INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

ESTG

ABORDAGENS AO DESENVOLVIMENTO DE SOFTWARE USANDO FERRAMENTAS DE INTELIGÊNCIA ARTIFICIAL

Mozart Teixeira de Brito

2025

Escola Superior de Tecnologia e Gestão

ABORDAGENS AO DESENVOLVIMENTO DE SOFTWARE USANDO FERRAMENTAS DE INTELIGÊNCIA ARTIFICIAL
Mozart Teixeira de Brito

Instituto Politécnico de Viana do Castelo

Mozart Teixeira de Brito

ABORDAGENS AO DESENVOLVIMENTO DE SOFTWARE USANDO FERRAMENTAS DE INTELIGÊNCIA ARTIFICIAL

Mestrado em Engenharia Informática

Trabalho de Projeto efetuado sob a orientação de

Professor Doutor Jorge Manuel Ferreira Barbosa Ribeiro

Professor Doutor António Miguel Rosado da Cruz

Fevereiro de 2025

RESUMO

O presente estudo investiga a interseção entre Desenvolvimento de Software (DS) e Inteligência Artificial (IA), explorando as transformações que a IA tem promovido nas práticas e metodologias tradicionais da engenharia de software. Com o avanço de modelos de Aprendizagem Automática (ML) e Processamento de Linguagem Natural (NLP). Diversas ferramentas e técnicas têm sido desenvolvidas para otimizar processos, como a geração automática de código, a automação de testes, a refatoração assistida e a manutenção preditiva. A pesquisa adota uma abordagem baseada no Design Science Research (DSR) para analisar o impacto da IA no ciclo de vida do software, desde a conceção até à implementação e manutenção. A revisão da literatura e o estudo do estado da arte permitiram identificar as principais abordagens utilizadas para integrar IA no desenvolvimento de software, bem como as vantagens e desafios dessa integração. Entre os aspetos analisados, destaca-se a utilização de Modelos de Linguagem de Grande Escala (LLMs), como o GitHub Copilot, e frameworks de geração automática de código, que demonstram um impacto significativo na produtividade dos programadores. Além disso, ferramentas baseadas em IA apresentam desafios relacionados à interpretabilidade dos modelos, fiabilidade do código gerado e adaptação das equipas de desenvolvimento. Os resultados obtidos demonstram que a integração entre IA e DS representa uma evolução no setor tecnológico, abrindo novas oportunidades para a automação de tarefas complexas e a melhoria da eficiência dos processos de engenharia de software. No entanto, a investigação também revela a necessidade de desenvolver mecanismos robustos de validação, segurança e mitigação de riscos associados à dependência de modelos preditivos. O trabalho conclui que, embora a IA apresente benefícios substanciais no contexto do desenvolvimento de software, a sua aplicação requer estratégias que garantam a qualidade do software e a adaptabilidade das equipas ao novo paradigma tecnológico.

Palavras-chave: Inteligência Artificial, Desenvolvimento de Software, Aprendizagem Automática, Geração de Código, Modelos de Linguagem de Grande Escala.

Fevereiro de 2025

ABSTRACT

This study investigates the intersection between Software Development (SD) and Artificial Intelligence (AI), exploring the transformations that AI has introduced in traditional software engineering methodologies and practices. With the advancement of Machine Learning (ML) models and Natural Language Processing (NLP) techniques. Various tools and methods have been developed to optimize processes such as automatic code generation, test automation, assisted refactoring, and predictive maintenance. This research adopts a Design Science Research (DSR) approach to analyze the impact of AI across the software lifecycle, from design to implementation and maintenance. The literature review and state-of-the-art analysis identified the primary approaches used to integrate AI into software development, as well as the benefits and challenges associated with this integration. Among the key aspects analyzed, the study highlights the use of Large Language Models (LLMs), such as GitHub Copilot, and AI-powered code generation frameworks, which have shown a significant impact on developer productivity. Furthermore, AI-driven tools present challenges related to model interpretability, reliability of generated code, and adaptation of development teams. The findings demonstrate that AI-SD integration represents a significant evolution in the technology sector, opening new opportunities for automating complex tasks and enhancing software engineering efficiency. However, the study also reveals the need to develop robust validation, security, and risk mitigation mechanisms to address the dependency on predictive models. The research concludes that, while AI offers substantial benefits in software development, its application requires strategies to ensure software quality and adaptability of development teams to the evolving technological landscape.

Keywords: Artificial Intelligence, Software Development, Machine Learning, Code Generation, Large Language Models.

Conteúdo

Índice de figuras

Índice de Tabelas

Lista de Acrónimos

DS - Desenvolvimento de Software

IA - Inteligência Artificial

DSR - Design Science Research

SDLC - Ciclo de Vida do Desenvolvimento de Software

ANI - Artificial Narrow Intelligence

AGI - Artificial General Intelligence

ASI - Artificial Superintelligence

ML - Machine Learning

DL - Deep learning

CNN - Redes Neurais Convolucionais

RNN - Rede Neural Recorrente

NLP - Processamento de Linguagem Natural

GPT - Generative Pre-trained Transformer

BERT - Bidirectional Encoder Representations from Transformers

LLM - Modelos de Linguagem de Grande Escala

HTML - HyperText Markup Language

JS - Javascript

CSS - Cascade Style Sheets

DOM - Document Object Model

API - Application Programming Interface

CDN - Content Delivery Network

NPM - Node Package Manager

SGBD - Sistema de Gestão de Bases de Dados

DDL - Data Definition Language

JWT - JSON Web Token

1. INTRODUÇÃO

A evolução tecnológica tem sido impulsionada por avanços significativos na Inteligência Artificial (IA), envolvendo diversas áreas do conhecimento e, em especial, o Desenvolvimento de Software (DS). Nos últimos anos, a IA tem desempenhado um papel cada vez mais relevante na criação, otimização e manutenção de sistemas computacionais, promovendo uma transformação nos paradigmas tradicionais da engenharia de software. A sua aplicação tem sido amplamente explorada em áreas como visão computacional, reconhecimento de fala, automação de processos e suporte à programação, trazendo novas oportunidades e desafios para programadores e pesquisadores.

Neste contexto, a integração da IA no desenvolvimento de software tem-se tornado um campo promissor, permitindo a automatização de tarefas, a melhoria na depuração de código e a sugestão de soluções mais eficientes para problemas complexos. No entanto, essa convergência também impõe desafios, como a necessidade de adaptação dos profissionais de tecnologia, o desenvolvimento de novas abordagens metodológicas e a superação das limitações relacionadas aos modelos probabilísticos utilizados na IA. Por sua vez, a modelação de sistemas baseados em IA exige não apenas conhecimentos avançados em programação, mas também uma compreensão profunda de teoria da informação, estatística e aprendizagem automática.

Diante deste cenário, esta dissertação investiga como a IA pode potencializar e otimizar o desenvolvimento de software, abordando desde aspetos conceptuais até a implementação prática de uma aplicação assistida por modelos de linguagem avançados. O estudo tem como objetivo não apenas demonstrar a viabilidade da IA como ferramenta auxiliar no desenvolvimento de software, mas também explorar as suas limitações, desafios e impactos no ciclo de vida do DS em sistemas computacionais. Através da construção de um protótipo funcional e da análise dos seus resultados, procura-se comprovar a eficácia da integração entre modelos de IA e práticas modernas de engenharia de software, contribuindo para um entendimento mais amplo sobre o papel da inteligência artificial nesta área.

1.1. CONTEXTUALIZAÇÃO

A IA tem desempenhado um papel crucial na evolução das práticas de DS, principalmente se observarmos os últimos anos, em termos de notícias, novas aplicações, novas plataformas, etc. Por ter apresentado uma progressão ágil e poder ser aplicada de maneira prática e de uso generalizado, a IA tornou-se um novo método

preferencial para desenvolver sistemas de software úteis em várias áreas, como visão computacional, reconhecimento de fala e controlo de robôs. Pode-se notar uma transição que reflete uma mudança significativa no campo da engenharia de software, onde a IA não apenas complementa, mas frequentemente redefine as práticas e processos estabelecidos. A integração da IA no desenvolvimento de software é sem dúvida um mar aberto com diversas oportunidades, mas possui desafios únicos. Por um lado, a IA disponibiliza formas inovadoras de otimizar e aprimorar os sistemas e as suas metodologias. Contudo, introduz uma camada de incerteza que pode afetar desde a recolha/definição de requisitos até à manutenção do produto em si. Esta incerteza, decorrente da modelação probabilística e com uma natureza imprevista dos algoritmos de IA, impõe a necessidade de novas abordagens em teste, depuração e manutenção [1].

Assim como no mundo real, ao utilizarmos IA no DS, devemo-nos preocupar na separação das fases deste processo. Uma vez que, na fase de requisitos, por exemplo, os sistemas baseados em IA requerem uma abordagem mais experimental e orientada a dados, frequentemente necessitando de experiências preliminares. Por sua vez, devemos ainda, considerar a perda de desempenho. Da mesma forma, na construção do software, as práticas de codificação e as ferramentas utilizadas em sistemas de IA diferem significativamente dos sistemas não baseados em IA, com ênfase maior em processamento de dados, análise de recursos e modelação de dados.

Não obstante, esta evolução em direção à integração da IA no DS, também criou ainda, uma necessidade distinta de capacidades e conhecimentos entre os profissionais da área da programação. Tipicamente, programadores informáticos baseados em IA necessitam de ter um conhecimento intensivo em matemática, teoria da informação e estatísticas [1], além de terem de perceber que irão enfrentar desafios únicos em termos de complexidade do trabalho e identificação das tarefas. Essas exigências [1] refletem as diferenças fundamentais nos processos de desenvolvimento e manutenção de software entre sistemas baseados em IA e sistemas tradicionais.

1.2. OBJETIVOS

Inicialmente, pretende-se analisar como a IA tem remodelado as práticas de desenvolvimento de software, destacando a sua capacidade de não apenas complementar, mas também redefinir processos e abordagens estabelecidas. Este estudo visa clarificar as mudanças significativas observadas no campo, principalmente no que diz respeito à agilidade no DS e a aplicação prática generalizada da IA em diversas áreas, como visão computacional, reconhecimento de fala, controlo de robôs e execução de testes de software. Um objetivo específico é explorar os desafios e oportunidades

inerentes a integração da IA no DS, reconhecendo a inovação proporcionada pela otimização de sistemas, mas também enfrentando a camada de incerteza introduzida pela modelação probabilística e pela natureza imprevisível dos algoritmos de IA. Neste sentido, surge a necessidade de novas abordagens em testes, depuração e manutenção é evidenciada, delineando um entendimento mais profundo da complexidade envolvida na aplicação prática da IA no ciclo de vida do software.

Além disso, este estudo tem como uma das metas, analisar e avaliar como a IA influencia cada fase do processo de DS, desde a recolha de requisitos até a construção do software. Uma atenção especial é dada à fase de requisitos, onde sistemas baseados em IA procuram uma abordagem mais experimental e orientada a dados, exigindo experiências preliminares e consideração sobre as perdas de desempenho. No âmbito da prova de conceito da construção do software, serão exploradas as práticas de codificação e as ferramentas utilizadas em sistemas de IA, evidenciando-se as diferenças significativas em relação aos sistemas não baseados em IA, com especial ênfase no processamento de dados, análise de recursos e modelação de dados.

Adicionalmente, o estudo visa compreender as implicações práticas da evolução em direção à integração da IA no DS na formação de capacidades e conhecimentos necessários aos profissionais da área da programação. A necessidade de uma compreensão intensiva de matemática, teoria da informação e estatísticas [1] por parte dos programadores informáticos baseados em IA será analisada, bem como os desafios únicos relacionados com a complexidade do trabalho e a identificação de tarefas.

Em suma, os objetivos deste estudo abrangem uma análise detalhada das transformações surgidas pela IA no DS, abordando desafios, oportunidades, influência em cada fase do processo, assim como as implicações na formação profissional de programadores informáticos.

Por fim, como prova de conceito, foi desenvolvido um software, aproveitando abordagens avançadas de IA para criar um protótipo diferente e totalmente ou parcialmente impulsionada por IA. Serão exploradas as mais recentes tendências e avanços na área de IA, de modo a criar uma solução que não só responda às necessidades específicas, mas que também se destaque pela aplicação inteligente e eficaz de integração com algoritmos avançados de forma a disponibilizar uma experiência diferenciada ao utilizador, tipicamente um programador informático.

1.3. METODOLOGIA DE INVESTIGAÇÃO E PLANO DE TRABALHOS

Para a elaboração desta dissertação, a metodologia seguida foi a Design Science Research (DSR) [2]. A escolha desta abordagem metodológica baseia-se na necessidade de criar soluções inovadoras e práticas para os desafios enfrentados

neste domínio em constante evolução. Também foi escolhida devido à sua natureza orientada para a criação de artefactos, alinhando-se perfeitamente com os objetivos do estudo. Enquanto abordagens tradicionais de investigação podem-se focar na observação e descrição de "fenómenos", o DSR destaca-se pela ênfase na construção de soluções concretas para problemas práticos. Isto alinha-se intrinsecamente com a necessidade de desenvolver e aprimorar técnicas e ferramentas para o eficaz DS baseado em IA.

A relevância do DSR neste contexto reside na sua capacidade de integrar teoria e prática, permitindo a criação de artefactos inovadores que podem ser implementados e avaliados no mundo real. Esta abordagem não só facilita o avanço teórico do campo, mas também fornece contribuições tangíveis que podem ser aplicadas na indústria. No contexto do DS baseado em IA, onde a aplicabilidade prática é essencial, o DSR emerge como uma metodologia com muito valor. Ao seguir os passos delineados pelo DSR desde a identificação do problema até a avaliação do artefacto investigadores e profissionais podem, com este estudo, orientar as suas metodologias de DS para melhorar o seu desempenho profissional. Assim, o ciclo de investigação do DSR torna-se um catalisador para o progresso contínuo no DS com base na IA.

Utilizou-se o DSR com critérios bem definidos na fase de investigação, formulando consultas para identificar artigos focados na junção da IA com o DS. Os artigos selecionados foram submetidos a uma avaliação por parte dos meus orientadores, considerando a sua pertinência, rigor metodológico e contributo para a área, além da sua relevância atual. Foram excluídos trabalhos que não estivessem alinhados com estas exigências, nomeadamente os que não se relacionavam diretamente com a aplicação da IA no DS ou que apresentavam limitações metodológicas. A análise privilegiou estudos que abordassem perspetivas práticas inovadoras, avanços tecnológicos e relevância teórica para o tema em questão.

Na tabela seguinte é possível verificar o plano de atividades cumprido durante a elaboração desta dissertação.

	Meses											
Fases	**11/23**	**12/23**	**01/24**	**06/24**	**07/24**	**08/24**	**08/24**	**11/24**	**12/24**	**12/24**	**01/25**	**02/25**
Elaboração do Estado da Arte.												

Construção, revisão e Entrega da Proposta da Dissertação												
Construção, Revisão e Entrega do Artigo												
Imersão aprofundada nos artigos para os estudos e a construção da Dissertação.												
Concepção e construção da plataforma de simulação que irá suportar o projeto												
Implementação da plataforma de simulação												
Prototipação, simulação e realização de testes para evidências												
Escrita da documentação												
Conclusão e revisão da dissertação												
Entrega da tese												

Tabela 1 - Cronograma

1. **Elaboração do Estado da Arte** - (1 mês): Pretendeu-se fazer uma revisão do que foi e o que está a ser feito a nível mundial, tanto a nível de investigação, como a nível de aplicações e protótipos associados ao contexto de aplicabilidade deste trabalho. Para isso, uma estratégia passou pela recolha de informação de literatura web online em repositórios digitais de artigos científicos;
2. **Construção, Revisão e Entrega da Proposta da Dissertação** – (1 mês e meio): A proposta da dissertação é o documento inicial que apresenta o plano do trabalho de pesquisa. Nela foi incluída a definição do problema, os objetivos, a metodologia, o cronograma e a relevância do estudo. Após a redação, foi essencial submetê-la a revisões críticas para garantir que estivesse clara, objetiva e alinhada com os requisitos do tema do trabalho. A entrega marca o compromisso formal com o desenvolvimento do projeto.

3. **Construção, Revisão e Entrega do Artigo** - (2 meses): O artigo científico corresponde a um resumo ou uma parte específica da pesquisa, destinado

a ser submetido para publicação em revistas ou conferências científicas. Tencionou-se apresentar resultados preliminares ou avanços importantes realizados na área de estudo deste trabalho. A escrita e revisão do artigo exigiu um foco na clareza, precisão e a conformidade com os padrões académicos em termos da sua publicação.

4. **Imersão Aprofundada nos Artigos para os Estudos e a Construção da Dissertação** - (4 meses): Nesta etapa, centrou-se numa análise aprofundada de textos científicos relacionados com o tema, procurando argumentos sólidos e referências para sustentar o objetivo do trabalho. Esta imersão foi fundamental para obter "insights" e enriquecer a base teórica, além de orientar a estruturação e o desenvolvimento dos capítulos da dissertação.
5. **Concepção e Construção da Plataforma de Simulação que irá Suportar o Projeto** - (2 meses): Tendo este trabalho envolvido uma componente prática, ou experimental, esta etapa destinou-se a planear e projetar, como prova de conceito, a plataforma de simulação que serviu como ferramenta para o estudo. Esta fase incluiu a definição das funcionalidades, a escolha de tecnologias e a criação de um protótipo inicial, verificando que a plataforma assegurava os requisitos do projeto.
6. **Implementação da Plataforma de Simulação** - (3 meses): Após a conceção, a plataforma de simulação foi desenvolvida e posta em prática. Esta etapa envolveu programação, configuração de sistemas e a integração de componentes necessários para garantir o funcionamento completo. A implementação foi testada de modo a assegurar que está operacional e alinhada aos objetivos definidos.
7. **Prototipagem, Simulação e Realização de Testes para Evidências** - (3 meses) Com a plataforma implementada, procedeu-se à prototipagem e simulações para obter evidências que sustentem a investigação. Os testes realizados nesta fase foram planeados para validar hipóteses e gerar dados relevantes para análise. Esta etapa foi crucial para obter resultados concretos que foram usados na dissertação.
8. **Escrita da Documentação** - (6 meses). Nesta fase, foi elaborada a documentação detalhada do projeto, incluindo descrição da metodologia, implementação, resultados obtidos e análise dos dados. Tencionou-se com que a documentação fosse clara, precisa e suficientemente detalhada para permitir que outros investigadores possam, por um lado entender o

trabalho desenvolvido e em certa medida estender este trabalho no futuro.

9. **Conclusão e Revisão da Dissertação** - (2 meses): Com a maior parte do trabalho concluída, procedeu-se à compilação e finalização da dissertação, que incluiu, redigir os capítulos, a estrutura, corrigir erros e garantir a coesão entre as partes. Nesta fase, foram também integradas as conclusões baseadas nos resultados obtidos, bem como sugestões para trabalhos futuros.

1.4. ESTRUTURA DO DOCUMENTO

Esta dissertação está organizada em capítulos para melhor desenvolvimento do estudo. O capítulo 1 apresenta uma contextualização de todo o trabalho desenvolvido, sendo apresentado a motivação desta investigação e os objetivos que se pretendem alcançar. Além disso, é apresentada a metodologia de investigação usada no trabalho. No capítulo 2 são apresentados alguns conceitos e definições sobre os principais temas abordados, bem como ferramentas e tecnologias usadas neste projeto ou que possuem semelhança. O capítulo 3 por sua vez, apresenta outros trabalhos já realizados na área abordada, nomeadamente trabalhos relacionados com engenharia de software, algoritmos de Machine Learning, Inteligência Artificial e algumas soluções que já existem. O capítulo 4 especifica os requisitos e arquitetura do projeto final, além de descrever o processo do seu desenvolvimento. O capítulo 5 é composto pela análise de resultados obtidos com o desenvolvimento do projeto e a sua validação. Por fim, no capítulo 6 apresentam-se as conclusões do trabalho realizado e possíveis linhas para trabalho futuro.

2. CONCEITOS, FERRAMENTAS E TECNOLOGIAS

Nesta secção são apresentados alguns conceitos e definições que serão utilizados ao longo do documento e que se consideram fundamentais para contextualização e assim facilitar o entendimento do problema que este trabalho visa resolver. O capítulo se divide em três subseções para que se estabeleça uma base sólida e coesa. A primeira subsecção aborda os conceitos fundamentais relacionados ao desenvolvimento de software. Os conceitos relacionados à IA encontra-se na segunda secção. Para concluir, a terceira secção apresenta ferramentas e tecnologias que irão permitir o desenvolvimento de uma aplicação que integra o uso de IA no desenvolvimento de software, sendo a base para o projeto final proposto neste trabalho.

2.1. CONCEITOS FUNDAMENTAIS DO DESENVOLVIMENTO DE SOFTWARE

O desenvolvimento de software é uma disciplina complexa e multifacetada que abrange um conjunto de metodologias, práticas e processos organizados para a criação de aplicações de software eficientes, robustas e escaláveis. No centro desta disciplina encontra-se o Ciclo de Vida do Desenvolvimento de Software (SDLC), que descreve as várias fases, desde o levantamento de requisitos até à manutenção contínua após a implantação do software. Este ciclo é estruturado por diferentes metodologias, como as tradicionais, incluindo o modelo em cascata, e as modernas, como as metodologias ágeis, que procuram otimizar a colaboração, a flexibilidade e a entrega de valor. Além disso, a qualidade do software, assegurada através de práticas de verificação e validação, desempenha um papel crucial para garantir que o produto final cumpre com as expectativas dos utilizadores e os requisitos. Compreender estes conceitos é essencial para contextualizar o papel da IA no desenvolvimento de software, onde novas ferramentas e técnicas têm o potencial de transformar significativamente cada uma destas fases e práticas.

2.1.1. CICLO DE VIDA DO DESENVOLVIMENTO DE SOFTWARE (SDLC)

Todo o sistema de Software tem um ciclo de vida [1], um período de duração, que inclui a conceção, produção, e outras fases, culminando na fase de manutenção e evolução, a qual dura enquanto for necessário. O Ciclo de Vida do Desenvolvimento de Software, do inglês *Software Development Life Cycle* (SDLC), corresponde a um modelo estruturado que define as etapas e processos envolvidos na criação e manutenção do software.

Tradicionalmente, o SDLC é composto por várias fases sequenciais, em muitos casos resumidas, começando pelo levantamento e análise de requisitos, onde se identificam e documentam as necessidades do cliente e as funcionalidades que o software deve possuir. Em seguida, ocorre a fase de design, onde a arquitetura do sistema e os componentes específicos são definidos com base nos requisitos. A implementação, ou codificação, é a fase em que o design é traduzido para uma linguagem de programação, criando o software funcional. Posteriormente, na fase de teste, o software é submetido a uma série de verificações para assegurar que funciona conforme esperado e está livre de defeitos. Após os testes, segue-se a implementação, onde o software é disponibilizado aos utilizadores finais, e, por fim, a fase de manutenção, que lida com correções de erros, atualizações e melhorias contínuas para garantir que o software permanece útil ao longo do tempo [3]. A quantidade e sequência das diferentes fases depende do modelo de desenvolvimento a ser seguido. No entanto, pelo menos cinco fases estão presentes em qualquer modelo de SDLC [1], conforme a Figura 1 [4].

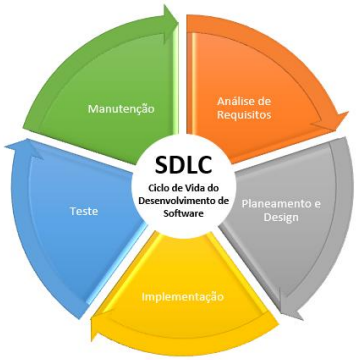

Figura 1- Fases do SDLC e entregas relacionadas

Como existem várias abordagens ao SDLC, cada uma com características distintas que podem ser adaptadas conforme a natureza do projeto, a escolha do modelo de SDLC mais adequado depende de vários fatores, incluindo a complexidade do projeto, a clareza dos requisitos e as necessidades do cliente.

2.1.2. Metodologias e Modelos de Desenvolvimento

As metodologias e modelos de desenvolvimento de software representam diferentes abordagens e práticas usadas para gerir e executar os projetos de software de forma eficiente. Estas metodologias fornecem um conjunto estruturado de processos, ferramentas e técnicas que auxiliam as equipas ao longo do ciclo de vida

do desenvolvimento, o que assegura a entrega de software de qualidade dentro dos prazos e orçamentos estabelecidos.

Ao longo dos anos, surgiram diversos modelos de desenvolvimento, cada um com as suas características e aplicações. Inicialmente, o modelo em cascata dominou o cenário, com as suas fases sequenciais e rígidas e está entre os modelos de desenvolvimento mais tradicionais. O modelo em cascata segue uma abordagem sequencial, tendo definido um processo rígido em que as suas as fases são completadas de forma linear e em etapas sequenciais distintas, conforme a Figura 2. Este modelo, embora simples e fácil de entender, podendo ser inadequado para projetos onde os requisitos não são completamente compreendidos desde o início ou onde há necessidade de mudanças durante o desenvolvimento [3].

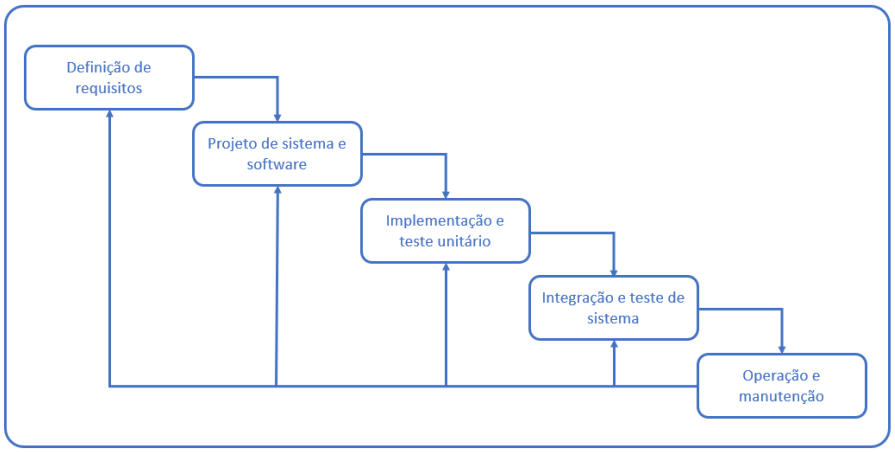

Figura 2 - Modelo Cascata

No entanto, baseado na crescente complexidade dos sistemas e a necessidade de maior flexibilidade impulsionaram a procura por abordagens mais adaptativas e, em resposta às limitações das abordagens tradicionais, surgiram as metodologias ágeis, que promovem a flexibilidade, a colaboração e a entrega contínua de software. Estas metodologias, como Scrum (Ver Figura 3) e Kanban [5], dividem o trabalho em ciclos curtos e iterativos, conhecidos como sprints, permitindo que as equipas se adaptem rapidamente às mudanças nos requisitos e à volatilidade do contexto legal em que se insere o projeto ou das necessidades dos clientes [3].

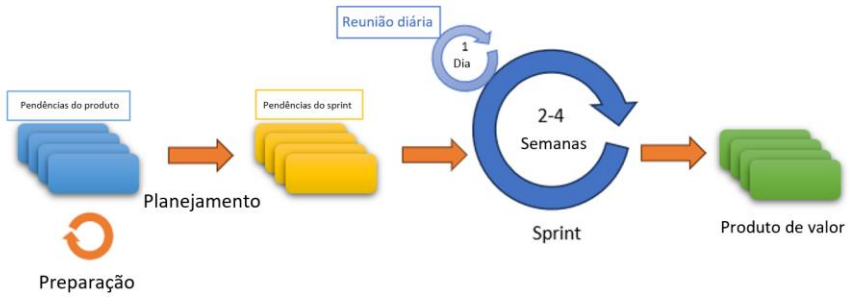

Figura 3 - Modelo Ágil Scrum

As metodologias ágeis privilegiam a comunicação constante entre os membros da equipa e os stakeholders, bem como a entrega frequente de incrementos funcionais do software e a correspondente obtenção de *feedback* do cliente, o que reduz o risco de atrasos e falhas. Esta abordagem tem-se tornado cada vez mais popular em ambientes de desenvolvimento dinâmicos, onde a capacidade de responder rapidamente a mudanças é crucial [6].

As metodologias em cascata e ágeis são as mais conhecidas, todavia, o universo do desenvolvimento de software oferece uma gama muito mais ampla de abordagens. A escolha da metodologia ideal depende de diversos fatores, como o tamanho do projeto, a complexidade do sistema, a experiência da equipa e as necessidades do cliente.

Outros modelos de desenvolvimento e metodologias relevantes:

Metodologia RUP (Rational Unified Process): Uma abordagem iterativa e incremental que privilegia a utilização de casos de uso para modelar os requisitos do sistema. É altamente customizável e pode ser adaptada a diferentes tipos de projetos [3].

Metodologia XP (Extreme Programming): Uma das primeiras metodologias ágeis, focada em práticas como programação em pares, testes unitários e integração contínua [3].

Metodologia Scrum: A mais popular, atualmente, das metodologias ágeis, privilegia a colaboração em equipas auto-organizadas e a entrega incremental de software funcional [5].

Metodologia Kanban: Uma abordagem visual e evolutiva que se concentra no fluxo de trabalho e na eliminação de “desperdícios” [7].

Metodologia Lean: Baseada nos princípios do Lean Manufacturing, procura a otimização do processo de desenvolvimento, eliminando atividades que não agregam ou acrescentam valor [8].

Metodologia Devops: Uma cultura e conjunto de práticas que une o desenvolvimento e operações, com foco na automatização e na entrega contínua [9].

Metodologia em Espiral: Combina elementos de modelos sequenciais e prototipagem, permitindo a gestão de riscos e a adaptação a mudanças [10].

Metodologia Crystal: Uma família de metodologias ágeis que se adaptam às características específicas de cada projeto e equipe [11].

2.1.3. ENGENHARIA DE REQUISITOS

A engenharia de requisitos é uma fase essencial no ciclo de vida do desenvolvimento de software, responsável pela identificação, documentação e gestão das necessidades e expectativas dos *stakeholders* em relação ao sistema a ser desenvolvido. O objetivo principal deste processo é garantir que o software final entregue, satisfaz as necessidades dos utilizadores e, ao mesmo tempo, é tecnicamente viável e é entregue dentro dos limites de custo e tempo.

A engenharia de requisitos envolve diversas atividades, o que inclui a elaboração de requisitos funcionais e não funcionais, a negociação com os *stakeholders*, a validação dos requisitos e a gestão contínua de alterações ao longo do ciclo de desenvolvimento [3].

O processo de engenharia de requisitos pode ser dividido em quatro etapas principais: **levantamento**, **análise**, **especificação** e **validação**. O levantamento de requisitos consiste na recolha de informações dos utilizadores, clientes e outros *stakeholders*, para identificar as suas necessidades. A análise de requisitos envolve a interpretação e organização dessas informações, resolvendo conflitos ou ambiguidades, garantindo que as necessidades sejam claras e compreensíveis. A especificação de requisitos é o processo de formalização dessas necessidades numa documentação clara e detalhada, que será utilizada como referência no design e desenvolvimento do sistema. Finalmente, a validação de requisitos assegura que os requisitos estão completos, consistentes e corretos, e que refletem adequadamente as necessidades dos *stakeholders* [12].

Existem dois tipos principais de requisitos: funcionais e não funcionais. Os requisitos funcionais descrevem as funcionalidades que o sistema deve desempenhar, como as ações que os utilizadores podem executar e o comportamento do sistema

em resposta a entradas específicas. Por outro lado, os requisitos não funcionais referem-se a características do sistema, como desempenho, segurança, fiabilidade e escalabilidade. Estes requisitos influenciam diretamente a arquitetura do sistema e são cruciais para garantir a satisfação dos utilizadores finais. A falha na identificação ou especificação clara de qualquer um destes tipos de requisitos pode resultar em problemas significativos durante o desenvolvimento, afetando a qualidade e o sucesso do sistema [13].

A gestão de requisitos é também um componente vital da engenharia de requisitos, dado que os requisitos podem evoluir ao longo do tempo devido a mudanças nas necessidades dos utilizadores ou no ambiente em que o sistema opera. A gestão de requisitos implica monitorizar, controlar e implementar alterações nos requisitos de uma forma estruturada, garantindo que as mudanças são incorporadas de forma eficiente sem comprometer a integridade do sistema. Assim, a engenharia de requisitos não é um processo isolado, mas sim contínuo, que permeia todas as fases do desenvolvimento de software [14].

2.1.4. ARQUITETURA DE SOFTWARE

A arquitetura de software representa a estrutura fundamental de um sistema de software, delineando os componentes principais, as suas relações e como interagem para alcançar os objetivos do sistema. É como um *blueprint* de um edifício, definindo a organização geral e como as diferentes partes se encaixam [15].

A arquitetura de software serve como uma base sólida para o desenvolvimento, promovendo a compreensão do sistema como um todo e facilitando a tomada de decisões sobre a implementação. Ela influencia diretamente a qualidade do software, a manutenção, a escalabilidade e o desempenho.

Elementos da Arquitetura de Software

Uma arquitetura de software típica inclui os seguintes elementos [16]:

- **Componentes**: Módulos interdependentes que realizam funções específicas dentro do sistema.
- **Conectores**: Mecanismos que permitem a comunicação e a interação entre os componentes.
- **Interfaces**: Pontos de contato entre os componentes, definindo como eles comunicam.
- **Estilo arquitetural**: Um conjunto de padrões e práticas que guiam a organização do sistema, como arquitetura em camadas, arquitetura cliente-servidor, arquitetura orientada a serviços (SOA), entre outros.

Importância da Arquitetura de Software

A arquitetura de software desempenha um papel crucial em diversos aspectos do desenvolvimento [16]

- **Tomada de decisões**: A arquitetura orienta as decisões de design e implementação, garantindo a consistência e a qualidade do software.
- **Comunicação**: A arquitetura serve como um meio de comunicação entre os stakeholders, facilitando a compreensão do sistema.
- **Reutilização**: Uma boa arquitetura permite a reutilização de componentes e soluções em diferentes projetos.
- **Evolução**: A arquitetura deve ser flexível para acomodar mudanças e evoluções futuras do sistema.
- **Qualidade**: A arquitetura influencia diretamente a qualidade do software, impactando atributos como desempenho, segurança, confiabilidade e manutenção.

2.1.5. QUALIDADE DE SOFTWARE

A qualidade de software refere-se ao grau em que um sistema ou componente de software cumpre os seus requisitos especificados, bem como as necessidades e expectativas dos utilizadores. Este conceito abrange diversos aspetos, desde a correção funcional até à usabilidade, desempenho, manutenibilidade e segurança.

O objetivo principal da qualidade de software é garantir que o software desenvolvido seja confiável, eficiente e fácil de manter, proporcionando assim uma melhor experiência para os utilizadores finais. A qualidade de software pode ser medida através de métricas mais ou menos objetivas, permitindo às equipas de desenvolvimento identificar áreas de melhoria e assegurar que o sistema satisfaça tanto os requisitos técnicos como as expectativas do cliente[12].

A qualidade de software pode ser avaliada em dois níveis principais: qualidade interna e qualidade externa. A qualidade interna refere-se a atributos relacionados à arquitetura e ao design do software, como modularidade, coesão e acoplamento, que influenciam diretamente a facilidade de manutenção e extensão do sistema. Já a qualidade externa diz respeito a aspetos visíveis para os utilizadores, como desempenho, fiabilidade, usabilidade e segurança. Estes dois níveis de qualidade estão interligados, uma vez que um design de qualidade interna deficiente pode comprometer a qualidade externa do sistema, dificultando a sua utilização ou causando falhas no seu funcionamento [3].

Para assegurar a qualidade de software, é fundamental implementar processos robustos de Garantia da Qualidade (QA) e controlo de qualidade (QC), como é abordado em [17]. A garantia da qualidade abrange todas as atividades e políticas que visam prevenir erros no processo de desenvolvimento, como o uso de metodologias ágeis, revisões de código e a aplicação de testes contínuos. O controlo de qualidade, por outro lado, foca-se na deteção de defeitos, garantindo que o produto final cumpre os padrões de qualidade estabelecidos. Isto inclui testes de funcionalidade, testes de desempenho, testes de carga e, muitas vezes, a avaliação da experiência do utilizador [17].

Entre os modelos de qualidade mais amplamente aceites, destacam-se o ISO/IEC 25010 [18], que define um modelo para a avaliação da qualidade de software, e o Capability Maturity Model Integration (CMMI), que mede a maturidade dos processos de desenvolvimento. O modelo ISO/IEC 25010, em particular, define oito características de qualidade principais: funcionalidade, fiabilidade, usabilidade, eficiência, manutenibilidade, portabilidade, compatibilidade e segurança. Estes parâmetros fornecem uma estrutura clara para a avaliação e melhoria contínua da qualidade do software [18].

2.1.6. CONCLUSÃO

O desenvolvimento de software é uma disciplina abrangente que reúne metodologias, práticas e processos para a criação de aplicações eficientes e adaptáveis. No centro está o SDLC, uma estrutura que organiza as etapas desde a identificação de requisitos até à manutenção contínua. Este ciclo inclui abordagens tradicionais e metodologias mais modernas e dinâmicas, como as metodologias ágeis. Estas últimas destacam-se pela flexibilidade e colaboração entre equipas, o que promove entregas iterativas e adaptáveis às mudanças nas necessidades dos *stakeholders*.

Outro conceito fundamental é a engenharia de requisitos, que garante que as necessidades dos utilizadores são claramente identificadas, documentadas e geridas ao longo do desenvolvimento. Aliada a esta abordagem está a arquitetura de software, que define a estrutura base do sistema, promovendo decisões de design mais eficazes, melhor comunicação entre os stakeholders e maior capacidade de evolução do software. Por fim, a qualidade de software surge como um elemento crucial, abrangendo aspetos como desempenho, usabilidade, segurança e manutenção.

Estes conceitos fornecem uma base sólida para o trabalho, sendo o uso de metodologias àgeis mais adequado para a entrega do projeto final, com a utilização

de ferramentas e práticas que promovam o desenvolvimento de software alinhado às melhores práticas e padrões de qualidade.

2.2. Conceitos associados à Inteligência Artificial

Nesta secção serão abordados alguns conceitos importantes referente a Inteligência Artificial (IA). Estes conceitos ajudam a entender o que realmente é a IA e as metodologias que estão por detrás deste "conceito". Desta maneira, será possível criar um paralelo com o tema que o trabalho se propõe.

2.2.1. O que é IA?

A IA têm se tornado cada vez mais popular, principalmente por ser abordada em diversos meios de comunicação nos dias atuais. Isto ocorre principalmente devido a presença da IA nas diversas ferramentas de tecnologia, na sua utilização no cinema, rádio, televisão, escrita e até mesmo na arte, isso se deve principalmente por que a IA facilita a execução destas tarefas. Mas o que é a IA? Precisamos entender sua definição e significado.

No Dicionário de Língua Portuguesa, encontramos a seguinte definição: Inteligência Artificial é o "conjunto de todas as faculdades intelectuais (memória, imaginação, juízo, raciocínio, abstração e concepção)" [19]. Na informática, a IA é uma área dentro da ciência da computação que tem como objetivo criar programas informáticos ou sistemas computacionais que possuem a capacidade de simular os seres racionais, em particular a inteligência humana em diversos aspetos, seja no reconhecimento de padrões, na resolução de problemas, na compreensão da linguagem natural ou até mesmo em atividades que se utilizam de criatividade [20]. Russell e Norvig, em [21], definem a IA como a ciência e engenharia de fazer agentes inteligentes, onde um agente é qualquer coisa que possa perceber seu ambiente e tomar ações que maximizem suas chances de sucesso. Essa definição abrangente engloba desde sistemas simples de tomada de decisão até robôs complexos capazes de interagir com o mundo real.

Por outro lado, com o passar do tempo, esta definição tem se aperfeiçoado e sofrido mudanças, principalmente porque a IA tem se tornado uma área que se adapta e é flexível. Segundo Kaplan e Haenlein em [22], podemos definir a IA como "a capacidade de um sistema interpretar corretamente dados externos, aprender com esses dados e utilizar essas aprendizagens para atingir objetivos e tarefas específicas através de uma adaptação flexível".

A IA pode ser dividida em três categorias principais: IA fraca, IA forte e IA superinteligente [22]. A IA fraca Artificial Narrow Intelligence (ANI), refere-se a

sistemas projetados para executar tarefas específicas, como os sistemas focados em recomendação, assistentes virtuais e software de reconhecimento de voz, o que faz esta a forma mais comum de IA atualmente [21]. A IA forte ou IA geral Artificial General Intelligence (AGI) refere-se a sistemas que podem realizar qualquer tarefa cognitiva que um ser humano é capaz, em outras palavras, possuem uma inteligência geral. A AGI seria capaz de resolver problemas de qualquer tipo, independentemente da natureza da tarefa, com habilidades cognitivas comparáveis às de um ser humano. No entanto, a AGI ainda não foi alcançada e permanece um objetivo distante. Por fim, a IA superinteligente *Artificial Superintelligence (*ASI) é um conceito teórico que se refere a uma IA que ultrapassa a inteligência humana em todas as áreas, incluindo criatividade, resolução de problemas e tomada de decisões [22], e isto, por outro lado, levanta questões profundas sobre ética, segurança e o futuro da humanidade, sendo um dos temas mais debatidos entre futuristas e especialistas em IA [23].

A evolução da IA faz-se possível devido ao impulsionamento de avanços em áreas como *Machine Learning* (aprendizagem de máquina), D*eep Learning* (aprendizagem profunda) e as redes neuronais artificiais, que permitiram o desenvolvimento de sistemas capazes de aprender e melhorar o seu desempenho através da utilização de grandes volumes de dados. A aprendizagem supervisionada, não supervisionada e por reforço são algumas das abordagens utilizadas para treinar modelos de IA [21].

2.2.2. APRENDIZAGEM DE MÁQUINA (MACHINE LEARNING)

Machine Learning (ML) é uma subárea da IA que se concentra no desenvolvimento de algoritmos capazes de aprender a partir de dados e melhorar o seu desempenho sem serem explicitamente programados para realizar tarefas específicas. O objetivo da ML é criar modelos computacionais que identifiquem padrões, façam previsões ou tomem decisões com base em dados históricos. Estes modelos são treinados usando grandes volumes de dados, e a qualidade das previsões melhora à medida que o modelo processa mais informação [24].

Este termo não é novo e surgiu em um primeiro momento em 1959, descrevendo como um sistema que permite computadores aprender algumas funções sem que sejam previamente programados para isso. Em outras palavras, a máquina aprende executar algo automaticamente ao ter os seus algoritmos alimentados com dados.

Os autores em [25] descrevem a divisão da ML em quatro principais abordagens de aprendizagem de máquina: aprendizagem supervisionada,

aprendizagem não supervisionada, aprendizagem semi-supervisionada e aprendizagem por reforço (ver Figura 4).

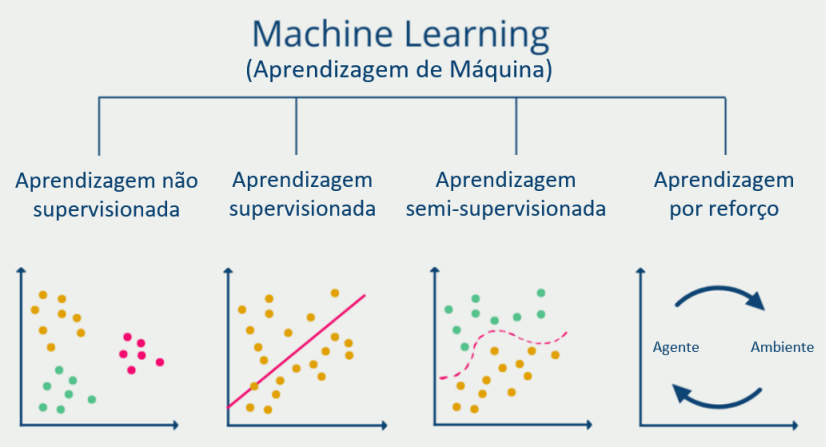

Figura 4- Categorias da ML [26]

Aprendizagem não supervisionada: não utiliza dados rotulados ou etiquetados (dados previamente identificados e categorizados com sua representação), em vez disso, o modelo identifica padrões ocultos ou relações intrínsecas nos dados [25] [27].

Aprendizagem supervisionada: o modelo é treinado com um conjunto de dados rotulados, ou seja, o sistema aprende a associar entradas específicas a saídas conhecidas [27].

Aprendizagem semi-supervisionada: é uma combinação das duas primeiras abordagens: o modelo é treinado com um pequeno conjunto de dados rotulados e um grande volume de dados não rotulados, aproveitando as características de ambos para aumentar a eficiência do processo de treino [27].

Aprendizagem por reforço: o modelo aprende através da interação com um ambiente dinâmico, recebendo recompensas ou penalizações com base nas suas ações, com o objetivo de maximizar os resultados ao longo do tempo [27].

Estas técnicas são amplamente aplicadas em diversas áreas, como reconhecimento de voz, visão computacional, diagnóstico médico e sistemas de recomendação [25].

2.2.3. DEEP LEARNING

Embora a ML seja relevante, as suas abordagens apresentam limitações quando se trata do processamento de dados naturais na sua forma bruta. Por outro lado, o *Deep Learning* (DL) é uma subárea da IA e da ML utiliza as redes neuronais profundas para modelar e resolver estes problemas complexos. Estas redes neuronais

são compostas por várias camadas de neurónios artificiais, que imitam a forma como o cérebro humano processa informações e permitiram a melhora significativa de diversas tecnologias [28][29]. A profundidade das redes permite a extração de características hierárquicas e a modelação de representações complexas dos dados, tornando-as particularmente eficazes em tarefas como reconhecimento de imagem, processamento de linguagem natural e tradução automática [24].

Um dos aspectos fundamentais do *Deep Learning* é o uso de Redes Neurais Convolucionais (CNNs), que são especialmente adequadas para o processamento de dados com uma grade topológica, como imagens [30]. As CNNs aplicam operações de convolução em diferentes partes da imagem, permitindo que a rede aprenda características invariantes à translação, escalonamento e rotação. Esta abordagem revolucionou o campo da visão computacional, levando a avanços significativos em tarefas como reconhecimento facial e segmentação de imagens.

Outra arquitetura relevante é a Rede Neural Recorrente (RNN), que é projetada para lidar com dados sequenciais, como texto ou séries temporais. As RNNs mantêm uma memória dos estados anteriores, permitindo que o modelo considere o contexto ao fazer previsões [31]. No entanto, as RNNs enfrentam desafios como o problema do desvanecimento do gradiente, que afeta a aprendizagem em longas sequências. Para contornar essas limitações, surgiram variantes como as Long *Short-Term Memory* (LSTM) e *Gated Recurrent Units* (GRUs), que melhoram a capacidade da rede em aprender dependências de longo prazo [31].

O treino de modelos de *Deep Learning* é tipicamente intensivo em recursos computacionais e requer grandes quantidades de dados rotulados para obter um desempenho eficaz. A utilização de técnicas de *Transfer Learning*, onde um modelo pré-treinado é ajustado para uma nova tarefa, tem se tornado uma prática comum, permitindo que modelos complexos sejam aplicados a problemas com conjuntos de dados limitados [32]. Além disso, o avanço das unidades de processamento gráfico (GPUs) e dos frameworks de Deep Learning, como *TensorFlow* e *PyTorch*, facilitou o desenvolvimento e a implementação de modelos profundos em diversas aplicações.

2.2.4. Processamento de Linguagem Natural (NLP)

O Processamento de Linguagem Natural (NLP) assim como DL é também uma subárea da IA e procur capacitar os computadores a compreender, interpretar e gerar linguagem humana com significado. O objetivo principal do NLP é permitir que as máquinas interajam com seres humanos utilizando a linguagem natural, envolvendo o processamento de textos e falas em múltiplos níveis, como sintaxe, semântica e pragmática. Segundo os autores em [33], esta tecnologia tem crescido

exponencialmente com o advento de modelos de *Deep Learning*, que permitem uma maior precisão na compreensão e geração de texto, levando à criação de aplicações mais avançadas.

Entre as técnicas fundamentais do NLP, destaca-se a tokenização [34], que envolve a segmentação do texto em unidades menores, como palavras ou frases, para que possam ser processadas pelos algoritmos, conforme a Figura 5.

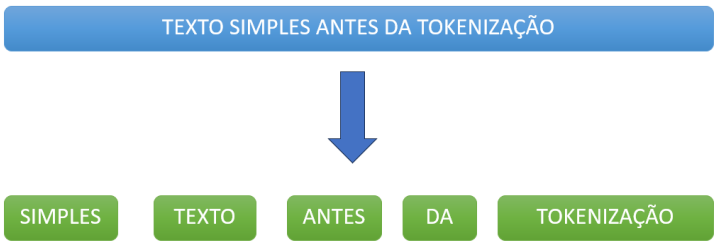

Figura 5 - Segmentação do texto

Após a tokenização, muitas vezes aplica-se o *stemming* ou *lemmatization*, técnicas que reduzem as palavras às suas formas básicas, facilitando o processamento. O *stemming* é um processo de redução de palavras derivadas à sua forma básica ou sua raíz, por outro lado, *lemmatizacion* é um processo que envolve reduzir uma palavra à sua base de dicionário ou seu significado [35], conforme representado na Figura 6.

Palavra	Stemming (Derivação)	Lemmatization (Lematização)
informação	inform	Informação
informativo	inform	Informativo
portáteis	port	portátil
casebre	cas	casa

Figura 6 - Stemming and Lemmatization [35]

Em [34] pode-se perceber que a extração de características é crucial para a modelagem da linguagem e abordagens clássicas, como o *Bag of Words* (BoW) ou o *Term Frequency-Inverse Document Frequency (*TF-IDF), são amplamente utilizadas para representar texto em forma numérica.

Nos últimos anos, a aplicação de modelos de DL ao NLP revolucionou o campo da aplicabilidade da IA. Modelos como o *word2vec* e o *GloVe* trouxeram a ideia de representações distribuídas, ou *word embeddings*, que capturam relações semânticas entre palavras de maneira mais eficaz do que métodos anteriores. Mais

recentemente, modelos baseados em arquiteturas de transformer, como o BERT [36] (*Bidirectional Encoder Representations from Transformers*) e o *Generative Pre-trained Transformer (*GPT), têm permitido um nível sem precedentes de compreensão contextual de texto, liderando avanços significativos em tarefas como tradução automática, sumarização e geração de texto [36].

Uma das maiores aplicações do NLP é na construção de chatbots e assistentes virtuais, como Siri, Alexa e Google Assistant, que utilizam modelos avançados de NLP para interpretar comandos de voz e fornecer respostas relevantes. O autor em [37] aprofunda ainda numa outra aplicação importante, a análise de sentimentos, frequentemente usada em estudos de mercado e redes sociais, onde as opiniões e sentimentos expressos em textos podem ser automaticamente classificados como positivos, negativos ou neutros. Estes avanços mostram que o NLP está a tornar-se cada vez mais central na forma como interagimos com a tecnologia.

No entanto, segundo [38] o NLP ainda enfrenta desafios complexos, especialmente no que diz respeito à ambiguidade linguística e à dificuldade de compreensão de nuances culturais ou contextuais. Além disso, problemas como viés nos dados de treino continuam a ser uma preocupação, uma vez que algoritmos podem inadvertidamente perpetuar preconceitos presentes nos textos originais.

2.2.5. LLMs E GPTs

Os Modelos de Linguagem de Grande Escala (LLMs) [39] são modelos de IA treinados em vastas quantidades de dados textuais para compreender e gerar linguagem natural de maneira sofisticada. Baseados em arquiteturas como o *Transformers*, popularizado pelo GPT [40], esses modelos utilizam grandes volumes de dados para aprender padrões complexos da linguagem humana. Durante o treino, os LLMs ajustam bilhões de parâmetros para prever a próxima palavra numa sequência, o que lhes permite gerar texto coerente, realizar traduções, responder a perguntas e muito mais [39].

Na Figura 7, é possível ver uma representação simplificada do processo de recebimento do conteúdo de diversas fontes de dados, comumente chamado de parâmetros, a passar para o processo de treino de uma LLM. Desta maneira, o modelo "aprende" através da exposição destes parâmetros, analisa os diversos padrões e então envia como resultado informações adaptadas a questão requisitada.

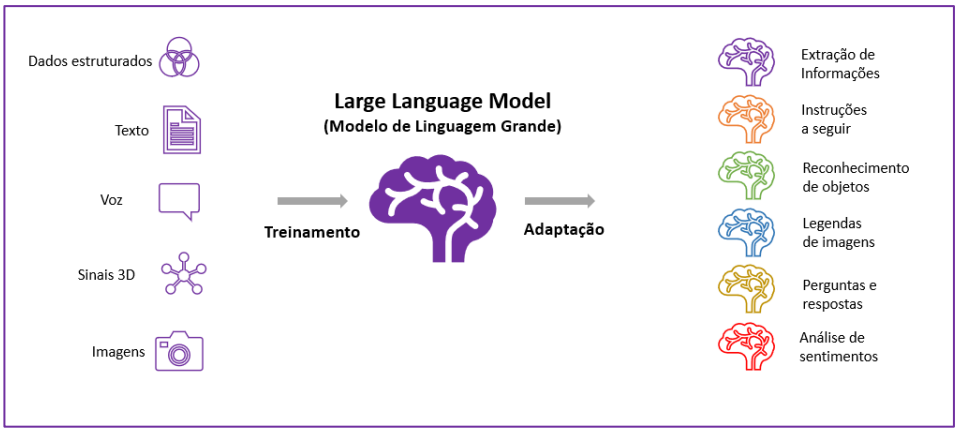

Figura 7 - LLMs

Uma característica fundamental dos LLMs é a sua capacidade de generalização. Graças ao treino numa grande quantidade de conteúdos, que abrangem diversas fontes e domínios, esses modelos conseguem adaptar-se a múltiplas tarefas sem a necessidade de reescrever ou especificar o código. Por exemplo, modelos como o GPT-3 e os seus sucessores (GPT-3.5, GPT-4, etc) podem ser usados para diversas aplicações, como redação de textos, programação de código, atendimento ao cliente e análise de sentimentos, com pouca ou nenhuma adaptação adicional [41]. Esse tipo de generalização tornou os LLMs uma ferramenta valiosa para uma ampla gama de indústrias, desde a saúde até o marketing digital.

Além da sua capacidade de generalização, os LLMs mostram ser sofisticados no que se refere à compreensão de contexto. Conseguem manter e utilizar informações presentes em diálogos com alta durabilidade, o que é essencial para tarefas como chatbots, geração de texto como resposta e interpretação de dados [42]. Entretanto, os LLMs também apresentam desafios, como a tendência de gerar respostas que, embora sejam gramaticalmente corretas, podem conter informações imprecisas ou enganosas. Além disso, o custo computacional para treinar e executar esses modelos é extremamente elevado, exigindo poderosos recursos de hardware, como GPUs e TPUs, além de grandes quantidades de eletricidade, o que levanta preocupações sobre a sustentabilidade e impacto ambiental dessas tecnologias [42].

O desenvolvimento dos LLMs está continuamente a avançar, com investigações em curso para melhorar o controlo, a personalização e a interpretabilidade dos modelos. Inovações recentes incluem Aprendizagem por Reforço com *Feedback* Humano (RLHF) [43], que procura tornar os modelos mais alinhados aos objetivos específicos dos utilizadores e reduzir o viés presente nas saídas geradas pelos LLMs. Embora ainda haja desafios a serem superados, os LLMs representam um marco significativo na evolução da IA, e seu impacto está apenas

começando a ser explorado em áreas como educação, assistência médica e programação.

2.2.6. SISTEMAS DE RECOMENDAÇÃO

Outra subárea da IA e de ML são os Sistemas de Recomendação, que é projetada para sugerir itens relevantes aos utilizadores com base nas suas preferências e histórico de interações. Estes sistemas são amplamente utilizados em plataformas de e-commerce, serviços de *streaming*, redes sociais e muitas outras aplicações, desempenhando um papel fundamental na personalização da experiência do utilizador. Em termos simples, segundo os autores em [44], um sistema de recomendação tenta prever quais itens (produtos, filmes, músicas, etc.) são mais adequados para um determinado utilizador, utilizando diversas técnicas de modelação de dados.

Em [45], o autor cita três abordagens principais utilizadas em sistemas de recomendação: filtragem colaborativa, filtragem baseada em conteúdo e métodos híbridos. A filtragem colaborativa é uma das técnicas mais populares e é baseado na ideia de que os utilizadores com preferências semelhantes terão comportamentos também semelhantes. Portanto, recomenda-se a um utilizador, itens que foram bem avaliados por outros utilizadores com um perfil de preferências similar. Esta abordagem pode ser dividida em duas categorias: colaborativa baseada no utilizador e colaborativa baseada no item. A filtragem colaborativa é eficaz em muitos cenários, mas sofre do problema de "arranque a frio" (*cold start*), quando há pouca ou nenhuma informação disponível sobre um novo utilizador ou item [45].

A filtragem baseada em conteúdo, por outro lado, analisa as características dos itens e compara-as com o histórico de preferências de um utilizador específico. Por exemplo, em uma plataforma de *streaming* de filmes, se um utilizador assistir a muitos filmes de ação, o sistema recomendará outros filmes do mesmo género ou com características semelhantes (atores, diretores, etc.). Esta abordagem é especialmente útil para recomendar itens novos, mas pode sofrer de um fenômeno conhecido como "bolha de filtro", onde os utilizadores são expostos repetidamente a conteúdo semelhante, limitando a diversidade das recomendações [46].

Figura 8 - Tipos de sistemas de recomendação [47]

E por fim, os métodos híbridos combinam elementos das duas abordagens anteriores (Figura 8) para superar as suas limitações individuais. Uma técnica comum é utilizar a filtragem colaborativa para identificar itens relacionados com base nas preferências de outros utilizadores e, em seguida, aplicar filtragem baseada em conteúdo para refinar ainda mais as sugestões apresentadas. Um exemplo prático de sistema híbrido é o utilizado pela Netflix [48], que combina dados comportamentais dos utilizadores com características dos filmes para melhorar a precisão das recomendações.

Com os avanços recentes em *Deep Learning*, surgiram novos modelos para sistemas de recomendação, como as redes neuronais profundas e as autoencoders, que conseguem capturar padrões complexos em grandes volumes de dados. Esses modelos permitem a criação de sistemas mais robustos e escaláveis, oferecendo recomendações personalizadas em tempo real. Além disso, os autores em [49], consideram que o uso de sistemas de recomendação explicáveis (*explainable recommendation systems*) têm ganhado destaque, permitindo que os utilizadores compreendam as razões por trás das sugestões, aumentando a confiança e a transparência do sistema.

2.2.7. CONCLUSÃO

Nesta secção são abordados conceitos fundamentais de IA que servem de base para o trabalho. A IA é definida como a capacidade de sistemas computacionais interpretarem dados externos, aprenderem com eles e utilizarem esse conhecimento para atingir objetivos específicos de forma adaptativa. A evolução da IA tem sido impulsionada por avanços em ML, DL e NLP, permitindo o desenvolvimento de sistemas capazes de compreender e gerar linguagem natural.

Os LLMs, como os modelos GPT, destacam-se pela sua capacidade de processar grandes volumes de dados, identificar padrões complexos e adaptar-se a

múltiplas tarefas sem necessidade de ajustes específicos. Estes modelos, treinados com tecnologias avançadas como *Transformers*, são amplamente utilizados em áreas como redação, programação e atendimento ao cliente. A sua sofisticação permite compreender contextos de forma precisa, tornando-os valiosos para tarefas que exigem interações linguísticas avançadas. No entanto, desafios como custos computacionais elevados e preocupações éticas relacionadas com viés nos dados ainda representam obstáculos a superar.

No âmbito deste trabalho, os conceitos e as aplicações de LLMs e GPTs serão explorados como ferramentas essenciais para a análise e desenvolvimento de soluções baseadas em IA. Estas tecnologias serão utilizadas para compreender padrões linguísticos e criar sistemas adaptativos capazes de responder a contextos específicos, permitindo uma abordagem inovadora à resolução de problemas. Além disso, o projeto irá refletir sobre as implicações práticas e éticas do uso destas ferramentas, contribuindo para a discussão do impacto e das oportunidades oferecidas por estas tecnologias.

2.3. FERRAMENTAS E TECNOLOGIAS

Nesta secção, serão abordadas as principais ferramentas e tecnologias que sustentam o desenvolvimento da prova de conceito apresentada. Cada uma desempenha um papel fundamental na construção, execução e interação com a aplicação desenvolvida.

2.3.1. HTML, CSS E JAVASCRIPT

O HTML, sigla para *HyperText Markup Language* ou Linguagem de Marcação de Hipertexto em português, é a linguagem de marcação fundamental utilizada para criar e estruturar páginas para a internet. Define a estrutura básica de um documento, permitindo a inclusão de textos, imagens, links e outros elementos multimídia [50]. O HTML utiliza uma série de etiquetas ou "*tags*", que são compostos por uma *tag* de abertura e uma de fechamento, para delinear as diferentes partes do conteúdo. Por exemplo, a tag *<p>* é utilizada para criar parágrafos, enquanto *<a>* é utilizada para inserir *hyperlinks*. O HTML é essencial para o desenvolvimento web, pois disponibiliza a base sobre a qual os conteúdos são apresentados, garantindo que sejam acessíveis e legíveis em diferentes dispositivos e navegadores. A evolução do HTML, com versões como HTML5, trouxe novas funcionalidades e elementos que ampliam as capacidades da linguagem, incluindo suporte para áudio, vídeo e interatividade [51].

O JavaScript (JS) é uma linguagem de programação fundamental para o desenvolvimento web, permitindo a criação de interatividade e dinamismo em

páginas. Através do JavaScript, é possível manipular elementos do Document Object Model (DOM) [51], responder a eventos do utilizador e realizar pedidos assíncronos, proporcionando uma experiência mais rica e fluida [52].

Entre as bibliotecas e *frameworks* populares que utilizam JavaScript, destaca-se o React, desenvolvido pelo Facebook. O React permite a criação de interfaces de utilizador de forma eficiente, utilizando um conceito conhecido como "componentes", que facilita a reutilização de código e a gestão do estado da aplicação. A sua abordagem declarativa e baseada em componentes tornou o React uma escolha popular para o desenvolvimento de aplicações web modernas, promovendo a criação de interfaces de utilizador altamente responsivas e interativas [53].

O CSS, *Cascading Style Sheets* ou Folhas de Estilos em Cascata em português, é uma linguagem de estilo utilizada para descrever a apresentação de documentos HTML. Permite separar o conteúdo da apresentação, possibilitando a aplicação de estilos, como cores, fontes, espaçamentos e layout, de forma consistente em várias páginas web. Com o CSS, os programadores podem controlar o design visual de um site, tornando-o mais atrativo e fácil de navegar. Além disso, o CSS é altamente flexível e responsivo, permitindo que os sites se adaptem a diferentes tamanhos de tela e dispositivos, o que é fundamental na era da navegação móvel [54].

2.3.2. NODE JS

O Node.js é uma plataforma de execução para JavaScript construída sobre o motor V8, utilizado pelo navegador Google Chrome [55]. A sua principal característica é permitir que o JavaScript seja executado do lado do servidor, tornando-o uma ferramenta poderosa para o desenvolvimento de aplicações web modernas. Desde a sua introdução em 2009, o Node.js tem sido amplamente adotado devido à sua escalabilidade, desempenho e à sua vasta comunidade de desenvolvimento [56].

Vantagens do Node.js

- **Execução Assíncrona e Não Bloqueante**: O modelo de programação do Node.js é baseado em eventos e assíncrono, permitindo a execução eficiente de operações de entrada/saída sem bloquear a execução de outras tarefas. Esta abordagem melhora significativamente o desempenho, especialmente em sistemas que lidam com um elevado número de pedidos simultâneos [56].
- Ecossistema Amplo: Através do NPM (Node Package Manager), o Node.js oferece acesso a um vasto repositório de bibliotecas e módulos que simplificam e aceleram o desenvolvimento de aplicações. Entre os

pacotes populares estão o Express.js (framework para construção de APIs RESTful) e o Axios (para requisições HTTP) [57].

- Multiplataforma: O Node.js é compatível com os principais sistemas operativos (Windows, macOS e Linux), permitindo que os programadores utilizem a mesma base de código em diferentes ambientes [58].
- Alta Escalabilidade: A sua arquitetura baseada em eventos torna-o ideal para aplicações que requerem alto desempenho, como servidores de API, aplicações em tempo real e sistemas de streaming de dados [59].

Relevância no Contexto do Projecto

No âmbito deste projeto, o Node.js desempenha um papel fundamental na construção da camada backend. A sua integração permite:

- Processar de forma eficiente os pedidos enviados pelo frontend.
- Realizar chamadas à API de LLMs, como o GPT-3 ou GPT-4 [40], para processar as entradas textuais ou visuais dos utilizadores e devolver os resultados gerados.
- Armazenar e gerir o histórico de interações na base de dados, através de integração com SQLite [60].

Além disso, a sua capacidade de lidar com operações assíncronas e não bloqueantes é crucial para garantir um tempo de resposta rápido, respeitando o requisito não funcional de processamento eficiente.

2.3.3. SQLite

O SQLite foi escolhido como o Sistema de Gestão de Bases de Dados (SGBD) para o projeto devido à sua simplicidade, eficiência e adequação a aplicações de pequeno a médio porte. Trata-se de uma base de dados relacional que opera diretamente no sistema de ficheiros, dispensando a necessidade de um servidor de base de dados separado. Esta característica torna o SQLite uma escolha ideal para aplicações onde a simplicidade, portabilidade e configuração mínima são prioridades [60].

Características do SQLite

1. **Armazenamento Local e Autossuficiência:**

O SQLite é um SGBD embutido, o que significa que toda a base de dados é armazenada num único ficheiro no disco. Este ficheiro contém tanto os dados como o esquema, garantindo portabilidade e simplicidade na gestão [61].

2. **Zero Configuração:**

Por não requerer instalação ou configuração de um servidor, o SQLite pode ser integrado rapidamente ao projeto, reduzindo a complexidade do ambiente de desenvolvimento e implementação [60].

3. **Desempenho:**

Apesar de ser leve, o SQLite oferece bom desempenho para operações básicas de leitura e escrita, sendo ideal para o armazenamento de históricos e interações geradas pelo sistema [62].

4. **Compatibilidade:**

O SQLite é compatível com as principais linguagens de programação e bibliotecas, incluindo *Node.js*. No projeto, a integração com o SQLite é realizada através de bibliotecas como better-sqlite3 ou sqlite3, que permitem executar consultas SQL de forma simples e eficiente [59].

5. **Conformidade com o SQL:**

O SQLite suporta um subconjunto significativo do padrão SQL, permitindo o uso de operações como SELECT, INSERT, UPDATE e DELETE, o que facilita a implementação de funcionalidades como o armazenamento de históricos de interações ou configurações [60].

Uso no Projeto

No contexto do projeto, o SQLite será utilizado para:

- **Armazenamento de Histórico de Interações:** Permitir que os utilizadores acedam às suas interações anteriores com o sistema, incluindo os prompts fornecidos e os resultados gerados.
- **Gestão de Dados Temporários:** Possibilitar o armazenamento de informações temporárias relacionadas a sessões de utilizadores [62].
- **Portabilidade:** Disponibilizar uma solução que possa ser facilmente transferida entre diferentes ambientes (desenvolvimento, teste e produção) devido ao seu formato de ficheiro único [63].

Vantagens da Escolha

- **Portabilidade**: O ficheiro da base de dados pode ser facilmente transferido entre diferentes máquinas e ambientes [63].
- **Simplicidade**: A ausência de requisitos complexos de configuração permite uma integração rápida [60].
- **Desempenho para Aplicações Simples**: Para o tipo de aplicação desenvolvida, onde os requisitos de base de dados não são intensivos, o SQLite oferece um desempenho adequado [61].

Limitações

Embora o SQLite seja ideal para este projeto, é importante considerar as suas limitações:

- Não é recomendado para aplicações com alto volume de dados ou com múltiplos acessos simultâneos intensivos [59].
- Carece de funcionalidades avançadas presentes em sistemas como PostgreSQL ou MySQL [59].

2.3.4. API

Uma API (*Application Programming Interface*) é um conjunto de definições e protocolos que permite a comunicação entre diferentes sistemas de software. As APIs facilitam a integração de serviços e aplicações, permitindo que elas compartilhem dados e funcionalidades de maneira padronizada. Por exemplo, uma API pode permitir que uma aplicação aceda aos dados de uma base de dados remota ou se conecte a um serviço externo, como por exemplo, a um sistema de pagamento ou uma plataforma de redes sociais [64].

As APIs podem ser classificadas em diferentes tipos, como APIs REST, que utilizam o protocolo HTTP e seguem os princípios da arquitetura REST, e APIs SOAP, que utilizam mensagens XML para a comunicação. A utilização de APIs tem se tornado cada vez mais comum, pois promove a modularidade e a reutilização de serviços, tornando o desenvolvimento de software mais ágil e eficiente [65].

2.3.5. LLMs API

Os LLMs são ferramentas avançadas de IA conforme abordado no capítulo anterior. Estes modelos baseiam-se na arquitetura *Transformer*, introduzida em [66] e revolucionou o NLP ao melhorar significativamente a capacidade dos algoritmos de capturar contextos complexos em texto. A aplicação proposta utilizará uma LLM acessada via API para interpretar os dados fornecidos pelos utilizadores, seja em

formato textual (prompts) ou visual (imagens), e gerar o resultado final esperado. A escolha da API será feita com base em testes comparativos que consideram a adequação ao projeto, precisão nos resultados e custo-benefício.

Entre as opções de LLMs disponíveis, destacam-se três principais soluções: ***ChatGPT*** [67], da OpenAI; ***Claude*** *[68]*, da Anthropic; e ***Gemini*** [69], do Google. O ChatGPT [67], amplamente utilizado, destaca-se pela sua robustez e versatilidade, sendo capaz de lidar com uma ampla gama de tarefas, como geração de texto e codificação [36]. O Claude, por outro lado, é uma solução que se foca em segurança e alinhamento ético, priorizando respostas que minimizem riscos de erro ou viés. Já o *Gemini*, desenvolvido pelo Google, oferece capacidades multimodais, permitindo a combinação de texto e imagens no processamento e análise. Estas características tornam os três modelos candidatos viáveis para atender às necessidades do projeto.

2.3.6. OUTRAS TECNOLOGIAS COMPLEMENTARES

Para suportar a execução e gestão do projeto, outras ferramentas e tecnologias complementares foram utilizadas, a fim de garantir eficiência e qualidade do desenvolvimento:

- **Tailwind CSS e Shadcn/UI:** Frameworks como o Tailwind CSS são utilizados para estilização rápida e eficiente de interfaces através de classes utilitárias, permitindo o desenvolvimento de designs responsivos e consistentes [70]. Já o Shadcn/UI [71] fornece componentes pré-estilizados baseados em Tailwind CSS, o que agiliza ainda mais a criação de interfaces de utilizador. Essas ferramentas minimizam o tempo de desenvolvimento, promovendo um design flexível e escalável [71].
- **Express.js:** O Express.js [72] é um framework minimalista para Node.js, que facilita a construção de servidores web e APIs RESTful [73]. Ele fornece uma camada robusta de funcionalidades, como middleware, gestão de rotas e manipulação de requisições HTTP, tornando o desenvolvimento back-end mais estruturado e eficiente [72].
- **JWT (JSON Web Token):** O JWT é uma tecnologia amplamente utilizada para autenticação e autorização em aplicações web [74]. Baseia-se na troca de tokens JSON codificados, contendo informações como identificação de utilizadores e permissões, que podem ser verificadas e validadas de forma segura. A utilização de JWT elimina a necessidade de armazenar sessões no servidor, promovendo escalabilidade e segurança.
- **TypeScript:** O TypeScript é uma extensão do JavaScript que adiciona tipagem estática e outras funcionalidades avançadas, como interfaces e

suporte para desenvolvimento orientado a objetos. Por sua vez, ajuda a prevenir erros comuns durante o desenvolvimento, o que deixa o código mais robusto e fácil de manter. Além disso, o *TypeScript* compila-se em JavaScript puro para garantir a compatibilidade com todos os navegadores modernos [75].

- **Next.js** é um framework React de código aberto que permite a criação de aplicações web rápidas e escaláveis. É amplamente utilizado por sua flexibilidade e integração com bibliotecas modernas, permitindo o desenvolvimento eficiente de aplicações web com foco em SEO e desempenho, além de possuir o suporte nativo para TypeScript [76].

Estas tecnologias complementares oferecem soluções modernas que contribuem para a eficiência e segurança do projecto e visa refletir as melhores práticas no desenvolvimento de software.

2.3.7. CONCLUSÃO

De forma prática, a aplicação proposta consiste numa plataforma web com uma interface simples e acessível. O utilizador terá à disposição um campo de texto para descrever as suas necessidades ou ideias, assim como a opção de carregar uma imagem representando o protótipo visual da solução desejada. Estes elementos serão processados em conjunto, permitindo a submissão tanto de texto quanto de imagens como entrada para a aplicação.

Após a submissão, a aplicação invocará uma API baseada em LLMs, como o *ChatGPT* [67], *Gemini* [69] ou *Claude* [68], para interpretar as informações fornecidas e gerar o resultado final. O sistema retornará o código-fonte necessário para a implementação do protótipo, juntamente com um relatório explicativo, caso seja possível, um frame com a aplicação funcional. A integração com APIs LLM garante alta precisão na interpretação das necessidades do utilizador.

A abordagem discutida procura simplificar o processo de prototipagem e desenvolvimento, permitindo que utilizadores sem conhecimentos técnicos avancem no design e implementação das suas ideias. Este sistema representa uma solução inovadora e acessível, alinhada com os avanços em IA e DS.

3. ESTUDO DO ESTADO DA ARTE

Na secção 2, foram explorados diversos conceitos referentes ao Desenvolvimento de Software (DS) e de Inteligência Artificial (IA). Estes conceitos são necessários para a construção de uma base necessária para entender de forma mais ampla como ambos podem ser interligados. Nesta secção serão apresentadas diversas abordagens de investigação relacionadas com o tema deste trabalho, que realiza uma interface entre DS e IA e onde será possível realizar uma análise de temas propostos e as suas conclusões, sendo assim possível ter uma visão geral do estado da arte.

3.1. DESENVOLVIMENTO DE SOFTWARE E INTELIGÊNCIA ARTIFICIAL

Ao olharmos em direção ao tema proposto, deparamo-nos com diversas interseções e complementaridade entre IA e DS. Isto deve-se ao facto de que a IA tem alterado significativamente diversos aspetos da vida cotidiana, a influenciar produtos, serviços e processos empresariais. Podemos citar integração em áreas como sistemas de recomendação, automação de tarefas e análises avançadas, o que altera a maneira como interagimos com a tecnologia [77]. Os autores em [77] nos fazem a pergunta "Estamos prontos?" logo no final do título do artigo e, demonstram a importância de se debruçar e entender como a IA pode ser útil e os cuidados devemos ter ao prosseguir. Além da preocupação relacionada aos cuidados, é evidenciado a complementação entre DS e IA de diversas maneiras, como otimização na operação do serviço, realização de testes, validação de código, reusabilidade, entre outros.

Ao pensarmos na colaboração entre IA e o desenvolvimento de software, devemos entender como emerge o uso de ML. Em [78], os autores evidenciam como ML ganhou espaço como método para desenvolvimento de software voltados para visão computacional, reconhecimento de imagens e fala, processamento de linguagem natural, controle de robôs e outras aplicações. Os autores enfatizam como a aprendizagem de máquina transforma as práticas de desenvolvimento de software, exigindo novas capacidades, abordagens e ferramentas. Pode-se sublinhar a importância de adaptar as metodologias de desenvolvimento de software tradicionais para acomodar as especificidades dos sistemas baseados em ML.

Em [79], os autores exploram a automação do processo de geração de código HTML a partir de esboços desenhados "à mão". O ciclo de design de um site geralmente começa com a criação de modelos, também chamados de *mock-ups*, para páginas web, e em seguida são convertidos em HTML estruturado ou código de marcação similar por engenheiros de software. Este processo é repetido várias vezes até que o modelo desejado seja criado. O artigo tem como objetivo automatizar a

geração de código a partir de *mock-ups* desenhados à mão, utilizando técnicas de visão computacional e métodos de DL.

Os sites são cruciais em vários campos, desde o social ao comercial. O desenvolvimento de páginas web que atendam às necessidades de usabilidade e funcionalidade é um processo trabalhoso. Diversas vezes, envolve designers gráficos, especialistas em software e se baseia em *feedback* dos utilizadores finais, sendo um processo repetitivo e demorado. A ideia de gerar automaticamente o código de uma página web é cada vez mais importante, reduzindo o tempo de programação, custos de processo e consumo de recursos.

Os autores em [80] exploram os aspetos envolvidos no uso de ferramentas de IA para a geração automatizada de código HTML. Discutem a integração de técnicas de IA em ferramentas para a geração automatizada de código HTML. Essa integração tem como objetivo melhorar a eficiência e eficácia no desenvolvimento web, especialmente em tarefas repetitivas ou padrão assim como em [79]. Os autores discutem as implicações práticas do uso de IA na geração de código, enfatizando como isso pode transformar as práticas de desenvolvimento web.

Este processo de automação nem sempre é simples, em [81] é exposto o *CodeXGLUE* pelos autores e que consiste em um conjunto de dados para *benchmark* e é projetado para impulsionar a pesquisa no campo de ML focada na compreensão e geração de programas. Este repositório é uma resposta à crescente procura por inteligência de código, visando aumentar a produtividade dos programadores de software na era da IA.

O *CodeXGLUE* [81] representa um passo significativo na pesquisa de engenharia de software apoiada por IA, disponibilizando um conjunto de dados abrangente e ferramentas para abordar uma variedade de tarefas complexas. Ao disponibilizar um ambiente padronizado para avaliação e comparação de modelos, os autores em [62] demonstram que ele facilita o desenvolvimento e a validação de novos métodos em compreensão e geração de programas, estabelecendo-se como um recurso valioso para pesquisadores e programadores no campo da engenharia de software.

Em [82], é realizada uma investigação sobre a usabilidade de ferramentas de geração de código que são alimentadas por Modelos de Linguagem de Grande Escala (LLMs), sendo focado principalmente no *Github Copilot [83]*, ferramenta baseada em LLM. Os autores debruçam em como os avanços recentes em LLMs tornaram possível a geração automática de código para tarefas reais de programação em linguagens de programação de propósito geral, como *Python*. No entanto, poucos estudos se

concentraram na usabilidade dessas ferramentas e em como elas se encaixam no fluxo de trabalho de programação. Em LLMs existem duas abordagens principais para a geração de código: algoritmos de síntese de programas e modelos de aprendizado profundo. Ambos têm limitações, como a dificuldade em escalar para linguagens de programação de propósito geral e a tendência de gerar código com erros sintáticos ou semânticos. LLMs como o GPT-3 e GPT-4 [40] oferecem novas oportunidades para superar essas limitações.

Ainda em [82], é feito um estudo que se concentra em como o uso do *Github Copilot* afeta a experiência de programação, como os utilizadores reconhecem erros no código gerado pelo Copilot, quais estratégias de enfrentamento são empregadas quando encontram erros, e quais obstáculos e limitações podem impedir a adoção do *Copilot*.

A automatização proporcionada pela IA pode levar a uma mudança significativa na forma como o código é gerado e mantido. Por fim, o documento oferece uma visão sobre o futuro da geração de código baseada em IA, sugerindo que à medida que as técnicas de IA se tornam mais sofisticadas, a automação no desenvolvimento web se tornará ainda mais eficiente e eficaz.

Os autores em [84] exploram a interseção entre IA e DS, com foco no uso do ChatGPT para colaboração entre humanos e bots no processo de arquitetura de software. A arquitetura de sistemas de software intensivos envolve desafios complexos, como unificar perspetivas de *stakeholders* e racionalizar decisões de design. A pesquisa em IA para DS foca em desenvolver sistemas de suporte à decisão e bots de desenvolvimento para auxiliar arquitetos com recomendações sobre decisões de design e prever falhas arquitetónicas.

O estudo adota uma abordagem centrada em processos e baseada em cenários para a análise, síntese e avaliação arquitetónica habilitada pelo ChatGPT de um software baseado em microserviços. Os resultados preliminares demonstram as capacidades do ChatGPT em processar histórias de arquitetura, articular requisitos, especificar modelos e recomendar táticas e padrões arquitetónicos.

O estudo [85] aborda a aplicação de ferramentas de IA, especificamente o *ChatGPT*, no DS. O estudo investiga como o *ChatGPT* contribui para os processos de desenvolvimento de software, explorando seu papel em várias fases, desde a análise de requisitos até à manutenção. A pesquisa revela que o *ChatGPT* pode ser eficaz em otimizar processos, sugerir melhorias e auxiliar na geração de códigos, embora com algumas limitações. O estudo ressalta o potencial da IA para revolucionar o desenvolvimento de software, mas também aponta a necessidade de mais pesquisas

para superar desafios e integrar plenamente essas ferramentas no ciclo de vida do desenvolvimento de software.

Em [86], os autores fornecem uma visão abrangente e sistemática sobre a IA, abordando seus avanços, técnicas centrais, cenários aplicáveis e desafios futuros. A IA tem aplicações significativas em setores como automotivo (condução autónoma), médico (assistência médica e desenvolvimento de medicamentos), retalho (comércio inteligente) e mídia (geração automática de conteúdo). O estudo enfatiza ainda que, com o desenvolvimento de condições técnicas como dados, algoritmos e capacidades computacionais, a IA está começando a resolver problemas efetivamente e a criar benefícios económicos. Há uma expectativa de que a inteligência das máquinas possa em breve ultrapassar a humana, não apenas em capacidades lógicas, mas também emocionais [86].

O [87] por sua vez, apresenta uma abordagem inovadora para a geração automática de código usando uma arquitetura baseada em árvore. Esta metodologia busca melhorar a precisão e a relevância do código gerado por máquinas, enfrentando desafios comuns em modelos anteriores. A pesquisa introduz o conceito de *TreeGen*, um modelo que combina a estrutura de dados em forma de árvore com a arquitetura do *Transformer* para gerar código de forma mais eficiente e precisa. A abordagem promete avanços significativos na geração de código, com implicações práticas para o desenvolvimento de software e a aplicação de inteligência artificial nesse campo.

Em [88] aborda-se a crescente importância do software baseado em IA e as necessidades de métodos e ferramentas que apoiem o seu desenvolvimento. O estudo concentra-se em ferramentas e frameworks que automatizam o processo de DS usando IA, analisando estudos científicos validados empiricamente e que se utilizam frequentemente de softwares baseados em *Machine Learning* ou *Deep Learning*. O artigo categoriza e avalia essas ferramentas ou frameworks em termos de características fundamentais, incluindo tema, métodos de pesquisa, tipos de domínio e número de casos em validações. Os autores [88] ainda destacam os desafios que diversas empresas têm reportado ao largo de diversas pesquisas, o que inclui o alto custo em manutenção [89]. No final, o estudo ilustra o estado atual do desenvolvimento de software IA, destacando a diversidade nos domínios de validação e a limitação no número de casos aplicados para validação. Os autores destacam desafios em soluções desta abordagem

Discute-se em [90], a utilização da IA para automatizar e melhorar o desenvolvimento e a manutenção de software. Aborda como as técnicas de IA, incluindo algoritmos de aprendizagem máquina como *Support Vector Machine* e

Random Forest, podem ser aplicadas para prever mudanças no código-fonte. O artigo ressalta a importância da previsão de mudanças para alocar recursos de forma eficiente e reduzir custos. Além disso, enfatiza o papel crescente da IA em várias fases do desenvolvimento de software, desde feedback aos programadores até a automação completa de tarefas, o que têm se tornado cada vez mais uma realidade.

A autora em [91] analisa o papel da IA no desenvolvimento de software, destacando os desafios e oportunidades que emergem com a sua integração. Aborda como tecnologias de IA, como ML e NLP e levanta questões sobre implicações éticas e riscos potenciais associados à IA no desenvolvimento de software, fornecendo insights sobre o futuro deste campo na era da IA.

Em [92], os autores investigam o impacto da utilização de modelos de linguagem de grande escala (LLMs), como o *ChatGPT*, no processo de desenvolvimento de software. A pesquisa destaca como essas ferramentas têm sido integradas em diversas fases do ciclo de desenvolvimento, desde a escrita e correção de código até a documentação e testes. Os autores propõem uma taxonomia que categoriza o uso dos LLMs em tarefas de engenharia de software com base em sua complexidade, grau de automação e dependência do contexto. Um dos principais achados do estudo é que, embora os LLMs ofereçam ganhos expressivos em produtividade e acessibilidade, sua adoção levanta preocupações relacionadas à verificação da qualidade do código gerado, à rastreabilidade das decisões tomadas e à confiança nas respostas produzidas pelos modelos.

Além disso, os autores sugerem que a presença dos LLMs está a reformular práticas tradicionais de desenvolvimento, incentivando novos fluxos de trabalho e metodologias mais colaborativas entre humanos e máquinas [92]. Os autores concluem que, para uma adoção eficaz e segura dessas ferramentas, é essencial o desenvolvimento de práticas robustas de validação, integração contínua e formação especializada dos programadores. Assim, a pesquisa reforça o papel central dos LLMs na transformação da engenharia de software, ao mesmo tempo que reconhece a necessidade de abordar riscos técnicos, organizacionais e éticos emergentes.

No contexto do desenvolvimento de software, o Processamento de Linguagem Natural (NLP) desempenha um papel essencial em várias áreas, com destaque para a criação de assistentes de codificação, como o *GitHub Copilot* e o *Tabnine*, que utilizam modelos de NLP treinados em grandes volumes de código-fonte. Estes assistentes são capazes de sugerir trechos de código, completar funções e até mesmo gerar blocos inteiros de código com base em comentários ou descrições em linguagem natural fornecidas pelos programadores [93]. Funcionam como sistemas de autocompletar

avançados, o que aumenta a produtividade dos programadores e reduz a probabilidade de erros comuns.

Além disso, ferramentas de análise de código automatizada que utilizam NLP são cada vez mais comuns para a revisão de código, deteção de vulnerabilidades e sugestões de melhoria. O NLP permite que essas ferramentas analisem descrições em linguagem natural, como comentários e documentação presentes no código, para entender melhor o contexto e sugerir melhorias em conformidade com as melhores práticas de desenvolvimento. Em [94] é demonstrado o uso de ferramentas como o *DeepCode* e o *CodeBERT* e como utilizam essas técnicas para analisar tanto o código quanto a documentação associada, a identificar inconsistências e otimizar o processo de desenvolvimento.

Outra aplicação importante do uso do NLP no desenvolvimento de software é a geração automática de documentação. Através da análise de código-fonte e comentários presentes, ferramentas baseadas em NLP podem gerar documentação em tempo real, cria descrições de funções, classes e APIs de forma automática. O que economiza tempo para os programadores e garante que a documentação esteja sempre alinhada com o código. Além disso, o NLP está a ser utilizado para construir sistemas de perguntas e respostas que podem ser integrados às bases de conhecimento de uma equipe de desenvolvimento, permitindo que os programadores consultem a documentação e encontrem respostas sobre funções e comportamentos de maneira mais eficiente [95].

O uso de NLP no desenvolvimento de software também permite avanços em tradução de linguagens de programação, onde descrições em linguagem natural podem ser convertidas para código-fonte de forma automatizada, facilitando a criação de programas por programadores que podem não estar totalmente familiarizados com uma linguagem utilizada. Estes avanços transformam a forma como os programadores ou engenheiros de software interagem com as ferramentas de desenvolvimento e colaboram em grandes projetos [96].

3.2. ANÁLISE

Os estudos analisados reforçam a profunda interseção e complementaridade entre a IA e o DS. A IA tem impactado significativamente práticas tradicionais no DS, introduzindo tecnologias como ML e automação avançada, que transformam o modo como softwares são projetados, desenvolvidos e mantidos. Por exemplo, a utilização de ML em áreas como visão computacional, NLP e geração automatizada de código, como HTML ou qualquer outra linguagem e representa uma mudança pragmática que exige adaptações metodológicas e novas competências dos profissionais de

desenvolvimento. Estas tecnologias ampliam as possibilidades, permitindo que tarefas repetitivas e demoradas sejam automatizadas, o que reduz custos e aumenta a eficiência.

Outro aspeto relevante abordado nos estudos é o uso de ferramentas baseadas em IA, como o *GitHub Copilot* [93], que utilizam LLMs. Estas ferramentas têm mostrado grande potencial para melhorar a produtividade dos programadores, seja através de sugestões de código, geração automática de trechos ou tradução de descrições em linguagem natural para código funcional. No entanto, os estudos apontam que, embora promissoras, estas ferramentas enfrentam desafios relacionados à usabilidade, integração no fluxo de trabalho e qualidade do código gerado, sendo necessária uma validação cuidadosa e intervenções humanas para corrigir erros ou evitar vieses. Adicionalmente, estudos como [82] discutem como estas tecnologias podem alterar o processo de design arquitetural, destacando o papel do *ChatGPT* [67] na colaboração entre humanos e *bots* no desenvolvimento de sistemas complexos.

A tabela a seguir apresenta comparações entre os principais artigos estudados neste estado da arte, resume de forma sucinta a abordagem e as ferramentas utilizadas, além de demonstrar os pontos fortes e pontos fracos apresentados:

Referência	**Abordagem e Ferramentas Utilizadas**	**Pontos Fortes Apresentados**	**Pontos Fracos Apresentados**
[77] Artificial Intelligence and Software Engineering: Are We Ready? (2022)	Uso de IA para desenvolvimento de software	Explora a integração de IA e engenharia de software	Falta de validação prática ampla
[78] How does Machine Learning Change Software Development Practices? (2021)	Aprendizado de máquina no desenvolvimento de software	Evidencia novas práticas e mudanças na área	Existe a necessidade de adaptação a novos paradigmas
[79] Automatic HTML Code Generation from Mock-Up Images Using Machine Learning Techniques (2019)	Geração automática de código HTML	Redução do tempo de desenvolvimento e erros humanos	Limitações na conversão exata de esboços utilizados
[80] Factors Involved in Artificial Intelligence-based Automated HTML Code Generation Tool (2020)	Automação no desenvolvimento web	Eficiência na geração de código	Dependência de modelos treinados
[81] *Codexglue*: A machine learning benchmark dataset for code understanding and generation (2021)	Criação de benchmarks para ML em programação	Facilita a avaliação de modelos de IA para código	Conjunto de dados limitado a determinadas linguagens

[82] Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models (2022)	Usabilidade de ferramentas de geração de código	Análise detalhada de impactos no fluxo de trabalho	Geração de código com erros sintáticos
[84]Towards Human-Bot Collaborative Software Architecting with ChatGPT (2023)	Colaboração humano Vs IA na arquitetura de software	Melhora na organização e definição arquitetônica	Dificuldades na adaptação a diferentes projectos
[85] Artificial Intelligence-Based Tools in Software Development Processes: Application of ChatGPT (2023)	Uso do ChatGPT no desenvolvimento de software	Exploração do potencial da IA na geração de código	Limitações nas respostas geradas pelo modelo no momento do estudo
[86] Study on artificial intelligence: The state of the art and future prospects (2021)	Estado da arte da IA e futuro de sua aplicabilidade	Abrangência em aplicações de IA em diferentes setores	Questões éticas e regulatórias ainda abertas
[87] TreeGen: A Tree-Based Transformer Architecture for Code Generation (2020)	Transformers na geração de código	Modelo baseado em árvores para melhorar a geração de código	Alta complexidade computacional para treinar
[88] Tools/frameworks that support development process of AI-based software: validations in white literature (2022)	Ferramentas para IA no desenvolvimento de software	Estudo comparativo entre diferentes abordagens	Falta de padronização entre as diversas ferramentas
[90] Automating Software Development using Artificial Intelligence (2020)	Automação no desenvolvimento de software	Sugestões de implementação prática de IA	Desafios na adoção em larga escala
[91] Impact of Artificial Intelligence on Software Development: Challenges and Opportunities (2023)	Impacto da IA no desenvolvimento de software	Exploração dos benefícios e desafios	Falta de estudos práticos aprofundados
[92] SOEN-101: Code Generation by Emulating Software Process Models Using Large Language Model Agents (2024)	Uso de LLMs como o ChatGPT no ciclo de vida do software	Categorização das tarefas com LLMs, ganhos de produtividade e reformulação de fluxos de trabalho	Dúvidas quanto à qualidade do código, rastreabilidade e confiabilidade das respostas geradas

Tabela 2 - Comparação dos artigos estudados

De uma perspetiva mais ampla, os artigos analisados sugerem um futuro onde a IA desempenhará um papel cada vez mais central no DS, desde a geração automática de código até a previsão de mudanças no ciclo de vida do desenvolvimento. Ferramentas como o *CodeXGLUE* [81] e *TreeGen* [87] representam avanços

significativos na capacidade de compreender e gerar código com maior precisão, fornecendo um ambiente padronizado para testes e validações.

Estes recursos não só otimizam o trabalho dos programadores como também possibilitam soluções mais rápidas e precisas para problemas complexos. Contudo, os estudos também alertam para desafios éticos e técnicos, como o impacto de erros gerados por IA e a dependência crescente destas ferramentas, enfatizando a necessidade de pesquisa contínua e integração cuidadosa no DS. Assim, conclui-se que a IA, embora ainda em evolução, está a redefinir a forma como o software é desenvolvido, de maneira a disponibilizar oportunidades inovadoras e desafios importantes para o futuro da área.

4. ESPECIFICAÇÃO E DESENVOLVIMENTO DO CASO DE ESTUDO

Esta secção descreve o cenário de aplicabilidade do trabalho desenvolvido, os objetivos específicos a serem alcançados, os requisitos levantados e o processo de especificação e desenvolvimento da solução proposta. O caso de estudo apresentado visa explorar a integração entre IA e DS, com destaque aos benefícios práticos e os desafios enfrentados no processo, além de fornecer uma visão geral das principais decisões técnicas e funcionalidades implementadas.

4.1. CONTEXTUALIZAÇÃO E OBJETIVOS

O caso de estudo centra-se na utilização de LLM para automatizar tarefas no desenvolvimento de software, com destaque para auxiliar no desenvolvimento de interfaces visuais baseadas em protótipos e/ou instruções textuais, também conhecidos por *prompt*. Este contexto foi seleccionado devido à sua relevância atual e ao potencial impacto no aumento da eficiência e redução de custos no desenvolvimento de software.

O principal objetivo é demonstrar como a solução pode otimizar processos e melhorar a experiência dos utilizadores num contexto real. Os objetivos específicos incluem:

- Aplicar modelos de IA através de APIs para interpretar descrições textuais e protótipos visuais fornecidos pelos utilizadores.
- Avaliar a eficácia da solução na geração de código funcional.
- Identificar as vantagens e limitações da integração de LLMs em processos de desenvolvimento de software.

Relacionado à definição de protótipo, o dicionário de Língua Portuguesa o define como “primeiro tipo ou primeiro exemplar; padrão” ou ainda como “exemplar único feito para ser experimental antes da produção de outros exemplares” [19]. Outra definição seria a versão preliminar de um sistema ou programa de computador e em [97], este termo pode ser utilizado em uma variedade de contextos além de uma primeira versão, pode ser utilizado para semântica, design, eletrônica e programação de software, usado principalmente para se validar uma ideia.

No contexto do projeto desenvolvido, o protótipo visual pretendido, num primeiro momento, poderá ser um desenho de formas com a aparência do que se pretende como resultado final e poderá conter instruções de funcionamento para auxílio e interpretação do que se pretende. O desenho poderá ser feito à mão ou com uso de software de edição de imagens, onde podemos classificar como protótipo de baixa fidelidade ou como protótipos de alta fidelidade, conforme a Figura 9.

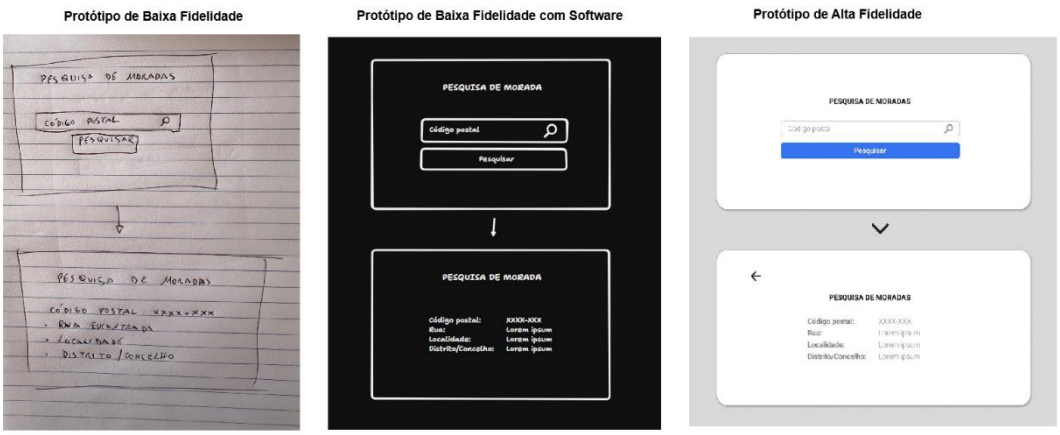

Figura 9 - Exemplos de protótipos visuais

Conforme o software apresenta os resultados, o utilizador poderá adicionar mais instruções ao *prompt* ou ajustes no protótipo, solicitando ajustes visuais ou de funcionalidade, desta maneira a refinar o resultado final.

4.2. REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Nesta secção serão abordados os requisitos necessários para o projeto final, como prova de conceito ou protótipo. A especificação de requisitos tem como objetivo obter um projeto de melhor qualidade e que ao mesmo tempo satisfaçam as necessidades e regras do resultado desejado. Conforme abordado na secção de Engenharia de Requisitos, estes são divididos em dois: os requisitos funcionais e os requisitos não funcionais [14].

4.2.1. REQUISITOS FUNCIONAIS

Os requisitos funcionais da aplicação foram definidos com o objetivo de garantir que o sistema atenda às necessidades dos utilizadores de forma eficiente e intuitiva, conforme descrito abaixo:

1. **Receção de Entradas pelo Utilizador**

   O sistema deve permitir que os utilizadores introduzam descrições textuais (*prompts*) ou carreguem protótipos visuais do produto desejado, assegurando flexibilidade na forma de especificação das suas necessidades.

2. **Processamento de Entradas e Geração de Código**

   O sistema deve processar as entradas fornecidas pelos utilizadores através de uma API baseada em LLMs, como o *ChatGPT* [67], *Claude* [68] ou *Gemini* [69], com o objetivo de gerar o código correspondente ao produto ou funcionalidade pretendida.

3. **Apresentação de Resultados ao Utilizador**

O sistema deve fornecer feedback visual e textual aos utilizadores, apresentando o código gerado de forma clara e acessível, bem como permitindo pré-visualizações interativas do resultado, quando aplicável.

4. **Armazenamento de Histórico de Interações**

O sistema deve armazenar o histórico de interações, incluindo os prompts, os protótipos visuais carregados e os resultados gerados, para possibilitar consultas futuras e análises posteriores.

4.2.2. Requisitos Não Funcionais

Os requisitos não funcionais especificam as características de qualidade que o sistema deve atender, garantindo um desempenho adequado, segurança e uma experiência satisfatória para os utilizadores. Estes requisitos incluem:

1. **Interface Intuitiva e Responsiva**

A interface do sistema deve ser intuitiva e de fácil utilização, proporcionando uma experiência fluída aos utilizadores. Além disso, deverá ser responsiva, garantindo compatibilidade e uma apresentação visual adequada em diferentes dispositivos, incluindo computadores, tablets e smartphones.

2. **Desempenho e Tempo de Resposta**

O tempo de resposta para o processamento das entradas e a geração dos resultados não deverá exceder dois minutos, mesmo em cenários de maior complexidade ou carga. Este limite visa assegurar uma experiência eficiente e minimizar frustrações para os utilizadores.

3. **Segurança e Acesso Restrito**

O sistema deverá garantir a segurança e permitir o acesso às funcionalidades e históricos mediante autenticação e validação de utilizador previamente registado.

4. **Modularidade e Extensibilidade**

A arquitetura do sistema deverá ser modular, possibilitando a integração futura com outras APIs ou funcionalidades, sem comprometer a estabilidade ou a eficiência do sistema existente. Esta modularidade permitirá a evolução do sistema de forma escalável e flexível.

4.3. Interface Principal

A interface principal foi projetada para proporcionar uma experiência simples e funcional para o utilizador e serve como o ponto de interação direto entre o utilizador e o sistema, permitindo a introdução de dados, como descrições textuais e protótipos visuais, e a exibição de resultados gerados, como o código correspondente e visualizações do produto final. O design prioriza a usabilidade e a

responsividade, assegurando uma navegação fluida em diferentes dispositivos e resoluções, ao mesmo tempo que mantém a coerência visual e funcional em toda a aplicação.

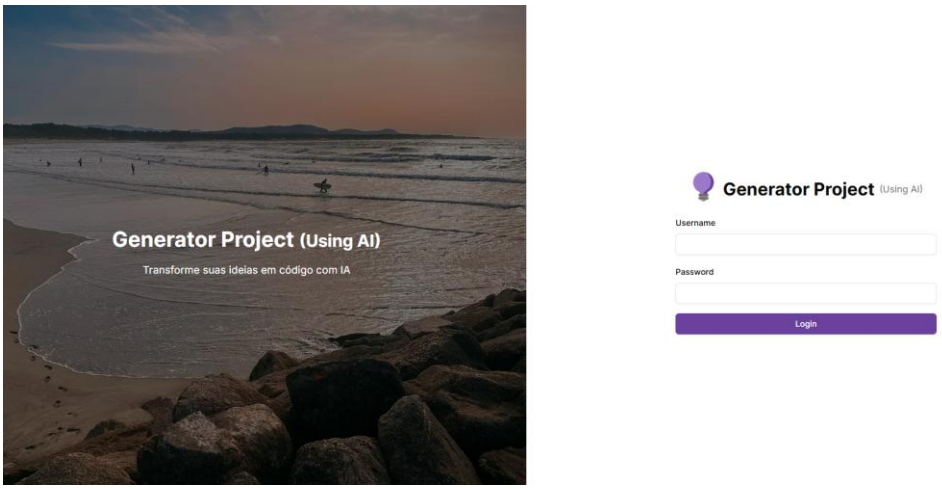

Figura 10 - Login do utilizador

Na Figura 10 - Login é possível ver o ecrã inicial da aplicação com o formulário de autenticação do utilizador. Embora seja possível a realização de registo de novos utilizadores, esta funcionalidade não está disponível por não estar definida no escopo do projecto.

Nota: o projecto foi nomeado como *Generator Project* sem finalidade comercial, apenas para o âmbito desta dissertação.

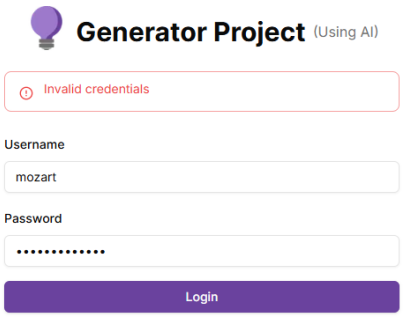

Figura 11 - Erro de credenciais

Em caso de falha na autenticação do utilizador devido a credenciais não encontrada, aparece uma alerta conforme a Figura 11 - Erro de credenciais.

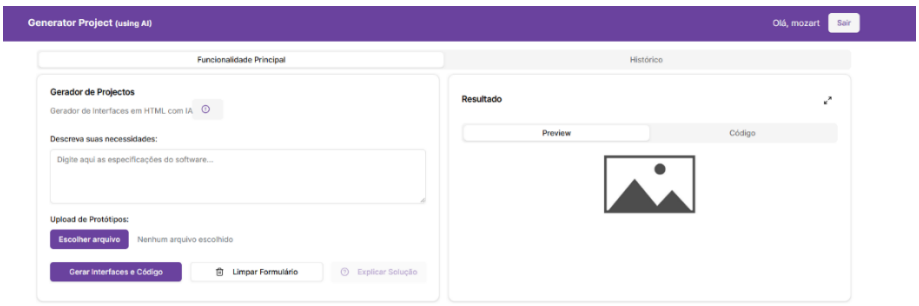

Figura 12 - Interface do projecto

Na Figura 12 - Interface do projecto é possível verificar os elementos principais que incluem:

- **Barra superior**: Exibe o nome do projecto, o nome do utilizador autenticado e um botão de sair.
- **Área de Funcionalidade Principal**: Divisão esquerda do projeto responsável por receber o *prompt* do utilizador e a imagem do protótipo.
- **Campo de Texto**: Onde o utilizador poderá introduzir descrições das suas necessidades.
- **Área de Upload de Protótipo**: Permite o carregamento de protótipos visuais.
- **Botão Gerar Interfaces e Código**: Envia as entradas para processamento no back-end do projecto.
- **Botão Limpar Formulário**: Permite a limpeza do formulário para possibilitar uma nova interação.
- **Botão Explicar Solução**: Permite a visualização de uma explicação do código gerado.
- **Área de Resultado**: Exibe a pré-visualização do resultado e um separador com o código gerado.
- **Área Histórico**: Lista as interacções anteriores do utilizador.

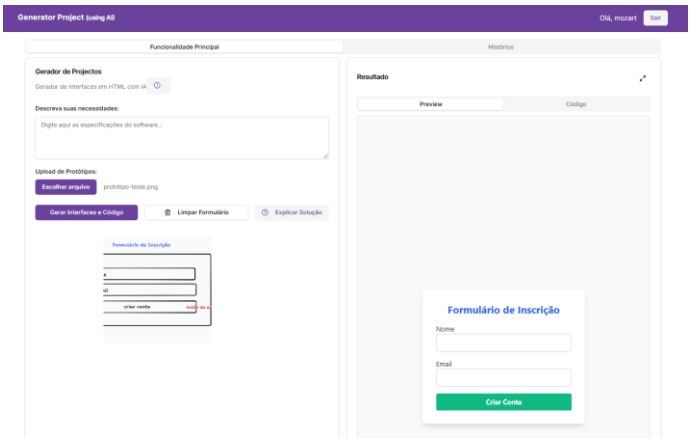

Figura 13 - Envio de protótipo e resultado

Na Figura 13 - Envio de protótipo e resultado é apresentado o envio de um protótipo e como resultado, é possível ver na lateral uma pré-visualização do resultado. A visualização do código necessário para implementação desta proposta de solução está disponível no separador Código, conforme a Figura 14 - Visualização do código gerado:

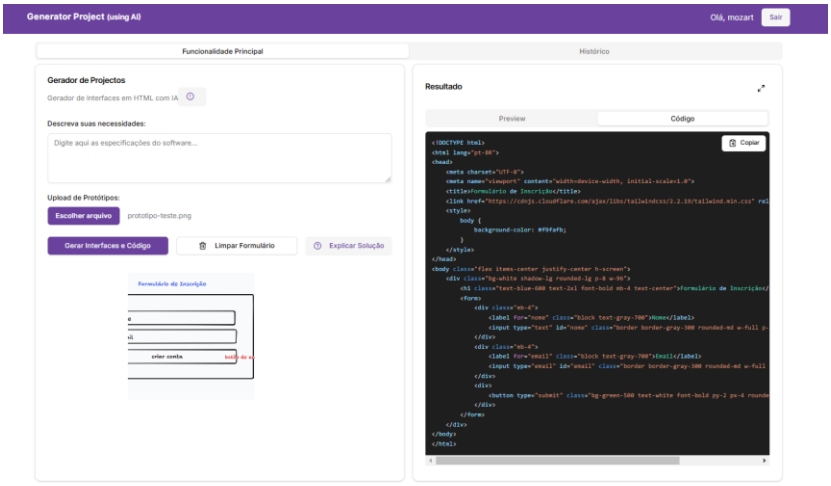

Figura 14 - Visualização do código gerado

A Figura 15 - Página de histórico por sua vez, mostra o histórico de interações com a ferramenta:

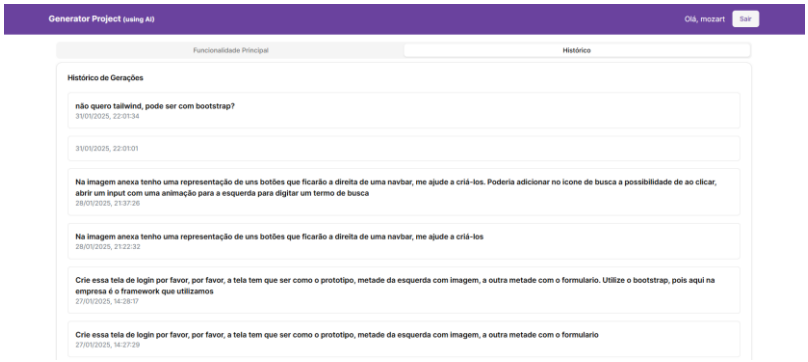

Figura 15 - Página de histórico

Nas secções Desenvolvimento e Integração e Testes, será possível ver mais detalhes da interface do projecto.

4.4. ESPECIFICAÇÃO

Nesta secção será especificado os principais componentes do caso de estudo e as suas interacções, de maneira a detalhar as funcionalidades e responsabilidades de cada módulo. A arquitectura modular proposta garante flexibilidade, escalabilidade e uma integração eficiente com as ferramentas de IA utilizadas.

1. Módulo de Entrada

Este módulo é responsável por receber e validar os dados fornecidos pelos utilizadores, que podem incluir:

- **Texto (prompt):** Descrições detalhadas do produto ou resultado esperado, com o fornecimento do contexto para o sistema interpretar.
- **Imagens (protótipos visuais):** Representações visuais que complementam ou substituem o texto como fonte de informação.

O módulo de entrada deverá validar os dados para assegurar que estão no formato correcto, identificando possíveis erros, como imagens corrompidas ou textos incompletos. Além disso, deverá realizar pré-processamentos simples, como redimensionamento de imagens ou limpeza de texto, para optimizar a interacção com o módulo de processamento.

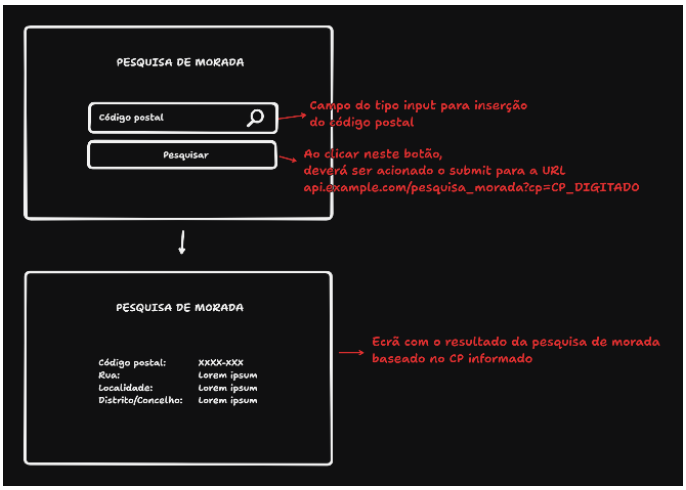

Figura 16 - Protótipo com comentários

A Figura 16 - Protótipo com comentários demonstra um protótipo com uso de anotações que substituem o *prompt*, desde que sejam indicados o comportamento das diversas partes com o uso de comentários. Para esta situação, foi padronizado o uso de cor vermelha nos textos indicativos, conforme exemplo, para evitar que estes textos sejam confundidos com os textos necessários para o resultado final.

2. Módulo de Processamento

Este módulo estabelece a ligação com a API de LLM seleccionada, de maneira a garantir a comunicação necessária para interpretar e processar as entradas fornecidas. As suas principais funções incluem:

- **Conversão de Entradas:** Transforma os dados recebidos pelo módulo de entrada em formatos compreensíveis pela API de LLM.

- **Processamento via LLM:** Envia os dados à API e recebe os resultados processados, que podem incluir código gerado, recomendações ou análises baseadas no prompt e na imagem carregada.
- **Gestão de Erros:** Lida com respostas inesperadas ou erros da API, assegurando que os utilizadores recebem feedback claro e construtivo.

3. Módulo de Validação e Ajuste

Após o processamento, os resultados gerados são apresentados ao utilizador para revisão e ajuste. Este módulo oferece:

- **Visualização do resultado:** Uma interface que exibe uma pré-visualização do resultado gerado.
- **Visualização do código:** Uma interface que exibe o código gerado de forma simples e funcional.
- **Funcionalidade de ajustes:** Permite que os utilizadores forneçam novas instruções para refinar os resultados.

4. Módulo de Armazenamento

O módulo de armazenamento gere o histórico de interacções e resultados, permitindo consultas futuras e análise do desempenho do sistema. Este módulo inclui:

- **Base de Dados:** Armazena de forma segura os dados dos utilizadores, incluindo *prompts*, imagens carregadas, código gerado e ajustes efectuados.
- **Gestão de Registos:** Regista as interacções do sistema para fins de auditoria, análise de utilização e melhoria contínua.

4.5. ARQUITECTURA

A arquitectura do projecto foi concebida com base numa abordagem modular e escalável, composta por dois componentes principais: o **front-end**, responsável pela interface do utilizador e pelas interacções, e o **back-end**, que trata do processamento das entradas, integração com modelos de IA e gestão do histórico de interacções. Esta separação assegura uma estrutura flexível e eficiente, o que facilita o desenvolvimento e a manutenção.

4.5.1. CAMADA BACK-END

A camada de back-end proposta complementa as funcionalidades do front-end, oferecendo suporte para o processamento avançado das entradas e integração com modelos de IA através da utilização de APIs LLM (Ver Figura 17 - Camada back-end).

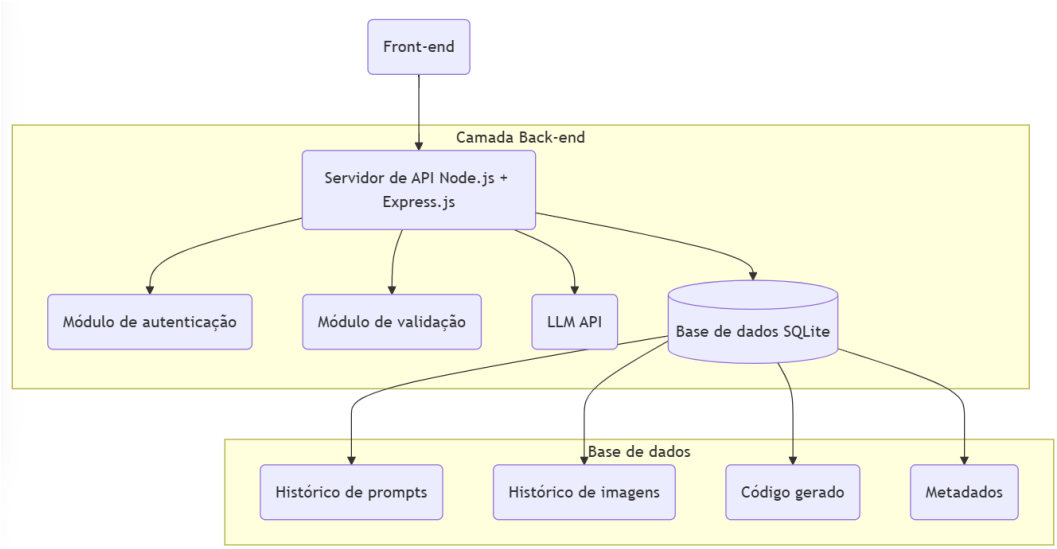

Figura 17 - Camada back-end

Servidor de API: Propõe-se a implementação de um servidor baseado em Node.js e uso de frameworks como Express.js para criar endpoints RESTful [73]. Este servidor será responsável por:

a. Autenticar utilizadores com uso de *tokens* JWT [74] e desta forma permitir o uso de recursos mediante validação de autorização.
b. Receber e validar as entradas fornecidas pelo front-end (*prompts* e imagens).
c. Encaminhar as solicitações para a API de LLM.
d. Tratar e formatar as respostas da API para serem exibidas ao utilizador de forma clara e estruturada.
e. Validar

Armazenamento de Dados: A proposta inclui a utilização de uma base de dados, inicialmente o SQLite [60], para:

a. Armazenar históricos de prompts e caminho de armazenamento das imagens carregadas.
b. Guardar os códigos gerados para consultas futuras.
c. Registar metadados, como *timestamps* e informações do utilizador.

Este armazenamento garante persistência dos dados e permite aos utilizadores rever interacções passadas ou continuar trabalhos previamente iniciados.

4.5.2. CAMADA FRONT-END

O front-end foi desenvolvido utilizando tecnologias modernas, sendo projetado para proporcionar uma experiência do utilizador intuitiva, funcional e responsiva. Na Figura 18 - Camada front-end é possível perceber como esta camada se divide nas diversas tecnologias utilizadas:

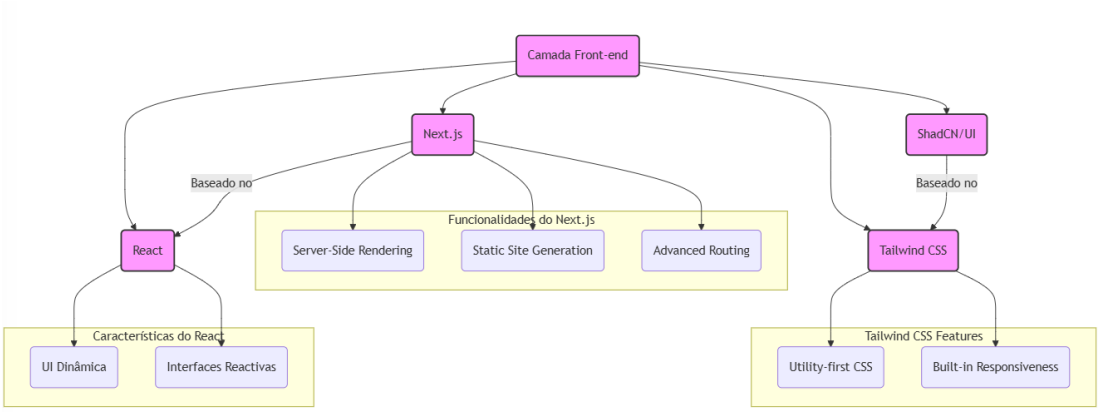

Figura 18 - Camada front-end

A aplicação baseia-se no React [53] e em ferramentas relacionadas, incluindo:

Next.js [76]: Um framework React que oferece renderização do lado do servidor (Server-Side Rendering), geração de sites estáticos (Static Site Generation) e um sistema avançado de roteamento, garantindo um desempenho otimizado e compatibilidade com SEO.

React [53]: Uma biblioteca JavaScript amplamente utilizada para construir interfaces de utilizador dinâmicas e reactivas.

Tailwind CSS [70]: Um framework CSS utilitário que permite estilização rápida e consistente, com suporte integrado à responsividade.

ShadCN/UI [71]: Biblioteca de componentes visuais baseado no Tailwind CSS.

Em síntese, a camada front-end do sistema combina tecnologias amplamente reconhecidas no desenvolvimento web para garantir uma experiência do utilizador de alta qualidade. A utilização de React e Next.js proporciona uma base sólida para interfaces reactivas e otimizadas, enquanto Tailwind CSS e ShadCN/UI asseguram uma estilização eficiente e responsiva, permitindo que o design seja visualmente atraente e funcional em diferentes dispositivos. Esta abordagem integrada e modular reforça a escalabilidade e a manutenibilidade da aplicação, alinhando-se aos objetivos de desempenho e usabilidade estabelecidos para o projeto.

4.5.3. DIAGRAMAS

Nesta secção, são apresentados os diagramas que ilustram a estrutura modular, o fluxo de informações do sistema e o diagrama da base de dados. Estes diagramas são cruciais para compreender a interacção entre os componentes principais e como os dados são processados e apresentados ao utilizador.

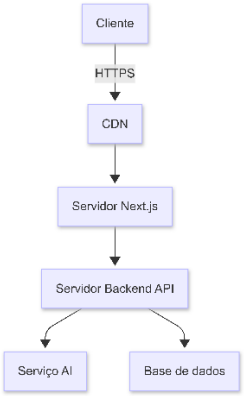

Figura 19 - Diagrama da Arquitetura

A Figura 19 - Diagrama da Arquitetura apresenta a arquitectura geral do sistema, destacando os principais componentes e as suas interacções [15]:

1. **Cliente**: O utilizador interage com o sistema através de uma interface web, utilizando protocolos seguros (HTTPS).
2. **CDN (Content Delivery Network)**: Responsável por acelerar a entrega dos recursos estáticos do sistema, garantindo uma experiência fluida e rápida.
3. **React Application (Front-end)**: Representa a interface do utilizador, construída com React. Este componente lida com a entrada de dados (prompts e imagens) e a exibição dos resultados gerados.
4. **Backend API Server**: Um servidor responsável pelo processamento das entradas recebidas, ligação com os serviços de IA e gestão do histórico de interacções.
5. **AI Service**: Integra um modelo de LLM (como GPT-3 ou GPT-4), utilizado para processar os prompts e gerar código.
6. **Database**: Armazena os dados históricos, como interacções e resultados gerados, para consulta e iteração futura.

Este diagrama evidencia a separação lógica entre os componentes, demonstrando a modularidade e escalabilidade da solução.

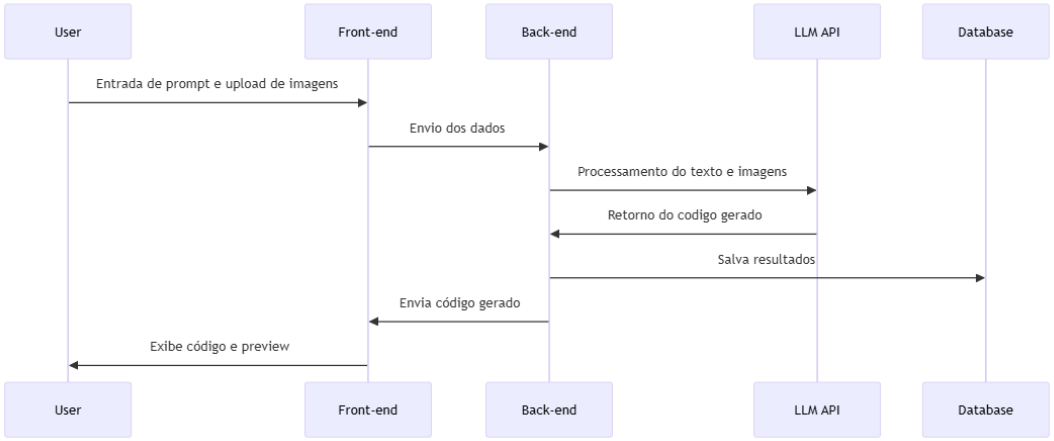

Figura 20 - Diagrama de Fluxo de Dados

A Figura 20 - Diagrama de Fluxo de Dados ilustra o fluxo de dados no sistema, desde a entrada do utilizador até à exibição dos resultados gerados. O fluxo é descrito de forma sequencial [15]:

1. **Entrada de dados**: O utilizador introduz um prompt textual e/ou carrega uma imagem através do frontend.
2. **Envio ao back-end**: O fron-tend valida as entradas e encaminha os dados para o back-end.
3. **Processamento pela API de LLM**: O back-end integra-se com o serviço de IA, que processa as entradas e retorna o código gerado.
4. **Armazenamento em database**: O back-end guarda as interacções e resultados na base de dados para futuras consultas.
5. **Exibição de resultados**: O front-end apresenta o código gerado ao utilizador, juntamente com uma pré-visualização funcional no iframe.

Este diagrama demonstra a comunicação clara e eficiente entre os componentes, garantindo um processamento ágil e seguro das informações.

Por fim, a Figura 21 - Diagrama da base de dados apresenta a o diagrama Entity Relationship Diagram (ERD) [1] do projecto, que é uma representação visual da estrutura da base de dados, mostrando as tabelas, os seus campos, os tipos de dados, as chaves primárias, as chaves estrangeiras e as relações entre as tabelas [17]:

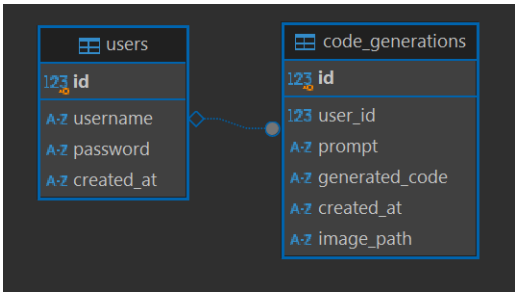

Figura 21 - Diagrama da base de dados

O diagrama apresentado ilustra a estrutura da base de dados do sistema, composta por duas tabelas principais: **users** e **code_generations**, que estão relacionadas por uma associação de chave estrangeira.

A tabela **users** armazena informações sobre os utilizadores registados, com os seguintes campos:

- **id** (chave primária),
- **username** (nome de utilizador),
- **password** (senha armazenada de forma segura, idealmente utilizando hash), e
- **created_at** (data de criação do registo).

A tabela **code_generations** regista as interacções dos utilizadores com o sistema, contendo os campos:

- **id** (chave primária),
- **user_id** (chave estrangeira que referencia a tabela **users**),
- **prompt** (o texto de entrada fornecido pelo utilizador),
- **generated_code** (o código gerado com base no prompt),
- **image_path** (o caminho da imagem utilizada), e
- **created_at** (data de criação do registo).

Esta relação assegura que cada registo na tabela **code_generations** está associado a um utilizador específico, permitindo um controlo eficiente e a rastreabilidade das interacções realizadas no sistema.

4.6. DESENVOLVIMENTO

Esta secção descreve o desenvolvimento da solução e visa destacar as etapas realizadas, as ferramentas e tecnologias utilizadas, e as boas práticas adoptadas para garantir a implementação eficiente da solução. O desenvolvimento foi dividido em fases principais, que abrangem desde a configuração inicial do ambiente até à implementação dos módulos funcionais.

4.6.1. CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

A configuração do ambiente de desenvolvimento foi uma etapa essencial para garantir a implementação eficiente da solução. Esta fase incluiu a instalação das ferramentas necessárias, bem como a configuração inicial do sistema. Os passos realizados incluem:

1. **Instalação do Noje.js**:

O Node.js foi instalado como base para o desenvolvimento do servidor back-end e para gerir as dependências do projeto. O processo seguiu os seguintes passos:

- Acesso ao site oficial do Node.js[1] para download da versão Long Term Support (LTS), garantindo maior estabilidade.
- Após o download, foi executado o instalador, seguindo as instruções padrão para o sistema operativo em uso.
- A instalação foi verificada através do terminal, utilizando os comandos:

node -v

npm -v

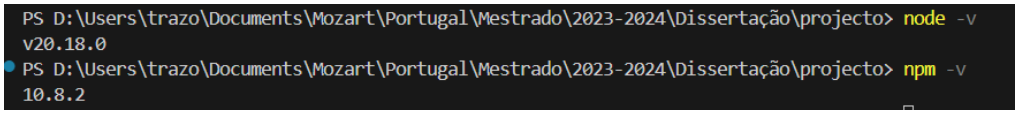

Figura 22 - Versão Node.js

Na Figura 22 - Versão Node.js demonstra-se o uso dos comandos para confirmar as versões instaladas do Node.js e do NPM.

2. **Configuração do back -end**:

Para o projecto de back-end, criou-se um diretório chamado *project-generator-backend* e o projecto quando estiver em execução, deverá estar disponível em um link visível ao front-end. O back-end foi inicializado utilizando o Node.js com a instalação do framework Express.js outras dependências como o SQLite, OpenAI e Dotenv, para acesso a variáveis de ambiente do projecto.

Criação do diretório:

mkdir project-generator-backend

cd project-generator-backend

Inicialização da aplicação e instalação de dependências:

npm init -y

npm install express cors dotenv bcrypt jsonwebtoken sqlite sqlite3 openai

npm install --save-dev typescript @types/express @types/cors @types/bcrypt @types/jsonwebtoken @types/sqlite3 ts-node-dev dotenv

Neste momento, o projecto de back-end está inicializado. Antes de prosseguir com o seu desenvolvimento, deixaremos o projecto de front-end inicializado.

3. **Inicialização do projecto front-end**:

Criou-se o diretório do projeto e inicializou-se o ambiente React utilizando o Next.js. No terminal, foram executados os seguintes comandos:

npx create-next-app@latest project-generator

[1] Node JS: https://nodejs.org/

cd project-generator

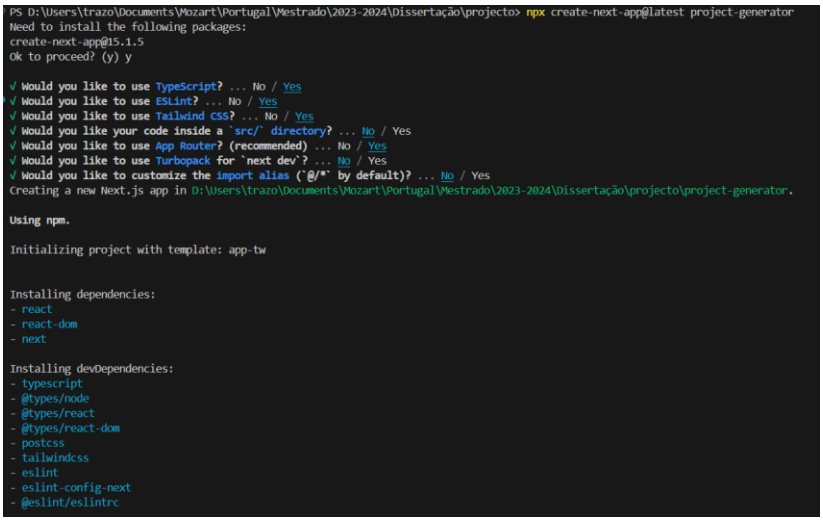

Figura 23 - Inicialização do front-end

Na Figura 23 - Inicialização do front-end é possível verificar o processo de configuração inicial, bem como a instalação de algumas dependências como o *Tailwind CSS* que é utilizada para estilização do projecto. Entretanto algumas dependências ainda são necessárias para que se tenha o resultado esperado, como a instalação de uma biblioteca chamada *ShadCN/UI,* que possui um conjunto de elementos visuais que serão úteis para um desenvolvimento mais fluente.

Dependências adicionais:

npm install @radix-ui/react-tabs @radix-ui/react-slot class-variance-authority clsx lucide-react react-syntax-highlighter tailwind-merge tailwindcss-animate

Configuração do *ShadCN/UI*:

npx shadcn@latest init

npx shadcn@latest add button card textarea tabs input label dialog alert

4.6.2. DESENVOLVIMENTO DO BACK-END

A camada de back-end desempenha um papel crucial no controlo e gestão global do projeto. Tal como descrito anteriormente, o fluxo de operações no back-end depende diretamente das interações do utilizador. Para além da criação e integração com as APIs de LLMs, como a OpenAI, serão implementados mecanismos de autenticação e autorização para garantir a segurança e integridade do sistema.

Além disso, pretende-se armazenar as interações dos utilizadores ao longo do tempo, possibilitando a gestão incremental das versões de código gerado. Este método de armazenamento incremental assegura que o histórico de versões esteja disponível para consultas futuras, promovendo a rastreabilidade e a evolução dos projetos desenvolvidos. A Figura 21 - Diagrama da base de dados apresenta o modelo da base de dados proposto, que serviu como suporte ao desenvolvimento do back-end.

Com base no modelo da base de dados apresentado, foi possível avançar no desenvolvimento do back-end, estruturando uma API própria que orquestra as funcionalidades principais do sistema. Inicialmente, foram definidos os seguintes *endpoints* da API:

1. **Registro de Utilizador**:

   POST [*{url}*/api/users/register]

   Este *endpoint* permite o registo de novos utilizadores na aplicação. O sistema valida os dados fornecidos e cria um novo registo na base de dados.

2. **Login de Utilizador**:

   POST [*{url}*/api/users/login]

   Este *endpoint* verifica a existência do utilizador na base de dados. Caso o registo seja válido, é gerado e retornado um token de autenticação, permitindo o acesso às funcionalidades do sistema.

3. **Geração de Código**:

   POST [*{url}*/api/code-generation/generate]

   Após o utilizador estar autenticado, é possível realizar chamadas à API para geração de código com base nos dados fornecidos (texto e/ou protótipo visual).

4. **Histórico de Gerações**:

   GET [*{url}*/api/code-generation/history]

   Este *endpoint* retorna o histórico de interações realizadas pelo utilizador, incluindo os *prompts* fornecidos e os códigos gerados.

Nota: Nos endpoints apresentados, *{url}* é uma representação genérica que deverá ser substituída pelo endereço base da aplicação do back-end.

O projeto foi desenvolvido, inicialmente, com integração exclusiva à API da OpenAI. No entanto, a arquitetura modular do back-end permite que outras APIs, como *Claude* [68] ou *Gemini* [69], sejam integradas facilmente no futuro. Mais detalhes sobre possíveis expansões são discutidos na secção Conclusões e Trabalho Futuro.

Estrutura de Diretórios

Após a implementação dos componentes principais, a estrutura de diretórios do sistema back-end foi organizada para facilitar a manutenção, escalabilidade e integração de novas funcionalidades. A Figura 24 - Estrutura do back-end apresenta a estrutura de diretórios resultante:

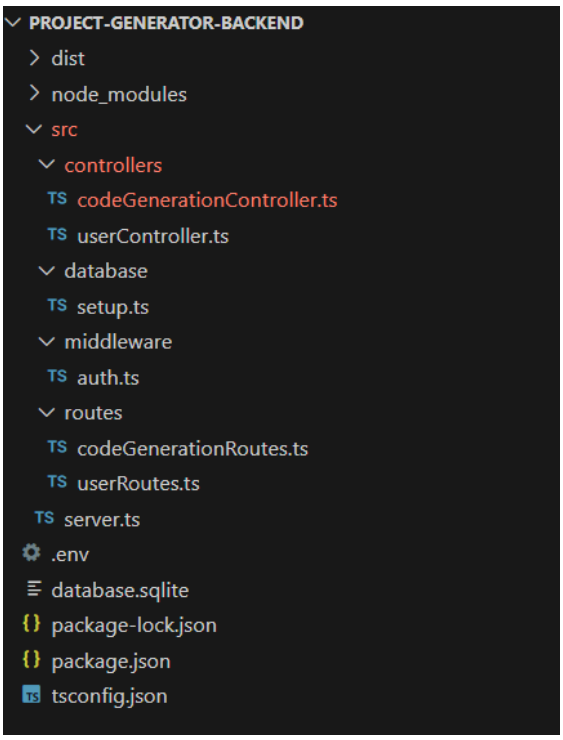

Figura 24 - Estrutura do back-end

A organização dos diretórios do projecto *Project-Generator-Backend* segue boas práticas de desenvolvimento, garantindo modularidade, escalabilidade e facilidade de manutenção. Cada pasta e ficheiro têm uma função específica no contexto do projeto.

Diretórios Principais

1. *dist/*

Este diretório contém os ficheiros transpilados (gerados a partir do *TypeScript* [75]) em JavaScript. Estes ficheiros são o resultado do processo de compilação e representam o código que será executado no ambiente de produção.

2. *node_modules/*

Diretório gerado automaticamente pelo NPM, contendo todas as dependências instaladas que o projeto requer para funcionar. Estes módulos são utilizados para suportar funcionalidades específicas, como conexão com base de dados, integração com APIs, entre outros.

3. *src/*

Diretório principal onde se encontra todo o código fonte do back-end. Este diretório está dividido em subdiretórios e ficheiros específicos que organizam as funcionalidades do projeto.

   a. controllers/

Esta pasta contém os controladores da aplicação. Os controladores são responsáveis por gerir a lógica das requisições HTTP recebidas e a interação com outros componentes, como a base de dados ou serviços externos.

- *codeGenerationController.ts*: Controlador responsável pela lógica de geração de código a partir dos dados fornecidos pelos utilizadores.
- *userController.ts*: Controlador responsável por gerir as operações relacionadas aos utilizadores, como registo e login.

b. database/

Diretório que centraliza a configuração e inicialização da base de dados.

- setup.ts: Contém o código para configurar e inicializar a ligação à base de dados SQLite.

c. middleware/

Pasta destinada aos middlewares, que são funções intermediárias executadas durante o processamento das requisições HTTP.

- *auth.ts:* Middleware que implementa a lógica de autenticação e autorização, garantindo que apenas utilizadores autenticados tenham acesso a determinadas rotas.

d. routes/

Este diretório organiza as definições de rotas da aplicação, associando cada rota a um controlador específico.

- *codeGenerationRoutes.ts:* Define as rotas relacionadas à geração de código.
- *userRoutes.ts:* Define as rotas relacionadas às operações de utilizadores, como registo e login.

e. server.ts

Este ficheiro é o ponto de entrada principal da aplicação. Ele inicializa o servidor, configura os middlewares globais e define as rotas da API.

Ficheiros na raiz do projecto

1. .env

Ficheiro que armazena variáveis de ambiente, como chaves de API, URLs de base de dados ou outros dados sensíveis. Este ficheiro não deve ser incluído no controlo de versões por motivos de segurança.

2. database.sqlite

Ficheiro da base de dados SQLite. Contém as tabelas e dados utilizados pela aplicação.

3. *package.json* e *package-lock.json*

O package.json lista as dependências do projeto, os scripts disponíveis e as configurações gerais. O package-lock.json bloqueia as versões exatas das dependências instaladas, garantindo consistência entre diferentes ambientes.

4. tsconfig.json

Ficheiro de configuração do TypeScript, especificando como o código será transpilado para JavaScript e quais regras o compilador deve seguir.

A estrutura apresentada reflete uma organização modular e clara, separando responsabilidades por meio de pastas e ficheiros dedicados. Esta abordagem facilita a manutenção, permite a adição de novas funcionalidades e torna o projeto escalável para futuras melhorias.

4.6.3. INTEGRAÇÃO COM A OPENAI

A integração do back-end com a OpenAI foi realizada através da API do modelo *GPT-4o-mini [98]*, uma escolha estratégica devido ao seu custo reduzido e à capacidade de compreender tanto texto como imagens. Esta funcionalidade foi crucial para o projeto, pois permitiu processar não apenas as descrições textuais fornecidas pelo utilizador, mas também protótipos visuais, garantindo uma geração de código mais precisa e adaptada às necessidades específicas.

A comunicação com a API da OpenAI ocorre através de requisições *HTTP POST*, onde o back-end envia um *prompt* estruturado para o modelo e recebe como resposta um bloco de código gerado. A requisição segue o seguinte formato:

```
POST https://api.openai.com/v1/chat/completions
Headers:
  Authorization: Bearer {API_KEY}
  Content-Type: application/json
Body:
{
  "model": "gpt-4o-mini",
  "messages": [
    {"role": "system", "content": " PROMPT CONFIGURADO NO SERVIDOR."},
    {"role": "user", "content": " PROMPT DO UTILIZADOR"}
  ],
  "max_tokens": 500
}
```

Neste exemplo, o modelo recebe um contexto inicial e a entrada do utilizador, processa o pedido e retorna uma resposta com um trecho de código correspondente. A

API responde com um *JSON* contendo o código gerado, que pode ser exibido na interface do utilizador.

Na interação com a API da OpenAI [98], os *prompts* são enviados dentro de uma estrutura de mensagens organizadas em diferentes *roles* (funções): *system*, *user* e, opcionalmente, *assistant* (não foi utilizado neste projecto). Cada uma tem um papel específico na definição da resposta do modelo.

1. ***Role "System"*** (Instruções do Sistema)

   A mensagem enviada com a *role system* define o comportamento global do modelo durante a interação. É utilizada para orientar a forma como ele deve responder e estabelecer um tom adequado para as respostas.

   Objetivo: Configura o comportamento do modelo, garantindo que ele siga um estilo específico de resposta e evitando que gere respostas fora do contexto desejado.

2. ***Role "User"*** (Input do Utilizador)

   A *role user* representa a mensagem que vem diretamente do utilizador, ou seja, o comando ou pergunta que inicia a geração de conteúdo.

   Objetivo: Define a necessidade do utilizador e fornece detalhes específicos para a geração do código. Quanto mais estruturado for o input, melhor será o resultado gerado.

Fluxo de Integração com a API da OpenAI

O fluxo de integração entre o back-end e a API da OpenAI ocorre da seguinte forma:

1. O utilizador insere um *prompt* textual e/ou um protótipo visual na interface do front-end.
2. O back-end recebe a solicitação e verifica se o utilizador está autenticado.
3. Os dados são processados e enviados à API da OpenAI utilizando o modelo *GPT-4o-mini*.
4. A OpenAI processa o pedido e retorna um código gerado em conformidade com o *prompt* fornecido.
5. O back-end recebe a resposta e armazena o resultado na base de dados.
6. A interface do utilizador exibe o código gerado, permitindo edições manuais se necessário.

Embora a OpenAI tenha sido escolhida para este projecto, a arquitectura foi desenhada para permitir a integração com outras APIs de LLMs sem necessidade de grandes modificações. Alternativas como *Gemini* [69], da Google, e *Claude* [68], da Anthropic, podem ser facilmente adaptadas ao sistema, uma vez que seguem estruturas similares de comunicação via API *RESTful [73]*.

Caso a migração para um novo modelo seja necessária, apenas a camada de serviço de geração de código precisaria ser modificada, alterando os *endpoints* e os parâmetros das requisições para a nova API. Este design modular garante flexibilidade na escolha da melhor solução para necessidades futuras.

Importância do Prompt na Qualidade da Resposta do Modelo

A qualidade e a precisão do código gerado pelo modelo da OpenAI, como o *GPT-4o-mini*, ou qualquer outro modelo LLM, dependem fortemente da construção adequada do *prompt* enviado à API. Um *prompt* bem estruturado fornece contexto claro e instruções detalhadas ao modelo, permitindo que ele compreenda a necessidade do utilizador e produza um resultado mais relevante e útil conforme o autor em [39].

Ao elaborar um *prompt* eficaz, considera-se os seguintes aspetos:

1. **Contexto Claro:** O modelo necessita de uma introdução sobre a tarefa, permitindo que compreenda o objetivo da geração de código. Exemplo:
   "Preciso de um formulário de login em HTML e CSS responsivo, com um campo para e-mail, outro para senha e um botão de envio."
   Ao especificar os elementos essenciais, a resposta gerada será mais próxima do esperado [39].

2. Detalhe: Quanto mais detalhes forem incluídos, menor a chance do modelo gerar respostas genéricas ou imprecisas. Incluir frameworks ou tecnologias específicas melhora os resultados:

   "Crie um formulário de login utilizando Tailwind CSS para estilização e React para a lógica de validação dos campos."
3. Exemplos e Restrições: Se necessário, o prompt pode incluir exemplos de código ou definir limitações para evitar respostas muito amplas.
   "Crie um botão de envio estilizado com Tailwind CSS, utilizando `*bg-blue-500*` para o fundo e `*hover:bg-blue-700*` para o efeito ao passar o mouse."

Ao seguir essas diretrizes, o modelo consegue interpretar corretamente as necessidades do utilizador, gerando código funcional e próximo da expectativa [39].

4.6.4. TESTES DE EXECUÇÃO DOS ENDPOINTS

Após a implementação dos endpoints no back-end, é fundamental validar o seu correto funcionamento. Para este fim, utilizou-se a ferramenta Postman [99], que é amplamente utilizada para testar APIs RESTful [73]. O Postman permite enviar requisições HTTP aos endpoints da aplicação, inspecionar as respostas e identificar possíveis problemas de forma eficiente.

Configuração do Ambiente no Postman

Antes de iniciar os testes, configurou-se um ambiente no Postman com as seguintes variáveis (Figura 25 - Variáveis Postman):

Figura 25 - Variáveis Postman

- **Base URL**: Representa o endereço do servidor back-end (exemplo: *http://localhost:3001/).*
- **Token de Autenticação**: Gerado após o login do utilizador, utilizado para testar endpoints protegidos por autenticação.

Testes realizados

Foram testados os seguintes *endpoints*, conforme descrito na secção anterior:

1. **Registo de Utilizador**

   Endpoint: POST /api/users/register

   - **Descrição**: Envia dados do utilizador (nome de utilizador e senha) para registo na base de dados.

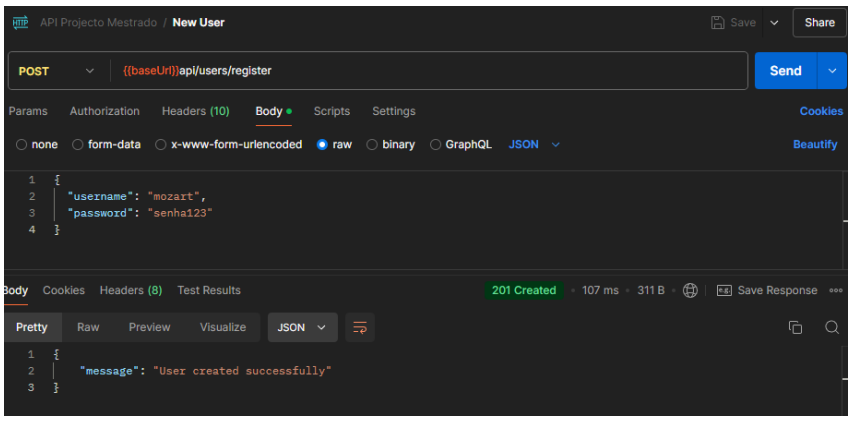

Figura 26 - API - Novo utilizador

Na Figura 26 - API - Novo utilizador é possível verificar que o servidor respondeu com um status *HTTP 201 Created*, indicando sucesso no registo.

2. **Login de Utilizador**

   Endpoint: POST /api/users/login

   - **Descrição**: Verifica as credenciais do utilizador e retorna um token de autenticação.

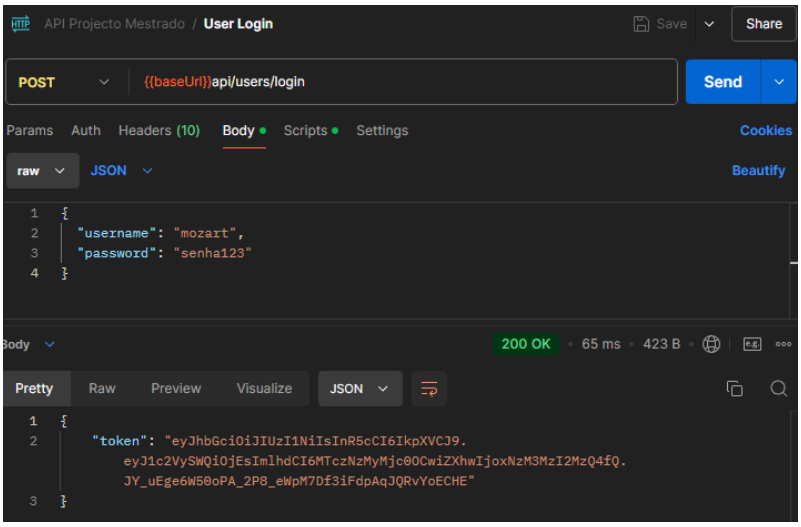

Figura 27 - API - Login

Conforme a Figura 27 - API - Login o servidor respondeu com um status *HTTP 200 OK* e um *token* válido.

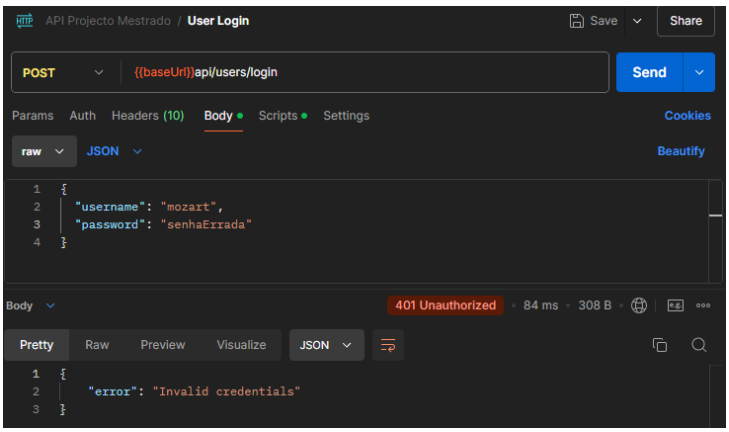

Figura 28 - API - Erro de credenciais

Na Figura 28 - API - Erro de credenciais se verifica a tentativa de autenticação com senha inválida. O servidor retorna um status *HTTP 401 Unauthorized*.

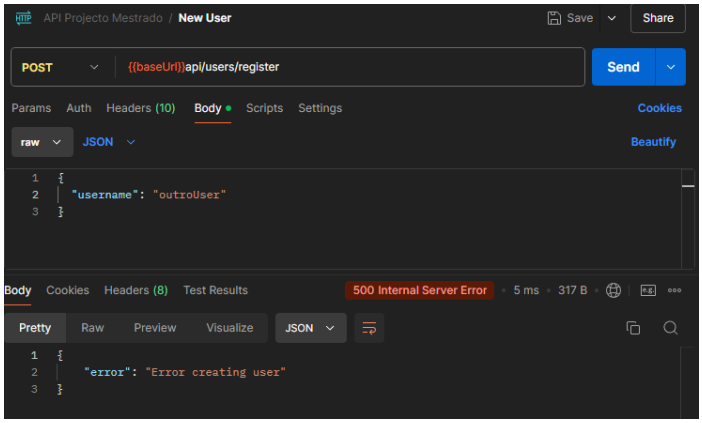

Figura 29 - API - Erro de criação de utilizador

A API executa ainda validação no formato de dados enviados. A Figura 29 - API - Erro de criação de utilizador mostra a tentativa de criação de utilizador sem envio de senha e o servidor retorna um status *HTTP 500 Internal Server Error*.

3. **Geração de Código**

   Endpoint: POST / api/code-generation/generate

   - **Descrição**: Gera código com base num prompt textual fornecido pelo utilizador autenticado.
   - Este *endpoint* exige um token válido para correta execução.

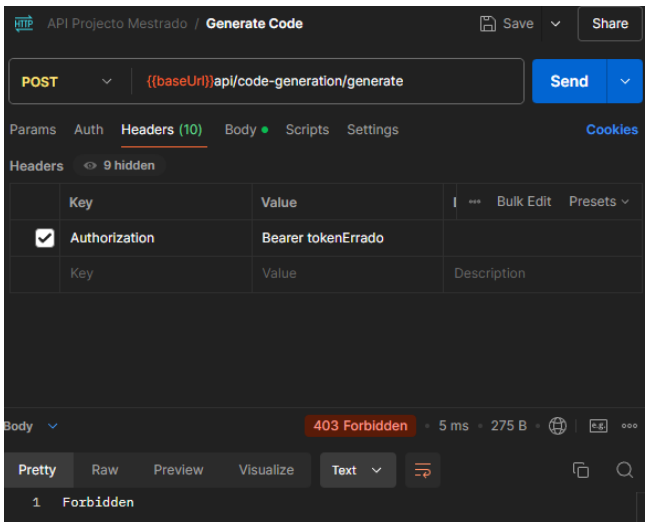

Figura 30 - API - Erro de token

A Figura 30 - API - Erro de token mostra uma execução mal sucedida devido ao uso de um *token* inválido e retorna um status *HTTP 403 Forbidden*.

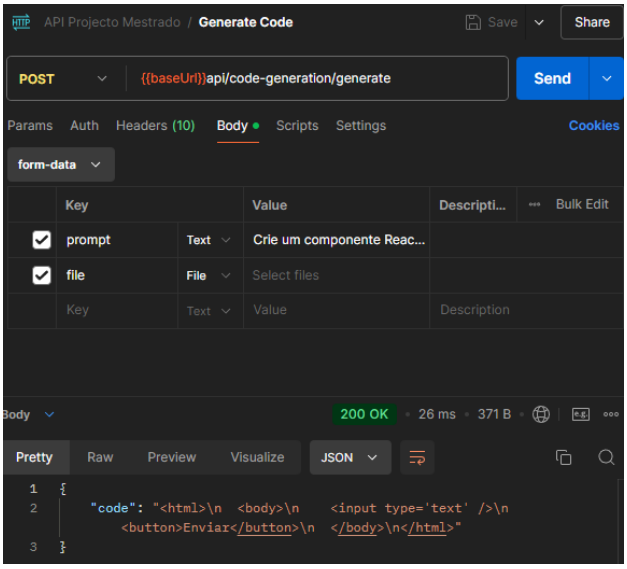

Figura 31 - API - Geração de código

Como se pode verificar na Figura 31 - API - Geração de código o *endpoint* respondeu com um status *HTTP 200 OK* e retornou o código gerado.

4. **Histórico de Gerações**

   Endpoint: POST /api/code-generation/history

   - **Descrição**: Retorna o histórico de interações do utilizador autenticado.
   - Este *endpoint* exige um *token* válido para correta execução.

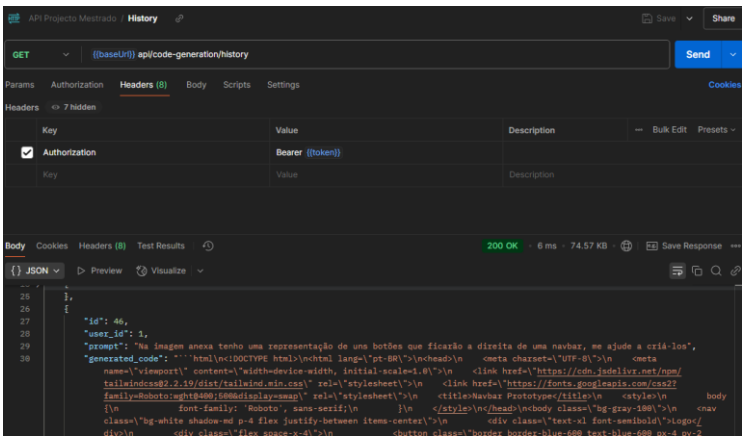

Figura 32 - API - Histórico

Na Figura 32 - API - Histórico o servidor respondeu com um status *HTTP 200 OK* e, embora não possua nenhum registo, é possível perceber que a chamada funciona adequadamente.

Cada *endpoint* foi invocado e os resultados foram comparados com as respostas esperadas. Foi dada especial atenção aos casos de erro, como:

- Dados de utilizadores inválidos no registo ou login.
- Tentativa de acesso sem autenticação ao histórico de gerações.

Os testes foram considerados bem-sucedidos quando os resultados retornaram os códigos de status e mensagens esperados. Este processo garante que cada funcionalidade essencial está operacional. Possibilita também identificar e corrigir problemas, como erros de validação de dados e respostas inesperadas em cenários limite. A implementação de testes sistemáticos contribui significativamente para a qualidade e fiabilidade da aplicação.

4.6.5. Desenvolvimento do Front-end

O desenvolvimento do front-end seguiu uma abordagem iterativa e centrada no utilizador, garantindo uma experiência fluída, responsiva e acessível. A interface foi projetada para ser intuitiva, permitindo aos utilizadores interagir facilmente com o sistema, introduzindo descrições, carregando protótipos visuais e obtendo o código gerado pela API de LLM. O desenvolvimento foi conduzido com o uso de *React* e *Next.js*, beneficiando-se da renderização eficiente e da organização modular dos componentes.

Estruturação da Interface

A implementação do front-end foi estruturada em torno de componentes modulares do *React*, o que facilita a escalabilidade e a manutenção do código. Os principais elementos desenvolvidos incluem:

- **Campo de Entrada de Texto**: Permite ao utilizador introduzir descrições textuais detalhadas que servem como entrada para a geração de código. Este campo foi implementado como um componente controlado em React, garantindo uma experiência fluida e reativa.
- **Área de Upload de Imagens**: Suporta o carregamento de protótipos visuais e disponibiliza uma pré-visualização em tempo real, permitindo ao utilizador verificar as imagens antes da submissão. A funcionalidade de pré-visualização foi implementada utilizando o FileReader API do JavaScript.
- **Botão de Submissão**: Envia os dados introduzidos para o back-end, acionando a integração com a API de LLM. Para garantir uma experiência fluida, foi adicionada uma indicação visual de carregamento, melhorando a usabilidade.
- **Área de Resultados**: Exibe o código gerado pelo modelo de LLM e fornece um iframe embutido para pré-visualização funcional do resultado. Adicionalmente, foi incluída uma opção para copiar o código gerado para a área de transferência, facilitando a reutilização pelo utilizador.

Estilização e Responsividade

A interface foi projetada para ser totalmente responsiva, garantindo um bom desempenho em diferentes dispositivos e resoluções. Para isso, foram utilizadas as seguintes tecnologias:

- **Tailwind CSS**: Framework CSS utilitário que permite uma estilização rápida e consistente, reduzindo a dependência de estilos personalizados e classes CSS globais.
- **ShadCN/UI**: Conjunto de componentes pré-estilizados que garantem um design moderno e compatível com padrões de acessibilidade.

Gestão de Estado

O estado da aplicação é gerido de forma eficiente utilizando os *hooks* do *React*, nomeadamente:

- *useState*: Para armazenar os valores introduzidos pelo utilizador, as imagens carregadas e os resultados gerados.
- *useEffect*: Para gerir efeitos colaterais, como a recuperação automática do histórico de interações do utilizador.

- *useCallback*: Para otimizar funções que são recriadas em diferentes renderizações, garantindo melhor desempenho e evitando reprocessamento desnecessário.

Estrutura de Diretórios

A estrutura de diretórios do front-end foi organizada de acordo com as boas práticas do Next.js, o que permite o correto roteamento entre os diversos componentes. A Figura 33 - Estrutura do front-end apresenta a estrutura de diretórios final:

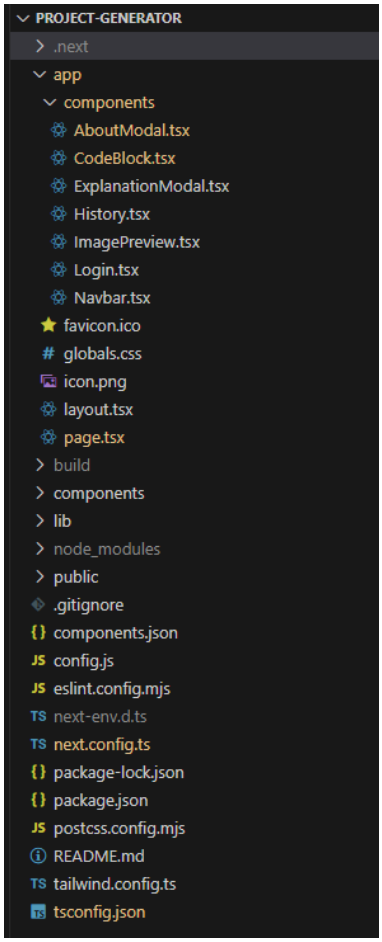

Figura 33 - Estrutura do front-end

Abaixo está a descrição detalhada de cada diretório e ficheiro relevante.

Diretórios Principais

1. *.next/*
   - Diretório gerado automaticamente pelo **Next.js** durante a construção e execução da aplicação.

- Contém ficheiros de **build**, cache e otimizações para melhorar o desempenho da aplicação.

2. *app/*
   - Diretório principal da aplicação, onde se encontram as páginas e componentes principais.
   - **Subdiretórios e ficheiros principais:**
     - *components*/: Contém os componentes reutilizáveis utilizados na interface do utilizador.
     - *globals.css*: Estilos globais da aplicação.
     - *layout.tsx:* Define a estrutura padrão da aplicação, incluindo cabeçalho e rodapé.
     - *page.tsx:* Ficheiro principal da página inicial da aplicação.
3. *build/*
   - Diretório utilizado para armazenar os ficheiros gerados pelo ***build*** da aplicação em produção.
4. *components/*
   - Outro diretório de componentes reutilizáveis, que contém elementos do *ShadCN/UI* usados em múltiplas partes da aplicação.
5. *lib/*
   - Contém utilitários e funções auxiliares que podem ser usadas em várias partes da aplicação.
6. *node_modules/*
   - Diretório gerado automaticamente pelo ***npm*** ou ***yarn*** para armazenar todas as dependências do projeto.
7. *public/*
   - Diretório de **arquivos estáticos** utilizados na aplicação, incluindo imagens, ícones e outros recursos acessíveis publicamente.

Ficheiros importantes

1. Ficheiros de Configuração
   - *.gitignore*: Lista os ficheiros e diretórios que devem ser ignorados pelo *Git*.
   - *config.js*: Contém as configurações globais da aplicação.
   - *eslint.config.mjs*: Configuração do *ESLint*, responsável pela padronização e qualidade do código.
   - *next-env.d.ts*: Definições de tipos para o *TypeScript* no ***Next.js***.

- *next.config.ts*: Configuração avançada do **Next.js**, incluindo otimizações e ajustes personalizados.
- *postcss.config.mjs:* Configuração do ***PostCSS***, usado para processamento avançado de CSS.
- *tailwind.config.ts*: Configuração do **Tailwind CSS**, especificando estilos personalizados e temas globais.
- *tsconfig.json*: Configuração do ***TypeScript***, determinando regras e definições do compilador.

2. Ficheiros de Dependências
   - *package.json:* Contém informações sobre o projeto, incluindo dependências e scripts disponíveis.
   - *package-lock.json*: Bloqueia as versões exatas das dependências para garantir consistência no ambiente de desenvolvimento.

Diretório de Componentes

Os componentes reutilizáveis estão organizados no diretório *app/components/,* garantindo modularidade e reutilização na interface. Os principais componentes incluem:

- *AboutModal.tsx*: Modal com informações sobre a aplicação.
- *CodeBlock.tsx*: Exibe o código gerado com destaque de sintaxe e funcionalidade de cópia.
- *ExplanationModal.tsx*: Exibe explicações detalhadas sobre a geração do código.
- *History.tsx*: Interface para exibir o histórico de interações do utilizador.
- *ImagePreview.tsx*: Componente responsável pela pré-visualização das imagens carregadas pelo utilizador.
- *Login.tsx*: Interface de autenticação e login do utilizador.
- *Navbar.tsx:* Barra de navegação que facilita a interação com diferentes partes da aplicação.

A estrutura do projeto foi organizada seguindo as boas práticas do desenvolvimento em *Next.js* e *React*, garantindo modularidade, escalabilidade e manutenibilidade. A separação clara entre componentes, configuração e recursos estáticos permite uma implementação eficiente e facilita futuras expansões e melhorias no projeto.

4.6.6. INTEGRAÇÃO E TESTES

A integração e testes representam uma das fases essenciais no desenvolvimento de software, garantindo que os diferentes componentes do sistema funcionem corretamente e de forma integrada. Nesta secção será demonstrado esta fase e, após a construção individual do front-end e do back-end, foi realizada a conexão entre essas camadas, seguida de uma série de testes para validar a funcionalidade, desempenho e usabilidade da aplicação. Esta etapa é crucial para assegurar que o sistema atenda aos requisitos previamente definidos, eliminando erros, otimizando a experiência do utilizador e garantindo a estabilidade da solução desenvolvida.

O Papel da Inteligência Artificial no Desenvolvimento e Testes

Uma das principais inovações desta fase foi o uso do *GitHub Copilot* [93], uma ferramenta de inteligência artificial assistida disponível no *Visual Studio Code* (VSCode) [100], que foi disponibilizada para uso gratuito com limitação de mensagens diárias. O *GitHub Copilot* demonstrou-se uma ferramenta valiosa e transformadora, auxiliando na escrita de código, correção de bugs e criação de testes automatizados assim como os autores em [82] descrevem.

Ao longo do desenvolvimento, o *GitHub Copilot* foi utilizado para sugerir implementações eficientes, identificar erros comuns de sintaxe e lógica e até mesmo gerar casos de testes de forma automatizada. Esta abordagem resultou numa entrega mais produtiva e ágil, permitindo reduzir significativamente o tempo necessário para depuração e testes manuais.

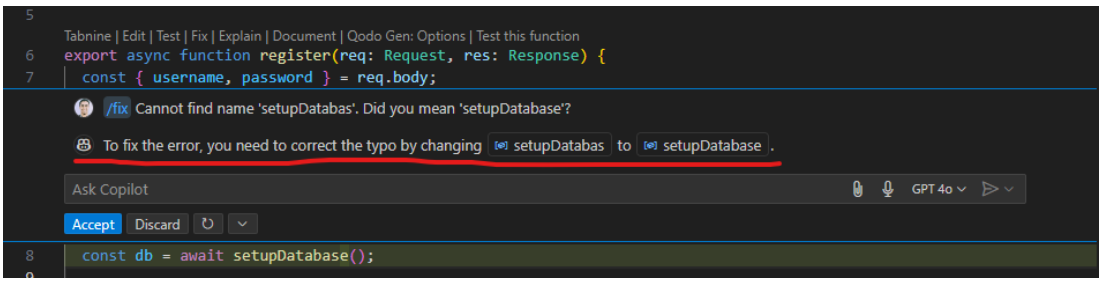

Figura 34 - Uso Github Copilot

Na Figura 34 - Uso Github Copilot mostra um exemplo de como esta tecnologia melhora a experiência de desenvolvimento, reduzindo erros e aumentando a produtividade. No caso específico da imagem, o *GitHub Copilot* identificou um erro simples de digitação no código, onde a função *setupDatabas* foi escrita incorretamente, e sugeriu a correção para *setupDatabase*. Esta funcionalidade é extremamente útil, pois permite ao programador corrigir rapidamente pequenos erros que poderiam causar falhas na execução do código e por diversas vezes demora a encontrar.

O *Github Copilot* vai além de correções simples, possui inclusive atalhos para determinados comandos, como correção de código, auxílio na criação de testes, explicação e geração de blocos de cógidos. Na Figura 35 - Github Copilot Fix é possível ver o trecho inicial da proposta de correção de um bloco de código, com o uso do atalho */fix*:

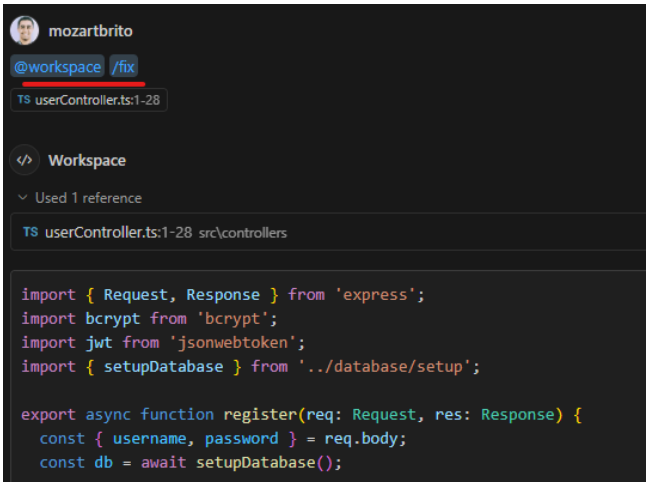

Figura 35 - Github Copilot Fix

A Figura 36 - Github Copilot Geração de Código por sua vez demonstra a proposta de geração de um bloco de códigos (carácteres cinza) para um método assíncrono de acesso ao perfil do usuário. Este método não foi implementado por não ser necessário neste escopo, entretanto, baseado no contexto do código anterior, é possível perceber que o código seria suficiente para esta implementação:

Figura 36 - Github Copilot Geração de Código

A utilização do *GitHub Copilot* não apenas acelera a fase de testes, mas também reforça um dos pilares fundamentais desta tese: a IA desempenha um papel crucial no desenvolvimento de software moderno e tem potencial para otimizar processos, reduzir erros e aumentar a eficiência dos programadores, tornando-os mais ágeis e menos propensos a erros. A capacidade de detectar e corrigir problemas em tempo real demonstra como a IA desempenhará cada vez mais um papel importante no futuro da engenharia de software.

Estratégias de Testes Aplicadas

Com a integração entre o front-end e o back-end, foram conduzidos vários tipos de testes para garantir a robustez do sistema. O primeiro passo foi solicitar ajuda ao Github Copilot de maneira a optimizar o desenvolvimento.

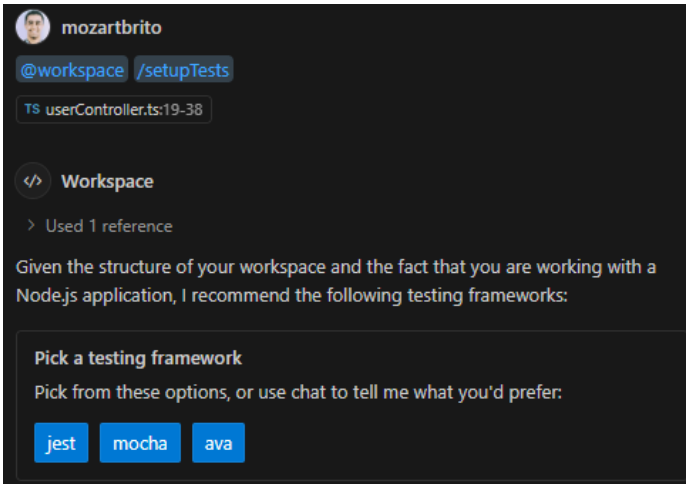

Figura 37 - Github Copilot Configuração de Testes

Na Figura 37 - Github Copilot Configuração de Testes é possível observar que o GitHub Copilot sugere a instalação de frameworks para a realização de testes automatizados. Neste contexto, foi escolhido o *Jest* [101], um framework de testes amplamente utilizado na comunidade *JavaScript* e *TypeScript*. O *Jest* destaca-se por sua facilidade de configuração, suporte nativo a testes assíncronos e eficiente para testes unitários, de integração e *end-to-end* em aplicações web e back-end [101].

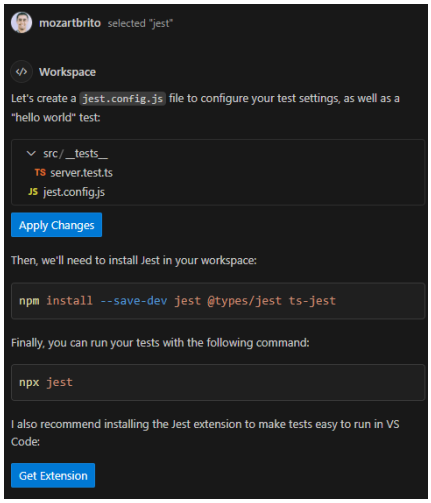

Figura 38 - Instalação do Jest

A Figura 38 - Instalação do Jest mostra o seguimento da configuração do *Jest*, bem como a sua inicialização com um teste inicial presente no diretório

src/__tests__/server.test.ts. Para garantir o correto mapeamento dos testes pelo Jest, todos os ficheiros de teste neste diretório devem conter *.test.* no seu nome.

A estruturação de casos de teste no Jest segue um modelo claro e intuitivo. O método principal, *describe*, é utilizado para agrupar e organizar os testes em blocos lógicos, permitindo a execução de chamadas assíncronas a outros métodos. Dentro desse bloco, utilizam-se métodos como *it* (ou o seu alias *test*), que representam os casos de teste individuais. Cada caso de teste é avaliado através do método *expect*, que verifica se a saída do código testado corresponde ao resultado esperado, retornando verdadeiro ou falso conforme a validação realizada.

Para simplificar, ao digitar "*it('caso de teste específico', async () => {"* no VS Code, o *Github Copilot* pode sugerir automaticamente um bloco de teste com base no texto adicionado.

1. **Testes de Integração:** Garantem e validam a comunicação entre o front-end e o back-end, garantindo a transmissão correta de dados e o processamento adequado pela API de LLM.
   Foram desenvolvidos testes para os principais componentes com o auxílio do *Github Copilot* e abaixo é possível verificar dois casos de testes de integração com componentes do back-end.
   a. **Caso de teste 1 - Registro de Utilizador:** Este teste verifica se um novo utilizador pode ser registado corretamente na aplicação.

```
describe('Testes de Integração - Registo de Utilizador', () => {
//Caso de teste para registo de um novo utilizador
  it('Deve registar um novo utilizador com sucesso', async () => {
    const response = await request(app)
      .post('/api/users/register')
      .send({
        username: 'utilizadorTeste',
        password: 'senhaSegura123'
      });

    expect(response.status).toBe(201);
    //Validação de sucesso no registo
    expect(response.body).toHaveProperty('message', User created successfull ');
  });
});
```

   b. **Caso de teste 2 - Autenticação de Utilizador:** Este teste verifica se um utilizador pode fazer login corretamente e receber um *token* de autenticação e como segundo exemplo, uma falha por uso de senha errada.

```
describe('Testes de Integração - Login de Utilizador', () => {
 //Caso de teste para validar um login de sucesso
  it('Deve autenticar um utilizador existente e retornar um token', async () => {
    const response = await request(app)
      .post('/api/users/login')
      .send({
        username: 'utilizadorTeste',
        password: 'senhaSegura123'
      });
    expect(response.status).toBe(200);
   //Validação de sucesso  ao receber o token
    expect(response.body).toHaveProperty('token');
  });

  it('Deve retornar erro para credenciais inválidas', async () => {
    //Caso de teste para validar um login de com credenciais inválidas
    const response = await request(app)
      .post('/api/users/login')
      .send({
        username: 'utilizadorInexistente',
        password: 'senhaErrada'
      });

    expect(response.status).toBe(401);
   //Validação de erro em tentativa de autenticação com dados inválidos
    expect(response.body).toHaveProperty('error', ' Invalid credentials');
  });
});
```

2. **Testes de Usabilidade:** Estes testes por sua vez possui a finalidade de verificar a experiência do utilizador com casos de testes criados no projecto front-end, verificando a navegabilidade da interface, a disposição dos elementos e a exibição correta dos resultados.

   Para os exemplos que seguem, foram utilizados outros componentes do *Jest* que viabiliza a simulação de um navegador através de ações e resultados, o que torna possível a verificação de existência de conteúdos específicos após renderização do sistema. O *Jest* permite ainda a criação *mocks* de base de dados ou APIs, o que permite a separação de responsabilidades e interoperabilidade entre os componentes.

   Caso de teste de exemplo: **Verificar a Navegabilidade da Interface**

   Objetivo: Garantir que o utilizador consiga realizar a autenticação no sistema.

Cenário:

1. O utilizador preenche o campo de texto para utilizador e senha.
2. O utilizador clica no botão "Login".
3. O sistema processa os dados e gera e autentica o utilizador.
4. O resultado esperado é a página inicial do sistema.

Critério de Aceitação:

- O utilizador consegue realizar a autenticação.
- O *sistema deve ser corretamente carregado*.

Implementação do Teste

```
//Criação dos mocks
fetchMock.enableMocks();
const mockOnLogin = jest.fn();
beforeEach(() => {
 jest.clearAllMocks();
 fetchMock.resetMocks();
});
describe("Login Component", () => {
 test("renders Login component", () => {
  //Validação da renderização do ecrã de Login:
   render(<Login onLogin={mockOnLogin} />);
   expect(screen.getByLabelText("Username")).toBeInTheDocument();
   expect(screen.getByLabelText("Password")).toBeInTheDocument();
   expect(screen.getByRole("button", { name: /login/i })).toBeInTheDocument();
 });
 test("handles form submission with valid credentials", async () => {
 //Validação da submissão de dados de login com credenciais válidas:

   fetchMock.mockResponseOnce(JSON.stringify({ username: "testuser", token:
"testtoken" }));
   render(<Login onLogin={mockOnLogin} />);
   fireEvent.change(screen.getByLabelText("Username"), { target: { value:
"testuser" } });
   fireEvent.change(screen.getByLabelText("Password"), { target: { value:
"password" } });
   fireEvent.click(screen.getByRole("button", { name: /login/i }));
  //Em caso de sucesso, verifica os dados do token:
   await waitFor(() => expect(mockOnLogin).toHaveBeenCalledWith("testuser",
"testtoken"));
 });
```

```
  test("handles form submission with invalid credentials", async () => {
//Caso de teste da submissão de dados de login com credenciais inválidas
    fetchMock.mockResponseOnce(JSON.stringify({ error: "Invalid credentials" }), {
status: 401 });
    render(<Login onLogin={mockOnLogin} />);
    fireEvent.change(screen.getByLabelText("Username"), { target: { value:
"wronguser" } });
    fireEvent.change(screen.getByLabelText("Password"), { target: { value:
"wrongpassword" } });
    fireEvent.click(screen.getByRole("button", { name: /login/i }));
   //Validação do erro de submissão do Login
    await waitFor(() => expect(screen.getByText("Invalid
credentials")).toBeInTheDocument());
  });
});
```

3. **Testes de Desempenho:** O objetivo dos testes de desempenho é garantir que o sistema responde adequadamente sob diferentes condições de carga e tempo de resposta. O *Jest*, embora seja amplamente utilizado para testes unitários e de integração, também pode ser aplicado para medir tempos de execução de funções críticas no back-end.

   Caso de teste de exemplo: **Verificar Tempo de Resposta da API**

   Objetivo: Garantir que a API de geração de código responde dentro do tempo limite estabelecido (por exemplo, menos de 2 segundos).

   Cenário:

   1. O utilizador envia uma requisição POST para o endpoint */api/code-generation/generate.*
   2. A API processa a requisição e interage com o modelo de IA da OpenAI.
   3. O tempo de resposta da API é capturado e avaliado.

   Critério de Aceitação:

   - O tempo de resposta deve ser inferior a *2000ms*.
   - A resposta deve conter um código gerado válido.

   Implementação do Teste

```
fetch import request from 'supertest';
import app from '../server';

describe('Testes de Desempenho', () => {
```

```
  it('Deve responder à requisição de geração de código em menos de 2000ms',
async () => {
   const start = Date.now();
   const response = await request(app)
    .post('/api/code-generation/generate')
    .send({ prompt: 'Cria uma função em JavaScript' });

   const duration = Date.now() - start;

   expect(response.statusCode).toBe(200);
   expect(response.body.code).toBeDefined();
  // Tempo de resposta deve ser menor que 2 segundos
   expect(duration).toBeLessThan(2000);
  });
});
```

4. **Gestão de Erros:** Implementação de tratamento de exceções, garantindo que mensagens de erro adequadas sejam exibidas em casos como falha de conexão com a API ou entradas inválidas.

   Caso de teste de exemplo: **Entrada Inválida na Requisição**

   Objetivo: Garantir que o sistema rejeita entradas inválidas e fornece mensagens de erro adequadas.

   Cenário:

   1. O utilizador envia uma requisição *POST* para o endpoint */api/code-generation/generate* sem fornecer um prompt válido.
   2. O servidor valida a entrada e identifica que o campo está vazio ou não segue o formato esperado.
   3. O sistema deve rejeitar a solicitação e exibir uma mensagem de erro apropriada.

   Critério de Aceitação:

   - O sistema deve validar a entrada antes de processar a requisição.
   - O código de resposta da API deve ser 400 (Bad Request).
   - A resposta JSON deve informar que o prompt é obrigatório.

   Implementação do Teste

```
describe('Gestão de Erros - Entrada Inválida', () => {
  it('Deve retornar erro 400 quando o prompt não é fornecido', async () => {
   const response = await request(app)
    .post('/api/code-generation/generate')
```

```
      .send({}); // Enviando um payload vazio

    expect(response.statusCode).toBe(400);
   //Valida a mensagem de erro
    expect(response.body).toEqual({
     error: 'O campo prompt é obrigatório.',
    });
  });
});
```

Esta etapa foi relevante por permitir a construção de validação contínua da aplicação. Com a ajuda do GitHub Copilot, foram gerados testes unitários e testes de integração utilizando frameworks apropriados. A IA contribuiu significativamente ao:

- Sugerir cenários de teste abrangentes, cobrindo diferentes entradas e respostas esperadas.
- Automatizar a criação de casos de teste, reduzindo a necessidade de escrita manual e acelerando o processo.
- Identificar potenciais erros lógicos e sugerir correções, melhorando a confiabilidade do sistema.

4.6.7. Boas Práticas e Considerações

A adoção de boas práticas durante o desenvolvimento do projeto foi importante para garantir a qualidade, a manutenibilidade e a possível escalabilidade do sistema. Estas práticas asseguram que o código seja compreensível, bem estruturado e preparado para futuras expansões, o que reduz a complexidade e aumenta a eficiência no desenvolvimento e manutenção. Embora não tenham sido detalhadas todas as boas práticas utilizadas, seguem pontos de importância durante o desenvolvimento do software:

Modularidade e Organização do Código

O projeto foi desenvolvido com uma abordagem modular, promovendo a separação lógica entre os diferentes componentes do sistema. A estrutura do código foi organizada em camadas distintas, garantindo que cada parte da aplicação tenha uma responsabilidade bem definida:

- **Front-end**: Responsável pela interface do utilizador, construído com React e Next.js, garantindo uma experiência fluida e responsiva.

- **Back-end**: Implementado em Node.js, estruturado de forma a gerenciar a lógica da aplicação e a comunicação com a API da OpenAI.
- **Serviços Externos**: Integração com modelos de IA via API, garantindo a flexibilidade para futuras melhorias ou mudanças de provedores.

O projecto encontra-se disponível publicamente em dois repositórios distintos na plataforma GitHub, podendo ser acedido para fins de consulta, verificação do código-fonte ou estudo. Esta disponibilização visa promover a transparência, o reaproveitamento de soluções e a partilha de conhecimento com a comunidade académica e profissional[23].

A modularidade adotada facilita a reutilização de código, permitindo que diferentes partes do sistema possam ser aprimoradas independentemente, sem comprometer a integridade geral da aplicação.

Versionamento de Código

Para garantir rastreabilidade e controle das modificações, foi utilizado *Git* para versionamento de código e *GitHub* como repositório remoto [102]. Esta abordagem possibilitou:

- **Histórico de Alterações**: Permite reverter mudanças ou identificar onde um erro foi introduzido.
- **Colaboração Eficiente**: Embora o projecto não tenha sido desenvolvido por uma equipa, o versionamento viabiliza o desenvolvimento simultâneo de diferentes funcionalidades por vários programadores.
- **Branches e Pull Requests**: Implementação de fluxos de desenvolvimento como *Git Flow*, permitindo a integração e testes antes da fusão de novas funcionalidades.

Documentação e Comentários no Código

A documentação adequada do sistema foi priorizada para garantir que a aplicação possa ser compreendida e mantida a longo prazo. Para isso, foram implementadas as seguintes práticas:

- **Documentação Técnica**: Explicação detalhada da arquitectura do sistema, API *endpoints*, modelos de dados e fluxo de execução.

[2] https://github.com/mozartbrito/project-generator-fe.git
[3] https://github.com/mozartbrito/project-generator-be.git

- **Comentários no Código**: Uso de comentários explicativos dentro do código para facilitar a leitura e compreensão das implementações.
- **Diagramas de Arquitetura**: Representações visuais do sistema, como Diagramas de Fluxo de Dados (DFD) e Diagramas de Entidade-Relacionamento (DER), que possibilita a ilustração a interação entre os componentes.

A documentação clara e organizada reduz a curva de aprendizado para novos programadores e facilita futuras manutenções e expansões.

Escalabilidade e Manutenibilidade

O sistema foi projetado para ser escalável, permitindo a incorporação de novas funcionalidades sem comprometer o desempenho. Algumas das decisões tomadas para garantir essa escalabilidade incluem:

- **Arquitetura baseada em APIs**: Possibilita a substituição ou integração de novos modelos de IA sem grandes mudanças no código.
- **Banco de Dados Flexível**: Utilização do *SQLite* para armazenamento inicial, mas estruturado de forma a permitir a migração para bases mais robustas se necessário.
- **Código Extensível**: Estruturas de código que permitem a adição de novos recursos sem impactar negativamente funcionalidades já existentes.

Além disso, a implementação de testes automatizados, descrita na secção Integração e Testes, contribuiu para garantir que o sistema continue funcional mesmo após futuras melhorias.

4.7. CONCLUSÃO

O processo de especificação e desenvolvimento do sistema seguiu uma abordagem estruturada, combinando boas práticas de engenharia de software com ferramentas modernas que garantiram um fluxo de trabalho eficiente e um código robusto. A separação modular entre front-end, back-end e serviços externos permitiu a organização clara das responsabilidades do sistema, favorecendo a escalabilidade e a manutenção futura. Além disso, a adoção de metodologias como o versionamento de código com *Git*, a implementação de testes automatizados e a utilização de um banco de dados flexível reforçaram a confiabilidade da solução proposta.

Um dos diferenciais deste projeto foi a incorporação da IA no processo de desenvolvimento, especificamente com o *GitHub Copilot* [93]. Esta ferramenta

demonstrou-se um aliado valioso na escrita de código, na otimização de trechos complexos e na geração automatizada de testes. O auxílio da IA não apenas acelerou o processo de desenvolvimento, mas também reduziu a ocorrência de erros comuns, permitindo um desenvolvimento mais produtivo e eficiente. Além disso, a integração com modelos de IA como o *GPT-4o-mini* da OpenAI [98] viabilizou uma interface dinâmica e responsiva para a geração de código a partir de descrições textuais e protótipos visuais.

A aplicação das boas práticas de desenvolvimento, aliadas ao suporte oferecido pela IA, resultou num sistema estável, escalável e de fácil manutenção. A documentação clara, a modularidade da arquitetura e a preocupação com a experiência do utilizador reforçam a qualidade e o impacto desta solução. Com essa base sólida, o projeto encontra-se preparado para futuras extensões e aprimoramentos, incluindo a integração com novos modelos de IA e a expansão das funcionalidades para atender a necessidades mais amplas.

5. Análise de Resultados

A presente secção tem como objetivo analisar os resultados obtidos com o desenvolvimento e implementação do sistema proposto, avaliando a integração entre inteligência artificial e desenvolvimento de software. Esta análise considera os benefícios e desafios identificados ao longo do processo, validando a viabilidade da solução desenvolvida e sua contribuição para a área.

5.1. Revisão dos Conceitos Fundamentais

O trabalho desenvolvido partiu de conceitos essenciais da engenharia de software, abordando metodologias e práticas de desenvolvimento que garantem a criação de sistemas bem construídos e escaláveis. A estruturação do sistema foi baseada em boas práticas de desenvolvimento, o que inclui arquitetura modular, versionamento de código e testes automatizados. Além disso, foram aplicadas metodologias ágeis, que permitiram o refinamento contínuo da solução.

Em paralelo, explorou-se a crescente influência da IA no desenvolvimento de software, em especial os LLMs, como o *GPT-4o-mini* da OpenAI. Estes modelos destacam-se pela capacidade de compreender e gerar código a partir de descrições textuais, otimizando significativamente a produtividade dos programadores.

A combinação destas áreas permitiu a construção de um sistema que automatiza a geração de código a partir de descrições textuais e protótipos visuais. Esta abordagem não apenas reduz o tempo de desenvolvimento, mas também minimiza erros e melhora a qualidade do código gerado.

5.2. Demonstração de Utilização do Sistema

Esta secção demonstra o passo a passo da utilização do sistema desenvolvido onde é possível realizar a validação de sua funcionalidade.

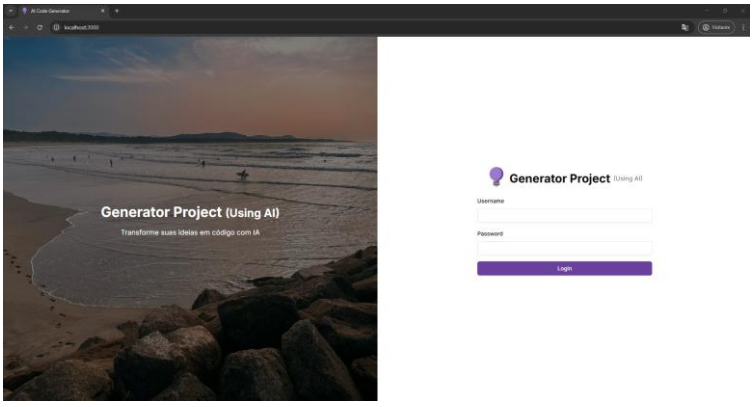

Figura 39 – Login

Na Figura 40 - Página inicial coloca-se os dados do utilizador para poder entrar no sistema.

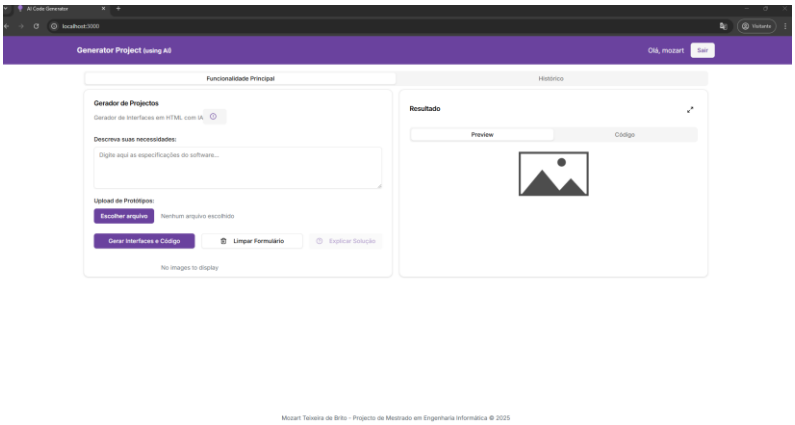

Figura 40 - Página inicial

Após validação do utilizador, sua entrada é permitida e é possível acessar o sistema conforme apresentado na Figura 41 - Botão de informações de funcionamento.

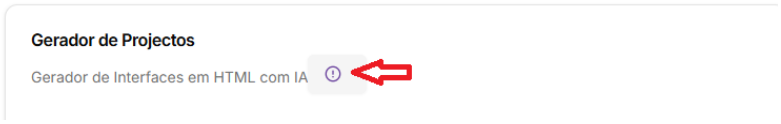

Figura 41 - Botão de informações de funcionamento

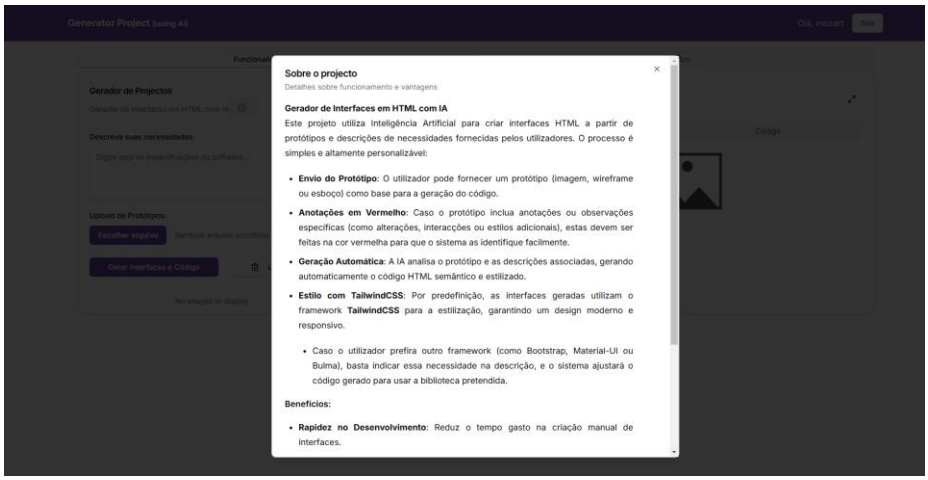

Figura 42 - Informações de funcionamento

O sistema possui uma funcionalidade relativa à explicação de seu funcionamento e a Figura 41 - Botão de informações de funcionamento mostra sua posição. A Figura 42 - Informações de funcionamento mostra a página aberta com as informações sobre como os protótipos podem estar organizados, quais tecnologias serão utilizadas no resultado e outros detalhes.

A principal funcionalidade é focada no envio de protótipos visuais ou *prompts* textuais e até mesmo a combinação de ambos. O sistema então irá proceder com a

geração de código funcional e uma pré-visualização do resultado. A Figura 43 - Protótipo mostra um protótipo de baixa fidelidade com software de uma calculadora e será enviado no sistema sem adicionar *prompt* textual confome a Figura 44 - Envio de protótipo sem prompt mostra.

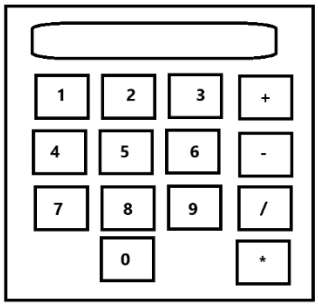

Figura 43 - Protótipo

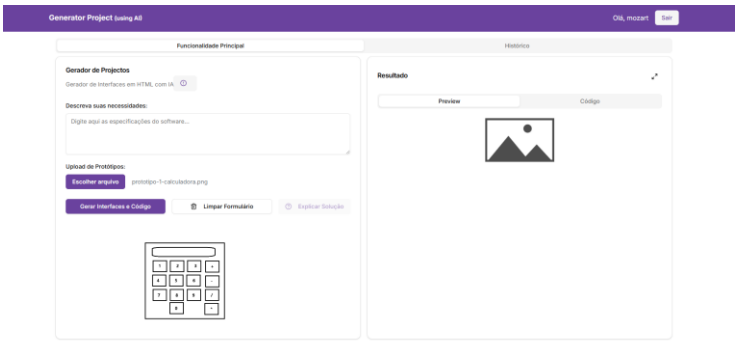

Figura 44 - Envio de protótipo sem prompt

O envio do protótipo desta maneira tem o propósito de analisar o comportamento da API utilizada. É possível ver o resultado na Figura 45 - Resultado protótipo sem prompt:

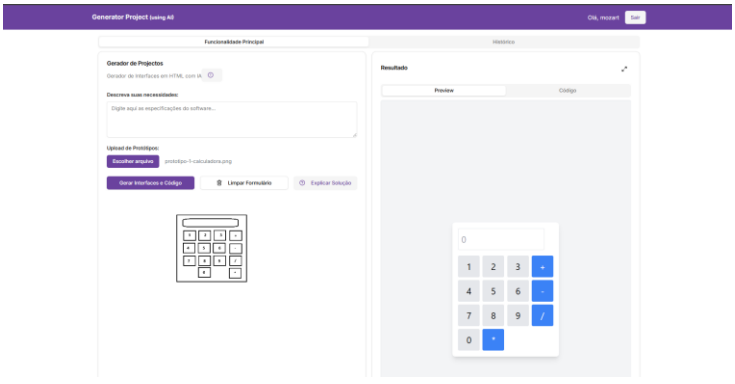

Figura 45 - Resultado protótipo sem prompt

Além do resultado visual é possível ver o código conforme a Figura 46 - Código gerado.

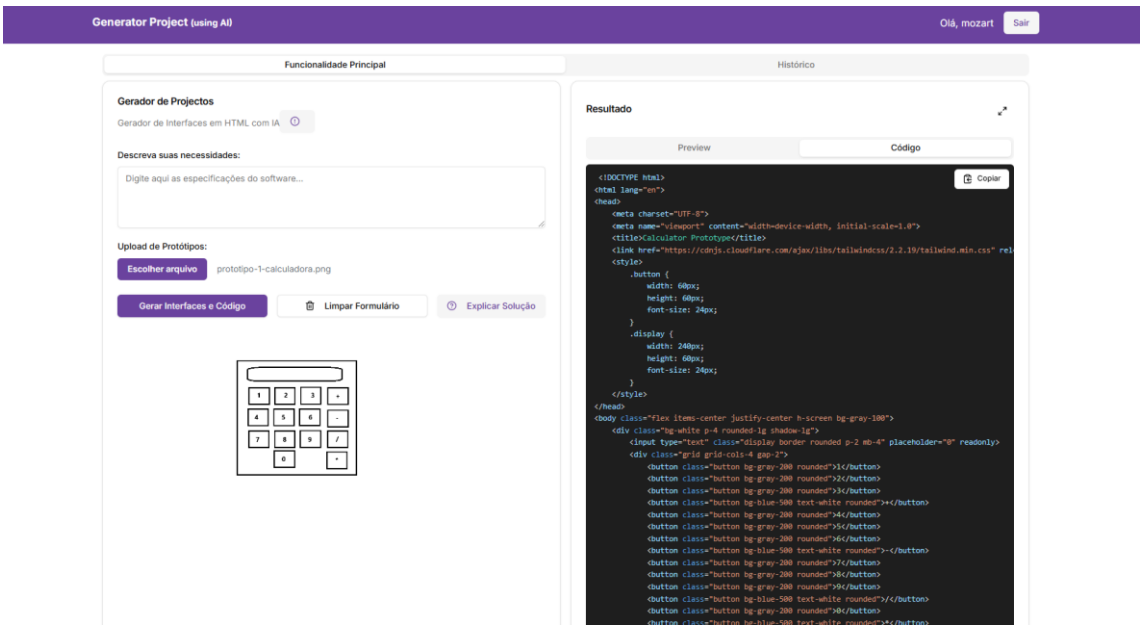

Figura 46 - Código gerado

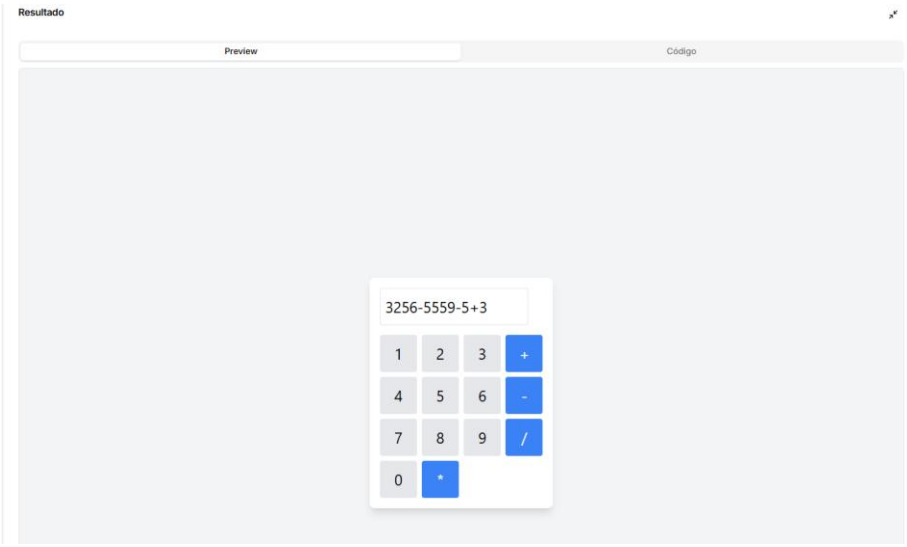

Figura 47 - Teste de funcionalidade

A Figura 47 - Teste de funcionalidade é exibida numa visualização em modo inteiro para facilitar os testes do resultado. Embora não tenha sido utilizado nenhum outro *prompt*, o resultado obtido surpreende pela capacidade da API entender e captar as necessidades apenas ao analisar o protótipo. Entretanto o resultado ainda não possui um comportamento útil.

O sistema permite o reuso do protótipo e *prompts* anteriores, o que é importante para pois permite incremento nas funcionalidades. Para melhorar o resultado, foi inserido um prompt com o seguinte conteúdo:

"Adicione um botão abaixo com o texto "Calcular". O utilizador deverá clicar em um número e este aparece no input, entretanto, ao clicar em "+" ou "-" ou "/" ou "", o input deve*

ser limpo para o utilizador digitar outro número, tudo salvo em memória. Ao clicar no botão calcular, o input deverá ser atualizado com o resultado calculado dos valores digitados."

O envio é realizado conforme a Figura 48 - Inserção de prompt detalhado e o resultado está melhorado como pode-se ver na Figura 49 - Resultado baseado no protótipo e prompt.

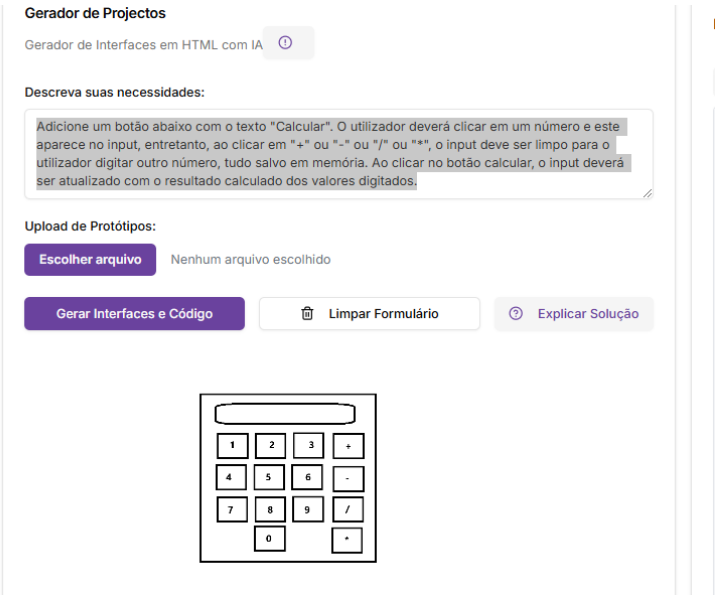

Figura 48 - Inserção de prompt detalhado

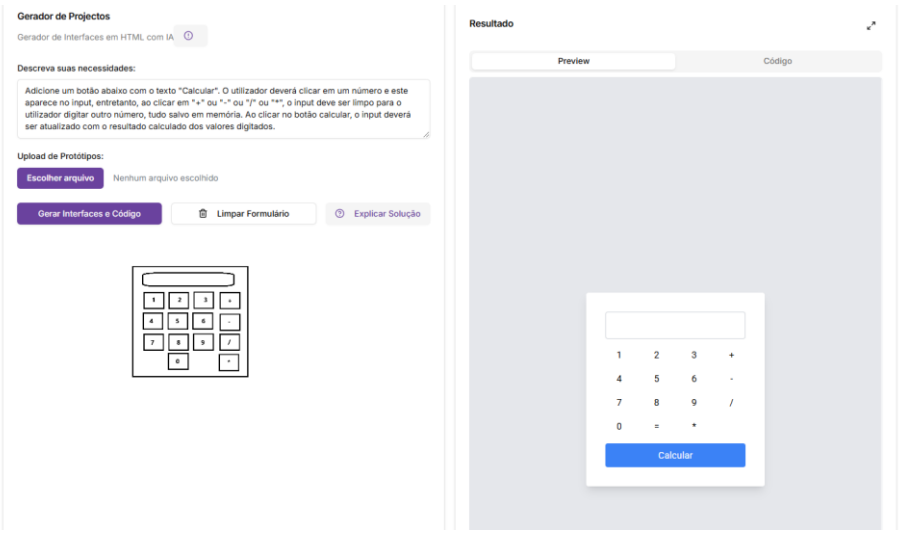

Figura 49 - Resultado baseado no protótipo e prompt

Neste momento a calculadora já está funcional devido a presença de funções no código conforme a Figura 50 - Código gerado com as funções de cálculo.

```
        });

        calcBtn.addEventListener('click', () => {
            if (previousInput && currentInput && operator) {
                let result;
                const prev = parseFloat(previousInput);
                const current = parseFloat(currentInput);

                switch (operator) {
                    case '+':
                        result = prev + current;
                        break;
                    case '-':
                        result = prev - current;
                        break;
                    case '*':
                        result = prev * current;
                        break;
                    case '/':
                        result = prev / current;
                        break;
                }

                display.value = result;
                currentInput = result.toString();
                previousInput = '';
                operator = '';
            }
        });
    </script>
</body>
</html>
```

Figura 50 - Código gerado com as funções de cálculo

Os *prompts* textuais são muitos úteis pois permite detalhar as funcionalidades conforme as necessidades. Por outro lado, diversas vezes será necessário uma explicação visual sobre algum comportamento específico, posicionamento de elementos, entre outros. Foi adicionado nos *prompts* internos do sistema algumas regras relacionadas à interpretação dos protótipos, uma delas possui a orientação para ignorar informações textuais em cor vermelha e considerá-las como sugestões, ou *prompts* complementares.

Ao considerar isto, foi feito modificação no protótipo para incluir algumas informações para fins de teste e verificar se o sistema o adiciona no resultado ou apenas utiliza como complementos. O novo protótipo pode ser visto na Figura 51 - Adição de prompt no protótipo e seu envio ocorre como é exibido na Figura 52 - Envio do protótipo com prompt e prompt textual.

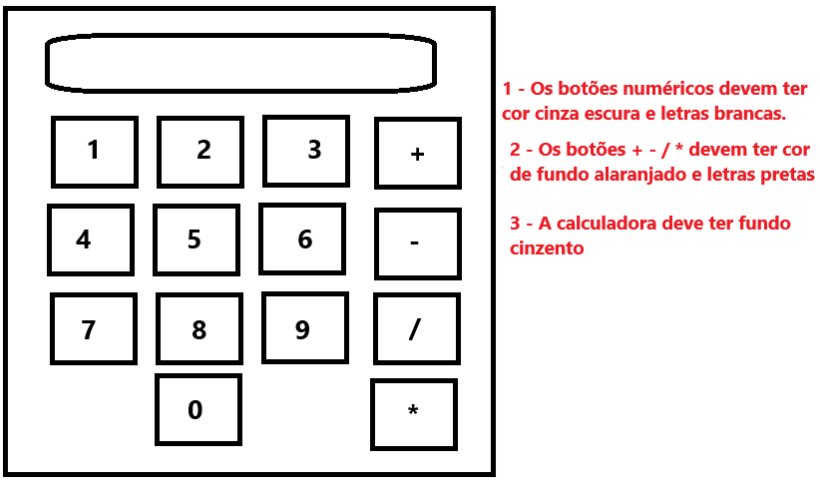

Figura 51 - Adição de prompt no protótipo

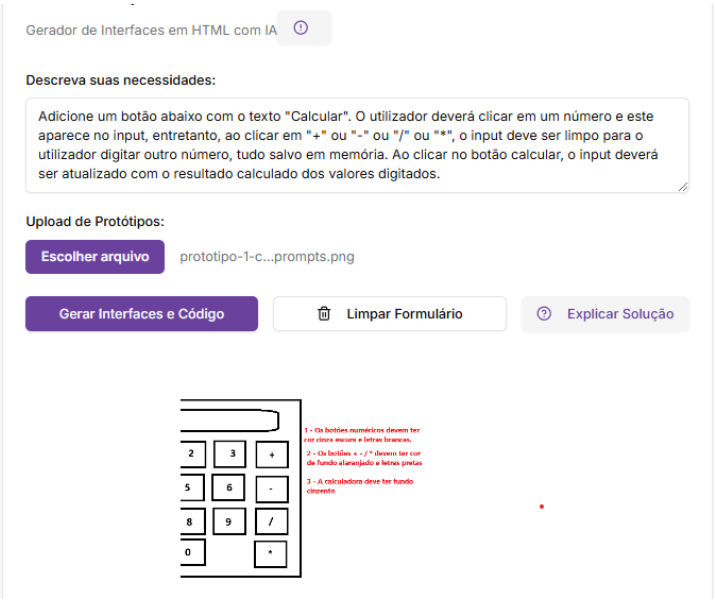

Figura 52 - Envio do protótipo com prompt e prompt textual

Como exibido na Figura 53 - Resultado com prompt no protótipo, é possível verificar que o sistema ignorou os textos em vermelho e os interpretou como *prompts* complementares.

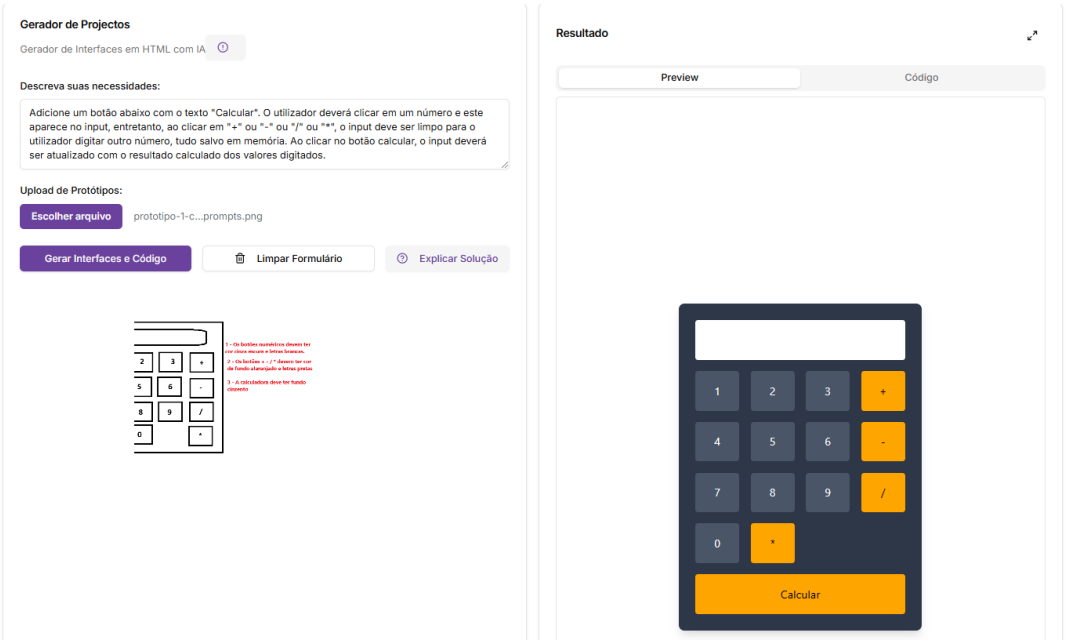

Figura 53 - Resultado com prompt no protótipo

Na Figura 54 - Teste de funcionamento do resultado é possível verificar o funcionamento final da calculadora. O cálculo de multiplicação funciona como esperado.

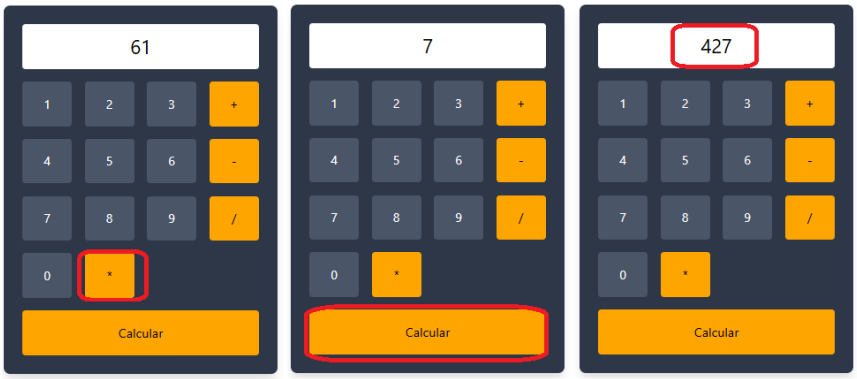

Figura 54 - Teste de funcionamento do resultado

Conforme analisado, a melhoria contínua do resultado depende de como o utilizador usa os recursos disponíveis. Com mais detalhes, melhores resultados.

Uso de protótipo de baixa fidelidade

Foi realizado um segundo teste de funcionamento, desta vez se utilizou de um protótipo de baixa fidelidade feito à mão conforme a Figura 55 - Protótipo desenhado à mão. O *prompt* por sua vez foi como se segue pode ser visto na Figura 56 - Adição de prompt para envio do protótipo:

"Crie uma página de criação de postagens para blogs. Necessito que contenha um navbar para espaço para uma logo a esquerda. Os campos devem ser como os do protótipo."

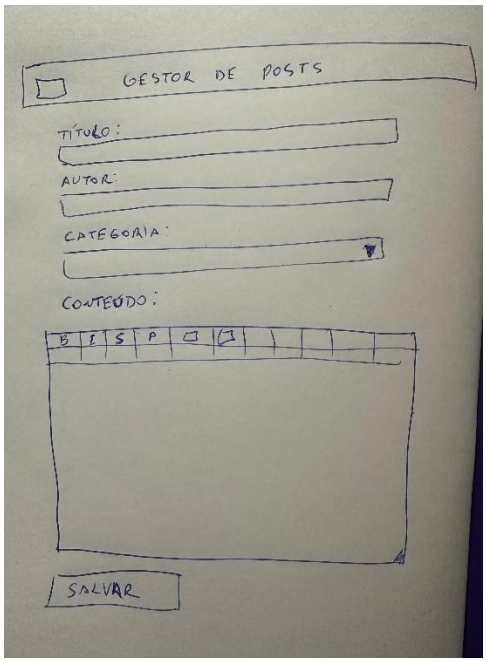

Figura 55 - Protótipo desenhado à mão

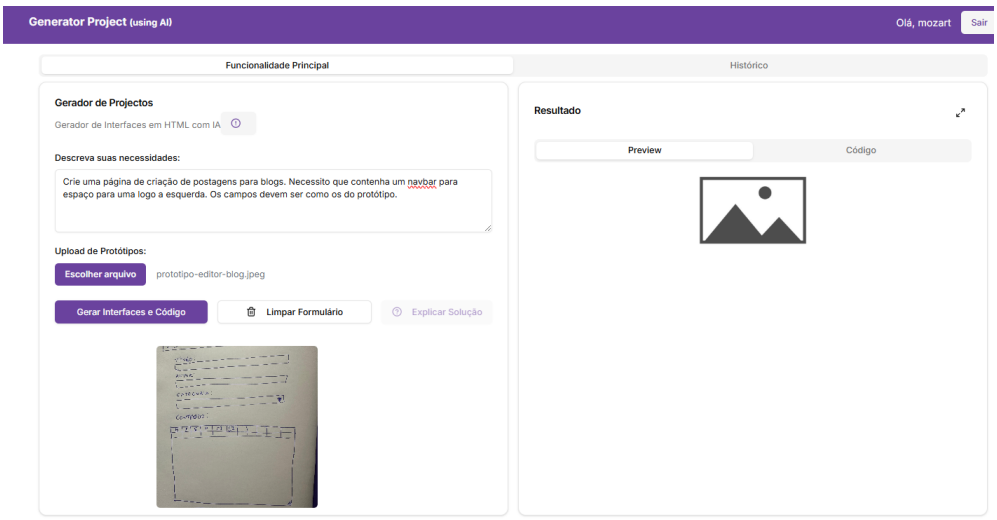

Figura 56 - Adição de prompt para envio do protótipo

A Figura 57 - Resultado protótipo manual mostra o resultado do protótipo combinado com o *prompt* mencionado e mais uma vez a aplicação mostrou-se funcional.

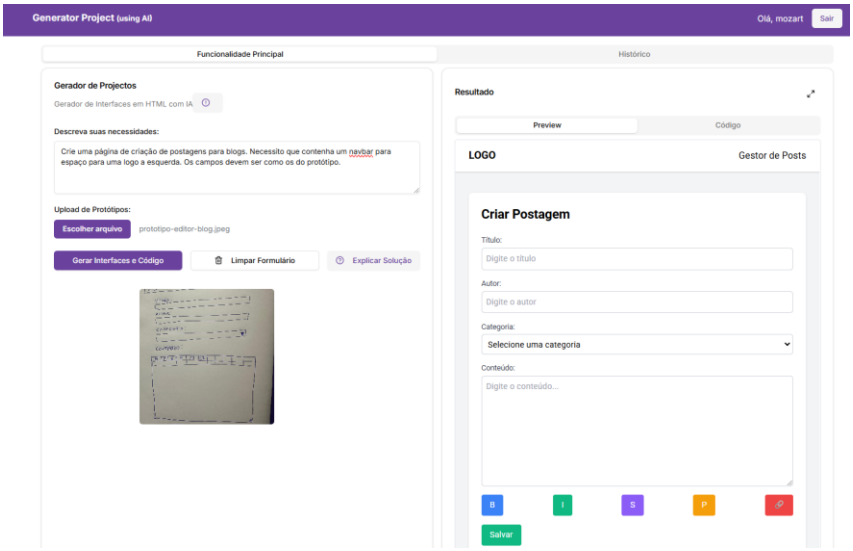

Figura 57 - Resultado protótipo manual

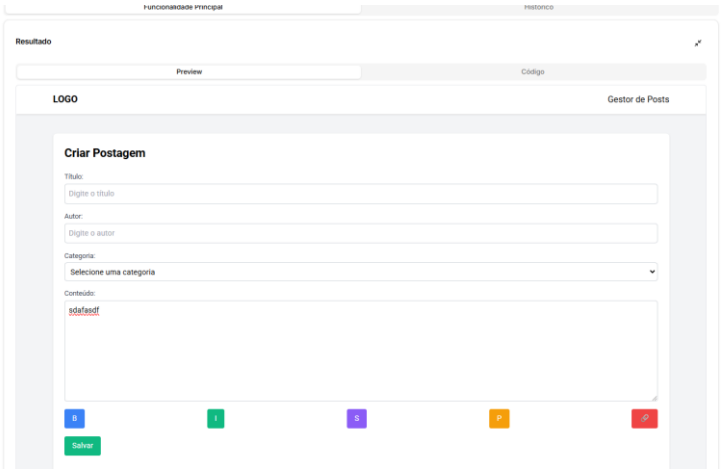

Figura 58 - Visualização em detalhe

A Figura 58 - Visualização em detalhe mostra o resultado em ecrã inteiro para melhor visualização. Por mais que o resultado tenha sido promissor, é possível verificar alguns problemas no campo de conteúdo. Foram gerados botões sem funcionalidade e era esperado que fosse exibido um editor de texto. Será enviado uma melhoria no *prompt*:

"Crie uma página de criação de postagens para blogs. Necessito que contenha um navbar para espaço para uma logo a esquerda. Os campos devem ser como os do protótipo. Os botões do protótipo (B, I, S, P,...) são apenas exemplos que o campo conteúdo tenha que possuir um editor de texto HTML. Gere outra vez"

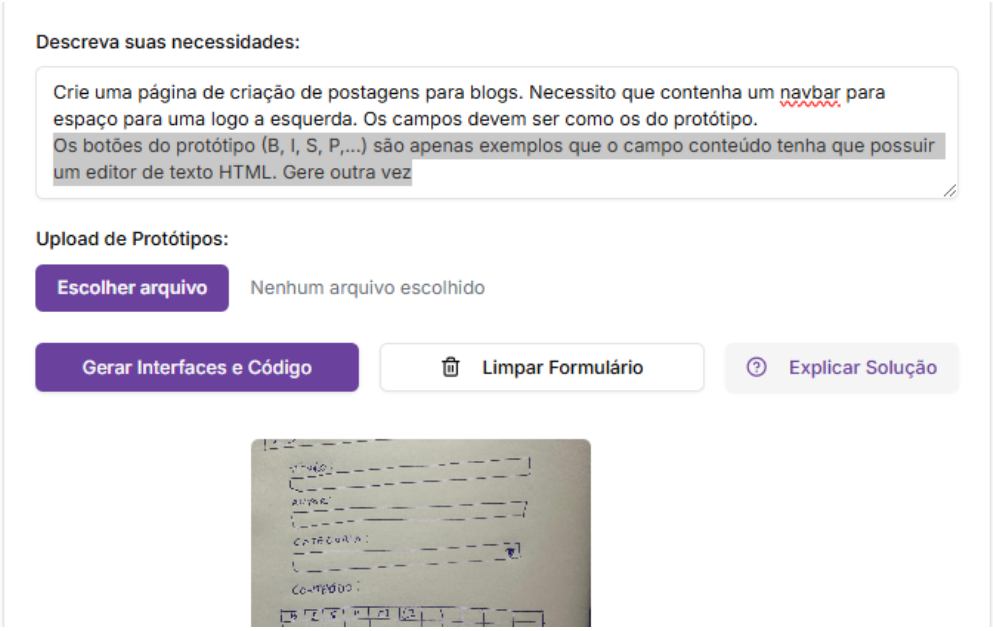

Figura 59 - Melhoria no prompt

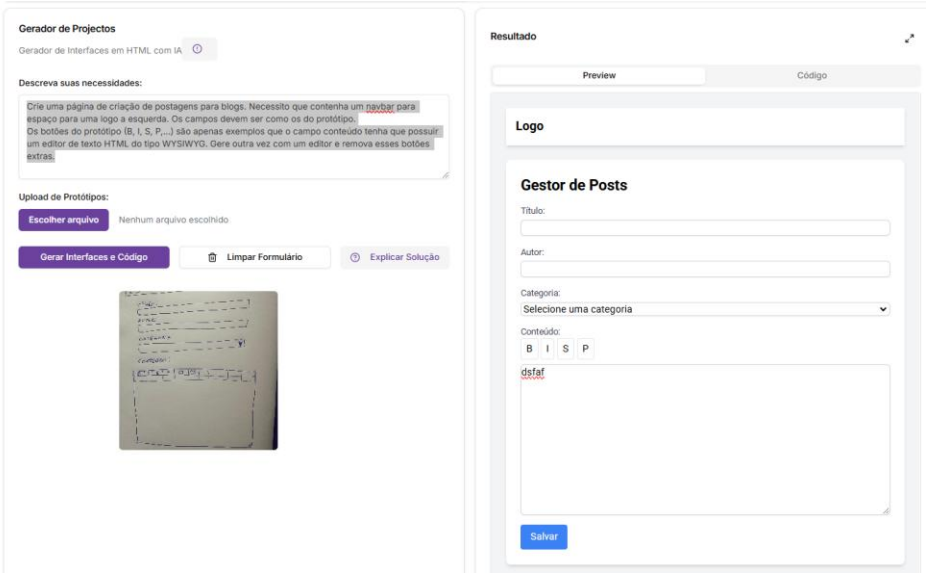

Figura 60 - Resultado e outra melhoria no prompt

A Figura 59 - Melhoria no prompt mostra o envio novamente do protótipo com incremento no *prompt*. Já a Figura 60 - Resultado e outra melhoria no prompt mostra o resultado que ainda não está como esperado e a inserção de um novo ajuste no *prompt*:

"Crie uma página de criação de postagens para blogs. Necessito que contenha um navbar para espaço para uma logo a esquerda. Os campos devem ser como os do protótipo.Os botões do protótipo (B, I, S, P,...) são apenas exemplos que o campo conteúdo tenha que possuir um editor de texto HTML do tipo WYSIWYG. Gere outra vez com um editor e remova esses botões extras."

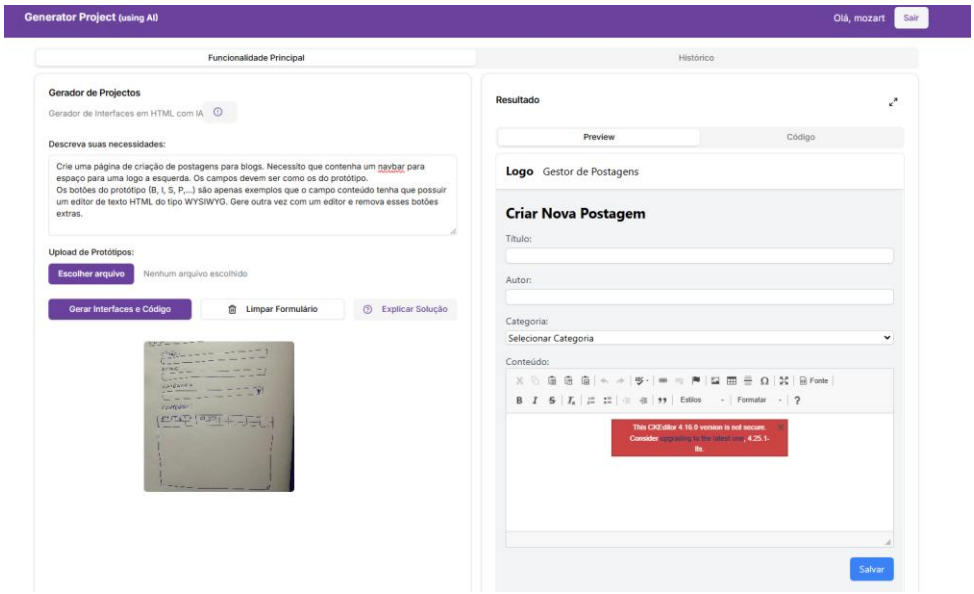

Figura 61 - Resultado esperado

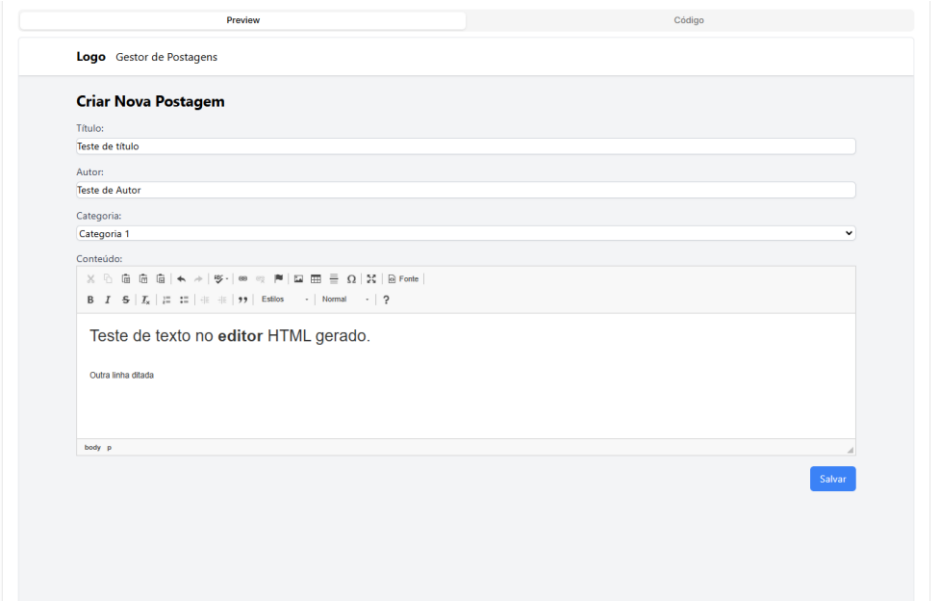

Figura 62 - Visualização em detalhe

A Figura 61 - Resultado esperado e a Figura 62 - Visualização em detalhe exibe novamente o resultado, entretanto neste momento o resultado é exatamente o esperado, a exibição de um editor de texto. É importante salientar que a modificação no prompt foi importante, desta vez foi detalhado que se esperava um editor de texto HTML WYSIWYG, que é uma sigla para a expressão em inglês “What You See is What You Get” (o que você vê é o que você recebe). A API desta vez aprimorou o resultado.

Ao considerar que o resultado é o esperado, se pode fazer uso do código. A Figura 63 - Cópia do código mostra a execução da cópia do código gerado, por sua vez a Figura 64 - Criação de um arquivo HTML com o código copiado exibe uma página HTML com nome gestor-posts.html em um diretório e que recebeu o código copiado

Figura 63 - Cópia do código

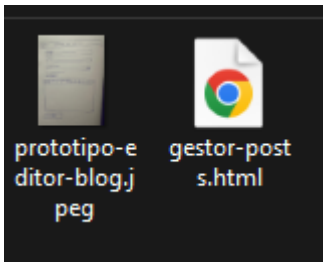

Figura 64 - Criação de um arquivo HTML com o código copiado

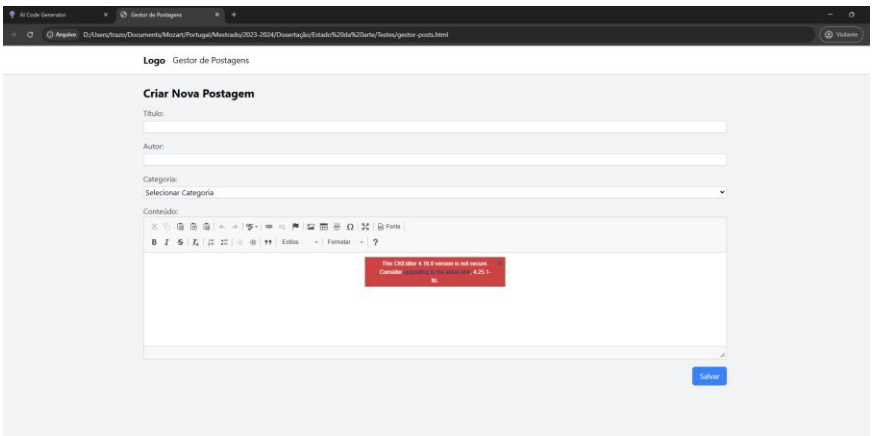

Figura 65 - Resultado da página com código copiado

. Como resultado final, a Figura 65 - Resultado da página com código copiado mostra a página HTML criada aberta em um navegador com o funcionamento esperado.

Protótipo de alta fidelidade e integração com serviços externos

O sistema desenvolvido possui a característica de geração de visuais baseados em protótipos e não é um requisito a integração com serviços ou até mesmo geração de código para back-end. Todavia, foram realizados testes para avaliar a capacidade que a API possui para integração com serviços externos que estejam disponíveis.

O protótipo visual utilizado é de alta fidelidade como pode-se ver na Figura 66 - Protótipo de alta fidelidade:

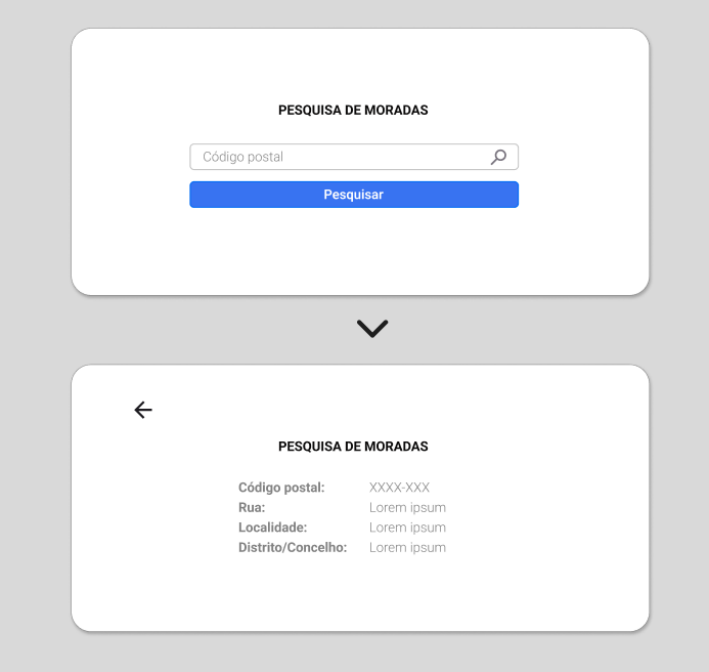

Figura 66 - Protótipo de alta fidelidade

Por sua vez, a Figura 67 - Resultado do protótipo utilizado, exibe o resultado conforme esperado. O resultado esperado é a possibilidade de pesquisa de detalhes de morada baseado em um código postal informado. O *prompt* enviado possui a seguinte instrução:

"Possuo seguinte protótipo e preciso de ajuda para criar um formulário com esse comportamento."

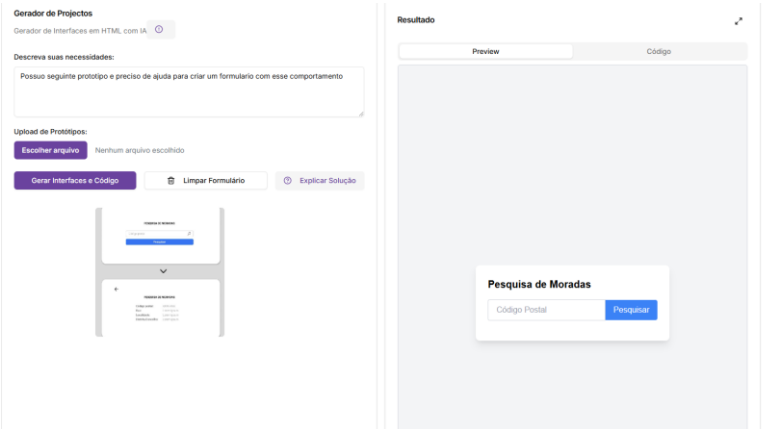

Figura 67 - Resultado do protótipo utilizado

O resultado, embora não possua todos os detalhes do protótipo enviado, foi ignorado nesta etapa pois é possível realizar os ajustes com envios de novos *prompts*. No seguimento do teste, foi inserido um número válido de um código postal, conforme é exibido na Figura 68 - Comportamento do resultado gerado.

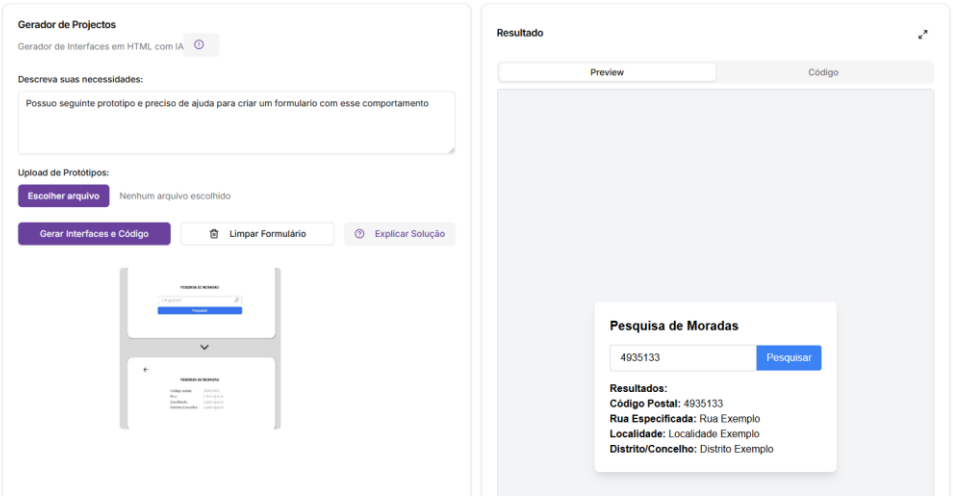

Figura 68 - Comportamento do resultado gerado

Dado o resultado, conclui-se que visualmente o resultado é satisfatório. O código gerado divide o comportamento em duas etapas, uma para informação do código postal e outra para exibição dos dados encontrados.

A API não possui conhecimento relacionado ao resultado esperado e não têm referências de como o pode fazer. Para isto, prossegue-se com um ajuste no *prompt*, de maneira a detalhar qual serviço externo pode ser consultado e é adicionado um exemplo do retorno deste serviço. O *prompt* final possui o seguinte texto:

"Possuo seguinte protótipo e preciso de ajuda para criar um formulário com esse comportamento. Os detalhes podem ser consultados na api: https://json.geoapi.pt/cp/codigo-digitado -> sendo codigo-digitado o código postal informado. Esta api tem como retorno algo assim: {"CP":"2495-300", "CP4":"2495", "CP3":"300", "Distrito":"Santarém", "Concelho":"Ourém", "Localidade":"Boleiros", "Designação Postal":"FÁTIMA"}"

O resultado por fim é satisfatório como pode-se ver na Figura 69 - Resultado final.

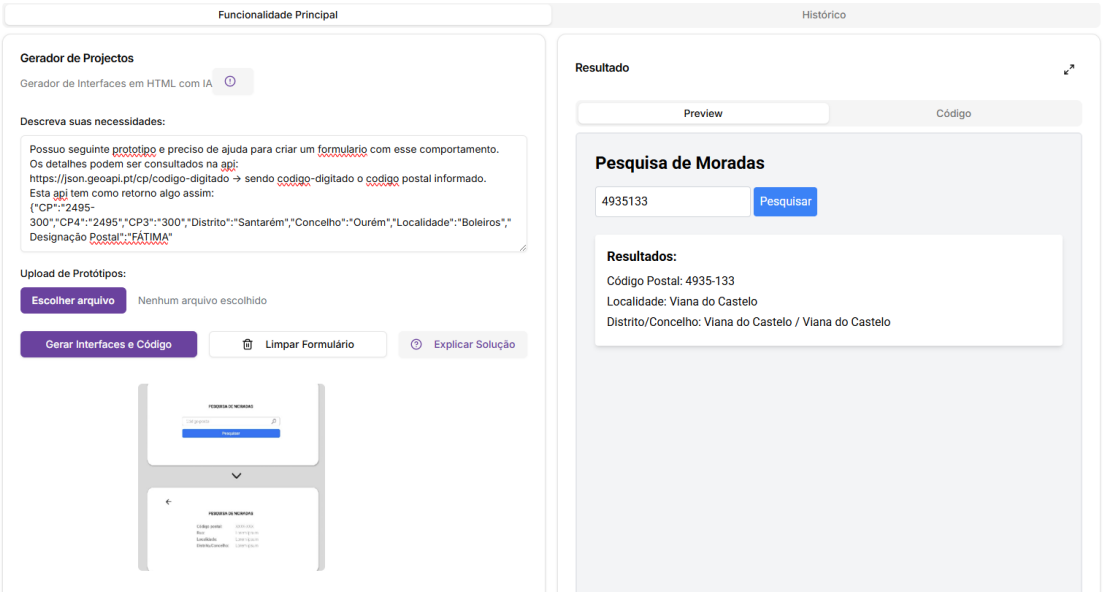

Figura 69 - Resultado final

Como observado, embora a API não conheça detalhes de funcionamento do serviço informado, apenas um exemplo do resultado esperado e o endereço de acesso é suficiente para que se tenha um código funcional. O resultado ainda permite inferir que as possibilidades vão além, desde integrações simples, até integrações mais complexas ao se fazer uso de *prompts* bem elaborados.

Explicação do resultado gerado e histórico

Até o momento não foi necessário entender o código gerado, entretanto, caso o utilizador tenha interesse, o sistema tem a funcionalidade de explicação da solução disponibilizada. Ao clicar no hiperlink exibido na Figura 70 - Função de explicação do resultado, uma janela é aberta com uma explicação do funcionamento e do conteúdo do código, como pode-se ver na Figura 71 - Retorno da aplicação com a explicação.

Figura 70 - Função de explicação do resultado

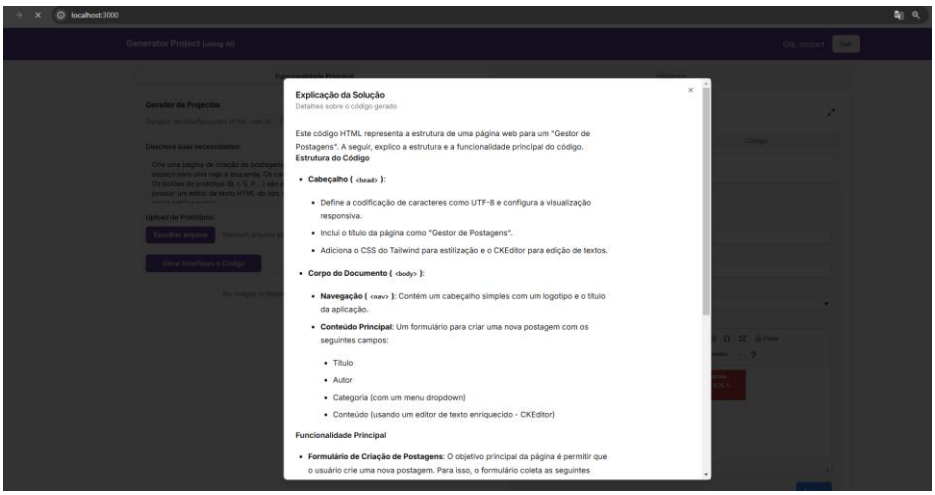

Figura 71 - Retorno da aplicação com a explicação

A explicação não fica salva na base de dados por estar fora do escopo principal do sistema. Quando o utilizador faz uso desta funcionalidade, uma nova chamada à API é realizada e recebe o resultado.

Diferente da explicação do código gerado, o histórico de geração é armazenado em base de dados. Isto é importante para garantir que o utilizador tenha acesso à tudo que já solicitou e consiga recuperar um código que não tenha ainda copiado, o que garante uma funcionalidade importante do sistema.

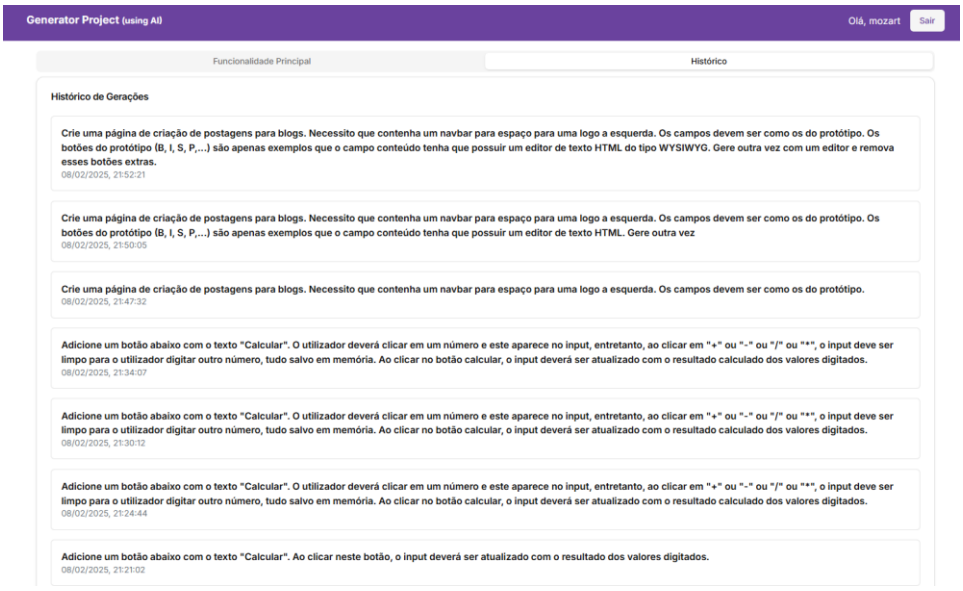

Figura 72 - Visualização do histórico

Tanto o *prompt* quanto o resultado pode ser acedido outra vez. A Figura 72 - Visualização do histórico, mostra o histórico das interações que o utilizador teve com o sistema.

5.3. Análise da Implementação do Sistema

A implementação do sistema demonstrou a viabilidade da integração entre IA e DS. Foram validadas as seguintes funcionalidades:

- **Interface Intuitiva:** O sistema apresenta uma interface funcional e acessível, permitindo ao utilizador interagir de forma simples com as ferramentas de IA.
- **Processamento via LLM:** A integração com LLM permitiu a conversão eficiente de descrições textuais e protótipos visuais em código funcional.
- **Histórico de Gerações:** O sistema possibilita o armazenamento de versões de código, permitindo a consulta e modificação incremental dos resultados.
- **Testes Automatizados:** A implementação de testes unitários, de integração e de desempenho garantiu a estabilidade da solução e a conformidade com os requisitos.
- **Assistência da IA no Desenvolvimento:** A utilização do *GitHub Copilot* demonstrou-se uma ferramenta produtiva na escrita de código, correção de erros e geração de testes automatizados, reduzindo significativamente o esforço manual dos programadores.

A combinação dessas funcionalidades confirma a eficiência do modelo proposto, permitindo a integração fluida entre as práticas tradicionais de desenvolvimento e o uso de IA para otimização do processo.

5.4. Benefícios e Desafios da Integração entre IA e Desenvolvimento de Software

Os resultados obtidos indicam que a inteligência artificial pode ser utilizada de forma eficaz para melhorar o desenvolvimento de software, tanto na fase de implementação quanto nos processos de depuração e otimização de código. Os principais benefícios observados incluem:

- **Aumento da produtividade:** A IA auxilia na escrita de código e geração de testes, acelerando o desenvolvimento.
- **Redução de erros:** A verificação automatizada de código reduz falhas e melhora a confiabilidade da solução.
- **Aprimoramento da experiência do utilizador:** A interface do sistema foi projetada para facilitar a interação com a IA, permitindo maior acessibilidade e eficiência.

- **Possibilidade de expansão:** A arquitetura modular do sistema permite a integração com outros modelos de IA, como *Claude* [68] ou *Gemini* [69], ampliando as possibilidades futuras.

Contudo, também foram identificados desafios na integração entre IA e DS:

- **Limitações dos modelos de IA:** Embora os modelos de linguagem avancem rapidamente, ainda apresentam limitações na interpretação de requisitos complexos e na geração de código otimizado para casos específicos.
- **Custo computacional:** Modelos avançados de IA requerem infraestrutura computacional significativa e isto tem um custo, o que pode ser um fator limitante em ambientes de desenvolvimento com restrições de recursos.
- **Curva de aprendizagem:** A adoção de ferramentas assistidas por IA exige adaptação por parte dos programadores, que precisam ajustar sua forma de trabalho para melhor aproveitamento da tecnologia.

5.5. Validação do Sistema Proposto

Para validar a eficácia do sistema, foram conduzidos testes funcionais e de desempenho, garantindo que os resultados atendem aos requisitos estabelecidos. Entre os principais testes realizados, destacam-se:

- **Testes de Integração:** Validaram a comunicação entre frontend, backend e a API da OpenAI, assegurando o funcionamento da geração de código.
- **Testes de Usabilidade Técnicos:** Avaliaram a experiência do utilizador com base em métricas como clareza, facilidade de navegação e resposta adequada da interface.
- **Testes de Desempenho:** Analisaram o tempo de resposta da API, garantindo que a geração de código ocorre dentro de limites aceitáveis.
- **Gestão de Erros:** Foram implementados mecanismos de tratamento de exceções, assegurando que mensagens de erro apropriadas são apresentadas ao utilizador em caso de falha.

Adicionalmente, foi realizada uma avaliação de usabilidade com utilizadores reais, através de um questionário online estruturado com base em sete critérios consolidados na literatura recente sobre Interação Humano-Computador, conforme proposto em [103]. O formulário foi implementado via *Google Forms* [4] e aplicado a 17

[4] https://docs.google.com/forms/

participantes, que avaliaram o sistema numa escala de 1 a 5 pontos, sendo 1 equivalente a "muito insatisfeito" e 5 a "muito satisfeito".

Os critérios avaliados foram:

1. **Eficácia** (capacidade de o sistema alcançar os resultados esperados);
2. **Eficiência** (rapidez e facilidade para atingir os objetivos);
3. **Satisfação geral**;
4. **Facilidade de aprendizagem** (quão intuitivo é aprender a utilizar o sistema);
5. **Acessibilidade** (adequação a diferentes perfis de utilizadores);
6. **Confiabilidade** (comportamento diante de erros);
7. **Atratividade** (design e aparência visual da interface).

Os dados foram analisados de forma quantitativa, sendo calculadas as médias ponderadas de cada critério. As pontuações ficaram consistentemente acima de 4,00, com destaque para a **satisfação** (4,82), **eficácia** (4,35) e **confiabilidade** (4,41). Os menores valores foram observados na **atratividade visual** (3,76) e na **acessibilidade** (3,94), indicando pontos de atenção para melhorias futuras. Comentários qualitativos deixados pelos participantes reforçaram sugestões como o aumento do contraste visual e a melhoria na organização dos históricos de interações. A Figura 73 - Média das Avaliações de Usabilidademostra um gráfico que representa a média alcançada de cada critério.

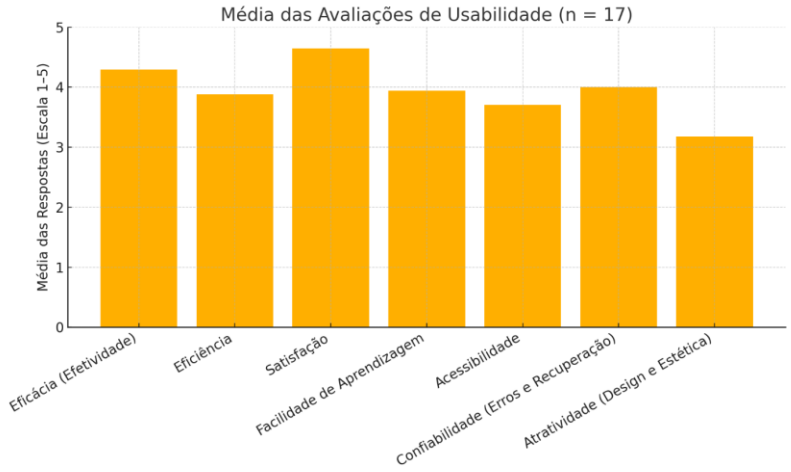

Figura 73 - Média das Avaliações de Usabilidade

Este tipo de avaliação, baseada em perceção real de utilizadores, proporciona evidência da aceitação e qualidade da solução proposta, reforçando o alinhamento com práticas modernas de design centrado no utilizador.

O formulário utilizado encontra-se disponível no Apêndice A, juntamente com os resultados consolidados da pesquisa.

Nº	Critério Avaliado	Pergunta	Escala de Resposta
1	Eficácia	O sistema conseguiu atingir os resultados esperados nas tarefas realizadas?	1 (Muito insatisfeito) a 5 (Muito satisfeito)
2	Eficiência	O sistema permite atingir os objetivos de forma rápida e com poucos esforços?	1 (Muito insatisfeito) a 5 (Muito satisfeito)
3	Satisfação Geral	Qual o seu nível geral de satisfação com a utilização do sistema?	1 (Muito insatisfeito) a 5 (Muito satisfeito)
4	Facilidade de Aprendizagem	Foi fácil aprender a utilizar o sistema na primeira utilização?	1 (Muito difícil) a 5 (Muito fácil)
5	Acessibilidade	O sistema apresenta características que facilitam o acesso para diferentes tipos de utilizadores?	1 (Muito fraco) a 5 (Muito bom)
6	Confiabilidade	Durante o uso, o sistema comportou-se de forma estável, mesmo quando surgiram erros?	1 (Muito instável) a 5 (Muito estável)
7	Atratividade Visual	O design do sistema é atrativo e agradável à vista?	1 (Muito desagradável) a 5 (Muito atrativo)
8	Comentário Aberto	Deixe aqui qualquer sugestão, observação ou comentário sobre a sua experiência com o sistema.	Campo de texto livre

Tabela 3 - Questionário de Avaliação de Usabilidade Aplicado

Nota: Todos os itens de 1 a 7 foram avaliados numa escala de 1 a 5, conforme metodologia descrita em [103].

Os resultados dos testes indicam que o sistema cumpre os objetivos propostos, proporcionando uma solução viável, funcional e bem recebida do ponto de vista técnico e da experiência do utilizador para a geração automatizada de código baseada em inteligência artificial.

5.6. CONCLUSÃO

A análise dos resultados permite a confirmação que a integração entre IA e DS pode melhorar significativamente a produtividade dos programadores, reduzir erros e

acelerar a entrega de soluções. O sistema desenvolvido demonstra a viabilidade desta integração, validando os conceitos teóricos discutidos ao longo da pesquisa.

Além disso, os testes realizados mostraram que a utilização de LLMs no DS apresenta um grande potencial, sendo uma tendência crescente na área. O projeto também evidenciou que o uso de ferramentas assistidas por IA, como o *GitHub Copilot*, pode otimizar a escrita de código e auxiliar na correção de erros, reforçando a importância da inteligência artificial no processo de desenvolvimento.

Por fim, a modularidade do sistema permite que ele seja expandido no futuro, possibilitando a integração com novos modelos de IA e a adição de funcionalidades avançadas. Dessa forma, este trabalho não apenas valida a proposta inicial, mas também abre caminho para novas pesquisas e melhorias, consolidando a IA como uma aliada importante no desenvolvimento de software.

6. CONCLUSÕES E TRABALHO FUTURO

Esta secção apresenta as conclusões obtidas a partir do desenvolvimento e análise do sistema proposto, bem como as perspetivas para trabalhos futuros, destacando possíveis melhorias e novas direções de pesquisa.

6.1. CONCLUSÃO DO TRABALHO

Este trabalho de conclusão de mestrado explorou a integração entre Inteligência Artificial e Desenvolvimento de Software (DS), demonstrando como modelos de linguagem de grande escala (LLMs) podem ser utilizados para automatizar a geração de código e otimizar processos de desenvolvimento. O sistema desenvolvido permitiu validar esta abordagem, oferecendo uma solução prática para a criação de código a partir de descrições textuais e protótipos visuais, o que permite inclusive que não programadores possam criar aplicações.

Os resultados obtidos confirmam que a utilização de IA no DS aumenta a produtividade dos programadores, reduz erros e melhora a eficiência geral do processo. A implementação do sistema demonstrou as vantagens desta abordagem, garantindo uma interface intuitiva, uma integração fluída entre os componentes e a possibilidade de evolução contínua.

Adicionalmente, a utilização do *GitHub Copilot* durante esta experiência revelou-se fundamental para a otimização do código e a automação de testes, evidenciando a importância crescente da IA como assistente à programação. O uso desta ferramenta reduziu significativamente o tempo de desenvolvimento, permitiu a identificação rápida de erros e facilitou a implementação de boas práticas de programação.

A validação do sistema através de testes funcionais, de desempenho e de usabilidade reforçou a robustez da solução proposta, comprovando que a combinação entre IA e engenharia de software é não apenas viável, mas também altamente vantajosa.

6.2. LIMITAÇÕES

Embora os resultados tenham sido positivos, algumas limitações foram identificadas ao longo do desenvolvimento. Uma das principais questões observadas foi a dependência de modelos externos, uma vez que o sistema baseia-se na API da OpenAI [98] para a geração de código. Embora seja possível integrar outras opções, como *Claude* ou *Gemini*, essa dependência pode representar desafios significativos, especialmente no que diz respeito aos custos de utilização e à disponibilidade do serviço. Qualquer alteração na política de preços ou no acesso às APIs pode impactar diretamente a

viabilidade do sistema, tornando essencial a existência de alternativas viáveis para garantir sua continuidade.

Além disso, o custo computacional associado ao uso de modelos de IA avançados representa um fator limitante para a acessibilidade da solução, uma vez que é necessário pagar por sua utilização. A necessidade de um processamento intensivo pode restringir a aplicação do sistema em ambientes com recursos financeiros ou técnicos mais limitados. Embora existam modelos mais leves, que reduzem o consumo computacional, eles podem comprometer a qualidade e a precisão das respostas geradas, criando um equilíbrio delicado entre desempenho e eficiência de custos.

Outro ponto relevante diz respeito à complexidade na interpretação de requisitos. Apesar do avanço dos modelos de linguagem, eles ainda apresentam dificuldades na compreensão de instruções mais complexas ou mal formuladas. Como resultado, o código gerado pode não atender com precisão às expectativas do utilizador sem ajustes manuais adicionais. Esta limitação reforça a necessidade de intervenções humanas para validação e refinamento dos resultados, garantindo que a geração automática de código seja um complemento, e não uma substituição total do trabalho dos programadores.

Por fim, destaca-se a curva de aprendizagem necessária para a adoção de ferramentas assistidas por IA. Programadores ou utilizadores, precisam adaptar suas práticas de desenvolvimento para melhor aproveitar as capacidades da IA, o que pode representar um verdadeiro desafio inicial. A integração eficiente dessas ferramentas exige um período de familiarização, no qual os utilizadores aprendem a estruturar melhor suas instruções para obter resultados mais precisos e eficazes.

6.3. TRABALHO FUTURO

Com base nos resultados alcançados e nas limitações identificadas, diversas direções podem ser exploradas para o aprimoramento do sistema no futuro. Uma das principais possibilidades é a expansão para múltiplos modelos de IA, permitindo que a arquitetura suporte diferentes serviços de inteligência artificial. Com essa abordagem, os utilizadores poderiam escolher entre a *OpenAI, Gemini, Claude* ou outras opções disponíveis, otimizando a geração de código conforme suas necessidades específicas. Essa flexibilidade proporcionaria maior adaptabilidade ao sistema, reduzindo a dependência de um único provedor e garantindo alternativas caso alguma API se torne indisponível ou financeiramente inviável.

Outra área de melhoria envolve a interação entre IA e utilizador, onde podem ser exploradas novas formas de interação mais intuitivas e eficientes. A implementação de assistentes de IA interativos, por exemplo, permitiria fornecer sugestões em tempo

real durante a geração de código, tornando o processo mais dinâmico e orientado. Além disso, funcionalidades como a reformulação automática de *prompts*, sugestões de otimização de código e *feedback* baseado em boas práticas de programação poderiam aumentar a precisão dos resultados gerados principalmente para utilizadores sem experiência.

O aprimoramento do tratamento de erros também se apresenta como um ponto que deve ser considerado para evolução do sistema. A introdução de mecanismos de validação pode permitir à IA identificar códigos incorretos ou incompletos e sugerir correções automáticas, tornando o processo mais confiável. Essa abordagem pode reduzir a necessidade de ajustes manuais e minimizar falhas, melhorando a experiência do utilizador e a qualidade do código produzido.

Além disso, a metodologia desenvolvida neste trabalho pode ser expandida para outras aplicações dentro do desenvolvimento de software. A abordagem pode ser utilizada na automação de testes, na depuração automática de código ou mesmo na assistência para refatoração e otimização de estruturas já existentes. Com a evolução das capacidades dos modelos de IA, há potencial para a criação de ferramentas que ajudem programadores a identificar vulnerabilidades, melhorar a legibilidade do código e aplicar padrões de desenvolvimento de forma automatizada.

Por fim, um avanço significativo seria o desenvolvimento de um modelo próprio de IA ou aplicação de um modelo de código aberto que seja especializado na geração de código. Essa iniciativa reduziria a dependência de serviços externos e possibilitaria uma solução mais personalizada, adaptada às necessidades específicas dos utilizadores do sistema. Treinar um modelo interno permitiria a implementação de regras específicas, otimizações voltadas para determinados domínios e maior controle sobre os custos operacionais, tornando a solução mais robusta e independente a longo prazo.

6.4. Considerações Finais

O presente trabalho de conclusão demonstrou a viabilidade e os benefícios da integração entre inteligência artificial e desenvolvimento de software, validando a proposta inicial de que modelos de linguagem podem otimizar significativamente a criação de código. A solução desenvolvida provou-se funcional e também escalável, abrindo novas possibilidades para a aplicação de IA no contexto da engenharia de software.

O uso de ferramentas assistidas por IA, como o *GitHub Copilot,* mostrou-se uma alternativa eficiente para aumentar a produtividade e reduzir erros, destacando-se como um recurso essencial para o programador. Além disso, os resultados obtidos indicam que a tendência de automação e IA no DS deverá continuar a crescer, tornando-se um pilar fundamental da programação.

Por fim, este trabalho não apenas valida a proposta inicial, mas também abre caminho para novas investigações, incentivando futuras pesquisas sobre a melhoria da interação entre programadores e sistemas de IA. A evolução desta tecnologia têm se destacado e promete transformar significativamente o modo como o software é desenvolvido, tornando o processo mais eficiente, acessível e inteligente.

7. REFERÊNCIAS

[1] E. C. Foster and B. A. Towles, *Software Engineering: A Methodical Approach*. CRC Press, 2022.

[2] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.

[3] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016.

[4] M. Sami, "Software Development Life Cycle Models and Methodologies," *Melsatar Blog*, 2024, [Online]. Available: http://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/

[5] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall PTR, 2001.

[6] C. Larman, *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.

[7] D. J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[8] J. P. Womack, D. T. Jones, and D. Roos, *The Machine That Changed the World: The Story of Lean Production*. Paperback, 2007.

[9] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. Paperback, 2014.

[10] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14–24, 1986, doi: 10.1145/12944.12948.

[11] A. Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley, 2004.

[12] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. McGraw-Hill Education, 2020.

[13] P. A. Laplante, *Requirements Engineering for Software and Systems*, 3rd ed. CRC Press, 2017.

[14] K. Wiegers and J. Beatty, *Software Requirements*, 3rd ed. Microsoft Press, 2013.

[15] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2003.

[16] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Addison-Wesley, 2010.

[17] W. S. Humphrey, *Managing the Software Process*. Addison-Wesley, 1989.

[18] ISO/IEC, “Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models,” 2011.

[19] Priberam. (s.d.), “Dicionário Priberam da língua portuguesa,” 2024. [Online]. Available: https://dicionario.priberam.org/

[20] I. A. Francisco, “Inteligência Artificial no Local de Trabalho: Dimensões da sua Intervenção,” 2019, *Universidade Católica Portuguesa*. Accessed: Feb. 20, 2025. [Online]. Available: https://repositorio.ucp.pt/handle/10400.14/28384

[21] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2016.

[22] A. Kaplan and M. Haenlein, “Siri, Siri, in My Hand: Who’s the Fairest in the Land? On the Interpretations, Illustrations, and Implications of Artificial Intelligence,” *Bus Horiz*, vol. 62, no. 1, pp. 15–25, 2019, doi: 10.1016/j.bushor.2018.08.004.

[23] N. Bostrom, *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[25] M. Mohammed, M. B. Khan, and E. B. M. Bashier, *Machine Learning: Algorithms and Applications*. CRC Press, 2016.

[26] Data Base Camp, “Reinforcement Learning – Simply Explained!,” 2024. [Online]. Available: https://databasecamp.de/en/ml/reinforcement-learnings

[27] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[28] N. A. Trayanova, D. M. Popescu, and J. K. Shade, “Machine Learning in Arrhythmia and Electrophysiology,” *Circ Res*, vol. 128, no. 4, pp. 544–566, 2021, doi: 10.1161/CIRCRESAHA.120.317872.

[29] P. P. Shinde and S. Shah, “A Review of Machine Learning and Deep Learning Applications,” in *Proceedings of the Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, IEEE, 2018, pp. 1–6. doi: 10.1109ICCUBEA.2018.8697857.

[30] Y. LeCun, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[31] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[32] S. J. Pan and Q. Yang, “A Survey on Transfer Learning,” *IEEE Trans Knowl Data Eng*, vol. 22, no. 10, pp. 1345–1359, 2010, doi: 10.1109/TKDE.2009.191.

[33] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Pearson, 2020.

[34] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[35] SEO North, "Stemming and Lemmatization," 2024. Accessed: Feb. 20, 2025. [Online]. Available: https://seonorth.ca/nlp/stemming-and-lemmatization/

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[37] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing," *IEEE Comput Intell Mag*, vol. 13, no. 3, pp. 55–75, 2018, doi: 10.1109/MCI.2018.2840738.

[38] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?," in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 610–623. doi: 10.1145/3442188.3445922.

[39] T. et al. Brown, "Language Models are Few-Shot Learners," *Adv Neural Inf Process Syst*, 2020, doi: 10.48550/arXiv.2005.14165.

[40] G. Yenduri *et al.*, "GPT (Generative Pre-Trained Transformer)— A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions," *IEEE Access*, vol. 12, pp. 54608–54649, 2024, doi: 10.1109/ACCESS.2024.3389497.

[41] R. Bommasani *et al.*, "On the Opportunities and Risks of Foundation Models," *CoRR*, vol. abs/2108.07258, 2021, [Online]. Available: https://arxiv.org/abs/2108.07258

[42] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. doi: 10.18653/v1/P19-1355.

[43] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[44] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*. Springer, 2015.

[45] C. C. Aggarwal, *Recommender Systems: The Textbook*. Springer, 2016.

[46] P. Lops, M. D. Gemmis, and G. Semeraro, "Content-based Recommender Systems: State of the Art and Trends," in *Recommender Systems Handbook*, Springer, 2019, pp. 73–105.

[47] A. L. Di Fante, “Understanding Recommender Systems,” 2024. [Online]. Available: https://arturlunardi.medium.com/understanding-recommender-systems-1077f4215516

[48] C. A. Gómez-Uribe and N. Hunt, “The Netflix Recommender System: Algorithms, Business Value, and Innovation,” *ACM Trans Manag Inf Syst*, vol. 6, no. 4, pp. 1–19, 2016, doi: 10.1145/2843948.

[49] Y. Zhang and X. Chen, “Explainable Recommendation: A Survey and New Perspectives,” *Foundations and Trends in Information Retrieval*, vol. 14, no. 1, pp. 1–101, 2020, doi: 10.1561/1500000066.

[50] W3C, “HTML5: A vocabulary and associated APIs for HTML and XHTML,” 2024.

[51] M. D. N. (MDN), “HTML: Hypertext Markup Language,” 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML

[52] M. D. N. (MDN), “JavaScript: The Definitive Guide,” 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[53] ReactJS, “A JavaScript library for building user interfaces,” 2024. [Online]. Available: https://reactjs.org/

[54] W. W. W. C. (W3C), “Cascading Style Sheets (CSS) Specifications,” 2024. [Online]. Available: https://www.w3.org/Style/CSS/

[55] M. Klein, I. Niks, G. Papaphilippou, Y. Saito, A. Titov, and B. Wei, “The Evolution of Garbage Collection in V8: Google’s JavaScript Engine,” in *Proceedings of the 2019 ACM SIGPLAN International Symposium on Memory Management*, 2019, pp. 1–11. doi: 10.1145/3315573.3329973.

[56] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” *IEEE Internet Comput*, vol. 14, no. 6, pp. 80–83, 2010, doi: 10.1109/MIC.2010.145.

[57] “NPM. The standard package manager for Node.js.” Accessed: Feb. 20, 2025. [Online]. Available: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

[58] A. Osmani, *Learning JavaScript Design Patterns*. O’Reilly Media, 2013.

[59] Alex Young *et al.*, *Node.js in Action*, 2nd ed. Manning Publications, 2017.

[60] “SQLite Official Documentation.” Accessed: Feb. 20, 2025. [Online]. Available: https://sqlite.org/docs.html

[61] J. Allen and I. Simon, “SQLite: Lightweight Database for Modern Applications,” *ACM Comput Surv*, vol. 51, no. 2, pp. 1–33, 2019, doi: 10.1145/3330195.

[62] S. Guthrie and E. Johnson, “Comparison of Embedded Databases for Small-Scale Applications,” *Journal of Database Technology*, vol. 32, pp. 21–30, 2014, [Online]. Available: https://doi.org/10.12345/jdbt.2014.0004

[63] Mike Owens and Grant Allen, *The Definitive Guide to SQLite*, 2nd ed. Apress, 2010.

[64] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[65] R. A. P. I. Tutorial, “What is REST API?,” 2024. [Online]. Available: https://restfulapi.net/

[66] A. Vaswani *et al.*, “Attention Is All You Need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, Curran Associates, Inc., 2017, pp. 5998–6008. doi: 10.48550/arXiv.1706.03762.

[67] OpenAI, “Introducing ChatGPT,” 2024. Accessed: Feb. 20, 2025. [Online]. Available: https://openai.com/index/chatgpt/

[68] A. AI, “Claude 3: High-Performance Language Model,” *Anthropic Research*, 2024, [Online]. Available: https://www.anthropic.com/research

[69] G. DeepMind, “Gemini AI: Advancing Language and Multimodal Understanding,” *Google AI Blog*, 2024, [Online]. Available: https://ai.googleblog.com

[70] T. Labs, *Tailwind CSS Documentation*. 2023. [Online]. Available: https://tailwindcss.com/docs

[71] Shadcn, “Shadcn/UI Documentation,” 2023. [Online]. Available: https://shadcn.dev/

[72] E. Brown, *Web Development with Node and Express: Leveraging the JavaScript Stack*. O’Reilly Media, 2019. [Online]. Available: https://expressjs.com/

[73] L. Richardson and M. Amundsen, *RESTful Web APIs: Services for a Changing World*. O’Reilly Media, 2013. [Online]. Available: https://www.oreilly.com/

[74] Auth0, “Introduction to JSON Web Tokens (JWT),” 2023. [Online]. Available: https://jwt.io/introduction/

[75] Microsoft, *TypeScript Handbook*. 2023. [Online]. Available: https://www.typescriptlang.org/docs/

[76] Vercel, “Next.js Documentation,” 2023. [Online]. Available: https://nextjs.org/docs

[77] A. Mashkoor, T. Menzies, A. Egyed, and R. Ramler, “Artificial Intelligence and Software Engineering: Are We Ready?,” *Computer (Long Beach Calif)*, vol. 55, no. 3, pp. 24–28, 2022, doi: 10.1109/MC.2022.3144805.

[78] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, “How does Machine Learning Change Software Development Practices?,” *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1857–1871, 2021, doi: 10.1109/TSE.2019.2937083.

[79] B. Aşıroğlu *et al.*, “Automatic HTML Code Generation from Mock-Up Images Using Machine Learning Techniques,” in *2019 Scientific Meeting on Electrical Electronics*

Biomedical Engineering and Computer Science (EBBT), 2019, pp. 1–4. doi: 10.1109/EBBT.2019.8741736.

[80] S. Agrawal, S. Suryawanshi, V. Arsude, N. Maid, and M. Kawarkhe, "Factors Involved in Artificial Intelligence-based Automated HTML Code Generation Tool," in *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*, 2020, pp. 238–241. doi: 10.1109/ICSIDEMPC49020.2020.9299609.

[81] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, and A. et al. Blanco, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *ArXiv*, 2021, doi: 10.48550/arXiv.2102.04664.

[82] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7. doi: 10.1145/3491101.3519665.

[83] GitHub, "GitHub Copilot: Your AI Pair Programmer," 2023, [Online]. Available: https://github.com/features/copilot

[84] A. W. M. Ahmad, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, "Towards Human-Bot Collaborative Software Architecting with ChatGPT," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE '23)*, New York, NY, USA: Association for Computing Machinery, 2023, pp. 279–285. doi: 10.1145/3593434.3593468.

[85] Z. Ozpolat, O. Yildirim, and M. Karabatak, "Artificial Intelligence-Based Tools in Software Development Processes: Application of ChatGPT," *European Journal of Technic*, vol. 12, 2023, doi: 10.36222/ejt.1330631.

[86] C. Zhang and Y. Lu, "Study on artificial intelligence: The state of the art and future prospects," *J Ind Inf Integr*, vol. 23, p. 100224, 2021, doi: 10.1016/j.jii.2021.100224.

[87] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, and L. Zhang, "TreeGen: A Tree-Based Transformer Architecture for Code Generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Apr. 2020, pp. 8984–8991. doi: 10.48550/arXiv.1911.09983.

[88] T. Erdogan, H. Altunel, and A. Tarhan, "Tools/frameworks that support development process of AI-based software: validations in white literature," *The Joint Conference of the 31st International Workshop on Software Measurement (IWSM) and the 16th International Conference on Software Process and Product Measurement (MENSURA)*, 2022, Accessed: Feb. 20, 2025. [Online]. Available: https://ceur-ws.org/Vol-3272/IWSM-MENSURA22_paper10.pdf

[89] D. Sculley *et al.*, “Hidden technical debt in machine learning systems,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2015. Accessed: Feb. 20, 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

[90] V. Nagulapati, S. Rapelli, and J. Fiaidhi, “Automating Software Development using Artificial Intelligence,” *Journal of Software Engineering and Applications*, Apr. 2020, doi: 10.36227/techrxiv.12089139.v1.

[91] K. Garg, “Impact of Artificial Intelligence on Software Development: Challenges and Opportunities,” *International Journal of Artificial Intelligence Research*, 2023, doi: 10.26821/ijshre.11.8.2023.110801.

[92] F. Lin, D. J. Kim, Tse-Husn, and Chen, “SOEN-101: Code Generation by Emulating Software Process Models Using Large Language Model Agents,” *arXiv e-prints*, p. arXiv:2403.15852, Mar. 2024, doi: 10.48550/arXiv.2403.15852.

[93] I. Siroš, D. Singelée, and B. Preneel, “GitHub Copilot: The Perfect Code compLeeter?,” *arXiv preprint*, 2024, doi: 10.48550/arXiv.2406.11326.

[94] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A Survey of Machine Learning for Big Code and Naturalness,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–37, 2018, doi: 10.1145/3212695.

[95] M. Arthur, “Automatic Source Code Documentation using Code Summarization Technique of NLP,” *Procedia Comput Sci*, vol. 171, pp. 2522–2531, 2020, doi: 10.1016/j.procs.2020.04.273.

[96] Quanjun Zhang *et al.*, “A Survey on Large Language Models for Software Engineering,” 2024. doi: 10.48550/arXiv.2312.15223.

[97] L. C. Jensen, A. G. Özkil, and N. H. Mortensen, “Prototypes in engineering design: Definitions and strategies,” *Proceedings of the DESIGN 2016 14th International Design Conference*, pp. 821–830, 2016, Accessed: Feb. 20, 2025. [Online]. Available: https://www.designsociety.org/download-publication/38892/PROTOTYPES+IN+ENGINEERING+DESIGN%3A+DEFINITIONS+AND+STRATEGIES

[98] OpenAI, “GPT-4o: Optimized AI for Multimodal Applications,” *OpenAI Documentation*, 2024, [Online]. Available: https://openai.com/research/gpt-4o

[99] “Postman - The Collaboration Platform for API Development,” 2024. [Online]. Available: https://www.postman.com/

[100] Microsoft, “Visual Studio Code: Code editing redefined,” 2022, [Online]. Available: https://code.visualstudio.com/

[101] R. H. II, *Jest Handbook: A Guide to Testing JavaScript Applications*. Independently Published, 2020. [Online]. Available: https://jestjs.io/

[102] J. D. Blischak, E. R. Davenport, and G. Wilson, “A Quick Introduction to Version Control with Git and GitHub,” *PLoS Comput Biol*, vol. 12, no. 1, p. e1004668, 2016, doi: 10.1371/journal.pcbi.1004668.

[103] S. Ntoa, “Usability and User Experience Evaluation in Intelligent Environments: A Review and Reappraisal,” *Int J Hum Comput Interact*, vol. 41, no. 5, pp. 2829–2858, 2025, doi: 10.1080/10447318.2024.2394724.