolher entre três versões diferentes (*Pro*, *Power* e *Enterprise*), com preços diferentes e oferecendo níveis distintos de funcionalidades. Ou seja, ao contrário do *Ext JS* (que disponibiliza todas as funcionalidades do *framework*, independentemente da licença) o *SmartClient* limita as funcionalidades mas dá total liberdade de venda/distribuição do *software* desenvolvido com o seu *framework*. A versão atual do *framework* SmartClient é a 9.0 tendo sido disponibilizada em Julho de 2013. O *SmartClient*, para além de disponibilizar inúmeros componentes visuais e não visuais, também possibilita uma integração destes com código que é executado do lado do servidor. Desta forma consegue-se uma maior integração entre o código que corre no lado do cliente e do servidor mas, em contrapartida, perde-se em independência. A lista seguinte apresenta algumas das caraterísticas que mais se destacam no *SmartClient*:

[66] http://smartclient.com/product/smartclient.jsp, Outubro 2013

- Oferece uma solução completa para o desenvolvimento de aplicações empresariais que vai desde os aspetos relacionados com a camada de apresentação até ao código que é executado do lado do servidor;
- Disponibiliza diversos componentes visuais prontos a utilizar, podendo, caso seja necessário, serem expandidos ou criados novos;
- Geração automática de interfaces do utilizador. O *SmartClient* contempla a geração automática de interfaces, as quais podem, posteriormente, ser trabalhadas ou configuradas. As interfaces de utilizador podem ser criadas livremente ou com base nos campos de tabelas que existam em bases de dados;
- O desenvolvimento do *framework* centra-se, fundamentalmente, na criação de aplicações *web* em detrimento de um *framework* para a criação de páginas *web*;
- Utilização em *offline*. O *framework* permite a criação de aplicações que trabalhem *offline* (ou em cenários em que as ligações sejam intermitentes) através de um mecanismo que regista as alterações efetuadas nos dados enquanto está a operar em modo desligado. Assim que a ligação com o servidor voltar a ser estabelecida, as alterações entretanto produzidas serão enviadas para o servidor;
- Modelo MVC. O *SmartClient* também facilita e prevê o desenvolvimento de aplicações *web* seguindo a arquitetura MVC;
- Apesar do *SmartClient* se integrar naturalmente com a plataforma *Java* (disponibilizando componentes não visuais para o efeito) também permite a utilização de outras plataformas no lado do servidor mediante o recurso a *web services*;
- Escalabilidade e desempenho. O *SmartClient* dispõe de diversas técnicas que permitem um bom resultado em cenários de grandes volumes de informação ou com um grande número de utilizadores em simultâneo. A reutilização de dados armazenados em *cache* (deixando de ser necessário o contacto com o servidor) e a transmissão/receção de dados à medida que forem sendo necessários são dois exemplos de técnicas que favorecem o desempenho e a escalabilidade.

7.3. *DOJO TOOLKIT*

O *Dojo Toolkit*[67] é um *framework JavaScript* de código aberto que é disponibilizado aos programadores sob uma dupla licença: BSD *License* ou *Academic Free License*, *version* 2.1. Com este modelo de licenciamento é possível, na prática, utilizar o *framework* sem qualquer limitação. Com efeito, é possível criar e distribuir aplicações comerciais ou gratuitas sem qualquer limite, assim como alterar o próprio *framework* sem ser necessário qualquer requerimento especial. A propriedade intelectual do *Dojo Toolkit* pertence à *Dojo Foundation*, que é uma organização sem fins lucrativos e que também é responsável pela evolução e promoção do *framework*. A versão atual do *Dojo Toolkit* é a 1.9.1 tendo sido disponibilizada em Junho de 2013.

[67] http://dojotoolkit.org/, Outubro 2013

O *framework* está dividido em módulos diferentes, os quais, em conjunto, constituem o *Dojo Toolkit*. Os módulos são apresentados na lista seguinte:

- ***Dojo***: Este é o módulo base do *framework*, contendo o código fonte que serve de suporte aos restantes módulos e aos componentes visuais;
- ***Dijit***: Este *package* contém componentes visuais do *toolkit*;
- ***Dojox***: Este módulo trata-se de um repositório de código (que podem ser componentes, outros módulos ou utilitários) e que pode pertencer ao *Dojo* ou ao *Digit*, e que ainda não está devidamente preparado para ser utilizado em ambientes de produção. Trata-se de um repositório para versões *Beta*, *Alfa* e *Release Candidate*;
- ***Util***: Neste *package* são colocadas diversas ferramentas que servem de suporte ao *framework*.

Com o *Dojo Toolkit* é possível desenvolver aplicações para as plataformas *web* e *mobile* sendo suportado pela generalidade dos *browsers*, especialmente pelos mais utilizados. Na lista seguinte apresentam-se as caraterísticas mais relevantes do *Dojo Toolkit*:

- Arquitetura MVC. O *framework* está desenvolvido segundo este paradigma, o qual é fomentado no processo de desenvolvimento das aplicações;
- O *framework* disponibiliza um extenso módulo para tratamento de dados que contém, entre outras funcionalidades, a possibilidade de definir relacionamentos, validações, gestão de transações e ligações automáticas entre os componentes e os módulos de dados que os alimentam;
- Possui diversos componentes visuais que podem ser modificados e ajustados às necessidades de cada programador permitindo, também, a possibilidade de criação de novos componentes;
- Todos os componentes estão preparados para a internacionalização, levando em conta, de forma automática, as caraterísticas do local onde estão a ser utilizados;
- Armazenamento local: o *Dojo Toolkit* dispõe de um mecanismo que permite o armazenamento local de dados através de um conjunto de componentes que constituem o *Data Storage*;
- Gráficos. O *Dojo* GFX é um *package* que pertence ao *framework* e que contém um vasto leque de tipos de gráficos prontos a serem utilizados nas aplicações;
- Ferramentas. Estão disponíveis para utilização várias ferramentas que facilitam o trabalho de desenvolvimento com o *Dojo Toolkit*. Uma dessas ferramentas chama-se *Maquetta* e permite a criação de protótipos de aplicações baseadas no HTML5. Também existem diversos *plugins* que facilitam a utilização do *Dojo Toolkit* com outras ferramentas como, por exemplo, o *Aptana Studio* ou o *General Interface*.

7.4. *QOOXDOO*

O *Qooxdoo*[68] é um *framework JavaScript* de código aberto sendo disponibilizado num modelo de duplo licenciamento: LGPL e *Eclipse Public License*. Estas licenças obrigam a que eventuais alterações introduzidas no *framework* sejam disponibilizadas à comunidade sob as mesmas condições de licenciamento. No entanto, caso se desenvolva uma aplicação *web* sem se alterar qualquer aspeto do *framework*, não existirá qualquer obrigatoriedade em colocar a aplicação sob as licenças já referidas. Apesar de se tratar de um *framework* de código aberto, o seu desenvolvimento e coordenação é feito pela empresa *1&1*, que tem estado ligada ao projeto desde o seu início em 2004. A versão atual do *Qooxdoo* é a 3.01, tendo sido lançado em Setembro de 2013. O *framework Qooxdoo* divide-se em quatro grandes grupos:

- ***Website*:** Contém código relacionado com o manuseamento do DOM, controlo de eventos, permitindo uma elevada abstração em relação aos *browsers*;
- ***Mobile*:** Módulo vocacionado para o desenvolvimento de aplicações móveis;
- ***Desktop*:** Módulo que permite a criação de aplicações *web* ricas;
- ***Server*:** Este módulo está relacionado com o desenvolvimento de código que será executado do lado do servidor, podendo o código resultante ser executado nas plataformas *JavaScript Node.js* ou *Rhino*.

Transversalmente a estes quatro grupos podemos identificar mais dois (*Core* e *Tooling*) que servem de suporte. O módulo *Core* é usado como base para todos os outros. É sobre este que todos os restantes módulos são desenvolvidos. O *Tooling* consiste num conjunto de ferramentas que podem ser utilizadas na criação e manutenção das aplicações criadas com o *framework*. Relativamente ao *Qooxdoo* salientam-se os seguintes aspetos:

- Possui uma extensa coleção de componentes visuais, os quais, à semelhança do que sucede nos restantes *frameworks*, podem ser expandidos ou usados em conjunto com outros para formarem estruturas mais complexas;
- Uma aplicação criada com o *Qooxdoo* pode ser considerada uma aplicação isolada ou ser inserida numa página *web* já existente. Embora esta caraterística também possa ser alcançada noutros *frameworks*, fazê-lo no *Qooxdoo* poderá ser mais simples, principalmente porque contém componentes não visuais que controlam o ambiente onde está a ser executado o código do *framework*;
- Pode ser utilizado nos principais *browsers* e *JavaScript Engines*;
- O código fonte é totalmente desenvolvido em *JavaScript*, não sendo necessário conhecimentos de HTML, CSS e DOM para se poder desenvolver aplicações;
- Desenvolvido usando diversos paradigmas de programação, sendo totalmente escrito com o recurso aos princípios da programação orientada a objetos; suporte ao desenvolvimento com base em eventos, contendo, ainda, módulos para testes unitários e testes funcionais;
- Suporte integrado para o desenvolvimento de aplicações internacionais (*i18n* e *l10n*).

[68] http://qooxdoo.org/, Outubro 2013

7.5. *jQUERY USER INTERFACE*

O *jQuery UI*[69] é uma biblioteca *JavaScript* de código aberto desenvolvido sobre o *framework jQuery*. É disponibilizado sob a licença MIT sendo a versão mais recente a 1.10.3, disponibilizada em Maio de 2013. Embora se trate de um projeto de código aberto e distribuído sob uma licença muito permissiva, o projeto é gerido pela *jQuery Foundation* que coordena todo o trabalho relacionado com a evolução e promoção do *framework*.

A biblioteca é constituída, essencialmente, por um conjunto de componentes visuais, efeitos, interações e temas. Em comparação com os outros *frameworks* analisados neste capítulo, o *jQuery UI* é o que apresenta o menor número de funcionalidades. Apesar deste aspeto, o *jQuery UI* é um projeto que se encontra associado a outros de grande dimensão (*jQuery*) ou com um elevado potencial de crescimento (*jQuery Mobile*). As caraterísticas mais significativas do *jQuery UI* encontram-se resumidas na lista seguinte:

- Disponibiliza diversos componentes, sendo de destacar os componentes *Accordion*, *Tabs* e *DatePicker*. Contudo, a biblioteca não disponibiliza, de forma nativa, componentes baseados em listas, tais como grelhas, árvores ou listas;
- Existe uma comunidade muito ativa de programadores que tentam colmatar as falhas e os pontos fracos do *framework*. Neste sentido, existe um elevado número de *plugins* disponíveis para utilização que expandem muito o *framework*;
- Possui diversos efeitos e interações que permitem configurar a forma como os componentes se apresentam e como interagem com os utilizadores;
- O aspeto das aplicações criadas com o *jQuery UI* pode ser altamente configurável sem, no entanto, perder a uniformidade e coerência. Isto é conseguido através de uma elevada quantidade de temas disponíveis;
- O *jQuery UI* não dispõe de qualquer componente para tratamento de dados nem componentes que possam interagir de forma automática com fontes de dados.

Apesar de se tratar de um projeto muito interessante, que se insere num grupo de projetos com uma importância considerável, o *jQuery UI* carece de algumas funcionalidades que o deixam em clara desvantagem quanto comparado com os restantes *frameworks* já analisados. Nem mesmo o facto de existirem muitos *plugins* disponíveis colmata esta situação. O aspeto negativo da utilização de muitos *plugins* para adicionar funcionalidades ao *framework* é que se fica muito dependente de projetos terceiros que não oferecem muitas garantias de continuidade e evolução. Por outro lado, a utilização de componentes com origens muito diversificadas pode causar alguma inconsistência na apresentação (apesar do *jQuery UI* possuir meios para atenuar essa situação). Esta desvantagem torna-se mais evidente porque estamos a analisar os *frameworks* para uma futura utilização no desenvolvimento de uma aplicação *web* que visa substituir uma aplicação *desktop*. Noutros cenários com outro tipo de exigências, as desvantagens e lacunas do *jQuery UI* não serão tão evidentes.

[69] http://jqueryui.com/, Outubro 2013

7.6. OUTROS *FRAMEWORKS*

No capítulo 4 foram abordados alguns *frameworks* que, pela sua natureza, também podem ser considerados na análise que está a ser feita neste capítulo. Os *frameworks* em causa são o *Microsoft Silverlight*, o *Google Web Toolkit* e o *JavaServer Faces*.

Quanto ao *Microsoft Silverlight* é excluído de uma possível escolha porque se trata de um *framework* que requer a instalação de um *plugin* no navegador e, tal como referimos no início do capítulo, esta abordagem está a entrar em declínio e não se pretende que a mesma seja utilizada no desenvolvimento da aplicação. Por outro lado, o *plugin* para correr o *Silverlight* não está disponível para todos os navegadores nem em todos os sistemas operativos.

Em relação ao *Google Web Toolkit* e ao *JavaServer Faces*, embora se tratem de *frameworks* que não requerem a instalação de *plugins*, estão muito relacionados com as tecnologias que correm no servidor, o que não ajuda na separação de conceitos e de camadas que se pretende obter. Enquanto que um *Ext JS* ou um *Dojo Toolkit* é totalmente independente da tecnologia que está alojada no servidor, o *Google Web Toolkit* e o *JavaServer Faces* estão muito interligados, o que os coloca em desvantagem relativamente aos *frameworks* desenvolvidos usando apenas o *JavaScript*.

7.7. SELEÇÃO DO *FRAMEWORK*

Com base na análise efetuada ao longo deste capítulo, verificamos que existem quatro *frameworks* que se adequam perfeitamente às necessidades da aplicação (*Ext JS*, *SmartClient*, *Dojo Toolkit* e *Qooxdoo*). Para cada um destes *frameworks* foram avaliados os seguintes critérios: suporte, custo de aquisição e funcionalidades disponibilizadas (e que sejam relevantes para a aplicação a desenvolver). Na tabela 7.1, apresentam-se os valores atribuídos a cada um dos fatores de seleção, em que o valor 5 corresponde à melhor avaliação. A atribuição destes valores baseou-se, essencialmente, na análise efetuada nos postos anteriores.

	Ext JS	***SmartClient***	***Dojo Toolkit***	***Qooxdoo***
Suporte	5	5	2	1
Custo	2	1	5	5
Funcionalidades	5	5	3	3
Total	**12**	**11**	**10**	**9**

Tabela 7.1 – Avaliação dos fatores de seleção dos *frameworks* da camada de apresentação

Os resultados obtidos na tabela 7.1 refletem que estamos perante ferramentas equivalentes em que a diferença se faz, apenas, nos pormenores. Os *frameworks* comerciais apenas conseguiram uma valorização superior porque a *Macwin* valoriza o suporte e, neste campo, o facto de existirem empresas cujo negócio é o próprio *framework* dá mais garantias de um suporte personalizado e imediato. Por este motivo, os *frameworks* de código aberto ficaram, automaticamente, em desvantagem.

Quanto aos dois *frameworks* comerciais, o mais valorizado acaba por ser o *Ext JS* mas apenas em virtude do seu custo ser inferior. Por 2.885€[70] é possível adquirir um pacote *Premium* que inclui licenças para 5 programadores, suporte técnico via telefone, correção de *bugs* de emergência e garantia de resposta em 48 horas. Relativamente ao *SmartClient*, se optarmos pela edição *Pro*, o custo será de 745€[71] por programador (sendo que para as versões *Power* e *Enterprise* o custo será de 1.950€ e 2.500€ mas as funcionalidades que estas oferecem não justificam a sua aquisição) aos quais se deverá adicionar um custo (não apresentado na página *web* da empresa) para a aquisição do suporte técnico. Outro aspeto que, embora não tenha sido levado em conta, poderia, em caso de empate, servir como fator de desempate é o facto de a *Macwin* utilizar uma aplicação de gestão de projetos chamada *Celoxis*[72] e que foi desenvolvida usando o *Ext JS*. Como o aspeto e o funcionamento do *Celoxis* é bastante agradável e apresenta muita qualidade, este pode ser considerado como um aspeto positivo para o *Ext JS*, embora não seja sinónimo de que não seria possível construir uma aplicação com as mesmas caraterísticas utilizando o *SmartClient*.

Assim, face ao exposto nesta análise documental de algumas das plataformas disponíveis no mercado, recomenda-se a utilização do *Sencha Ext JS* para o desenvolvimento da interface de utilizador e de todo o código que será executado do lado do cliente.

[70] https://www.sencha.com/store/extjs, Outubro 2013
[71] http://smartclient.com/product/editions.jsp, Outubro 2013
[72] http://www.celoxis.com, Outubro 2013

8. CONCLUSÃO

Este projeto tinha como principal objetivo a identificação da plataforma/*framework* mais adequada para o desenvolvimento de uma aplicação de gestão *web* que substituirá o *software* GM *Macwin*, disponível para a plataforma *desktop*. Para darmos resposta a esta questão efetuamos uma análise documental a alguns *frameworks* em cada uma das plataformas (.NET, *Java* e PHP) e desenvolvemos testes de desempenho que simularam as operações que são executadas pelo *software* atual com mais frequência ou que sejam mais demoradas. Para a parte visual da aplicação foram analisadas algumas soluções tecnológicas e, dentro da tecnologia selecionada, foi estudada a documentação de algumas bibliotecas atualmente disponíveis.

As análises efetuadas nos capítulos anteriores permitem concluir que qualquer uma das três plataformas (.NET, Java e PHP) poderá ser utilizada no desenvolvimento da versão *web* da aplicação GM *Macwin*. Analisando as plataformas pela perspetiva do desempenho, verifica-se que existem algumas diferenças mas estas apenas se fazem sentir em situações muito específicas e com volumes de informação pouco usuais neste tipo de *software*. Se a análise se centrar nas funcionalidades, verifica-se que todas elas disponibilizam as funcionalidades necessárias para o desenvolvimento de uma aplicação de gestão, dando resposta às solicitações e necessidades deste tipo de aplicações. Em todas as plataformas existem diversos *frameworks* que podem ser utilizados pelos programadores com o objetivo de simplificar, acelerar e melhorar o processo de desenvolvimento do *software*. Qualquer uma das plataformas e *frameworks* foram e continuam a ser utilizadas no desenvolvimento dos mais variados tipos de *software* e das mais variadas dimensões e complexidades.

Para dar resposta ao objetivo deste projeto, sugere-se a utilização da plataforma *Java* e do *Spring Framework* (com base na comparação efetuada no capítulo de seleção dos potenciais *frameworks*). A opção pela plataforma *Java* baseou-se nos seguintes critérios:

- Custos de licenciamento de *software*;
- Qualidade do código produzido;
- Desempenho;
- *Know-How* atualmente existente na empresa.

No item dos custos de licenciamento, tanto a plataforma *Java* como a PHP levem uma clara vantagem relativamente ao .NET uma vez que, ao contrário desta última, é possível desenvolver uma aplicação completa sem despender qualquer valor monetário em licenciamentos de *software*. No que respeita ao *Java*, esta afirmação poderá não ser totalmente correta se atendermos a que, para grandes volumes de informação, os servidores de aplicações gratuitos poderão não dar a melhor resposta. Contudo, para os volumes de informação que os clientes da *Macwin* geram e para a capacidade de processamento necessária, as soluções *open source* atualmente disponíveis apresentam-se como suficientes. Neste critério, a plataforma .NET tem uma grande desvantagem porque existirão sempre, pelo menos, custos de aquisição e licenciamento das ferramentas de

desenvolvimento (embora a *Microsoft* disponibilize versões *Express* das suas ferramentas de desenvolvimento de forma gratuita, estas versões têm algumas limitações que dificultam muito a utilização por uma equipa de desenvolvimento composta por vários programadores).

Relativamente ao critério da qualidade de código produzido, as vantagens centram-se na plataforma *Java* e .NET. Em qualquer uma das plataformas é possível escrever código bem estruturado que facilite a manutenção e evolução de aplicações mas, atendendo à natureza da linguagem PHP, a probabilidade de se criar código de fraca qualidade é maior, o que dificulta o desenvolvimento em equipa.

Ao nível do desempenho, os testes efetuados demonstraram que a plataforma .NET é a mais rápida, ficando o *Java* em segundo e o PHP em terceiro. Contudo, as diferenças de desempenho (principalmente entre o .NET e o *Java*) foram muito pequenas podendo-se tornar insignificantes perante volumes de informação menores. Assim, neste critério, podemos colocar ao mesmo nível o .NET e o *Java* ficando o PHP numa posição ligeiramente inferior.

Finalmente, no critério do *know-how* existente na empresa, a vantagem pende, claramente, para a plataforma *Java* uma vez que a equipa de desenvolvimento integra vários programadores com conhecimentos da plataforma *Java* e experiência no desenvolvimento de aplicações *web*, incluindo na criação de aplicações usando o *Spring Framework*.

Como em cada um dos critérios de seleção a plataforma *Java* apresentou quase sempre vantagens, a opção por esta plataforma afigura-se como sendo a mais adequada.

Para o desenvolvimento da parte visual da aplicação recomenda-se a utilização do *Sencha Ext JS*. Este *framework* apresenta diversas funcionalidades que serão uma mais-valia no momento do desenvolvimento, destacando-se a enorme quantidade de componentes, o módulo de gestão de dados e uma documentação extensa e de qualidade. A estas caraterísticas e vantagens deve juntar-se um pacote de suporte que garante um apoio rápido e eficiente caso venha a ser necessário. Apesar de não se tratar de *software* gratuito (embora seja *open source*) o custo de aquisição e do suporte está dentro dos valores considerados razoáveis pela *Macwin.*, a qual, no presente, despende quantias muito mais elevadas para a aquisição de licenças e suporte da ferramenta de desenvolvimento *Omnis Studio*.

Baseados nas diversas análises efetuadas no decurso deste projeto, estamos em condições de afirmar que a combinação *Java*/*Spring Framework*/*Ext JS* é a solução mais adequada para a empresa. Por um lado, permite baixar os custos de desenvolvimento e aproveitar o conhecimento que já existe na sua equipa e, por outro lado, permitirá que a empresa desenvolva uma solução *web* robusta, flexível, escalável e com uma experiência de utilização adequada para uma aplicação de gestão.

Após a elaboração do presente trabalho surgem, naturalmente, novas perspetivas a explorar em desenvolvimentos futuros. De seguida, apresentam-se algumas possíveis tarefas nesse sentido:

- Analisar outras plataformas e linguagens como, por exemplo, *Python, Groovy* e *Ruby* e respetivos *frameworks* (p. ex. *Django, Grails* e *Ruby On Rails*);
- Realizar testes dos *frameworks* para a interface do utilizador, nomeadamente no que se refere ao seu desempenho no carregamento das bibliotecas e dados em diversos *browsers*;
- Analisar outras plataformas que tenham ganho relevância nas comunidades de utilizadores e com atividade recente, como, por exemplo, o *Vaadin*, *Kendo* UI, *AngularJS*, ZK, *Bootstrap* e *JavaFX*;
- Realizar testes de desempenho em diferentes ambientes de desenvolvimento como, por exemplo, num equipamento com sistema operativo *Linux*;
- Estudar a possibilidade de se optar por outros sistemas de gestão de bases de dados, nomeadamente o *PostgreSQL* e o *Oracle Database*, e realizar testes de desempenho.

REFERÊNCIAS

[Allen09] Allen, R., Lo, N., Brown, S., *Zend Framework in Action*, Manning, 2009

[Anderson12] Anderson, C., Pro Business Applications with Silverlight 5, Apress, 2012

[Bari08] Bari, A., Syam, A., *CakePHP Application Development,* Packt Publishing, 2008

[Brown08] Brown, D., Davis, C. M., Stanlick, S., *Struts 2 in Action*, Manning, 2008

[Brown12] Brown, P., *Silverlight 5 in Action*, Manning, 2012

[CSS3MOD] *CSS current work & how to participate*, http://www.w3.org/Style/CSS/current-work.en.html, Setembro 2013

[DOM] *What is the Document Object Model?*, http://www.w3.org/TR/DOM-Level-3-Core/introduction.html, Setembro 2013

[DOMTR] *Document Object Model (DOM) Technical Reports*, http://www.w3.org/DOM/DOMTR, Setembro 2013

[Ecma262] *Standard ECMA-262 - ECMAScript Language Specification - Edition 5.1 (June 2011),* http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, Outubre 2013

[Esposito09] Esposito, D., *Comparing Web Forms And ASP.NET MVC, MSDN Magazine - July 2009 Issue*, 2009

[Esposito10] Esposito, D., *Programming Microsoft ASP.NET MVC*, Microsoft Press, 2010

[Esposito11] Esposito, D., *Programming Microsoft ASP.NET 4*, Microsoft Press, 2011

[Freeman11] Freeman, A., Sanderson, S., *Pro ASP.NET MVC 3 Framework*, 3.ª Ed., Apress, 2011

[Fowler02] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., Stafford, R., *Patterns of Enterprise Application Architecture*, Addison Wesley, 2002

[Galloway12] Galloway, L., Haack, P., Wilson, B., Allen, K., *Professional ASP.NET MVC 4*, Wiley, 2012

[Gamma95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[Garret05] *Ajax: A New Approach to Web Applications,* http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/, Outubro 2013

[Gilmore10] Gilmore, W., *Beginning PHP and MySQL: From Novice to Professional*, 4.ª Ed., Apress, 2010

[Golding08] Golding, D., *Beginning CakePHP - From Novice to Professional*, Apress, 2008

[Goncalves10] Goncalves, A., *Beginning Java EE 6 Platform with Glassfish*, 2.ª Ed., Apress, 2010

[Goodman07] Goodman, D., *JavaScript Bible*, 6.ª Ed., Wiley Publishing, Inc., 2007

[Griffiths10] Griffiths, A., *CodeIgniter 1.7 Professional Development*, Packt Publishing, 2010

[Hanselman13] Hanselman, S., Gaylord, J., Wenz, C., Rastogi, P., Miranda, T., *Professional ASP.NET 4.5 in C# and VB*, John Wiley & Sons, Inc., 2013

[Hanson13] Hanson, R., Tacy, A., Essington, J., Tokke, A., *GWT In Action*, 2.º Ed., Manning, 2013

[HTML401] *HTML 4.01 Specification - World Wide Web Consortium,* http://www.w3.org/TR/REC-html40/, Setembro 2013

[IBMnewto] *New to Web Development (IBM Developer Works),* http://www.ibm.com/developerworks/web/newto/, Abril 2013

[Jacobi06] Jacobi, J., Fallows, J., *Pro JSF and Ajax – Building Rich Internet Components,* Apress, 2006

[JavaEETut12] *The Java EE 6 Tutorial*, Oracle, 2012

[Johnson05] Johnson, R., Hoeller, J., Arendsen, A., Risberg, T., Sampaleanu, C., *Professional Java Development with the Spring Framework*, Wiley Publishing, Inc., 2005

[Johnson13] Johnson, R., et al., *Spring Framework Reference Documentation, 3.2.4 Release*, 2013, http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/pdf/spring-framework-reference.pdf, Maio 2013

[Knudsen05] Knudsen, J., Niemeyer, P., *Learning Java,* 3.ª Ed., O'Reilly, 2005

[Lair12] Lair, R., *Beginning Silverlight 5 in C#,* Apress, 2012

[Lerdorf06] Lerdorf, R., MacIntyre, P., Tatroe, K., *Programming PHP*, 2.ª Ed., O'Reilly, 2006

[MacDonald12] MacDonald, M., *Pro Silverlight 5 in C#,* Apress, 2012

[MacIntyre10] MacIntyre, P., *PHP: The Good Parts*, O'Reilly, 2010

[Mann05] Mann, K., *JavaServer Faces in Action*, Manning, 2005

[Martins09] Martins, F., *Java 6 e Programação Orientada pelos Objectos*, FCA - Editora Informática, 2009

[Mateu10] Mateu, C., Mas, J., *Introduction to Web Applications Development*, Free Technology Academy, 2010

[McPeak07] McPeak, J., Fawcett, J., Zakas, N., *Professional Ajax,* 2.ª Ed. Wiley Publishing, Inc., 2007

[MDNDHTML] DHTML, https://developer.mozilla.org/en/docs/DHTML, Setembro 2013

[Merkel10] Merkel, D., *Expert PHP 5 Tools*, Packt Publishing, 2010

[Morris11] Felke-Morris, T., *Web Development and Design Foundations wih XHTML*, 5.ª Ed., Addison-Wesley, 2011

[Nagel12] Nagel, C., Evjen, B., Glynn, J., Watson, K., Skinner, M., *Professional C# 2012 and .NET 4.5*, John Wiley & Sons, Inc., 2012

[Palermo12] Palermo, J., Bogard, J., Hexter, E., Hinze, M., Skinner, J., *ASP.NET MVC 4 in Action*, Manning, 2012

[Pordel08] Pordel, M., Yekeh, F., *JSF vs ASP.NET, What are their limits?*, IDT Workshop on Interesting Results in Computer Science and Engineering, Mälardalen University, Sweden, 2008

[Porebski11] Porebski, B., Przystalski, K., Nowak, L., *Building PHP Applications with Symfony, CakePHP, and Zend Framework*, Wiley Publishing, Inc., 2011

[Reid13] Reid, J., Valentine T., *JavaScript Programmer's Reference*, Apress, 2013

[RFC1866] *Hypertext Markup Language - 2.0*, http://tools.ietf.org/html/rfc1866, Setembro 2013

[RFC2318] *The text/css Media Type*, http://tools.ietf.org/html/rfc2318, Setembro 2013

[RFC2616] *Hypertext Transfer Protocol - HTTP/1.1*, http://tools.ietf.org/html/rfc2616, Agosto 2013

[RFC3986] *Uniform Resource Identifier (URI): Generic Syntax,* http://tools.ietf.org/html/rfc3986, Setembro 2013

[RFC4627] *The application/json Media Type for JavaScript Object Notation (JSON),* http://tools.ietf.org/html/rfc4627, Setembro 2013

[Roughley07] Roughley, I., *Practical Apache Struts2 Web 2.0 Projects*, Apress, 2007

[Shan06] Shan, T., Hua, W., *Taxonomy of Java Web Application Frameworks*, IEEE International Conference on e-Business Engineering, 2006

[Shklar03] Shklar, L., Rosen, R., *Web Application Architecture - Principles, protocols and practices*, John Wiley & Sons Ltd, 2003

[Smeets08] Smeets, B., Boness, U., Bankras, R., *Beginning Google Web Toolkit: From Novice to Professional*, Apress, 2008

[Suehring09] Suehring, S., Converse, T., Park, J., *PHP 6 and MySQL 6 Bible*, Wiley Publishing, Inc., 2009

[Suehring13] Suehring, S., *JavaScript Step by Step*, 3.ª Ed., Microsoft Press, 2013

[Teague06] Teague, J., *Visual Quickstart Guide CSS, DHTML, and Ajax*, 4.ª Ed., Peachpit Press, 2006

[Thomas01] Thomas, S., *HTTP Essentials Protocols for Secure, Scaleable Web Sites*, 2001, John Wiley & Sons, Inc.

[Troelsen12] Troelsen, A., *Pro C# 5.0 and the .NET 4.5 Framework*, 6.ª Ed., Apress, 2012

[Ullman07] Ullman, C., Dykes, L., *Beginning Ajax*, Wiley Publishing, Inc., 2007

[Upton07] Upton, D., *CodeIgniter for Rapid PHP Application Development*, Packt Publishing, 2007

[Walls11] Walls, C., *Spring in Action*, 3.ª Ed., Manning, 2011

[Wilton10] Wilton, P., McPeak, J., *Beginning JavaScript*, 4.ª Ed., Wiley Publishing, Inc., 2010

[Winesett12] Winesett, J., *Web Application Development with Yii and PHP*, 2.ª Ed., Packt Publishing, 2012

[Wolff08] Wolff, E., *Spring - A Manager's Overview*, SpringSource, 2008, http://assets.spring.io/drupal/files/Spring_A_Managers_OverviewVer05.pdf, Maio 2013

[XHR] *The XMLHttpRequest Object*, http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/, Setembro 2013

[XHTML10] *XHTML 1.0 The Extensible HyperText Markup Language,* http://www.w3.org/TR/xhtml1/, Setembro 2013

[XHTML11] *XHTML 1.1 - Module-based XHTML*, http://www.w3.org/TR/2001/REC-xhtml11-20010531/, Setembro 2013

[XML10] *Extensible Markup Language (XML) 1.0 (Fifth Edition),* http://www.w3.org/TR/REC-xml/, Setembro 2013

[XML11] *Extensible Markup Language (XML) 1.1 (Second Edition),* http://www.w3.org/TR/xml11/, Setembro 2013

[Zakas12] Zakas, N., *Professional JavaScript for Web Developers*, 3.ª Ed., John Wiley & Sons, Inc., 2012

[Zaninotto07] Zaninotto, F., Potencier, F., *The Definitive Guide to Symfony*, Apress, 2007

Outras fontes de informação consultadas

Livros

MacDonald, M., *Beginning ASP.NET 4.5 in C#*, Apress, 2012

Shepherd, G., *Microsoft ASP.NET 4 Step by Step*, Microsoft Press, 2010

Walther, S., Hoffman, K., Dudek, N., *ASP.NET 4 Unleashed*, SAMS, 2010

Spaanjaars, I., Beginning *ASP.NET 4 in C# and VB*, Wiley Publishing, Inc., 2010

Chadwick, J., Snyder, T., Panda, H., *Programming ASP.NET MVC 4*, O'Reilly, 2012

Walther, S., *ASP.NET MVC Framework Unleashed*, SAMS, 2009

Arnold, K., Gosling, J., Holmes, D., *The Java Programming Language*, 4.ª Ed., Addison Wesley Professional, 2005

Gosling, J., et al., *The Java Language Specification*, 3.ª Ed., Addison Wesley Professional, 2005

Crawford, W., Farley, J., *Java Enterprise in a Nutshell*, 3.ª Ed., O'Reilly, 2005

Bodoff, S., et al., *The J2EE Tutorial*, 2.ª Ed., Addison Wesley Professional, 2004

Ho, C., Harrop, R. *Pro Spring 3*, Apress, 2012

Johnson, R., *Expert One-on-One J2EE Design and Development*, Wiley Publishing, Inc., 2003

Goodwill, J., *Mastering Jakarta Struts*, Wiley Publishing, Inc., 2002

Sklar, D., *Learning PHP 5*, O'Reilly, 2004

David Flanagan, *JavaScript: The Definitive Guide*, 6.ª Ed., O'Reilly, 2011

Artigos/Outros documentos

Vuksanovic, I., Sudarevic, B., *Use of Web Application Frameworks in the Development of Small Applications*, MIPRO 2011

Lancor, L., Katha, S., *Analyzing PHP Frameworks for Use in a Project-Based Software Engineering Course, 2013*

Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A., *Web Engineering: A New Discipline for Development of Web-based Systems*

Hightower, R., *Getting Started With JavaServer Faces 1.2, Part 1: Building Basic Applications*, 2007, IBM DeveloperWorks

Pack, R., *Choosing Your Java Based Web Framework: A Comparison*, JavaOne Conference, 2008

Raible, M., *Comparing Java Web Frameworks JSF, Spring MVC, Stripes, Struts 2, Tapestry and Wicket*, ApacheCon 2007

CakePHP Cookbook Documentation - Release 2.x, Cake Software Foundation, 2013

Zend Framework 2 Documentation, Release 2.2.1 dev, Zend Techologies Ltd., 2013

The Book for Symfony 2.3, SensioLabs, 2013

The Components Book for Symfony 2.3, SensioLabs, 2013

Páginas *web*

ASP.NET Web Forms, http://www.asp.net/web-forms, Fevereiro 2013

ASP.NET MVC, http://www.asp.net/mvc, Fevereiro 2013

Silverlight, http://msdn.microsoft.com/en-us/silverlight/bb187358.aspx, Fevereiro 2013

Getting Started with the .NET Framework, http://msdn.microsoft.com/en-us/library/hh425099, Fevereiro 2013

Silverlight Architecture, http://msdn.microsoft.com/pt-pt/library/bb404713, Fevereiro 2013

Oracle Java Timeline, http://oracle.com.edgesuite.net/timeline/java/, Março 2013

Struts2 jQuery Plugin, http://code.google.com/p/struts2-jquery/, Maio 2013

PHP Manual, http://php.net/manual/en/index.php, Junho 2013

PHP Extension and Application Repository, http://pear.php.net/, Junho 2013

PHP Extension Community Library, http://pecl.php.net/, Junho 2013

W3Techs - World Wide Web Technology Surveys, http://w3techs.com/, Junho 2013

Zend Framework, http://framework.zend.com/, Junho 2013

PHP Frameworks, http://www.phpframeworks.com/, Junho 2013

Best Web Frameworks: Compare PHP Frameworks, http://www.bestwebframeworks.com/compare-web-frameworks/php/, Junho 2013

PHP performance and memory usage, http://we-love-php.blogspot.pt/2012/10/php-framework-comparison.html, Julho 2013

Web Framework Benchmarks, http://www.techempower.com/benchmarks/#section=intro&hw=i7&test=db, Julho 2013