Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Microfrontends e Arquitetura Limpa: Um Estudo Exploratório Orientado a Provas de Conceito

Autor: Paulo Víctor da Silva
Orientadora: Profª. Drª Milene Serrano
Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF
2023

Paulo Víctor da Silva

Microfrontends e Arquitetura Limpa: Um Estudo Exploratório Orientado a Provas de Conceito

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientadora: Profª. Drª Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2023

Paulo Víctor da Silva

Microfrontends e Arquitetura Limpa: Um Estudo Exploratório Orientado a Provas de Conceito/ Paulo Víctor da Silva. – Brasília, DF, 2023-

166 p. : il. (algumas color.) ; 30 cm.

Orientadora: Profª. Drª Milene Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Arquitetura Limpa. 2. *Micro Front-end*. I. Profª. Drª Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Microfrontends e Arquitetura Limpa: Um Estudo Exploratório Orientado a Provas de Conceito

CDU 02:141:005.6

Paulo Víctor da Silva

Microfrontends e Arquitetura Limpa: Um Estudo Exploratório Orientado a Provas de Conceito

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 22 de Dezembro de 2023:

Profª. Drª Milene Serrano
Orientadora

Prof. Dr. Maurício Serrano
Coorientador

Prof. Dr. Fernando William Cruz
Convidado

Prof. Dr. John Lenon Cardoso Gardenghi
Convidado

Brasília, DF
2023

Agradecimentos

Agradeço, em primeiro lugar, a Deus e à Santíssima Virgem Maria, que me concederam a graça de estar concluindo mais uma etapa na minha vida. Agradeço também à minha esposa, que, no início da graduação, ainda era apenas minha namorada e esteve ao meu lado desde então, me apoiando, aconselhando, intercedendo e auxiliando em tudo o que foi necessário para que eu pudesse concluir essa graduação. Agradeço à minha família, especialmente à minha mãe e meu irmão, que sempre me incentivaram a alcançar meus sonhos, e que, mesmo com todas as dificuldades, me ajudaram a chegar até aqui. Agradeço à minha comunidade católica, Mel de Deus, que me ensinou a fazer o melhor para a glória de Deus, além de me ensinar princípios e valores que me ajudaram a ser uma pessoa melhor.

Agradeço, de forma especial, à minha orientadora, por ter-me apoiado nessa jornada, mostrando-me o caminho a seguir, além de todo o exemplo de dedicação, disciplina e esforço para entregar o melhor em tudo. Muito obrigado!

Por fim, agradeço a todos os amigos que estiveram ao meu lado durante esses anos na graduação.

A esses, a minha eterna gratidão!

“Software development is a young profession, and we are still learning the techniques and building the tools to do it effectively.“

(Martin Fowler)

Resumo

Nos últimos anos, as aplicações *front-end* têm assumido novas responsabilidades que, anteriormente, eram atribuídas ao servidor, causando um aumento na complexidade e no código-fonte. Por causa desse novo contexto, as equipes de desenvolvimento precisam superar os desafios de manter e evoluir essas aplicações que comumente transformam-se em grandes monolitos. Com isso, é importante ressaltar a importância da arquitetura apropriada no desenvolvimento do software em questão. Para contornar o problema apresentado, a comunidade tem buscado arquiteturas que permitam decompor as aplicações em módulos menores com o intuito de facilitar o desenvolvimento através da separação desses módulos segundo suas responsabilidades. Diante disso, surgem alternativas para esses novos desafios, como a Arquitetura Limpa e *Micro Front-end*. Este trabalho apresenta um estudo exploratório, orientado a provas de conceito, sobre a aplicação combinada de Arquitetura Limpa e *Micro Front-end* no desenvolvimento de aplicações web. O estudo revela comportamentos observados durante a aplicação dessas arquiteturas, compila métricas derivadas da análise estática do código-fonte e inclui a revisão das soluções apresentadas por especialistas na área. Foi desenvolvida uma loja on-line através da combinação dessas arquiteturas, aderindo a um método de desenvolvimento híbrido, que combina práticas de Scrum e Kanban. Os resultados obtidos com o desenvolvimento dessa loja on-line são apresentados, destacando-se os aspectos positivos e negativos identificados durante a conclusão de cada um dos desafios propostos no trabalho.

Palavras-chave: Arquitetura de Software.Arquitetura Limpa. *Micro Front-end*. Aplicações Web. Modularização. Reutilização de Software.

Abstract

In recent years, front-end web applications have taken on new responsibilities that were previously attributed to the server, resulting in an increase in complexity and source code. Due to this new context, development teams need to overcome the challenges of maintaining and evolving these applications that often become massive monoliths. With this in mind, it's important to highlight the importance of appropriate architecture in the development of the software in question. To address the problem presented, the community has been seeking architectures that allow the applications to be broken down into smaller modules in order to facilitate development through the separation of these modules according to their responsibilities. In light of this, alternatives emerge for these new challenges, such as Clean Architecture and Micro Front-end. This work presents an exploratory study, guided by proofs of concept, on the combined application of Clean Architecture and Micro Front-end in the development of web applications. The study reveals behaviors observed during the application of these architectures, compiles metrics derived from the static analysis of the source code, and includes a review of the solutions presented by specialists in the field. An online store was developed through the combination of these architectures, adhering to a hybrid development method that combines Scrum and Kanban practices. The results obtained with the development of this online store are presented, highlighting the positive and negative aspects identified during the completion of each of the challenges proposed in the work.

Key-words: Software Architecture. Clean Architecture. Micro Front-end. Web Applications. Modularization. Software Reuse.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

CDN	*Content Delivery Network*
CSS	*Cascading Style Sheets*
CSR	*Client-Side Rendering*
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JS	*JavaScript*
MVC	*Model-View-Controller*
POC	*Proof of Concept*
SEO	*Search Engine Optimization*
SSR	*Server-Side Rendering*
TCC	Trabalho de Conclusão de Curso
UI	*User Interface*
XML	*Extensible Markup Language*

Sumário

1 Introdução

Nesse capítulo, será apresentado o contexto, no qual o trabalho está inserido, com destaque para a área de atuação, no caso: Arquitetura de Software. Adicionalmente, ainda na Contextualização, tem-se uma breve descrição da problemática, mitigada nesse trabalho, com olhar centrado na inerente importância de uma adequada arquitetura para que um software seja mais facilmente mantido e evoluído. Na sequência, será abordado sobre a Questão de Pesquisa, que norteou o trabalho; apontada a Justificativa para a realização do mesmo, bem como especificados os principais Objetivos do trabalho. Por fim, tem-se a Organização da Monografia.

1.1 Contextualização

Segundo Martin (2017), a arquitetura de um software refere-se à estrutura utilizada pelos seus criadores em um sistema. Essa estrutura é representada pela divisão do sistema em componentes, organização e interação entre os mesmos, sendo seu principal intuito direcionar de forma estratégica e operacional o projeto de um sistema, impactando diretamente o ciclo de vida do mesmo (BASS; CLEMENTS; KAZMAN, 2012). A arquitetura, quando aplicada corretamente, resulta em fácil entendimento, desenvolvimento, manutenção e implantação do software (MARTIN, 2017).

Sendo assim, conforme Martin (2017), a definição de uma arquitetura de software é uma etapa essencial no desenvolvimento de sistemas robustos e escaláveis. Existem vários tipos/modelos de arquitetura, tais como: Arquitetura Monolítica (AWATI; WIGMORE, 2022); Arquitetura Portas e Adaptadores (GARG, 2022), Arquitetura *Model-View-Controller* (MVC) (MDN, 2023b), Arquitetura Limpa (MARTIN, 2012), dentre outras.

A Arquitetura Limpa, conceito proposto por Robert C. Martin, é uma abordagem que visa separar diferentes contextos em camadas independentes, facilitando a manutenção e a evolução do sistema (MARTIN, 2017). Em aplicações web, são encontrados diversos conceitos associados à Arquitetura de Software, sendo um deles o conceito de *micro front-end* (GEERS, 2017). *Micro front-end* consiste na composição de funcionalidades isoladas, desenvolvidas por times independentes e especializados em diferentes contextos, permitindo a evolução dessas funcionalidades também de forma independente (GEERS, 2017).

O desenvolvimento viabilizado usando como base o conceito de *micro front-end*, ainda segundo Geers (2017) e corroborado por Sousa (2021), tende a resultar em uma

organização de componentes arquiteturais menos acoplados e mais coesos, diferentemente do que comumente ocorre em aplicações construídas com Arquiteturas Monolíticas. Em uma Arquitetura Monolítica, toda a base de código consta centralizada, ou seja, em uma única aplicação (JACKSON, 2019).

O uso, geralmente sem planejamento, de arquiteturas como a monolítica em aplicações web, cuja adequação depende do projeto em que são aplicadas, pode acarretar dificuldades de manutenção e escalabilidade (MONTELIUS, 2021). Consequentemente, essas dificuldades tendem a incorrer em efeitos também negativos nas equipes de TI (Tecnologia da Informação), compelindo as empresas a procurar soluções alternativas (PONCE; MÁRQUEZ; ASTUDILLO, 2019). Ainda segundo Ponce, Márquez e Astudillo (2019), uma solução comumente adotada é a contratação de novos desenvolvedores com o objetivo de manter e evoluir as aplicações existentes, afinal os escopos dessas aplicações tendem a aumentar muito em tempo de manutenção. Contudo, Martin (2017) acorda que essa solução nem sempre resulta em um aumento na quantidade de entregas, pelo contrário, o mesmo relata que a tendência é que, com o passar do tempo, o número de desenvolvedores aumente enquanto a quantidade de entregas diminui ou permanece a mesma.

Diante do exposto, conceitos como Arquitetura Limpa e *micro front-end* estão cada vez mais populares no desenvolvimento de software. Um sistema bem estruturado, com baixo acoplamento e alta coesão, é capaz de proporcionar um valor significativo, simplificando a incorporação de novas funcionalidades e a correção de defeitos (BROWN; WOOLF, 2016). Consequentemente, e também enfatizado em Brown e Woolf (2016), aplicações construídas usando como base módulos menores e independentes podem representar soluções mais coerentes e adequadas.

1.2 Questão de Pesquisa

Ao longo deste trabalho, a seguinte questão de pesquisa foi respondida:

- Quais são os comportamentos de relevância revelados ao longo do desenvolvimento de uma loja on-line usando de forma combinada Arquitetura Limpa e *micro front-ends*?

Cabe esclarecer que se entende por comportamento relevante toda e qualquer evidência, principalmente de cunho arquitetural, que ocorra ao longo do desenvolvimento. Como exemplos, podem ser mencionados: (i) Há independência entre os componentes arquiteturais?; (ii) Há facilidade para se trabalhar com diferentes tecnologias?; (iii) Os repositórios de código são de fato menores?; (iv) Há facilidade em termos de escalabilidade?; (v) Há facilidade para substituir um dado aspecto no *front-end*?; (vi) Há replicação de

código; (vii) Há redundância de dependências?, e (viii) O código apresenta alto nível de complexidade ciclomática?

Nesse sentido, há literaturas que acordam quais são as principais vantagens e desvantagens esperadas ao se desenvolver aplicações web orientadas a *micro front-ends*, tal como Filho (2021). A intenção é perceber se essas vantagens e desvantagens são ou não mantidas (ou até mesmo potencializadas/mitigadas) ao combinar *micro front-ends* e Arquitetura Limpa.

No intuito de conferir maior clareza no tratamento e na exposição desses comportamentos, adotou-se um método baseado em Provas de Conceito. Na verdade, uma adaptação inspirada no método adotado em Queiroz (2022). Demais detalhes em termos metodológicos são conferidos no Capítulo 4 - Metodologia.

Além disso, de acordo com Jain, Malviya e Arya (2021), empresas em todo o mundo estão investindo cada vez mais em digitalização de seus serviços, o que é demonstrado pelo aumento no número de lojas on-line criadas. Dessa forma, esse trabalho optou por desenvolver uma loja on-line, devido a sua popularidade difundida no mundo digital (JAIN; MALVIYA; ARYA, 2021), permitindo fornecer uma visão aproximada da realidade enfrentada por diversos desenvolvedores diariamente.

1.3 Justificativa

Decompor uma aplicação em módulos menores, no intuito de facilitar a manutenção desses módulos, representa uma prática comum em Arquiteturas de Microsserviços (LEWIS; FOWLER, 2019) (REDHAT, 2023). Apesar da popularidade da Arquitetura de Microsserviços, sendo essa uma arquitetura de referência para os desenvolvedores de *micro front-ends*, muitas empresas ainda enfrentam dificuldades na implementação de *micro front-ends*, devido à falta de uma visão mais prática em termos de implementação (JACKSON, 2019).

Segundo Jackson (2019), *micro front-ends* apresentam benefícios bastante similares aos dos microsserviços, destacando-se:

- Bases de código menores, mais coesas e facilmente mantidas;
- Maior escalabilidade por meio da distribuição e da autonomia das equipes, e
- Possibilidade de atualização, aprimoramentos ou até mesmo reescrita de trechos de código *front-end* de maneira mais incremental do que anteriormente era possível, por exemplo, com as Arquiteturas Monolíticas (BARZOTTO; FARIAS, 2022).

Mesmo diante desses benefícios, com base em uma análise preliminar da literatura, acerca de *micro front-ends*, torna-se evidente que há aspectos ainda não ou pouco explorados. Dentre os materiais consultados, destacam-se: (MONTELIUS, 2021); (BASTOS, 2020), e (SILVA, 2021). Esses trabalhos concentram-se em analisar os principais resultados da construção de aplicações orientadas a *micro front-ends*.

Como um diferencial, o presente trabalho propõe estudar os principais comportamentos revelados ao longo do desenvolvimento de uma típica aplicação web - no caso, uma loja on-line - orientando-se pela combinação de Arquitetura Limpa e *micro front-ends*. A premissa é que ambas as abordagens - Arquitetura Limpa e *micro front-ends* - compartilham a busca por menor acoplamento e maior coesão, além da independência de componentes arquiteturais. Sendo assim, acredita-se que a Arquitetura Limpa tende a valorizar ainda mais os benefícios de *micro front-ends*.

Adicionalmente, buscou-se conduzir a pesquisa bem como documentar tais evidências, usando como base Provas de Conceito. Apesar da premissa positiva, anteriormente acordada, a presente pesquisa buscou revelar os comportamentos observados, independentemente dos mesmos potencializarem ou não as vantagens de *micro front-ends*, e/ou ainda mitigarem ou não as desvantagens de *micro front-ends*. A ideia foi justamente conduzir um estudo exploratório capaz de revelar todo e qualquer comportamento de cunho arquitetural observado.

Nesse contexto, não ocorreu pretensão em conduzir um viés comparativo entre essa pesquisa e outras encontradas na literatura, até mesmo devido ao fato de serem trabalhos conduzidos com propósitos diferentes e/ou orientados por metodologias específicas. Contudo, nesse estudo exploratório, as revelações possuem foco em conceitos, etapas orientadas a Provas de Conceito (POCs), dificuldades encontradas, facilidades vivenciadas, dentre outras particularidades do desenvolvimento baseado no modelo combinado de Arquitetura Limpa e Arquitetura de *micro front-ends*.

1.4 Objetivos

No intuito de responder a Questão de Pesquisa, foi estabelecido o principal objetivo do trabalho. Consequentemente, visando cumprir com esse Objetivo Geral, foram especificados alguns Objetivos Específicos. Esses objetivos constam acordados nas próximas subseções.

1.4.1 Objetivo Geral

Estudo exploratório via Provas de Conceito, revelando os comportamentos observados ao longo do desenvolvimento de uma loja on-line, e orientando-se pela combinação de Arquitetura Limpa e *micro front-ends*.

1.4.2 Objetivos Específicos

- Identificação, estudo, documentação e aplicação das principais técnicas relacionadas ao desenvolvimento de aplicações web utilizando-se de *micro front-ends*;
- Identificação, estudo, documentação e aplicação dos principais conceitos relacionados à Arquitetura Limpa em *micro front-ends*. Foi utilizada uma solução computacional com quatro *micro front-ends*, obtendo, nesse ponto, quatro *micro front-ends* desenvolvidos orientando-se à Arquitetura Limpa;
- Integração dos *micro front-ends*, procurando manter suas respectivas independências e responsabilidades, e
- Condução e documentação do processo de desenvolvimento - inerente ao cumprimento dos objetivos específicos anteriores - orientando-se por Provas de Conceito.

De acordo com Kojo (2021), uma das primeiras necessidades ao se desenvolver orientando-se por *micro front-ends* é definir a abordagem. Segundo os autores, há duas possibilidades de abordagens: (i) horizontal, na qual cada *micro front-end* corresponde, por exemplo, a uma parte da página, e (ii) vertical, na qual cada *micro front-end* atende a um contexto do negócio. Como exemplo, na abordagem horizontal, pode-se ter uma página desenvolvida com base em quatro *micro front-ends*, sendo: cabeçalho, menu, produtos e rodapé. Já um exemplo de abordagem vertical consiste em desenvolver cada contexto de negócio (ex. Carrinho de Compra e Pagamento) em *micro front-ends* específicos. Nesse sentido, será utilizada a abordagem vertical, no intuito de desenvolver quatro contextos de negócio na solução proposta - i.e. loja on-line. Outros detalhes serão tratados adiante, ao longo da monografia.

1.5 Organização da Monografia

Esta monografia está dividida em capítulos, sendo eles:

- Capítulo 2 - Referencial Teórico: Introduz os termos importantes relacionados à Arquitetura Limpa e *micro mront-ends*, bem como suas vantagens e desvantagens. Também são apresentados os principais aspectos referentes ao desenvolvimento web, bem como boas práticas, convenções e demais tópicos relevantes.
- Capítulo 3 - Suporte Tecnológico: Apresenta as tecnologias e ferramentas que conferem apoio ao desenvolvimento das provas de conceitos e aos demais aspectos relacionados a esse trabalho;

- Capítulo 4 - Metodologia: Descreve o processo metodológico adotado para condução do estudo exploratório, bem como para o desenvolvimento da loja on-line;
- Capítulo 5 - Estudo Exploratório: Aborda o estudo exploratório do trabalho, com foco na sua motivação, no seu planejamento, em características e identidade visual, além de apresentar a loja on-line desenvolvida, bem como os principais aspectos relacionados à sua construção;
- Capítulo 6 - Análise de Resultados: Apresenta os principais resultados obtidos com o estudo exploratório, conferindo um resumo das análises de resultado abordadas no Capítulo 5 - Estudo Exploratório, e
- Capítulo 7 - Conclusão: Descreve a conclusão do trabalho, retomando o contexto, objetivos concluídos, questão de pesquisa, bem como propondo ideias para trabalhos futuros.

2 Referencial Teórico

Este capítulo apresentará a fundamentação teórica utilizada para o desenvolvimento do trabalho, com o objetivo de facilitar a compreensão de tópicos fundamentais ao tema. No intuito de conferir um roteiro de leitura coerente, são apresentadas definições de termos importantes relacionados à Arquitetura de Software, perpassando por Padrões Arquiteturais, e pelas Arquitetura Limpa e *Micro Front-ends*, evidenciando sobre suas vantagens e desvantagens, de acordo com a literatura. Além disso, é acordada uma visão geral sobre Aplicações Web, sendo apresentados, na sequência, os principais aspectos referentes ao Desenvolvimento Web e *Micro Front-end*, com destaque às boas práticas, convenções e demais tópicos relevantes. Por fim, têm-se as Considerações Finais do Capítulo.

2.1 Arquitetura de Software

A sociedade atual está cada vez mais dependente de produtos de software em larga escala, que, geralmente, possuem alta complexidade e estão operando em um ambiente em constante evolução (VENTERS et al., 2018). A preocupação que surge no cenário de evolução e adaptação torna-se um desafio para manter e evoluir diante das mudanças apresentadas pelos *stakeholders* e requisitos do software (VENTERS et al., 2018). Dessa forma, a arquitetura de software torna-se alvo relevante ao se estruturar sistemas de software, por ser capaz de prover insumos para lidar com esse ambiente em constante evolução e mudança.

De acordo com Jaiswal (2019), a arquitetura de software pode ser definida como o projeto de um software. A arquitetura é responsável por abstrair a complexidade de um sistema e estabelecer a comunicação entre seus elementos (JAISWAL, 2019).

Jaiswal (2019) também enfatiza que, através da arquitetura de software, é esperado que as necessidades técnicas e de negócio sejam atendidas. É esperado que aspectos como desempenho, segurança, manutenibilidade e o sucesso do software estejam relacionados à arquitetura aplicada no sistema (JAISWAL, 2019).

2.1.1 Classificação

Segundo Richards e Ford (2020), a arquitetura de software pode ser classificada em duas categorias principais: Monolítica, que é a unificação de todo o código da aplicação em uma única base, e Distribuída, cujas unidades da aplicação são independentes e conectadas com algum protocolo. Cada categoria tem as suas particularidades, benefícios e malefícios,

o que faz com que não exista um tipo que seja melhor em todos os casos, conforme abordado pelos tópicos seguintes.

2.1.1.1 Arquitetura Monolítica

Arquitetura monolítica é um padrão de projeto para desenvolvimento de software, no qual todos os componentes de um sistema são integrados em um único e grande sistema (SAKOVICH, 2023). Nessa arquitetura, todos os recursos do sistema, desde a interface do usuário até o acesso aos dados, são agrupados em um único código-fonte e implantados em um único ambiente (SAKOVICH, 2023).

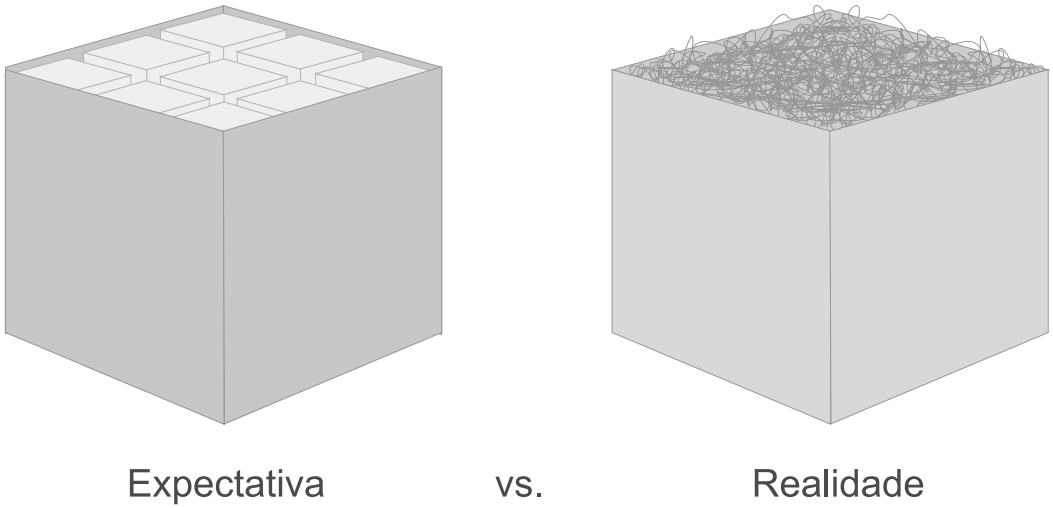
Também segundo Sakovich (2023), a aplicação da arquitetura monolítica pode se tornar mais benéfica dependendo do contexto e dos requisitos do sistema. Alguns dos benefícios dessa arquitetura são:

- Quando comparada com outras arquiteturas de software, a arquitetura monolítica é geralmente mais fácil de compreender, desenvolver e implantar. Isso se dá pelo fato de todos os componentes estarem no mesmo código-fonte, o que torna mais fácil compreender a estrutura e o fluxo de dados do sistema;
- A abordagem monolítica permite que equipes de desenvolvimento trabalhem colaborativamente e iterativamente, acelerando o ciclo de desenvolvimento. As melhorias podem ser implementadas com mais facilidade, uma vez que não há a necessidade de coordenar e administrar diferentes serviços;
- Essa arquitetura pode ser de maior desempenho em alguns casos de uso. Dado que todos os componentes estão sendo executados no mesmo ambiente, a comunicação entre eles pode ser mais eficiente do que comparado à arquitetura distribuída, e
- Com apenas um sistema monolítico para compreender e trabalhar, a experiência para novos desenvolvedores pode ser mais tranquila, uma vez que eles podem se familiarizar com o sistema como um todo, em vez de terem que compreender a interconexão entre vários serviços.

A arquitetura monolítica, apesar de ser amplamente utilizada e de seus benefícios, é criticada por sua falta de escalabilidade, flexibilidade e dificuldade de manutenção (RICHARDSON, 2019). Devido ao sistema ser desenvolvido como uma entidade coesa, na qual todas as partes estão fortemente acopladas, alterações em um módulo pode ter um grande impacto em todo o sistema, o que requer a execução da suíte de testes e a implantação do sistema como um todo a cada alteração (SAKOVICH, 2023). À medida que os sistemas crescem em complexidade e tamanho, as desvantagens dessa abordagem tornam-se mais evidentes (RICHARDSON, 2019).

Como mostra a Figura 1, pode ser tentador assumir que existem diversos módulos perfeitamente separados e isolados em um sistema monolítico, permitindo que sejam extraídos facilmente. Entretanto, na verdade, é extremamente difícil evitar criar diversas interconexões, sejam elas planejadas ou não (TILKOV, 2015). No final, tende a ter uma grande aplicação repleta de interconexão e acoplamento excessivo, dificultando a autonomia dos módulos entre si (TILKOV, 2015), conforme pode ser observado na Figura 1. Nessa figura, há uma comparação entre expectativa e realidade. Na expectativa, percebe-se um isolamento claro entre os módulos, causando a sensação de que há menor acoplamento e maior coesão no sistema. Na realidade, não se consegue evitar as várias interconexões entre os módulos, resultando em alto acoplamento e, por vezes, baixa coesão, e incorrendo em manutenções evolutivas complicadas em tempo, recursos e custos.

Figura 1 – Comparação entre expectativa vs realidade da arquitetura monolítica



Fonte: Martin Fowler[1] (Tradução: Autor)

2.1.1.2 Arquitetura Distribuída

Arquitetura distribuída é um modelo de design de software que envolve a distribuição de componentes e tarefas em uma rede de computadores conectados (TANENBAUM, 2007). Essa arquitetura permite que diferentes componentes do sistema sejam executados em máquinas diferentes, compartilhando recursos e trabalhando em conjunto para fornecer um serviço ou uma funcionalidade (TANENBAUM, 2007).

Outro ponto relevante para essa arquitetura, segundo Jaiswal (2019), é que a interação com o sistema deve ser consistente e uniforme, para que o usuário não perceba que ela está dividida em componentes distintos. Além disso, essa arquitetura tem outras características, como:

[1] https://martinfowler.com/articles/dont-start-monolith.html. Acessado pela última vez em Maio de 2023.

- Divisão do sistema em componentes menores, cada um responsável por uma função específica. Essa decomposição torna mais fácil a modularização, a reutilização de código e a manutenção do sistema;
- Comunicação e divisão dos componentes distribuídos devem permitir coordenar suas atividades para fornecer funcionalidades completas. Isso é alcançado, geralmente, através da troca de mensagens, chamadas de procedimento remoto ou eventos assíncronos, e
- Criação de mecanismos de tolerância a falhas, como a replicação de dados, monitoramento da saúde dos componentes e recuperação automática. Isso ocorre, pois é previsível, em sistemas distribuídos, que existam falhas em componentes individuais.

A arquitetura distribuída oferece diversos benefícios para o desenvolvimento de software, dentre eles:

- A distribuição da carga de trabalho entre várias máquinas torna possível que a aplicação atenda a diversos usuários, lidando com volumes crescentes de dados e tráfego;
- Ao distribuir o sistema em várias máquinas, é possível evitar pontos de falha, tornando o sistema mais tolerante e disponível;
- A arquitetura distribuída permite o uso de diferentes tecnologias em componentes diferentes, permitindo que equipes utilizem ferramentas e linguagens de programação mais adequadas para cada tarefa específica, e
- Como o sistema é dividido em componentes menores, a manutenção e a evolução do sistema podem ser realizadas de forma mais específica, sem afetar o funcionamento do sistema como um todo.

Em suma, uma arquitetura distribuída bem aplicada sugere que o sistema ofereça acesso fácil a seus recursos; deve ocultar o fato de que os recursos são distribuídos por uma rede; deve ser flexível e poder ser expandido (JAISWAL, 2019).

2.1.2 Padrões Arquiteturais

O padrão arquitetural é uma solução estudada, testada e documentada de um problema recorrente no desenvolvimento de aplicações de software (BALDISSERA, 2021). A aplicação de um ou mais padrões arquiteturais depende do objetivo da aplicação e do problema a ser solucionado. De acordo com Baldissera (2021), os seguintes padrões são os mais populares:

- *N-Layers* (N-camadas);
- *Client-server* (cliente-servidor);
- *Model-view-controller* (MVC);
- *Microservices* (microsserviços);
- *Pipes-and-filters* (PF);
- *Peer-to-Peer* (P2P);
- *Service-Oriented Architecture* (SOA), e
- *Publish-Subscribe* (Pub/Sub).

Dentre os padrões apresentados, optou-se por abordar os padrões Cliente-Servidor e N-Camadas, que foram os estilos mais relevantes para o desenvolvimento desse trabalho.

2.1.2.1 Cliente-Servidor

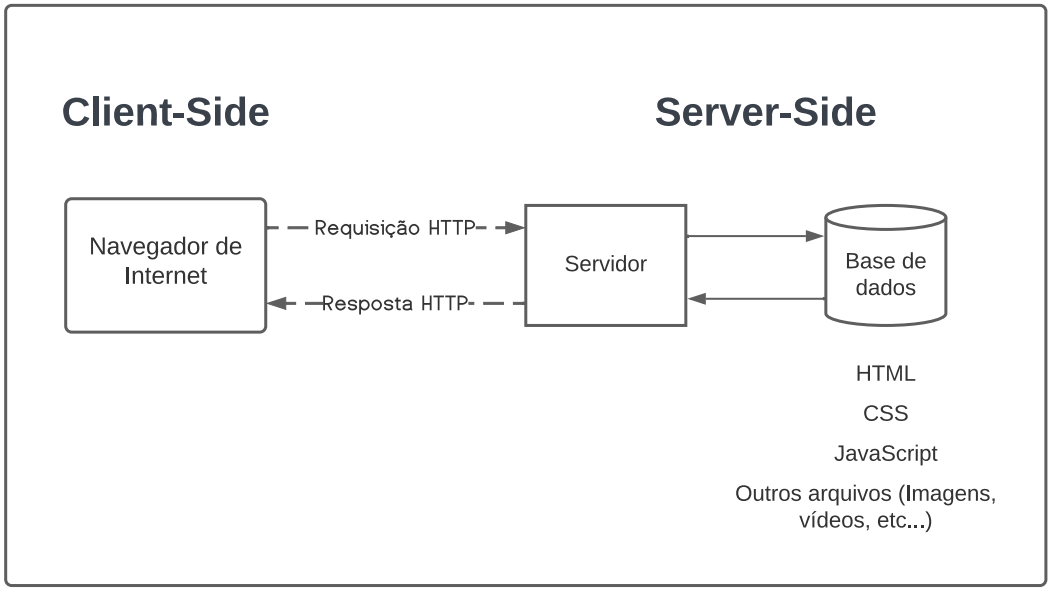
Quando entramos em uma página web, uma requisição é enviada para um servidor solicitando os arquivos necessários para mostrar o conteúdo da página para o usuário. O servidor, com os arquivos necessários, responde ao cliente (Navegador de Internet do usuário) com os códigos HTML, CSS e JS associados (FREECODECAMP, 2015).

Esse tipo de solução é bastante utilizado em páginas estáticas, pois garante que todos os dados da página estejam armazenados no servidor, permitindo que a resposta enviada para o navegador seja a mesma, e garantindo que todos os clientes tenham acesso à mesma página (MDN, 2023a).

Esse modelo ainda torna possível que todos os dados das páginas sejam enviados para uma rede de distribuição de conteúdo (em inglês, *Content Distribution Network* - CDN), o que torna possível que o mesmo conteúdo esteja presente em servidores espalhados por uma rede global (FREECODECAMP, 2015). Assim, quando um cliente solicitar dados de uma página para um servidor, ele terá uma resposta mais rápida, pois será respondida pelo servidor mais próximo geograficamente (FREECODECAMP, 2015).

Nesse contexto, esse modelo é uma requisição usando o protocolo HTTP para o servidor, com as informações de qual conteúdo espera-se que seja retornado (MDN, 2023a). O servidor, com essas informações, procura todos os dados em seu banco de dados e os envia em forma de resposta para o cliente. O cliente, a partir da resposta, exibe na tela todas as informações que o servidor respondeu (MDN, 2023a). Esse fluxo pode ser visto na Figura 2.

Figura 2 – Modelo Cliente-Servidor



Fonte: Autor

2.1.2.2 N-Camadas

A arquitetura n-camadas refere-se à qualquer arquitetura de aplicativos com mais de uma camada (IBM, 2023). A divisão por camadas permite com que cada camada execute sua própria infraestrutura, possibilitando que cada uma seja desenvolvida simultâneamente por uma equipe independente de desenvolvimento, além de poder ser atualizada ou ajustada, conforme necessário, sem impactar as outras camadas do software (IBM, 2023).

Segundo IBM (2023), é comum que aplicações de n-camadas, principalmente as aplicações Web, se limitem a três camadas, pois um aumento do número de camadas pode oferecer poucos benefícios, e podem acarretar em aplicações mais lentas; mais dificéis de gerenciar, e mais caras de serem executadas. Com isso, costuma-se dividir esse tipo de aplicação nas seguintes camadas:

- Camada de apresentação ou interface do usuário, que é a camada de comunicação do aplicativo (ou aplicação), na qual o usuário final interage. Seu principal objetivo é exibir e coletar informações, podendo ser executada em um navegador web ou desktop;
- Camada de aplicação, na qual os dados coletados pela camada de apresentação são processados, e
- Camada de dados, na qual os dados associados ao aplicativo (à aplicação) são armazenados e gerenciados em um banco de dados.

Cabe ressaltar, entretanto, que apesar dessa organização em camadas comumente encontrada, dividida em Apresentação, Aplicação e Dados, há outras propostas de arquiteturas e padrões arquiteturais, orientados ao estilo N-Camadas, e que sugerem organizações diferenciadas. O *Model-View-Controller* (MVC) (LEFF; RAYFIELD, 2001), por exemplo, propõe três camadas, sendo: Visão, Controle, e Modelo. A camada Controle é uma intermediária entre a camada Visão e a camada Modelo, diminuindo acoplamento e proporcionando maior coesão. Já a persistência (i.e. o banco de dados), nesse caso, é representada em uma quarta camada. Há propostas de organizações das camadas ainda mais diferenciadas, conforme ocorre com a Arquitetura Limpa e a Arquitetura de Microsserviços, apresentadas na sequência.

2.1.3 Arquitetura Limpa

Com passar do tempo, diversas ideias sobre arquitetura de sistemas de software surgiram. Dentre essas, destacam-se a arquitetura hexagonal(ou Portas e Adaptadores), arquitetura cebola e *screaming architecture*, que se assemelham pelo objetivo de separar responsabilidades (MARTIN, 2012). Todas alcançam esse objetivo através da divisão do software em camadas (MARTIN, 2012). Nessa mesma linha, há a Arquitetura Limpa.

Segundo MARTIN (2012), autor da Arquitetura Limpa, a aplicação de qualquer uma dessas arquiteturas produz sistemas com as seguintes características:

- Autonomia sobre *frameworks*, uma vez que a arquitetura não depende da existência de um *framework*. Ao contrário, o sistema os usa como ferramentas, permitindo a construção do software sem limitá-lo com as restrições do *framework*;

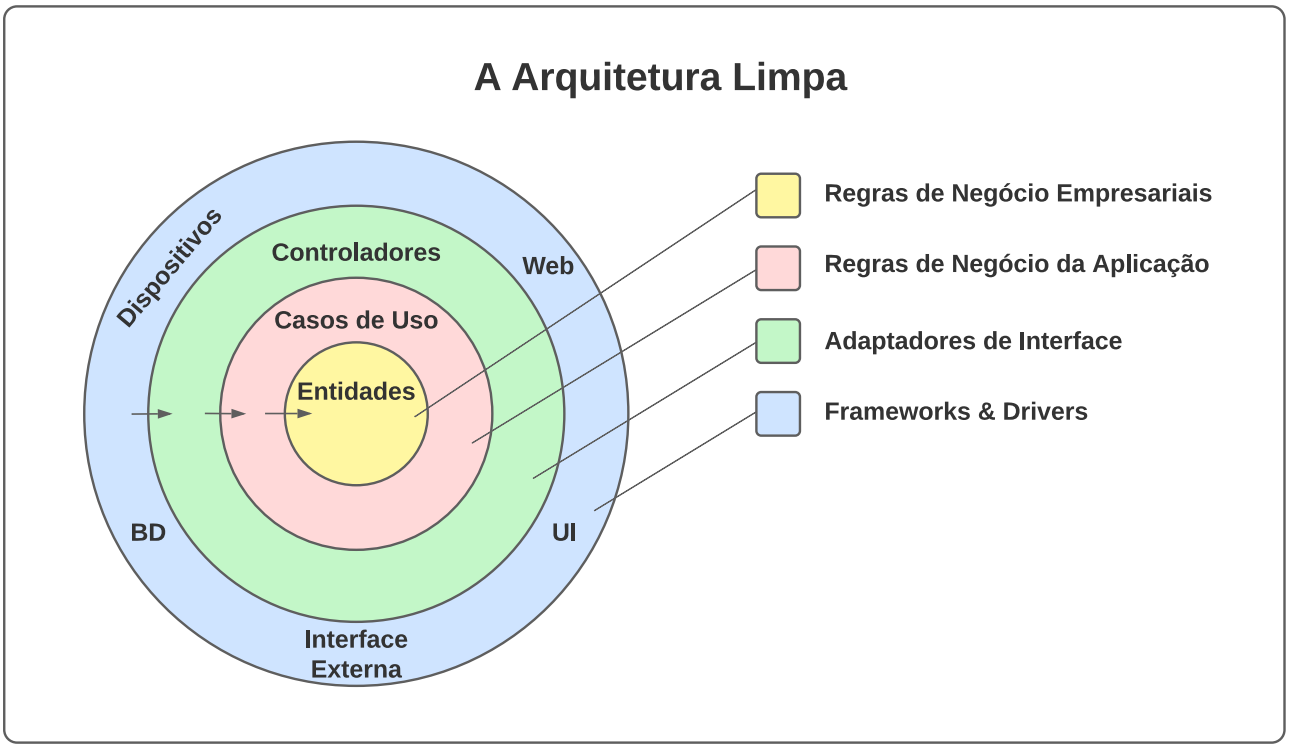
- Independência de elementos externos para testar as regras de negócio, que podem ser as mesmas testadas sem a interface gráfica, os servidores web, ou qualquer outro elemento externo;

- Independência da interface gráfica, permitindo que ela seja modificada facilmente, sem precisar alterar as demais partes do sistema;

- Liberdade sobre os bancos de dados, uma vez que as regras de negócio não estão ligadas a um banco de dados específico, e

- Independência de qualquer agente externo, devido às regras de negócio não terem ciência da existência de algo relacionado à qualquer camada superior.

Figura 3 – Arquitetura Limpa



Fonte: Robert C. Martin[2] (Tradução: Autor)

A Figura 3, do autor Robert C. Martin, representa a organização da Arquitetura Limpa. Dada a organização apresentada, cada camada tem uma responsabilidade específica, e a comunicação entre elas depende de uma regra de dependência (MARTIN, 2012). Essa regra de dependência estabelece que as dependências das camadas só podem ser apontadas para dentro. Isso significa que uma camada interna não pode saber sobre uma camada externa (MARTIN, 2012).

Outra camada relevante, de acordo com MARTIN (2012), é a de Entidades. Essa camada está ligada às regras de negócio de toda a aplicação. Essa camada pode ser um objeto com métodos ou um conjunto de estruturas de dados e funções.

Em seguida, a camada dos Casos de Uso apresenta as regras de negócio específicas da aplicação (MARTIN, 2012). É responsável por encapsular e implementar de todos os casos de uso do sistema. Esses casos de uso gerenciam o fluxo de dados entre as entidades para alcançar seus objetivos. Além disso, não é esperado que mudanças nessa camada afetem as entidades (MARTIN, 2012).

Os Adaptadores de Interface convertem os dados para o formato mais adequado para serem usados por casos de uso, entidades, banco de dados e interface gráfica (MARTIN, 2012). A camada seria a base para toda a arquitetura MVC, por exemplo.

[2] https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. Acessado pela última vez em Maio de 2023.

Finalmente, a camada de *Frameworks* e *Drivers* é composta por *frameworks* e ferramentas, como banco de dados, *framework front-end*, dentre outros (MARTIN, 2012). É nessa camada que são armazenados todos os detalhes do sistema. Por detalhes, entende-se o banco de dados, o *framework web*, dentre outros. São considerados detalhes porque devem ser mantidas do lado de fora, onde podem causar pouco impacto, caso precisem ser alteradas (MARTIN, 2012).

Contudo, conforme abordado por MARTIN (2012), a utilização da Arquitetura Limpa também apresenta desvantagens, tais como:

- Maior curva de aprendizado, tendo em vista que todas as camadas devem funcionar em conjunto, o que requer certo tempo de aprendizado e adaptação dos desenvolvedores, e

- Ampliação no número de classes para seguir as diretrizes da arquitetura, por isso, a sua utilização pode não ser ideal em aplicações de baixa complexidade.

2.1.4 *Micro Front-ends*

Nos últimos anos, a arquitetura baseada em microsserviços tem despertado muita atenção, tornando-se um tema relevante nas pesquisas da área (YANG; LIU; SU, 2019). Ao analisar aplicações monolíticas, percebe-se que há um grande número de módulos, suas dependências podem não ser claras, e o acoplamento entre os mesmos é alto (YANG; LIU; SU, 2019). Em tal situação, a alteração de qualquer módulo requer que todo o projeto, ou boa parte dele, seja reconstruído e implementado novamente.

Segundo Yang, Liu e Su (2019), os microsserviços, uma variante da arquitetura baseada em serviços, separam a aplicação em uma coleção de serviços que são desacoplados e independentes. Dessa forma, cada serviço é responsável por um aspecto do sistema e pode ser desenvolvido, testado e implementado de forma independente, facilitando assim o trabalho de diferentes equipes em conjunto.

Micro Front-ends aplicam o conceito de microsserviços em aplicações web, transformando-as de uma única aplicação para uma combinação de múltiplas pequenas aplicações *micro front-end* (YANG; LIU; SU, 2019). Assim como os microsserviços, *micro front-ends* podem ser criados, testados e implementados de forma independente, pois são tratados como uma combinação de funcionalidades que podem ser de responsabilidade de diferentes equipes (YANG; LIU; SU, 2019). Dessa forma, um *micro front-end* pode ser executado de forma independente, mesmo que os outros *micro front-ends* não estejam em execução (MONTELIUS, 2021).

Montelius (2021) relata os seguintes benefícios na adoção de *micro front-ends*:

- Facilidade no desenvolvimento de novas funcionalidades, pois todos os conhecimentos necessários estão em uma equipe, diminuindo as barreiras de comunicação entre os desenvolvedores *front-end* e *back-end*, pois ambos estão dentro do mesmo contexto;

- A manutenção de aplicações *micro front-end* é mais simples em relação às aplicações monolíticas, pois seu escopo é menor, o que torna mais fácil compreender o código de todo o *micro front-end* em comparação ao esforço necessário para compreender toda a base de código monolítica;

- Além disso, as atualizações tornam-se mais fáceis. Como cada aplicação funciona de forma independente, a equipe responsável consegue realizar alterações sem afetar as aplicações mantidas por outras equipes. Isso é relevante porque as tecnologias *front-end* mudam com rapidez, e o que era considerado como o melhor em um ano, no outro, pode ser considerado legado, e

- A utilização de *micro front-ends* também torna mais fácil a migração de aplicações legadas, uma vez que a arquitetura permite que uma nova aplicação *micro front-end* seja executada junto com a legada.

Contudo, Nascimento e Sotto (2020) abordam algumas dificuldades que podem surgir na utilização da arquitetura *micro front-ends*, tais como:

- Maior número de arquivos que o navegador precisa carregar ao acessar uma aplicação devido à duplicação de dependências;

- Testes aprimorados devem ser feitos para assegurar que, caso um *micro front-end* não esteja funcionando corretamente, ele não seja disponibilizado para o usuário;

- Permanência de um padrão na experiência do usuário pode ser difícil, uma vez que cada *micro front-end* é desenvolvido por uma equipe independente e com autônoma, e

- Maior complexidade operacional da aplicação, que é uma desvantagem também enfrentada pelos microsserviços, onde a arquitetura distribuída cria mais elementos para serem gerenciados. Como exemplo, podem ser citados: (i) maior número de repositórios; (ii) configuração de pipelines; (iii) ferramentas, dentre outros.

Logo, a arquitetura *micro front-end* torna-se uma solução inovadora para resolver o problema criado por aplicações monolíticas, que podem ser complexas, difíceis de manter e atualizar (NASCIMENTO; SOTTO, 2020).

2.2 Aplicações Web

Nesta Seção, serão analisadas as diferentes abordagens para o desenvolvimento de aplicações web, com foco em *server-side*, *client-side* e *single-page applications* (SPAs). O objetivo é compreender as características e os benefícios de cada uma dessas abordagens e analisar suas aplicações no contexto do desenvolvimento web.

2.2.1 Aplicações *Server-Side*

Usando o modelo Cliente-Servidor ou N-Camadas, a estratégia de criação de páginas web renderizadas pelo servidor torna possível criar experiências personalizadas para cada usuário no momento de usar a aplicação (COCCA, 2023). Quando uma requisição chega ao servidor, ele busca as informações necessárias do banco de dados e as substitui dentro do modelo da página. Depois, com todo o conteúdo estruturado, o servidor responde ao cliente com todo o HTML já compilado (BREUX, 2018).

Em um cenário semelhante ao apresentado anteriormente, uma página que usa SSR (*Server-Side Rendering*) pode, por exemplo, mostrar o nome do usuário logado na aplicação, mesmo que diversos clientes estejam acessando a mesma página web ao mesmo tempo. Nesse cenário, os clientes poderão ver o seu próprio nome, sem a necessidade de se criar diversos arquivos HTML estáticos para cada cliente com antecedência e os armazenar no banco de dados (COCCA, 2023).

Além de permitir a personalização do que é mostrado para o usuário, o avanço digital e a necessidade de aumentar sua visibilidade na web fizeram com que diversas aplicações optassem por essa estratégia para serem indexadas pelos buscadores que, alguns anos atrás, conseguiam ler apenas o conteúdo presente no HTML da página, mas sem suporte a código JavaScript (COCCA, 2023).

No entanto, essa estratégia coloca todo o processamento no servidor, e o tempo de resposta até o cliente ver o conteúdo da página pode variar de centenas de milissegundos até segundos (BREUX, 2018). Para cada página visitada pelo usuário, o navegador deve enviar uma requisição para o servidor. Este processará a requisição; buscará os dados pertinentes para a construção da página; compilará tudo em HTML, e responderá para o navegador com todos os códigos HTML, CSS e JS necessários para a renderização da página (BREUX, 2018).

2.2.2 Aplicações Client-Side

Client-Side rendering ou CSR é uma técnica de renderização que realiza todo o processamento e a renderização da interface do usuário no navegador, ao invés de serem realizados no servidor (PATTERNS, 2023). Com CSR, lógica, carregamento de dados,

estruturação das páginas, e roteamento de uma página são responsabilidade do JavaScript.

O CSR também usa o modelo Cliente-Servidor, mas, ao contrário do servidor retornar o HTML completo da página, ele responde com um código HTML simples e sem conteúdo, como mostrado no Código 1. Com esse código disponível no cliente, o JavaScript entra em ação, e começa a popular a DOM (*Document Object Model*) com todos os elementos necessários para a página em questão.

Código 1 – Exemplo de HTML enviado pelo servidor em aplicações de página única

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/src/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Loja Virtual</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Fonte: Autor

A utilização do CSR como método para criação de aplicações de página única tornou-se bastante popular, além de proporcionar uma ótima experiência ao usuário, uma vez que é possível navegar pela aplicação sem precisar recarregar a página (PATTERNS, 2023). A responsabilidade do *Cliente-Side* de construir o conteúdo das páginas, torna a navegação mais rápida e gera uma sensação de maior responsividade para o usuário (PATTERNS, 2023).

Como Patterns (2023) relata, essa abordagem também tem alguns problemas, sendo:

- SEO (*Search Engine Optimization*) - É frequente que aplicações renderizadas pelo cliente sejam prejudicadas na indexação por buscadores, uma vez que é necessário realizar diversas requisições até que todo o código necessário para a construção das páginas esteja disponível para o cliente, o que faz com que os buscadores indexem o conteúdo retornado inicialmente pelo servidor, como mostrado na Figura 1, ao contrário do conteúdo real da página que seria renderizado pelo JavaScript, e
- Desempenho - Apesar da melhora no tempo de resposta nas interações, que é causada por não precisar esperar o servidor gerar todo o conteúdo HTML, os usuários

podem sentir uma demora no primeiro carregamento da aplicação. Isso acontece, pois todo o código JavaScript necessário para a aplicação é reunido em um *bundle* que costuma crescer conforme a aplicação vai ganhando novas funcionalidades, e precisa ser carregado no navegador para a construção da página.

2.2.3 Aplicações Single-Page

A partir do surgimento de *frameworks* como Angular, Vue, React e outros, aplicações de página única ou *Single-Page Applications* tornaram-se cada vez mais comuns. A popularidade ainda é maior quando o buscador Google adicionou a funcionalidade de "ler"o código JavaScript, permitindo que o conteúdo dessas aplicações seja indexado (BREUX, 2018).

Ao contrário do servidor gerar todo o código da página, como acontece no Cliente-Servidor, a aplicação recebe um arquivo mínimo HTML e os arquivos JS necessários. Assim que todo o carregamento de *assets* é finalizado, o JavaScript constrói todos os elementos da tela e os apresenta para o usuário (BREUX, 2018).

As aplicações de página única são chamadas assim porque, ao precisar atualizar algo em tela ou acessar outra página, não há uma nova requisição para o servidor, pois o JavaScript consegue atualizar os elementos da tela em tempo real ou criar novos elementos de outra página sem precisar recarregar a mesma (PATTERNS, 2023). Tudo isso é feito com base na estratégia de Aplicações Client-Side.

2.3 Desenvolvimento Web e *Micro Front-end*

Dentro do contexto das boas práticas para o desenvolvimento de aplicações web, esta Seção abordará três aspectos essenciais: orientação a objetos com foco nos princípios SOLID; orientação a componentes, e orientação a convenções para a comunicação de aplicações *micro front-end*. O objetivo é analisar alguns conceitos relevantes para o desenvolvimento de software web.

As boas práticas no desenvolvimento de software são diretrizes e abordagens recomendadas para a criação de sistemas de software de qualidade, eficiência e manutenibilidade (WILSON et al., 2014). São baseadas em experiências de profissionais da área e têm como objetivo aumentar a produtividade do desenvolvedor, além de melhorar a robustez e a escalabilidade do software desenvolvido (WILSON et al., 2014).

2.3.1 Orientação a Objetos

A orientação a objetos é um paradigma de programação que orienta o design de um sistema em termos de entidades, objetos e seus relacionamentos (ROVEDA, 2022). A sua

utilização traz vantagens como encapsulamento e reutilização de código, o que facilita o entendimento acerca do código, e aumenta a produtividade para quem está desenvolvendo (ROVEDA, 2022). De acordo com Roveda (2022), esse paradigma é composto por quatro pilares fundamentais:

- Abstração é o processo de extrair entidades do mundo real para dentro do sistema seguindo suas responsabilidades;
- Encapsulamento, que protege os atributos de um objeto, assegurando que apenas o objeto possa alterar seus valores;
- Herança é o compartilhamento de características e responsabilidades entre objetos, e
- Polimorfismo, que permite especializar classes do sistema, adicionando características que não estão presentes na classe superior da hierarquia.

No contexto de orientação a objetos, é comum a utilização dos princípios SOLID para o desenvolvimento de aplicações (OLORUNTOBA, 2021). O acrônimo SOLID significa: S - *Single Responsibility Principle* (Princípio da responsabilidade única); O - *Open Closed Principle* (Princípio aberto-fechado); L - *Liskov Substitution Principle* (Princípio da substituição de Liskov); I - *Interface Segregation Principle* (Princípio da segregação de interfaces), e D - *Dependency Inversion Principle* (Princípio da inversão de dependência). Sendo assim, seguem explicações para cada princípio:

- Princípio da responsabilidade única - Uma classe deve ter apenas uma responsabilidade. Se uma classe tem diversas responsabilidades, a possibilidade de surgirem erros aumenta, uma vez que modificar alguma de suas responsabilidades pode afetar outra responsabilidade (OLORUNTOBA, 2021);
- Princípio aberto-fechado - Classes devem ser abertas à extensão, mas fechadas à modificação. Esse princípio impede que a alteração do comportamente de uma classe afete o sistema que a usa. Se for necessário adicionar funcionalidades a uma classe, elas devem ser adicionadas nas funções que já existem e não modificadas (THELMA, 2020);
- Princípio da substituição de Liskov - Cada subclasse ou classe derivada deve poder ser substituída pela sua classe base (OLORUNTOBA, 2021);
- Segregação de interfaces - Impede que uma classe seja forçada a implementar uma interface que não usa ou a depender de métodos que não são usados (OLORUNTOBA, 2021), e

- Princípio da inversão de dependência - Entidades devem depender de abstrações, e não de implementações. Isso quer dizer que um módulo de alto nível não deve depender de um módulo de baixo nível, mas sim de uma abstração (OLORUNTOBA, 2021).

2.3.2 Orientação a Componentes

De acordo com Crnkovic (2001), o objetivo principal do desenvolvimento de software baseado em componentes é criar aplicações, que são a junção de diferentes componentes. Um componente, no contexto de software, pode ser definido como uma unidade com interface e dependências específicas, que pode ser criado de forma independente, além de permitir a combinação com outros componentes (CRNKOVIC, 2001).

Assim, a evolução desse tipo de aplicação é alcançada por meio da manutenção e da atualização desses componentes (CRNKOVIC, 2001). Ainda de acordo com Crnkovic (2001), a criação de novas funcionalidades e a combinação de diferentes contextos são fatores relevantes para que uma aplicação seja bem sucedida. Dessa forma, o desenvolvimento de aplicações baseadas em componentes facilita a integração de diferentes módulos que agrupam funcionalidades em contextos diferentes.

A abordagem de desenvolvimento com base em componentes apresenta diversas vantagens (CRNKOVIC, 2001). Dentre elas, destacam-se:

- Melhor gerenciamento de complexidade;
- Diminuição do tempo para a disponibilização do software;
- Aumento de produtividade;
- Melhor qualidade de código;
- Maior grau de consistência, e
- Reusabilidade.

O desenvolvimento orientado a componentes também tem desvantagens, segundo Crnkovic (2001). Algumas delas são:

- Aumento de tempo e esforço necessários para o desenvolvimento de componentes;
- Ambiguidade e não legibilidade dos requisitos;
- Conflito entre criar componentes suficientemente genéricos, e comprometer a usabilidade dos mesmos;
- Altos custos para a manutenção de componentes, e
- Conflito entre promover facilidade de implementar mudanças, e perder confiabilidade no código.

Dessa forma, a reutilização de código é uma das principais vantagens do desenvolvimento orientado a componentes, pois permite reaproveitar código que já foi testado e validado (CRNKOVIC, 2001). Além disso, a manutenibilidade é facilitada, uma vez que a alteração de um componente não afeta os demais componentes do sistema. A escalabilidade também é uma vantagem, pois é possível adicionar novos componentes sem que isso afete o sistema como um todo. Por último, a facilidade de integração é benéfica, já que é viável combinar componentes de diferentes contextos, o que torna mais fácil a criação de novas funcionalidades (CRNKOVIC, 2001).

2.3.3 Orientação a Convenções

Ao serem analisados os benefícios inerentes à adoção da arquitetura *Micro Front-end*, destacam-se a modularidade e a independência das pequenas aplicações. Todavia, essa liberdade exige mais cuidado quando pretende-se manter a comunicação entre os *micro front-ends* (SHARMA, 2022).

Segundo Sharma (2022), existem diversas maneiras de lidar com esse desafio. Alguns exemplos são: (i) Uso de Propriedades; (ii) Armazenamento Local, e (iii) Eventos Customizados.

A forma mais simples de comunicação é a comunicação usando propriedades. Essa técnica mantém a informação na aplicação principal, que embarca todos os *micro front-ends*, e repassa os dados para cada um por propriedades (SHARMA, 2022). O Código 2 mostra um exemplo de como seria a aplicação dessa solução em uma aplicação React. Essa solução é atendida por diversos *frameworks*, além de ser uma das técnicas mais populares para compartilhamento de dados em uma arquitetura baseada em componentes (SHARMA, 2022).

Código 2 – Exemplo de comunicação usando propriedades

```
import { Checkout } from '@microfrontend/checkout';
import { Cart } from '@microfrontend/cart';

export function Ecommerce() {
  const [productQuantity, setProductQuantity] = useState([]);

  function handleChangeProductQuantity(newQuantity) {
    setProductQuantity(newQuantity);
  }
  return (
    <>
      <Cart
        handleChangeProductQuantity={handleChangeProductQuantity}
      />
      <Checkout productQuantity={productQuantity} />
    </>
  )
}
```

Fonte: Autor

Outra solução é usar a API de armazenamento dos navegadores de Internet, como *Local Storage*, *Session Storage* e *Cookies*. Essa abordagem é atendida em diversos tipos de aplicação *front-end*, facilitando a sua implementação, uma vez que cada *micro front-end* pode ler e escrever os dados de maneira direta (SHARMA, 2022). O Código 3 mostra um exemplo de como essa técnica poderia ser aplicada.

Dado que os *micro front-ends* têm acesso direto aos dados, não é mais necessário que os dados estejam armazenados em uma aplicação global para repassar para cada aplicação (SHARMA, 2022). De acordo com Sharma (2022), essa técnica funciona melhor quando a informação que está sendo transmitida é simples e pequena. No entanto, sua utilização pode dificultar distinguir qual aplicação está escrevendo o dado, além de não proteger os dados armazenados.

Outra proposta para a comunicação é a de usar a API de eventos presente nos navegadores para conseguir transmitir dados através da publicação e da inscrição dos mesmos (SHARMA, 2022). Essa abordagem é bastante semelhante à arquitetura orientada a eventos no contexto de microsserviços. Essa técnica é fácil de escalar conforme a aplicação cresce, além de ser atendida pela maioria dos navegadores de Internet. No entanto, pode ser necessário bastante tempo para sua configuração em todo o sistema (SHARMA, 2022). O Código 4 apresenta um exemplo do seu uso.

Código 3 – Exemplo de comunicação usando API de armazenamento

```
// ***************  CHECKOUT  ***************

export function Checkout() {
  const [productQuantity] = useState(() => {
    const storedQuantity = localStorage.getItem("productQuantity");
    return storedQuantity ? JSON.parse(storedQuantity) : 0;
  });

  return (
    <h1>
      A quantidade de produtos adicionado é
      de {productQuantity} unidades
    </h1>
  );
}

// ***************  CART  ***************

export function Cart() {
  const [productQuantity, setProductQuantity] = useState(0);

  function handleChangeProductQuantity(newQuantity) {
    setProductQuantity(newQuantity);

    localStorage.setItem("productQuantity", JSON.stringify(newQuantity));
  }

  return (
    <div>
      <p>Quantidade no carrinho: {productQuantity}</p>

      <button
        onClick={() => handleChangeProductQuantity(productQuantity + 1)}
      >
        Adicionar uma unidade
      </button>
    </div>
  );
}
```

Fonte: Autor

Código 4 – Exemplo de comunicação usando eventos customizados

```
// ****************  PRODUCTS PAGE  ****************

import { Product } from "./Product";

export function Products() {
  const [productsInCart, setProductsInCart] = useState([]);
  function handleAddToCart(newProduct) {
    setProductsInCart([...productsInCart, newProduct]);
    const addToCartEvent = new CustomEvent(
      'ADD_TO_CART',
      { detail: { newProduct: newProduct } }
    );
    window.dispatchEvent(addToCartEvent);
  }
  return (
    <Product handleAddToCart={handleAddToCart} />
  )
}

// ****************  CART  ****************

import { ProductInCart } from "./ProductInCart";

export function Cart() {
  const [productsInCart, setProductsInCart] = useState([]);
  function handleAddToCartEvent(data) {
    setProductsInCart([...productsInCart, data.newProduct]);
  }
  useEffect(() => {
    window.addEventListener('ADD_TO_CART', handleAddToCartEvent)
    return () => {
      window.removeEventListener('ADD_TO_CART', handleAddToCartEvent);
    }
  }, []);
  return (
    <div>
      {productsInCart.map(product => (
        <ProductInCart key={product.id} product={product} />
      )}
    </div>
  )
};
```

Fonte: Autor

2.4 Considerações Finais do Capítulo

Esse capítulo proveu uma visão sobre aspectos de relevância relacionados ao trabalho. Inicialmente, foram apresentadas as classificações mais comuns na arquitetura de software. A arquitetura monolítica é caracterizada por concentrar todas as funcionalidades e os aspectos de uma aplicação em um único bloco de código, tornando-se assim, uma aplicação única e centralizada. A arquitetura distribuída, por outro lado, divide o sistema em módulos independentes, distribuídos por diferentes componentes.

Após isso, foram apresentados os principais estilos arquiteturais usados em aplicações web. O modelo Cliente-Servidor é usado para separar as responsabilidades entre o cliente, que constrói a interface do usuário, e o servidor, que processa as requisições e fornece respostas com os recursos solicitados. Além disso, o modelo N-Camadas divide a aplicação em camadas funcionais, como a camada de apresentação, a lógica de negócio e o acesso a dados, o que proporciona uma estruturação organizada e modular.

Outro tópico abordado no capítulo diz respeito à Arquitetura Limpa, uma abordagem que enfatiza a separação de responsabilidades e a independência de *frameworks* e bibliotecas específicas. A Arquitetura Limpa propõe uma estruturação mais clara do código-fonte, tornando-o mais testável, reutilizável e adaptável a mudanças. Em seguida, introduziu-se o *Micro Front-end*, uma arquitetura que divide a interface do usuário em partes menores e independentes, facilitando o desenvolvimento e a evolução de aplicações *front-end* em larga escala. Essa abordagem proporciona benefícios como a escalabilidade, a reutilização de componentes, e a flexibilidade tecnológica.

No que diz respeito às aplicações web, também foram introduzidas: as aplicações *single-page*, aplicações *client-side*, e aplicações *server-side*. As aplicações *single-page* proporcionam uma experiência contínua ao usuário, carregando o conteúdo de forma dinâmica sem a necessidade de recarregar a página inteira. As aplicações *client-side* transferem parte das regras de negócio para o navegador do usuário, proporcionando uma interação mais rápida e fluida; enquanto as aplicações *server-side* mantêm as regras de negócio no servidor, fazendo com que o navegador apenas acesse e apresente as informações providas pelo servidor.

Mais ao final, foram apresentadas boas práticas no desenvolvimento web e de *Micro Front-ends*, começando com os princípios SOLID, depois de uma breve introdução sobre Orientação a Objetos. Além disso, apresentou-se a orientação a componentes, cuja base está na reutilização e na composição de elementos independentes, tornando o desenvolvimento, a manutenção e a evolução dos sistemas mais fáceis. A orientação a convenções, por fim, aborda padrões definidos pela comunidade sobre aspectos como a comunicação entre *micro front-ends*.

3 Suporte Tecnológico

Serão apresentadas, neste capítulo, as principais ferramentas e tecnologias escolhidas para a condução do trabalho. O capítulo foi divido em três seções, sendo: a primeira, Ferramentas para Desenvolvimento da Loja On-line, sobre as tecnologicas utilizadas no código da aplicação Loja On-line; a segunda, Ferramentas de Apoio, sobre outras ferramentas significativas que conferiram apoio para a elaboração do trabalho, e a terceira, Considerações Finais do Capítulo, na qual consta um quadro resumo de todas as tecnologias abordadas e suas respectivas versões.

3.1 Ferramentas para Desenvolvimento da Loja On-line

Essa Seção apresenta as ferramentas e tecnologias escolhidas para o desenvolvimento do software. Cabe colocar que optou-se por desenvolver uma Loja On-line, uma vez que a mesma compreende uma típica aplicação web, e permitindo, segundo Gentil (2020), vivenciar desafios que são tópicos de interesse desse estudo exploratório, dentre eles: um software que demanda constante evolução e adequação ao público alvo, com inserção, remoção e atualização de *features* e funcionalidades. Nesse contexto, muitas vezes, necessitando de abordagens adaptativas (GENTIL, 2020), ou seja, que permitem adaptações em tempo de execução, sendo parcial ou totalmente autogerenciáveis. O conceito de abordagens adaptativas não está compreendido no escopo desse trabalho. Entretanto, o estudo exploratório permitiu conhecer o comportamento de uma típica aplicação web sob a perspectiva do uso combinado de Arquitetura Limpa e Arquitetura de *Micro front-end*. Diante das vantagens percebidas, pode ser interessante um estudo futuro sobre abordagens adaptativas nesse cenário combinado de arquiteturas.

3.1.1 React 18

Dentre as diversas ferramentas que auxiliam na construção de aplicações *front-end*, é possível destacar três como sendo de relevância. São elas: React[1], Vue[2] e Angular[3].

Esse trabalho foi construído utilizando o React, uma biblioteca para a criação de interfaces gráficas, cuja utilização não se limita apenas a aplicações Web, mas também *mobile* e até mesmo TVs. O principal objetivo do React é o de minimizar a quantidade de erros no código durante a construção de UIs (*User Interfaces*) através de componentes reutilizáveis (KINSTA, 2023).

[1] https://react.dev. Acessado pela última vez em 23/04/2023.
[2] https://vuejs.org. Acessado pela última vez em 23/04/2023.
[3] https://angularjs.org. Acessado pela última vez em 23/04/2023.

Entende-se por componentes reutilizáveis segmentos da interface gráfica que podem ser reutilizados em diversos contextos sem a necessidade de se duplicar o código, bastando o carregamento do componente aonde for necessário. Pode-se descrever os elementos que compõem uma interface gráfica como: botões, textos, imagens, dentre outros. Sendo assim, qualquer elemento da UI pode ser transformado em um componente (REACT, 2023).

Além disso, o React também é uma aplicação de página única, ou *single-page application*, conforme descrito em Aplicações Single-Page. Isso significa que ao invés de ser enviada uma requisição ao servidor a cada página carregada, todo o conteúdo mostrado em tela é renderizado pelo JavaScript através do React. Essa abordagem garante ganhos de desempenho, pois não são necessários recarregamentos de tela (KINSTA, 2023).

A renderização de elementos na tela, no contexto web, é realizada pelo React DOM[4], usando uma sintaxe chamada JSX ou JavaScript XML. Essa sintaxe permite a combinação de lógica JS (JavaScript) com elementos HTML em um único arquivo, facilitando a organização do código e a criação dos componentes (KINSTA, 2023). Através do JSX, é eliminada a necessidade de interação com a DOM usando os métodos nativos do JS de seleção e manipulação. No Código 5, é mostrado um exemplo de JSX em uma aplicação React (KINSTA, 2023). O trecho de código evidencia um componente nomeado de cabeçalho (em inglês, *Header*), que renderiza na tela o título da loja que está armazenado dentro de uma variável *title*. O exemplo ilustra a utilização de código JavaScript diretamente dentro de *tags* HTML, construído usando a sintaxe JSX.

Código 5 – Exemplo da sintaxe JSX utilizada no React

```
1  export function Header() {
2    const title = "Loja Online";
3
4    return (
5      <div className="header">
6        <h1>{title}</h1>
7      </div>
8    )
9  }
```

Fonte: Autor

A escolha dessa ferramenta originou-se pela facilidade de reutilização de código e a separação de responsabilidades, algo de muita relevância para o contexto do trabalho, além de ainda fornecer funções prontas com funcionalidades específicas conhecidas como *hooks*, que auxiliaram no desenvolvimento da aplicação.

[4] https://legacy.reactjs.org/docs/react-dom.html. Acessado pela última vez em 23/04/2023.

3.1.2 TypeScript 4.4

TypeScript[5] é uma linguagem de programação fortemente tipada, construída sobre o JavaScript, que fornece novas funcionalidades não presentes no JS. Durante o processo de compilação do software, o código TypeScript é convertido em JavaScript, permitindo com que essa aplicação seja executada em qualquer ambiente que suporte o JS.

Entende-se, como o maior benefício do TypeScript, a adição de tipos estáticos ao JavaScript. Isso significa que o TypeScript permite a definição de tipos para variáveis, parâmetros de funções, dentre outros. Assim, todos os tipos definidos na aplicação são verificados em tempo de compilação, permitindo a detecção de erros no código-fonte antes mesmo da execução do código, garantindo maiores robustez e confiabilidade ao software.

A possibilidade de desenvolvimento com segurança de tipos foi o principal fator que motivou a escolha dessa linguagem no trabalho. Com ela, obteve-se subsídios para a construção de uma aplicação mais robusta e confiável, facilitando a manutenção e a evolução do código.

3.1.3 Module Federation

Para facilitar a comunicação e o compartilhamento entre os projetos em uma arquitetura *micro front-end*, foi utilizado o Module Federation[6] que facilita o compartilhamento de código e dependências entre os projetos, expondo os módulos, e permitindo que os mesmos sejam carregados de forma dinâmica, diminuindo a duplicação de código e otimizando a instalação de dependências. A ideia é trabalhar orientando-se por *building blocks*. Portanto, o Module Federation oferece diferentes *plugins*, de baixo nível de abstração e de alto nível de abstração. Um típico *building block* oferecido como *plugin*, e que representa um recurso de baixo nível de abstração, é o *ContainerPlugin*, que auxilia na criação de *container* adicional. Já um exemplo de *building block* oferecido como um *plugin*, e que representa um recurso de alto nível de abstração, é o *ModuleFederationPlugin*, que justamente combina *plugins* auxiliares, de mais baixo nível de abstração, conferindo, adicionalmente, acesso a pacotes de cunho específico (em alto nível de abstração), que permitem o uso e o compartilhamento de bibliotecas orientadas à(s) sintaxe(s) padronizada(s). Essa abordagem orientada a *building blocks*, novamente, facilita reutilização de código, além do gerenciamento de dependências.

3.1.4 Tailwind 3.3.1

Para a estilização dos componentes de UI, utilizou-se o Tailwind[7], que é um conjunto de classes utilitárias de CSS (*Cascading Style Sheets*), facilitando a padronização

[5] https://www.typescriptlang.org/. Acessado pela última vez em 14/07/2023.
[6] https://webpack.js.org/concepts/module-federation. Acessado pela última vez em 23/04/2023.
[7] https://tailwindcss.com. Acessado pela última vez em 23/04/2023.

visual, reutilização de código, além de seguir as melhores práticas de CSS. Dentre essas práticas, podem ser mencionadas: orientação por uma folha de estilo; prevenção quanto à redundância; maior acessibilidade ao CSS, e nomeação dos componentes de UI baseada em convenções. A utilização do Tailwind também auxilia na melhora de desempenho, por analisar o código e remover as classes que não foram aplicadas, diminuindo o tamanho de código CSS que o usuário final precisa carregar ao acessar a aplicação desse trabalho.

3.1.5 Jotai 2.0.4

Jotai[8] permite a construção de um gerenciador de estados globais na aplicação React através de uma abordagem atômica, que seriam pequenos segmentos isolados de estados, os quais combinados formam o estado global da aplicação. Essa ferramenta, em projetos *micro front-end*, permite o compartilhamento do estado entre diferentes aplicações, facilitando a integração e o sincronismo de informações como: carrinho, usuário logado, cupom de desconto aplicado, entre outros.

Outro motivo da escolha dessa ferramenta relaciona-se ao ganho de desempenho, devido à utilização de uma estratégia combinada de átomos e renderizações. Essa estratégia possibilita que um componente só seja renderizado novamente quando alguma dependência relacionada ao mesmo for atualizada. Sendo assim, remove-se a necessidade de algum tipo de memoização interna nos componentes.

3.1.6 SonarQub 10.0

SonarQub[9] é uma ferramenta, criada pela SonarSource, de revisão automática de código autogerenciada que, de forma sistemática, auxilia a manter a qualidade do código adicionado a um projeto dentro dos padrões de qualidade configurados previamente. Através da integração aos fluxos de entrega e integração contínua, a ferramenta realiza inspeções, gerando relatórios com uma análisa quantitativa sobre aspectos como: (i) complexidade ciclomática, uma métrica relacionada ao cálculo do números de caminhos presentes no código; (ii) quantidade de linhas duplicadas no código; (iii) análise da quantidade de problemas de coesão do projeto, e (iv) outros aspectos relacionados a qualidade e manutenabilidade de uma base de código.

Há duas versões da ferramenta de revisão automática de código oferecidas pela SonarSource, as versões SonarQub e SonarCloud. No contexto do trabalho, foi empregado o SonarCloud, sendo uma versão da ferramente disponível na nuvem, com integração fácil a repositórios no Github e sem a necessidade de hospedagem manual. Já o SonarQub oferece a possibilidade de personalização da ferramenta, bem como a hospedagem em infraestruturas particulares.

[8] https://jotai.org. Acessado pela última vez em 24/04/2023.
[9] https://www.sonarsource.com/. Acessado pela última vez em Maio de 2023.

Devido ao seu vínculo com uma comunidade de grande porte que possui o espírito de software livre, chamada Sonar Community[10], a ferramenta mantém-se atualizada às melhores práticas em termos de métricas de qualidade de código, bem como em termos de usabilidade, experiência de usuário, facilidades de configuração e integração com outras ferramentas, através de fórum público e da realização de *webinars*. Com atualizações frequentes[11] e aprimoramentos na ferramenta, novas diretrizes e padrões de análise são introduzidos ou aprimorados, resultando em uma análise mais abrangente e segura.

Este trabalho orientou-se por quatro métricas principais. A intenção foi prover análise, prioritariamente, qualitativa. Cabe colocar que é possível, dado o uso da ferramenta SonarQub, coletar dados quantitativos também. Além disso, e de forma complementar, serão reportados na análise sobre aspectos observados ao longo das atividades de desenvolvimento. As principais métricas que nortearam a análise desse trabalho foram:

- Complexidade ciclomática;
- Duplicação de código;
- Manutenibilidade, e
- Testabilidade.

  A Seção Método de Análise de Resultados (4.5) apresenta descrições sobre cada uma dessas métricas.

A manutenibilidade foi avaliada com o objetivo de verificar a complexidade de modificar, atualizar e corrigir a aplicação proposta. Manutenibilidade é o grau de facilidade com que um sistema ou componente de sistema pode ser modificado para corrigir algum *bug*, atender a novos requisitos ou refatoração, sendo medido pelo quantidade de débitos técnicos existentes.

Vale ressaltar que o débito técnico, no contexto do SonarCloud, é um valor estimado para corrigir os defeitos identificados, que é calculado em dias/horas de trabalho necessários para resolver todos os problemas encontrados.

Critérios como a clareza e a organização do código fonte; a presença de boas práticas de programação, e a modularidade do sistema foram considerados, com o objetivo de assegurar a sua sustentatibilidade a longo prazo.

O SonarCloud atribui cinco notas diferentes para a manutenibilidade de acordo com a qualidade do código do sistema. As notas podem variar de A a E, representando níveis que variam de excelente a insatisfatório.

[10] https://community.sonarsource.com/. Acessado pela última vez em Setembro de 2023.
[11] https://community.sonarsource.com/c/sq/releases/24. Acessado pela última vez em Setembro de 2023.

- A: O código apresenta uma boa manutenibilidade, o que torna possível realizar alterações futuras com menos esforço;

- B: A manutenção do código é razoável, com oportunidades de aperfeiçoamento em alguns aspectos;

- C: A manutenibilidade do código é aceitável, mas há espaço para melhorias significativas;

- D: A manutenibilidade do código está abaixo do recomendado, sendo altamente recomendada a realização de melhorias, e

- E: O código é difícil de manter, requerendo melhorias imediatas.

A testabilidade da aplicação foi avaliada com foco nos testes automatizados da aplicação. Foram identificados aspectos como a cobertura dos testes automatizados e a facilidade de se implementar novos testes.

A duplicação de código da aplicação foi avaliada com intuito de verificar se existia código duplicado nas soluções desenvolvidas e a sua quantidade. O objetivo foi confirmar se a solução apresentada possuia ou não duplicação, o que pode causar um impacto direto na manutenibilidade e qualidade do código.

Além disso, realizou-se uma análise da complexidade ciclomática para avaliar a estrutura do código fonte e o número de dependências cíclicas indesejadas. Para isso, uma análise estática do código serviu para verificar se ocorriam ciclos, com o objetivo de assegurar uma solução sólida (mais coesa e menos acoplada) e de fácil compreensão.

O valor da complexidade ciclomática é calculado, pelo SonarCloud, com base no número de caminhos de um código. Sempre que uma condicional é encontrada, o valor de complexidade aumenta em uma unidade. Em aplicações JavaScript, cada função, método, procedimento, *short-circuit*, conjunção lógica, expressão condicional ternária, *loop*, *try*, *catch* e o caso de um *switch* somam um ao número total de complexidade ciclomática. Quanto maior o número, maior a complexidade necessária para compreender a aplicação.

3.2 Ferramentas de Apoio

Essa Seção fornece uma breve descrição sobre algumas ferramentas de apoio utilizadas durante a realização do trabalho, como ferramentas para prototipação, versionamento de código e implantação do mesmo.

3.2.1 Figma

Figma[12] é uma aplicação para prototipação, design de elementos e criação de jornadas do usuário, podendo ser acessada on-line, sem a necessidade de instalação local. Isso facilita o compartilhamento de protótipos via endereço web. Possui um acervo comunitário de projetos de terceiros que servem como uma boa base de inspiração para criação de novas ideias. Essa ferramenta foi utilizada na criação do protótipo da loja on-line, o qual será apresentado no Capítulo 5 - Estudo Exploratório.

3.2.2 Git 2.40.0

Git[13] é um software de versionamento *open source* e gratuito, que permite organização, trabalho colaborativo, histórico de modificações, retorno a versões antigas e muitas outras funcionalidades. A ferramenta pode ser usada em diversos contextos e, no presente trabalho, foi a ferramenta usada para versionar todos os insumos gerados (ex. Provas de Conceito).

3.2.3 Netlify

Netlify[14] é uma plataforma de hospedagem de aplicações, com plano gratuito, que permite a construção, a criação de ambientes de homologação, além de servir a aplicação em sua rede global. Essa plataforma foi utilizada para a hospedagem da Loja On-line.

3.2.4 Ferramentas Gerais

Por fim, mas não menos importante, e ainda no escopo das Ferramentas de Apoio, cabe mencionar:

- Ferramenta para auxiliar no Quadro Kanban: Trello[15];
- Ferramentas de Comunicação: (i) Telegram[16], para troca de mensagens rápidas entre autor e orientadores; (ii) Slack[17], para manter os rastros das orientações ao longo do trabalho como um todo, sendo essencial para alinhamento entre autor e orientadores, e (iii) Teams[18], para reuniões periódicas (semanais) e chats;

[12] https://www.figma.com. Acessado pela última vez em 23/04/2023.
[13] https://git-scm.com/. Acessado pela última vez em 23/04/2023.
[14] https://www.netlify.com/. Acessado pela última vez em 23/04/2023.
[15] https://trello.com/. Acessado pela última vez em Junho de 2023.
[16] https://telegram.org/. Acessado pela última vez em Maio de 2023.
[17] https://slack.com/intl/pt-br. Acessado pela última vez em Maio de 2023.
[18] https://www.microsoft.com/pt-br/microsoft-teams/group-chat-software. Acessado pela última vez em Maio de 2023.

- Ferramenta de Modelagem: principalmente, a LucidChart[19], para modelagens pontuais, seja na notação BPMN, conforme constam nos processos apresentados no Capítulo 4 - Metodologia; seja na notação UML, pela necessidade de modelos estáticos e dinâmicos na representação de nuances das arquiteturas estudadas, e
- Ferramenta de Escrita de Texto: uso do Latex[20].

[19] https://www.lucidchart.com/pages/pt/landing. Acessado pela última vez em Maio de 2023.
[20] https://www.latex-project.org/. Acessado pela última vez em Maio de 2023.

3.3 Considerações Finais do Capítulo

Esse capítulo apresentou uma breve descrição das ferramentas e tecnologias utilizadas no desenvolvimento da Loja On-line, além de suas respectivas versões. No intuito de conferir um resumo sobre as tecnologias utilizadas, segue a Tabela 1.

Tabela 1 – Tecnologias utilizadas para o desenvolvimento da Loja On-line

Nome	**Descrição**	**Versão**
React	Biblioteca para criação de componentes Web	18
TypeScript	Superconjunto de JavaScript	4.4
Module Federation	Ferramenta para compartilhamento de módulos	-
Tailwind	*Superset* de classes CSS	3.3.1
Jotai	Gerenciador de estado global em aplicações React	2.0.4
SonarQub	Ferramenta de análise de código automatizada	10.0
Figma	Aplicação para prototipação	-
Git	Ferramenta de versionamento	2.40.0
Netlify	Plataforma de hospedagem de aplicações	-

Fonte: Autor

4 Metodologia

Este capítulo apresentará o método usado no desenvolvimento teórico e prático do trabalho. Inicialmente, serão abordados aspectos relevantes para a Classificação da Pesquisa, tais como a Abordagem adotada, a Natureza do estudo, os Objetivos propostos e os Procedimentos adotados. Em seguida, será apresentado o Método Orientado a Provas de Conceito, que é o fundamento metodológico do trabalho, enfatizando a sua relevância para o trabalho e as etapas de sua implementação. Depois, será discutido o Método de Desenvolvimento do Estudo Exploratório, destacando-se as metodologias ágeis e sua aplicação prática no trabalho. Também será apresentado o Método de Análise de Resultados descrevendo seus procedimentos e características. Além disso, serão apresentados os Fluxos de Atividades/Subprocessos e Cronogramas de Atividades/Subprocessos, tanto para o escopo da primeira etapa, quanto para o escopo da segunda etapa do TCC. Por fim, serão fornecidas as Considerações Finais do Capítulo, resumindo todos os tópicos abordados no mesmo.

4.1 Classificação da Pesquisa

De acordo com Gerhardt e Silveira (2009), a metodologia é formada pela junção da palavra *methodos*, cujo significado é organização, com a palavra *logos*, que está relacionada à pesquisa ou investigação. Dessa forma, a metodologia pode ser definida como o estudo da organização ou passo a passo para atingir um objetivo específico. Essa Seção também apresenta, com base nas definições de Gerhardt e Silveira (2009), a classificação da pesquisa deste trabalho em termos de Abordagem, Natureza, Objetivos e Procedimentos. A Tabela 2 apresenta um resumo da classificação deste trabalho.

Tabela 2 – Resumo de classificação da metodologia

Categoria	**Classificação**
Abordagem	Qualitativa
Natureza	Aplicada
Objetivos	Exploratória
Procedimentos	Provas de Conceito

Fonte: Autor

4.1.1 Abordagem

A pesquisa realizada neste trabalho pode ser classificada como **qualitativa**. A abordagem qualitativa tem como objetivo aprofundar a compreensão sobre um tema,

através da explicação do porquê de algo, sem a necessidade de estabelecer valores ou submeter-se à prova de fato, pois os dados desse tipo de abordagem são não-métricos (GERHARDT; SILVEIRA, 2009).

Entretanto, apesar da pesquisa ser predominantemente qualitativa, foram utilizados como insumos dados quantitativos, tais como valores concretos associados às métricas obtidas via SonarQub (ex. complexidade ciclomática e quantidade de linhas duplicadas no código). Com base nesses dados, e nos comportamentos observados ao longo das atividades de desenvolvimento, acordou-se uma análise de viés qualitativo.

4.1.2 Natureza

O presente trabalho pode ser classificado como uma **pesquisa aplicada**, cujo objetivo é gerar conhecimento para aplicação prática, buscando resolver problemas específicos (GERHARDT; SILVEIRA, 2009). Sendo assim, é uma pesquisa de cunho prático buscando mitigar as questões previamente estabelecidas na Questão de Pesquisa.

4.1.3 Objetivos

Como Gerhardt e Silveira (2009) apontam, pode-se classificar pesquisas em três grupos. São eles: (i) a pesquisa exploratória; (ii) a pesquisa descritiva, e (iii) a pesquisa explicativa.

Este trabalho segue os objetivos estabelecidos na **pesquisa exploratória**, o que resulta em maior familiaridade com o problema exposto na seção Justificativa (1.3), de modo a torná-lo mais claro ou formular hipóteses a respeito.

4.1.4 Procedimentos

O presente trabalho é caracterizado por uma pesquisa com base em provas de conceito, conforme abordado na Seção Método Orientado a Provas de Conceito (4.3), na qual se busca investigar um tema específico, criando diversas provas de conceito, que são descritas neste trabalho como desafios, a fim de demonstrar as características de um software utilizando os conceitos estabelecidos pelas bases científicas.

Com esses desafios, o programador/especialista tem a oportunidade de compreender de forma mais aprofundada as nuances percebidas ao longo do processo, evidenciando pontos positivos e pontos negativos, dentre outros aspectos. Adicionalmente, como procedimento complementar, foi utilizada Pesquisa Bibliográfica, cujos detalhes encontram-se em Método Investigativo.

4.2 Método Investigativo

De acordo com Gil et al. (2002), após a definição do tema, a pesquisa bibliográfica preliminar tem como objetivo proporcionar uma maior familiaridade do pesquisador com a área de estudo em questão. Essa etapa é relevante, pois auxilia na definição do escopo do trabalho a ser desenvolvido. Dessa forma, após a definição do tema deste trabalho, realizou-se uma pesquisa bibliográfica nas bases científicas. A Tabela 3 apresenta as *strings* de busca que foram usadas durante essa etapa.

Tabela 3 – *Strings* de busca utilizadas na pesquisa bibliográfica

Base	***String* de busca**	**Nº de artigos obtidos**
IEEE	"micro-frontend"	2
IEEE	"frontend"AND "architecture", 2010 - 2023	280
IEEE	"clean architecture"	8
IEEE	"front-end"AND "architecture", 2010 - 2023	2.137
IEEE	"software"AND "clean"AND "architecture"	209
IEEE	"microservices"AND "monolithic architecture"	3
ACM	"front-end"AND "clean architecture"	6
ACM	"front-end"AND "architecture", 2010 - 2023	7.359
ACM	"micro-frontend"AND "architecture", 2010 - 2023	3
ACM	"micro-frontend"	8
ACM	"microservices"AND "monolithic architecture"	110
Science Direct	"micro-frontend"	2
Google Acadêmico	"microfrontend"	232
Google Acadêmico	"clean architecture"AND "frontend", 2019 - 2023	209
Google Acadêmico	"monolithic architecture"	97

Fonte: Autor

4.2.1 Critérios de Seleção

A seleção do material foi realizada por meio de uma análise exploratória, que consistiu na revisão dos resumos e das palavras-chave dos trabalhos encontrados nas bases científicas, seguindo os seguintes critérios de seleção:

- Relacionado à Arquitetura Limpa e/ou *Micro Front-end*;
- Relacionado à arquitetura de aplicações web;
- Relacionado à arquitetura de software;
- Relacionado ao desenvolvimento de aplicações *front-end*, e
- Disponíveis gratuitamente.

A falta de artigos nas bases científicas que atendam aos critérios de seleção, conforme listado na Tabela 3, pode ser explicada pelo fato de que o tema Micro Front-end tem ganhado relevância apenas nos últimos anos, e sua combinação com a arquitetura limpa é praticamente inexistente.

Em relação aos últimos anos, houve um grande aumento no interesse por abordagens para o desenvolvimento de software mais modulares e escaláveis. Nesse contexto, Micro Front-end tem se mostrado uma solução para lidar com os desafios de desenvolvimento de interfaces de usuário em aplicações web complexas. Contudo, sua popularidade ainda está em crescimento, o que pode explicar a falta de material científico disponível sobre o tema.

Ressalta-se que a associação específica de Arquitetura Limpa com *Micro Front-end*, que tem como objetivo separar as preocupações e manter um código limpo e modular, ainda é um campo de estudo relativamente recente. A aplicação da Arquitetura Limpa ao Micro Front-end, de modo a obter os benefícios de ambas as abordagens, ainda não está amplamente estudada pela comunidade científica, com base no levantamento realizado pelo autor desse trabalho.

Sendo assim, é compreensível que a variedade de artigos que abordem a combinação de Arquitetura Limpa com *Micro Front-end* seja limitada. A recente emergência do tema *Micro Front-end*, aliada à popularidade da Arquitetura Limpa, torna a combinação de ambas um campo de conhecimento interessante de ser explorado, sendo esse nicho a contribuição de estudo desse trabalho.

4.2.2 Resultados

Após uma análise prévia, alguns artigos selecionados orientaram a pesquisa do trabalho, bem como suas referências. São eles:

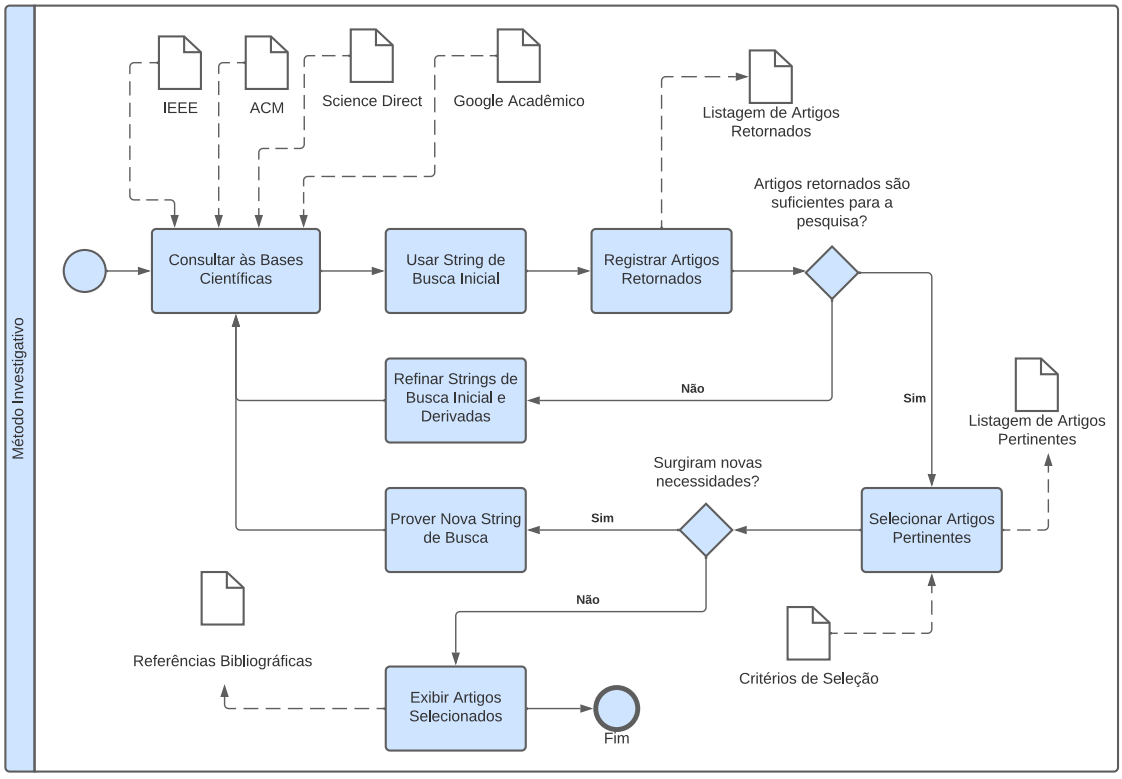
- Microfrontends com Web Components e WebPack: Uma abordagem de implementação agnóstica em relação ao *framework* (QUEIROZ, 2022);
- *An Exploratory Study of Micro Frontends* (MONTELIUS, 2021), e

- Um guia para micro frontends: estudo aprofundado e aplicação de caso de uso no *ecommerce* (KOJO, 2021).

A visão resultante do processo de Pesquisa Bibliográfica consta no Referencial Teórico, com foco para os tópicos: Arquitetura de Software, Aplicações Web, e Desenvolvimento Web e *Micro Front-end*. Há detalhamento em cada um desses tópicos, procurando obter adequada fundamentação para a realização do trabalho.

Complementarmente, ocorrem estudos, usando como base a literatura especializada, visando atendimento aos desafios acordados nas provas de conceito. Nesses casos, o processo investigativo será o mesmo descrito anteriormente: consulta às bases científicas e afins usando *strings* de busca; ajustes nas *strings* de busca para atendimento das principais palavras-chave; recuperação e registro dos artigos retornados; seleção dos artigos retornados com base em critérios, e exibição clara dos materiais utilizados para embasamento da solução que mitiga o desafio. Um resumo desse subprocesso, que representa as atividades do método investigativo, é ilustrado via modelagem BPMN, na Figura 4.

Figura 4 – Modelagem do método investigativo



Fonte: Autor

4.3 Método Orientado a Provas de Conceito

De acordo com Prasanna et al., (2021), uma Prova de Conceito (em inglês, *Proof of Concept* - POC), pode ser uma ferramenta muito apropriada para demonstrar a capacidade e o potencial de um software orientando-se pelas necessidades dos clientes/usuários e/ou do público alvo. Os autores ainda comentam sobre a aplicação de provas de conceitos em diferentes domínios, incluindo os não relacionados à área de software. No desenvolvimento de software, a ideia é provar sobre um conceito, um desafio, uma teoria, uma premissa, via o desenvolvimento de uma solução computacional. Nesse sentido, sugere-se o uso de um breve roteiro, já apresentado nesse trabalho com adaptações ao contexto de interesse dessa pesquisa.

Basicamente, optou-se por uma etapa inicial, conhecida como Definição do Desafio. Nesse momento, o foco foi conferir uma visão clara sobre o desafio a ser abordado na prova de conceito. Essa apresentação foi realizada de forma, exclusivamente, textual, ou ainda complementada com imagens (normalmente, modelagens) e códigos-exemplo.

Na sequência, embasou-se na literatura para prover uma construção sólida da solução ao desafio apresentado. Nesse sentido, ocorreu uma etapa, chamada Estudo da Literatura, na qual apresentou-se uma contextualização, conferindo uma orientação fundamentada na literatura especializada, visando uma solução mais coerente para o desafio. Esse estudo de cunho investigativo conferiu um olhar mais teórico, conceitual, mas também técnico, com levantamento de linguagens de programação e tecnologias adequadas para maior solidez da solução.

Uma vez compreendida a visão da literatura, seguiu-se com a etapa preocupada em apresentar a solução, chamada Apresentação da Solução. Nesse caso, fez-se uso de imagens, código e outros recursos mais específicos, no intuito de prover clareza à exposição dessa solução. Nesse contexto, entendeu-se como solução algo embasado na literatura, e que não necessariamente resolve de forma completa, permanente o desafio. Uma solução, na verdade, atende/mitiga o desafio, de forma satisfatória, e com base em orientações fundamentadas na literatura especializada.

A satisfação, por ser um critério qualitativo e subjetivo, foi "mensurada" com base na percepção do especialista e orientada à visão consenso na área de software (ou seja, à visão predominante). Por exemplo, supondo que a solução incorre em uma Complexidade Ciclomática, na análise estática de um dado método, em um valor compreendido entre 1 e 10. Segundo a área de software (WATSON; WALLACE; MCCABE, 1996), esse método pode ser considerado simples, e de baixo risco. Já se o valor está compreendido entre 51 ou mais, trata-se de um método de alto risco e bastante instável. Cabe ressaltar que a referência de base para tratamento do valor de Complexidade Ciclomática pode parecer antiga, datada de 1996, mas ainda é vastamente utilizada na área, a exemplo de Portais

de Informação como Microsoft Visual Studio[1], que fazem menção aos autores Watson e McCabe, na Seção "O Número Mágico".

Por fim, teve-se a preocupação de conferir uma análise sobre os resultados obtidos, em uma etapa chamada Análise de Resultados. Essa análise foi qualitativa, com base em percepções e comportamentos observados pelo programador/especialista, ao longo do processo de desenvolvimento da solução. Mas, ainda teve um viés quantitativo, usando insumos como métricas e outros parâmetros mais concretos, oriundos, por exemplo, de verificações realizadas via ferramentas de teste. O importante é que esse relato do programador possibilita entender de forma mais adequada as nuances percebidas ao longo do processo, evidenciando pontos positivos e pontos negativos, dentre outros aspectos.

Quando são utilizadas várias provas de conceito, as abordagens metodológicas conferem graus de liberdade e adequação aos pesquisadores, sendo possível, por exemplo, encontrar na literatura vasto uso de abordagens orientadas a provas de conceito, e que não compartilham do mesmo *modus operandi*.

Nesse trabalho, orientou-se pelo método estabelecido em (QUEIROZ, 2022). Entretanto, Queiroz (2022) trata de forma bem flexível a condução das provas de conceito, apenas estabelecendo que as mesmas são executadas de forma sequencial, sendo uma prova de conceito realizada posteriormente sempre dependente do sucesso da prova de conceito realizada anteriormente. Dessa forma, pode-se dizer que se trata de uma orientação em *Building Blocks* (TURILLI et al., 2016).

Em Turilli et al. (2016), os autores abordam o conceito de *Building Blocks* de forma detalhada e criteriosa, no contexto de Desenvolvimento em Internet das Coisas. No contexto do presente trabalho, fez-se uso dessa abordagem, orientada a *Building Blocks*, no intuito de conduzir um processo em etapas, de forma que só ocorrerá a etapa posterior, caso a anterior esteja concluída com sucesso. Tal estratégia permite modularizar um problema muito maior e mais complexo em pequenas partes, mais tratáveis e compreensíveis. Lembrando que o objetivo geral desse trabalho, conforme especificado no Capítulo 1 - Introdução, compreende: "Estudo exploratório via Provas de Conceito, revelando os comportamentos observados ao longo do desenvolvimento de uma loja on-line, e orientando-se pela combinação de Arquitetura Limpa e *Micro front-ends*.". Nesse sentido, trata-se de um desafio complexo.

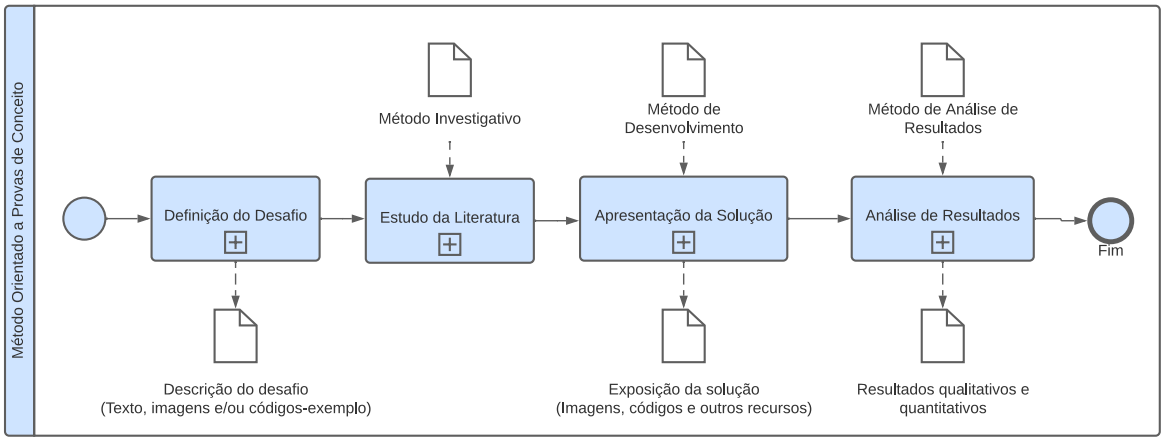
Portanto, a modularização em pequenas partes (i.e. provas de conceitos organizadas em etapas - Definição do Desafio, Estudo da Literatura, Apresentação da Solução e Análise de Resultados), e a condução do processo em *Building Blocks* conferiram uma estratégia pertinente para exposição do estudo exploratório como um todo. Caso uma prova de conceito não fosse bem sucedida, ao ponto de incorrer no não cumprimento das

[1] https://learn.microsoft.com/pt-br/visualstudio/code-quality/code-metrics-cyclomatic-complexity. Acessado pela última vez em 01 Junho 2023.

próximas provas de conceito, ajustes foram realizados, de forma pontual e justificada, e revelando os contratempos e/ou momentos de acerto.

Na Figura 5, há uma modelagem em BPMN, ilustrando as principais etapas estabelecidas para condução do Método Orientado a Provas de Conceito.

Figura 5 – Modelagem do método orientado a provas de conceito



Fonte: Autor

4.4 Método de Desenvolvimento

O método aplicado ao desenvolvimento prático da Loja On-line foi escolhido para combinar dois elementos das metodologias ágeis, o Scrum e o Kanban, por se adequarem bem ao contexto de desenvolvimento de software. Essa abordagem combinada de Scrum e Kanban foi adequada à realidade do trabalho, como será apresentado a seguir.

A escolha do Kanban foi feita para permitir uma visualização clara do progresso do projeto, além de possibilitar a priorização de tarefas e a identificação de gargalos. De acordo com Kanbanize (2023), o Kanban é uma metodologia ágil de gestão de projetos que se baseia em um quadro de tarefas, no qual as tarefas são movidas de uma coluna para a outra de acordo com o seu progresso.

Kanbanize (2023) ainda explica que o Kanban é composto, na sua forma mais simples, por três colunas: *To Do*, *Doing* e *Done*. A coluna *To Do* representa as tarefas que ainda não foram iniciadas. A coluna *Doing* representa as tarefas que estão em desenvolvimento. A coluna *Done* representa as tarefas que já foram concluídas. Além disso, o Kanban oferece uma certa flexibilidade e adaptabilidade, permitindo ajustes de prioridade, adequando, dessa forma, às mudanças de requisitos e necessidades do projeto.

Sobre o Scrum, utilizou-se, sobretudo, o *Product Backlog*, que é uma lista de requisitos que pode ser modificada conforme o desenvolvimento do projeto avança (DRUMOND, 2023). Contudo, preferiu-se uma abordagem mais leve em processos em comparação ao

Scrum tradicional, não utilizando papéis ou rituais, como reuniões diárias e retrospectivas. Mas, na versão combinada com o Kanban, o Quadro Kanban, especificado via Trello, representa o *Sprint Backlog* a cada momento do projeto.

Ressalta-se ainda que o método de desenvolvimento, aqui descrito, foi usado de forma conjunta ao Método Orientado a Provas de Conceito, já abordado anteriormente.

4.5 Método de Análise de Resultados

A análise dos resultados deste trabalho foi fundamentada na perspectiva do especialista, que, neste caso, é o autor, que, além de experiência com desenvolvimento de software em geral, possui mais de dois anos de experiência no desenvolvimento de aplicações *e-commerce* de larga escala, criando soluções (em conjunto com equipes especializadas) que impactam milhões de clientes, além da revisão por pares, através da análise realizada por especialistas em desenvolvimento de aplicações web e arquitetura de software. No que diz respeito ao aspecto qualitativo, busca-se compreender a pertinência da solução proposta através da avaliação dos seguintes aspectos: manutenibilidade, testabilidade, duplicação de código e complexidade cliclomática, conforme detalhado na subseção SonarQub 10.0 (**??**).

Com o objetivo de fornecer mais insumos para a validação da solução proposta e permitir que outros especialistas na área pudessem revisar a solução proposta e sua relevância, optou-se pelo uso da técnica de revisão por pares. A revisão por pares foi realizada por meio de um formulário on-line, com perguntas direcionadas à questão de pesquisa do trabalho, com o objetivo de confirmar ou não se a solução proposta atendia à Questão de Pesquisa. Dois revisores especialistas em desenvolvimento de software participaram do processo.

O primeiro revisor, chamado de Revisor A, é um arquiteto de software, sendo responsável por colaborar de forma ativa na definição dos padrões de engenharia de software, em contextos distintos de sua atual empresa, buscando a excelência e a diminuição da complexidade do processo de desenvolvimento. Além disso, tem vasta experiência com a arquitetura hexagonal (Portas e Adaptadores), além de grande conhecimento em design de sistemas e padrões de código. Com mais de sete anos no mercado, já teve a oportunidade de desenvolver software utilizando Arquitetura Limpa, microsserviços e diversas outras arquiteturas de software.

O segundo revisor, denominado Revisor B, com mais de seis anos de experiência no mercado, atua como líder técnico de um grupo de desenvolvedores *front-end*, orientando sobre o desenvolvimento de soluções de alta qualidade, seguindo padrões de código, arquiteturas e design de sistemas. Além disso, o revisor também teve a oportunidade de desenvolver produtos voltados à educação de novos desenvolvedores, além de trabalhar

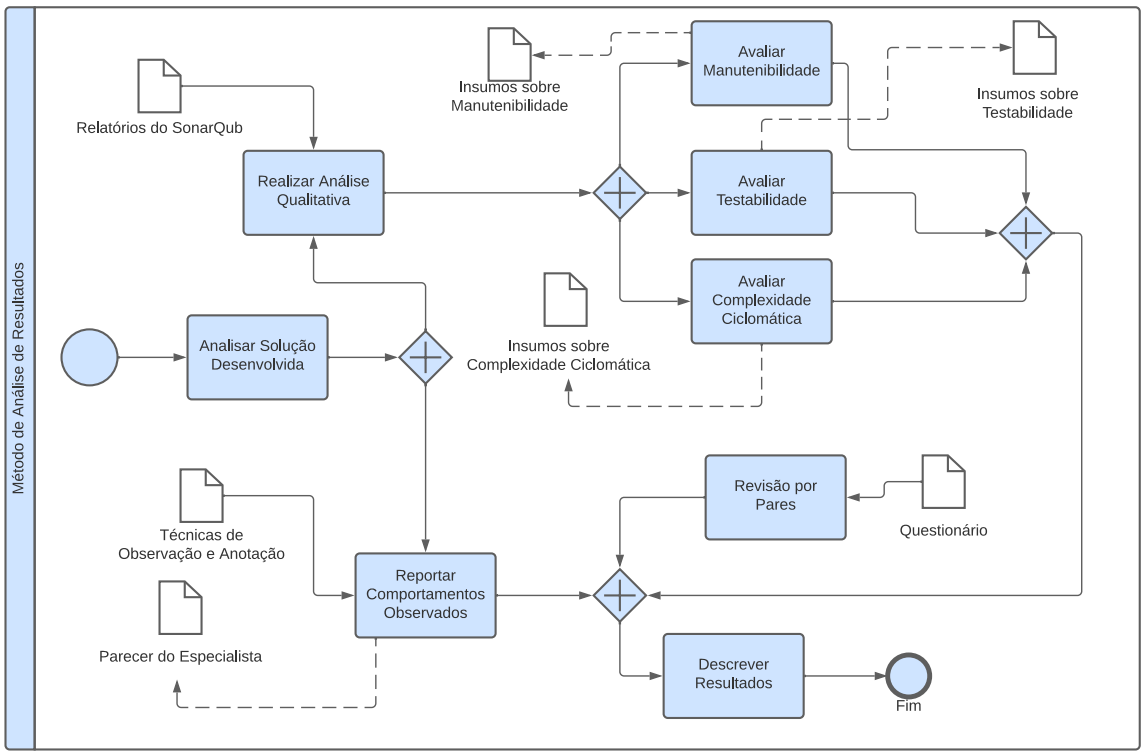
em grandes *e-commerces*.

As respostas fornecidas pelos revisores para cada prova de conceito estão disponíveis no apêndice Questionários de Revisão por Pares.

Esse método de análise de resultados permitiu uma avaliação aprofundada da solução proposta, identificando pontos de melhoria e fornecendo insights relevantes para o trabalho. Ao combinar a avaliação do autor/especialista e demais especialistas na área com os critérios de manutenibilidade, testabilidade, duplicação de código e complexidade ciclomática, obteve-se uma visão abrangente e fundamentada sobre a pertinência do trabalho, e as reais contribuições do mesmo.

A Figura 6 ilustra o método de Análise de Resultados, com foco nas atividades: Avaliar Manutenibilidade, Avaliar Testabilidade, Avaliar Complexidade Ciclomática, Reportar Comportamentos Observados e Revisão por Pares.

Figura 6 – Modelagem do método de análise de resultados



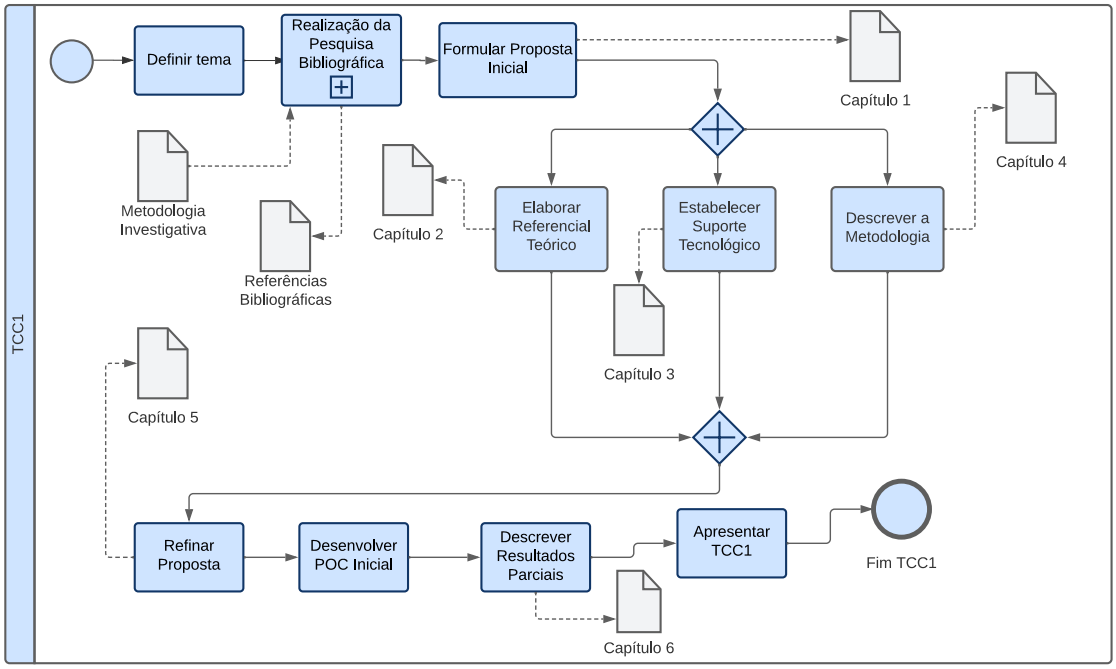
Fonte: Autor

4.6 Fluxos de Atividades/Subprocessos

Ao longo da elaboração do trabalho, foram realizadas diversas atividades com o objetivo de conduzir as demandas de forma coerente e e dentro do prazo estabelecido. Na primeira etapa do TCC, as atividades começam com a definição do tema, até terminarem com a apresentação dos resultados para a banca. Assim, foi estabelecido um plano de atividades específico, conforme descrito a seguir e ilustrado na Figura 7.

4.6.1 Primeira Etapa do TCC

Figura 7 – Fluxo de atividades/subprocesso da primeira etapa do TCC



Fonte: Autor

1. **Definir Tema:** Atividade que teve como objetivo selecionar material relevante sobre o tema escolhido, disponível nas bases científicas, além de fornecer apoio para o pesquisador sobre trabalhos anteriores sobre o tema.
   Status: Concluída;

2. **Realização da Pesquisa Bibliográfica:** Subprocesso consistiu na seleção de material relevante sobre o tema escolhido, disponível nas bases científicas e de acordo com os critérios apresentados na subseção 4.2, além de fornecer apoio para o pesquisador sobre trabalhos anteriores sobre o tema, resultando em todo o embasamento constante nessa monografia, bem como nas Referências Bibliográficas reveladas ao final desse documento.
   Status: Concluído;

3. **Formular Proposta Inicial:** Atividade desenvolvida com o objetivo de elaborar o Capítulo 1 - Introdução, que aborda a Contextualização, a Justificativa, a Questão de Pesquisa e os Objetivos.
   Status: Concluída;

4. **Elaborar Referencial Teórico:** Atividade que resultou no Capítulo 2 - Referencial Teórico, compreendendo embasamento conceitual sobre Arquitetura de Software, Arquitetura Limpa, *Micro Front-ends*, Aplicações Web e aspectos de seu

desenvolvimento.
Status: Concluída;

5. **Estabelecer Suporte Tecnológico:** Atividade para descrever as ferramentas de apoio para o processo de desenvolvimento, pesquisa e escrita, resultando no Capítulo 3 - Suporte Tecnológico.
Status: Concluída;

6. **Descrever a Metodologia:** Atividade para documentar os procedimentos metodológicos utilizados durante a execução do trabalho. Os resultados estão apresentados neste capítulo.
Status: Concluída;

7. **Refinar Proposta:** Atividade que analisou sobre o escopo previamente definido e, caso necessário, realizou alterações. Isso ocorre a partir da melhor compreensão do tema, adquirida ao longo da elaboração das atividades anteriores. Essa atividade resultou no Capítulo 5 - Estudo Exploratório.
Status: Concluída;

8. **Desenvolver POC Inicial:** Atividade para verificar se a proposta era viável, implementando uma prova de conceito. Isso tornou possível identificar possíveis riscos e estabelecer as bases para a implementação efetiva da solução, como protótipos de interface e configuração de ambiente de desenvolvimento. Essa atividade resultou em seções descritas no Capítulo 5 - Estudo Exploratório.
Status: Concluída;

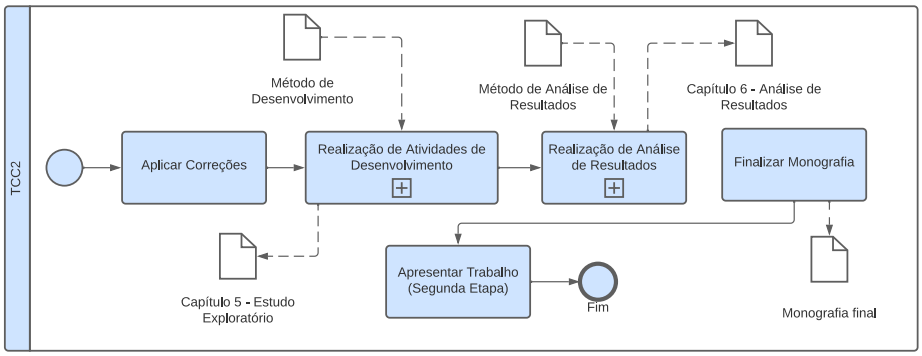
9. **Descrever Resultados Parciais:** Atividade para revelar sobre o estado atual do trabalho antes da apresentação da primeira etapa do TCC, descrevendo os avanços alcançados até aquele momento e as perspectivas para a segunda etapa do TCC.
Status: Concluída, e

10. **Apresentar TCC1:** Atividade que compreendeu a apresentação para a banca do que foi desenvolvido durante a primeira etapa do TCC.
Status: Concluída.

4.6.2 Segunda Etapa do TCC

Nesta etapa do TCC, as atividades começaram com aplicação das correções solicitadas pela banca, seguida pela realização das atividades de desenvolvimento, análise de resultados, finalização da monografia, e apresentação do trabalho para a banca. A Figura 8 apresenta esse plano de atividades específico para a segunda etapa da monografia.

Figura 8 – Fluxo de atividades/subprocessos da segunda etapa do TCC



Fonte: Autor

1. **Aplicar Correções:** Atividade para adequar e corrigir a monografia de acordo com os apontamentos da banca conferidos durante a apresentação da primeira etapa do TCC.
   Status: Concluída;

2. **Realização de Atividades de Desenvolvimento:** Subprocesso para o desenvolvimento da Loja On-line, conforme descrito no Capítulo 5 - Estudo Exploratório. As atividades de desenvolvimento foram realizadas com base no Método Orientado a Provas de Conceito e Método de Desenvolvimento.
   Status: Concluída;

3. **Realização de Análise de Resultados:** Subprocesso que compreendeu realizar uma análise dos aspectos resultantes do desenvolvimento do trabalho. Essa atividade orientou-se pelo Método de Análise de Resultados descrito na Seção 4.5.
   Status: Concluída;

4. **Finalizar Monografia:** Atividade para reportar o estado final da monografia, realçando os objetivos alcançados e descrevendo aspectos passíveis de trabalhos futuros.
   Status: Concluída, e

5. **Apresentar Trabalho (Segunda etapa):** Atividade para a apresentação dos resultados finais alcançados para à banca examinadora.
   Status: Não iniciada.

4.7 Cronogramas de Atividades/Subprocessos

A Tabela 4 apresenta o cronograma de atividades realizadas na primeira etapa do TCC. Já a Tabela 5 foca no cronograma de atividades realizadas na segunda etapa do TCC.

Tabela 4 – Cronograma de atividades/subprocessos da primeira etapa do TCC

Atividades/Subprocessos	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**
Definir tema	X				
Realização de Pesquisa Bibliográfica	X				
Formular Proposta Inicial	X				
Elaborar Referencial Teórico			X		
Estabelecer Suporte Tecnológico		X			
Descrever a Metodologia				X	
Refinar Proposta				X	
Desenvolver POC Inicial				X	
Descrever Resultados Parciais					X
Apresentar TCC1					X

Fonte: Autor

Tabela 5 – Cronograma de atividades/subprocessos da segunda etapa do TCC

Atividades/Subprocessos	**Ago**	**Set**	**Out**	**Nov**	**Dez**
Aplicar Correções	X				
Realização das Atividades de Desenvolvimento	X	X	X	X	
Realização da Análise de Resultados				X	X
Finalizar Monografia					X
Apresentar TCC2					X

Fonte: Autor

4.8 Considerações Finais do Capítulo

Este capítulo tratou sobre as principais decisões que dizem respeito à organização metodológica do trabalho. Em termos de classificação, a pesquisa é de abordagem, predominantemente, Qualitativa; de natureza Aplicada; com objetivos Exploratórios, e conduzida com Pesquisa Bibliográfica e Provas de Conceito como procedimentos. Os elementos de pesquisa do trabalho foram definidos com base em critérios estabelecidos na literatura científica, bem como no método baseado em provas de conceito. Foram contemplados ainda métodos específicos, sendo: (i) investigativo, para condução da Pesquisa Bibliográfica; (ii) de desenvolvimento, combinando Scrum/Kanban, e orientando-se por Provas de Conceito, e (iii) de análise de resultados, com abordagem qualitativa em relação aos critérios de manutenibilidade, testabilidade, duplicação de código e complexidade ciclomática, com os resultados observados e anotados em parecer pelo especialista e suportados pela revisão por pares. Por fim, constam ainda fluxos de atividades em BPMN

e cronogramas, tanto para o escopo da primeira etapa, quanto para o escopo da segunda etapa do TCC.

5 Estudo Exploratório

Este capítulo apresenta o estudo exploratório conduzido nesse trabalho, conferindo Motivação e Contextualização para essa realização. Depois, consta a descrição da Loja On-line, sendo essa a aplicação desenvolvida com base na combinação das arquiteturas Limpa e de *Micro Front-end*. Como o desenvolvimento incorreu no uso de um Método Orientado a Provas de Conceito, além da aplicação de práticas ágeis, são apresentadas as provas de conceito de forma mais detalhada (POC 1, POC 2, POC 3, POC 4 e POC 5). Para cada prova de conceito, há a definição do desafio a ser transposto, além dos principais requisitos inerentes ao desafio. Adicionalmente, são conferidas a apresentação da solução e a análise de resultados, procurando destacar de forma clara sobre cada passo do desenvolvimento: Integração dos *Micro Front-ends*, Desenvolvimento do Catálogo de Produtos, Desenvolvimento do Carrinho de Compras e Desenvolvimento do *Checkout*. Por fim, têm-se as Considerações Finais do Capítulo.

5.1 Motivação

A motivação para a escolha de um tema dentro do contexto de aplicações web teve início há aproximadamente um ano e meio, após o autor iniciar sua jornada como desenvolvedor responsável por uma loja on-line de grande porte, que conquistou mais de 5 milhões de clientes em todo o Brasil. Ao longo dessa jornada, o autor enfrentou desafios relevantes no campo de desenvolvimento de software, desde a refatoração de aplicações legadas até melhorias de desempenho e diversas outras otimizações. Além disso, esse desafio incluiu criar soluções de *Static Site Generation* (SSG) capazes de gerar mais de 12 mil páginas web automaticamente, além de criar soluções de monitoramento de status de páginas web.

No entanto, esse não foi o único desafio enfrentado pelo autor. Ele aceitou, também, o desafio de evoluir e refatorar o painel de parceiros de uma grande empresa de *delivery*. O painel em questão permite que cada parceiro da empresa registre seus produtos, acompanhe as entregas, acesse relatórios de desempenho de vendas, dentre outros. Essa aplicação atende milhões de clientes diariamente, em mais de 180 cidades brasileiras. Além do painel, o autor evoluiu soluções web de *backoffice*, permitindo que a equipe de suporte pudesse atender os clientes e parceiros de forma mais eficiente.

Ao conhecer e lidar com esses desafios no desenvolvimento web, ocorreu a oportunidade de se aprofundar ainda mais nesse cenário, realizando um estudo de cunho exploratório e aplicado no Trabalho de Conclusão de Curso (TCC). Sendo assim, pensou-se em arquiteturas consideradas mais atuais, sendo as escolhidas a Arquitetura Limpa

e Arquitetura de Micro Front-end, com base nas colocações apontadas no Referencial Teórico, com destaque para modularização (maior coesão, menor acoplamento), além de manutenibilidade e testabilidade facilitadas.

Entretanto, há muitas formas de condução de um estudo com esse viés. Nesse sentido, um primeiro desafio para o autor foi ponderar, dentre essas possibilidades, a escolha de uma que permitisse reportar sobre os resultados obtidos de forma a conferir insumos mais concretos para os demais interessados no mesmo tema. Surgiu, assim, a ideia de se especificar provas de conceitos, orientando-se por desafios tipicamente apontados na comunidade de desenvolvedores de software e característicos na construção de uma típica aplicação web.

O principal intuito, dentre outros anseios, é estimular que este trabalho seja consultado pela comunidade, e compreendido como um estudo exploratório, orientado a Provas de Conceito, capaz de revelar os principais comportamentos observados ao longo do desenvolvimento de uma típica loja on-line, usando a combinação de Arquitetura Limpa e *Micro front-ends*. Acredita-se que, dessa forma, seja possível divulgar, conferindo insumos concretos sobre as principais particularidade dessas arquiteturas, em visão combinada. Essas particularidades podem ser comportamentos positivos ou negativos. Quando positivo, o comportamento específico poderá - naturalmente - ser replicado, estimulando a reutilização. Quando negativo, o comportamento específico - provavelmente - provocará reflexões, tanto visando mitigá-lo com alternativas de desenvolvimento, quanto evitando que outros incorram nesses problemas.

5.2 Contextualização

Conforme já acordado, a arquitetura de software tem o objetivo de facilitar a compreensão de um sistema, considerando desenvolvimento, manutenção e implantação. Sendo assim, o processo de definição da arquitetura de um sistema é algo essencial. Se bem especificada a arquitetura, há maiores chances de resultar na construção de sistemas robustos e escaláveis.

Diante disso, destacam-se algumas arquiteturas previamente abordadas, tais como: Arquitetura Monolítica, Arquitetura Portas e Adaptadores, Arquitetura *Model-View-Controller* (MVC), Arquitetura Limpa, dentre outras. Esse trabalho compreendeu o estudo exploratório quanto ao desenvolvimento de aplicações web usando de forma combinada a Arquitetura Limpa e *Micro Front-end*.

O desenvolvimento web tem aumentado nos últimos anos, causado pelos avanços tecnológicos, permitindo com que os usuários consigam acessar aplicações de qualquer lugar e em qualquer momento (BP; ANGGRAINI, 2022). No entanto, o desenvolvimento de aplicações web pode ser um desafio, principalmente quando se trata de aplicações de

maior porte (MONTELIUS, 2021).

Outro ponto interessante de ser colocado é sobre a presença de relatos de aplicações web que começaram pequenas, e foram crescendo ao longo do tempo, tornando-se cada vez mais complexas e difíceis de manter. Essa complexidade pode ser resultado de uma arquitetura mal definida ou, até mesmo, de sua falta (FILHO, 2021).

Diante disso, o uso de arquiteturas que buscam organizar de forma mais coerente e modularizada os componentes arquiteturais presentes em um sistema, como é a proposta da Arquitetura Limpa ou de *Micro Front-end*, surge como um possível caminho visando mitigar os desafios de se manter e evoluir essas aplicações (MONTELIUS, 2021).

5.3 Loja On-line

Com o intuito de prover uma solução que representa uma realidade comum em aplicações web, optou-se pelo desenvolvimento de uma loja on-line, permitindo aplicar os conceitos acordados no Capítulo 2 - Referencial Teórico.

A loja on-line desenvolvida neste trabalho é uma loja de roupas. Nessa aplicação, os usuários podem navegar pelos produtos, adicionando-os ao carrinho, além de finalizar a compra. O escopo de funcionalidades da aplicação não é abrangente, uma vez que pretendeu-se avaliar, dentre outras demandas: se era possível modularizar, testar e manter mais facilmente a aplicação. Sendo a mesma de menor porte ou de maior porte, não importa.

O relevante foi observar, quando implementada uma dada funcionalidade: (i) se era possível tratá-la de forma mais coesa e menos acoplada, ou se demandava usar várias dependências para implementá-la; (ii) se era possível mantê-la/evoluí-la facilmente ao incorporar algo novo, ou se demandava alterar e intervir em muita coisa correlacionada, e (iii) se era fácil testá-la, ou se essa prática era dificultada de alguma forma. Ao lidar com um objeto de estudo de menor escopo, ocorreu a possibilidade de ser mais claro e didático na apresentação dos resultados obtidos. Sendo os comportamentos favoráveis nessa visão preliminar, há também evidente incentivo para realização de um estudo de maior escala, sendo uma excelente proposta de abordagem para trabalhos futuros. Com as percepções coletadas e analisadas, procurou-se responder à questão de pesquisa definida no Capítulo 1 - Introdução.

A loja on-line é composta por quatro *micro front-ends*, sendo eles: Catálogo de Produtos, Detalhes do Produto, Carrinho e *Checkout*. Cada *micro front-end* é responsável por um contexto específico da loja on-line, e foi desenvolvido utilizando a abordagem combinada de Arquitetura Limpa e *Micro Front-end*.

5.4 POC 1 - Integração dos *Micro Front-ends*

Essa Seção descreve a prova de conceito relacionada à configuração do ambiente de desenvolvimento do trabalho, além da integração dos *micro front-ends*. A Seção começa apresentando a Definição do Desafio, seguida dos Requisitos do Desafio. Depois, tem-se a Apresentação da Solução, que precede a Seção final relacionada à Análise de Resultados.

5.4.1 Definição do Desafio

A primeira prova de conceito consistiu em estabelecer o ambiente de desenvolvimento da loja on-line, configurando cada *micro front-end* previsto para a aplicação, além de integrá-los, permitindo com que essa primeira prova de conceito fosse acessada e utilizada pelo usuário como uma única aplicação.

5.4.2 Requisitos do Desafio

Com o intuito de alcançar os objetivos propostos para essa prova de conceito, foram definidos os seguintes requisitos:

- A aplicação deve ser acessível através de um único link, sem a necessidade de acessar cada *micro front-end* separadamente;
- A página de catálogo de produtos deverá ser um *micro front-end*;
- A página de carrinho deverá ser um *micro front-end*;
- A página de detalhes de um produto deverá ser um *micro front-end*, e
- Todos os *micro front-ends* devem conseguir se comunicar entre si, permitindo com que o usuário possa navegar entre as páginas sem perder o estado do carrinho de compras.

5.4.3 Apresentação da Solução

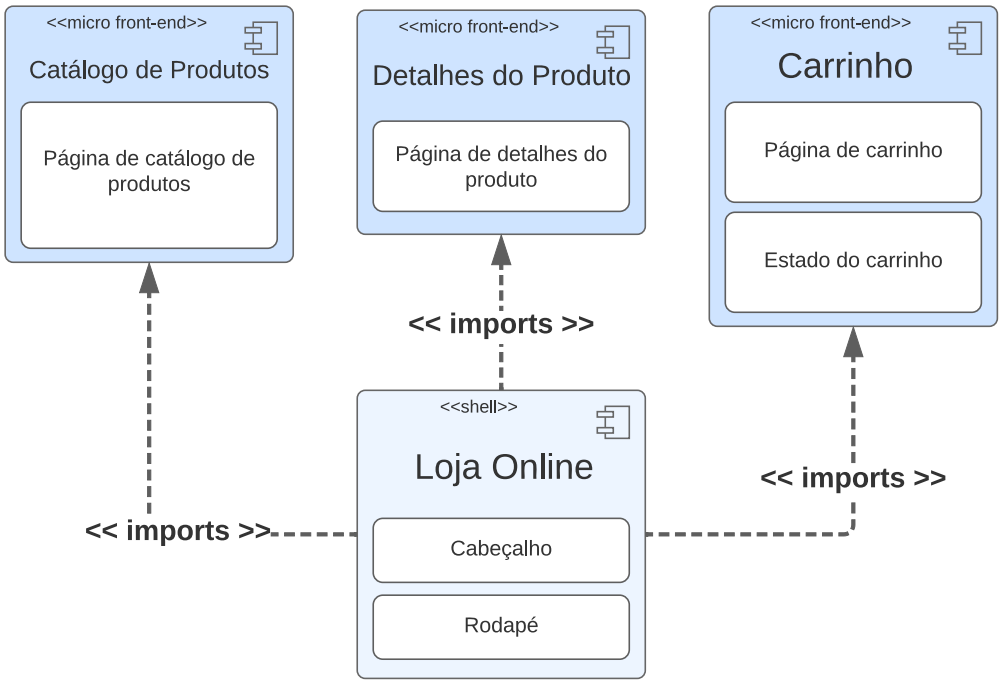
Tem-se, na primeira etapa do desenvolvimento da solução, a configuração do ambiente, criando cada *micro front-end* utilizado pela loja on-line. Para isso, utilizou-se a biblioteca apresentada no Capítulo 3 - Suporte Tecnológico, o React, para a criação das aplicações *client-side*. Outra biblioteca relevante para essa primeira prova de conceito foi a Tailwind CSS, utilizada para construção visual dos componentes presentes em cada aplicação. Por fim, tem-se a configuração do *Module Federation*, permitindo com que cada *micro front-end* seja integrado à aplicação principal e às demais aplicações conforme necessário.

É comum que aplicações *micro front-end* sejam carregadas por uma aplicação principal, comumente chamada de *shell* ou *host*. A responsabilidade dessa aplicação é a

de encapsular todas as outras aplicações menores dentro de um mesmo pacote, fornecendo para o usuário uma experiência contínua, como se fosse uma única aplicação. No trabalho, optou-se por chamar essa aplicação principal de Loja On-line.

Nessa primeira prova de conceito, a aplicação Loja On-line é responsável por importar os demais *micro front-ends*, além de prover dois componentes comuns à aplicação. São eles: o cabeçalho e o rodapé. Além disso, são responsabilidades da Loja On-line definir e gerenciar as rotas da aplicação. A Figura 9 apresenta a organização da solução.

Figura 9 – Solução da primeira prova de conceito



Fonte: Autor

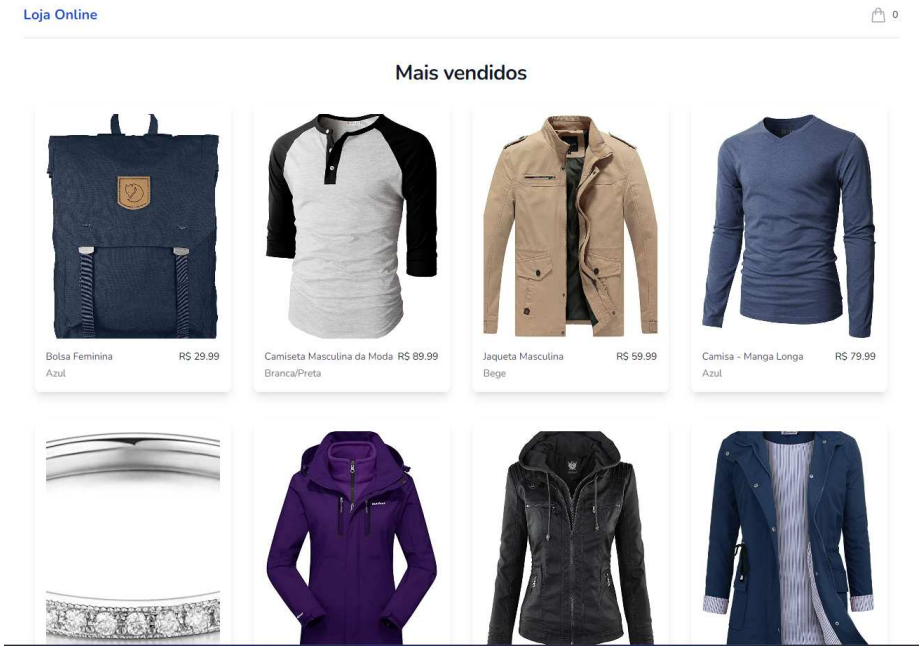
Durante a criação das aplicações *micro front-ends* estabelecidas pelo desafio, buscou-se simular um ambiente real de desenvolvimento, onde cada *micro front-end* é desenvolvido por uma equipe diferente. Para isso, criou-se um repositório para cada *micro front-end*, além de um repositório para a aplicação principal, a Loja On-line. A seguir, encontram-se os endereços para acesso de cada repositório:

- **Loja On-line**: https://github.com/twistershark/tcc-ecommerce-main
- **Catálogo de Produtos**: https://github.com/twistershark/tcc-ecommerce-products
- **Detalhes do Produto**: https://github.com/twistershark/tcc-ecommerce-product-details
- **Carrinho**: https://github.com/twistershark/tcc-ecommerce-cart

5.4.3.1 Protótipo da solução

A Figura 10 apresenta a tela inicial da aplicação, em que são apresentados todos os produtos da Loja On-line. Essa tela é provida pelo *micro front-end* de catálogo de produtos.

Figura 10 – Tela inicial relacionada à primeira POC



Fonte: Autor

Em seguida, a Figura 11 apresenta a tela de detalhes do produto, onde é possível adicionar um produto ao carrinho. O cabeçalho e rodapé de todas as páginas são provenientes da Loja On-line, aplicação essa que envolve todos os outros *micro front-ends*, enquanto que o conteúdo dessa página é fornecido pela aplicação de detalhes do produto.

Por fim, a Figura 12 apresenta a tela de carrinho, onde é possível visualizar todos os produtos que foram adicionados lá, além de remover um produto do carrinho ou alterar sua quantidade. O conteúdo dessa tela é exposto pelo *micro front-end* de carrinho.

As imagens dos produtos utilizados no trabalho são fornecidas por uma API pública chamada *Fake Store API*[1].

5.4.3.2 Configuração do Ambiente

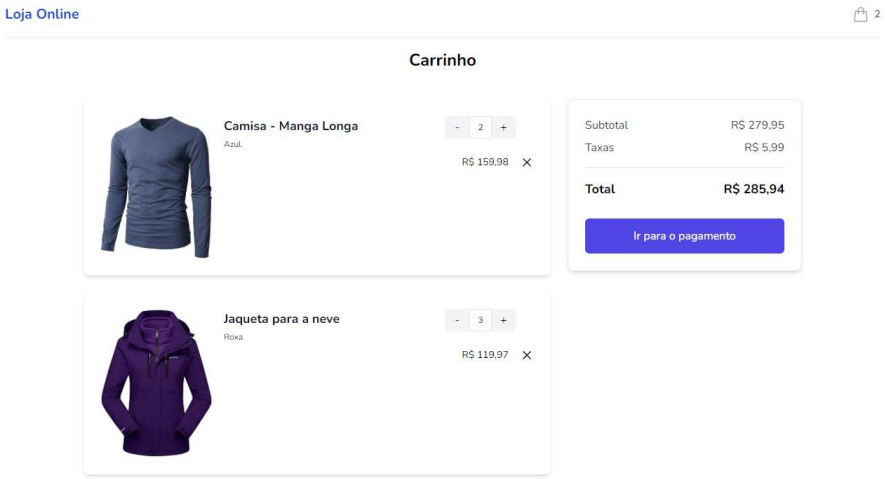
O desenvolvimento dessa solução revelou um aspecto sobre a configuração do ambiente. A utilização do *Module Federation* prevê dois cenários de compartilhamento de componentes. São eles: tempo de execução ou tempo de construção. Devido ao contexto do trabalho, optou-se por utilizar o compartilhamento em tempo de execução, de forma

[1] Disponível em: <https://fakestoreapi.com/>. Último acesso em: Dezembro. 2023

Figura 11 – Tela de detalhes do produto relacionado à primeira POC

Fonte: Autor

Figura 12 – Tela de carrinho relacionado à primeira POC



Fonte: Autor

a permitir com que os *micro front-ends* possam ser desenvolvidos e executados de forma independente, sem a necessidade de serem construídos e importados para a aplicação principal depois de cada modificação realizada. Diante desse contexto de compartilhamento, é relevante ressaltar que existe uma nomeclatura específica para a aplicação que consome um módulo, sendo essa o *host*, e a aplicação que expõe o módulo, chamada de *remote*.

Essa decisão, enquanto facilita o processo de desenvolvimento, dificulta o acesso à tipagem pelas aplicações que estão importando os componentes compartilhados. Isso ocorre porque o TypeScript é executado em tempo de construção, e não em tempo de execução. Para superar esse problema, existem algumas soluções que expõem as tipagens

relacionadas aos módulos de forma automática. Outra alternativa é a de criar um pacote de tipagem, que deve ser importado por todas as aplicações que utilizam os componentes compartilhados. Ou ainda, é possível duplicar essas definições de tipos entre os projetos; contudo, essa solução pode gerar dependência, *bugs* e inconsistências entre as aplicações.

Além do cenário relacionado a tipagens, é importante ressaltar que a integração das diferentes aplicações é realizada através do carregamento dos diferentes módulos em tempo de execução. Isso significa que, caso uma aplicação não esteja disponível no momento em que a aplicação principal for carregada, o usuário pode ter sua experiência prejudicada, uma vez que parte da aplicação não estará disponível. Devido a isso, é comum hospedar as aplicações em servidores que garantam redundância e alta disponibilidade, buscando garantir que as aplicações estejam sempre disponíveis. Tem-se também a necessidade de utilizar uma estratégia de *fallback*, que consiste em prover um componente padrão, caso algum módulo não esteja disponível.

5.4.3.3 Compartilhamento de estado entre os *micro front-ends*

Conforme acordado no Capítulo 3 - Suporte Tecnológico, optou-se pela utilização da Biblioteca Jotai como estratégia para o compartilhamento de estado reativo entre as aplicações. Entende-se por estado reativo aquele que causa algum tipo de efeito colateral quando é modificado. No caso da loja, essa estratégia foi utilizada para compartilhar o estado do carrinho de compras entre as aplicações. Com isso, pode-se perceber que ao adicionar um produto ao carrinho, o número de produtos no carrinho presente no cabeçalho é atualizado. Além disso, a página de detalhes do produto, quando ele já está no carrinho, apresenta o botão de "Ir para o carrinho", ao contrário do botão "Adicionar ao carrinho".

Para alcançar esse resultado, a Biblioteca Jotai disponibiliza um *hook* chamado *useAtom*, que permite com que o estado compartilhado seja acessado e modificado. Dessa forma, aplicações que consomem esse estado podem reagir, renderizando novamente a tela ou realizando diferentes ações, quando o estado é atualizado. Na tela de carrinho, é possível observar um comportamento similar pois, quando a quantidade de um produto no carrinho é atualizada, as informações de preço da página são atualizadas automaticamente. O Código 6 apresenta o módulo responsável por gerenciar o estado do carrinho de compras entre as aplicações.

Código 6 – Módulo responsável por gerenciar o estado do carrinho de compras

```
1  import { atom, useAtom } from "jotai";
2  import { Product } from "../components/Cart/types";
3
4  export type ProductInCart = Product & {
5    quantity: number;
6  };
7
8  const cartAtom = atom([] as ProductInCart[]);
9
10 const useCart = () => useAtom(cartAtom);
11
12 export default useCart;
```

Fonte: Autor

5.4.4 Análise de Resultados

Será apresentado, nessa subseção, os resultados obtidos através do desenvolvimento da solução abordada anteriormente. Com o intuito de prover uma demonstração funcional da configuração de ambiente realizada nessa prova de conceito, desenvolveu-se uma aplicação provisória, que não reflete o protótipo final da loja on-line, o qual será apresentado nas próximas provas de conceito. As seguintes atividades foram concluídas nessa solução: (i) configuração do ambiente; (ii) integração dos *micro front-ends*; (iii) configuração do SonarQub para captura e análise estática do código, e (iv) *deploy* da solução na Netlify. Por fim, a aplicação resultante da solução descrita anteriormente está disponível no seguinte endereço temporário: https://tccpaulo.netlify.app/.

A construção da solução dessa prova de conceito cumpriu com todos os requisitos estabelecidos nos Requisitos do Desafio. São eles:

- A aplicação deve ser acessível através de um único link, sem a necessidade de acessar cada *micro front-end* separadamente;
- A página de catálogo de produtos deverá ser um *micro front-end*;
- A página de carrinho deverá ser um *micro front-end*;
- A página de detalhes de um produto deverá ser um *micro front-end*, e
- Todos os *micro front-ends* devem conseguir se comunicar entre si, permitindo com que o usuário possa navegar entre as páginas sem perder o estado do carrinho de compras.

Através dos resultados obtidos com essa solução, têm-se os seguintes objetivos do trabalho alcançados:

- Identificação, estudo, documentação e aplicação das principais técnicas relacionadas ao desenvolvimento de aplicações web utilizando *micro front-ends*, e
- Integração dos *micro front-ends*, procurando manter suas respectivas independências e responsabilidades.

Conforme acordado na Seção 4.5 (4.5), a análise quantitativa das provas de conceito tem como métricas principais manutenibilidade, testabilidade, duplicação de código e complexidade ciclomática. A aplicação de testes automatizados não estava prevista nessa prova de conceito, sendo requisito para as próximas provas, em conjunto com a aplicação da Arquitetura Limpa.

Diante disso, os dados relacionados à manutenibilidade dos *micro front-ends* são apresentados na Tabela 6. A descrição detalhada sobre manutenibilidade está na subseção SonarQub 10.0 (3.1.6).

Tabela 6 – Resultados quantitativos da primeira prova de conceito

Micro Front-end	**Métrica**	**Nota**
Catálogo de Produtos	Manutenibilidade	A
Detalhes do Produto	Manutenibilidade	A
Carrinho	Manutenibilidade	A
Loja On-line	Manutenibilidade	A

Fonte: Autor

De acordo com a análise do SonarCloud apresentada na Tabela 6, os códigos dos *micro front-ends* desenvolvidos nessa primeira prova de conceito são facilmente manuteníveis, o que torna mais simples a implementação de novas funcionalidades e a correção de possíveis problemas.

5.4.4.1 Aspectos Positivos

Ao longo do desenvolvimento dessa solução, observou-se os seguintes aspectos positivos na utilização de *micro front-ends* na construção de aplicações web, segundo o especialista apresentado na Seção Método de Análise de Resultados (4.5) com experiência na construção de *e-commerces*, corraborado também pelo revisor A, arquiteto de soluções especializado em arquiteturas de microsserviços, hexagonal e outras, e pelo revisor B, líder técnico responsável pelo desenvolvimento de aplicações front-end com experiência no desenvolvimento de *e-commerces* que são utilizados por milhões de usuários. As visões de cada revisor encontram-se apresentadas no Questionários de Revisão por Pares:

- **Separação de responsabilidades**: cada *micro front-end* é responsável por uma parte da aplicação, possibilitando que o desenvolvedor mantenha seu foco em uma parte específica da aplicação, sem preocupar-se com o restante do sistema, e
- **Facilidade na manutenção**: a separação de responsabilidades facilita a manutenção da aplicação, uma vez que cada *micro front-end* é responsável por uma parte específica da aplicação.

5.4.4.2 Aspectos Negativos

Contudo, também foram observados alguns aspectos negativos, segundo a opinião do especialista e corroborado pelos revisores, relacionado a *micro front-ends*:

- **Dependência cíclica**: o *Module Federation* permite que as aplicações compartilhem módulos entre si. Para que esse compartilhamento seja realizado, é necessário que a aplicação que irá consumir os módulos, no caso o *host*, tenha acesso ao endereço da aplicação que está expondo os módulos, a *remote*. Diante desse cenário, o problema surge quando ambas aplicações consomem e expõem módulos umas para as outras. O resultado é uma dependência cíclica, pois para que seja construído a aplicação, é necessário ter acesso ao endereço da aplicação que se pretende consumir. Contudo, em ambiente de produção, esse problema pode não ocorrer, uma vez que cada aplicação estará em um domínio ou subdomínio fixo, de forma que não haja endereços dinâmicos, facilitando a configuração do *Module Federation*;
- **Conflito de classes CSS**: Durante o desenvolvimento da solução, percebeu-se que o Tailwind, em sua última versão, não aplica os estilos previamente importados na aplicação *shell*, a Loja On-line. Por tanto, cada módulo exportado precisa carregar novamente os estilos do Tailwind. Com isso, quando todos os módulos são carregados na aplicação *shell*, ocorre um conflito de classes, uma vez que cada módulo possui classes CSS iguais, sobreescrevendo os estilos e causando problemas visuais nos componentes. Para solucionar esse problema, é necessário adicionar um prefixo para cada classe. A configuração desse prefixo é feita dentro do arquivo de configurações do Tailwind. Após essa modificação, os módulos retornam a funcionar de forma independente e sem conflitos visuais, e
- **Curva de aprendizado**: A configuração do *Module Federation*, mesmo que aconteça em sua maioria apenas no começo do projeto, possui uma curva de aprendizado alta, necessitando dedicação e proatividade para corrigir eventuais problemas que possam surgir mediante o compartilhamento de módulos.

Ainda, o Revisor A, através da revisão por pares, salienta, de acordo com sua experiência como arquiteto de software, que deve-se ter cautela ao planejar a implementação

de uma arquitetura baseada em microsserviços, como os *micro front-ends*, para que os serviços sejam realmente microsserviços e não nano serviços, tendo conhecimento do contexto de aplicação e dos limites de cada um. Além disso, o Revisor destaca a importância de optar por essa solução quando a complexidade da aplicação é tão grande que exija essa abordagem arquitetural. A parecer do Revisor A completo está disponível na Seção Resultado da Revisão por Pares (6.3)

5.4.5 Conclusão

Diante dos aspectos positivos e negativos observados, é possível concluir que a utilização de *micro front-ends* no desenvolvimento de aplicações web é uma abordagem que pode ser aplicada em contextos em que a aplicação apresenta uma grande complexidade, o que requer a separação em módulos menores, com atenção para não realizar uma separação prematura e sem necessidade. Além disso, é necessário que a equipe de desenvolvimento tenha familiaridade com as tecnologias utilizadas, bem como com o *Module Federation*, uma vez que a curva de aprendizado é alta e a configuração inicial pode ser complexa.

5.5 POC 2 - Desenvolvimento do Catálogo de Produtos

Essa Seção descreve o desenvolvimento do *micro front-end* de catálogo de produtos, sendo esse responsável por apresentar os produtos da loja. A Seção é introduzida com a Definição do Desafio, seguida dos Requisitos do Desafio, Apresentação da Solução e por fim a Análise de Resultados.

5.5.1 Definição do Desafio

Espera-se que, além de ser desenvolvida usando a Arquitetura Limpa, essa prova de conceito seja capaz de desenvolver o catálogo de produtos, permitindo ao usuário filtrar os produtos apresentados baseado nas categorias da Loja On-line. Também é esperado que o usuário possa buscar produtos pelo nome.

5.5.2 Requisitos do Desafio

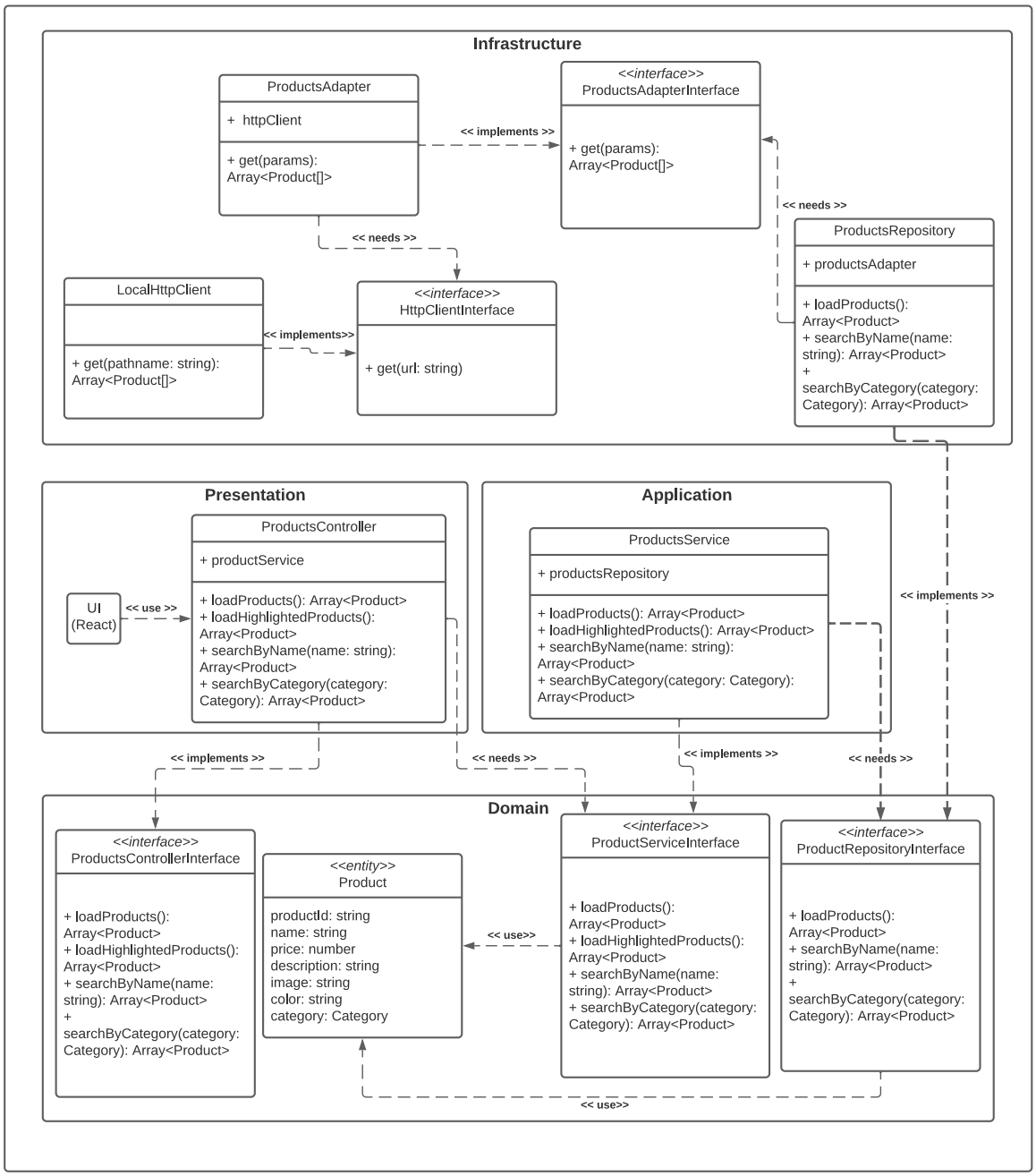
Para que essa prova de conceito seja considerada concluída, os seguintes requisitos devem ser atendidos:

- O usuário deve conseguir visualizar todos os produtos da loja;
- O usuário deve conseguir filtrar os produtos por categoria;
- O usuário deve conseguir buscar por produtos pelo nome, e
- O usuário deve conseguir navegar para a tela de detalhes de um produto, ao clicar em um produto na tela de catálogo de produtos.

5.5.3 Apresentação da Solução

A solução desta segunda prova de conceito teve início com uma análise da literatura sobre Arquitetura Limpa e sua utilização em aplicações *front-end*. Encontrar exemplos coerentes com a literatura sobre Arquitetura Limpa não foi uma tarefa simples. Além disso, a ausência de padrões nas implementações tornou difícil compreender as diferentes camadas previstas na Arquitetura Limpa e como desenvolver cada camada no contexto do trabalho. Dessa forma, obteve-se um aumento no tempo estimado para a implementação da solução. A Figura 13 apresenta a organização em camadas da solução.

Figura 13 – Solução da segunda prova de conceito



Fonte: Autor

Na camada de domínio, estabeleceu-se as entidades e interfaces que representam as regras de negócio da aplicação (MARTIN, 2017). É esperado que essa camada não sofra alterações significativas durante o ciclo de vida do software, uma vez que representa o núcleo da aplicação. Com o objetivo de atender à regra de dependência da Arquitetura Limpa, foram declaradas as interfaces de diversos componentes da aplicação nesta camada de domínio, permitindo que cada camada só dependa das camadas mais internas, de acordo com a inversão de dependências, descrita na Seção Orientação a Objetos (2.3.1).

De acordo com a literatura sobre Arquitetura Limpa, a camada de aplicação está em um nível mais externo à camada de domínio. A camada de aplicação é responsável pela implementação dos casos de uso do sistema. Dessa forma, implementou-se a interface *ProductsServiceInterface* nessa camada, com o objetivo de fornecer os produtos da loja para visualização na camada de apresentação. Os serviços dessa camada têm as responsabilidades de acessar e processar os dados de uma fonte de informação, quando necessário, de acordo com as normas estabelecidas na camada de domínio. É esperado que a implementação da *ProductsServiceInterface* possua métodos que permitam listar todos os produtos, identificar os produtos em destaque, buscar produtos pelo nome e por categoria.

A fim de permitir que a camada de aplicação acesse a camada de persistência sem a necessidade de ser acoplada a outra solução, optou-se por utilizar o padrão de repositórios. Esse padrão de repositórios tem como objetivo separar a regra de negócio da camada de persistência dos dados, tornando mais simples a troca da fonte de dados, caso seja necessário, além de facilitar a manutenção/evolução da aplicação. Dessa forma, a camada de aplicação não precisa saber como os dados são mantidos, apenas que existe um repositório que fornece os dados necessários para a aplicação através de uma interface chamada *ProductsRepositoryInterface*.

A implementação da interface *ProductsRepositoryInterface* depende de um adaptador para a fonte de dados, sendo esse a implementação da interface *ProductsAdapterInterface*. O padrão de adaptadores tem como objetivo fornecer uma forma de comunicação entre duas interfaces distintas. Sendo assim, a camada de persistência pode ser implementada com qualquer tecnologia, desde que seja respeitada a interface *ProductsAdapterInterface*.

A *ProductsAdapter*, implementação da interface *ProductsAdapterInterface*, é responsável por garantir a compatibilidade entre os repositórios e a fonte de dados. Nessa prova de conceito, a *ProductsAdapter* requer uma implementação da interface *HttpClientInterface*, sendo essa a representação de algum cliente HTTP que forneça um método *get* e que possa receber, de forma opcional, os parâmetros de nome e categoria para filtragem.

Com o objetivo de fornecer os produtos de maneira demonstrativa, optou-se por implementar a interface *HttpClientInterface*, retornando dados salvos em memória na aplicação, a fim de evitar a necessidade de uma aplicação externa para o funcionamento desta prova de conceito. É esperado que as alterações na implementação da *HttpClientInterface*, quando necessárias, sejam simplificadas, sem a necessidade de modificar outros aspectos da aplicação, uma vez que nenhuma outra camada tem conhecimento de como os dados estão sendo carregados. As demais camadas da aplicação possuem conhecimento apenas de que receberão dados de acordo com a entidade Produto, não possuindo diferença de onde esses dados estão sendo carregados.

Além disso, é importante salientar que a interface *ProductsControllerInterface* foi

criada para definir a *controller* do contexto de produtos. As *controllers* têm a responsabilidade de tratar as ações do usuário com os serviços da camada de aplicação, que representam as regras de negócio do sistema.

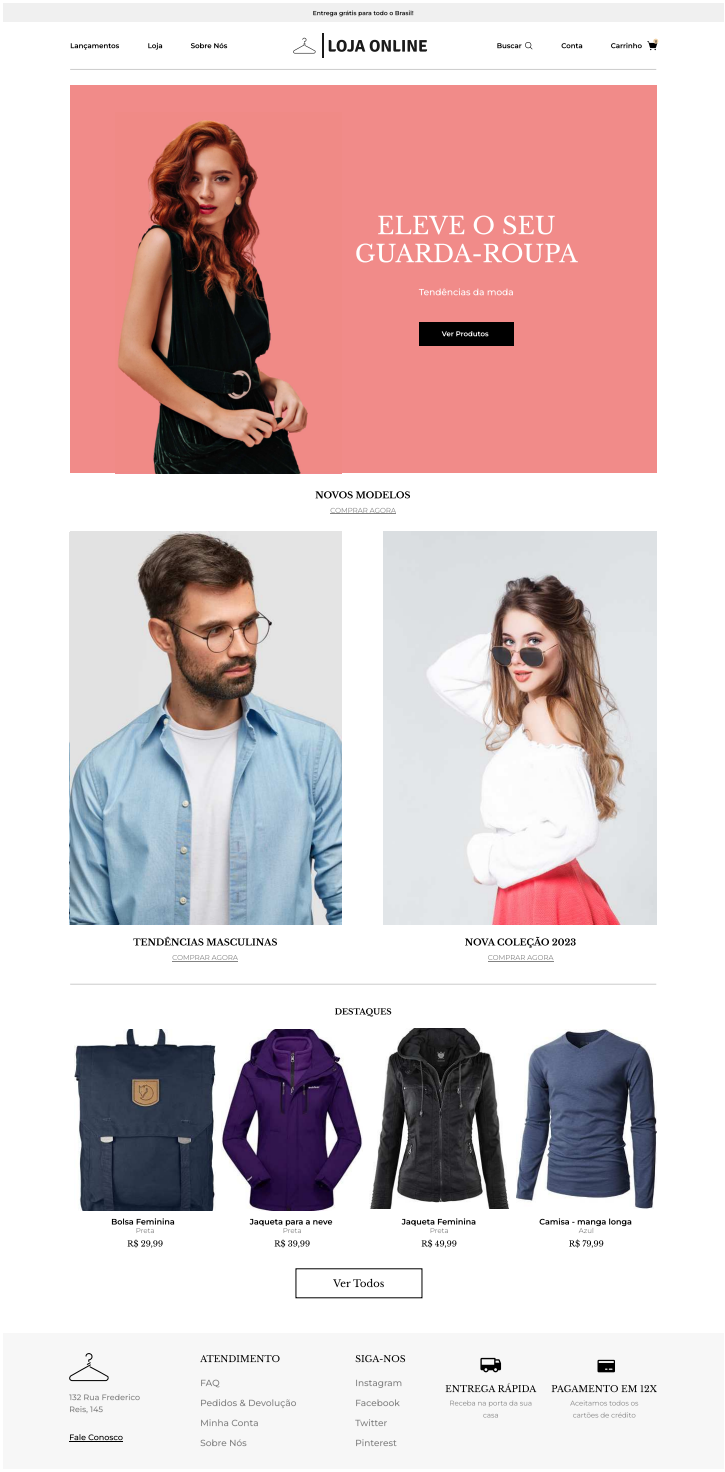
5.5.4 Análise de Resultados

Serão apresentados, nessa subseção, os resultados obtidos através do desenvolvimento da solução abordada anteriormente. As seguintes atividades foram concluídas nessa solução: (i) estudo da literatura sobre aplicações *front-end* com Arquitetura Limpa, e (ii) desenvolvimento da prova de conceito segundo os requisitos estabelecidos.

Em seguida, será provida uma visão geral das telas dessa prova de conceito. Posteriormente, serão apresentados os resultados obtidos em relação aos requisitos definidos para essa prova de conceito, além dos resultados obtidos em relação aos objetivos específicos do trabalho. Por fim, tem-se a descrição da análise quantitativa relacionada a esse segundo desafio.

A Figura 14 apresenta a tela inicial da aplicação, agora na versão final prevista pelo protótipo da Loja On-line. Nela, são exibidos banners com chamativos para ações específicas, com o objetivo de simular uma loja real, onde esses banners seriam úteis para incentivar o usuário a clicar e percorrer um fluxo específico ou encontrar produtos que ainda não conhecia, com base nos produtos mais relevantes da loja apresentados nessa tela.

Figura 14 – Tela inicial da Loja On-line relacionada à segunda POC

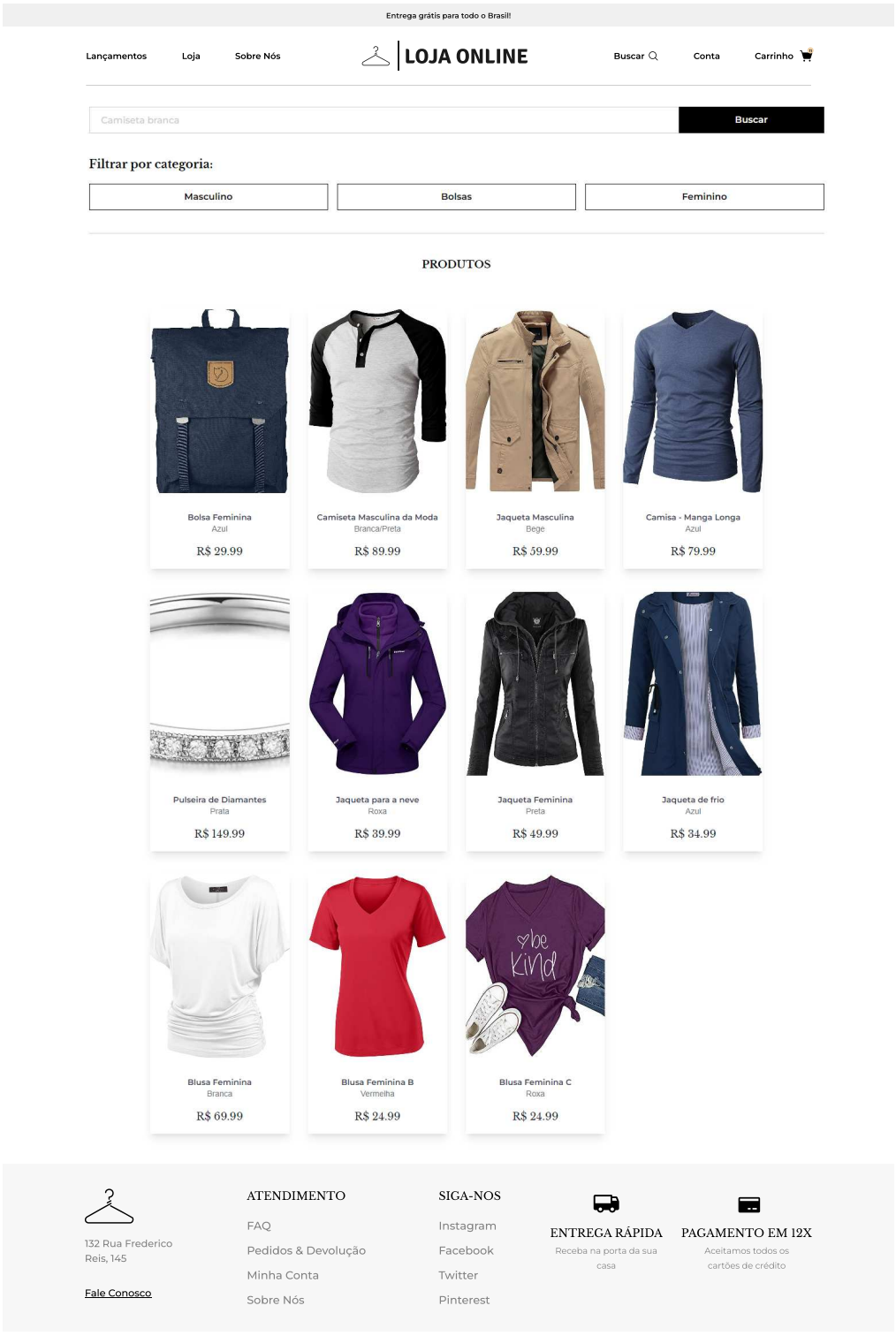


Fonte: Autor

A Figura 15 apresenta, em seguida, a tela de catálogo de produtos, sendo essa responsável por filtrar os produtos por categoria, permitindo que o usuário visualize apenas aqueles que são relevantes para a sua busca. Essa filtragem é feita pela seleção de alguma categoria apresentada na tela. Ao clicar em uma determinada categoria, os produtos lis-

tados são atualizados, exibindo apenas produtos da mesma categoria. Além disso, nesta tela, o usuário pode selecionar produtos pelo nome, o que torna a busca mais rápida e direcionada para produtos específicos.

Figura 15 – Tela de catálogo de produtos relacionado à segunda POC



Fonte: Autor

5.5.4.1 Aspectos Positivos

Os seguintes aspectos positivos na utilização de Arquitetura Limpa, de acordo com o especialista apresentado na primeira prova de conceito, foram percebidos ao longo do desenvolvimento dessa solução:

- **Separação de responsabilidades**: Ao se dividir uma aplicação em diferentes camadas, torna-se mais fácil compreender o que cada uma delas realiza, uma vez que as responsabilidades são únicas e desacopladas;
- **Testabilidade**: Dado que as camadas têm responsabilidades distintas, realizar testes automatizados torna-se simples, uma vez que se pode testar cada uma delas de forma independente, sem a necessidade de testar a aplicação como um todo, e
- **Manutenibilidade**: A divisão de responsabilidade também torna a manutenção da aplicação mais fácil, uma vez que cada camada é responsável por uma área específica da aplicação. Sendo assim, é esperado que a manutenção de uma camada não afete as outras, desde que a interface entre elas seja respeitada.

5.5.4.2 Aspectos Negativos

No entanto, também foram apontados alguns pontos negativos, segundo o especialista, que estão relacionados à utilização de Arquitetura Limpa em aplicações web:

- **Complexidade**: A utilização de Arquitetura Limpa pode aumentar a complexidade de compreensão dos diversos módulos da aplicação, principalmente para quem ainda não está familiarizado com a mesma, uma vez que é necessário criar diversas camadas para separar as responsabilidades. Adicionalmente, é preciso criar interfaces para cada camada, o que aumenta a quantidade de código necessária para a criação da aplicação;
- **Curva de aprendizado**: É necessário que a equipe de desenvolvimento tenha familiaridade com os conceitos de Arquitetura Limpa, bem como com os padrões de projeto e a orientação a objetos, e
- **Tempo de desenvolvimento**: Dada a complexidade e a curva de aprendizado, é esperado que o tempo de desenvolvimento de uma aplicação que utiliza a Arquitetura Limpa seja maior do que a aplicação que não adote essa arquitetura.

A construção da solução dessa prova de conceito cumpriu com todos os requisitos estabelecidos nos Requisitos do Desafio. São eles:

- O usuário deve conseguir visualizar todos os produtos da loja;

- O usuário deve conseguir filtrar os produtos por categoria;
- O usuário deve conseguir buscar por produtos pelo nome, e
- O usuário deve conseguir navegar para a tela de detalhes de um produto, ao clicar em um produto na tela de catálogo de produtos.

Através dos resultados obtidos com essa solução, têm-se os seguintes objetivos do trabalho alcançados parcialmente:

- Identificação, estudo, documentação e aplicação dos principais conceitos relacionados à Arquitetura Limpa em *micro front-ends*, e
- Condução e documentação do processo de desenvolvimento, orientando-se por Provas de Conceito.

No decorrer do desenvolvimento, procurou-se manter o código do desafio dentro dos padrões, de acordo com o perfil de verificação do SonarCloud para aplicações Javascript/Typescript.

A Figura 16 apresenta um histórico de problemas no código, como *bugs* e falhas de segurança, ao longo do desenvolvimento dessa segunda prova de conceito. Ao analisar o histórico, é possível notar que, inicialmente, a aplicação não apresenta problemas, como demonstrado pela análise da primeira prova de conceito, na qual as aplicações obtiveram a maior nota em termos de manutenibilidade. No final do desenvolvimento das duas telas que estão relacionadas a esta prova de conceito, há um pico, indicando a existência de cinco ocorrências de problemas no código. O gráfico também demonstra que após os *commits* relacionados à correção desses problema, a aplicação retornou ao seu estado inicial, sem apresentar nenhum problema a ser corrigido, conforme a análise do SonarCloud.

Alguns dos problemas foram resolvidos ainda no decorrer do desenvolvimento. mas outros necessitaram de *commits* específicos para sua solução. Esses *commits* foram:

- *Commit* [2]:

  Tipo de Problema: Confiabilidade.

  Contexto: O método *get* da classe *MockHttpClient* estava recebendo um parâmetro chamado *pathname*, mas não estava sendo utilizado esse parâmetro.

  Problema: Não usar variáveis que foram iniciadas pode significar que algum trecho de código não foi implementado de forma correta ou completa, o que pode causar problemas inesperados.

[2] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/809f2948f98d3068bd1660718d4844471159f70c>. Último acesso em: Novembro. 2023

Solução: Para essa prova de conceito, isso não é um problema, pois os dados dos produtos estão armazenados em memória, o *endpoint* que seria usado em uma integração com API torna-se inútil. No entanto, para remover o alerta sobre este problema, foi adiciona um *underscore* no início do nome da variável para indicar que ela não será utilizada, sendo essa prática recomendada pelo SonarCloud.

- *Commit* [3]:

  Tipo de Problema: Manutenibilidade.

  Contexto: A lista de produtos renderizados na página de catálogo não continha a propriedade *key* informada.

  Problema: O React utiliza a propriedade *key* para identificar quais itens foram modificados, adicionados ou removidos da lista. A ausência de uma propriedade key pode causar problemas de renderização, prejudicando o desempenho da aplicação e causando possíveis *bugs*.

  Solução: Foi adicionada a propriedade *key* utilizando o identificador de cada produto como valor.

- *Commit* [4]:

  Tipo de Problema: Manutenibilidade.

  Contexto: Não foi declarado que as propriedades recebidas em determinados componentes eram somente de leitura.

  Problema: As propriedades dos componentes devem ser de apenas leitura, pois ajudam a impor o princípio da imutabilidade nos componentes funcionais no React. Isso torna as propriedades somente para modo leitura, garantimos que os dados passados de um componente pai para um componente filho não sejam modificados diretamente pelo componente filho.

  Solução: O utilitário do TypeScript chamado Readonly foi adicionado para definir que as propriedades seriam somente de leitura.

Já a Figura 17 apresenta o histórico de cobertura de testes. Realizou-se um total de quinze *commits* com o objetivo de atingir o nível de 100% de cobertura de código nesse desafio. Em geral, todos os commits foram dedicados à criação de testes para módulos do sistema. Foram eles:

[3] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/68cd95907b435c351bbfaaf62c231a3317fbbd90>. Último acesso em: Novembro. 2023

[4] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/68cd95907b435c351bbfaaf62c231a3317fbbd90>. Último acesso em: Novembro. 2023

Figura 16 – Histórico de problemas de código relacionado à segunda POC

Fonte: Autor

- *Commit* [5]: Criação de testes automatizados para o módulo Products Controller;
- *Commit* [6]: Criação de testes automatizados para o módulo LocalHttpClient;
- *Commit* [7]: Ajuste de nome de pasta de *mocks*;
- *Commit* [8]: Ajuste de configurações da ferramenta de testes, instalação de biblioteca para criação de dados falsos para testes, criação de *mock* de produtos, criação de *mock* e testes automatizados do módulo ProductsRepository, criação de *mock* para o módulo ProductsService, criação de *mock* e testes automatizados para o módulo ProductAdapter, criação de *mock* para o módulo HttpClient, criação de *mock* para o módulo ProductsService e ajustes no testes do módulo ProductsController;
- *Commit* [9]: Criação de testes automatizados para o componente de botão;

[5] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/f44a9a0644ca4868db69420c552ff21e441e9570>. Último acesso em: Novembro. 2023

[6] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/32a8345b7472a1271753dc87a111826ba6e8ec48>. Último acesso em: Novembro. 2023

[7] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/c016a729a991b7fd48a3e6644db19d8688b82d16>. Último acesso em: Novembro. 2023

[8] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/c016a729a991b7fd48a3e6644db19d8688b82d16>. Último acesso em: Novembro. 2023

[9] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/2fc10f6118f3e716228b0e1fe08da38001b4c6f7>. Último acesso em: Novembro. 2023

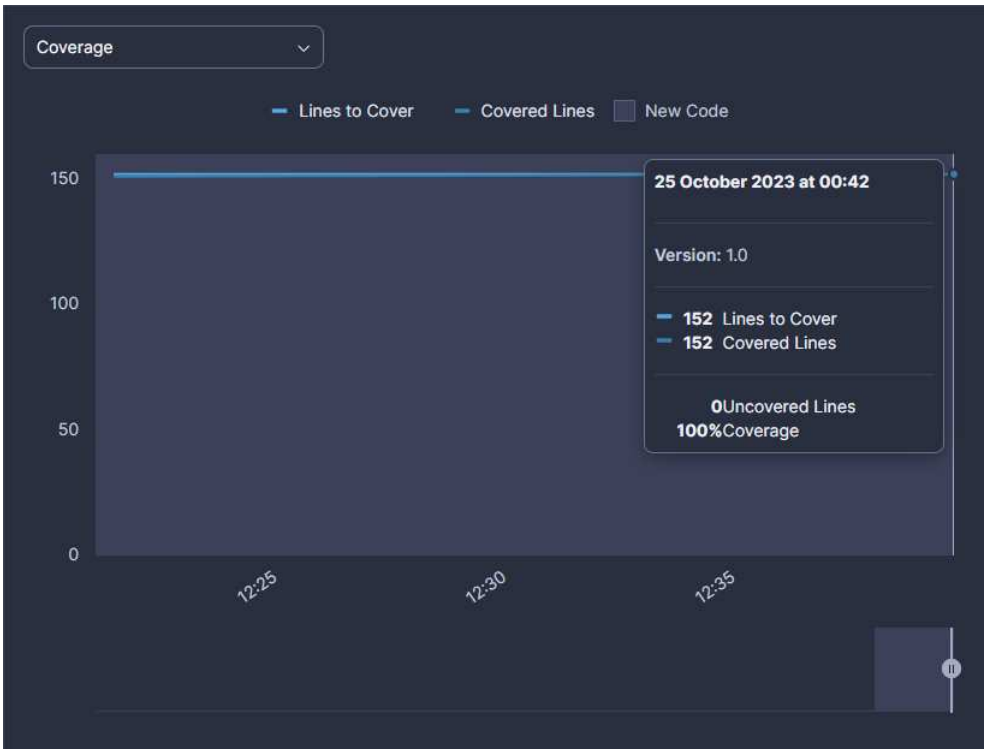
- *Commit* [10]: Criação de testes automatizados para o componente de navegação;
- *Commit* [11]: Criação de testes automatizados para o componente de produto;
- *Commit* [12]: Criação de testes automatizados para a página inicial;
- *Commit* [13]: Criação de testes automatizados para o componente *hero*;
- *Commit* [14]: Criação de testes automatizados para o componente de categorias promocionais;
- *Commit* [15]: Criação de testes automatizados para o componente de produtos em destaque;
- *Commit* [16]: Criação de testes automatizados para o módulo de ProductsService;
- *Commit* [17]: Criação de testes automatizados para a página de catálogo;
- *Commit* [18]: Ajuste nos testes automatizados da página inicial, e
- *Commit* [19]: Adição de novo caso de teste para o componente *hero*.

A Figura 17 não mostra um aumento na porcentagem de cobertura de código, uma vez que foram necessárias diversas alterações nos arquivos de configuração do SonarCloud para a análise correta da cobertura. Essa etapa de configuração ocorreu após a criação dos testes automatizados dessa prova de conceito.

A Figura 18 apresenta, por fim, o registro de duplicação de código durante o desenvolvimento desta segunda prova de conceito. É possível notar que, no início do desenvolvimento desta prova de conceito, havia uma certa quantidade de código duplicado

[10] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/7a8007b8fd88aa7fa9ad2fc80f81ea31b913973b>. Último acesso em: Novembro. 2023
[11] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/a1568c0c00cd7ba7af744102925c996776099de8>. Último acesso em: Novembro. 2023
[12] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/bf2022d034bc0812b69964ee093324456cae1894>. Último acesso em: Novembro. 2023
[13] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/f4552468f2cca0bb7d8173b6f1047e661d2fadf9>. Último acesso em: Novembro. 2023
[14] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/78c38a038828df812f096e01a09f419fa546bb26>. Último acesso em: Novembro. 2023
[15] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/56c62b48381db02a6564428d60644840a85687c4>. Último acesso em: Novembro. 2023
[16] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/c50e1620f5310258de628e24170bb80fa88285a0>. Último acesso em: Novembro. 2023
[17] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/6ad7a6494708ab45413de0b25809699d35eea42f>. Último acesso em: Novembro. 2023
[18] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/b03e8249999b24a44be645c301b32e4cad06f275>. Último acesso em: Novembro. 2023
[19] Disponível em: <https://github.com/twistershark/tcc-ecommerce-products/commit/b03e8249999b24a44be645c301b32e4cad06f275>. Último acesso em: Novembro. 2023

Figura 17 – Histórico de cobertura de código relacionado à segunda POC



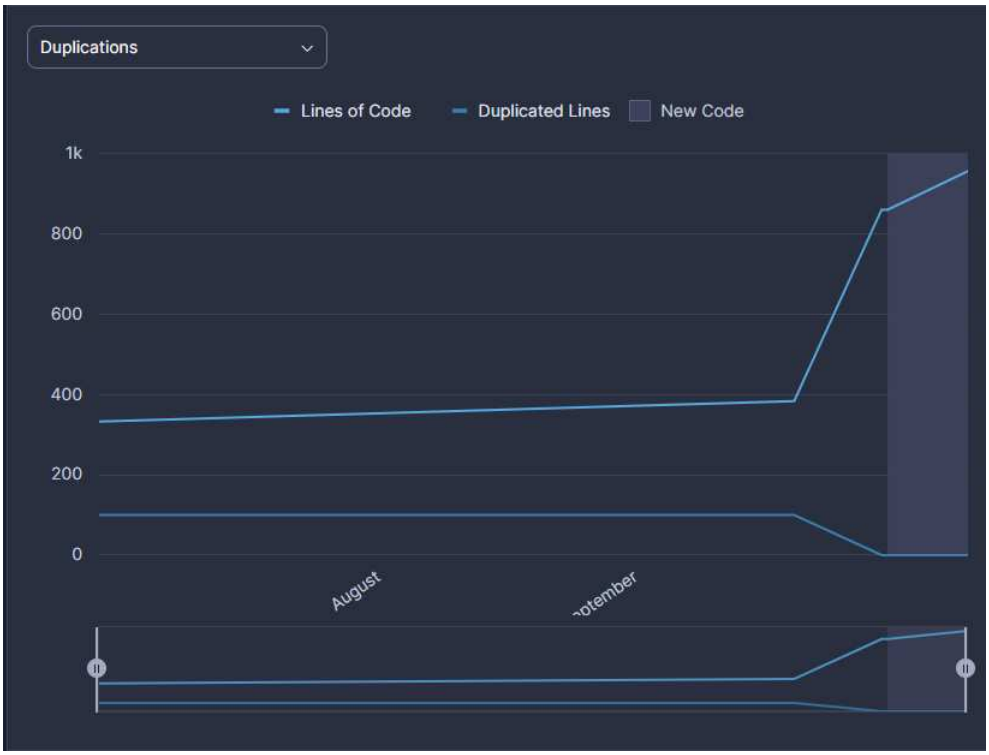
Fonte: Autor

na aplicação, que era remanescente da primeira prova de conceito, e que essa quantidade permaneceu constante durante quase todo o desenvolvimento dessa segunda prova de conceito. Há um ponto em que, ao mesmo tempo em que a quantidade de linhas de código do projeto aumentou de cerca de quatrocentas para mais de oitocentas, a quantidade de código duplicado foi reduzida a zero e permaneceu assim até o final do desenvolvimento.

A Figura 19 apresenta a complexidade ciclomática presente nesta segunda prova de conceito. A descrição detalhada da complexidade ciclomática está disponível na subseção SonarQub 10.0 (3.1.6). Conforme apresentado pela Figura 19, a complexidade foi aumentando à medida que a solução foi sendo desenvolvida, atingindo o seu ponto culminante ao final do desenvolvimento. A métrica de complexidade ciclomática, apresentada na Figura 19, corrobora o aspecto negativo sobre a complexidade abordado nesta prova de conceito, devido ao grande número de métodos, classes, funções e outros elementos presentes em uma aplicação *front-end* construída com Arquitetura Limpa. Não há uma medida padronizada de qual seria o valor bom ou ruim de complexidade ciclomática, sendo algo específico de cada aplicação, mas sempre procura-se manter esse valor o mais baixo possível. É importante salientar que todos os testes automatizados também são avaliados e somados ao resultado final.

Dessa forma, a Figura 20 apresenta um sumário do SonarCloud sobre o código dessa segunda prova de conceito. Observam-se notas A em todos os aspectos fundamentais

Figura 18 – Histórico de duplicação de código relacionado à segunda POC



Fonte: Autor

da análise do SonarCloud, sendo essa nota a melhor possível. Esses aspectos são:

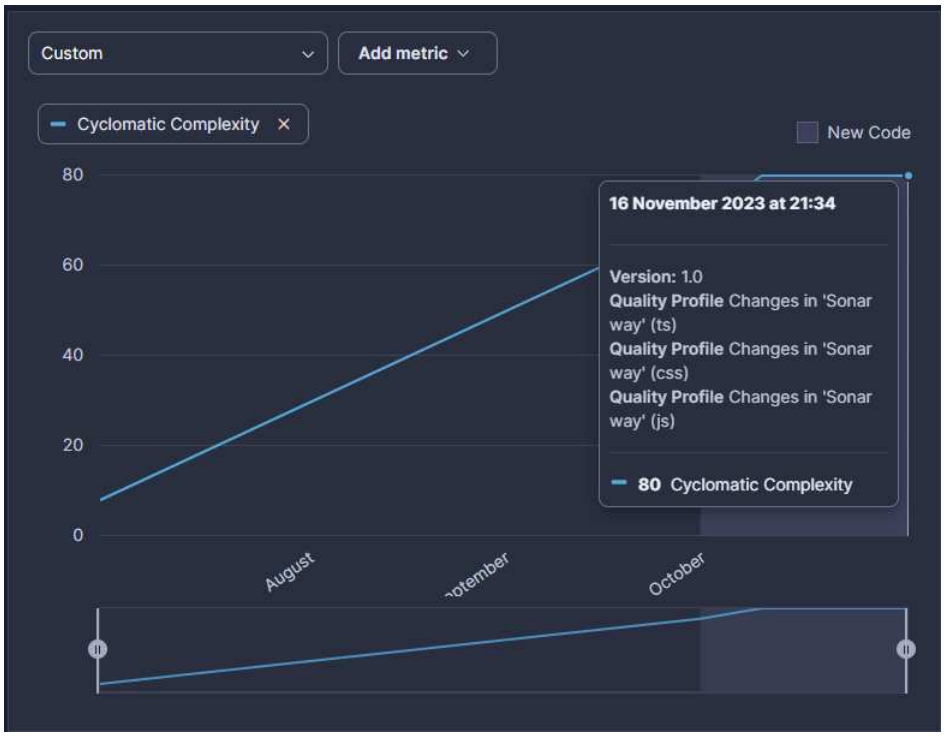
- *Reliability*, que representa os *bugs* da aplicação, onde a nota A significa que a quantidade de *bugs* é igual a zero;

- *Maintainability*, que está associada aos problemas e débitos técnicos da aplicação e tem nota A que representa que são necessário menos que 5% do tempo investido no desenvolvimento da aplicação para a resolução de problemas, e

- *Security*, que está ligado aos aspectos de vulnerabilidade de código, sendo a nota A representação de que não foi encontrada nenhuma vulnerabilidade conhecida.

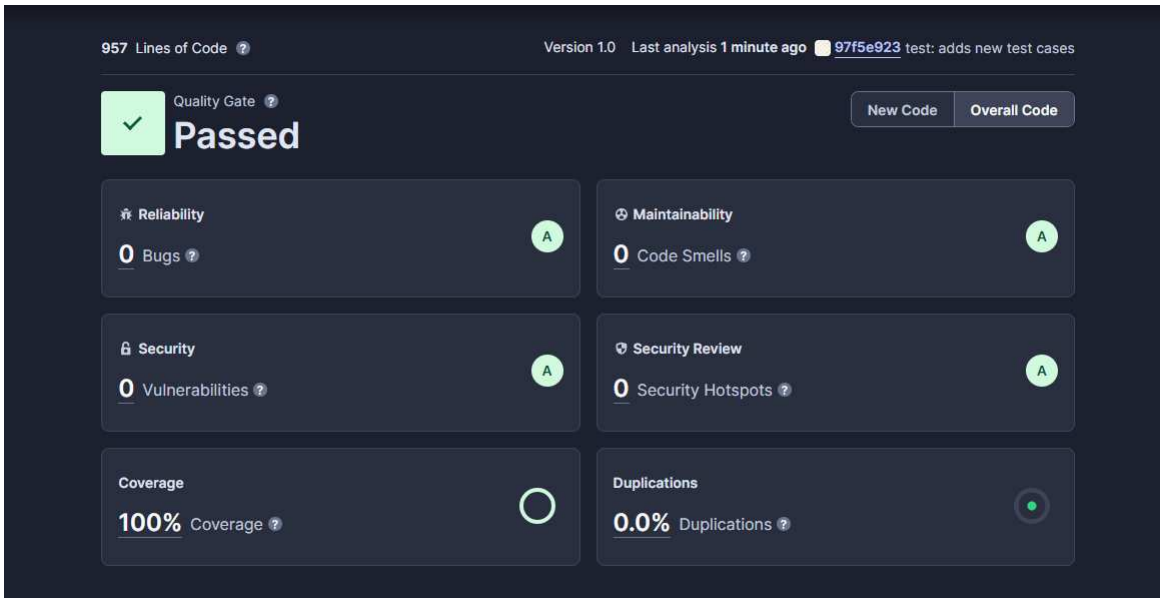
Além disso, é possível observar que a cobertura de código dessa prova de conceito foi igual a 100%, e a quantidade de duplicação de código foi de 0%. Com isso, tem-se, através do sumário, que a solução desenvolvida nessa prova de conceito atingiu todos os padrões de qualidade de aplicações JavaScript/TypeScript segundo os critérios de avaliação do SonarCloud.

Figura 19 – Complexidade ciclomática relacionada à segunda POC



Fonte: Autor

Figura 20 – Sumário da análise do SonarCloud relacionado à segunda POC



Fonte: Autor

5.5.4.3 Conclusão

Diante dos aspectos positivos e negativos, é possível concluir que a Arquitetura Limpa é relevante para o desenvolvimento de aplicações web e pode ser aplicada de forma eficaz em diferentes cenários. No entanto, para que a implementação seja bem-sucedida, é esperado que a equipe esteja familiarizada com a literatura e esteja disposta a assumir os aspectos positivos e negativos apresentados anteriormente. Dentre esses pontos, um dos relevantes é a equipe atuar de forma a mitigar pequenos desajustes orientando-se por uma ferramenta de análise de código. Conforme exposto anteriormente, essa disciplina, através de iterações, permitirá ajustar o código de forma a deixá-lo mais coeso, menos acoplado, evitando duplicidade. Portanto, mais manutenível. As escolhas arquiteturais acordadas nesse trabalho, segundo o estudo realizado, mostraram-se capazes de permitir ao desenvolvedor obter um código muito bem alinhado em termos de cobertura de testes e outras métricas de relevância (ex. *Reliability*, com foco em *bugs* e *Maintainability*, com foco em *code smells*).

5.6 POC 3 - Desenvolvimento dos Detalhes do Produto

Essa Seção descreve o desenvolvimento do *micro front-end* relacionado aos detalhes do produto, apresentando a Definição do Desafio relativa a essa prova de conceito, seguido pelos Requisitos do Desafio.

5.6.1 Definição do Desafio

Espera-se que essa prova de conceito seja capaz de desenvolver a tela de detalhes do produto, permitindo com que o usuário adicione um produto ao carrinho; ou navegue para o carrinho, caso o produto já tenha sido adicionado. Essa prova de conceito deve aplicar a Arquitetura Limpa de forma combinada com a Arquitetura *Micro Front-end*.

5.6.2 Requisitos do Desafio

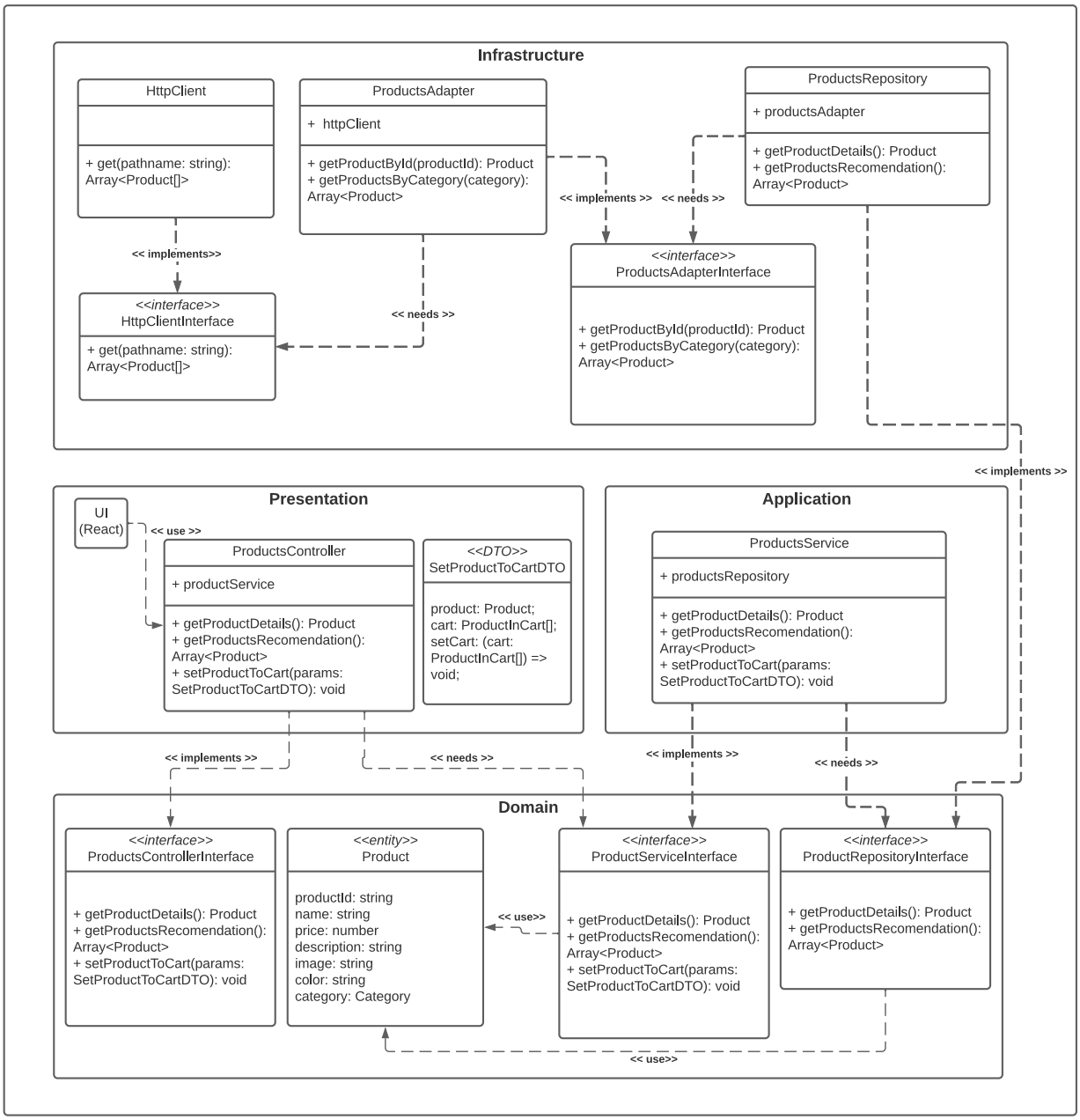
Para que essa prova de conceito seja considerada concluída, os seguintes requisitos devem ser atendidos:

- O usuário deve conseguir visualizar os detalhes de um produto;
- O usuário deve conseguir adicionar um produto ao carrinho de compras, e
- O usuário deve conseguir navegar para a tela de carrinho de compras, ao clicar no botão de adicionar ao carrinho.

5.6.3 Apresentação da Solução

A solução desta terceira prova de conceito iniciou-se com o desenvolvimento do protótipo da tela de detalhes do produto, seguindo-se o desenvolvimento orientado à Arquitetura Limpa e a criação dos testes automatizados. A Figura 21 apresenta a organização em camadas da solução dessa prova de conceito.

Figura 21 – Solução da terceira prova de conceito



Fonte: Autor

A organização das camadas desta prova de conceito segue os mesmos princípios abordados na segunda prova de conceito. A camada de domínio é responsável por definir as entidades e interfaces que representam as regras de negócio da aplicação. Na camada de aplicação, foram elaboradas as regras de negócio para esta prova de conceito, sendo elas compostas por três métodos. São eles:

- **getProductDetails**: Responsável por carregar, da fonte de dados, os dados de um determinado produto por meio de um identificador único de cada produto.;

- **getProductsRecomendation**: Método que carrega até quatro produtos da mesma categoria, visando apresentar outras opções ao usuário, simulando, dessa forma, um comportamento comum em lojas virtuais, e
- **setProductToCart**: Esse método, em particular, é usado para adicionar produtos ao estado global de carrinho. Ele faz isso através dos parâmetros recebidos, sendo eles: (i) produto que será adicionado ao carrinho; (ii) estado atual do carrinho, e (iii) função para atualização do carrinho. Neste caso específico, optou-se por repassar a função de atualização do estado para o serviço de produtos com o objetivo de preservar os benefícios de bibliotecas do ecossistema do React, dentre eles, o benefício de reatividade, que ocorre quando o estado é alterado e o componente que está consumindo esse estado é renderizado novamente, apresentando o layout conforme o novo estado, sendo importante para o bom funcionamento de aplicações React.

Além disso, utilizou-se, nesta prova de conceito, o padrão de repositórios para abstrair o conhecimento sobre a fonte de dados da regra de negócio. Assim, torna-se mais fácil manter o princípio de responsabilidade única entre os módulos. Além disso, foram usados adaptadores para a fonte de dados e clientes para o carregamento desses dados. Todos esses módulos foram descritos com mais detalhes na Apresentação da Solução da segunda prova de conceito.

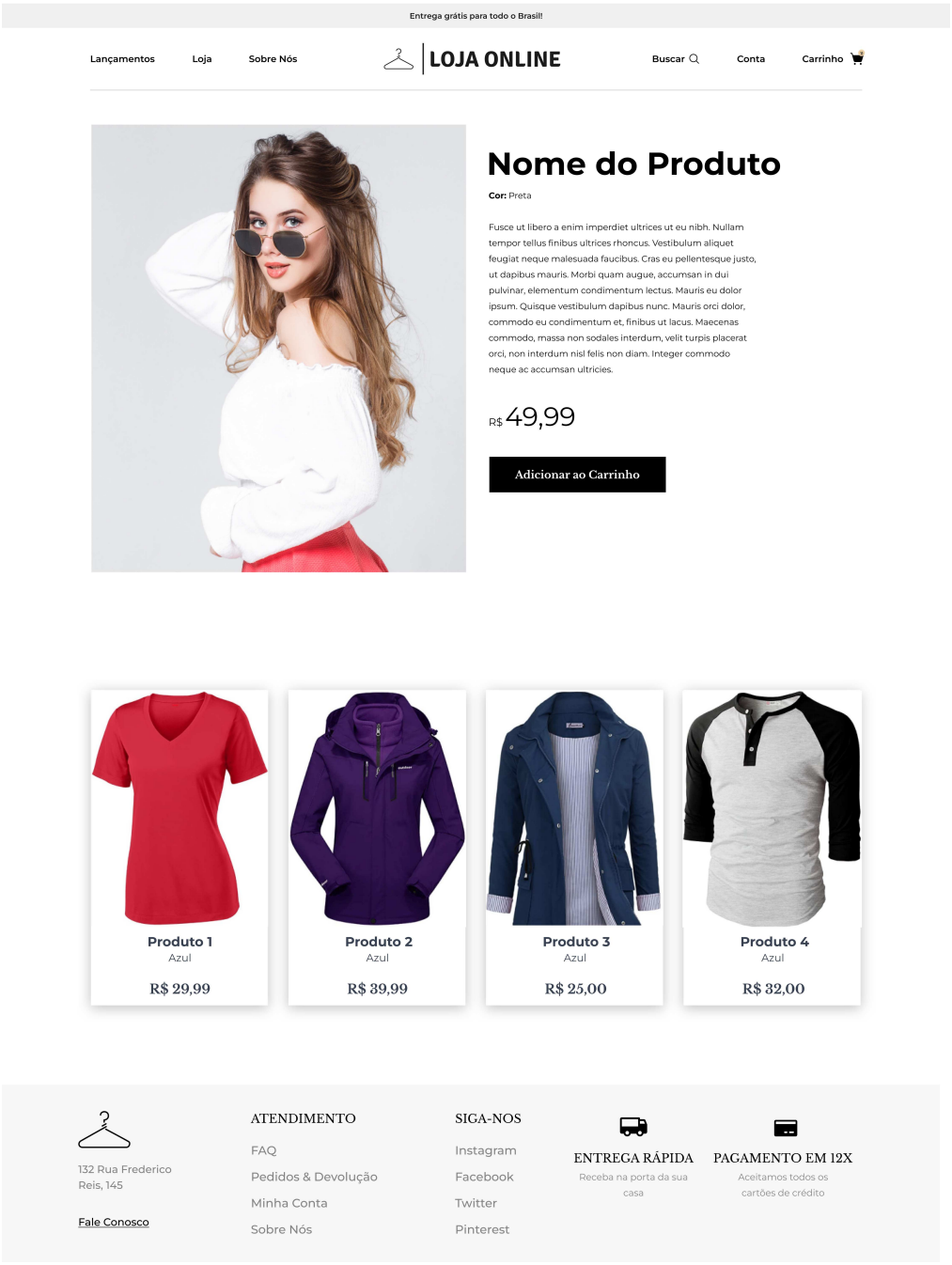
5.6.4 Análise de Resultados

Nesta subseção, serão apresentados os resultados alcançados com o desenvolvimento da solução mencionada anteriormente. As seguintes atividades foram concluídas nesta solução: a elaboração de um protótipo para a terceira prova de conceito, e (ii) o desenvolvimento da prova de conceito, segundo os princípios da Arquitetura Limpa e os requisitos dessa POC.

Em seguida, será apresentada uma visão da tela referente a essa prova de conceito. Em seguida, serão apresentados os resultados alcançados em relação aos requisitos estabelecidos para esta prova de conceito, bem como os resultados alcançados em relação aos objetivos específicos do trabalho. Finalmente, é apresentada a descrição da avaliação quantitativa referente a esse terceiro desafio.

A Figura 22 apresenta a tela de detalhes do produto, o que é o foco principal desta terceira prova de conceito. Nesta tela, apresenta-se uma imagem em tamanho grande do produto em questão, além de uma descrição detalhada, contendo informações sobre o preço, a categoria e a cor. Além disso, nesta tela, está disponível um botão que permite adicionar um produto ao carrinho. Este botão, após a adição do produto ao carrinho, permite que o usuário navegue até o carrinho para visualizar o seu estado atual. Em seguida, são apresentadas recomendações de produtos da mesma categoria para o usuário.

Figura 22 – Tela de detalhes do produto relacionado à terceira POC



Fonte: Autor

Durante o desenvolvimento dessa prova de conceito, o especialista, cujo perfil já foi abordado nas provas anteriores, não percebeu nenhum aspecto positivo ou negativo diferente dos já acordados na primeira e segunda provas de conceito. Ao contrário, o especialista reafirma a sua opinião sobre todos os aspectos anteriores também neste desafio.

A solução desta prova de conceito atendeu a todos os requisitos estabelecidos nos Requisitos do Desafio. São eles:

- O usuário deve conseguir visualizar os detalhes de um produto;
- O usuário deve conseguir adicionar um produto ao carrinho de compras, e
- O usuário deve conseguir navegar para a tela de carrinho de compras, ao clicar no botão de adicionar ao carrinho.

Com os resultados obtidos com essa solução, os seguintes objetivos do trabalho foram alcançados parcialmente:

- Identificação, estudo, documentação e aplicação dos principais conceitos relacionados à Arquitetura Limpa em *micro front-ends*, e
- Condução e documentação do processo de desenvolvimento, orientando-se por Provas de Conceito.

Da mesma forma que na segunda prova de conceito, buscou-se estabelecer o código da solução conforme os padrões estabelecidos pelo SonarCloud para aplicações JavaScript/TypeScript.

A Figura 23 apresenta o histórico de problemas no código, tais como *bugs* e falhas de segurança, ao longo do desenvolvimento desta terceira prova de conceito. Assim como ocorreu no desenvolvimento da segunda prova de conceito, neste terceiro desafio também é possível notar que, inicialmente, a aplicação não apresenta problemas. No entanto, ao final do desenvolvimento desse *micro front-end*, houve um aumento, indicando um problema no código. O gráfico também evidencia que, após o *commit* referente à correção desse problema, a aplicação voltou ao seu estado inicial, sem qualquer problema a ser resolvido.

O problema apresentado pelo SonarCloud, na solução desse desafio, foi solucionado através do seguinte *commit* específico:

Commit [20]:

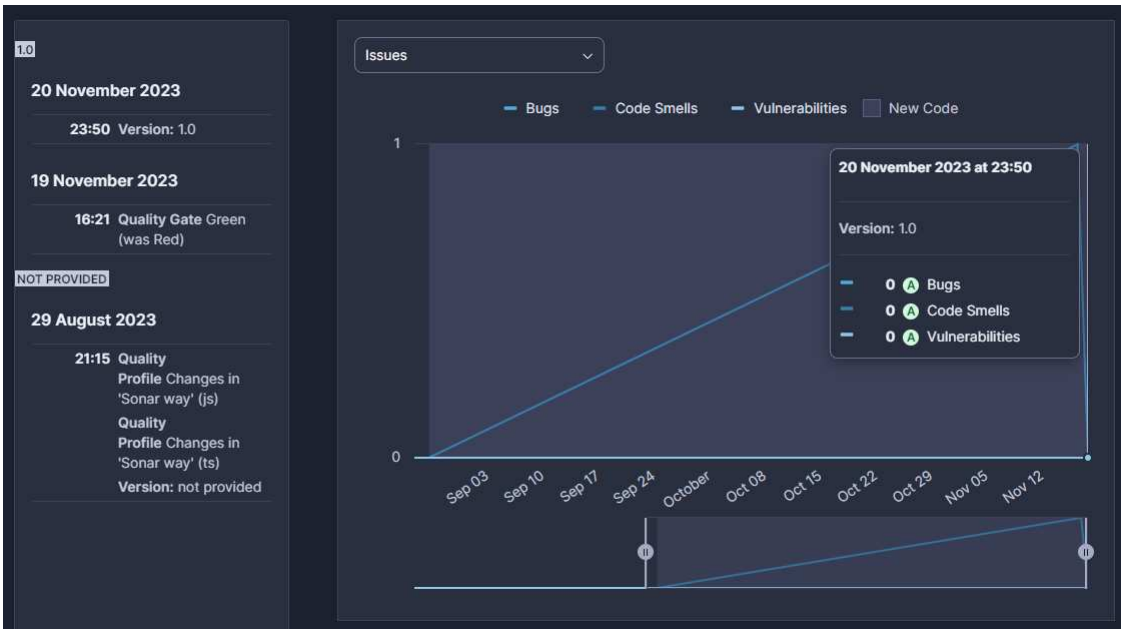
Tipo de Problema: Manutenibilidade.

Contexto: O *mock* da classe *ProductsRepository* estava importando uma função desnecessariamente, sendo essa função utilizada nesse *mock*.

Problema: Importações desnecessárias não acrescentam nada em termos de funcionalidade no código, apenas aumentam o tamanho do código JavaScript que o usuário precisará carregar para usar a aplicação.

Solução: A importação desnecessária foi removida.

Figura 23 – Histórico de problemas de código relacionado à terceira POC



Fonte: Autor

A Figura 24 mostra o histórico de cobertura de testes. Realizou-se um total de catorze *commits* visando atingir a maior cobertura possível no código deste desafio. Conforme a Figura, foi possível atingir o percentual de 97% de cobertura de código. Em geral, todos os commits concentraram-se na criação de testes para os módulos do sistema. Foram eles:

- *Commit* [21]: Configuração do Jest;
- *Commit* [22]: Instalação de bibliotecas para testes automatizados;
- *Commit* [23]: Configuração de scripts para testes;
- *Commit* [24]: Adição de arquivos de configuração para os testes;
- *Commit* [25]: Criação de *mock* para a entidade Produto;

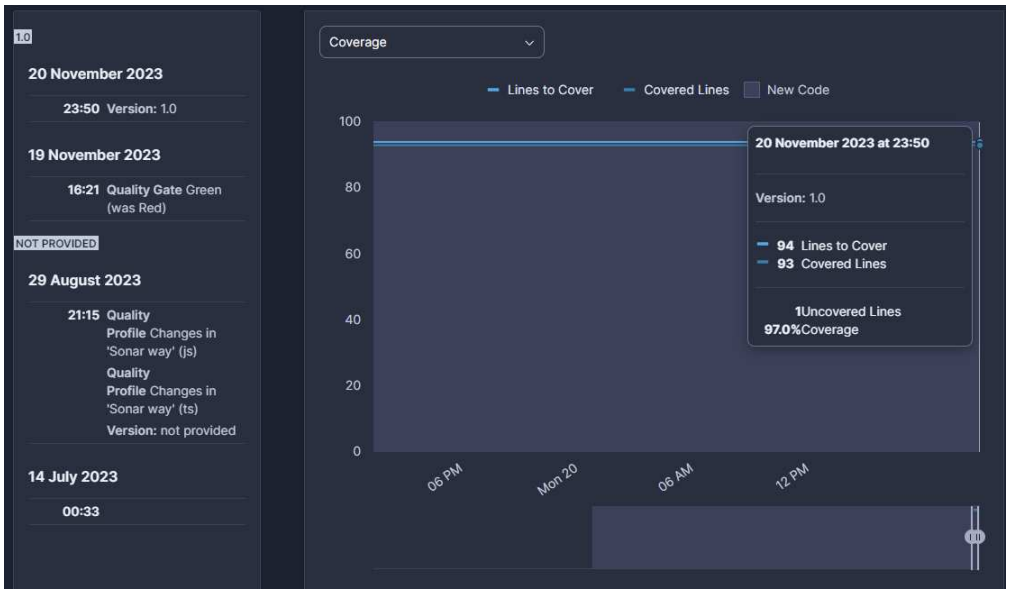
[20] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/6601e4ca95e9a8df2487c7caa945f3512cabe1fe>. Último acesso em: Novembro. 2023
[21] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/cd0c3a6c3243a1da473139248038c85b36d74f32>. Último acesso em: Novembro. 2023
[22] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/9d1ad96c6d37ed20f95c1d68000b76e4bd2c5fc9>. Último acesso em: Novembro. 2023
[23] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/3930f4f85b95eb1fb61ada0fdee204095d4bd528>. Último acesso em: Novembro. 2023
[24] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/fa73bd866a1d6329d198ff811e8eb62074c13914>. Último acesso em: Novembro. 2023
[25] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/d4f434181227e32642ffcdcfa9f07f283317f9ae>. Último acesso em: Novembro. 2023

- *Commit* [26]: Atualização das bibliotecas para testes automatizados;
- *Commit* [27]: Criação de *mocks* para *ProductsRepository*, *ProductsService*, *ProductsAdapter* e *HttpClient*;
- *Commit* [28]: Criação de testes automatizados para o módulo *ProductsService*;
- *Commit* [29]: Criação de testes automatizados para o módulo *ProductsAdapter*;
- *Commit* [30]: Criação de testes automatizados para o módulo *HttpClient*;
- *Commit* [31]: Criação de testes automatizados para o módulo *ProductsRepository*;
- *Commit* [32]: Criação de testes automatizados para o módulo *ProductsController*;
- *Commit* [33]: Criação de testes automatizados para o componente de botão, e
- *Commit* [34]: Criação de testes automatizados para a tela de detalhes do produto.

A Figura 25 apresenta, por fim, o registro de duplicação de código durante o desenvolvimento desta terceira prova de conceito. Há uma certa quantidade de código duplicado na aplicação no início do desenvolvimento deste desafio, que era remanescente da primeira prova de conceito, assim como ocorreu na segunda POC. Essa quantidade permaneceu constante durante quase todo o desenvolvimento desta terceira prova de conceito. No entanto, o código duplicado apontado pela análise do SonarCloud está relacionado ao arquivo de configuração do *Module Federation*, responsável pela organização dos *micro front-ends*, como apresentado no Capítulo 3 - Suporte Tecnológico. A duplicação de código apontada foi causada pelo fato de existirem dois arquivos de configuração que estão relacionados aos módulos expostos para os outros *micro front-ends*, sendo que um é usado no ambiente de desenvolvimento e o outro no ambiente de produção. Como se trata de um arquivo de configuração, e não de um código da aplicação em si, a solução foi adicionar

[26] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/c0343fe8c9b87b79a923fef13e6041566a9b0882>. Último acesso em: Novembro. 2023
[27] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/928216ca7e26f007356e0f71eb6de22496fd685c>. Último acesso em: Novembro. 2023
[28] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/1f7b82e6205f0348bd8ccaea1f0c6e1360364dba>. Último acesso em: Novembro. 2023
[29] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/57d6335d92f0f183e8c6b358f1ea408c9d163742>. Último acesso em: Novembro. 2023
[30] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/737191a99d464aea5fd10718c3d4725c1ade82a8>. Último acesso em: Novembro. 2023
[31] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/17cca79ca9bdd3fdeb53f238c942fbdc00ebede8>. Último acesso em: Novembro. 2023
[32] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/9f4052fc5fb8f6c702bf1408a26ecadc5eccb852>. Último acesso em: Novembro. 2023
[33] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/da004e28ebe97c83c9a13dae5db91d7c091d8c3b>. Último acesso em: Novembro. 2023
[34] Disponível em: <https://github.com/twistershark/tcc-ecommerce-product-details/commit/f07db82ec19c3e510478bb3a4486012caf149293>. Último acesso em: Novembro. 2023

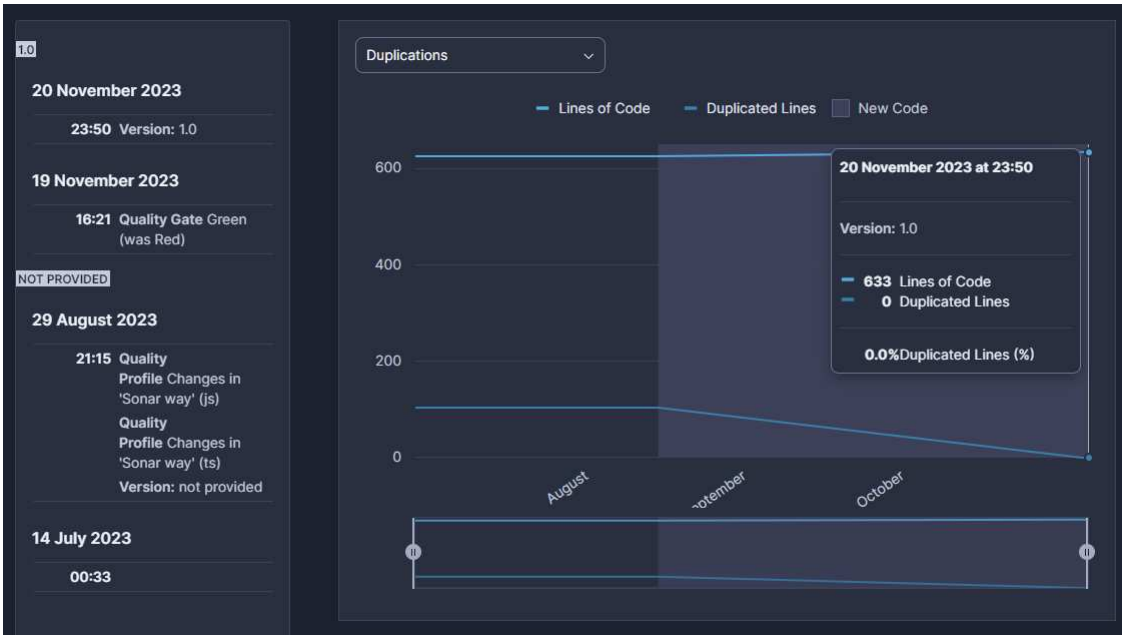
Figura 24 – Histórico de cobertura de código relacionado à terceira POC



Fonte: Autor

esses dois arquivos ao conjunto de arquivos ignorados durante a análise do SonarCloud. Sendo assim, o número de linhas duplicadas foi reduzido para zero.

Figura 25 – Histórico de duplicação de código relacionado à terceira POC

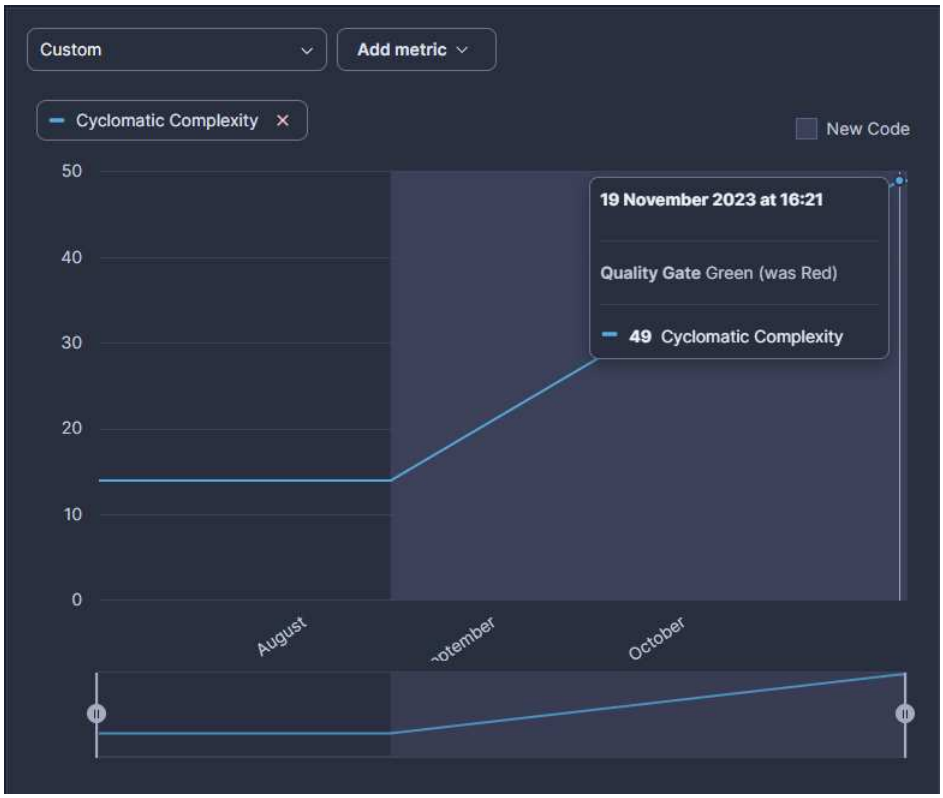


Fonte: Autor

A Figura 26 apresenta a complexidade ciclomática presente nesta terceira prova de conceito. Da mesma forma que ocorreu na segunda avaliação de conceito, a complexidade foi aumentando à medida que a solução foi sendo desenvolvida, atingindo seu ponto ápice no final do desenvolvimento. O valor de 49 para a complexidade ciclomática desta solução

indica que, em comparação com a segunda prova de conceito, a sua complexidade é menor, o que se deve ao fato de o principal insumo desta prova de conceito ser a criação de uma tela com detalhes de um produto específico.

Figura 26 – Complexidade ciclomática relacionada à terceira POC

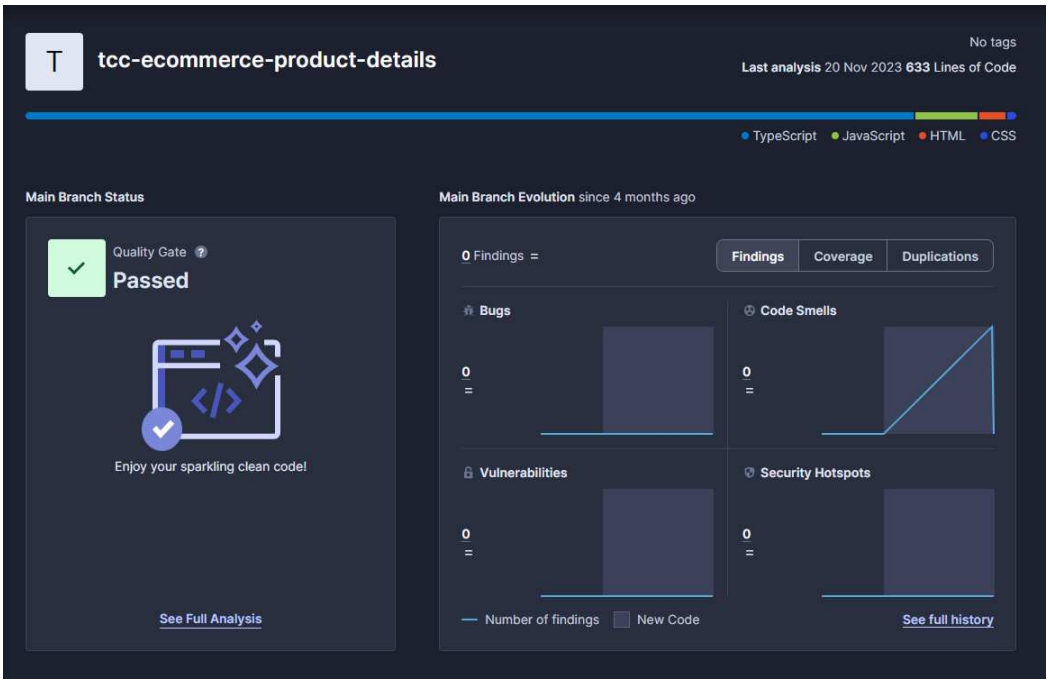


Fonte: Autor

A Figura 27 apresenta um breve resumo do SonarCloud sobre o código da terceira prova de conceito. Observa-se a presença de notas A em todos os aspectos de relevância da avaliação do SonarCloud, sendo essa a nota mais elevada, conforme detalhado na Seção de Análise de Resultados (5.5.4) da segunda prova de conceito.

Além disso, é possível notar que a cobertura de código nesta prova de conceito foi de 97%, e a quantidade de duplicações de código foi de 0%. Dessa forma, percebe-se, através do sumário, que a solução apresentada nesta prova de conceito atendeu a todos os requisitos de qualidade de aplicações JavaScript/TypeScript, conforme os critérios de avaliação do SonarCloud.

Figura 27 – Sumário da análise do SonarCloud relacionado à terceira POC



Fonte: Autor

5.6.4.1 Conclusão

Após a Análise de Resultados desta prova de conceito, constatou-se que ela atendeu a todos os requisitos e padrões de qualidade estabelecidos. Além disso, foi possível reforçar os pontos apresentados nas provas de conceito anteriores, como a testablidade e a separação das responsabilidades dos diferentes módulos da aplicação. Um ponto relevante a ser destacado é que, após aprender sobre a utilização de Arquitetura Limpa em aplicações *front-end*, percebeu-se uma diminuição significativa no tempo de desenvolvimento da solução, uma vez que os conceitos na literatura já haviam sido analisados na segunda prova de conceito, o que facilita a tomada de decisão e a organização da solução como um todo.

5.7 POC 4 - Desenvolvimento do Carrinho de Compras

Essa Seção descreve o desenvolvimento do *micro front-end* de carrinho da loja online, provendo a descrição da Definição do Desafio, seguido pelos Requisitos do Desafio.

5.7.1 Definição do Desafio

O objetivo dessa prova de conceito é desenvolver, utilizando a Arquitetura Limpa, a tela de carrinho de compras, presente no *micro front-end* de carrinho, além de todos os componentes e regras de negócio que serão necessários para ela. Espera-se que o usuário consiga alterar a quantidade de produtos presentes no carrinho, remover produtos, bem como adicionar produtos sugeridos.

5.7.2 Requisitos do Desafio

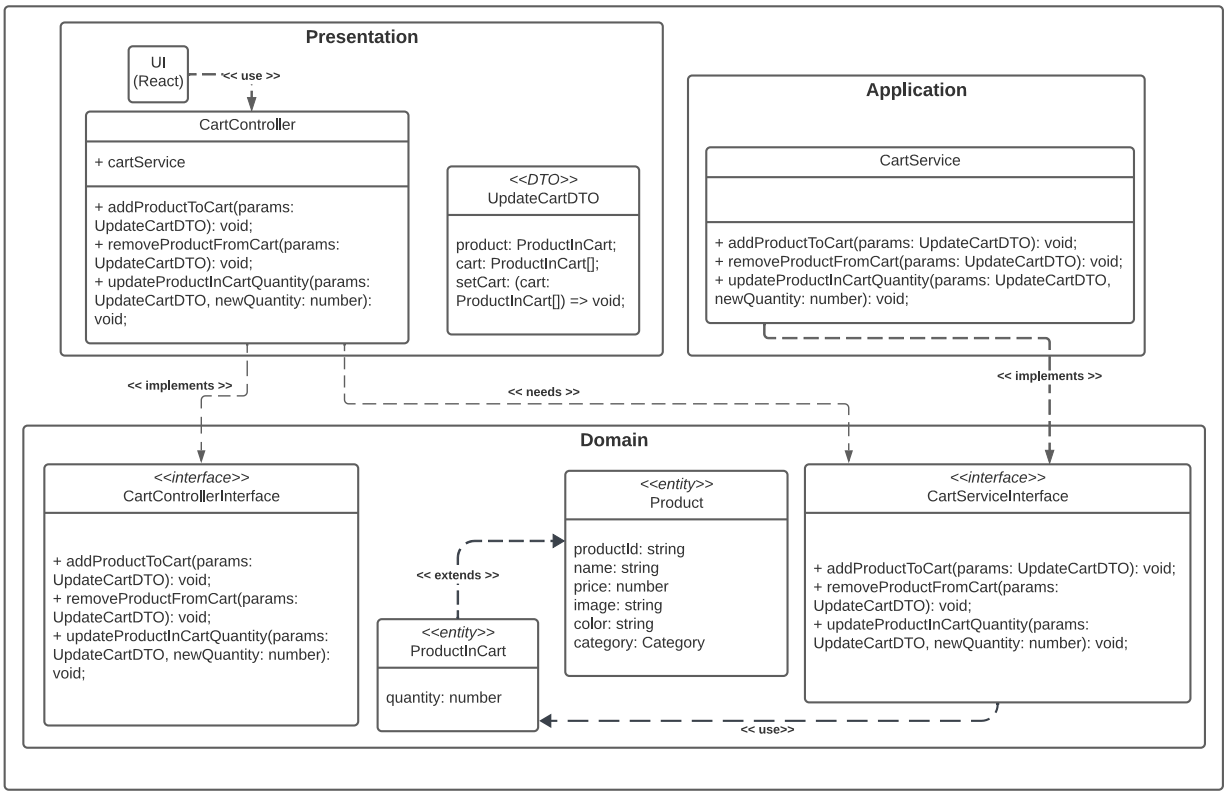
Os seguintes requisitos devem ser atendidos para que essa prova de conceito seja considerada concluída:

- O usuário deve conseguir remover produtos ao carrinho;
- O usuário deve conseguir alterar a quantidade de produtos do carrinho;
- O usuário deve conseguir visualizar os valores estimados relacionados ao carrinho;
- O usuário deve conseguir adicionar produtos sugeridos ao carrinho;
- O usuário deve conseguir navegar para a tela de *checkout* a partir do carrinho de compras, e
- O usuário deve conseguir navegar pela aplicação sem perder o estado dos produtos do carrinho.

5.7.3 Apresentação da Solução

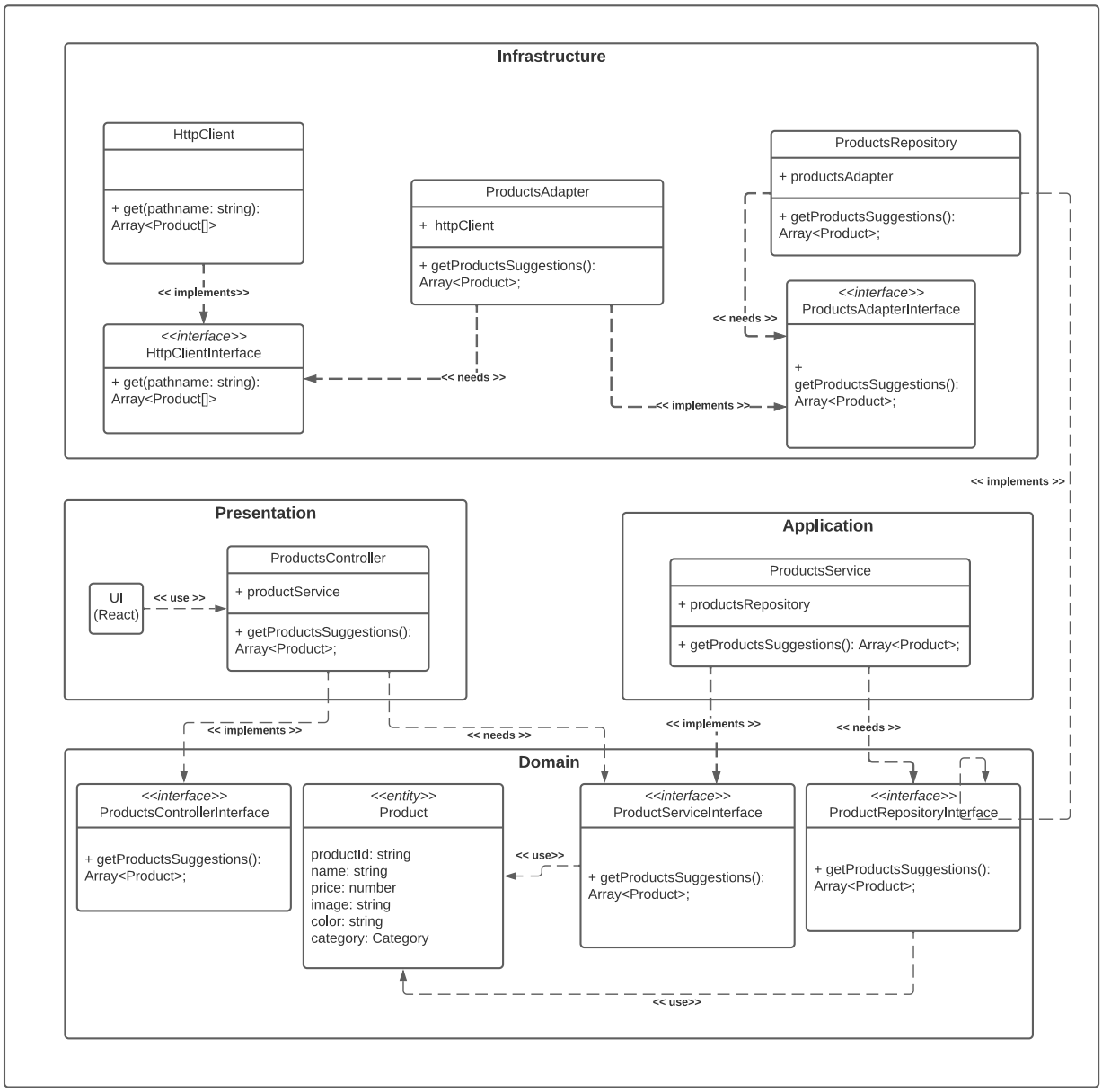
A partir do conhecimento da literatura adquirido e aplicado nas provas de conceito anteriores, iniciou-se o desenvolvimento da solução através da criação do protótipo da tela de carrinho, seguindo-se o seu desenvolvimento orientado pela Arquitetura Limpa e *Micro Front-end*, além da criação de testes automatizados para toda a aplicação. A fim de facilitar a leitura, optou-se por separar a organização em camadas dessa solução em duas figuras distintas. Primeiramente, a Figura 28 apresenta todas as camada que envolvem o contexto de carrinho dessa prova de conceito, enquanto a Figura 29 apresenta a solução relacionada ao contexto de produtos sugeridos para o usuário.

Figura 28 – Solução da quarta prova de conceito - 1



Fonte: Autor

Figura 29 – Solução da quarta prova de conceito - 2



Fonte: Autor

A quarta prova de conceito adotou uma abordagem arquitetural consistente com as decisões tomadas nos desafios anteriores, seguindo os princípios da Arquitetura Limpa. A principal diferença está na introdução de *controllers*, *services* e outros módulos específicos para os contextos de "carrinho" e "produto". A Arquitetura Limpa estabelece uma separação clara de responsabilidades e dependências em uma aplicação, favorecendo a manutenção, a escalabilidade e a testabilidade do sistema. Dessa forma, a separação desses módulos visou proporcionar uma organização mais eficiente e modular.

Além disso, a separação entre carrinho e produtos contribuiu para manter a coesão e o baixo acoplamento, características fundamentais para o desenvolvimento de software.

Essa abordagem facilita a manutenção e evolução do código, uma vez que as alterações em uma parte específica do sistema podem ser realizadas sem afetar desnecessariamente outras partes. Essa prova de conceito mostrou como contextos distintos seriam integrados na aplicação, demonstrando a separação das responsabilidades, facilitando a realização de testes e o desenvolvimento das funcionalidades, uma vez que são independentes uma das outras.

Em suma, a solução adotada na prova de conceito manteve a coerência com as decisões arquiteturais das provas anteriores, seguindo os princípios e responsabilidades de cada camada da Arquitetura Limpa.

5.7.4 Análise de Resultados

Nesta subseção, serão apresentados os resultados alcançados com o desenvolvimento da solução apresentada anteriormente. Nesta solução, foram concluídas as seguintes tarefas: (i) a criação de um protótipo para a quarta prova de conceito, e (ii) a criação da prova de conceito segundo os princípios da Arquitetura Limpa e as exigências dessa POC.

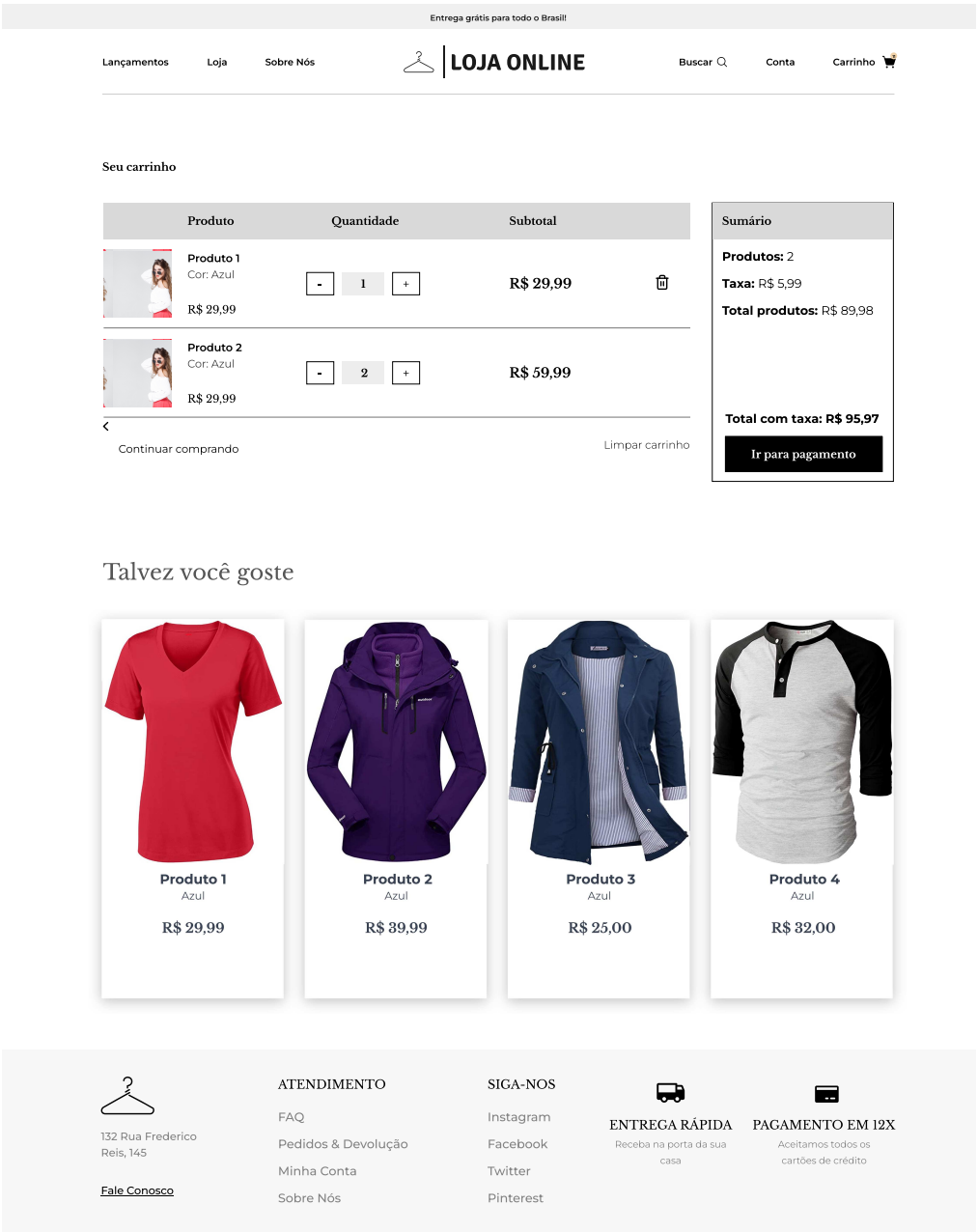
A seguir, será apresentada a tela de carrinho, sendo essa o principal artefato desse desafio. Posteriormente, serão apresentados os resultados alcançados em relação aos requisitos estabelecidos para a quarta prova de conceito, bem como os resultados alcançados em relação aos objetivos específicos do trabalho. Finalmente, é apresentada a descrição da avaliação quantitativa referente a esse quarto desafio, seguido da conclusão.

A Figura 30 apresenta a tela de carrinho, na qual o usuário consegue ter controle sobre todos os produtos adicionados ao carrinho. Além disso, o usuário também consegue realizar as seguintes ações: (i) alterar quantidade de um produto no carrinho; (ii) remover um produto do carrinho; (iii) visualizar o preço total de um produto no carrinho, sendo esse a multiplicação do preço do produto e da sua quantidade no carrinho; (iv) remover todos os produtos do carrinho; (v) retornar a tela de catálogo de produtos; (vi) ir para a tela de *checkout*, e (vii) adicionar produtos sugeridos ao carrinho.

Durante o desenvolvimento dessa prova de conceito, o especialista, cujo perfil já foi abordado nas provas anteriores e no Método de Análise de Resultados, percebeu o seguinte aspecto negativo nessa prova de conceito:

Devido às particularidades do React, se a camada de apresentação fosse modificada para outra solução, praticamente todos os testes relacionados às páginas e componentes da camada de apresentação seriam refeitos. É claro que a implementação de uma alteração dessa natureza, devido à Arquitetura Limpa, não teria um impacto significativo, podendo ser facilmente executada. No entanto, acredita-se que o cenário ideal seria que os testes pudessem ser executados independentemente do modo como o *front-end* está sendo

Figura 30 – Tela de carrinho relacionado à quarta POC



Fonte: Autor

renderizado em segundo plano.

A solução desta prova de conceito atendeu a todos os requisitos estabelecidos nos Requisitos do Desafio. São eles:

- O usuário deve conseguir remover produtos ao carrinho;
- O usuário deve conseguir alterar a quantidade de produtos do carrinho;
- O usuário deve conseguir visualizar os valores estimados relacionados ao carrinho;

- O usuário deve conseguir adicionar produtos sugeridos ao carrinho;
- O usuário deve conseguir navegar para a tela de *checkout* a partir do carrinho de compras, e
- O usuário deve conseguir navegar pela aplicação sem perder o estado dos produtos do carrinho.

Com os resultados obtidos com essa solução, os seguintes objetivos do trabalho foram alcançados parcialmente:

- Identificação, estudo, documentação e aplicação dos principais conceitos relacionados à Arquitetura Limpa em *micro front-ends*, e
- Condução e documentação do processo de desenvolvimento, orientando-se por Provas de Conceito.

A Figura 31 apresenta o histórico de problemas no código, incluindo *bugs* e falhas de segurança, durante o desenvolvimento da quarta avaliação de conceito. Inicialmente, é possível notar um pico de seis problemas no código, mas é importante salientar que esses problemas estavam relacionadas à fase inicial deste estudo, quando a abordagem adotada ainda não estava de acordo com os princípios da Arquitetura Limpa. Durante as primeiras etapas do desenvolvimento, esse número foi reduzido para zero, uma vez que os módulos problemáticos foram excluídos do código. Em seguida, ocorreram dois aumentos no número de problemas identificados, mas, em ambos os casos, foram realizados *commits* de correção para eliminar quaisquer problemas anteriormente detectados.

Os problemas apresentados pelo SonarCloud, na solução desse desafio, foram solucionados através dos seguintes *commits*:

- *Commit* [35]:

  Tipo de Problema: Manutenibilidade.

  Contexto: A interface *ProductsRepositoryInterface* estava importando uma outra interface desnecessariamente.

  Problema: Importações desnecessárias não acrescentam em termos de funcionalidade no código, apenas aumentam o tamanho do código JavaScript carregado por um determinado módulo.

  Solução: A importação desnecessária foi removida.

[35] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/4063365b23686f69210ce65bb2833c889b0b21e4>. Último acesso em: Novembro. 2023

- *Commit* [36]:

  Tipo de Problema: Manutenibilidade.

  Contexto: O *mock* da classe *ProductsService* estava importando uma interface desnecessariamente.

  Problema: Importações desnecessárias não acrescentam em termos de funcionalidade no código, apenas aumentam o tamanho do código JavaScript carregado por um determinado módulo.

  Solução: A importação desnecessária foi removida.

Figura 31 – Histórico de problemas de código relacionado à quarta POC

Fonte: Autor

A Figura 32 mostra o histórico de cobertura de testes. Realizou-se um total de vinte *commits* visando atingir a maior cobertura possível no código deste desafio. Conforme a Figura, foi possível atingir o percentual de 95% de cobertura de código. Os commits relacionados a testes automatizados nessa solução foram:

- *Commit* [37]: Criação de *mock* para a entidade *ProductInCart*;
- *Commit* [38]: Instalação de bibliotecas para testes automatizados;

[36] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/7e4850bed007a4047d4b630a56c85b5e151efcc5>. Último acesso em: Novembro. 2023

[37] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/df42e780459ad97beb69f7f359fc72145b155fd3>. Último acesso em: Novembro. 2023

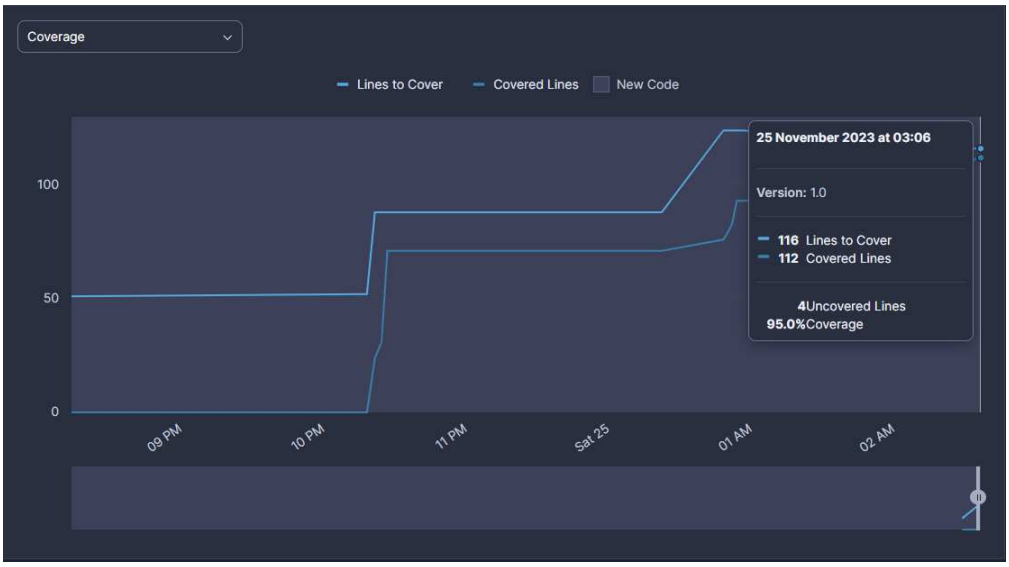
[38] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/3874fbfbee99877d4a496ddc47418ca865967fb0>. Último acesso em: Novembro. 2023

- *Commit* [39]: Criação de testes para o módulo *CartService*;
- *Commit* [40]: Criação de testes para o módulo *CartController*;
- *Commit* [41]: Criação dos primeiros casos de teste para a página de carrinho;
- *Commit* [42]: Adição de novos casos de teste para o módulo *CartService*;
- *Commit* [43]: Adição de caso de teste para o fluxo de alteração da quantidade de produtos no carrinho;
- *Commit* [44]: Criação de testes automatizados para o componente *ProductSuggestion*;
- *Commit* [45]: Criação de testes automatizados para o módulo *ProductsService*;
- *Commit* [46]: Criação de *mock* do módulo *ProductAdapter*;
- *Commit* [47]: Criação de *mock* do módulo *HttpClient*;
- *Commit* [48]: Criação de *mock* do módulo *ProductsRepository*;
- *Commit* [49]: Criação de novos casos de teste para a página de carrinho;
- *Commit* [50]: Criação de testes automatizados para o módulo *HttpClient*;
- *Commit* [51]: Criação de testes automatizados para o módulo *ProductsAdapter*;

[39] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/54fb28f6be7e3b1059b143e9ac3162b0446534c0>. Último acesso em: Novembro. 2023
[40] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/32a9eae3fbc3b1199fecf0fc77fdb87957683b8f>. Último acesso em: Novembro. 2023
[41] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/f3eb8362be32f62f5da854d148efecd29663eef9>. Último acesso em: Novembro. 2023
[42] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/30f5a1ce140b8dbaa19035d3e6e0b17a6083d27b>. Último acesso em: Novembro. 2023
[43] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/1e82454971fa3917db1f89d094ddc4460f127787>. Último acesso em: Novembro. 2023
[44] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/1a822a54eba8e5c13eaeeb0fdf48a184b19fb5ea>. Último acesso em: Novembro. 2023
[45] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/8fb4f6a8811dec9bdff76258ab5f5ef08f78136d>. Último acesso em: Novembro. 2023
[46] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/2f2d9fb83a36bc8ced0b47c20880fbd3f0c81ca3>. Último acesso em: Novembro. 2023
[47] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/b6232da8422504556329e95caea2ed0753aae3e0>. Último acesso em: Novembro. 2023
[48] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/faf0c27052a9454e9c2304a8dfec8fd1d78857a5>. Último acesso em: Novembro. 2023
[49] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/e3fb4c4d04e444ce314b46c0909c9fa6e15e5503>. Último acesso em: Novembro. 2023
[50] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/21a3103076d3101a6ce48e101f97024e348d193f>. Último acesso em: Novembro. 2023
[51] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/21b3ca909112688b9d0244df53a4eab3246ac1ea>. Último acesso em: Novembro. 2023

- *Commit* [52]: Criação de testes automatizados para o módulo *ProductsService*;
- *Commit* [53]: Adição de novos casos de teste para a tela de carrinho;
- *Commit* [54]: Criação de *mock* do módulo *ProductsService*;
- *Commit* [55]: Criação de testes automatizados para o módulo *ProductsController*, e
- *Commit* [56]: Criação de testes automatizados para a função *formatCurrency*.

Figura 32 – Histórico de cobertura de código relacionado à quarta POC



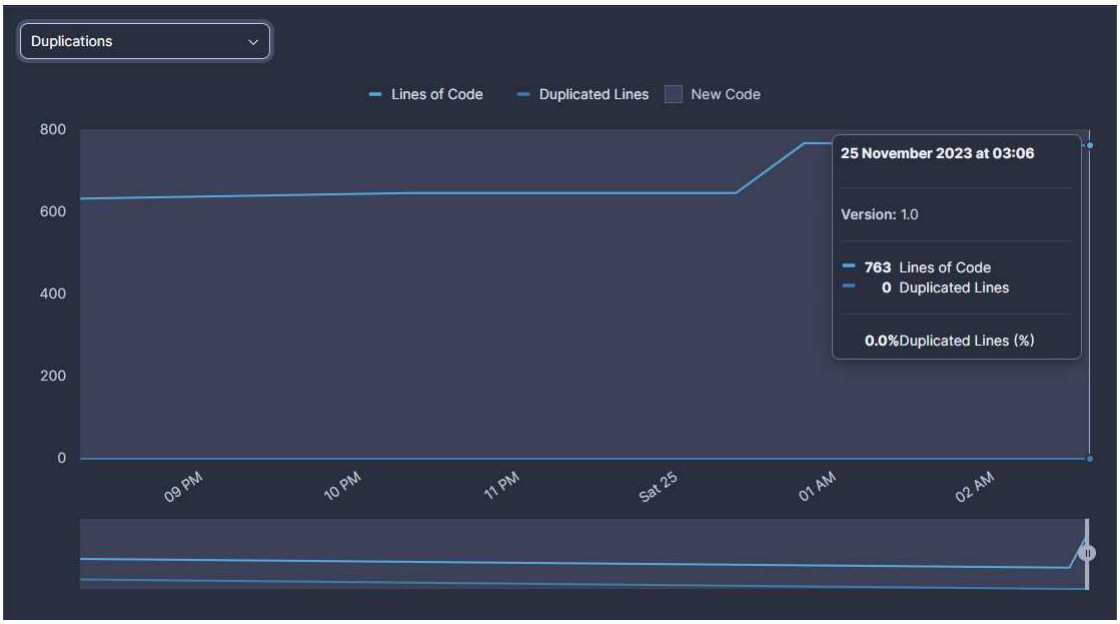
Fonte: Autor

A Figura 33 apresenta, finalmente, o registro de duplicação de código durante o desenvolvimento desta quarta prova de conceito. Observa-se pela Figura que, desde o início até o término do processo de desenvolvimento, não foram detectados códigos duplicados na solução desta prova de conceito.

A Figura 34 apresenta a complexidade ciclomática dessa quarta prova de conceito. Como nas provas de conceito anteriores, percebe-se um aumento na complexidade à medida que a prova se desenvolve, chegando ao valor máximo de 65. Esse crescimento é devido ao fato de estarem sendo analisadas aplicações novas e não cenários de refatoração de uma aplicação já existente, o que poderia causar uma diminuição desse valor.

[52] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/7cd9a7615e4d882d60ccfb6a839c632e67e6d3ec>. Último acesso em: Novembro. 2023
[53] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/9ad48b0a77be49f2c33fffc3abfb4718f50041a2>. Último acesso em: Novembro. 2023
[54] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/44ab0b77f9e90d03fb29f1108e78eb255deacc74>. Último acesso em: Novembro. 2023
[55] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/27b6a0378f561e8b0ae973c34374cfb6497b664a>. Último acesso em: Novembro. 2023
[56] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/dd8d8c7e0ba1096c04ef87c369d8e9d21a3f216a>. Último acesso em: Novembro. 2023

Figura 33 – Histórico de duplicação de código relacionado à quarta POC

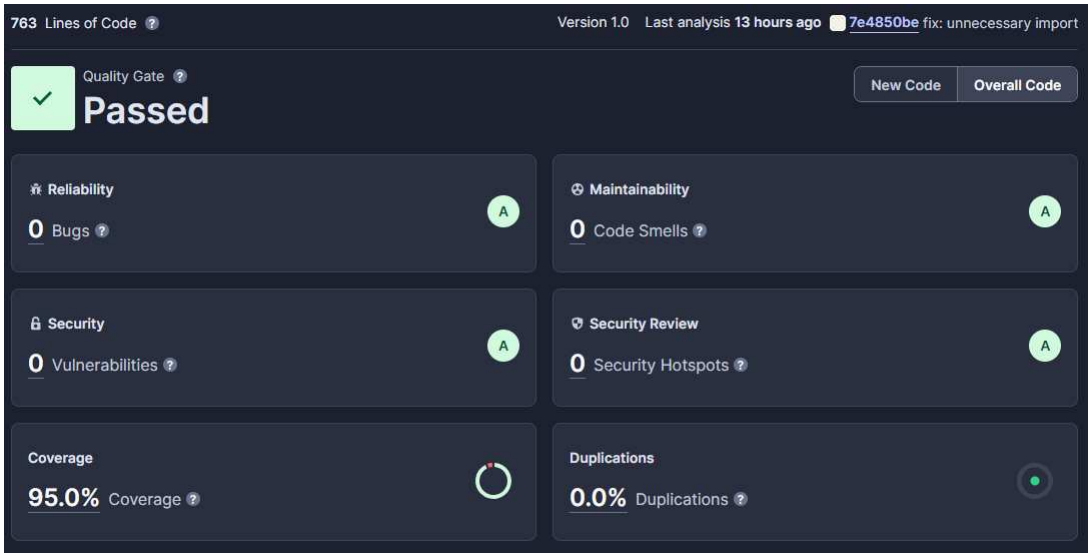


Fonte: Autor

A Figura 35 apresenta um breve resumo do SonarCloud sobre o código da quarta prova de conceito. Assim como nas provas de conceito anteriores, observa-se a presença de notas A em todos os aspectos de relevância da avaliação do SonarCloud.

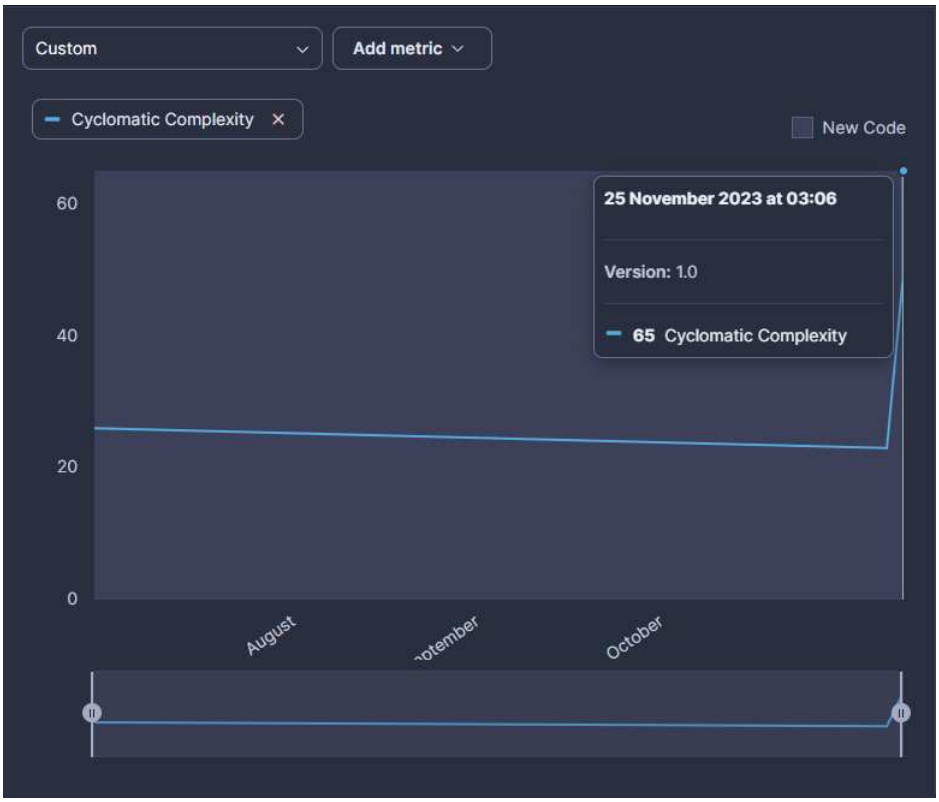
Além disso, é possível notar que a cobertura de código nesta prova de conceito foi de 95%, e a quantidade de duplicações de código foi de 0%. Dessa forma, percebe-se, através do sumário, que a solução apresentada nesta prova de conceito atendeu a todos os critérios de avaliação do SonarCloud.

Figura 35 – Sumário da análise do SonarCloud relacionado à quarta POC



Fonte: Autor

Figura 34 – Complexidade ciclomática relacionada à quarta POC



Fonte: Autor

5.7.4.1 Conclusão

Após a Análise de Resultados desta prova de conceito, constatou-se que ela atendeu a todos os requisitos e padrões de qualidade estabelecidos anteriormente. Além disso, foi possível apresentar o cenário de integração de contextos diferentes em uma mesma aplicação quando orientado à Arquitetura Limpa. É possível estabelecer um padrão de teste entre as aplicações criadas com Arquitetura Limpa, uma vez que elas facilmente alcançam uma cobertura de código superior a 80%, sendo esse valor uma meta comum para a cobertura de códigos de aplicações.

5.8 POC 5 - Desenvolvimento do Checkout

Essa Seção descreve o desenvolvimento do *micro front-end* de *checkout*, sendo este o responsável por permitir com que o usuário finalize a compra dos produtos adicionados ao carrinho. A Seção apresenta a Definição do Desafio, seguido pelos Requisitos do Desafio.

5.8.1 Definição do Desafio

Espera-se que essa prova de conceito seja capaz de prover a tela de *checkout*, permitindo com que o usuário finalize um pedido de compra, validando as informações fornecidas e calculando os valores relacionados ao pedido. Além disso, é esperado que a solução seja construída utilizando de forma combinada a Arquitetura Limpa e *Micro Front-end*.

5.8.2 Requisitos do Desafio

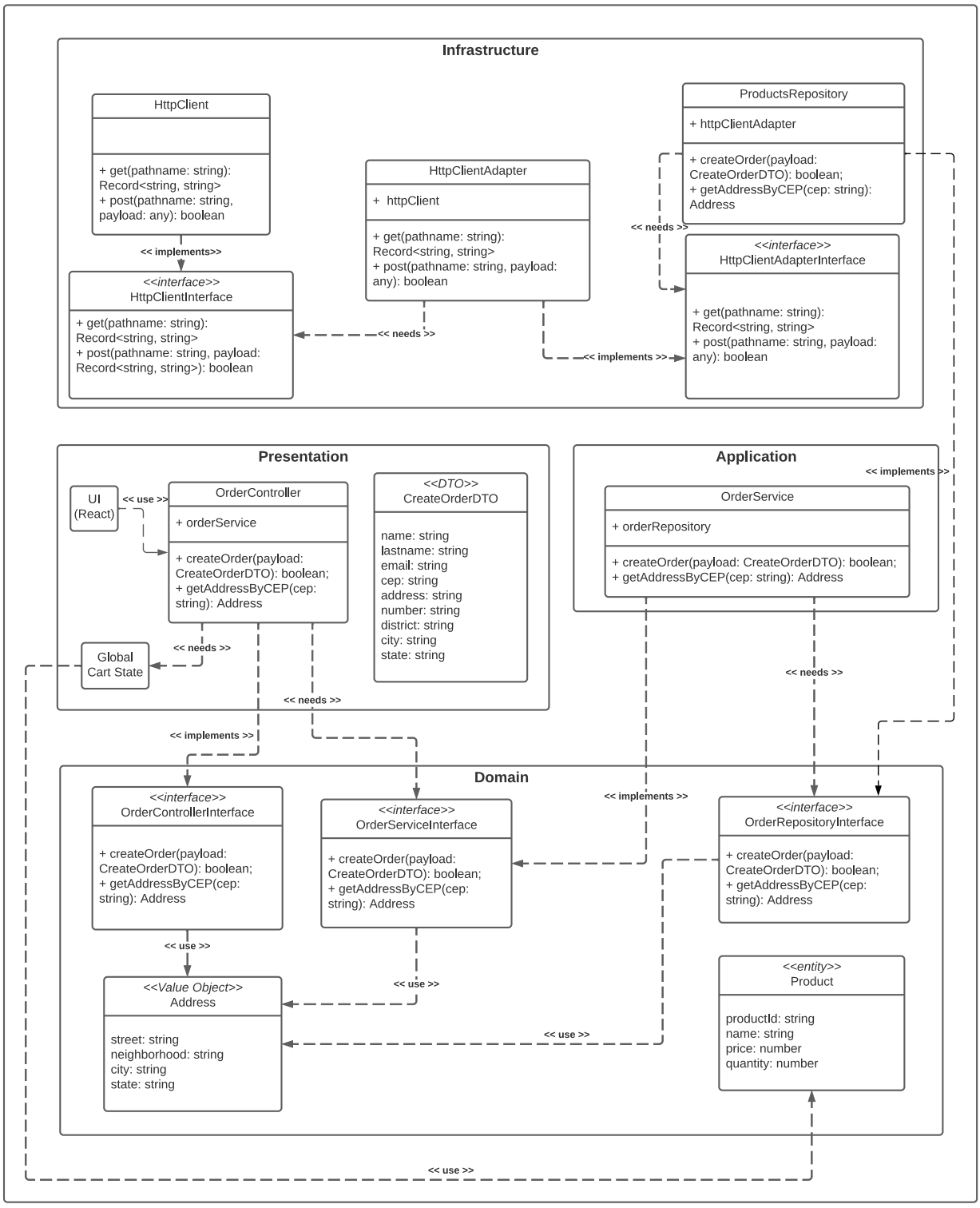
Para que essa prova de conceito seja considerada concluída, os seguintes requisitos devem ser atendidos:

- O usuário deve conseguir preencher todos os dados do formulário de *checkout*;
- O usuário deve conseguir visualizar todos os valores relacionados ao seu pedido;
- O usuário deve conseguir navegar de volta para outras telas sem perder o estado do carrinho de compras, e
- O usuário deve conseguir finalizar o pedido de compra, caso todos os dados do formulário de *checkout* estejam preenchidos corretamente.

5.8.3 Apresentação da Solução

A última prova de conceito concentrou todos os conceitos e conhecimentos adquiridos nas anteriores, através do Método Orientado a Provas de Conceito em formato *building block*, onde cada prova de conceito era a base para a próxima, fornecendo os elementos necessários para a construção dos diferentes cenários previstos em cada POC. A primeira atividade para o desenvolvimento da solução dessa prova de conceito foi a criação do protótipo usando a ferramenta Figma, descrita no Capítulo 3 - Suporte Tecnológico. Em seguida, iniciou-se o desenvolvimento do código da solução, com base no protótipo e na Arquitetura Limpa. Finalmente, foram elaborados os testes automatizados da solução. A Figura 28 apresenta a solução.

Figura 36 – Solução da quinta prova de conceito



Fonte: Autor

De acordo com os padrões de camadas estabelecidos nos desafios anteriores, o quinto desafio apresenta uma novidade: a introdução de objetos de valores. Na prova de conceito em questão, foi criada a capacidade de localizar o endereço utilizando o CEP fornecido pelo usuário. Para representar o formato de retorno desta busca, foi criado

um objeto de valor chamado *Address*. De acordo com Khorikov (2016), objetos de valor diferem de entidades por não possuírem um identificador próprio, sendo comparados pelos seus campos, e não por um identificador específico. Isso significa que objetos de valores não têm um identificador e, se dois objetos tiverem os mesmos campos com os mesmos valores, serão tratados como se fossem iguais. No entanto, se duas entidades tiverem campos com os mesmos valores e identificadores diferentes, elas não serão consideradas a mesma entidade.

Além disso, nessa solução específica, houve a integração com uma API (*Application Programming Interface*) externa para consulta dos dados de endereço. Este cenário apresentou-se como uma situação distinta, mas não causou alterações significativas na estrutura de camadas e módulos herdada das provas de conceito anteriores. A alteração mais significativa ocorreu no módulo *httpClient*, que, ao invés de retornar um dado salvo em memória, realizou uma requisição HTTP a um endpoint público de consulta de endereços e, posteriormente, retornou o resultado obtido.

Em suma, nesta solução, caso seja necessário alterar a fonte de dados de uma API externa para outra fonte de dados, basta criar um novo cliente que se adapte a essa fonte e atenda às *interfaces* estabelecidas para os clientes, de forma que a aplicação continuará funcionando normalmente sem alterações significativas.

5.8.4 Análise de Resultados

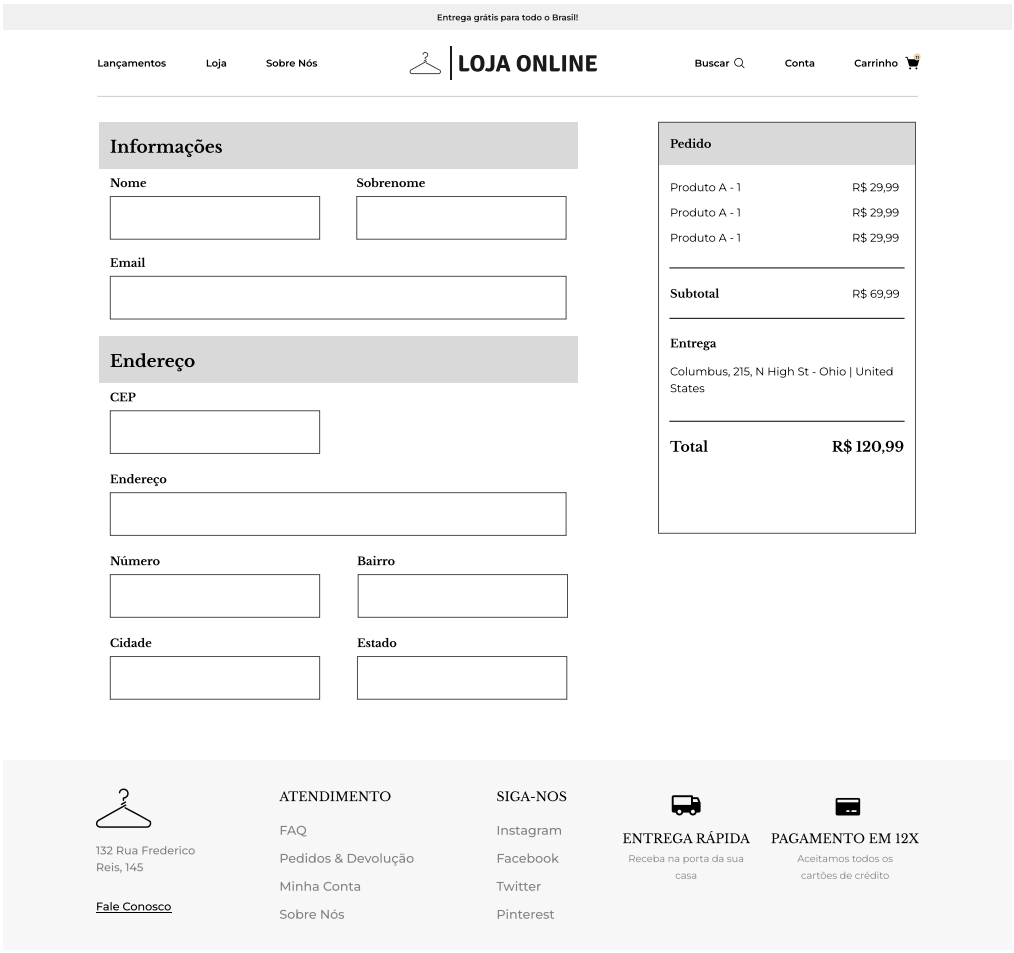
Nesta subseção, serão apresentados os resultados alcançados com o desenvolvimento da solução apresentada anteriormente. Nesta solução, foram concluídas as seguintes tarefas: (i) a criação de um protótipo para a quinta prova de conceito, e (ii) desenvolvimento da quinta prova de conceito segundo os princípios da Arquitetura Limpa e as exigências dessa POC.

A seguir, será apresentada a tela de *checkout*, sendo essa o principal artefato desse desafio, e a tela de pedido realizado. Após essas telas, serão apresentados os resultados alcançados em relação aos requisitos estabelecidos para a quinta prova de conceito, bem como os resultados alcançados em relação aos objetivos específicos do trabalho. Finalmente, é apresentada a descrição da avaliação quantitativa referente a esse quinto desafio, seguido da conclusão.

A Figura 37 apresenta a tela de *checkout*, na qual o usuário pode preencher todos os dados necessários para o pedido, como nome, e-mail, endereço e outros. Além disso, foi implementada a função de busca de endereço por CEP. Quando o usuário informa o CEP, é realizada uma busca em uma API on-line gratuita para encontrar as informações como a rua, a cidade e o estado. Dessa forma, preenche-se os campos com o objetivo de acelerar o processo de preenchimento do formulário. Além disso, esta tela apresenta, na

barra ao lado direito, as informações do pedido, tais como produtos selecionados, preços, endereço de entrega e o botão para finalizar o pedido. As informações fornecidas nessa página não são utilizadas ou enviadas para algum servidor. O objetivo deste formulário era criar uma tela comum em aplicações de loja on-line que atendesse aos princípios de Arquitetura Limpa.

Figura 37 – Tela de *checkout* relacionado à quinta POC



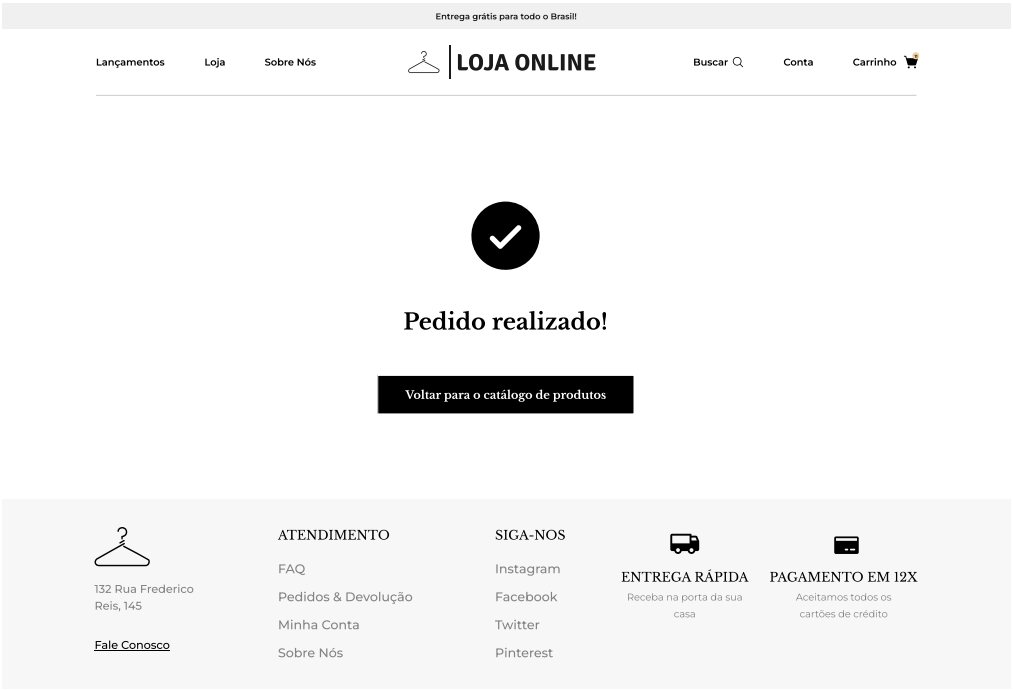
Fonte: Autor

A Figura 38 apresenta a tela final da aplicação, responsável por informar ao usuário que o pedido foi realizado. Além disso, essa tela permite que o usuário retorne ao início do processo.

O especialista, que é o mesmo das provas anteriores, percebeu o seguinte comportamento positivo da Arquitetura Limpa em aplicações *front-end*:

Dada a utilização de adaptadores e clientes, foi possível, com maior facilidade, integrar a solução com uma API externa. Além disso, dada a organização em camadas, caso seja necessário, posteriormente, alterar a fonte de dados, será possível fazer isso sem a necessidade de modificar os outros módulos. Isso corrobora a ideia de que as diferentes

Figura 38 – Tela de pedido realizado relacionado à quinta POC



Fonte: Autor

camadas apresentadas na Arquitetura Limpa possibilitam desenvolver soluções coesas e com baixo acoplamento, uma vez que cada uma delas é responsável por um aspecto específico da aplicação.

A solução desta prova de conceito atendeu a todos os requisitos estabelecidos nos Requisitos do Desafio. São eles:

- O usuário deve conseguir preencher todos os dados do formulário de *checkout*;
- O usuário deve conseguir visualizar todos os valores relacionados ao seu pedido;
- O usuário deve conseguir navegar de volta para outras telas sem perder o estado do carrinho de compras, e
- O usuário deve conseguir finalizar o pedido de compra, caso todos os dados do formulário de *checkout* estejam preenchidos corretamente.

Com os resultados obtidos com essa solução, os seguintes objetivos do trabalho foram alcançados integralmente:

- Identificação, estudo, documentação e aplicação dos principais conceitos relacionados à Arquitetura Limpa em *micro front-ends*, e

- Condução e documentação do processo de desenvolvimento, orientando-se por Provas de Conceito.

Da mesma forma que nos desafios anteriores, a Figura 39 apresenta um histórico de problemas no código. Inicialmente, é possível notar que não houve problema algum no início do desenvolvimento da solução. No entanto, ocorreu um pico de dois problemas na metade do desenvolvimento, mas, logo em seguida, foram realizadas as correções desses dois problemas identificados. Os dois problemas eram idênticos e estavam no mesmo arquivo, sendo assim, foi possível resolver os dois ao mesmo tempo.

O problema apresentado pelo SonarCloud, na solução desse desafio, foi solucionado através do seguinte *commit*:

Commit [57]:

Tipo de Problema: Manutenibilidade.

Contexto: O componente de *Input* realizava uma conversão de texto para booleano para validar se o campo está inválido ou não.

Problema: No JavaScript, qualquer tipo de dado pode ser transformado em booleano. A conversão de um dado para booleano pode ser realizada mediante uma negação dupla. Conforme o contexto, essa conversão pode ser redundante, uma vez que o JavaScript realiza a conversão automática em operadores lógicos, condicionais ou qualquer outro contexto booleano.

Solução: A conversão para booleano através de negação dupla foi excluída.

A Figura 40 mostra o histórico de cobertura de testes. Realizou-se um total de 15 *commits* visando atingir a maior cobertura possível no código deste desafio. Conforme a Figura demonstra, foi possível atingir o percentual de 95% de cobertura de código. Os commits foram:

- *Commit* [58]: Criação de testes para o componente de *input*;
- *Commit* [59]: Criação de testes para o módulo *HttpClient*;
- *Commit* [60]: Criação de testes para o módulo *HttpClientAdapter*;

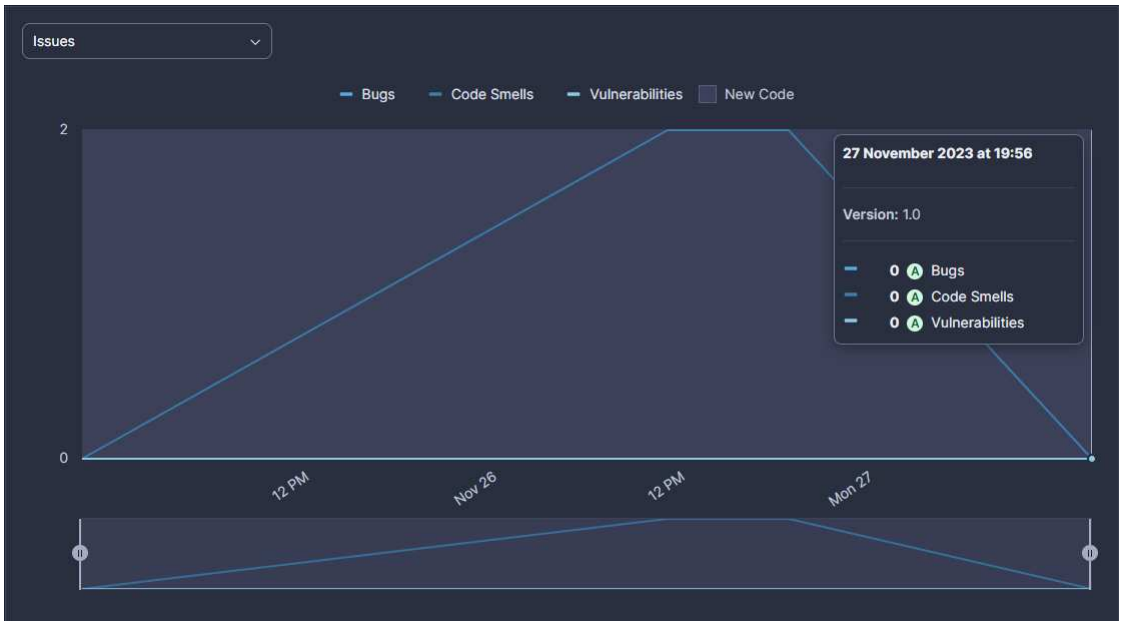
[57] Disponível em: <https://github.com/twistershark/tcc-ecommerce-cart/commit/4063365b23686f69210ce65bb2833c889b0b21e4>. Último acesso em: Novembro. 2023

[58] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/eeda6ffb47c5ea6d3ecd981cc420be09636be3f8>. Último acesso em: Novembro. 2023

[59] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/5c001393fbef7138e7920bed6eb58e52568f8191>. Último acesso em: Novembro. 2023

[60] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/3ce5195ef5387a6d6086097a635c32344afbbddc>. Último acesso em: Novembro. 2023

Figura 39 – Histórico de problemas de código relacionado à quinta POC



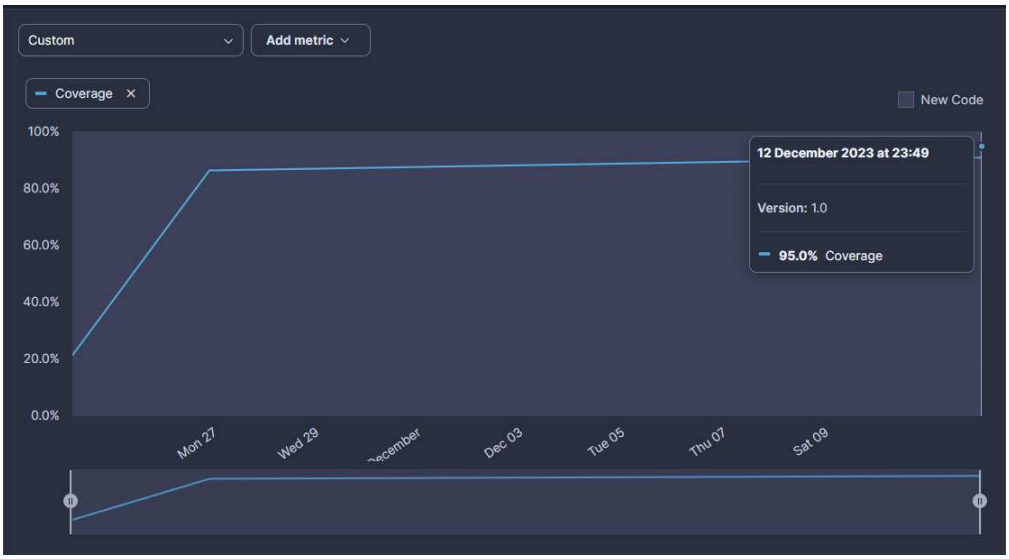
Fonte: Autor

- *Commit* [61]: Criação de *mock* do módulo *HttpClient*;
- *Commit* [62]: Criação dos testes do módulo *OrderService*;
- *Commit* [63]: Criação de *mock* do módulo *OrderRepository*;
- *Commit* [64]: Adição de caso de teste para a função *formatCurrency*;
- *Commit* [65]: Criação de *mock* do módulo *OrderService*;
- *Commit* [66]: Criação de testes automatizados para o módulo *OrderRepository*;
- *Commit* [67]: Criação de testes automatizados para o módulo *OrderController*;
- *Commit* [68]: Atualização dos casos de uso do componente *input*;

[61] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/256ec4205d5872350657a6151b2a4213aafd8c80>. Último acesso em: Novembro. 2023
[62] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/f7e560b07f1845ea003e69745024d4b4672f7b6b>. Último acesso em: Novembro. 2023
[63] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/6dbc4bf635af1f073cbc0d150f4c8ab2dfd21361>. Último acesso em: Novembro. 2023
[64] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/2c5908df205b9223663a02131b3091ca05c70e40>. Último acesso em: Novembro. 2023
[65] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/12ff7b92da7eb53173587681a649beeac879f46e>. Último acesso em: Novembro. 2023
[66] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/4e3f281e7810fdd8263ce23d5bad3f58a18f96a0>. Último acesso em: Novembro. 2023
[67] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/ae00c8035167df5be8405d8dc0c71aabd36afda0>. Último acesso em: Novembro. 2023
[68] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/c63e6b6bfb25b119db22733c122734cf31129808>. Último acesso em: Novembro. 2023

- *Commit* [69]: Criação de testes para a página de *checkout*, e
- *Commit* [70]: Criação de para a página de pedido realizado.
- *Commit* [71]: Adição de novos casos de teste para a página de checkout.
- *Commit* [72]: Novos casos de teste para o checkout.

Figura 40 – Histórico de cobertura de código relacionado à quinta POC



Fonte: Autor

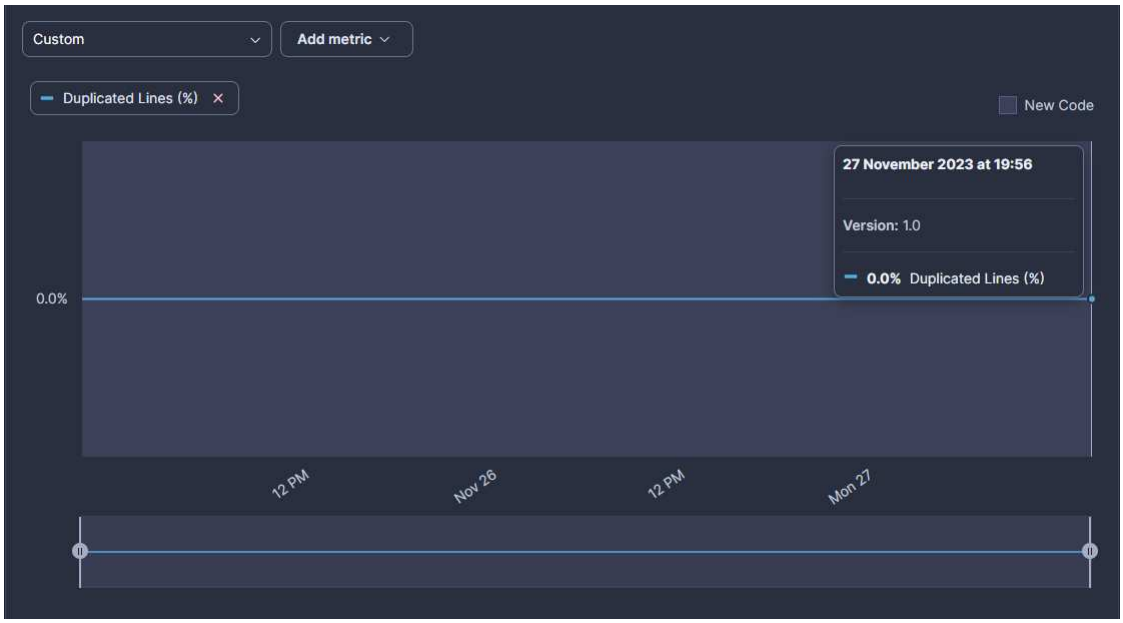
A Figura 41 apresenta, finalmente, o registro de duplicação de código durante o desenvolvimento desta quinta prova de conceito. Observa-se pela Figura que, desde o início até o término do processo de desenvolvimento, não foram detectados códigos duplicados na solução desta prova de conceito.

A Figura 42 apresenta a complexidade ciclomática dessa quinta prova de conceito. Mantendo o padrão das provas de conceito anteriores e o cenário de criação de uma nova aplicação ao invés da refatoração de uma já existente, ocorre um aumento na complexidade à medida que a prova se desenvolve até atingir o valor de 58.

A Figura 43 apresenta um resumo da análise do SonarCloud sobre o código da quinta prova de conceito. A Figura apresenta que, em todos os quesitos de avaliação

[69] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/8b0055a3e7308d73e7ab8bb769af095f0e34868a>. Último acesso em: Novembro. 2023
[70] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/b0654d28ccbb2a4d68e79a136e9194df72506449>. Último acesso em: Novembro. 2023
[71] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/97e74bdcdfaa42fefd9ddb7a4cc08d3fe73dc127>. Último acesso em: Dezembro. 2023
[72] Disponível em: <https://github.com/twistershark/tcc-ecommerce-checkout/commit/5ee6d5235caa9ca218c257d635ec45dc7d870482>. Último acesso em: Dezembro. 2023

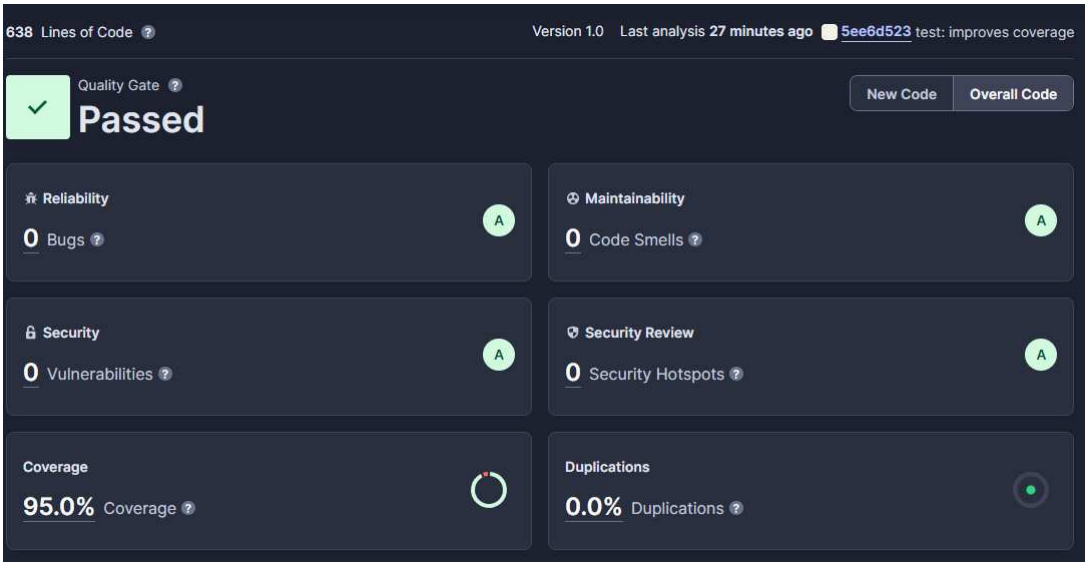
Figura 41 – Histórico de duplicação de código relacionado à quinta POC



Fonte: Autor

relevantes para esse trabalho, a solução desenvolvida obteve a melhor nota possível, representada no SonarCloud pela letra A. A Figura também reforça a cobertura de código total que foi de 95% e a quantidade de duplicação de código que manteve-se em 0%.

Figura 43 – Sumário da análise do SonarCloud relacionado à quinta POC

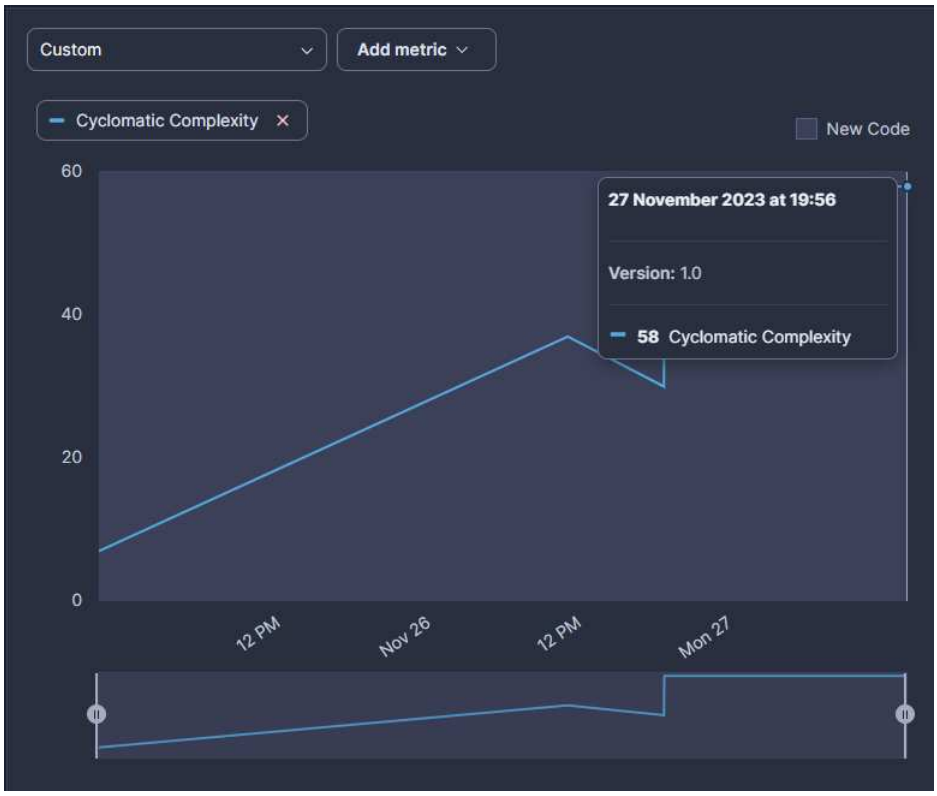


Fonte: Autor

5.8.4.1 Conclusão

Com base na Análise de Resultados desta prova de conceito, verificou-se que ela cumpriu com todos os requisitos e padrões de qualidade estabelecidos no trabalho. Além

Figura 42 – Complexidade ciclomática relacionada à quinta POC



Fonte: Autor

disso, esta prova de conceito tratou do cenário de integração com uma API externa ao invés de uma simulação com dados em memória. Esse cenário é relevante porque demonstra a estrutura de código de uma aplicação *micro front-end* que segue os princípios de Arquitetura Limpa e que também consome dados de uma fonte externa, o que a aproxima mais do cenário real de aplicações *front-end*.

5.9 Considerações Finais do Capítulo

Esse capítulo apresentou o processo de desenvolvimento do estudo exploratório do trabalho: uma Loja On-line, descrevendo desde a motivação, a contextualização, perpassando pela descrição da proposta, seguida da documentação das provas de conceito necessárias e dos objetivos do trabalho. Para cada prova de conceito documentada, foram apresentados os objetivos e requisitos necessários para a sua conclusão, além de documentar o seu desenvolvimento e os resultados obtidos.

6 Análise de Resultados

Este capítulo apresentará um resumo da análise de resultado realizada em cada prova de conceito no Capítulo 5 - Estudo Exploratório. Inicialmente, será fornecida uma tabela contendo os Resultados das Provas de Conceito, bem como uma descrição de cada resultado. Depois, será apresentada uma compilação dos Aspectos Observados pelo especialista, cujo perfil está descrito na Seção Método de Análise de Resultados (4.5). Em seguida, será apresentado o Resultado da Revisão por Pares com a compilação das respostas aos Questionários de Revisão por Pares das Provas de Conceito, bem como um resumo das opiniões dos revisores sobre cada prova de conceito. Finalmente, serão apresentadas as Considerações Finais do Capítulo.

6.1 Resultados das Provas de Conceito

Após uma análise aprofundada da literatura, planejamento, desenvolvimento e apresentação das soluções de cada prova de conceito, foi descrita a análise realizada pelo SonarCloud sobre a solução em questão, conforme descrito na Seção Método de Análise de Resultados (4.5). Os aspectos considerados relevantes para o trabalho foram: Problemas no código[1], cobertura de código, duplicação de código e complexidade ciclomática. A Tabela 7 apresenta um resumo desses aspectos na solução desenvolvida em cada prova de conceito.

Tabela 7 – Resultados da Análise Quantitativa das Provas de Conceito

Prova de Conceito	**Critério 1**	**Critério 2**	**Critério 3**	**Critério 4**
POC 1	0	-	-	-
POC 2	0	100%	0	80
POC 3	0	97%	0	49
POC 4	0	95%	0	65
POC 5	0	95%	0	58

- Critério 1 representa a quantidade de problemas no código;
- Critério 2 representa a cobertura de testes;
- Critério 3 representa a quantidade de linhas duplicadas na solução, e
- Critério 4 representa a complexidade ciclomática.

Fonte: Autor

[1] Entende-se problemas como *bugs*, falhas de segurança, código não utilizado, entre outros.

O SonarCloud segue um padrão de qualidade para o código de aplicações JavaScript/TypeScript, analisando aspectos como a quantidade de *bugs*, vulnerabilidades, cobertura de código, duplicação, dentre outros. Quando uma aplicação atinge o nível mínimo necessário em todos os aspectos, o SonarCloud considera que aquela aplicação atingiu o padrão de qualidade esperado. Sendo assim, é relevante salientar que todas as soluções desenvolvidas atenderam aos requisitos de qualidade estabelecidos pelo SonarCloud.

6.2 Aspectos Observados

Durante o desenvolvimento da solução de cada prova de conceito, alguns aspectos positivos e negativos foram observados pelo especialista. Esses aspectos são importantes para o trabalho, pois através deles pode-se reforçar ou não algumas características da Arquitetura Limpa e *micro front-end*, além de identificar cenários que surgiram durante o uso combinado das duas. Os aspectos positivos e negativos observados ao longo das cinco provas de conceito foram:

Os aspectos positivos observados ao longo das cinco provas de conceito foram:

- Separação de responsabilidades
  Descrito na Análise de Resultados da Primeira Prova de Conceito e Análise de Resultados da Segunda Prova de Conceito.
- Facilidade na manutenção
  Descrito na Análise de Resultados da Primeira Prova de Conceito.
- Testabilidade
  Descrito na Análise de Resultados da Segunda Prova de Conceito.
- Manutenibilidade
  Descrito na Análise de Resultados da Segunda Prova de Conceito.
- Facilidade para integração com APIs externas sem afetar outros módulos da aplicação
  Descrito na Análise de Resultados da Quinta Prova de Conceito.

Contudo, alguns aspectos negativos também foram observados ao longo do estudo. Foram eles:

- Dependência cíclica
  Descrito na Análise de Resultados da Primeira Prova de Conceito.
- Conflito de classes CSS
  Descrito na Análise de Resultados da Primeira Prova de Conceito.

- Curva de aprendizado
  Descrito na Análise de Resultados da Primeira Prova de Conceito e Análise de Resultados da Segunda Prova de Conceito.

- Complexidade
  Descrito na Análise de Resultados da Segunda Prova de Conceito.

- Tempo de desenvolvimento
  Descrito na Análise de Resultados da Segunda Prova de Conceito.

- Testes de integração diretamente dependentes do React
  Descrito na Análise de Resultados da Quarta Prova de Conceito.

6.3 Resultado da Revisão por Pares

As Tabelas 8 e 9 apresentam as respostas dos revisores, apresentados na Seção Método de Análise de Resultados (4.5), sobre a solução e os comportamentos apresentados em cada prova de conceito. As perguntas foram:

- Pergunta 1: Há independência entre os componentes arquiteturais?

- Pergunta 2: Há facilidade em termos de escalabilidade?

- Pergunta 3: Há facilidade para substituir um dado aspecto no front-end?

- Pergunta 4: Você concorda com os comportamentos relatados pelo autor? Se não concordar, explique quais são esses comportamentos e o motivo.

Tabela 8 – Resumo dos questionários respondidos pelo Revisor A

Prova de Conceito	**Pergunta 1**	**Pergunta 2**	**Pergunta 3**	**Pergunta 4**
Questionário da POC 1	Sim	Sim	Sim	Sim
Questionário da POC 2	Sim	Sim	Sim	Sim
Questionário da POC 3	Sim	Sim	Sim	Sim
Questionário da POC 4	Sim	Sim	Sim	Sim
Questionário da POC 5	Sim	Sim	Sim	Sim

Fonte: Autor

Tabela 9 – Resumo dos questionários respondidos pelo Revisor B

Prova de Conceito	**Pergunta 1**	**Pergunta 2**	**Pergunta 3**	**Pergunta 4**
Questionário da POC 1	Sim	Sim	Sim	Sim
Questionário da POC 2	Sim	Sim	Sim	Sim
Questionário da POC 3	Sim	Sim	Sim	Sim
Questionário da POC 4	Sim	Sim	Sim	Sim
Questionário da POC 5	Sim	Sim	Sim	Sim

Fonte: Autor

Em cada questionário, cada revisor teve a oportunidade de compartilhar, de acordo com sua experiência, algo que desejasse acrescentar, com base na sua própria experiência. O Revisor B não encontrou algo relevante para acrescentar, mas o Revisor A destacou um aspecto relevante de aplicações criadas com a arquitetura de *micro front-end*. Esse foi o aspecto descrito pelo Revisor A:

> [Revisor A] "É fundamental destacar a importância de evitar a transformação de um micro frontend em um nano frontend. Portanto, compreender o contexto operacional é essencial, especialmente ao lidar com sistemas distribuídos, que apresentam desafios específicos exigindo uma consideração cuidadosa.
>
> Quanto ao estudo acadêmico, algumas partes da exploração podem assumir um caráter simbólico, tornando-se complexo demonstrar todos os benefícios dos padrões mencionados. Em ambientes de negócios, a separação de preocupações em torno da lógica de negócios é evidente, e os limites entre os contextos ficam explícitos, resultando em uma abordagem mais focada em torno de um contexto de negócio.
>
> Tomemos como exemplo um sistema de comércio eletrônico, representado por um contexto delimitado, onde os limites e domínios são claros. Nesse cenário, teríamos domínios como (Pedido, Produto, Cobrança e Entrega). Notavelmente, teríamos um sistema de pedidos como um componente central, que não lida apenas com o domínio de pedidos, mas também com produtos, cobranças após a criação do pedido e todo o processo de entrega. Isolar cada domínio em um serviço único se tornaria um nano serviço, algo custoso e com alta dependência entre contextos. Por exemplo, o serviço de pedidos não poderia operar independentemente se o serviço de produtos estivesse inativo, pois existe uma dependência explícita no contexto, o que representa a própria coesão.
>
> Dado esse exemplo, é crucial entender como os limites entre os micro frontends são definidos. Eles abordam contextos específicos, ou são baseados em

> domínios ou funcionalidades ao dividir uma interface de usuário? Essa questão é vital em cenários reais, pois a eficiência técnica, especialmente em termos de custo de implementação, é um fator crucial para o sucesso de qualquer negócio."

6.4 Considerações Finais do Capítulo

Este capítulo apresentou os resultados obtidos através do estudo exploratório realizado neste trabalho, fornecendo uma visão sobre a análise quantitativa e qualitativa realizada em cada prova de conceito. Apresentaram-se os resultados referentes aos problemas de código, duplicação de código, cobertura de testes e complexidade ciclomática de cada prova de conceito. Além disso, os aspectos positivos e negativos observados pelo especialista em cada prova de conceito foram reportados em conjunto, dando uma visão geral de tudo o que foi observado ao longo do desenvolvimento da Loja On-line. Finalmente, foram apresentadas tabelas contendo uma compilação das respostas fornecidas pelos revisores para cada POC, terminando com um resumo das experiências compartilhadas pelos revisores em relação ao contexto do trabalho.

7 Conclusão

Este capítulo apresentará a conclusão do trabalho. Inicialmente, será retomado o Contexto que motivou a realização deste trabalho. Em seguida, será apresentado o Status do Trabalho, fornecendo uma visão geral dos objetivos do trabalho, da questão de pesquisa e dos principais comportamentos revelados ao longo do Estudo Exploratório. Finalmente, será apresentada uma perspectiva sobre Futuros Trabalhos, enfatizando as oportunidades de aprimoramento e evolução deste Estudo Exploratório.

7.1 Contexto

Conforme abordado no Capítulo 1 - Introdução, a arquitetura de software, quando aplicada corretamente, resulta em fácil entendimento, desenvolvimento, manutenção e implantação do software (MARTIN, 2017). Atualmente, a decomposição de aplicações em módulos menores é uma prática bastante comum em arquiteturas de microsserviços (LEWIS; FOWLER, 2019). A arquitetura *micro front-end* surge no contexto de microsserviços, visando permitir a criação de pequenas aplicações *front-end* que, quando combinadas, formam uma aplicação completa. No entanto, muitas empresas ainda têm dificuldades para implementar *micro front-ends*, devido à falta de uma visão mais prática em termos de implementação (JACKSON, 2019)

Além disso, o uso de um padrão arquitetural sem planejamento pode acarretar dificuldades na manutenção e na escalabilidade do software (MONTELIUS, 2021). Nesse cenário, além dos *micro front-ends* que permitem a divisão de uma aplicação complexa em aplicações menores, existem arquiteturas que separam os diferentes módulos de um sistema em diferentes camadas, a fim de proporcionar maior coesão e menos acoplamento ao código (MARTIN, 2017).

Dessa forma, surge a oportunidade de estudar quais comportamentos são revelados ao criar-se uma aplicação *front-end* utilizando, de forma combinada, a Arquitetura Limpa e a *micro front-end*, sendo este o contexto abordado neste trabalho.

7.2 Status do Trabalho

Ainda na Introdução, estabeleceu-se o objetivo geral do trabalho. Consequentemente, visando cumprir esse objetivo, foram estabelecidos os seguintes objetivos específicos:

- Identificação, estudo, documentação e aplicação das principais técnicas relacionadas

ao desenvolvimento de aplicações web utilizando-se de *micro front-ends*.
Status: Concluído na POC 1 - Integração dos *Micro Front-ends*;

- Integração dos *micro front-ends*, procurando manter suas respectivas independências e responsabilidades.
  Status: Concluído na POC 1 - Integração dos *Micro Front-ends*;

- Identificação, estudo, documentação e aplicação dos principais conceitos relacionados à Arquitetura Limpa em *micro front-ends*.
  Status: Concluído no Capítulo 5 - Estudo Exploratório, e

- Condução e documentação do processo de desenvolvimento - inerente ao cumprimento dos objetivos específicos anteriores - orientando-se por Provas de Conceito.
  Status: Concluído no Capítulo 5 - Estudo Exploratório.

Com os objetivos específicos concluídos, tem-se que o objetivo geral também foi concluído, sendo ele o de realizar um estudo exploratório orientado a Provas de Conceito, revelando os comportamentos observados ao longo do desenvolvimento de uma Loja Online, e orientando-se pela combinação de Arquitetura Limpa e *Micro front-ends*.

A definição de um objetivo geral tem como propósito responder a uma questão específica de pesquisa. O presente trabalho responde à seguinte questão de pesquisa: quais são os comportamentos de relevância revelados ao longo do desenvolvimento de uma loja on-line usando de forma combinada Arquitetura Limpa e *Micro front-ends*?

É possível concluir que esta questão foi respondida através do Estudo Exploratório conduzido, que descreveu todos os comportamentos relevantes para o trabalho. Conforme o Método de Análise de Resultados, após a análise do especialista, revisão por pares e relatórios do SonarCloud, foram identificados, sobretudo, os seguintes comportamentos na utilização combinada de Arquitetura Limpa e *Micro Front-ends*:

- Independência entre os componentes arquiteturais, característica resultando da separação dos diversos aspectos das aplicações em diferentes camadas com responsabilidades específicas;

- A facilidade de se trabalhar com diferentes tecnologias, sendo um dos benefícios de se usar uma arquitetura em camadas, como a Arquitetura Limpa, na qual a camada mais externa, responsável por criar a interface do usuário, deveria funcionar de forma *Plug and Play*, podendo ser alterada sem causar danos às camadas mais internas da aplicação;

- Facilidade de escalabilidade alcançada pelo uso correto das arquiteturas do estudo e suportada pela análise do SonarCloud. Essa análise mostrou que todas as provas de

conceito desenvolvidas obtiveram nota máxima no que diz respeito à manutenibilidade, o que significa que correções ou novas funcionalidades são mais rápidas e fáceis de serem desenvolvidas devido à ausência de bugs, débitos técnicos, duplicação de código, além de todas as aplicações terem atingido o nível de cobertura de testes esperado;

- Repositórios menores, resultado de uma arquitetura que se baseia em microsserviços, como a arquitetura *Micro Front-end*, dividindo uma aplicação maior em diversas outras menores e com contextos próprios;
- Ausência de código duplicado, conforme documentado na análise de resultados de cada prova de conceito, e
- A substituição de um determinado aspecto no *front-end* é uma tarefa simples, uma vez que a maioria dos elementos da *interface* gráfica contém apenas o necessário para a criação da página, não possuindo regras de negócio.

7.3 Futuros Trabalhos

Após atingir os objetivos estabelecidos neste trabalho, é possível prosseguir com os estudos sobre a arquitetura de aplicações *front-end*, explorando outros padrões arquiteturais e a sua possível combinação. Dado o escopo mais restrito do trabalho, algumas oportunidades de futuros trabalhos podem ser aproveitadas. São elas:

- Aplicar a Arquitetura Limpa em uma aplicação *front-end* com um escopo maior e diversas funcionalidades, verificando se os aspectos apresentados neste trabalho permanecem iguais;
- Aplicar a arquitetura *Micro Front-end* com outros padrões arquiteturais, a fim de criar estudos com abordagens distintas, fornecendo assim insumos relevantes para outros desenvolvedores, e
- Com base na Arquitetura Limpa e *Micro Front-end*, desenvolver uma aplicação utilizando outras tecnologias e frameworks com o objetivo de identificar a complexidade necessária para replicar a abordagem deste trabalho utilizando ferramentas diferentes.

Referências

AWATI, R.; WIGMORE, I. *What is monolithic architecture in software?* 2022. Disponível em: <https://www.techtarget.com/whatis/definition/monolithic-architecture>. Acesso em: Maio. 2023. Citado na página 27.

BALDISSERA, O. Quais são os tipos de arquitetura de software e como escolher o melhor para seu projeto. 2021. Citado na página 36.

BARZOTTO, T. R. H.; FARIAS, K. Avaliação dos impactos da decomposição de uma aplicação monolítica para microsserviços: Um estudo de caso. 2022. Citado na página 29.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice: Software Architect Practice*. [S.l.]: Addison-Wesley, 2012. Citado na página 27.

BASTOS, J. P. d. S. *Desenvolvimento Web Orientado a Micro Frontends*. Tese (Doutorado), 2020. Citado na página 29.

BP, I. W. K. D.; ANGGRAINI, D. a development of modern web application frontend structures using micro frontends. 2022. Citado na página 82.

BREUX, G. *Client-side vs. Server-side vs. Pre-rendering for Web Apps*. 2018. Disponível em: <https://www.toptal.com/front-end/client-side-vs-server-side-pre-rendering>. Acesso em: Maio. 2023. Citado 2 vezes nas páginas 43 e 45.

BROWN, K.; WOOLF, B. Implementation patterns for microservices architectures. In: *Proceedings of the 23rd Conference on Pattern Languages of Programs*. USA: The Hillside Group, 2016. (PLoP '16). Citado na página 28.

COCCA, G. *Rendering Patterns for Web Apps. Server-Side, Client-Side, and SSG Explained*. 2023. Disponível em: <https://www.freecodecamp.org/news/rendering-patterns/#server-side-rendering-ssr->. Acesso em: Maio. 2023. Citado na página 43.

CRNKOVIC, I. Component-based software engineering—new challenges in software development. *Software focus*, Wiley Online Library, v. 2, n. 4, p. 127–133, 2001. Citado 3 vezes nas páginas 46, 47 e 48.

DRUMOND, C. What is scrum and how to get started. 2023. Citado na página 72.

FILHO, M. A. A. Arquitetura de micro frontends no desenvolvimento web. 2021. Citado 2 vezes nas páginas 29 e 83.

FREECODECAMP. *How the Web Works Part II: Client-Server Model and the Structure of a Web Application*. 2015. Disponível em: <https://www.freecodecamp.org/news/how-the-web-works-part-ii-client-server-model-the-structure-of-a-web-application-735b4b6d76e3/> Acesso em: Maio. 2023. Citado na página 37.

GARG, R. *Demystifying software architecture patterns*. 2022. Disponível em: <https://www.thoughtworks.com/insights/blog/architecture/demystify-software-architecture-patterns>. Acesso em: Maio. 2023. Citado na página 27.

GEERS, M. *Micro frontends*. neuland - Büro für Informatik, 2017. Acesso em: 27 mar. 2023. Disponível em: <https://micro-frontends.org/>. Citado na página 27.

GENTIL, R. J. Um estudo de caso sobre como a introdução de comportamentos adaptativos em uma aplicação web legada impacta a cobertura de código. Universidade Federal de São Carlos, 2020. Citado na página 55.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de pesquisa*. [S.l.]: Plageder, 2009. Citado 2 vezes nas páginas 65 e 66.

GIL, A. C. et al. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas São Paulo, 2002. v. 4. Citado na página 67.

IBM. O que é arquitetura de três camadas? IBM, 2023. Citado na página 38.

JACKSON, C. *Micro Frontends*. 2019. Acesso em: 27 mar. 2023. Disponível em: <https://martinfowler.com/articles/micro-frontends.html>. Citado 3 vezes nas páginas 28, 29 e 145.

JAIN, V.; MALVIYA, B.; ARYA, S. An overview of electronic commerce (e-commerce). *Journal of Contemporary Issues in Business and Government*, Society of Business and management, v. 27, n. 3, p. 665–670, 2021. Citado na página 29.

JAISWAL, M. Software architecture and software design. *International Research Journal of Engineering and Technology (IRJET) e-ISSN*, p. 2395–0056, 2019. Citado 3 vezes nas páginas 33, 35 e 36.

KANBANIZE. What is kanban? explained for beginners. 2023. Citado na página 72.

KHORIKOV, V. Entity vs value object: the ultimate list of differences. 2016. Citado na página 131.

KINSTA. *What Is React.js? A Look at the Popular JavaScript Library*. 2023. Disponível em: <https://kinsta.com/blog/what-is-react-js>. Acesso em: Abril. 2023. Citado 2 vezes nas páginas 55 e 56.

KOJO, R. H. H. Um guia para micro frontends: estudo aprofundado e aplicação de caso de uso no ecommerce. 2021. Citado 2 vezes nas páginas 31 e 69.

LEFF, A.; RAYFIELD, J. T. Web-application development using the model/view/controller design pattern. In: IEEE. *Proceedings fifth ieee international enterprise distributed object computing conference*. [S.l.], 2001. p. 118–127. Citado na página 39.

LEWIS, J.; FOWLER, M. *Microservices*. 2019. Acesso em: Março. 2023. Disponível em: <https://martinfowler.com/articles/microservices.html>. Citado 2 vezes nas páginas 29 e 145.

MARTIN, R. C. *The Clean Architecture*. 2012. Disponível em: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Acesso em: Maio. 2023. Citado 4 vezes nas páginas 27, 39, 40 e 41.

MARTIN, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 1st. ed. USA: Prentice Hall Press, 2017. ISBN 0134494164. Citado 4 vezes nas páginas 27, 28, 94 e 145.

MDN. *Introduction to the server side*. 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction>. Acesso em: Maio. 2023. Citado na página 37.

MDN. *MVC*. 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>. Acesso em: Maio. 2023. Citado na página 27.

MONTELIUS, A. *An Exploratory Study of Micro Frontends*. 2021. Citado 7 vezes nas páginas 28, 29, 41, 68, 82, 83 e 145.

NASCIMENTO, C. P.; SOTTO, E. C. S. Microfrontend: um estudo sobre o conceito e aplicação no frontend. *Revista Interface Tecnológica*, v. 17, n. 1, p. 153–165, 2020. Citado na página 42.

NATURE, S. Processo de revisão por pares. 2023. Citado na página 155.

OLORUNTOBA, S. Solid: os primeiros 5 princípios do design orientado a objeto. 2021. Citado na página 46.

PATTERNS. *Client-side Rendering*. 2023. Disponível em: <https://www.patterns.dev/posts/client-side-rendering>. Acesso em: Maio. 2023. Citado 3 vezes nas páginas 43, 44 e 45.

PONCE, F.; MÁRQUEZ, G.; ASTUDILLO, H. Migrating from monolithic architecture to microservices: A rapid review. In: IEEE. *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. [S.l.], 2019. p. 1–7. Citado na página 28.

PRASANNA, K. et al. Poc design: a methodology for proof-of-concept (poc) development on internet of things connected dynamic environments. *Security and Communication Networks*, Hindawi Limited, v. 2021, p. 1–12, 2021. Citado na página 70.

QUEIROZ, R. C. P. de. Microfrontends com web components e webpack: Uma abordagem de implementação agnóstica em relação ao framework. 2022. Citado 3 vezes nas páginas 29, 68 e 71.

REACT. *React - The library for web and native user interfaces*. 2023. Disponível em: <https://react.dev>. Acesso em: Abril. 2023. Citado na página 56.

REDHAT. *What are microservices?* 2023. Acesso em: Março. 2023. Disponível em: <https://www.redhat.com/pt-br/topics/microservices/what-are-microservices>. Citado na página 29.

RICHARDS, M.; FORD, N. *Fundamentals of software architecture: an engineering approach*. [S.l.]: O'Reilly Media, 2020. Citado na página 33.

RICHARDSON, C. Pattern: Monolithic architecture. 2019. Citado na página 34.

ROVEDA, U. O que é programação orientada a objetos e quais são seus pilares? 2022. Citado na página 45.

SAKOVICH, N. Microservices vs. monolithic architecture comparison. 2023. Citado na página 34.

SHARMA, V. Techniques for cross micro frontend communication. 2022. Citado 2 vezes nas páginas 48 e 49.

SILVA, R. A. P. da. *A Micro Frontends Solution–Analyzing quality attributes*. Tese (Doutorado) — Instituto Politecnico do Porto (Portugal), 2021. Citado na página 29.

SOUSA, F. R. T. C. d. *Reengenharia de Arquitetura Monolítica para Arquitetura Micro Frontend*. Tese (Doutorado), 2021. Citado na página 27.

TANENBAUM, A. S. *Sistemas Distribuídos: Princípios e Paradigmas*. 2nd. ed. São Paulo, Brazil: Pearson Addison Wesley, 2007. ISBN 9788576050739. Citado na página 35.

THELMA, U. The s.o.l.i.d principles in pictures. 2020. Citado na página 46.

TILKOV, S. Don't start with a monolith. 2015. Citado na página 35.

TURILLI, M. et al. Designing workflow systems using building blocks. *arXiv preprint arXiv:1609.03484*, 2016. Citado na página 71.

VENTERS, C. C. et al. Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, Elsevier, v. 138, p. 174–188, 2018. Citado na página 33.

WATSON, A. H.; WALLACE, D. R.; MCCABE, T. J. *Structured testing: A testing methodology using the cyclomatic complexity metric*. [S.l.]: US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996. v. 500. Citado na página 70.

WILSON, G. et al. Best practices for scientific computing. *PLoS biology*, Public Library of Science San Francisco, USA, v. 12, n. 1, p. e1001745, 2014. Citado na página 45.

YANG, C.; LIU, C.; SU, Z. Research and application of micro frontends. In: IOP PUBLISHING. *IOP conference series: materials science and engineering*. [S.l.], 2019. v. 490, n. 6, p. 062082. Citado na página 41.

Apêndices

APÊNDICE A – Questionários de Revisão por Pares

Como abordado na Seção Método de Análise de Resultados (4.5), além dos relatos do especialista e da análise quantitativa baseada nos relatórios gerados pelo SonarQub, também foi realizada a revisão por pares da solução apresentada em cada prova de conceito do estudo exploratório. A revisão por pares é uma parte relevante das publicações científicas, confirmando a validade da ciência relatada (NATURE, 2023). De acordo com Nature (2023), é esperado que os revisores sejam especialistas, voluntários e dispostos a dedicar seu tempo para aprimorar o trabalho em questão. O perfil de cada especialista foi apresentado na Seção Método de Análise de Resultados (4.5) do Capítulo 4 - Metodologia.

Com o intuito de validar a solução apresentada em cada prova de conceito do trabalho, elaborou-se um questionário com seis perguntas focadas na questão de pesquisa do trabalho, com o objetivo de auxiliar o trabalho a responder à questão de pesquisa abordada no capítulo de introdução.

O questionário iniciava com um Termo de Consentimento Livre e Esclarecido (TCLE), que indicava que o revisor poderia desistir de responder ao questionário a qualquer momento, sem causar qualquer tipo de prejuízo para o revisor e sem a necessidade de explicar o motivo da desistência. Adicionalmente, foi mencionado que o propósito do questionário consistia em realizar uma revisão por pares das provas de conceito elaboradas. É importante salientar que o revisor foi informado de que sua participação não geraria qualquer tipo de remuneração, uma vez que sua participação era voluntária.

Após o termo de consentimento ser aceito, o revisor foi apresentado à Seção de revisão por pares, onde as opções de resposta eram sim, não ou outra, sendo a última através de um campo de texto livre. As perguntas apresentadas foram:

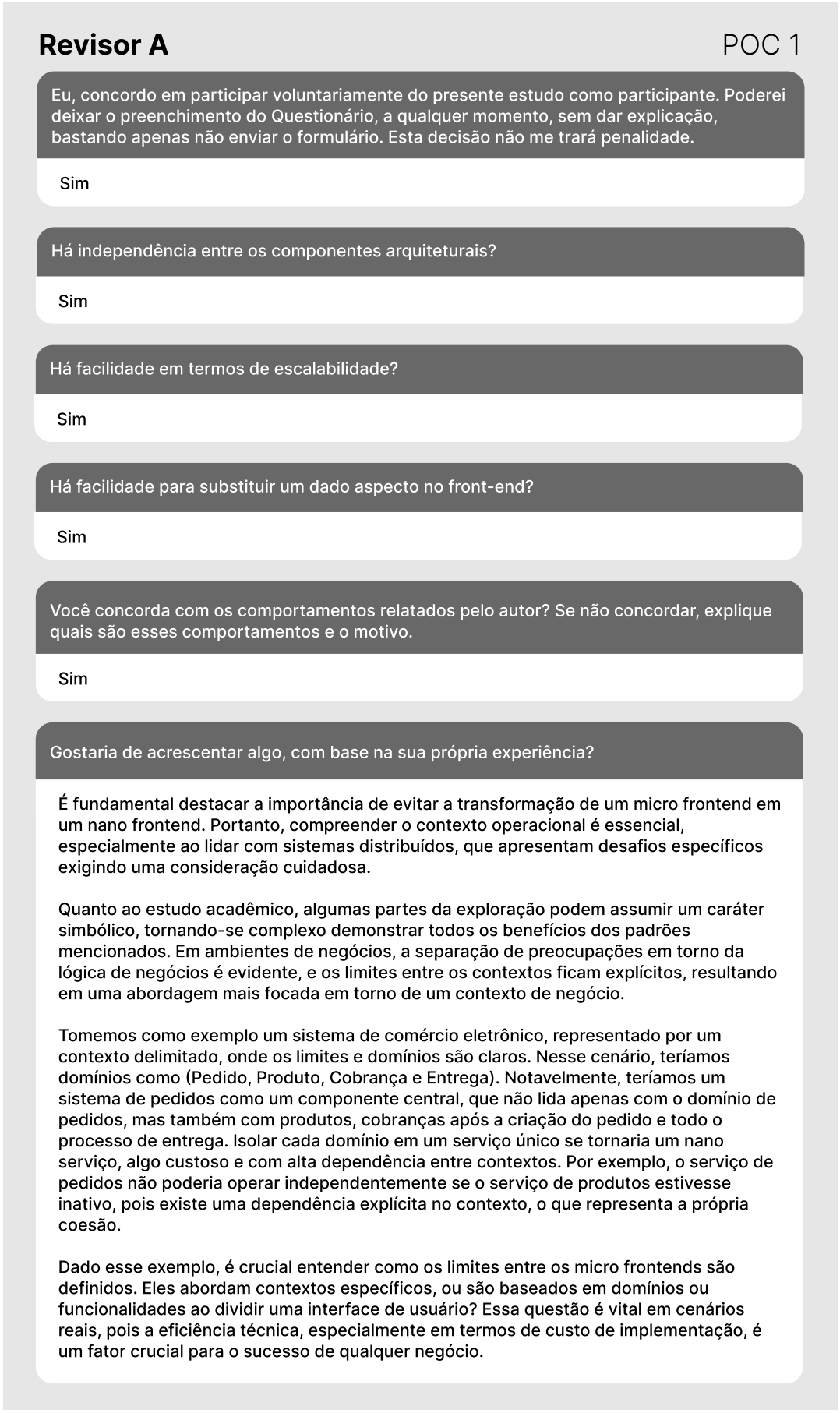
- Há independência entre os componentes arquiteturais?
- Há facilidade em termos de escalabilidade?
- Há facilidade para substituir um dado aspecto no front-end?

Em seguida, o revisor respondeu à seguinte pergunta: Você concorda com os comportamentos relatados pelo autor? Se não concordar, explique quais são esses comportamentos e o motivo.

Se o revisor desejasse acrescentar alguma observação que não estivesse prevista nas perguntas anteriores, responderia à última pergunta: Gostaria de acrescentar algo, com base na sua própria experiência?

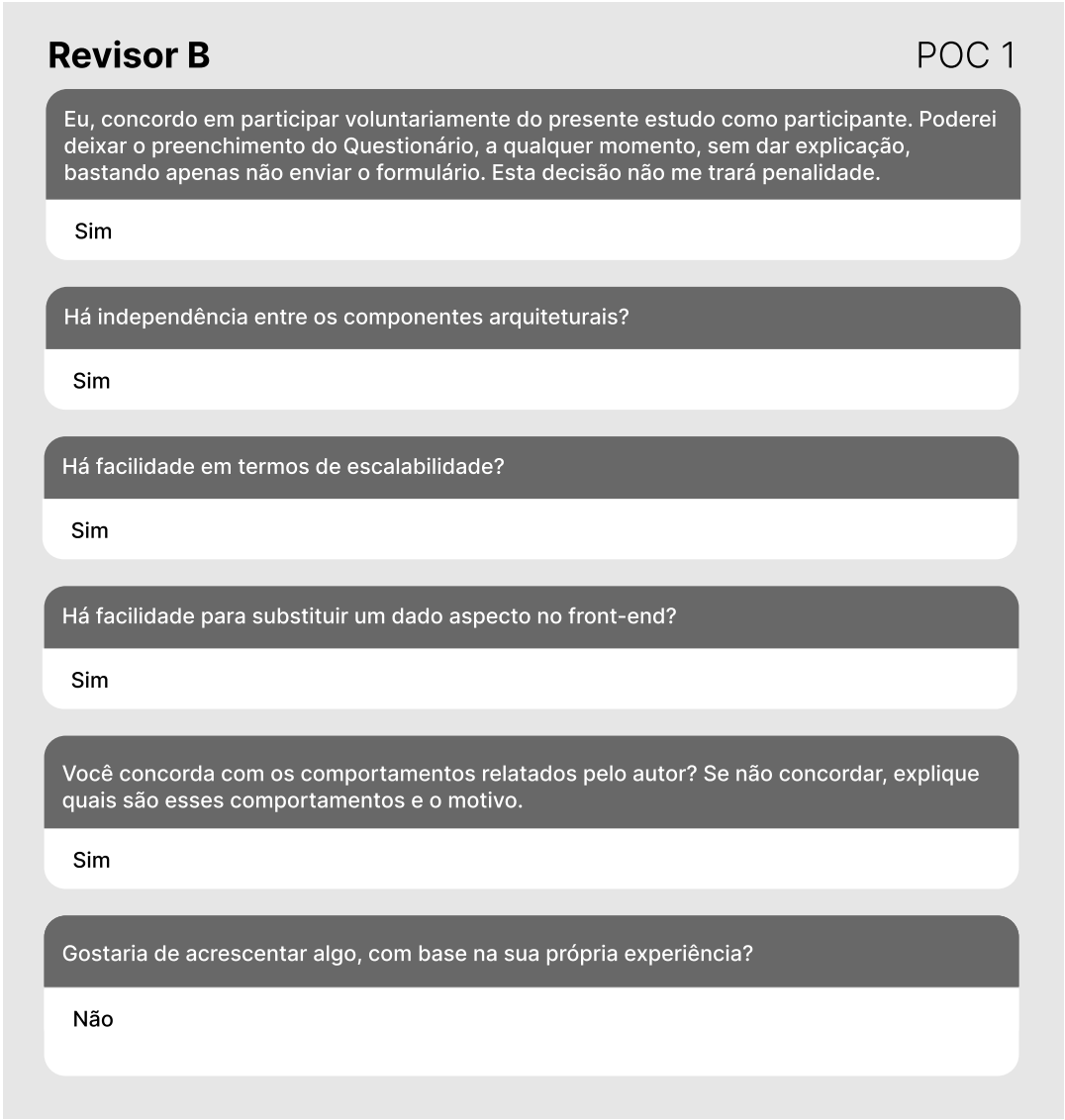
Esse formulário foi aplicado para cada solução das provas de conceito descritas no Estudo Exploratório.

Figura 44 – Revisão A sobre a solução elaborada na primeira prova de conceito



Fonte: Autor

Figura 45 – Revisão B sobre a solução elaborada na primeira prova de conceito

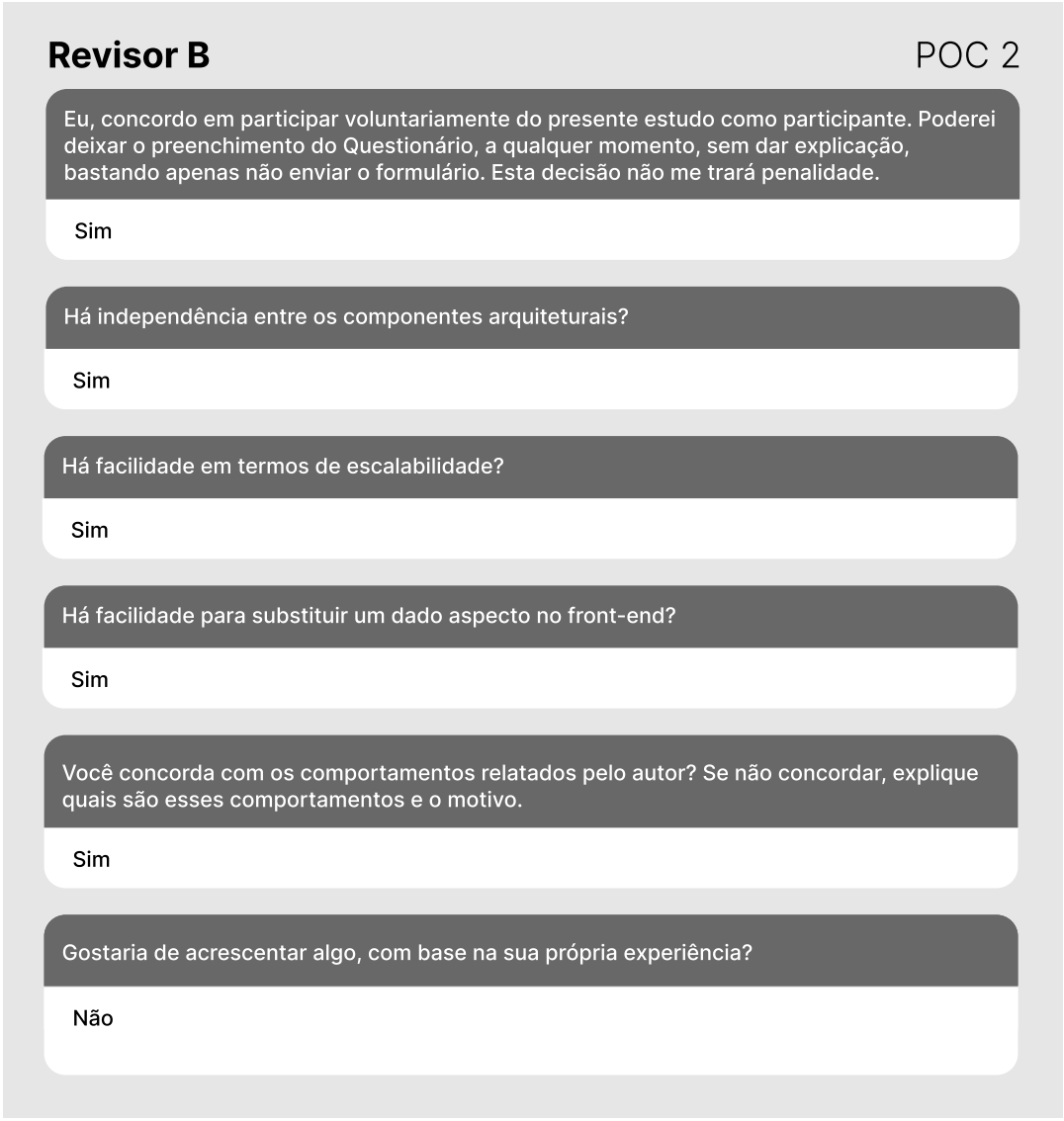


Fonte: Autor

Figura 46 – Revisão A sobre a solução elaborada na segunda prova de conceito

Fonte: Autor

Figura 47 – Revisão B sobre a solução elaborada na segunda prova de conceito

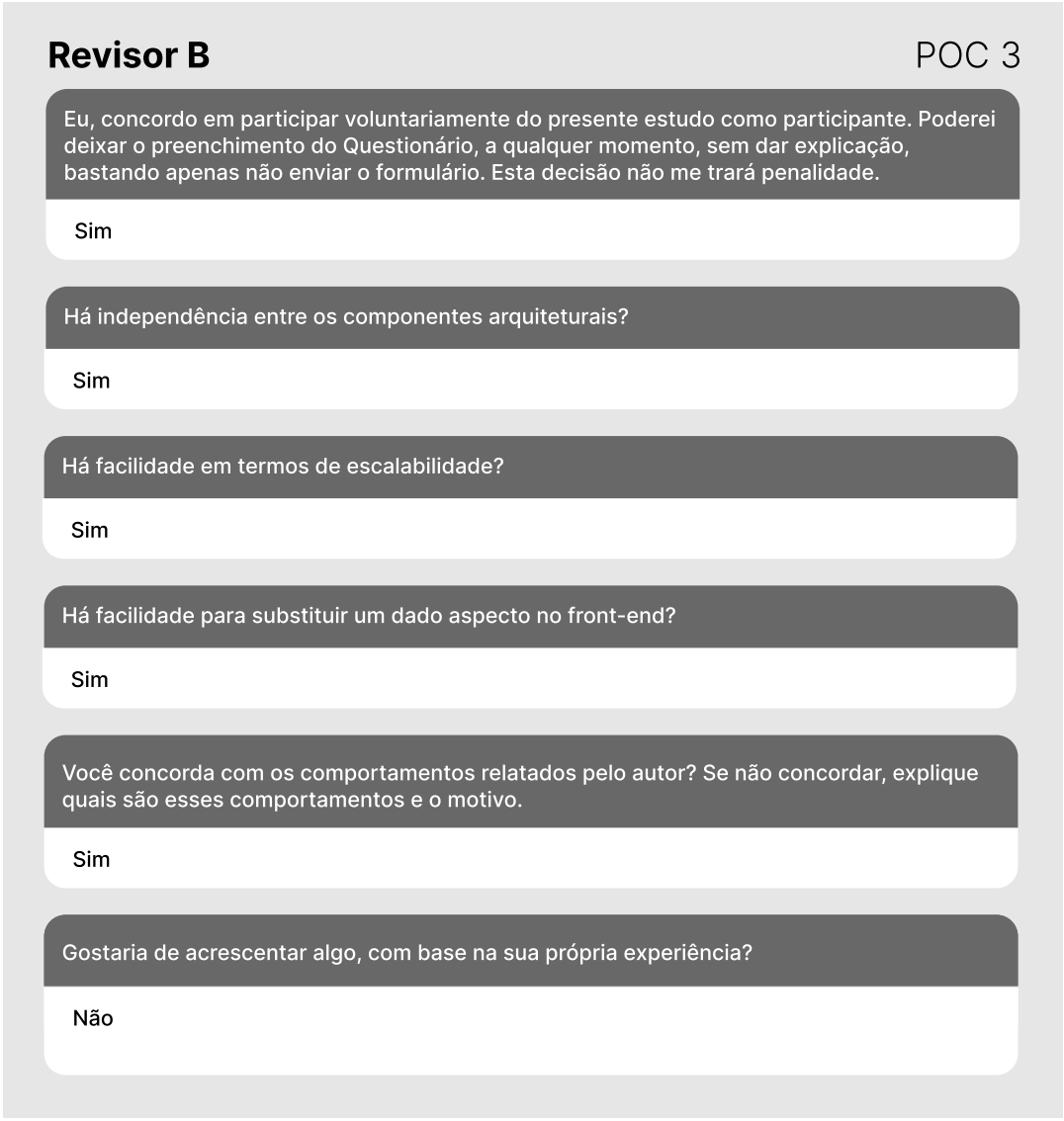


Fonte: Autor

Figura 48 – Revisão A sobre a solução elaborada na terceira prova de conceito

Fonte: Autor

Figura 49 – Revisão B sobre a solução elaborada na terceira prova de conceito

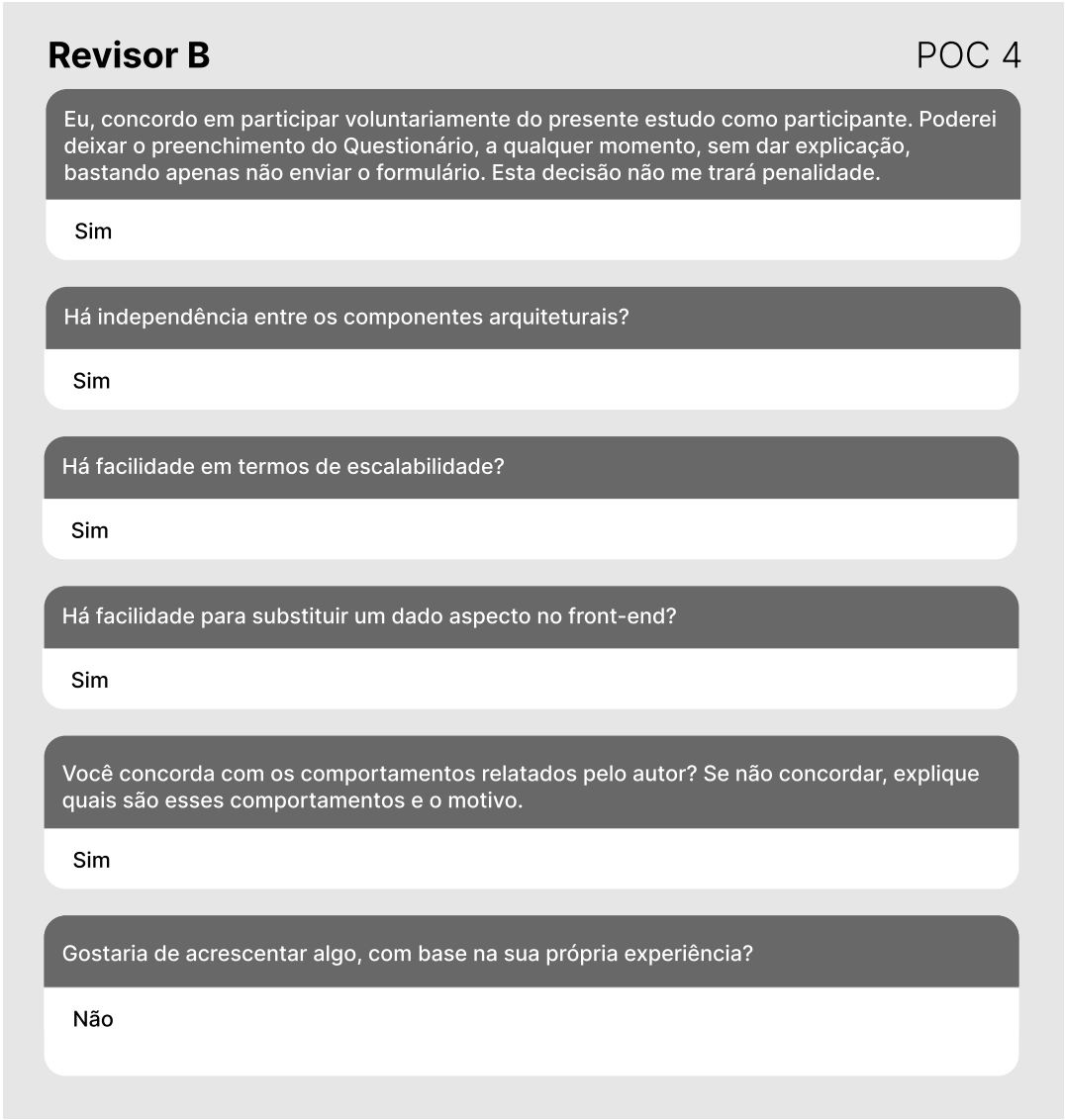


Fonte: Autor

Figura 50 – Revisão A sobre a solução elaborada na quarta prova de conceito

Fonte: Autor

Figura 51 – Revisão B sobre a solução elaborada na quarta prova de conceito

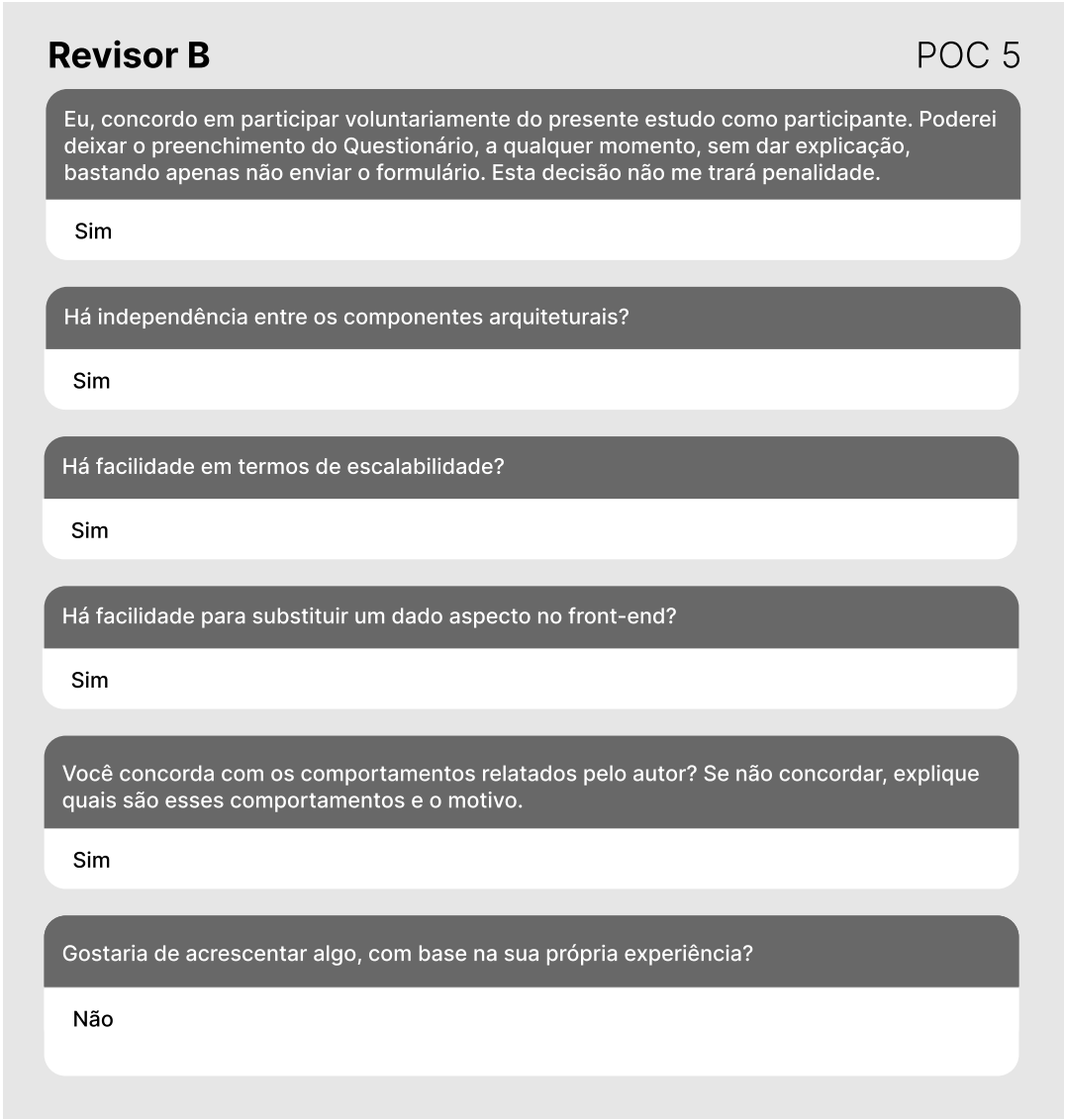


Fonte: Autor

Figura 52 – Revisão A sobre a solução elaborada na quinta prova de conceito

Fonte: Autor

Figura 53 – Revisão B sobre a solução elaborada na quinta prova de conceito



Fonte: Autor