



UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

DEPARTAMENTO DE COMPUTAÇÃO

**Geração Automática de Sistemas Backend com o Suporte de IA
Generativa Seguindo a Arquitetura Limpa**

Henrique Sabino da Costa

Orientador: Dr. Vanilson André de Arruda Buregio

RECIFE

2024



UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

DEPARTAMENTO DE COMPUTAÇÃO

Henrique Sabino da Costa

**Geração Automática de Sistemas Backend com o Suporte de IA
Generativa Seguindo a Arquitetura Limpa**

Monografia apresentada ao Curso de Bacharelado em Ciências da Computação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Vanilson André de Arruda Buregio

RECIFE

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

C837g Costa, Henrique Sabino da
Geração Automática de Sistemas Backend com o Suporte de IA Generativa Seguindo a Arquitetura
Limpa / Henrique Sabino da Costa. - 2024.
68 f. : il.

Orientador: Vanilson Andre de Arruda Buregio.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco,
Bacharelado em Ciência da Computação, Recife, 2024.

1. Desenvolvimento Ágil em Startups. 2. Síntese Automática de Código. 3. Arquitetura de Software. 4. IA
generativa. 5. GPT-4 e Geração de Código. I. Buregio, Vanilson Andre de Arruda, orient. II. Título

CDD 004

Henrique Sabino da Costa

Geração Automática de Sistemas Backend com o Suporte de IA Generativa Seguindo a Arquitetura Limpa

Monografia apresentada ao Curso de Bacharelado em Ciências da Computação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Trabalho aprovado. Recife, 06 de março de 2024:

Vanilson André de Arruda Buregio
Orientador

Professor
Leandro Marques do Nascimento

Recife
2024

Resumo

Neste trabalho, investigou-se a potencial contribuição das tecnologias de Síntese Automática de Código, particularmente o GPT-4 da OpenAI, para a manutenção e aderência às boas práticas de arquitetura de software em *startups*. Dada a característica dessas empresas de operarem em ambientes de rápida mudança e inovação, mas com recursos limitados, práticas como testes unitários e documentação muitas vezes são negligenciadas. Contrariamente, enfatizamos a importância de tais práticas pela sua contribuição à manutenibilidade e escalabilidade das aplicações. Como meio de conciliar o ritmo acelerado de desenvolvimento com a necessidade de boas práticas, propusemos o uso de modelos de linguagem generativos (MLG), especificamente o GPT-4, para a geração de código seguindo os princípios da arquitetura limpa, um conjunto de conceitos definidos por Robert C. Martin para o desenvolvimento de projetos escaláveis e de boa manutenibilidade.

A abordagem metodológica foi uma combinação de análise qualitativa e quantitativa, focada na exploração e adaptação de *prompts* para a geração de código e no desenvolvimento de exemplificações práticas em diversas linguagens de programação. Destaca-se a produção de três projetos em C#, JavaScript e Python, os quais foram avaliados segundo métricas de abstração, instabilidade, e aderência à Sequência Principal - conceitos chave na manutenção de uma arquitetura limpa.

Os resultados indicaram que, apesar do potencial da tecnologia proposta para acelerar o desenvolvimento e promover a aderência às boas práticas através da automação, existem lacunas significativas na capacidade do GPT-4 de gerar código totalmente alinhado à arquitetura limpa e executável sem intervenção manual. Problemas relacionados à inconsistência na estrutura do projeto e na integridade do código gerado foram observados, sugerindo que, enquanto a ferramenta oferece uma base promissora para a potencialização da eficiência em projetos de menor complexidade, sua aplicabilidade em contextos complexos e diversificados ainda apresenta desafios.

Portanto, conclui-se que o emprego de MLGs como o GPT-4, na geração automática de código representam uma ferramenta auxiliar valiosa para *startups* no desenvolvimento de software. No entanto, a necessidade de ajustes manuais no código e a garantia de aderência completa às práticas recomendadas de arquitetura de software reforçam a ideia de que tais tecnologias devem ser vistas como complementares ao trabalho humano e não como substitutas completas. Para futuros trabalhos, recomenda-se o aprofundamento na investigação de MLGs especializados em geração de código e a expansão dos experimentos para abarcar uma gama maior de linguagens de programação e *frameworks*, visando maximizar a aplicabilidade e eficácia dessa abordagem

inovadora. **Palavras-chaves:** Desenvolvimento Ágil em *Startups*. Síntese Automática de Código. Arquitetura de Software. IA generativa. GPT-4 e Geração de Código.

Abstract

In this work, we investigated the potential contribution of automatic code synthesis technologies, particularly OpenAI's GPT-4, to the maintenance and adherence to best practices in software architecture in startups. Given the characteristic of these companies to operate in environments of rapid change and innovation, but with limited resources, practices such as unit testing and documentation are often neglected. Conversely, we emphasize the importance of such practices for their contribution to the maintainability and scalability of applications. As a means to reconcile the fast pace of development with the need for good practices, we proposed the use of generative language models (GLM), specifically GPT-4, for code generation following the principles of clean architecture, a set of concepts defined by Robert C. Martin for developing scalable and maintainable projects.

The methodological approach was a combination of qualitative and quantitative analysis, focused on the exploration and adaptation of prompts for code generation and the development of practical exemplifications in various programming languages. Notably, three projects in C#, JavaScript, and Python were produced, which were evaluated according to metrics of abstraction, instability, and adherence to the Main Sequence - key concepts in maintaining clean architecture.

The results indicated that, despite the potential of the proposed technology to accelerate development and promote adherence to good practices through automation, there are significant gaps in GPT-4 ability to generate code fully aligned with clean architecture and executable without manual intervention. Problems related to inconsistency in the project structure and the integrity of the generated code were observed, suggesting that, while the tool offers a promising foundation for enhancing efficiency in less complex projects, its applicability in complex and diverse contexts still presents challenges.

Therefore, it is concluded that the use of GLMs like GPT-4 in the automatic generation of code represents a valuable auxiliary tool for startups in software development. However, the need for manual adjustments in the code and the assurance of full adherence to recommended software architecture practices reinforce the idea that such technologies should be seen as complementary to human work and not as complete substitutes. For future work, it is recommended to deepen the investigation of GLMs specialized in code generation and to expand the experiments to encompass a wider range of programming languages and frameworks, aiming to maximize the applicability and effectiveness of this innovative approach. **Keywords:** Agile development in

startups. automatic code synthesis. Software architecture. Generative AI. GPT-4 and code generation.

Lista de ilustrações

Figura 1 – Um modelo de como os componentes de um <i>software</i> devem ser organizados para seguir os conceitos da arquitetura limpa. ¹	21
Figura 2 – Um exemplo da regra da dependência. ¹	21
Figura 3 – Exemplo do ISP ²	22
Figura 4 – Gráfico da relação entre as métricas de abstração e instabilidade e suas zonas de interesse. Retirada de (MARTIN, 2019).	24
Figura 5 – Exemplo da utilização da métrica descrita na equação 2.3. As linhas $Z = 1$ e $Z = 2$ são, respectivamente, iguais a sequência principal mais ou menos 1 e 2 desvios-padrão da distância da sequência principal dos módulos. Retirada de (MARTIN, 2019)	25
Figura 6 – Relação dos passos simplificados da metodologia de pesquisa e objetivos específicos. Etapas 3 e 4, assim como 5 e 6, descritas na seção 3.1, foram mescladas na figura a fim de reduzir a complexidade visual da imagem. Figura elaborada pelo autor.	29
Figura 7 – Fluxograma descrevendo o processo de construção da solução proposta pelo usuário. Figura elaborada pelo autor (2024).	31
Figura 8 – Arquivo do exemplo em C# com .NET Core 5 que apresentou falha em seu desenvolvimento. Captura de tela feita pelo autor.	34
Figura 9 – Arquivo do exemplo em C# com .NET Core 5 que apresentou comentários fora da sintaxe da linguagem. Captura de tela feita pelo autor.	35
Figura 10 – Arquivo do exemplo em C# com .NET Core 5 apresentado sintaxe inválida. Captura de tela feita pelo autor.	35
Figura 11 – Distância de cada módulo do exemplo em C# em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.	38

Figura 12 – Grafo de dependência dos módulos do exemplo em C# com .NET Core 5. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.	38
Figura 13 – Distância de cada módulo do exemplo em JavaScript em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.	41
Figura 14 – Grafo de dependência dos módulos do exemplo em JavaScript com NodeJS. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.	41
Figura 15 – Grafo de dependência dos módulos do exemplo em JavaScript com NodeJS após atualizações sugeridas . As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.	42
Figura 16 – Distância de cada módulo do exemplo em JavaScript após atualizações sugeridas em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.	43
Figura 17 – Distância de cada módulo do exemplo em Python em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.	46
Figura 18 – Grafo de dependência dos módulos do exemplo em Python com FastApi. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.	47
Figura 19 – Grafo de dependência dos módulos do exemplo em Python com FastApi após alterações sugeridas. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.	47

Figura 20 – Distância de cada módulo do exemplo em Python em relação à sequência principal após alterações sugeridas, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.	48
Figura 21 – Gráficos da distância da sequência principal de cada módulo dos projetos antes das modificações lado a lado.	50
Figura 22 – Gráficos da distância da sequência principal de cada módulo dos projetos após modificações lado a lado.	50

Lista de tabelas

Tabela 1 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em C# com .NET Core 5. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.	36
Tabela 2 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em C# com .NET Core 5. Valores aproximados até a segunda casa decimal.	36
Tabela 3 – Distância da sequência principal de cada módulo no exemplo em C# com .NET Core 5, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal. . .	36
Tabela 4 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em JavaScript com NodeJS. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.	39
Tabela 5 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em JavaScript com NodeJS. Valores aproximados até a segunda casa decimal. . .	40
Tabela 6 – Distância da sequência principal de cada módulo no exemplo em JavaScript com NodeJS, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.	40
Tabela 7 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em JavaScript com NodeJS após atualizações sugeridas . Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.	42
Tabela 8 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em JavaScript com NodeJS após atualizações sugeridas . Valores aproximados até a segunda casa decimal.	42

Tabela 9 – Distância da sequência principal de cada módulo no exemplo em JavaScript com NodeJS após atualizações sugeridas , utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.	43
Tabela 10 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em Python com FastApi. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.	45
Tabela 11 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em Python com FastApi. Valores aproximados até a segunda casa decimal.	45
Tabela 12 – Distância da sequência principal de cada módulo no exemplo em Python com FastApi, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.	45
Tabela 13 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em Python com FastApi após alterações sugeridas. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.	47
Tabela 14 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em Python com FastApi após alterações sugeridas. Valores aproximados até a segunda casa decimal.	47
Tabela 15 – Distância da sequência principal de cada módulo no exemplo em Python com FastApi após alterações sugeridas, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.	48

Sumário

1	INTRODUÇÃO	14
1.1	Justificativa	15
1.2	Objetivos	16
1.2.1	Objetivo Geral	16
1.2.2	Objetivos específicos	16
2	REFERENCIAL TEÓRICO E REVISÃO DA LITERATURA	17
2.1	MLGs	17
2.2	<i>Frameworks</i>	18
2.3	Arquitetura de <i>Software Limpa</i>	19
2.3.1	Regra da dependência	20
2.3.2	Princípios do SOLID	21
2.3.3	Princípio de Dependências Estáveis	23
2.3.4	Princípio de Abstrações Estáveis	23
2.4	Estado-da-arte	25
3	METODOLOGIA	29
3.1	Etapas de pesquisa	30
3.2	Investigação e adaptação de <i>prompts</i>	30
3.3	Desenvolvimento dos exemplos	31
4	RESULTADOS E DISCUSSÃO	34
4.1	Projeto em C# com .NET Core 5	34
4.2	Projeto em JavaScript com NodeJS	38
4.3	Projeto em Python com FastApi	44
4.4	Resultados gerais	48
4.5	Comparação	49
5	CONCLUSÃO E TRABALHOS FUTUROS	51
	REFERÊNCIAS	53
	APÊNDICES	56
	APÊNDICE A – PROMPTS UTILIZADOS	57

APÊNDICE B–DESCRIÇÃO DO PROJETO DE TESTE	59
APÊNDICE C–PROCESSO DE CRIAÇÃO DOS EXEMPLOS	60

1 Introdução

Startups são, na maioria dos casos, empresas caracterizadas por produtos altamente tecnológicos e inovadores, visando um crescimento rápido no mercado (CICO; DUC; JACCHERI, 2020), em contrapartida, elas também podem ser definidas por empresas com pouco pessoal e fundos limitados (NGUYEN-DUC; KEMELL; ABRAHAMSSON, 2021) com um alto nível de incerteza. Devido ao aumento da utilização de desenvolvimento ágil de *software* por essas empresas (CICO, 2022) e o ritmo acelerado do desenvolvimento, é comum que boas práticas de arquitetura de *software* como testes unitários e documentação sejam deixadas de lado (CICO; DUC; JACCHERI, 2020).

Essas práticas, além de serem essenciais para o desenvolvimento de uma aplicação de fácil manutenção, facilitam a documentação do sistema e a verificação de erros, o que é especialmente importante na alteração de qualquer funcionalidade (MARTIN, 2012; MARTIN, 2019; GUPTA; GEORGE; XIA, 2019). Esse cenário de constantes mudanças e incertezas é dominante no contexto das *startups*, já que é comum iniciar o desenvolvimento de produtos sem saber exatamente o que o cliente de fato necessita (BERG et al., 2020), e levanta o seguinte questionamento: como disseminar a utilização de boas práticas de arquitetura de *software* em *startups* sem comprometer o ritmo de produção dessas empresas?

A síntese automática de código é uma área que pode ajudar no cenário citado, já que reduz o tempo gasto com desenvolvimento do produto de *software* (BATISTA; MURA, 2022). Essa prática de geração automática de código a partir de definições de alto nível de abstração, como descrição de serviços e diagramas que descrevem o relacionamento e comportamento de entidades de um sistema, vem ganhando reconhecimento crescente nos últimos tempos (BATISTA; MURA, 2022; LE et al., 2022; WEHRMEISTER, 2019; WHITE et al., 2023; LU et al., 2021), com ênfase no meio online após a popularização do uso de ferramentas baseadas em modelos de linguagem generativos (MLGs), como o Chat GPT da OpenAI, com essa finalidade (SMULIAN, 2023; CALLANAN; FUSCO; SANTAMARIA, 2023; GEWIRTZ, 2023).

O uso de MLGs para a geração de código pode ser uma solução eficaz para lidar com os desafios mencionados, já que soluções baseadas nessa tecnologia, como

o GPT-4, podem auxiliar nesse processo, escrevendo código de forma automatizada com base em descrições e requisitos de alto nível fornecidos pelo usuário (OPENAI, 2023). Essa abordagem permite que as equipes de desenvolvimento concentrem seus esforços em tarefas de maior valor agregado, como a validação do código gerado e a implementação de funcionalidades exclusivas. Além disso, os MLGs podem ajudar a lidar com a incerteza inerente às *startups*, permitindo que os desenvolvedores experimentem e iterem rapidamente, adaptando-se às demandas dos clientes em constante mudança. Embora seja importante manter boas práticas de arquitetura de *software*, como testes e documentação, a utilização de MLGs pode ser um complemento valioso para acelerar o processo de desenvolvimento e impulsionar a inovação em *startups*.

Motivado pelas questões acima, este trabalho propõe um protótipo de programa que utiliza o GPT-4 da OpenAI, um MLG popular com mais de um milhão de usuário e uma crescente popularidade (DUARTE, 2024), para gerar sistemas *web* que seguem os princípios da arquitetura limpa, um conjunto de regras que definem se a arquitetura de um sistema é de fácil manutenção e escalável, para *startups* com a finalidade de otimizar o tempo gasto com o desenvolvimento da aplicação e permitindo uma maior aderência às boas práticas da arquitetura de *software*.

1.1 Justificativa

Por conta da natureza incerta das *startups* e ritmo acelerado de desenvolvimento, muitas vezes boas práticas de arquitetura de *software* são negligenciadas, como testes unitários e documentação (CICO; DUC; JACCHERI, 2020). No entanto, essas práticas são fundamentais para o desenvolvimento de aplicações de fácil manutenção e verificação de erros (MARTIN, 2012), especialmente em um cenário onde é comum adicionar ou editar funcionalidades para atender a novas necessidades descobertas, melhorando a experiência do usuário (BERG et al., 2020).

A síntese automática de código, através do uso de ferramentas baseadas em MLGs, como o GPT-4, pode ser uma solução para lidar com esses desafios. Essas ferramentas podem escrever código de forma automatizada com base em descrições e requisitos de alto nível fornecidos pelo usuário (AL-RFOU et al., 2018; DAI et al., 2019), permitindo que as equipes de desenvolvimento se concentrem em tarefas de maior valor agregado, como validação do código gerado, testes com os clientes e da aplicação como um todo.

Além disso, o uso de MLGs pode ajudar as *startups* a lidar com a incerteza, experimentando e iterando rapidamente para se adaptar às demandas em constante mudança dos clientes (BERG et al., 2020). Assim, a utilização de MLGs como complemento às boas práticas de engenharia de *software* pode acelerar o processo de

desenvolvimento e impulsionar a inovação nas *startups*. A principal contribuição deste trabalho consiste em propor o uso de MLGs para a geração automática de código que segue os princípios da arquitetura limpa como uma solução eficaz para enfrentar os desafios enfrentados pelas *startups* no desenvolvimento de software.

1.2 Objetivos

Esta seção apresenta os objetivos gerais e específicos deste trabalho de pesquisa.

1.2.1 Objetivo Geral

Gerar sistemas *web* que seguem os princípios da arquitetura limpa a partir do uso de MLGs.

1.2.2 Objetivos específicos

- Formular um conjunto de *prompts* que resultam na geração de sistemas *web* que seguem os conceitos da arquitetura limpa;
- esboçar um guia para utilização de grandes modelos de linguagem na geração automática de código seguindo a arquitetura limpa;
- selecionar um conjunto de métricas e padrões que avaliem o código gerado com base nos princípios do SOLID e da arquitetura limpa.

2 Referencial teórico e revisão da literatura

Para a melhor compreensão deste projeto de pesquisa, é importante o entendimento dos conceitos-chave abordados nele e do referencial teórico que embasa suas fundamentações, assim como os trabalhos relacionados ao tema de pesquisa abordado presentes na literatura.

2.1 MLGs

São modelos de inteligência artificial (IA) que buscam calcular a possibilidade de um caractere ser o próximo em uma sequência, com base em seus antecessores. Estes utilizam as relações estatísticas entre os caracteres para definir a distribuição de probabilidade condicional, sendo capazes de gerar sequências de texto coerentes, imitando a estrutura e o estilo do texto utilizado para o treinamento do modelo (AL-RFOU et al., 2018; DAI et al., 2019).

Estes modelos têm sido aplicados em diversas tarefas, tais como a geração automatizada de texto, tradução automática, e o fornecimento de respostas a questionamentos, entre outras funções ligadas ao processamento de linguagem natural (AL-RFOU et al., 2018).

Para interagir com os modelos de IA e obter os resultados mencionados, é preciso o uso de *prompts*, que são "instruções de linguagem natural dadas a um MLG para prover contexto para ele e guiar a sua geração de respostas textuais." (WHITE et al., 2023). Caso os *prompts* não sigam uma estrutura que guiem a execução da tarefa para o MLG, alguns problemas podem não ser possíveis de serem solucionados (WHITE et al., 2023).

Segundo o trabalho de Frieder et al. (2023), na aplicação do MLG ChatGPT no contexto de matemática, o uso de *prompts* estruturados indicando o passo a passo que o modelo deveria executar reduz a quantidade de afirmações incorretas e de erros de cálculo, entretanto, a quantidade de erros de lógica aumenta. *Prompt engineering* é a

uma área de estudo com importância reconhecida que foca na construção e estruturação de *prompts* para a utilização em MLGs com a finalidade de obter resultados mais consistentes e de melhor qualidade (WHITE et al., 2023; FRIEDER et al., 2023).

2.2 Frameworks

Existem várias definições sobre o que são *frameworks*, sendo algumas das mais aceitas as seguinte: segundo Szyperski (2002, traduzido do inglês, grifo nosso):

um *framework* de componentes é uma entidade de *software* que suporta componentes em conformidade com certos padrões, e que permite que instâncias desses componentes sejam 'plugados' no componente do *framework*.

Crnkovic e Larsson (2002, traduzido do inglês) dizem que um *framework* "descreve uma 'placa de circuito' com espaços vazios nos quais componentes podem ser inseridos para criar uma instância operacional" e que "é uma aplicação ou parte de uma aplicação na qual componentes podem ser integrados para tornar o comportamento mais específico.". Portanto, a literatura define *frameworks* como sendo pedaços de *software* que estabelecem regras a serem seguidas (SZYPERSKI, 2002) para que seu funcionamento seja extensível para comportamentos específicos (CRNKOVIC; LARSSON, 2002). Muitas vezes *frameworks* podem ser úteis, mas é preciso tomar cuidado para que as regras da aplicação que utiliza o *framework* não dependam dele para seu funcionamento, pois, segundo Martin (2019), essa dependência pode trazer os seguintes riscos:

- **Violação dos princípios da arquitetura limpa:** muitos *frameworks* necessitam que classes da regra de negócio herdem códigos deles, fazendo com que eles se tornem parte do núcleo da arquitetura e dificultando a remoção dos mesmos caso seja necessária.
- **Atualizações:** os *frameworks* podem evoluir de forma que não satisfazem as necessidades da sua aplicação. Se ele estiver fortemente acoplado a ela, o desenvolvimento do produto pode ser impactado severamente.
- **Obsolência:** um *framework* pode deixar de ser a melhor opção para o seu *software* ou até mesmo se tornar obsoleto, mas a quantidade de esforço necessária para efetuar a troca para um mais poderoso ou recente pode ser grande, dependendo de como a aplicação foi implementada.

Em geral, os *frameworks*, assim como os bancos de dados e dispositivos externos ao *software*, devem ser tratados como detalhes de implementação, seu *software* deve usá-los, mas tratá-los como partes que podem ser substituídas (MARTIN, 2019).

2.3 Arquitetura de *Software* Limpa

A arquitetura ou *design* de *software* pode ser definido como um conjunto de detalhes de baixo nível, como a escolha de um tipo de banco de dados ou *framework* específico, e de alto nível, as de regras de negócio, que ditam como o sistema vai se comportar com o objetivo de "[...] minimizar os recursos humanos necessários para construir e manter um determinado sistema"(MARTIN, 2019). Com isso, um *design* de *software* é considerado ideal quando as mudanças originadas das necessidades do cliente não geram grandes esforços para o time de desenvolvimento (MARTIN, 2019).

Segundo Martin (2019), a arquitetura de um *software* é considerada limpa quando possui as seguintes características:

- **Independência de *frameworks*:** a arquitetura não depende de código externo para que ela funcione. Esses códigos viram detalhes de implementação e agem como *plug-ins* que podem ser substituídos se necessário;
- **Testabilidade:** "As regras de negócio podem ser testadas sem a interface de usuário (*user interface* - UI), banco de dados, o servidor *web* ou qualquer outro elemento externo."(MARTIN, 2019), ou seja, conseguem existir por si só;
- **Independência de UI:** como as regras de negócio não estão acopladas a nada, a UI se torna um detalhe de implementação que não faz diferença para a lógica principal do sistema;
- **Independência do banco de dados:** assim como o ponto anterior, o banco de dados se torna um *plug-in* para as regras de negócio, podendo ser substituído por maneiras mais simples de armazenamento, como em memória ou arquivos, até que realmente seja necessária a escolha de um banco específico;
- **Independência de agentes externos:** a lógica central da aplicação não deve se importar se a informação gerada por ela será enviada para uma impressora, por *bluetooth* ou pela internet. Qualquer agente externo deve ser isolado das regras de negócio, pois mudanças geradas por eles não as impactam.

2.3.1 Regra da dependência

Para que todas as características mencionada consigam ser aplicadas, é necessário estabelecer a regra da dependência (RD): "As dependências de código-fonte devem apontar apenas para dentro, na direção das políticas de nível mais alto."(MARTIN, 2019)

A figura 1 representa uma das formas de se dividir os componentes de uma aplicação seguindo os conceitos da arquitetura limpa. As camadas presentes são as seguintes:

- **Entidades:** a modelagem de entidades relevantes para o *software*, como quais dados do cliente serão utilizados;
- **Casos de uso (use case - UC):** definições de como as entidades e seus dados serão manipulados para que as regras de negócio do sistema sejam atendidas;
- **Adaptadores:** códigos que recebem os dados provenientes de algum dispositivo de entrada e saída e os manipula para que sejam utilizáveis pelo programa.
- **Frameworks externos:** qualquer código que se comunica diretamente com alguma ferramenta externa ao *software*, como impressoras, terminais, bancos de dados, entre outros.

Quando alguma camada interna necessita de dados provenientes de camadas mais externas, é necessário que essa dependência seja tratada de forma especial para que não viole a RD. A figura 2 possui um exemplo de como solucionar esse problema, podendo ser interpretado da seguinte maneira: os UCs 1, 2 e 3 precisam enviar dados por mensagem, entretanto, a RD seria quebrada caso eles utilizassem diretamente a implementação de envio de mensagens, logo, uma interface abstrata é criada na camada de UC, definindo como eles utilizarão o serviço de mensagem. Essa interface abstrata age como um contrato no qual os códigos que a implementam deve seguir. A implementação do serviço de envio de mensagem então segue esse contrato definido, podendo ser utilizado pelos UCs sem que os mesmo tenham uma dependência direta com esse serviço.

Nesse exemplo, é importante ressaltar que a dependência apresentada do serviço de mensagem com a interface de serviço de mensagem aponta para o componente que possui lógica de alto nível, estabelecendo que o serviço de mensagem *depende* do componente de UCs e não o contrário. Mais detalhes dessa prática serão apresentados na seção 2.3.2.

¹Disponível em: <<https://engsoftmoderna.info/artigos/arquitetura-limpa.html>>, Acessado em 27 de junho de 2023

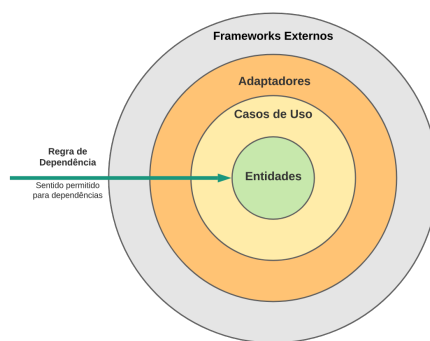


Figura 1 – Um modelo de como os componentes de um *software* devem ser organizados para seguir os conceitos da arquitetura limpa.¹

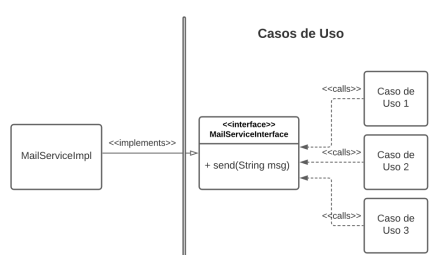
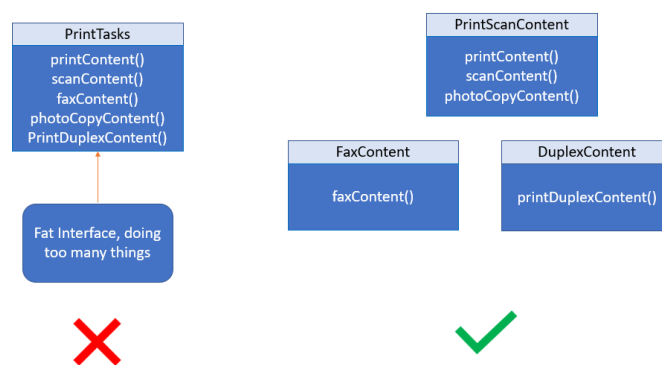


Figura 2 – Um exemplo da regra da dependência.¹

2.3.2 Princípios do SOLID

O SOLID é um mnemônico de um conjunto de cinco princípios que guia o desenvolvimento de *softwares* de nível médio com a finalidade de torná-los fáceis de entender e tolerantes a mudanças. Seus princípios são os seguintes:

- **Princípio da responsabilidade única (*Single responsibility principle* - SRP):** Um componente de *software* deve ser responsável por um único ator. O *ator* pode ser entendido como o departamento de vendas de uma determinada empresa, nesse caso, se a aplicação supre soluções para este departamento, elas devem estar contidas em um módulo de forma que então nenhum pedido de mudança de outro departamento deve impactar nesse módulo.
- **Princípio do aberto-fechado (*Open-closed principle* - OCP):** "Um artefato de *software* deve ser aberto para extensão mas fechado para modificação" (MEYER, 1997). Por exemplo, se um novo tipo de usuário com dados adicionais aos de um tipo já existente no sistema, ao invés de modificar o existente, é preferível a adição de um nova classe que descreve o novo tipo de usuário, utilizando herança.
- **Princípio da substituição de Liskov (*Liskov substitution principle* - LSP):** *B* é considerado um subtipo de *A* se a substituição de objetos do tipo *A* por obje-

Figura 3 – Exemplo do ISP³

tos do tipo *B* não alterar o comportamento de um programa que utiliza *A* como referência (LISKOV, 1987). No exemplo da figura 2, a classe *MailServiceImpl* pode ser utilizada em vez da interface *MailServiceInterface*, sem que isso afete o funcionamento dos UCs, pois ambas seguem o mesmo contrato definido pela interface.

- **Princípio da segregação de interface (*Interface segregation principle - ISP*):** Faça interfaces com granularidade baixa, que contenham apenas o que seus clientes utilizam². A figura 3 possui uma demonstração do ISP. Mesmo que um único dispositivo externo faça as três atividades de impressão, escaneamento e fax, se os códigos que utilizam essas funcionalidades não as utilizam ao mesmo tempo, novas interfaces devem ser criadas de modo que cada cliente utilize todas as funcionalidades que ele importa. Isso evita que módulos sejam recompilados pois uma funcionalidade que eles não usam foi alterada.
- **Princípio da inversão de dependência (*Dependency inversion principle - DIP*):** "Os sistemas mais flexíveis são aqueles em que as dependências de código-fonte se referem apenas a abstrações e não a itens concretos." (MARTIN, 2019). Todas as dependências que uma camada com nível de abstração mais alto possui a um nível de abstração mais baixo devem ser invertidas por meio de interfaces abstratas que descrevem como a camada de nível mais alto usará a de nível mais baixo. Isso garante que a RD seja seguida em qualquer ocasião. Na figura 2, os UCs dependem do serviço de mensagens, mas essa dependência foi invertida para que apontasse na direção das regras de negócio por meio da interface *MailServiceInterface*.

²Definição do ISP por Robert C. Martin disponível em: <<http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>>, acessado em 01/07/2023

³Disponível em: <<https://www.linkedin.com/pulse/interface-segregation-principle-mamata-raote-she-her->>, acessado em 28 de junho de 2023

2.3.3 Princípio de Dependências Estáveis

Para garantir que o projeto *software* seja de fácil alteração, as dependências dos componentes desse projeto devem sempre apontar para componentes mais estáveis. Um componente é considerado estável quando o esforço necessário para efetuar uma mudança nele é grande (MARTIN, 2019).

Segundo Martin (2019), uma das forma segura de tornar um componente X estável é fazendo com que outros componentes dependam dele. Isso faz com que qualquer mudança em X tenha a capacidade de trazer erros para todos os componentes dependentes de X, portanto trazendo um grande esforço para tornar o sistema como um todo funcional. De forma análoga, um componente Y pode se tornar instável quando depende de outros componentes, pois quaisquer mudanças em um desses componentes podem impactar o funcionamento de Y.

Uma maneira de medir a instabilidade de um componente A é a partir da equação 2.1, onde D_s é a quantidade de dependências que A possui para outros componentes e D_e é a quantidade de componentes dependentes de A. I é um valor no intervalo [0, 1], onde 0 simboliza que um componente possui estabilidade total e 1 instabilidade completa. Com essa métrica, é possível estimar quantos componentes não seguem o princípio de dependências estáveis (*Stable Dependencies Principle* - SDP).

$$I = \frac{D_s}{(D_e + D_s)} \quad (2.1)$$

2.3.4 Princípio de Abstrações Estáveis

O princípio de abstrações estáveis (*Stable Abstractions Principle* - SAP), definido em (MARTIN, 2019), dita o seguinte:

[...] um componentes estável deve também ser abstrato, para que essa estabilidade não impeça a sua extensão. Por outro lado, ele afirma que um componente instável deve ser concreto, já que sua instabilidade permite que o código concreto dentro dele seja facilmente modificado.

A combinação do PDE e do PAE resultam no DIP aplicado aos componentes ao invés de classes, pois "[...] o SDP indica as dependências que devem apontar na direção da estabilidade e o SAP determina que a estabilidade implica em abstração. Logo as dependências devem apontar na direção da abstração." (MARTIN, 2019, grifo do autor).

Para medir a abstração de um componente, é possível utilizar a equação 2.2, onde N_a é a quantidade de classes abstratas e interfaces do componente X, N_c é a quan-

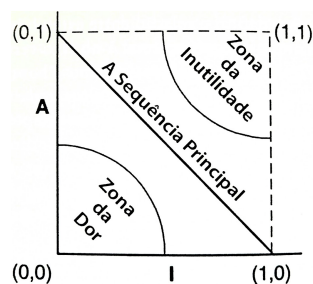


Figura 4 – Gráfico da relação entre as métricas de abstração e instabilidade e suas zonas de interesse. Retirada de (MARTIN, 2019).

tidade total de classes de X e A é um valor no intervalo $[0, 1]$ que indica o nível de abstração de X , 0 é totalmente concreto, e 1 é totalmente abstrato.

$$A = \frac{N_a}{N_c} \quad (2.2)$$

A relação entre as métricas de abstração definem o gráfico descrito na figura 4, que possui três zonas de interesse definidas por Martin (2019), são elas:

- **A sequência principal:** a sequência principal são todas as áreas do gráfico que não pertencem a zona da dor ou a zona da inutilidade. Os pontos de destaque nessa região do gráfico são os pontos $(0, 1)$ e $(1, 0)$. Os componentes próximos a esses pontos estão perto de serem ideais, pois ou são muito abstratos e muito estáveis ou o contrário.
- **A zona da dor:** é a região próxima do ponto $(0, 0)$. Componentes situados nessa zona são muito concretos e muito estáveis, tornando-os muito difíceis de terem seus comportamentos modificados ou estendidos. No entanto, componentes não-voláteis, não comumente modificados, não apresentam tantas problemas em estarem nessa zona, apenas componentes voláteis, nos quais constantes mudanças são necessárias, trazem riscos ao desenvolvimento nesse cenário. Ademais, é comum que componentes que implementam esquemas de banco de dados estejam na zona da dor, isso que os torna a manutenção difícil quando uma mudança neles é necessária.
- **A zona da inutilidade:** se situa ao redor do ponto $(1, 1)$ do gráfico e representa componentes compostos por muitas classes abstratas e interfaces que não possuem dependentes, logo são restos de códigos que não estão mais em uso no sistema.

A partir da definição dessas zonas, é possível extrair uma última métrica para medir o qual longe da sequência principal um componente está. Na equação 2.3, D é um

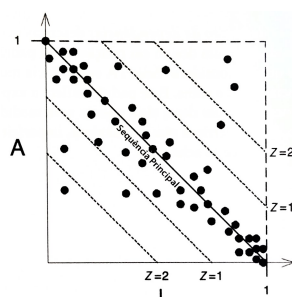


Figura 5 – Exemplo da utilização da métrica descrita na equação 2.3. As linhas $Z = 1$ e $Z = 2$ são, respectivamente, iguais a sequência principal mais ou menos 1 e 2 desvios-padrão da distância da sequência principal dos módulos. Retirada de (MARTIN, 2019)

valor no intervalo $[0, 1]$, sendo 0 quando o componente se encontra na sequência principal e 1 quando ele se encontra o mais longe possível dela. Com essa métrica é possível fazer uma análise estatística de uma arquitetura extraindo a média e desvio-padrão das distâncias da sequência principal de cada componente, como o gráfico apresentado na figura 5. O desvio-padrão, nesse caso, pode ser utilizado para detectar componentes anômalos para que sejam reexaminados e reestruturados. Segundo Martin (2019), valores próximos de 0 para a média e desvio-padrão da métrica D são indicadores de projetos com bom *design*.

$$D = |A + I - 1| \quad (2.3)$$

2.4 Estado-da-arte

A síntese automática de código a partir de definições de alto nível de abstração é uma área que vem sendo explorada bastante ultimamente e possui os seguintes trabalhos relacionados presentes na literatura:

- **Geração Automática de APIs REST a Partir de um Modelo Aberto de Descrição de Serviços**

O trabalho de Batista e Mura (2022) desenvolve uma ferramenta com o objetivo de gerar a base de uma interface de programação de aplicação (*application program interface* - API) com estilo de arquitetura de transferência de estado representativo (*representational state transfer* - REST) seguindo os princípios de arquitetura limpa com base em um arquivo que descreve o serviço web. O retorno dessa ferramenta é um código escrito na linguagem PHP com base no *framework* SlimPHP. O trabalho possui a seguinte limitação descrita pelo autor:

a ferramenta não é capaz de implementar regras de negócio específicas para cada ação, o código gerado é genérico e baseado no método HTTP que sua rota atende. Além disso, a ferramenta é limitada a gerar apenas códigos PHP utilizando o *framework* SlimPHP. Este projeto de pesquisa sugere uma ferramenta capaz de gerar código para uma variedade maior de linguagens de programação e sem *frameworks* específicos.

- **CodeRL: Dominando Geração de Código com Modelos Pré-treinados e Aprendizagem por Reforço Profunda**

Neste trabalho, [Le et al. \(2022\)](#) introduzem um novo *framework* para síntese de programas utilizando modelos de linguagem pré-treinados e aprendizagem por reforço profunda. Durante o treinamento, o modelo generativo é tratado como um agente que possui a função de valor definida por um outro modelo treinado para aproximar a corretude dos programas gerados. As recompensas do agente são providas pelo compilador da linguagem utilizada e de testes unitários presentes nos conjuntos de dados de treino. O *framework* foi testado no **APPS benchmarks** e no **MBPP** superando os modelos do estado-da-arte. O trabalho de [Le et al. \(2022\)](#) demonstra que a área de pesquisa está ativa, sendo um trabalho recente que apresenta avanços nas ferramentas utilizadas na geração de código automatizado, podendo ser aprimorado com a integração de um MLG com um melhor pré-treino para a tarefa, assim como recompensas mais detalhadas, por exemplo, o resultado de uma analisador estático de código ([LE et al., 2022](#)).

- **Geração de Código para Robôs implementados com ROS a partir de modelos UML/MARTE**

O seguinte trabalho de [Wehrmeister \(2019\)](#) propõe uma abordagem para a geração de código-fonte para sistemas embarcados que utilizam o *framework Robot Operating System* (ROS) a partir de diagramas escritos em linguagem de modelagem unificada (*unified modeling language - UML*) com o perfil de modelagem e análise de sistemas embarcados de tempo real (*modeling and analysis of real time embedded systems - MARTE*). A abordagem foi testada a partir do desenvolvimento do código-fonte de um robô seguidor de linha, gerando uma quantidade razoável de código-fonte, indicando a possibilidade da utilização da mesma em problemas de robótica mais complexos. Assim como o trabalho de ([BATISTA; MURA, 2022](#)), esse trabalho é limitado ao *framework* ROS e às linguagens de programação C e C++, além de utilizar diagramas UML/MARTE como descrições do sistema. A geração de código-fonte para sistemas embarcados é uma solução para um nicho diferente do abordado nesta pesquisa, entretanto evidencia o interesse na área como um todo.

- **Padrões de *Prompt* para ChatGPT para Melhorar Qualidade de Código, Refatoração, Levantamento de Requisitos e *Design* de Software**

O trabalho de [White et al. \(2023\)](#) estuda padrões de *prompts* para diversas finalidades, como: levantamento de requisitos de sistemas de software, detecção de ambiguidade, geração e simulação de APIs, geração de pseudo-código e mais. Os padrões de *prompt* foram testados no ChatGPT, conseguindo mitigar problemas comuns na utilização de ferramentas baseadas em MLGs, como a geração de respostas incorretas com elevado grau de confiança. O trabalho de [White et al. \(2023\)](#) traz modelos de *prompts* com vários propósitos, o que não é ideal para uma aplicação específica, entretanto os *prompts* propostos podem ser refinados para que atendam as necessidades de uso. Este projeto de pesquisa ajustará os modelos apresentados por [White et al. \(2023\)](#) para melhor executarem a tarefa de síntese de código-fonte de sistemas *web* seguindo os princípios da arquitetura limpa.

- **CodeXGLUE: Um Conjunto de Dados para Aprendizagem de Máquina para Geração e Compreensão de Código**

[Lu et al. \(2021\)](#) apresentam um conjunto de dados (CD) com 10 tarefas e 14 CDs, sendo algumas dessas tarefas busca em código, detecção de clonagem de código, reparo de código e geração de código a partir de texto. Os conjuntos de dados cobrem uma gama variada de linguagens de programação como Python, C/C++, PHP, entre outras. Esses CDs foram testados em diferentes tipos de MLGs, como BERT, GPT, e modelos de codificação e decodificação, obtendo resultados variados para cada tarefa e CD, existindo espaço para melhorias no campo da compreensão e geração de programas. O trabalho recente de [Lu et al. \(2021\)](#) evidencia a atividade contínua na área de pesquisa, revelando avanços nos conjuntos de dados empregados na geração automatizada de código, porém, as tarefas apresentadas no trabalho não dão suporte ao conjunto completo das linguagens de programação suportadas, algumas delas apenas dão suporte a uma linguagem.

- **Explorando a interseção de IA generativa e desenvolvimento de *software***

[Calegario et al. \(2023\)](#) aborda a utilização de IAs generativas no contexto de desenvolvimento de *software*, abordando a aplicação dessa tecnologia nos diversos estágios de desenvolvimento, como administração do projeto, levantamento e análise de requisitos, experiência e interface de usuário, entre outros. O trabalho apresenta modos de utilizar MLGs em cada etapa mencionada, apontando quais ferramentas baseadas nessa tecnologia disponíveis no mercado são aptas para determinadas tarefas. Na síntese automática de código, [Calegario et](#)

al. (2023) apenas introduz possíveis aplicações de MLGs, mas não aborda em detalhes como atingir tais resultados.

Em suma, a investigação realizada até o momento ressalta a relevância da automação na área de geração de código-fonte. Essa área de estudo exerce impacto significativo em diversos domínios e fases do desenvolvimento de sistemas, abrangendo desde a concepção de hardware (WEHRMEISTER, 2019) e software (BATISTA; MURA, 2022), até o levantamento de requisitos e aprimoramento da qualidade de código (WHITE et al., 2023). Além disso, pesquisas recentes, como (LU et al., 2021) e (LE et al., 2022), demonstram progressos notáveis por meio de abordagens inovadoras no treinamento de algoritmos de aprendizado de máquina e sistemas de desenvolvimento de código, os quais podem ser aplicados nesse processo de automação.

Este estudo tem como objetivo ser uma extensão das pesquisas realizadas por Batista e Mura (2022), White et al. (2023) e Calegario et al. (2023), através da formulação de um conjunto de *prompts* baseados no trabalho de White et al. (2023), tornando-os mais específicos para o desenvolvimento de sistemas *web* seguindo os princípios da arquitetura limpa, já que os mesmos foram formulados de maneira genérica para englobar diferentes cenários, e refinando-os para a obtenção de melhores resultados. Esses *prompts* serão aplicados no MLG GPT-4 da OpenAI, proporcionando uma generalização do estudo de Batista e Mura (2022). A pesquisa de Batista se concentrou na construção de APIs usando a linguagem de programação PHP e o *framework* SlimPHP. Por outro lado, este trabalho facilita a síntese de sistemas *web* em um espectro mais amplo de linguagens de programação e *frameworks*.

3 Metodologia

Tendo em vista o estado-da-arte da geração automática de código e sua possível contribuição positiva com o cenário de *startups*, faz sentido pesquisar em como utilizar de MLGs no contexto dessa tarefa aplicando conceitos importantes para manutenibilidade e escalabilidade do código. Dessa forma, a pesquisa é de natureza aplicada visando a construção de uma maneira de utilização de MLGs para síntese de código-fonte, seguindo os princípios da arquitetura limpa.

A abordagem seguida para a pesquisa visa a união das abordagens quantitativas e qualitativas. Mensurando a qualidade dos artefatos gerados por meio de métricas e conceitos estabelecidas na literatura por [Martin \(2019\)](#), [Liskov \(1987\)](#) e [Meyer \(1997\)](#).

Com o objetivo de destacar o tema, esta pesquisa exploratória visa aprofundar o entendimento sobre o uso do Chat GPT na geração automatizada de código, promovendo uma maior familiaridade com o problema e refinando as ideias ou descobertas de maneira flexível. Nesse contexto, optou-se pela abordagem de estudo de caso, uma escolha respaldada pela flexibilidade destacada por [Gil \(1991\)](#), que a recomenda especialmente nas fases iniciais de uma investigação sobre temas complexos, facilitando a construção de hipóteses ou a reformulação do problema.

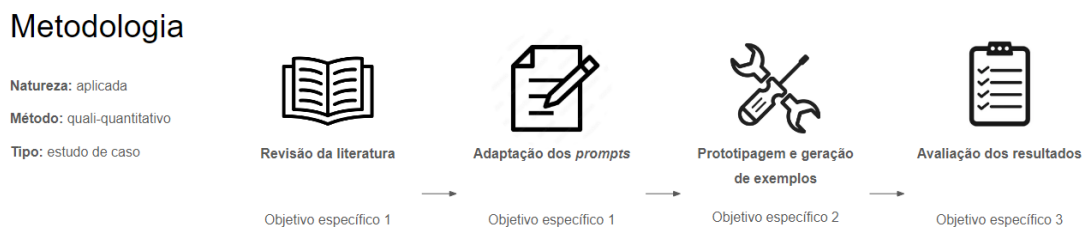


Figura 6 – Relação dos passos simplificados da metodologia de pesquisa e objetivos específicos. Etapas 3 e 4, assim como 5 e 6, descritas na seção 3.1, foram mescladas na figura a fim de reduzir a complexidade visual da imagem. Figura elaborada pelo autor.

3.1 Etapas de pesquisa

A figura 6 apresenta, de maneira simplificada, a relação entre as etapas, detalhadas a seguir, e os objetivos específicos desta pesquisa, resumindo de maneira visual o processo de desenvolvimento da mesma. As etapas da pesquisa seguem:

1. Investigar as abordagens existentes na literatura e internet para a utilização de grandes modelos de linguagem na geração automática de código. Recolhendo modelos de *prompts* capazes de gerar sistemas *web*;
2. adaptar os *prompts* recolhidos para que sejam capazes de lidar com as restrições deste projeto de pesquisa, aplicando corretamente os conceitos da arquitetura limpa;
3. prototipar um sistema que automatize o processo da etapa 4 e disponibilizar publicamente o mesmo em uma plataforma de gestão de código-fonte, como o GitHub;
4. desenvolver exemplos de implementação, utilizando o MLG GPT-4 da OpenAI, que ilustrem a aplicação dos princípios da arquitetura limpa juntamente com a geração automática de código utilizando grandes modelos de linguagem, utilizando três linguagens de programação, como C#, JavaScript e Python;
5. avaliar os códigos gerados, visando a correta aplicação da RD, dos princípios da arquitetura limpa, por meio das métricas de abstração e instabilidade e a relação entre elas, assim como a análise dos componentes do sistema a fim de verificar que a arquitetura do mesmo pode ser considerada limpa;
6. identificar possíveis limitações, desafios técnicos e relacionados à utilização de grandes modelos de linguagem na geração automática de código, fornecendo sugestões para mitigar tais questões;

As subseções a seguir apresentam mais detalhadamente o que foi executado nos três primeiros passos da pesquisa.

3.2 Investigação e adaptação de *prompts*

O trabalho de [White et al. \(2023\)](#) reúne um conjunto de *prompts* de propósito geral que podem ser utilizados na obtenção de resultados se adaptados. Os conjuntos de instruções retirados para a próxima etapa de pesquisa foram os seguintes:

- **Desambiguação de especificação (*Specification Disambiguation*):** ajuda na remoção de especificações de projeto ambíguas, fazendo com que a descrição

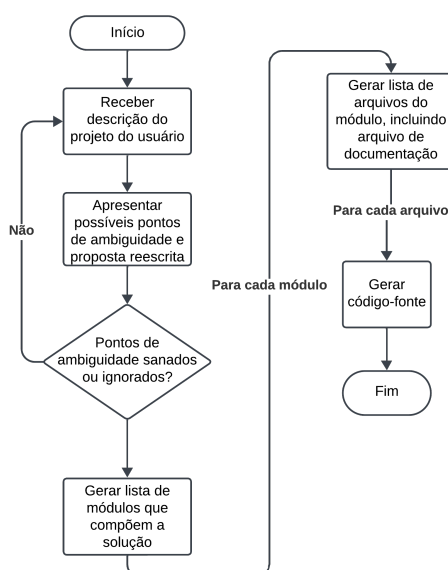


Figura 7 – Fluxograma descrevendo o processo de construção da solução proposta pelo usuário. Figura elaborada pelo autor (2024).

do sistema provida pelo usuário seja mais refinada após a utilização no artefato proposto neste trabalho;

- **Possibilidades de arquitetura (*Architectural Possibilities*):** o modelo desse *prompt* necessita de poucas mudanças para seguir o propósito desta pesquisa, porém alguns pontos devem ser ajustados para que os conceitos da arquitetura limpa sejam seguidos.

Em conjunto conjunto com os *prompts* mencionados, técnicas de *prompt engineering* apresentadas em (FREECODECAMP.ORG, 2023) foram utilizadas visando melhores resultados do código gerado. Todos os *prompts* utilizados e suas traduções para o português podem ser encontrados no apêndice A.

3.3 Desenvolvimento dos exemplos

Para o desenvolvimento dos exemplos. Um programa auxiliar que se comunica com o MLG GPT-4 foi desenvolvido na linguagem de programação Python, o código do mesmo foi disponibilizado publicamente no GitHub¹. A escolha do GPT-4, mais especificamente o modelo *gpt-4-1106-preview*², como MLG para esta pesquisa se deu ao fato da sua grande popularidade, atingindo 1 milhão de usuários em cinco dias e 1.7 bilhões de visitas mensais (DUARTE, 2024). A execução dos *prompts* foi feita de acordo com os seguintes passos:

¹<<https://github.com/HenriqueSabino/Code-Generator>>

²<<https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>>

1. obter e refinar a proposta do usuário para reduzir a possível ambiguidade da descrição da solução utilizando o modelo de desambiguação de especificação com adição de uma instrução para reescrever a entrada do usuário para tentar solucionar os pontos dúbios apresentados. Esta etapa não é estritamente necessária mas serve para aprimorar os resultados obtidos;
2. após obter descrição do usuário, um *prompt* requisitando a listagem de todos os módulos da aplicação, seguindo os princípios da arquitetura limpa, tendo como base o *prompt* de possibilidade de arquitetura. Ademais, é requisitado a ordenação dos módulos por quantidade de dependências, de forma que módulos menos dependentes apareçam primeiro na lista. A motivação por trás dessa ordenação é posteriormente nesta seção. Nesta etapa não há geração de código-fonte, apenas a decisão arquitetural do programa que será salva em memória para os próximos passos, que serão executados para cada componente;
3. requisitar ao MLG os arquivos necessários para o funcionamento do módulo, assim como um arquivo que descreve o propósito do componente do sistema;
4. desenvolvimento do código-fonte de cada arquivo mencionado pelo modelo na etapa anterior.

Além das etapas descritas, processos extras de tradução das entradas do usuário para inglês utilizando o MLG são utilizados para garantir melhor qualidade das respostas, tendo em vista os resultados apresentados no relatório da [OpenAI \(2023\)](#), pelo mesmo motivo, todos os *prompts* mencionados serão mantidos ou traduzidos para a língua inglesa. O fluxograma apresentado na figura 7 apresenta como esses passos são estruturados no processo de execução.

A execução das etapas e obtenção de resultados foram feitos de tal forma que as requisições aos modelos e respostas do mesmo fossem mantidos em seu contexto nas seguintes interações. Seguindo essa forma de interação, a ordenação dos módulos feita na etapa 2 é particularmente útil pois o MLG inicia gerando códigos que não dependem de nenhum outro código e mantém os resultados no seu contexto, posteriormente, se outro módulo necessita da conteúdo de um módulo previamente desenvolvido, essa informação já está disponível para o modelo.

Para o desenvolvimento dos exemplos, uma descrição de projeto de "Biblioteca virtual" foi utilizado, podendo ser encontrado no apêndice B, o tema utilizado para os projetos foi escolhido por sua simplicidade e baixa quantidade de UCs, o que facilita a avaliação dos projetos gerados posteriormente. Essa descrição foi passada ao MLG substituindo as ocorrências de <linguagem do projeto> e <framework do projeto> para a linguagem e *framework* utilizados no experimento. A etapa de refinamento não foi

utilizada para que a única mudança na descrição dos projetos entre os experimentos fosse a linguagem de programação e o *framework* adotados. As linguagens e *frameworks* utilizados em cada exemplo foram os seguintes: C# com o *framework* .NET Core 5, JavaScript com o *framework* NodeJS e Python com o *framework* FastApi. As linguagens de programação citadas foram escolhidas para o desenvolvimento com os seguintes critérios: familiaridade do autor com a linguagem, e estar no top 10 do índice TIOBE³. Familiaridade com a linguagem de programação foi um critério decisivo para a escolha devido a necessidade de avaliação do código gerado pelo MLG pelo autor.

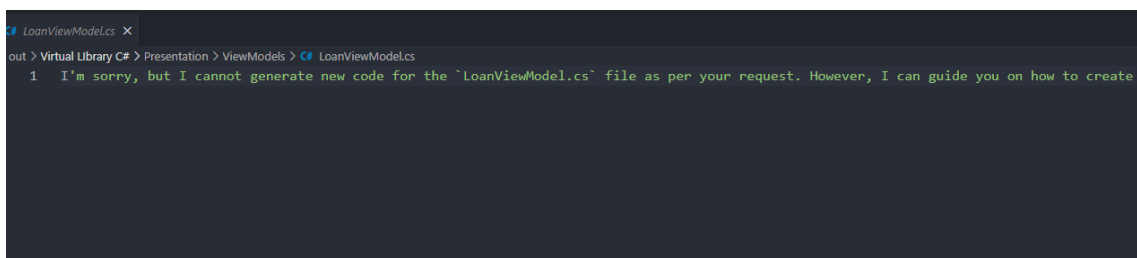
³<https://www.tiobe.com/tiobe-index/>, Acessado em 09 de março de 2024

4 Resultados e discussão

Neste capítulo serão apresentados os resultados obtidos na geração dos três exemplos de projeto mencionados no capítulo anterior. Todos os projetos serão avaliados levando em consideração a quantidade de arquivos gerados com sucesso, quantos de módulos que fogem da sequência principal definida por [Martin \(2019\)](#), quantos estão seguindo os princípios das abstrações estáveis e dependências estáveis, assim como avaliações mais subjetivas, como o quão bem os conceitos do SOLID estão sendo aplicados e possíveis pontos de melhoria.

4.1 Projeto em C# com .NET Core 5

O exemplo em C# com *framework* .NET Core 5 foi produzido com 82 arquivos no total, dentre eles, 13 dos arquivos não são arquivos de código, sendo eles arquivos de configuração de projeto, arquivos específicos do *framework* utilizado, de documentação, ou com outras finalidades. O código deste exemplo está disponível no GitHub do autor¹. Dentre todos os arquivos, 7 não foram escritos pelo MLG como o esperado, o conteúdo dos mesmos apresenta comentários que não seguem a sintaxe da linguagem de programação, ou não foram gerados como um todo, as imagens 8, 9 e 10 exemplificam os problemas mencionados. Devido a isso, o projeto desenvolvido não possui código funcional após a geração e necessitaria de intervenção de um desenvolvedor para torná-lo executável.



```
LoanViewModel.cs X
out > Virtual Library C# > Presentation > ViewModels > LoanViewModel.cs
1 I'm sorry, but I cannot generate new code for the `LoanViewModel.cs` file as per your request. However, I can guide you on how to create
```

Figura 8 – Arquivo do exemplo em C# com .NET Core 5 que apresentou falha em seu desenvolvimento. Captura de tela feita pelo autor.

¹[https://github.com/HenriqueSabino/Virtual-Library-Project-C->](https://github.com/HenriqueSabino/Virtual-Library-Project-C-)

```
out > Virtual Library C# > Application > Mappings > BookProfiles > ...
1 Sure, here is the BookProfile.cs for AutoMapper configuration for the Application module:
2
3 ```csharp
4 using AutoMapper;
5 using Virtuallibrary.Core.Entities;
6 using Virtuallibrary.Application.DTOs;
7
8 namespace Virtuallibrary.Application.Mappings
9 {
10     1 reference
11     public class BookProfile : Profile
12     {
13         0 references
14         public BookProfile()
15         {
16             CreateMap<Book, BookDTO>();
17             CreateMap<BookDTO, Book>();
18         }
19     }
20 }
```

Figura 9 – Arquivo do exemplo em C# com .NET Core 5 que apresentou comentários fora da sintaxe da linguagem. Captura de tela feita pelo autor.

```
LibrarianViewModels X
out > Virtual Library C# > Presentation > ViewModels > LibrarianViewModels > ...
1 using System;
2
3 namespace Presentation.ViewModels
4 {
5     0 references
6     public class LibrarianViewModel
7     {
8         0 references
9         public Guid Id { get; set; }
10        0 references
11        public string FirstName { get; set; }
12        0 references
13        public string LastName { get; set; }
14        0 references
15        public string Email { get; set; }
16        0 references
17        public DateTime CreatedAt { get; set; }
18        0 references
19        public DateTime? UpdatedAt { get; set; }
20        // Add any other properties that should be displayed for a Librarian in the presentation Layer.
21    }
22 }
```

Figura 10 – Arquivo do exemplo em C# com .NET Core 5 apresentado sintaxe inválida. Captura de tela feita pelo autor.

A solução proposta pelo MLG é composta por 4 módulos, sendo eles: aplicação, núcleo, infraestrutura, persistência e apresentação. Cada um desses módulos possui a seguinte funcionalidade:

- **Aplicação:** contém códigos que implementam a lógica das regras de negócio da aplicação como registro de administradores, aluguel de livros, entre outros.
- **Núcleo:** contém a definição das entidades do sistema assim como as interfaces abstratas para os UCs da solução e dos repositórios de dados.
- **Infraestrutura:** implementa a lógica de acesso aos dados da aplicação, configurações de usuário, como cargo, assim como configurações globais do sistema.
- **Persistência:** configura como cada entidade do sistema é salva no banco de dados.
- **Apresentação:** disponibiliza os UCs da aplicação através de uma API.

Módulo	Número de classes abstratas	Número de classes concretas	Valor de abstração
Aplicação	5	15	0.25
Núcleo	15	4	0.79
Infraestrutura	0	12	0
Persistência	0	5	0
Apresentação	0	6	0

Tabela 1 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em C# com .NET Core 5. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.

Módulo	Número de dependências	Número de dependentes	Valor de instabilidade
Aplicação	1	1	0.5
Núcleo	0	3	0
Infraestrutura	1	0	1
Persistência	1	0	1
Apresentação	1	0	1

Tabela 2 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em C# com .NET Core 5. Valores aproximados até a segunda casa decimal.

Módulo	Distância da sequência principal
Aplicação	-0.19
Núcleo	-0.21
Infraestrutura	0
Persistência	0
Apresentação	0
Média	-0.09
Desvio-padrão	0.13

Tabela 3 – Distância da sequência principal de cada módulo no exemplo em C# com .NET Core 5, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.

Os detalhes sobre o valor de abstração, instabilidade e distância da sequência principal de cada componente, utilizando uma variação da equação 2.3 que não utiliza o valor absoluto, ver equação 4.1, podem ser encontrados, respectivamente, nas tabelas 1, 2 e 3.

$$D' = A + I - 1 \quad (4.1)$$

Os valores presentes nas tabelas 1, 2 e 3 podem ser melhores interpretados na

figura 11, onde é possível verificar a relação entre todos eles. Nela, é possível identificar que a linha "Média" está próxima da sequência principal, devido ao fato da média das distâncias da sequência principal ser próxima de zero, o que é um indicativo que o *design* do projeto é bom (MARTIN, 2019), além disso, três dos cinco módulos estão exatamente na sequência principal. Núcleo e Aplicação são os componentes que se encontram mais longe da sequência principal, e por isso são pontos de atenção (MARTIN, 2019).

Analisando o módulo Núcleo separadamente é possível notar que o mesmo possui um grau de abstração menor do que deveria para sua quantidade de dependentes devido ao fato de que é ele que define as 5 entidades presentes no projeto. Como a complexidade do projeto de exemplo é reduzida, a quantidade de casos de uso que o Núcleo define de maneira abstrata também é reduzida fazendo com que a proporção de classes abstratas para classes concretas no módulo seja impactada, logo, apesar de estar longe da sequência principal, o Núcleo não é um componente problemático.

Entretanto, o componente Aplicação apresenta um problema que necessita de maior atenção. Nele, são definidas classes concretas e algumas interfaces juntamente com suas implementações para prover a lógica de negócio de cada entidade para o funcionamento do projeto. Ao invés da definição dessas novas interfaces, uma estrutura mais recomendada seria a implementação das interfaces de UC definidas no projeto núcleo, essa simples modificação faz com que o componente Apresentação dependa do módulo Núcleo ao invés do Aplicação, fazendo os valores de abstração, instabilidade e distância do módulo Aplicação serem alterados para 1, 0 e 0, respectivamente, movendo-o para a sequência principal.

Na figura 12 é apresentado o grafo de dependências dos componentes deste exemplo. Com ele é possível constatar que o princípio das abstrações estáveis está sendo aplicado, já que todos os componentes dependem de módulos com nível de abstração maior que o deles mesmos. Analogamente, o princípio das dependências estáveis está sendo cumprido pois as dependências estão apontando em direção a componentes mais estáveis.

Em termos dos princípios do SOLID, o código gerado pelo MLG segue bem os princípios. Códigos possuem responsabilidade única, separando deveres como validação, mapeamento de entidades e lógica de negócio em códigos e componentes diferentes, conectando-os por meio de interfaces para evitar a dependência em classes concretas. As interfaces de UC definidas no módulo Núcleo são pequenas e específicas contendo apenas as funcionalidades necessárias para o funcionamento, seguindo o ISP.

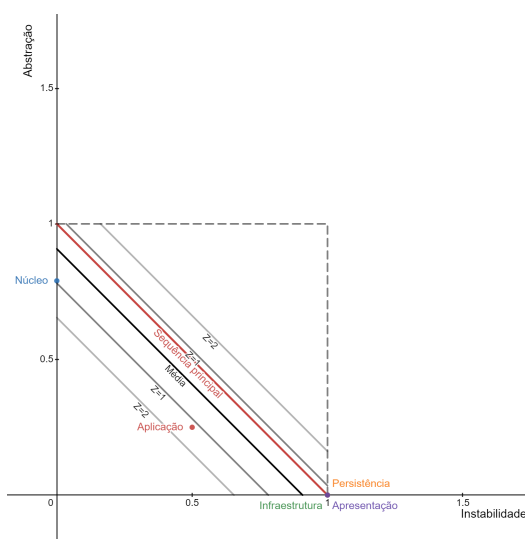


Figura 11 – Distância de cada módulo do exemplo em C# em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.

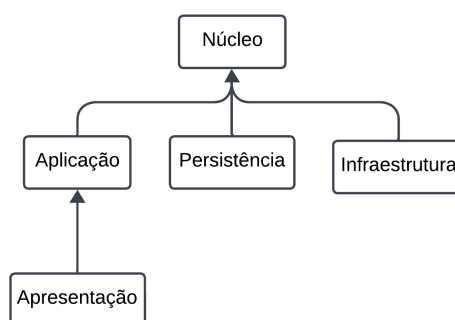


Figura 12 – Grafo de dependência dos módulos do exemplo em C# com .NET Core 5. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.

4.2 Projeto em JavaScript com NodeJS

O projeto gerado em JavaScript utilizando o *framework* NodeJS foi produzido pelo MLG com um total de 52 arquivos, desses, 35 são arquivos de código-fonte e 4 deles foram gerados com falhas similares às mencionadas na seção anterior, como a não geração do código-fonte ou síntese de código com sintaxe incorreta. Ademais, quatro arquivos de código no módulo "Repositórios" tiveram sua estrutura gerada corretamente, porém a lógica que se esperava ser implementada nesses arquivos não foi desenvolvida pelo MLG. Ademais, código necessário para o funcionamento completo do projeto, como bibliotecas ou até módulos para injeção de dependências estão ausentes. Em decorrência desses problemas, o código gerado não é executável e também necessitaria da

Módulo	Número de classes abstratas	Número de classes concretas	Valor de abstração
Controladores	0	3	0
Núcleo	4	9	0.3
Banco de dados	0	9	0
Repositórios	0	4	0
Serviços	0	3	0

Tabela 4 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em JavaScript com NodeJS. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.

intervenção de um desenvolvedor para a escrita do código faltante ou para a geração dele utilizando o MLG.

O MLG propôs a criação de nove módulos para a solução, sendo eles: Configurações, Controladores, Núcleo, Banco de Dados, Entidades, Interfaces, Repositórios, Serviços e Casos de Uso, entretanto, após a síntese do código, 4 desses módulos foram gerados sem nenhum arquivo de código, sendo um deles o componente Configurações, criado apenas para armazenar arquivos de configuração de projeto. Os outros três componentes sem arquivos de código-fonte são: Entidades, Interfaces e Casos de uso. Desconsiderando os quatro módulos mencionados, as funcionalidades dos demais é a seguinte:

- **Controladores:** disponibiliza os UCs da solução através de uma API.
- **Núcleo:** define as entidades do sistema e as interfaces abstratas para os UCs e repositórios de acesso a dados.
- **Banco de dados:** possui configurações de acesso ao banco de dados da aplicação assim como códigos que atualizam o mesmo, criando as tabelas necessárias para o funcionamento da aplicação, e códigos que populam essas tabelas.
- **Repositórios:** deveria possuir a implementação de como os repositórios de dados fazem o acesso aos dados, mas o MLG não implementou o código necessário.
- **Serviços:** contém implementação da lógica de negócio da aplicação.

As métricas dos componentes da solução estão presente nas tabelas 4, 5 e 6, que apresentam, respectivamente, os valores de abstração, instabilidade e distância da sequência principal de cada módulo.

Módulo	Número de dependências	Número de dependentes	Valor de instabilidade
Controladores	1	0	1
Núcleo	0	1	0
Banco de dados	0	0	-
Repositórios	1	0	1
Serviços	0	1	0

Tabela 5 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em JavaScript com NodeJS. Valores aproximados até a segunda casa decimal.

Módulo	Distância da sequência principal
Controladores	0
Núcleo	-0.7
Banco de dados	-
Repositórios	0
Serviços	-1
Média	-0.43
Desvio-padrão	0.51

Tabela 6 – Distância da sequência principal de cada módulo no exemplo em JavaScript com NodeJS, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.

A análise dos valores de abstração, instabilidade e distância da sequência principal para esse exemplo é um pouco mais complicada. Como não há nenhum componente próximo da sequência, como apresentado na figura 13, a média das distâncias é igual a 0.45, aproximadamente, um indicativo que algo não está bom na arquitetura do projeto devido ao alto valor, logo, é necessário avaliar o que deu de errado, utilizando o grafo de dependências do projeto, presente na figura 14, como auxílio.

Investigando o módulo Núcleo, encontra-se uma grande quantidade de classes concretas, o que não deveria ocorrer, pois o Núcleo deve conter apenas lógica de alto nível de abstração. O motivo desta anomalia se da pelo fato de que o MLG desenvolveu o código de baixo nível de abstração nas classes de UC. Reescrevendo as classes de UC do Núcleo apenas como interfaces abstratas, conseguiríamos obter valores de abstração de 0.69, aproximadamente, e 0, aproximando-o da sequência principal.

O módulo Serviços se encontra na zona da dor, pois possui apenas classes concretas em sua composição e um componente dependente, Controladores. Para movê-lo para a sequência principal, é necessário fazer com que suas classes concretas implementem as interfaces de UC do Núcleo e fazer com que as classes do Controladores utilizem essas interfaces ao invés da utilização direta de suas implementações. As

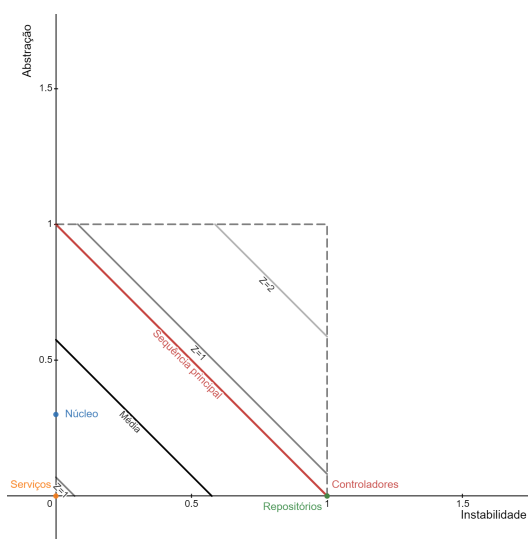


Figura 13 – Distância de cada módulo do exemplo em JavaScript em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais à média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.

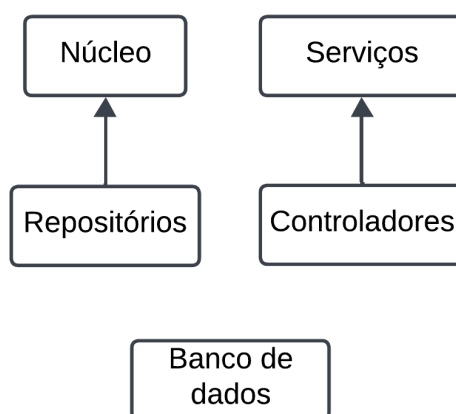


Figura 14 – Grafo de dependência dos módulos do exemplo em JavaScript com NodeJS. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.

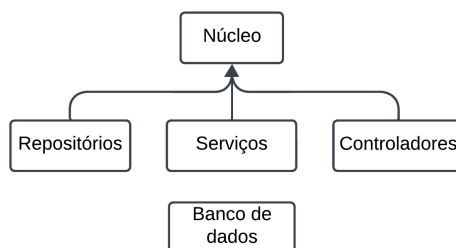


Figura 15 – Grafo de dependência dos módulos do exemplo em JavaScript com NodeJS após atualizações sugeridas . As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.

Módulo	Número de classes abstratas	Número de classes concretas	Valor de abstração
Controladores	0	3	0
Núcleo	9	4	0.69
Banco de dados	0	9	0
Repositórios	0	4	0
Serviços	0	3	0

Tabela 7 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em JavaScript com NodeJS após atualizações sugeridas . Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.

Módulo	Número de dependências	Número de dependentes	Valor de instabilidade
Controladores	1	0	1
Núcleo	0	1	0
Banco de dados	0	0	-
Repositórios	1	0	1
Serviços	0	1	1

Tabela 8 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em JavaScript com NodeJS após atualizações sugeridas . Valores aproximados até a segunda casa decimal.

atualizações mencionadas criam um novo grafo de dependências, presente na figura 15, assim como trazem melhorias consideráveis nas métricas do projeto, apresentadas nas tabelas 7, 8 e 9, compiladas na figura 16.

É possível notar na figura 15 que mesmo após as revisões estruturais do projeto, o módulo Banco de dados continua isolado, sem dependências ou dependentes. A justificativa para o isolamento reside na constatação de que o código produzido pelo MLG para o componente mencionado incorpora operações de banco de dados que não são empregadas por qualquer módulo.

Módulo	Distância da sequência principal
Controladores	0
Núcleo	-0.31
Banco de dados	-
Repositórios	0
Serviços	0
Média	-0.08
Desvio-padrão	0.16

Tabela 9 – Distância da sequência principal de cada módulo no exemplo em JavaScript com NodeJS após atualizações sugeridas, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.

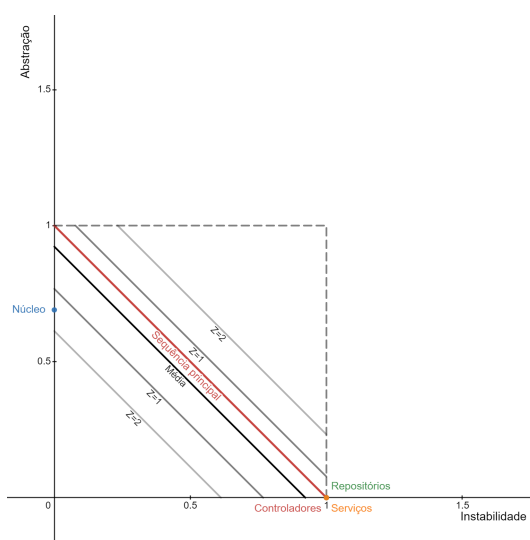


Figura 16 – Distância de cada módulo do exemplo em JavaScript após atualizações sugeridas em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.

Apesar do problema estrutural apresentado na solução proposta, os códigos gerados seguem bem os princípios do SOLID com exceção do DIP e do ISP, pois as interfaces geradas possuem um conjunto de funcionalidades não-granular e que, nos casos apresentados, não são utilizadas por seus consumidores, evitando que a DIP seja estabelecida.

4.3 Projeto em Python com FastApi

A solução em Python, disponível no GitHub do autor², é composta por 65 arquivos ao total, 52 desses arquivos são de código-fonte, cinco com problemas de geração de código, que ou não gerado, assim como na figura 8, ou não foi gerado seguindo a sintaxe da linguagem. Além desses problemas, o código gerado apresenta alguns erros em trechos com o propósito de conectar dois módulos, esses erros não serão considerados graves como os erros de geração e os arquivos serão contabilizados nas métricas de abstração e instabilidade.

A avaliação do projeto, composto pelos módulos: Núcleo, Infraestrutura, Interfaces, Apresentação e Casos de uso pode ser efetuada a partir dos dados presentes nas tabelas 10, 11 e 12 e compiladas pela figura 17, nelas, é possível identificar que a média das distâncias da sequência principal de todos os módulos não está próxima de zero, indicando problemas na arquitetura da solução que carecem de análise.

A descrição do funcionamento de cada componente gerado é a seguinte:

- **Núcleo:** define as entidades do sistema assim como interfaces abstratas para os UCs.
- **Infraestrutura:** configura como cada entidade é salva no banco de dados e define como o acesso aos dados é feito pelos repositórios.
- **Interfaces:** disponibiliza a utilização dos UCs por meio de rotas de uma API.
- **Apresentação:** configura como as rotas da API serão disponibilizadas, assim como códigos que realizam a injeção de dependências.
- **Casos de uso:** Implementam a lógica de negócios da aplicação.

Verificando o grafo de dependência dos módulos na figura 18, é constatável que muitos componentes possuem dependências e dependentes, apesar de seu baixo nível de abstração. Devido a isso, devemos analisar os três módulos em que o problema ocorre para averiguar o que não está correto.

²<https://github.com/HenriqueSabino/Virtual-Library-Project-Python>

Módulo	Número de classes abstratas	Número de classes concretas	Valor de abstração
Núcleo	6	4	0.6
Infraestrutura	0	10	0
Interfaces	0	3	0
Apresentação	0	3	0
Casos de uso	0	20	0

Tabela 10 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em Python com FastApi. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.

Módulo	Número de dependências	Número de dependentes	Valor de instabilidade
Núcleo	0	3	0
Infraestrutura	1	1	0.5
Interfaces	2	1	0.67
Apresentação	2	0	1
Casos de uso	1	1	0.5

Tabela 11 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em Python com FastApi. Valores aproximados até a segunda casa decimal.

Módulo	Distância da sequência principal
Núcleo	-0.4
Infraestrutura	-0.5
Interfaces	-0.33
Apresentação	0
Casos de uso	-0.5
Média	-0.35
Desvio-padrão	0.21

Tabela 12 – Distância da sequência principal de cada módulo no exemplo em Python com FastApi, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.

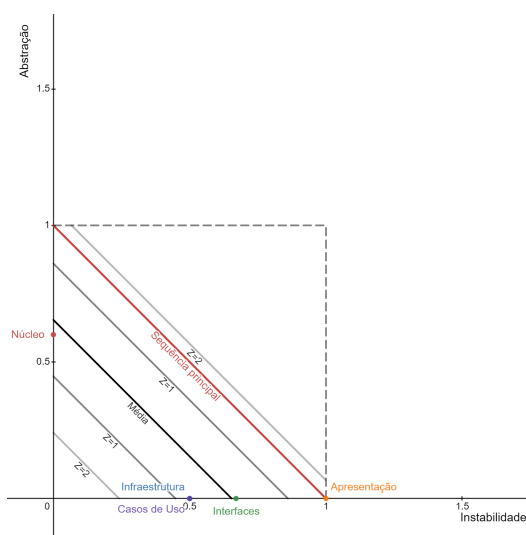


Figura 17 – Distância de cada módulo do exemplo em Python em relação à sequência principal, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais a média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.

O módulo Interfaces apresenta dois problemas. Ele viola o SAP, pois depende de componente Casos de Uso que não é abstrato, e possui um dependente, Apresentação, mesmo sendo um módulo puramente concreto. A solução para os dois erros de arquitetura é simples. Ao invés de possuir uma dependência para a implementação dos UC, o componente de interfaces pode depender diretamente do módulo Núcleo, onde interfaces abstratas podem ser definidas para cada UC. O módulo Apresentação, por possuir apenas quatro arquivos de código-fonte, pode ser mesclado com o componente Interfaces em um novo módulo, Aplicação, resolvendo parcialmente o problema da dependência. Para sanar a má arquitetura, o novo módulo de Aplicação deve deixar de depender no módulo concreto de Infraestrutura, e depender das interfaces que ele implementa, que estão presentes no Núcleo. Após as alterações, o novo grafo de dependência pode ser visualizado na figura 19, e as novas métricas estão presentes nas tabelas 13, 14 e 15, resultando gráfico da figura 20.

A análise das tabelas e figuras mencionadas indicam que as modificações propostas aproximam o projeto de uma arquitetura ideal. Entretanto, não só o design de do projeto é necessário para que o mesmo possua boa manutenibilidade, mas também é importante que o código-fonte sigam os princípios do SOLID, o projeto gerado pelo MLG não modificado, assim como no exemplo em JavaScript, não segue o DIP e o ISP. As interfaces geradas possuem não possuem um conjunto de funcionalidades granulares e não possuíam clientes que as utilizavam.

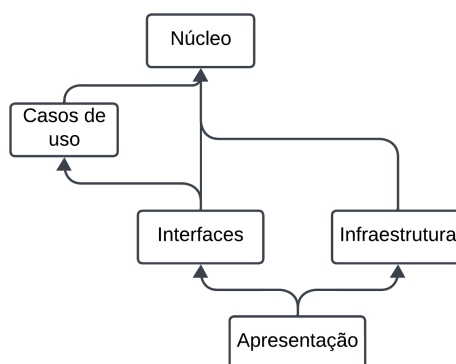


Figura 18 – Grafo de dependência dos módulos do exemplo em Python com FastApi. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.

Módulo	Número de classes abstratas	Número de classes concretas	Valor de abstração
Núcleo	26	4	0.87
Infraestrutura	0	10	0
Aplicação	0	6	0
Casos de uso	0	20	0

Tabela 13 – Número de classes abstratas e concretas, assim como o valor de abstração de cada módulo no exemplo em Python com FastApi após alterações sugeridas. Contagem exclui classes com código gerado com defeitos. Valores aproximados até a segunda casa decimal.

Módulo	Número de dependências	Número de dependentes	Valor de instabilidade
Núcleo	0	3	0
Infraestrutura	1	0	1
Aplicação	1	0	1
Casos de uso	1	0	1

Tabela 14 – Número de dependências e módulos dependentes, assim como o valor de instabilidade de cada módulo no exemplo em Python com FastApi após alterações sugeridas. Valores aproximados até a segunda casa decimal.

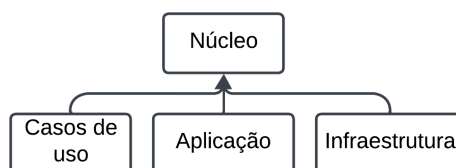


Figura 19 – Grafo de dependência dos módulos do exemplo em Python com FastApi após alterações sugeridas. As setas indicam a relação de dependência entre dois módulos, a direção das setas saem do módulo dependente e vão até o módulo alvo. Figura elaborada pelo autor.

Módulo	Distância da sequência principal
Núcleo	-0.13
Infraestrutura	0
Aplicação	0
Casos de uso	0
Média	-0.03
Desvio-padrão	0.06

Tabela 15 – Distância da sequência principal de cada módulo no exemplo em Python com FastApi após alterações sugeridas, utilizando a equação 4.1, média e desvio-padrão dos valores. Valores aproximados até a segunda casa decimal.

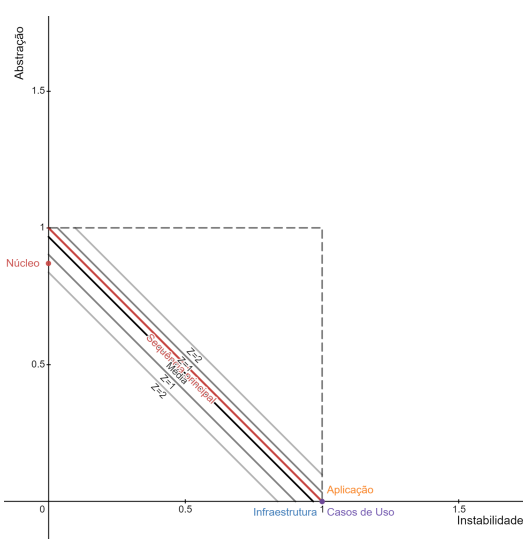


Figura 20 – Distância de cada módulo do exemplo em Python em relação à sequência principal após alterações sugeridas, que está destacada em vermelho. As linhas Média, $Z = 1$ e $Z = 2$ são, respectivamente, iguais à média da distância da sequência principal, e a média mais ou menos 1 e 2 desvios-padrão. Figura elaborada pelo autor.

4.4 Resultados gerais

Os resultados obtidos possuem características em comum que, apesar de não possuir relação com a arquitetura de *software*, impactam o projeto final em termos de manutenibilidade, como documentação, configurações globais, e utilização de containerização, uma técnica de virtualização que encapsula processos de sistema operacional juntamente algum programa em específico a fim de facilitar a escalabilidade dos sistemas (WATADA et al., 2019).

Nas etapas descritas na seção 3.3, é mencionado o desenvolvimento de um arquivo com a descrição do propósito de cada módulo com a finalidade de iniciar uma documentação de cada componente do sistema a partir da descrição de alto nível provida

pelo usuário. Para todos os módulos, a documentação gerada o padrão de descrição do propósito do módulo, estrutura do mesmo e, ocasionalmente, uma breve explicação de como melhor utilizar o componente, servindo como um início de documentação do sistema que pode ser utilizada como meio de aprendizagem para novos desenvolvedores. Manter um arquivo de documentação por módulo faz com que mudanças nas regras de negócio sejam mais fáceis de serem documentadas, é necessário atualizar a documentação apenas dos locais foram impactados pelas atualizações.

Em todos as soluções propostas, o MLG adotou formas flexíveis de armazenar configurações globais dos projetos, como endereço para o banco de dados, tempo padrão de duração de aluguéis, entre outras. Essa prática possibilita fácil alteração de determinados comportamentos dos projetos sem a necessidade de uma nova compilação do código para que as mudanças sejam implantadas, facilitando a manutenção.

Os projetos em C# e Python foram gerados com arquivos de configuração da ferramenta Docker, uma plataforma de containerização. A inclusão desses arquivos não torna obrigatório o uso da ferramenta, mas introduz a possibilidade de facilitar o processo de implantação e execução dos aplicativos. Ao utilizar o Docker, os desenvolvedores podem encapsular suas aplicações juntamente com suas dependências, garantindo consistência e isolamento (WATADA et al., 2019). Assim, a inclusão de arquivos de configuração Docker nos projetos em C# e Python oferece uma abordagem flexível e eficiente para a implementação e distribuição de software, destacando-se como uma opção valiosa para desenvolvedores que buscam maior portabilidade e confiabilidade em suas soluções.

4.5 Comparação

Dentre os resultados obtidos, é possível averiguar na figura 21 que o projeto que se sobressai em termos de adequação aos princípios da arquitetura limpa é o projeto em C#, já que seus componentes estão notoriamente mais próximos da sequência principal assim como sua média e linhas de desvio padrão, mostrando que o desempenho do MLG utilizando a abordagem desta pesquisa sem auxílio humano adicional é maior para a linguagem em comparação às outras.

Entretanto, ao aplicar as modificações nos projetos de JavaScript e Python, é possível notar que o projeto em C# deixa de ser o que melhor se adequa à arquitetura limpa, tendo um desempenho similar ao projeto em JavaScript revisado em relação às métricas propostas. O projeto em Python após as revisões se torna o projeto mais alinhado aos princípios da arquitetura limpa.

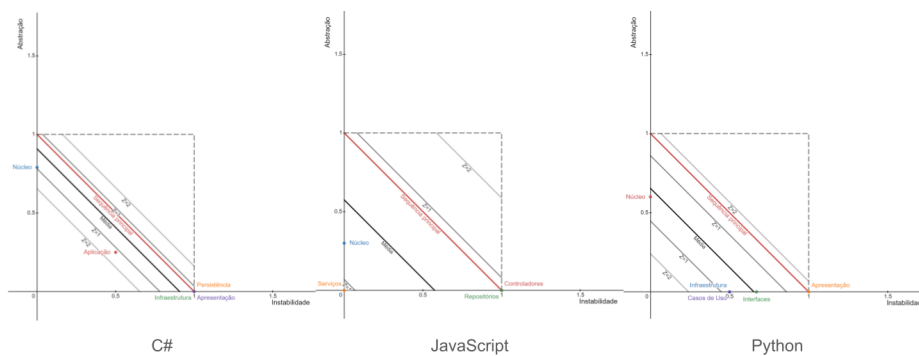


Figura 21 – Gráficos da distância da sequência principal de cada módulo dos projetos antes das modificações lado a lado.

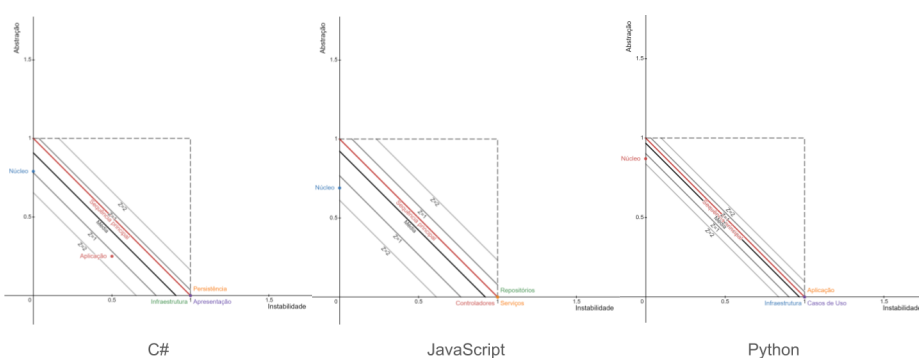


Figura 22 – Gráficos da distância da sequência principal de cada módulo dos projetos após modificações lado a lado.

5 Conclusão e trabalhos futuros

Neste trabalho discutimos como as *startups* caracterizam-se por operar em contextos de alta incerteza e pressão por inovação, o que frequentemente leva à adoção de práticas como o desenvolvimento ágil de software. Observou-se, entretanto, que práticas essenciais como testes unitários e documentação acabam sendo negligenciadas, impactando negativamente a manutenção e escala dos produtos digitais. Nesse sentido, propusemos a utilização do GPT-4 como ferramenta para a síntese automática de código, aderindo aos princípios da arquitetura limpa, com o objetivo de otimizar o tempo de desenvolvimento enquanto se mantém aderência a boas práticas de arquitetura de software. A metodologia aplicada visou à exploração qualitativa e quantitativa dos artefatos gerados, utilizando como base os princípios consolidados na literatura, e abordou tanto a investigação quanto a adaptação de *prompts* para a geração de código, prototipação e desenvolvimento de exemplos práticos em linguagens de programação diversificadas.

A aplicação do GPT-4 para a síntese de código alinhado aos princípios da arquitetura limpa revelou inconsistências nos resultados obtidos. Dos três projetos gerados, dois apresentaram deficiências em seu *design*, que embora fossem sanáveis nos exemplos tratados, poderiam representar desafios significativos em contextos mais complexos. Problemas arquiteturais foram observados, além de falhas no próprio desenvolvimento do código, onde a ferramenta de IA não conseguiu gerar código executável conforme a sintaxe esperada das linguagens de programação utilizadas, ou falhou em desenvolver integralmente as funcionalidades desejadas.

Apesar dos desafios identificados, para projetos de menor complexidade e em estágios iniciais, o fluxo de desenvolvimento proposto mostrou-se promissor, gerando códigos que se aproximam dos ideais de acordo com as métricas estabelecidas, porém, ainda requerem intervenção humana para ajustes finais. Tal intervenção visa otimizar ainda mais o tempo de desenvolvimento e a qualidade da documentação do projeto.

Como direcionamento para trabalhos futuros, sugere-se o uso de MLGs especializados em geração de código, tais como CodeRL (LE et al., 2022), buscando melhorar

a qualidade tanto sintática quanto semântica do código gerado e evitar os declínios observados no desempenho do GPT-4 em termos de geração de código, adesão a limitações, formatação de texto entre outros (CHEN; ZAHARIA; ZOU, 2023) e a limitação de conhecimento desse modelo, já que foi treinado com dados providos da internet até dezembro de 2023¹ e não possuirá conhecimento de novas linguagens, bibliotecas e *frameworks* posteriores a essa data (OPENAI, 2023). Além disso, recomenda-se a revisão e o refinamento do processo de geração de projetos descrito neste trabalho, ampliando os experimentos para abarcar uma gama mais diversificada de linguagens de programação e *frameworks*. A investigação da eficácia dos MLGs em contextos que envolvem linguagens dinamicamente tipadas em comparação às estaticamente tipadas também se mostra uma linha de pesquisa promissora, dadas as nuances na aplicação dos princípios da arquitetura limpa em tais contextos descritas por Martin (2019).

¹<<https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>>

Referências

- AL-RFOU, R. et al. Character-level language modeling with deeper self-attention. 8 2018.
- BATISTA, B. C.; MURA, W. A. D. Geração automática de apis rest a partir de um modelo aberto de descrição de serviços. In: . Sociedade Brasileira de Computação - SBC, 2022. p. 69–78. Disponível em: <<https://sol.sbc.org.br/index.php/latinoware/article/view/22970>>.
- BERG, V. et al. Achieving agility and quality in product development - an empirical study of hardware startups. *Journal of Systems and Software*, v. 167, p. 110599, 9 2020. ISSN 01641212.
- CALEGARIO, F. et al. Exploring the intersection of generative ai and software development. 12 2023.
- CALLANAN, R.; FUSCO, P.; SANTAMARIA, C. The graphic truth: Chatgpt breaks the internet. *GZERO*, 2023. Matéria sobre o crescimento de popularidade do ChatGPT. Disponível em: <<https://www.gzeromedia.com/living-beyond-borders-articles/the-graphic-truth-chatgpt-breaks-the-internet>>. Acesso em: 6 de janeiro de 2024.
- CHEN, L.; ZAHARIA, M.; ZOU, J. How is chatgpt's behavior changing over time? 7 2023.
- CICO, O. Lean software startup practices and software engineering education. In: *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. [S.l.: s.n.], 2022. p. 281–285.
- CICO, O.; DUC, A. N.; JACCHERI, L. An empirical investigation on software practices in growth phase startups. In: *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (EASE '20), p. 282–287. ISBN 9781450377317. Disponível em: <<https://doi.org/10.1145/3383219.3383249>>.
- CRNKOVIC, I.; LARSSON, M. *Building Reliable Component-Based Software Systems*. [S.l.: s.n.], 2002.
- DAI, Z. et al. Transformer-xl: Attentive language models beyond a fixed-length context. 1 2019.
- DUARTE, F. Number of chatgpt users (mon 2024). *Exploding Topics*, 2024. Matéria quantidade de usuários do ChatGPT-4. Disponível em: <<https://explodingtopics.com/blog/chatgpt-users>>. Acesso em: 6 de janeiro de 2024.

FREECODECAMP.ORG. *Use ChatGPT to Code a Full Stack App – Full Course*. 2023. Workshop online sobre uso do ChatGPT para desenvolvimento de sistemas Web. Disponível em: <<https://www.youtube.com/watch?v=GizsSo-EevA>>. Acesso em: 25 de dezembro de 2023.

FRIEDER, S. et al. Mathematical capabilities of chatgpt. 1 2023.

GEWIRTZ, D. How to use chatgpt to write code. *ZDNET*, 2023. Materia sobre como utilizar o ChatGPT para geração de código. Disponível em: <<https://www.zdnet.com/article/how-to-use-chatgpt-to-write-code/>>. Acesso em: 6 de janeiro de 2024.

GIL, A. *Como elaborar projetos de pesquisa*. Atlas, 1991. ISBN 9788522407248. Disponível em: <<https://books.google.com.br/books?id=m5VDtwAACAAJ>>.

GUPTA, M.; GEORGE, J. F.; XIA, W. Relationships between it department culture and agile software development practices: An empirical investigation. *International Journal of Information Management*, v. 44, p. 13–24, 2 2019. ISSN 02684012.

LE, H. et al. *CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning*. 2022.

LISKOV, B. Keynote address - data abstraction and hierarchy. In: *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages and Applications (Addendum)*. New York, NY, USA: Association for Computing Machinery, 1987. (OOPSLA '87), p. 17–34. ISBN 0897912667. Disponível em: <<https://doi.org/10.1145/62138.62141>>.

LU, S. et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. 2 2021.

MARTIN, R. *O Codificador Limpo: Um código de conduta para programadores profissionais*. 1. ed. [S.l.]: Alta Books, 2012. ISBN 978-8576086475.

MARTIN, R. *Arquitetura limpa: O guia do artesão para estrutura e design de software*. 1. ed. [S.l.]: Alta Books, 2019. ISBN 978-8550804606.

MEYER, B. *Object-Oriented Software Construction*. 2. ed. Upper Saddle River, NJ: Prentice Hall, 1997. ISBN 978-0-13-629155-8.

NGUYEN-DUC, A.; KEMELL, K.-K.; ABRAHAMSSON, P. The entrepreneurial logic of startup software development: A study of 40 software startups. *Empirical Software Engineering*, v. 26, p. 91, 9 2021. ISSN 1382-3256. Disponível em: <<https://link.springer.com/10.1007/s10664-021-09987-z>>.

OPENAI. Gpt-4 technical report. 3 2023.

SMULIAN, P. Chatgpt is breaking the internet, but what does it mean for your business? *Clever Clicks*, 2023. Materia sobre o ChatGPT. Disponível em: <<https://cleverclicksdigital.com/blog/chatgpt-is-breaking-the-internet-but-what-does-it-mean-for-your-business/>>. Acesso em: 6 de janeiro de 2024.

SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*. 2nd. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201745720.

WATADA, J. et al. Emerging trends, techniques and open issues of containerization: A review. *IEEE Access*, v. 7, p. 152443–152472, 2019.

WEHRMEISTER, M. Geração de código para robôs implementados com ros a partir de modelos uml/marte. In: . [S.l.]: Sociedade Brasileira de Computação - SBC, 2019. p. 39–46.

WHITE, J. et al. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. 3 2023.

Apêndices

A Prompts utilizados

1. *Translate the following message to <target language>.* **Tradução:** Traduza a seguinte mensagem para <língua desejada>.
2. *Within a backend project scope, consider the following project description and point out any areas of ambiguity or potentially unintended outcomes. Also provide a revised version that solves the ambiguities. If there are no ambiguity points, respond with "No changes needed"..* **Tradução:** Dentro do escopo de um projeto de backend, considere a seguinte descrição de projeto e destaque quaisquer áreas de ambiguidade ou resultados potencialmente não intencionados. Além disso, forneça uma versão revisada que resolva as ambiguidades. Se não houver pontos de ambiguidade, responda com "Nenhuma mudança necessária".
3. *In the context of the following project description, the system must adhere to Robert C. Martin's clean architecture principles, with a "Core" module that will contain the project entities and interfaces for the use cases. Keep in mind to use dependency inversion to avoid higher abstraction level modules to depend on lower abstraction level modules. List all the system components that will be present in the final version of the project ordered by dependency count (less dependent modules first), include a "Root" where the root level files will be if needed. Write the names as they will be in the project folders. Respond with only the module names as a JSON array without markdown formatting and without comments or explanations about your choices. Project description: <project description>.* **Tradução:** No contexto da seguinte descrição de projeto, o sistema deve seguir os princípios da arquitetura limpa de Robert C. Martin, com um módulo "Core" que conterá as entidades e interfaces do projeto para os casos de uso. Lembre de utilizar inversão de dependência para evitar que módulos com maior nível de abstração dependam de módulos com menor nível de abstração. Liste todos os componentes que estarão presentes na versão final do sistema ordenados por quantidade de dependências (módulos menos dependentes primeiro), inclua um "Root" onde os arquivos de nível raiz irão ficar se necessário. Escreva apenas os nomes como eles serão nas pastas do projeto. Responda apenas com nomes dos módulos como um *array JSON* sem formatação *Markdown* e sem comen-

tários ou explicações das suas escolhas. Descrição do projeto: <descrição do projeto>.

4. *You're now a senior developer that received this project description and modules and is starting to create the application. List all the files that need to be created for the component <module name>, including a README file, as a JSON array without markdown formatting. Return raw json only without comments or any explanation about your choices. For files that should be inside a directory include the directory name in the file name, separating them with a /. Do not include the module directory in the file name. If it is the Root module, don't include files for other modules..* **Tradução:** Você agora é um desenvolvedor sênior que recebeu esta descrição de projeto e módulos e está começando a criar a aplicação. Liste todos os arquivos do componente <nome do componente>, incluindo um *README*, como um *array JSON* sem formatação *Markdown*. Retorne um *JSON* cru sem comentários ou explicações de suas escolhas. Para arquivos que devem ficar dentro de pastas, inclua o nome do diretório no nome do arquivo, separando-os com uma /. Não inclua a pasta do módulo no nome do arquivo. Se for o módulo "Root", não inclua arquivos de outros módulos.
5. *Now develop the <file name> file for the module <module name>. Your responses should be raw code text only, without explanation and without markdown formatting wrapping the code.* **Tradução:** Agora desenvolva o arquivo <nome do arquivo> para o módulo <nome do módulo>. Seu retorno deve ser apenas texto de código cru, sem nenhuma explicação e formatação *Markdown* ao redor do código.

B Descrição do projeto de teste

O projeto de *backend* para a biblioteca virtual, a ser desenvolvido em <linguagem do projeto>, utilizando o *framework* <*framework* do projeto> e o Microsoft SQL Server como banco de dados, realizará o registro e o gerenciamento de livros e estudantes. Os dados dos alunos consistirão em detalhes pessoais e histórico de aluguel, enquanto para os livros incluirão título, autor, ISBN, status etc. Apenas estudantes registrados podem alugar um livro desde que esteja disponível. Uma vez que o livro é alugado, seu status no sistema é atualizado para "Alugado".

A devolução de cada livro será manualmente confirmada por um bibliotecário ou administrador, que pode então atualizar o status do livro para "Disponível", "Reservado" ou "Em manutenção" com base na condição física do livro e nas futuras reservas.

Para gerenciar efetivamente os períodos de empréstimo, os administradores do sistema podem definir durações de empréstimo padrão. Este período de empréstimo padrão se aplica a todos os livros, sem exceções.

Os bibliotecários serão registrados no sistema e podem ser gerenciados (CRUD) por administradores. Um usuário administrador tem amplos direitos de sistema, incluindo gerenciamento de todas as entidades do sistema e alteração de configurações do sistema. Os bibliotecários têm direitos mais limitados, focados principalmente na realização de operações CRUD para os dados de livro e aluno.

Os usuários administradores serão inicialmente pré-carregados no banco de dados com suas informações, que incluem credenciais. Essas credenciais e configurações gerais do sistema serão preenchidos a partir de um arquivo de configuração externo, que inclui configurações como a duração padrão do empréstimo do livro.

O sistema diferenciará entre entidades - estudantes e livros - e usuários do sistema - bibliotecários e administradores. Os primeiros podem ser gerenciados mas não têm direitos de usuário do sistema, enquanto os últimos têm os direitos respectivos conforme definido em suas funções.

C Processo de criação dos exemplos

1. **Processo de criação do exemplo em C# com .NET Core 5: *OpenAI API Key*:**
<chave da API>

Your lanuguage (program will follow the language you choose): português Insira a descrição do seu projeto:O projeto de *backend* para a biblioteca virtual, a ser desenvolvido em C#, utilizando o *framework* .NET Core 5 e o Microsoft SQL Server como banco de dados, realizará o registro e o gerenciamento de livros e estudantes. Os dados dos alunos consistirão em detalhes pessoais e histórico de aluguel, enquanto para os livros incluirão título, autor, ISBN, status etc. Apenas estudantes registrados podem alugar um livro desde que esteja disponível. Uma vez que o livro é alugado, seu status no sistema é atualizado para "Alugado". A devolução de cada livro será manualmente confirmada por um bibliotecário ou administrador, que pode então atualizar o status do livro para "Disponível", "Reservado" ou "Em manutenção" com base na condição física do livro e nas futuras reservas. Para gerenciar efetivamente os períodos de empréstimo, os administradores do sistema podem definir durações de empréstimo padrão. Este período de empréstimo padrão se aplica a todos os livros, sem exceções. Os bibliotecários serão registrados no sistema e podem ser gerenciados (CRUD) por administradores. Um usuário administrador tem amplos direitos de sistema, incluindo gerenciamento de todas as entidades do sistema e alteração de configurações do sistema. Os bibliotecários têm direitos mais limitados, focados principalmente na realização de operações CRUD para os dados de livro e aluno. Os usuários administradores serão inicialmente pré-carregados no banco de dados com suas informações, que incluem credenciais. Essas credenciais e configurações gerais do sistema serão preenchidos a partir de um arquivo de configuração externo, que inclui configurações como a duração padrão do empréstimo do livro. O sistema diferenciará entre entidades - estudantes e livros - e usuários do sistema - bibliotecários e administradores. Os primeiros podem ser gerenciados mas não têm direitos de usuário do sistema, enquanto os últimos têm os direitos respectivos conforme definido em suas funções.

A descrição do projeto fornecida acima é geralmente clara, mas possui algumas

áreas de ambiguidade que poderiam levar a resultados potencialmente não intencionados. Abaixo estão as ambiguidades identificadas e uma versão revisada que as aborda:

Ambiguidades:

1. "Detalhes pessoais" nos dados dos estudantes é ambíguo. É necessário especificar quais detalhes pessoais devem ser coletados e armazenados. 2. A frase "etc." para os detalhes do livro é vaga; deixa o escopo das informações do livro a ser gerenciado em aberto. 3. A afirmação "este período padrão de empréstimo se aplica a todos os livros, sem exceção" não aborda se existem casos especiais como livros de referência que não podem ser retirados da biblioteca ou períodos de empréstimo prolongados para educadores ou pesquisadores. 4. A descrição não esclarece o processo de registro de bibliotecários ou as credenciais necessárias. 5. A descrição não especifica as ações a serem tomadas caso um livro não seja devolvido dentro do período padrão de empréstimo.

Descrição Revisada do Projeto:

O projeto *back-end* para a biblioteca virtual, a ser desenvolvido em C#, usando o framework .NET Core 5, e fazendo uso do Microsoft SQL Server para operações de banco de dados, lidará com o registro e a gestão de livros e estudantes.

- ****Dados do Estudante****: Os dados de cada estudante incluirão nome completo, número de identificação de estudante, informações de contato (*email* e telefone) e histórico de aluguel. - ****Dados do Livro****: As informações para cada livro incluirão título, autor, ISBN, status atual (por exemplo, Disponível, Alugado, Reservado, Em Manutenção) e outros detalhes relevantes como ano de publicação, gênero e edição.

Regras de Empréstimo: - Apenas estudantes registrados podem alugar livros, e isso é condicional ao livro estar marcado como "Disponível". - Quando um livro é alugado, seu status é imediatamente atualizado para "Alugado". - Ao regressar, o status de um livro é ajustado por um bibliotecário ou administrador para refletir sua disponibilidade ou condição ("Disponível", "Reservado" ou "Em Manutenção").

- ****Gestão do Período de Empréstimo****: Os administradores podem definir durações padrão de empréstimo através das configurações do sistema. Embora o período padrão de empréstimo seja aplicado globalmente, deve-se disponibilizar a provisão para marcar itens específicos como materiais de referência como "Não para Empréstimo". Além disso, o sistema deve suportar exceções configuradas para durações prolongadas de empréstimo em casos especiais (como educadores ou pesquisadores), conforme determinado pelas políticas da biblioteca.

- ****Funções de Usuário e Gestão****: - Bibliotecários e administradores são entidades de usuário distintas dentro do sistema. Todos são registrados com credenciais únicas. - Os administradores desfrutam de direitos de sistema abrangentes, incluindo a gestão de todas as entidades (estudantes, livros, bibliotecários) e modificação de configurações em todo o sistema. - Bibliotecários possuem direitos mais limitados, principalmente limitados a operações CRUD para livros e dados de estudantes. - Contas iniciais de usuário administrativo serão pré-carregadas no sistema, com credenciais e outras configurações essenciais do sistema provenientes de um arquivo de configuração externo.

- ****Livros em Atraso****: O sistema deve ter um protocolo para livros em atraso, que aciona lembretes aos estudantes e alertas aos bibliotecários.

O sistema diferenciará entre entidades (estudantes, livros) e usuários do sistema (bibliotecários, administradores). Entidades podem ser geridas dentro do sistema, mas não possuem direitos de usuário do sistema. Os usuários têm direitos de acordo com suas funções, conforme definido na configuração do sistema.

Esta descrição revisada visa eliminar ambiguidades, definindo pontos de dados específicos para informações de estudantes, detalhando os dados necessários do livro, esclarecendo regras e gestão de empréstimos e especificando as funções e direitos dos usuários dentro do sistema.

Digite a descrição do seu novo projeto ou pressione enter para continuar: <enter pressionado>

Projeto gerado com sucesso!

2. **Processo de criação do exemplo em JavaScript com NodeJS: OpenAI API** Key: <chave da api>

Your language (program will follow the language you choose): português Insira a descrição do seu projeto: O projeto de *backend* para a biblioteca virtual, a ser desenvolvido em javascript, utilizando o *framework* NodeJS e o Microsoft SQL Server como banco de dados, realizará o registro e o gerenciamento de livros e estudantes. Os dados dos alunos consistirão em detalhes pessoais e histórico de aluguel, enquanto para os livros incluirão título, autor, ISBN, status etc. Apenas estudantes registrados podem alugar um livro desde que esteja disponível. Uma vez que o livro é alugado, seu status no sistema é atualizado para "Alugado". A devolução de cada livro será manualmente confirmada por um bibliotecário ou administrador, que pode então atualizar o status do livro para "Disponível", "Reservado" ou "Em manutenção" com base na condição física do livro e nas futuras reservas. Para gerenciar efetivamente os períodos de empréstimo, os administradores do sistema podem definir durações de empréstimo padrão. Este período

de empréstimo padrão se aplica a todos os livros, sem exceções. Os bibliotecários serão registrados no sistema e podem ser gerenciados (CRUD) por administradores. Um usuário administrador tem amplos direitos de sistema, incluindo gerenciamento de todas as entidades do sistema e alteração de configurações do sistema. Os bibliotecários têm direitos mais limitados, focados principalmente na realização de operações CRUD para os dados de livro e aluno. Os usuários administradores serão inicialmente pré-carregados no banco de dados com suas informações, que incluem credenciais. Essas credenciais e configurações gerais do sistema serão preenchidos a partir de um arquivo de configuração externo, que inclui configurações como a duração padrão do empréstimo do livro. O sistema diferenciará entre entidades - estudantes e livros - e usuários do sistema - bibliotecários e administradores. Os primeiros podem ser gerenciados mas não têm direitos de usuário do sistema, enquanto os últimos têm os direitos respectivos conforme definido em suas funções.

Existem algumas áreas de ambiguidade e potencial para resultados não intencionados na descrição do projeto fornecida:

1. "Os dados dos alunos consistirão de detalhes pessoais e histórico de aluguel" é ambíguo. Os detalhes pessoais específicos a serem incluídos não são delineados, o que pode levar a preocupações com a privacidade dos dados ou a um modelo de dados inadequado.
2. "Os livros incluirão título, autor, ISBN, status, etc." – O uso de "etc." é ambíguo porque não especifica todos os campos que precisam ser incluídos para um livro, o que pode levar a inconsistências.
3. "Uma vez que o livro é alugado, seu status no sistema é atualizado para 'Alugado'." – Não está claro quem ou o que é responsável por atualizar o status do livro. Pode ser necessário ter um sistema automático no lugar ou um procedimento claro para os bibliotecários.
4. "A devolução de cada livro será manualmente confirmada por um bibliotecário ou administrador" – Isso pode ser um gargalo e sujeito a erros humanos. Além disso, não é indicado o que acontece em caso de disputas ou discrepâncias.
5. "Este período de empréstimo padrão aplica-se a todos os livros, sem exceções." – A falta de exceções pode ser problemática para casos especiais, como livros de referência ou livros raros que exigem um período de empréstimo diferente.
6. "Um usuário administrativo tem amplos direitos de sistema..." – Não está claro como esses direitos são diferenciados daqueles dos bibliotecários além das operações CRUD, especialmente em cenários de casos-limite.

7. "Essas credenciais e configurações gerais do sistema serão preenchidas a partir de um arquivo de configuração externo"— Isso introduz potenciais riscos de segurança, e o protocolo para manuseio seguro de credenciais e configurações não é especificado.

8. "O sistema diferenciará entre entidades"— Embora a diferenciação seja mencionada, as permissões e capacidades exatas para cada função de usuário não são definidas minuciosamente.

Descrição Revisada do Projeto:

O projeto *back-end* para a biblioteca virtual será desenvolvido em JavaScript usando a estrutura Node.js, com o Microsoft SQL Server como banco de dados. O sistema lidará com o registro e gerenciamento de livros e alunos. Armazenará os seguintes dados específicos para estudantes: nome, informações para contato e histórico de aluguel. Cada registro de livro incluirá o título, autor, ISBN e status, juntamente com metadados adicionais relevantes conforme decidido pelo bibliotecário.

Quando um aluno, que deve estar registrado no sistema, alugar um livro disponível, o sistema atualizará automaticamente o status do livro para "Alugado". Ao devolver o livro, um bibliotecário ou administrador inspecionará a condição do livro e atualizará manualmente seu status para "Disponível", "Reservado" ou "Manutenção", com notas especificando a condição e quaisquer ações necessárias a serem tomadas.

Os administradores do sistema podem definir durações de empréstimo padrão, que por padrão se aplicam a todos os livros. No entanto, uma provisão será incluída para permitir que os administradores atribuam durações específicas de empréstimo para certas categorias de livros, se necessário.

Bibliotecários e administradores serão duas funções de usuário distintas no sistema. Os administradores podem gerenciar todos os aspectos do sistema, incluindo gerenciamento de contas de usuário para ambos, bibliotecários e administradores, e ajustar configurações do sistema como durações de empréstimo. Contas de usuário administrativo serão inicialmente criadas com base nos dados fornecidos em um arquivo de configuração externo seguro. Os bibliotecários serão principalmente responsáveis pelas operações CRUD relacionadas a livros e dados de alunos.

Um robusto sistema de controle de acesso baseado em função (RBAC) será implementado para garantir que permissões e capacidades sejam claramente definidas e aplicadas para cada função de usuário. Todas as interações com dados sensíveis, como credenciais, aderirão a práticas de segurança padrão da indústria para mitigar riscos de segurança.

Digite a descrição do seu novo projeto ou pressione enter para continuar: <enter pressionado>

Projeto gerado com sucesso!

3. **Processo de criação do exemplo em Python com FastApi:** *OpenAI API Key:* <chave da api>

Your lanuguage (program will follow the language you choose): português Insira a descrição do seu projeto: O projeto de backend para a biblioteca virtual, a ser desenvolvido em python, utilizando o framework FastApi e o Microsoft SQL Server como banco de dados, realizará o registro e o gerenciamento de livros e estudantes. Os dados dos alunos consistirão em detalhes pessoais e histórico de aluguel, enquanto para os livros incluirão título, autor, ISBN, status etc. Apenas estudantes registrados podem alugar um livro desde que esteja disponível. Uma vez que o livro é alugado, seu status no sistema é atualizado para "Alugado". A devolução de cada livro será manualmente confirmada por um bibliotecário ou administrador, que pode então atualizar o status do livro para "Disponível", "Reservado" ou "Em manutenção" com base na condição física do livro e nas futuras reservas. Para gerenciar efetivamente os períodos de empréstimo, os administradores do sistema podem definir durações de empréstimo padrão. Este período de empréstimo padrão se aplica a todos os livros, sem exceções. Os bibliotecários serão registrados no sistema e podem ser gerenciados (CRUD) por administradores. Um usuário administrador tem amplos direitos de sistema, incluindo gerenciamento de todas as entidades do sistema e alteração de configurações do sistema. Os bibliotecários têm direitos mais limitados, focados principalmente na realização de operações CRUD para os dados de livro e aluno. Os usuários administradores serão inicialmente pré-carregados no banco de dados com suas informações, que incluem credenciais. Essas credenciais e configurações gerais do sistema serão preenchidos a partir de um arquivo de configuração externo, que inclui configurações como a duração padrão do empréstimo do livro. O sistema diferenciará entre entidades - estudantes e livros - e usuários do sistema - bibliotecários e administradores. Os primeiros podem ser gerenciados mas não têm direitos de usuário do sistema, enquanto os últimos têm os direitos respectivos conforme definido em suas funções.

Existem algumas áreas de ambiguidade e possíveis resultados não intencionais na descrição fornecida do projeto:

1. "Os dados dos estudantes consistirão de detalhes pessoais e histórico de aluguel- A ambiguidade reside no que constitui 'detalhes pessoais'. É importante aderir às regulamentações de proteção de dados (como o GDPR), portanto, o

escopo e a natureza dos detalhes pessoais coletados precisam ser claramente definidos.

2. "Os dados dos livros incluirão título, autor, ISBN, status, etc.- O termo 'etc.' é vago. Seria benéfico listar explicitamente todos os atributos de dados de livros necessários para o sistema.

3. "Uma vez que um livro é alugado, seu status no sistema é atualizado para 'Alugado'.- Não há menção de como um livro 'Alugado' é associado a um estudante. Deve haver uma clara ligação entre os livros alugados e os estudantes que os alugaram.

4. "Para gerenciar efetivamente os períodos de empréstimo, os administradores do sistema podem definir durações padrão de empréstimo. Esse período de empréstimo padrão se aplica a todos os livros, sem exceção.- Isso pode levar à inflexibilidade, já que alguns livros podem exigir diferentes durações de empréstimo devido à alta demanda ou categorização especial.

5. "Essas credenciais e configurações gerais do sistema serão preenchidas a partir de um arquivo de configuração externo- Não está claro quão seguro é esse processo. Armazenar credenciais em um arquivo externo pode ser um risco significativo de segurança se o arquivo não for adequadamente protegido.

6. "O sistema diferenciará entre entidades - estudantes e livros - e usuários do sistema - bibliotecários e administradores.- Esta afirmação é clara em sua intenção de diferenciar papéis, mas não especifica como os controles de acesso serão implementados ou aplicados.

Descrição Revista do Projeto:

O projeto de backend para a biblioteca virtual, a ser desenvolvido em Python, utilizando o framework FastAPI e aproveitando o Microsoft SQL Server como banco de dados, facilitará as seguintes funções:

- Cadastro e gerenciamento de livros, com cada registro de livro contendo um título, autor, número único de ISBN e status atual (por exemplo, Disponível, Alugado, Reservado, Em Manutenção).
- Cadastro e gerenciamento de estudantes, incluindo detalhes pessoais armazenados com segurança (como nome, número de contato e endereço de e-mail) e um histórico de aluguel acessível para cada estudante.
- Um sistema de papéis de usuário, com estudantes podendo registrar-se e visualizar históricos pessoais de aluguel, bibliotecários capacitados para realizar operações CRUD em livros e registros de estudantes, e administradores possuindo a habilidade de gerenciar configurações do sistema e todos os usuários dentro do sistema.
- Funcionalidade do sistema de aluguel que permite apenas a estudantes registrados alugar livros disponíveis e rastreia qual estu-

dante alugou cada livro em particular. Ao alugar um livro, o sistema atualizará automaticamente o status do livro para 'Alugado' e o vinculará ao registro do estudante que o alugou. - Uma interface para bibliotecários ou administradores confirmarem manualmente a devolução dos livros, com opção de definir o status do livro para 'Disponível', 'Reservado' ou 'Em Manutenção', dependendo da condição física e da fila de reserva do livro. - Capacidade para administradores do sistema definirem um período de empréstimo padrão para todos os livros por meio de uma configuração de sistema segura. No entanto, os administradores também devem ter a flexibilidade de ajustar a duração do empréstimo em uma base por livro, quando necessário. - Autenticação segura para bibliotecários e administradores com detalhes iniciais da conta de usuário carregados a partir de um arquivo de configuração criptografado e controlado por acesso. Este arquivo também conterá outras configurações de sistema seguras, como durações padrão de empréstimo de livros. - Um sistema de controle de acesso bem definido, com permissões claras para cada papel de usuário, garantindo que estudantes, livros, bibliotecários e administradores tenham níveis apropriados de acesso ao sistema com base em suas funções e responsabilidades.

Ao implementar essas especificações, o sistema de backend para a biblioteca virtual aprimorará suas capacidades de gerenciamento de usuários, eficiências no manuseio de livros e medidas de segurança, oferecendo flexibilidade nas durações de empréstimo.

Digite a descrição do seu novo projeto ou pressione enter para continuar: <enter pressionado>

Projeto gerado com sucesso!