CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

Faculdade de Tecnologia Baixada Santista Rubens Lara

Curso Superior de Tecnologia em Sistemas para Internet

THALLES GALVÃO DA SILVA

FIRESHOES: Estudo de caso de desenvolvimento de *front-end* utilizando estratégias de regeneração estática incremental e renderização pelo lado do servidor

Santos, SP
2022

THALLES GALVÃO DA SILVA

FIRESHOES: Estudo de caso de desenvolvimento de *front-end* utilizando estratégias de regeneração estática incremental e renderização pelo lado do servidor

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia Rubens Lara, como exigência para a obtenção do Título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Ricardo Pupo Larguesa

Santos, SP
2022

RESUMO

A necessidade crescente de se ter performance aliada aos benefícios do desacoplamento de *front-end* e *back-end* em *sites* e aplicações *web* modernas provocou uma revolução na forma com que se desenvolve e consume conteúdo. A última década teve um cenário fértil para o surgimento e popularização das Aplicações de Página Única — *Single Page Applications (SPA's)* — modelo este que tem como principal mote a construção e renderização de *sites* inteiros pelo navegador (cliente). Contudo, apesar de trazerem uma série de bônus, esse tipo de aplicação é também pernicioso em outras frentes. A partir daí surge a necessidade de buscar outras estratégias para solucionar esses problemas, tais como a Geração estática de *sites*, Regeneração estática incremental e Renderização pelo lado do servidor. Este estudo de caso se propõe a aplicar esses conceitos dentro de uma aplicação factível e que atende os critérios estabelecidos ao longo do projeto.

Palavras-chaves: Front-end. Aplicações de página única. Geração estática de *sites*. Regeneração estática incremental. Renderização pelo lado do servidor

ABSTRACT

The growing need for performance allied to the benefits of decoupling front-end and back-end in modern websites and web applications has caused a revolution in the way we develop and consume content. The last decade had a fertile scenario for the emergence and popularization of Single Page Applications (SPA's) - a model whose main motto is to build and render entire sites through the browser (client). However, despite bringing a series of bonuses, this type of application is also pernicious on other fronts. From there arises the need to look for other strategies to solve these problems, such as Static Site Generation, Static Incremental Regeneration, and Server-side Rendering. This case study proposes to apply these concepts within a feasible application that meets the criteria established throughout the project.

Keywords: Front-end. Single-page applications. Static generation of sites. Static incremental regeneration. Server-side rendering

LISTA DE ABREVIATURAS E SIGLAS

MPA *Multi-Page Application* (Aplicação de Páginas Múltiplas)

SPA *Single-Page Application* (Aplicação de Página Única)

CSR *Client-side Rendering* (Renderização pelo Lado do Cliente)

SSR *Server-side Rendering* (Renderização pelo Lado do Servidor)

SSG *Static Site Generation* (Geração Estática)

ISR *Incremental Static Regeneration* (Regeneração Estática Incremental)

HMTL *HyperText Markup Language* (Linguagem de Marcação de Hipertexto)

CSS *Cascading Style Sheets* (Folha de Estilos em Cascata)

JSON *JavaScript Object Notation* (Notação de Objeto JavaScript)

SEO *Search Engine Optimization* (Otimização para Motores de Busca)

URL *Uniform Resource Locator* (Localizador Uniforme de Recursos)

AJAX *Asynchronous JavaScript and XML* (Javascript Assíncrono e XML)

HTTP *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)

CDN *Content Delivery Networks* (Redes de Fornecimento de Conteúdo)

UX *User Experience* (Experiência do Usuário)

UML *Unified Modeling Language* (Linguagem de Modelagem Unificada)

SQL *Structured Query Language* (Linguagem de Consulta Estruturada)

NoSQL *Not Only SQL* (Não Somente SQL)

BFF *Back-end For Front-end* (Back-end Para Front-end)

RF Requisitos Funcionais

RNF Requisitos Não-Funcionais

CMS *Content Management System* (Sistema de Gerenciamento de Conteúdo)

API *Application Programming Interface* (Interface de Programação de Aplicação)

RESTful Representational State Transfer (Interface de Programação de Aplicação com Transferência Representacional de Estado)

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O comércio eletrônico realizado na internet já é uma realidade global. O consumo de bens e serviços praticados no ambiente digital já é um hábito comum às pessoas do mundo moderno. O que poucos anos atrás era impensável, ter acesso rápido a um item que há pouco estava a quilômetros de distância em um curto espaço de tempo, hoje já uma atividade quase trivial.

Para se ter uma ideia, conforme dados da Pesquisa Anual de Comércio do Instituto Brasileiro de Geografia e Estatística (IBGE) e em levantamento da Receita Federal, a Confederação Nacional do Comércio de Bens, Serviços e Turismo (CNC) estima que o faturamento do comércio eletrônico brasileiro somou R$ 224,7 bilhões no ano de 2020 (ALVARENGA, 2021).

Esse cenário se instalou por múltiplos fatores, especialmente em razão da revolução tecnológica trazida pelo advento da internet e meios digitais congruentes como o smartphone, a popularização da banda larga e o avanço tecnológico de plataformas. Isso inclui, naturalmente, a forma como *websites* são confeccionados e distribuídos.

E é justamente neste contexto que este projeto está amparado: na evolução e transformações do jeito de se construir *sites*. Isto vai desde a utilização de arquivos *HTML* (Linguagem de Marcação de Hipertexto) estáticos até a utilização da linguagem *JavaScript* e suas bibliotecas e *frameworks* para atingir performance e outros requisitos inerentes a projetos. Este trabalho dialoga, sobretudo, a respeito das temáticas e discussões trazidas pelas *SPA's* (*Single-page Application*) e seus *trade-offs* a depender do contexto de cada aplicação — neste caso, um *site* de comércio eletrônico (*e-commerce*).

No formato clássico da *web MPA (*M*ultiple-page Application*), onde conteúdos são servidos em arquivos estáticos, *sites*, por exemplo, têm as requisições de cada página distribuídas individualmente pelo servidor. Ou seja, cada acesso à uma seção do *site* gera um recarregamento do navegador. Nesse cenário, naturalmente, não há qualquer divisão de camadas de *front-end* e *back-end*. A aplicação é modelada em um sistema que abarca diversos arquivos *HTML* um correspondente a cada página, juntamente com pedaços de código responsáveis pelas regras de negócio do sistema.

Em *MPA*, muitas vezes, a maior parte da lógica da aplicação é executada no lado do servidor, e não no lado do cliente (como é mais comum em SPA). Tecnologias do lado do servidor, como linguagens de esfera *back-end*, são usadas para servir o *HTML* para o cliente. "Quando um usuário carrega a página inicial, uma solicitação é feita ao servidor para buscar uma página HTML estática que será renderizada pelo navegador. Qualquer ação que seja feita no navegador, digamos, um clique de botão, fará uma solicitação ao servidor para recuperar uma página HTML estática totalmente nova" (KHALIFA, 2020)

Como forma antagônica a este conceito, existem as *SPA's*. Elas surgem a partir da necessidade de atender as demandas de aplicações mais performáticas e ao mesmo tempo separar formalmente o *front-end* do *back-end*.

> Um aplicativo de página única (em inglês "single-page application", ou *SPA*) é uma aplicação *web* ou *site* que consiste em uma única página web com o objetivo de fornecer uma experiência do usuário similar à de um aplicativo desktop. Em um *SPA*, todo o código necessário (*HTML*, *JavaScript*, e *CSS*) ou é obtido com um único carregamento de página, ou os recursos apropriados são carregados dinamicamente e adicionados à página conforme necessário, geralmente em resposta a ações do usuário. A página não é recarregada em qualquer momento do processo, tampouco ocorre a transferência de controle para outra página, embora a *URL* no navegador ou a *API* de história do *HTML5* possam ser usadas para fornecer a percepção e navegabilidade de páginas separadas ao aplicativo (CAVALCANTE, 2018).

Nesse modelo, quando se entra em um *site*, toda a aplicação carrega de uma única vez e sua renderização passa a ser responsabilidade do cliente (navegador) — não do servidor como no modelo estático. Dessa forma, é possível apresentar uma a resposta imediata e o *feedback* visual instantâneo da interface a cada interação e navegação realizada pelo usuário. Isso, sem dúvida, traz uma série de benefícios: o principal deles talvez seja a experiência do usuário (CAVALCANTE, 2018). Termo esse tanto propagado como fundamental dentro da consolidação de aplicações digitais.

É possível que essa boa experiência se estenda também aos próprios desenvolvedores, já que uma das principais características das *SPA's* é a componentização, ou seja, quebrar pedaços de *layout* ou lógicas em fragmentos reutilizáveis.

Se tratando de um *e-commerce*, a importância de uma boa experiência do usuário se torna ainda mais exigida. Segundo o estudo "Comunicação e acompanhamento fazem a diferença na experiência de compra do cliente", realizado

pela *IDC* — empresa global de pesquisa e análise de dados de mercado — e encomendado pela Infobip, "65% dos entrevistados indicaram que fizeram recompras online devido a uma boa experiência, como navegação rápida, facilidade de uso, rastreamento de pedidos e disponibilidade de produtos" (ECBR, 2021).

Conforme dados apresentados no artigo "Nova pesquisa aponta como a tecnologia pode ajudar os *e-commerces* brasileiros a melhorar a experiência do usuário" publicado pela plataforma do *Google*, *Think with Google*, durante a *Black Friday* de 2020 constatou-se que o tempo de carregamento de páginas foi um fator decisivo para a performance de 25 grandes lojas virtuais brasileiras. "Reduzir o tempo de carregamento em apenas 0,1 segundo traz ganhos de desempenho: as taxas de conversão aumentaram 8,4% para o varejo, por exemplo. Cinco *players* performaram acima de 4,8 segundos, aumentando a taxa de rejeição em 90%" (KINOSHITA, 2021).

Esses fatores vantajosos motivaram a minha escolha pela utilização de bibliotecas e *frameworks JavaScript* que permitem o desenvolvimento de uma *SPA*. Contudo, essa opção traz ônus e bônus. Talvez o mais relevante dentro de um contexto de *e-commerce* seja os prejuízos trazidos para a indexação de buscadores e o *SEO* (otimização para motores de busca) que as aplicações *SPA* trazem.

Quando os *crawlers (*robôs usados pelos buscadores para encontrar, indexar e ranquear páginas de um *site*) inspecionam uma *SPA*, e se deparam com código *JavaScript* ao invés de um conteúdo com marcação (*HTML*), eles não conseguem identificar adequadamente o conteúdo do *site*, tampouco atender os critérios necessários para hierarquizar uma página no *Google*, por exemplo.

> *SPA's* são leves, rápidas de baixar e tipicamente fornecem uma boa experiência de usuário se programados corretamente. Mas como toda coisa boa, tem coisa não tão boa assim também. Se sua aplicação se importa com *SEO* (lembrando que nem todo app vai se importar com isso, como painéis de administrador ou qualquer parte da sua aplicação atrás de um login), *SPA's* podem ser detrimentais, uma vez que os *crawlers* de *SEO* vão enxergar uma página praticamente vazia e sem conteúdo. Outro potencial problema é que se criado de forma ingênua, um SPA pode forçar o cliente a baixar a aplicação inteira, mesmo partes dela que ele nunca vai usar ou acessar, visto que não existem *URLs* ou documentos explícitos. Ambos esses problemas podem ser resolvidos, mas nem sempre é trivial trazer essas otimizações dependendo como sua aplicação foi programada. (PEDRONI, 2021).

Isso ocorre porque a página ainda não foi montada pelo navegador durante a inspeção do *crawler*. Como se sabe, muitas vezes, essas aplicações são hidratadas com *APIs* (Interface de Programação de Aplicação) e *CMSs* (Sistema de Gestão de Conteúdos) e não com os dados diretos (CAVANCANTE, 2018). Por essa razão, algumas informações são trazidas às páginas de forma dinâmica e assíncrona através de *scripts* de linguagem de programação ainda não estão disponíveis no momento da varredura do *crawler*. Estes não conseguem aguardar a resolução de uma tarefa assíncrona para prosseguir com a leitura de informações.

Para atacar esse problema, o projeto vai usar estratégias de *SSG/ISR/SSR* (*Static Site Generation*, I*ncremental Static Regeneration*, *Server Side Rendering*). O *framework Next.js* permite a utilização dessa tecnologia em um modelo híbrido, onde parte da aplicação é previamente renderizada e construída pelo servidor (PEDRONI, 2021). Ou seja, os métodos mencionados conseguem se aproveitar das vantagens de uma *SPA* e mesclá-las com características do modelo estático tradicional, pré-renderizando a página no servidor, antes de servi-la.

Portanto, a intenção do trabalho é desenvolver o *front-end* de um *site* de *e-commerce* performático, que seja amigável aos buscadores e responda aos critérios adereçados anteriormente.

1.1 OBJETIVO

Desenvolver um projeto que atenda prioritariamente à demanda de excelência na experiência do usuário durante o uso da aplicação ao mesmo tempo que se enquadre nos requisitos para indexação em buscadores virtuais. E ainda traga a vivência e enriquecimentos profissional e pessoal no fluxo de desenvolvimento de uma aplicação de porte, desde a prática na escrita de um código adequado e funcional até a aplicação de tecnologias modernas e amplamente utilizadas por grandes empresas do Brasil e mundo afora.

1.1.1 OBJETIVO GERAL

Este trabalho tem o escopo de desenvolver o *front-end* de uma aplicação de *e-comerce* utilizando *frameworks* e bibliotecas *JavaScript* para utilizar uma *SPA* e atender as demandas de boa experiência do usuário, utilizando tecnologias

modernas de desenvolvimento de aplicações. Aliado a isso, aproveitar-se das estratégias de *SSG/SSR* para atacar o problema de *SEO* trazido por *SPA's*.

Em suma: apresentar o desenvolvimento do *front-end* em uma aplicação performática e indexável exemplificado em um caso de uso real onde essas duas demandas são necessárias, um *e-comerce*.

1.1.2 OBJETIVOS ESPECÍFICOS

a) Idealizar a interface e produzir o *front-end* de um *e-commerce* utilizando bibliotecas e *frameworks JavaScript* capazes de produzir uma *SPA* (*React.js* e *Next.js*).

b) Desenvolver funcionalidades básicas de um *e-commerce*, como exibição de produtos, filtros, busca, carrinhos de compra e outros.

c) Consumir, atualizar, criar e deletar dados trazidos pela plataforma *Firebase*.

d) Lidar com autenticação de usuários utilizando a plataforma *Firebase*.

e) Utilizar funcionalidades de Geração estática de *sites* (SSG), Regeneração Estática Incremental (ISR) e Renderização pelo Lado do Servidor (SSR) trazidas pelo *framework Next.js*.

1.2 ESTADO DA ARTE

Praticamente todos os grandes *e-commerces* do Brasil e do Mundo utilizam bibliotecas e *frameworks JavaScript* para desenvolver seu *front-end*. *React.js* é utilizado por *Amazon*, *Wallmart*, *Americanas*, *Magazine Luiza*, entre outros. O meta-*framework* de *React.js*, o *Next.js* — que oferece tecnologias de *SSG/SSR* — é utilizado por algumas dessas como *Wallmart* e *Magazine Luiza*.

Não à toa grandes *players* dão importância para a performance de seus *e-commerces* e consequentemente a boa experiência do usuário que essas ferramentas trazem.”. Maile Ohye, do *Google*, em 2010 afirmou que 2 segundos é o tempo aceitável para um *site* de *e-commerce*. No *Google*, espera-se que todos fiquem abaixo de meio segundo (ANDERSON, 2020).

2 DESENVOLVIMENTO

A fim de inserir em um contexto de uso real as temáticas abordadas por esse projeto: performance aliada à experiência do usuário e indexação em buscadores, optou-se pelo desenvolvimento de uma ‘aplicação teste’ de um *e-commerce*. Já que esta temática ataca perfeitamente todas as proposições feitas ao longo deste trabalho.

Para tal, optou-se por uma loja virtual fictícia de calçados denominada “*Fireshoes*”. A escolha da temática se deu em razão deste ser o segmento preferido pelos consumidores brasileiros, de acordo com a pesquisa “Comunicação e acompanhamento fazem a diferença na experiência de compra do cliente” realizada pelo *IDC* em 2020. “A pesquisa aponta que os produtos mais adquiridos são: Moda/Vestuário/Calçados (74,7%), Alimentos/Supermercados (57,6%), Eletrônicos/Eletrodomésticos (57,2%), Cosméticos/Perfumaria (47,1%), Farmácia/Remédios (42,5%), Móveis/Decoração (37%), e Material de construção (13,4%)”.

No decorrer deste capítulo serão abordados aspectos diversos a respeito do desenvolvimento do projeto. Em ‘Análise do Sistema’, são especificadas as escolhas das tecnologias e modelagens. ‘Análise de Requisitos’ traz a relação dos requisitos funcionais e não funcionais definidos. A seção é seguida por ‘Diagrama de Casos de Uso’ e ‘Fluxo de Eventos’ onde são descritos detalhadamente os cenários de uso da aplicação, bem como os caminhos interativos da aplicação. Em ‘Banco de Dados’ são tratadas as justificativas que motivaram a escolha de um banco de dados não relacional. E por fim, na ‘Camada de Apresentação’ há o esboço das telas.

2.1 ANÁLISE DO SISTEMA

Levando em consideração a premissa fundamental deste projeto — o desenvolvimento de um *site* de compras utilizando *SPA’s* — a temática escolhida para a demonstração e aplicabilidade do assunto foi um e-commerce de sapatos.

A preferência pela biblioteca *JavaScript React* juntamente com o *framework Next.js* se justificou pela grande popularidade e massiva utilização dessas tecnologias no contexto de aplicações *web* performáticas. Já o emprego do *Firebase* no sistema foi fundamentado na sua versatilidade como facilitador para introdução

de funcionalidades comumente desenvolvidas na camada de *back-end*, tais como autenticação, e manipulação de banco de dados.

O planejamento para o desenvolvimento da aplicação se deu pela popular *United Modeling Language* (UML) por conta do seu uso consolidado quando o assunto é engenharia de *software*. “A *UML* é como uma linguagem universal para profissionais de produção de *software*, ela ajuda muito a comunicação clara e objetiva entre pessoas envolvidas no processo de produção” (VENTURA, 2019).

2.1.1 ANÁLISE DE REQUISITOS

Com o tema e o modelo do trabalho definido, foram elicitados os requisitos funcionais e não funcionais da aplicação, à luz do que sugere a modelagem *UML* e outras formas modernas de gerência de projetos. Através da análise de requisitos, é possível compreender, planejar e implementar funcionalidades em sistemas. Desta forma, há uma maneira prática de se verificar e garantir que a interação e experiência do usuário sejam alcançadas com sucesso.

Abaixo, a listagem de requisitos funcionais (RF) e requisitos não-funcionais (RNF) do projeto:

[RF01] O sistema deve permitir o cadastro e manutenção de contas de acesso.
[RF02] O sistema deve permitir que o usuário se autentique.
[RF03] O sistema deve permitir que o usuário recupere sua conta.
[RF04] O sistema deve permitir que o usuário delete sua conta.
[RF05] O sistema deve permitir que o usuário faça logoff.
[RF06] O sistema deve permitir que o usuário adicione produtos ao carrinho.
[RF07] O sistema deve permitir que o usuário manipule produtos do carrinho.
[RF08] O sistema deve exibir os produtos em prateleiras na página inicial.
[RF09] O sistema deve permitir que o usuário realize uma compra.
[RF10] O sistema deve permitir que o usuário veja detalhes da sua compra.
[RF11] O sistema deve permitir que o usuário cadastre e manipule endereços de entrega.
[RF12] O sistema deve permitir que o usuário cadastre e manipule meios de pagamento.
[RF13] O sistema deve permitir que o usuário edite dados cadastrais.

[RF14] O sistema deve permitir que o usuário visualize o seu histórico de compras.
[RF15] O sistema deve permitir que o usuário cancele uma compra.
[RF16] O sistema deve permitir que o usuário visualize informações e detalhes do produto individualmente.
[RF17] O sistema deve permitir que o usuário pesquise por produtos na barra de busca.
[RF18] O sistema deve permitir a filtragem de produtos por parâmetros.
[RF19] O sistema que o administrador cadastre e manipule categorias de produtos.
[RF20] O sistema que o administrador cadastre e manipule produtos.
[RF21] O sistema deve permitir que o usuário seja notificado ao manipular o carrinho.
[RF22] O sistema deve permitir que o usuário seja notificado ao manipular endereços de entrega.
[RF23] O sistema deve permitir que o usuário seja notificado ao manipular meios de pagamento.
[RF24] O sistema deve permitir que o usuário seja notificado ao editar dados cadastrais.
[RF25] O sistema deve permitir que sejam acessadas categorias de produtos.

[RNF01] O front-end da aplicação deve construído com uma SPA.
[RNF02] O sistema deve atender aos critérios de ranqueamento em buscadores.
[RNF03] A interface deve ser agradável e de fácil navegação.
[RNF04] O front-end deve atender aos critérios de acessibilidade descritos no Diretrizes de Acessibilidade para o Conteúdo da Web (WCAG)
[RNF05] Sempre que possível, a aplicação deve realizar funcionalidades sem a necessidade de carregamento da página.
[RNF06] O sistema deve permitir que as informações da loja sejam distribuídas e editadas via API REST através do Firebase.

2.1.2 DIAGRAMA DE CASO DE USO

Uma vez elicitados os requisitos do sistema, é possível utilizar-se da estratégia do Diagrama de Casos de Uso, conforme preconiza o modelo UML. Conforme definição de Ventura (2016), “o Diagrama de Caso de Uso serve para

representar como os casos de uso interagem entre si no sistema e com os usuários (atores), ou seja, como as funcionalidades se relacionarão umas com as outras e como serão utilizadas pelo usuário, durante o uso do sistema".

A principal vantagem da utilização do Diagrama de Caso de Uso é a sua eficiência em comunicar funcionalidades e fluxos de uma aplicação. Esta foi uma grande valia durante o planejamento deste trabalho, uma vez que essa ferramenta faz com que as reflexões acerca das questões que envolvem a aplicação sejam mais céleres, pragmáticas e, especialmente, mais eficientes.

"Em projetos de *software*, um dos maiores desafios é a boa comunicação. E neste contexto, fica claro que o uso racional de diagramas *UML* com o objetivo de transmitir ideias entre membros de um mesmo time, é uma ferramenta que favorece muito uma cultura ágil" (VENTURA, 2016). O Diagrama de Caso de Uso deste projeto pode ser visualizado no Apêndice A.

2.1.3 FLUXO DE EVENTOS

Esmiuçar e tratar com riqueza de detalhes todo o fluxo de ações e interações entre os atores também é um ponto fundamental no planejamento e documentação do trabalho. Através dele, é possível detalhar e pré-conceber todos os eventuais cenários de uso da aplicação bem como os seus fluxos alternativos. Desta maneira, facilita-se desde o desenvolvimento do código, até a idealização da jornada do usuário, conceito esse tão enaltecido pelas boas práticas de Experiência do Usuário (*UX*).

A descrição dos fluxos de eventos desse sistema pode ser consultada no Apêndice B. Complementando os fluxos, existe ainda um mapa mental contido no Apêndice D.

2.2 BANCO DE DADOS

Este trabalho tem como objetivo primário o desenvolvimento de um *front-end* de uma aplicação de *e-commerce,* como mencionado anteriormente. Por esse motivo, para manter um ponto focal definido na premissa do projeto, optou-se pelas tratativas relacionadas ao *back-end e* banco de dados serem gerenciadas através de um serviço terceiro com funcionalidades de *BFF* (*Back-end for front-end*). É o caso

da popular plataforma do *Google*, conhecida como *Firebase*, que foi escolhida para compor o arsenal de tecnologias do projeto. Ele vai lidar com duas partes importantes da aplicação: autenticação e banco de dados.

A motivação da escolha do *Firebase* foi a sua facilidade, popularidade, acessibilidade em razão de planos iniciais gratuitos e, especialmente, pelo seu potencial de tratar de forma abstraída e simples questões complexas ligadas à outras camadas que não o *front-end*.

As opções de banco de dados oferecidos pela plataforma são caracterizadas como não relacionais, isto é, “um banco de dados que não usa o esquema de tabela de linhas e colunas encontrado na maioria dos sistemas de banco de dados tradicionais” (TEJADA, 2021). Quando o assunto é banco de dados não relacional, existe uma imensa gama de tecnologias e características particulares dessa temática. Nesse contexto, tanto os serviços de banco de dados do *Firebase: Realtime Database* e *Cloud Firestore* apresentam algumas das várias características que podem ser identificadas como naturais de um banco de dados não relacional. A mais perceptível talvez seja a condução de operações em dados *JSON (JavaScript Object Notation* — formato de troca de dados amplamente utilizado*)*.

Entre as principais vantagens da utilização de um banco relacional em detrimento de sistemas tradicionais está a sua flexibilidade. Isso se dá por conta da sua ausência de amarras e esquemas. Para ser inserido, um dado não deve atender a um padrão pré-definido de tabelas e colunas como ocorre em um recurso de *SQL* (Linguagem de Consulta Estruturada). Consultas expressivas, disponibilidade e fácil escalabilidade são outros pontos altos encontrados em bancos de dados não relacionais.

No entanto, a flexibilidade trazida por ele pode, em alguns casos, trazer situações problemáticas durante o desenvolvimento como a necessidade de implementação de tratativas para retorno de dados diferentes para documentos de uma mesma coleção. Por exemplo, um campo “idade” pode ser inserido com o tipo número em um registro, como texto em outro e até mesmo não existir em um terceiro. Essa falta de previsibilidade e consolidação de dados jamais ocorreria em um banco de dados relacional. Uma forma de remediar essa situação seria a utilização de *Typescript* durante o desenvolvimento.

Contudo, examinando os *trade-offs*, ainda assim, por conta de sua característica volátil e flexível optou-se por não conceber Modelo de Entidade

Relacionamento (MER), tampouco um esquema de tabelas e colunas. Estas dão lugar a coleções e documentos que podem ser inseridos e manipulados livremente.

Para ilustrar uma possível e parcial estruturação dos dados, optou-se pela exemplificação de interfaces em *Typescript.* Desta forma, fica demonstrado o tipo de dado inicialmente esperado por cada coleção e documento. É importante se salientar, todavia, que esse foi um exemplo meramente ilustrativo e com planejamento prévio à implementação. As interfaces idealizadas podem ser vistas no Apêndice E.

2.3 CAMADA DE APRESENTAÇÃO

Durante a concepção do projeto foi utilizado uma técnica costumeira do desenvolvimento de interfaces e de planejamentos de navegação e integração: a criação de um w*ireframe*. Ele nada mais é do que um esboço de telas simplificado. “Dessa forma, os *wireframes* conseguem demonstrar de forma direta a arquitetura final de uma interface, posicionando os elementos de forma simples e organizada. Portanto, o *wireframe* reflete apenas o necessário da proposta de uma interface digital” (EDITORIAL AELA, 2019).

Inicialmente as telas idealizadas foram: página inicial (*Home*): porta de entrada por onde o usuário tem o primeiro contato com o *site*, página de login: onde o usuário faz a autenticação e se credencia para finalizar compras, página de cadastro: lugar onde o usuário faz seu cadastro ou recupera uma conta, página de usuário: onde são realizadas manipulações cadastrais e de compras, página de *checkout*: lugar em que o usuário faz o fluxo para finalizar uma compra, página de categoria: onde são exibidas prateleiras baseadas em parâmetros de busca, filtros ou categorias e por fim, página de produto: onde há detalhes e informações sobre o produto.

Todos os *wireframes* podem ser consultados no Apêndice C.

3 RESULTADO

A fim de atingir os objetivos propostos no início do projeto aliando estratégias de Aplicação em página única (*SPA*), Geração estática de *Sites* (*SSG*), Regeneração estática Incremental (*ISR*) e Renderização pelo lado do servidor (*SSR*) foi necessário se debruçar sobre as particularidades de cada um dos modelos de modo a escolher qual se utilizar em cada pedaço da aplicação do estudo de caso: um *e-commerce.*

A principal temática que este projeto busca atacar são alguns cenários problemáticos trazidas pelas *SPA'*s. Entre eles se destacam: dificuldades de otimizações para motores de busca (*SEO*) e requisições desnecessariamente frequentes ao *back-end* através de Interface de Programação de Aplicação com Transferência Representacional de Estado (*API's RESTful*).

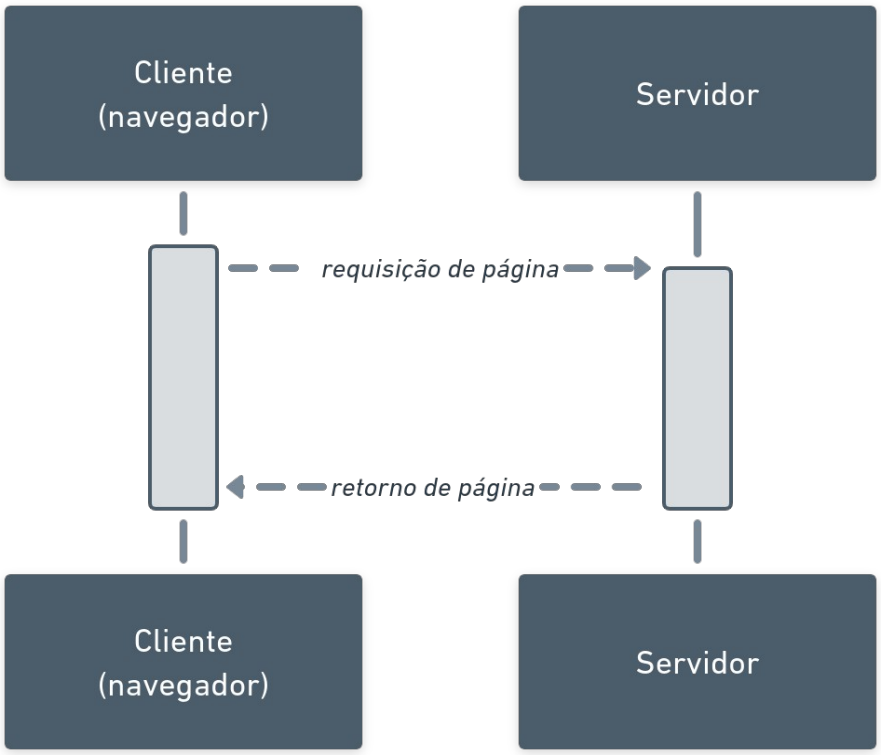
Antes de trazer contratempos, as *SPA'*s surgiram justamente para resolver outros tipos de problemas que ocorriam nas aplicações *web* tradicionais monolíticas e no formato de Aplicação de páginas múltiplas (MPA). E estas, por sua vez, solucionavam problemas da *WEB 1.0.*

No início da *web*, havia apenas documentos com *links* entre eles. Basicamente, quando um navegador fazia uma requisição a um servidor para obter uma página específica, esse sistema retornava o arquivo guardado em seu disco rígido de volta ao cliente. Dada a simplicidade das páginas e do processo em si, as interações na rede eram simplesmente compostas pela relação básica entre cliente e servidor.

Tratava-se de uma rotina relativamente veloz, considerando a largura de banda da época, onde arquivos *HTML* estáticos eram distribuídos sem qualquer informação dinâmica. Todos os usuários recebiam o exato mesmo arquivo, sem alterações.

> Era natural que o modelo predominante de serviço *web* na época fosse o tradicional cliente-servidor, onde um cliente (e.g. um *browser*) pede um recurso (e.g. uma página localizada num endereço específico) e o servidor, aprovando esse pedido, devolve esse recurso. E quando esse recurso é algo completamente estático como eram os *sites* na época, o que o servidor devolvia era exatamente o recurso pedido — i.e. a página do *site* em questão. Inclusive, nota-se que a maioria das páginas termina com a extensão *.html,* significando que o que você recebeu é realmente o próprio arquivo *HTML*). (PEDRONI, 2021).

Figura 01 - Fluxo de cliente-servidor na *Web* 1.0



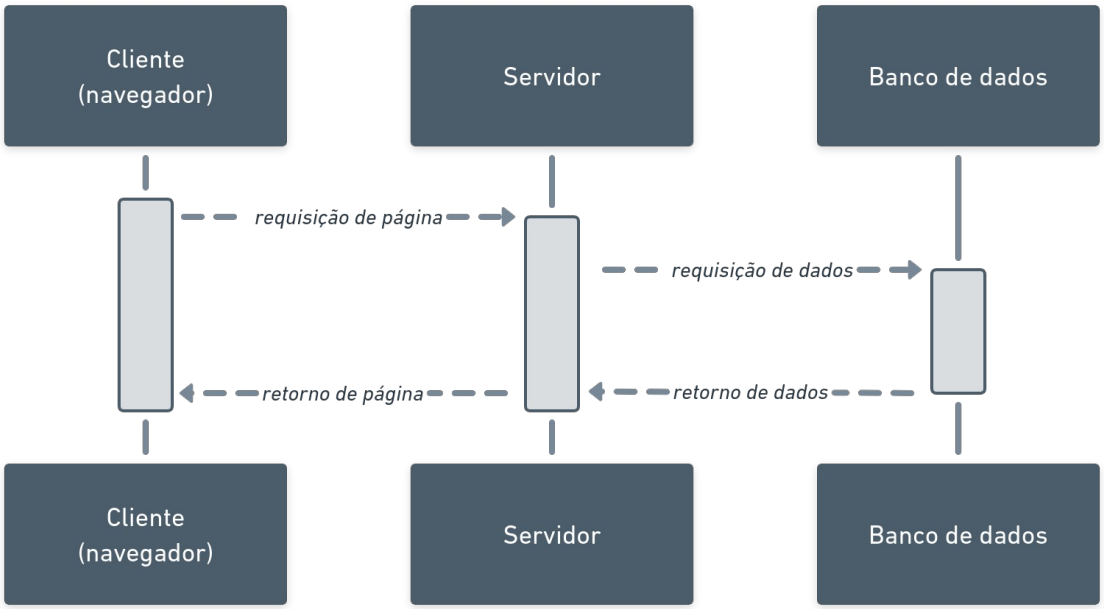
Fonte: AUTOR, 2022

Pouco tempo depois, houve a primeira grande transformação da *web* de forma a começar a explorar as muitas oportunidades que essa tecnologia oferece. Ao invés de servir arquivos estáticos, servidores começaram a pré-processar o *HTML* antes de ser enviado ao cliente. Ou seja, sempre que um navegador fizer uma requisição a um servidor enviando parâmetros como *cookies* ou cabeçalhos de autenticação, o servidor gera um arquivo *HTML* na mesma hora personalizado para aquele usuário específico.

Nesse momento, os servidores deixam de ser apenas distribuidores de arquivos de somente leitura, em um movimento que foi se retroalimentando e possibilitou o surgimento de grandes portais e posteriormente as redes sociais. Estes criaram a necessidade de respostas altamente mutáveis, customizadas, personalizadas e sobretudo dinâmicas. Comportamento incompatível com um sistema de arquivos estáticos e documentos prontos. Nesse momento surge a denominada *WEB 2.0*.

“Com essa nova web, não era mais possível ter todas as informações prontas de antemão. Por mais que ainda era responsabilidade dos servidores devolverem os documentos aos clientes, ao invés de devolver um arquivo pronto estático, os documentos começaram a ser renderizados do lado do servidor. Na prática, isso significa apenas que o documento devolvido em partes era estático, mas continha alguma informação que o servidor colocou ali, justamente na hora que foi feito o pedido por aquele documento” (PEDRONI, 2021).

Figura 02 - Fluxo de cliente-servidor na *Web 2.0* utilizando renderização pelo lado do servidor



Fonte: AUTOR, 2022

Dessa forma, o cliente continua recebendo um documento *HTML* pronto, como na WEB 1.0, porém com informações inseridas trazidas por um banco de dados e inseridas pelo *back-end.* Trata-se de uma renderização pelo lado do servidor *(SSR).* No entanto, ao mesmo tempo que oferece benefícios de uma customização de dados dinâmicos, traz prejuízos à performance, uma vez que naturalmente, há camadas extras no fluxo (consulta a banco de dados, execução de *scripts* por linguagens de programação, entre outros).

Outro ponto de atenção foi colocado a respeito da arquitetura das aplicações monolíticas — mais popular do início dos tempos de desenvolvimento *web* e dominante na *WEB 2.0.* Nela não existe separação formal entre *back-end* e *front-end.* A mesma aplicação é responsável por tratar as regras de negócios, lidar com

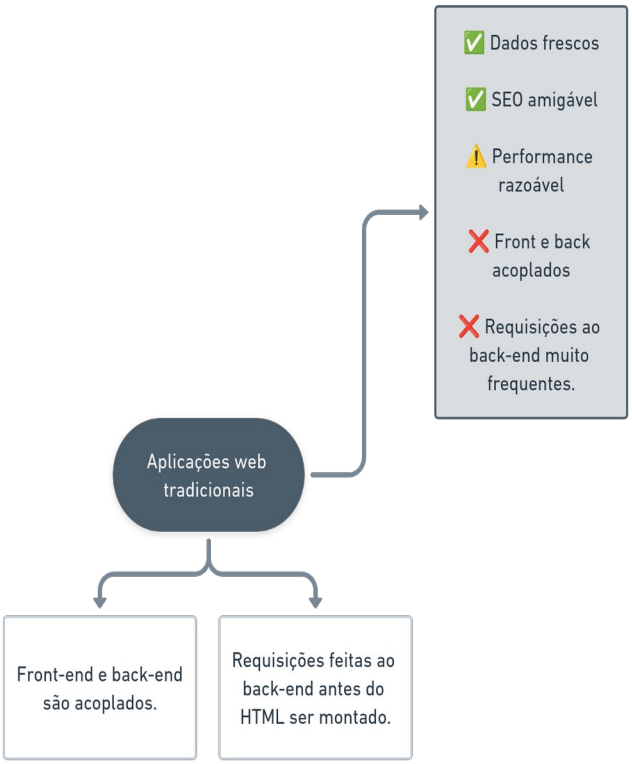
autenticações, comunicações do banco de dados, e construir as telas que serão acessadas pelos usuários. O alto acoplamento dessas camadas pode ser interpretado como problemático em alguns cenários.

Contudo, há uma série de vantagens neste tipo de modelo, especialmente quando o assunto é *SEO* já que a página *HTML* é gerada pelo próprio *back-end* mesmo antes de ser servida e com isso, pode ser vasculhada pelos *crawlers* (robôs usados pelos buscadores para encontrar, indexar e ranquear páginas de um *site*). Segundo Khalifa (2020), *MPA* geralmente é mais otimizado para *SEO*, pois cada página tem sua própria *URL* e pode ser recuperada por rastreadores da web separadamente.

Outro claro benefício é que os dados apresentados são sempre os mais recentes possíveis já que não há dependência de uma requisição feita a um serviço possuidor das informações por meio de uma *API*, por exemplo.

A Figura 03 exibe um diagrama das aplicações web tradicionais, suas principais características e uma tabela com vantagens (simbolizadas por um sinal de checado), pontos de atenção (sinalizados por um triângulo com exclamação) e desvantagens (sinalizados com um X).

Figura 03 - Diagrama de aplicações web tradicionais



Fonte: AUTOR, 2022

Nos idos dos anos 2000, um novo divisor de águas na forma como as aplicações web eram consumidas e construídas surge com a criação e popularização de uma nova *API* conhecida como *AJAX (JavaScript Assíncrono e XML)*. Essa nova tecnologia permitia que navegadores enviassem e recebessem dados para o servidor sem a necessidade de recarregar a página. Isto é, pedaços da interface poderiam ser alterados dinamicamente e instantaneamente.

De acordo com Pedroni (2021), essa mudança de paradigma possibilitou o cliente ter muitas responsabilidades, e por isso “cada vez mais criamos aplicações complexas do lado dele, indo ao ponto de criarmos aplicações inteiras que vivem apenas do lado do cliente”.

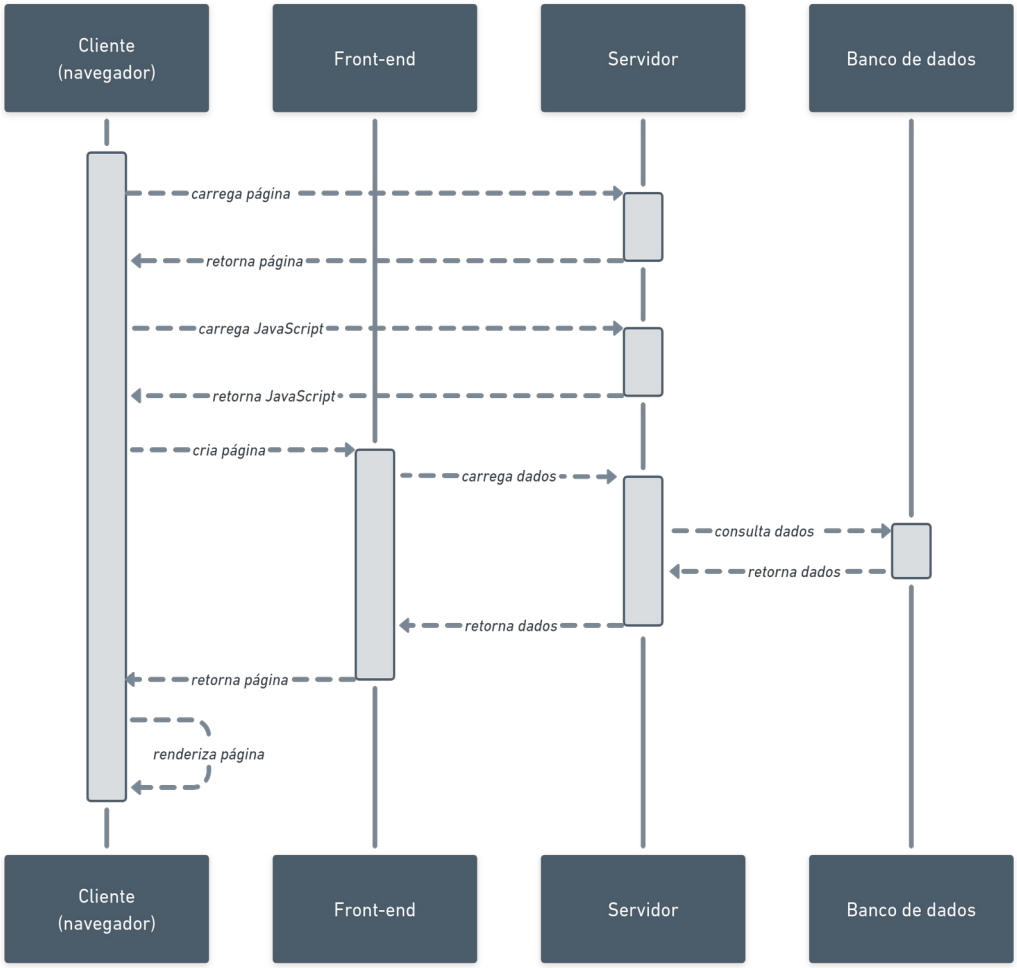
A união do *AJAX* seguida de aprimoramentos do *JavaScript,* linguagem interpretada pelos navegadores, foi o berço de uma nova forma de se construir aplicações que ganha enorme aderência até os dias de hoje: as *SPA’s.*

> As *SPA’s* começaram no início de 2010, o que foi considerado uma evolução do *MPA* + AJAX juntos. A *SPA* executa a maior parte da lógica da interface do usuário em um navegador da *web*. Ele se comunica com o servidor usando *APIs* da web para obter e enviar dados. Inicialmente, todos os recursos de interface do usuário (*HTML*, *JavaScript* e *CSS*) são obtidos por um único carregamento de página uma única vez. Qualquer atualização na página acontece dinamicamente em resposta às ações do usuário. A página não é recarregada (ou carrega outras páginas) em nenhum momento da vida útil do aplicativo”. (KHALIFA, 2020).

De forma prática, uma *SPA é* caracterizada por uma aplicação que carrega de uma única vez logo na primeira requisição do cliente ao servidor. Quando um usuário acessa uma *SPA,* é realizado o *download* de um único arquivo *HTML* que tem todo o seu conteúdo, comportamento e aparência manipulado dinamicamente e sob demanda através de *scripts* determinados pela linguagem de programação *JavaScript*.

Nas *SPA’s*, ao invés de ocorrer uma nova requisição ao servidor sempre que um usuário visita uma nova página, a aplicação carrega todo o *HTML*, *CSS* e *JavaScript* necessário para aplicação na requisição inicial. Isso significa que mudanças de rotas ocorrem inteiramente no lado do cliente.

Figura 04 - Fluxo clássico de uma Aplicação de Página Única (SPA)



Fonte: AUTOR, 2022

Com a chegada das *SPA's*, houve um desacoplamento entre *back-end* e *front-end*. Dessa maneira, o *back-end* passou a ter como ponto focal resolver questões sensíveis e de regra de negócios enquanto o *front-end* tem para si toda a responsabilidade de montar e exibir uma interface performática a fim de atender às demandas de experiência de usuários. Neste modelo, quando há necessidade de comunicação entre os dois segmentos, são realizadas requisições via *API's:* uma estratégia semelhante ao *AJAX* mencionado anteriormente.

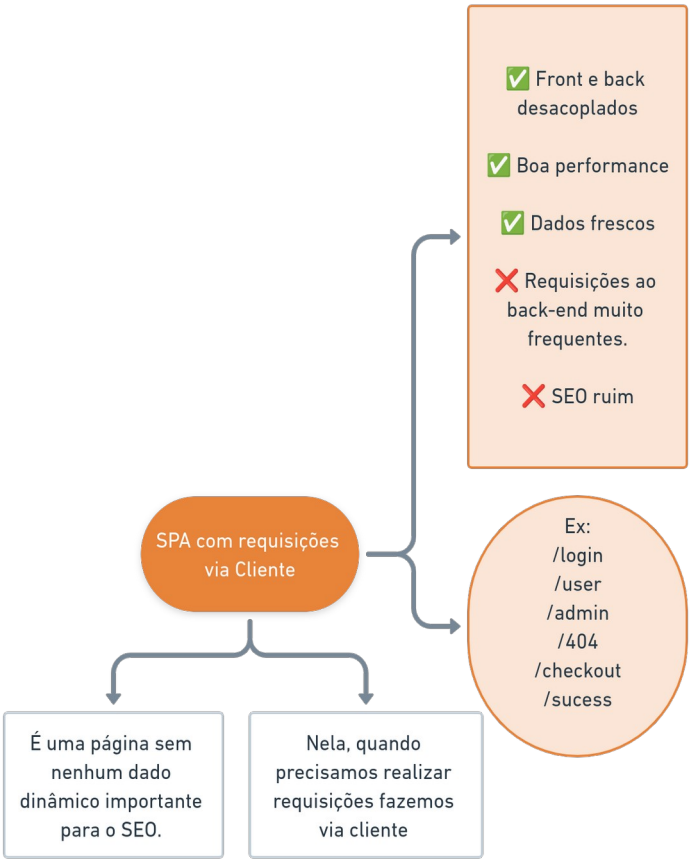
Agora, o *front-end* é desenvolvido a partir de bibliotecas e *frameworks JavaScript* totalmente a parte das tratativas de *back-end*. Inclusive, há a possibilidade de mais de um *front-end* se comunicar com um mesmo *back-end*. Como por exemplo, um aplicativo móvel desenvolvido em uma linguagem X e uma aplicação *web* desenvolvida em uma tecnologia Y consultarem um *back-end* Z

através de *API's*. Nessa hipótese, uma eventual indisponibilidade de um dos *front-ends* não impactaria de maneira nenhuma no outro.

Contudo, apesar dos diversos benefícios já elencados, talvez o principal problema que acompanha o modelo de *SPA* seja a dificuldade de se ter uma boa Otimização para Motores de Busca (*SEO*). Ou seja, a capacidade da aplicação ser ranqueada em buscadores, como o Google, fica prejudicada.

Ao contrário das aplicações web tradicionais, nas *SPA's* o *HTML* é construído em tempo de execução pelo próprio cliente. É justamente essa característica que inviabiliza a atuação de *crawlers,* já que estes são apenas capazes de identificar dados estáticos em linguagem de marcação. No momento de suas varreduras, essas informações ainda não estão disponíveis. A Figura 05 apresenta um diagrama sobre as *SPA's*, listando suas particularidades (quadros inferiores), benefícios e prejuízos (quadro superior) e em quais rotas do estudo de caso foram utilizadas (círculo).

Figura 05 - Diagrama de Aplicações de Página Única (SPA)



Fonte: AUTOR, 2022

Dentro de um *e-commerce* há páginas em que o ranqueamento e *SEO* são extremamente importantes, como por exemplo, páginas de produtos, *home etc.* Em contrapartida, há outras que são exclusivas para uso interno ou administrativo. Nessas, naturalmente, o *SEO* é totalmente irrisório. Dentro da aplicação que compõe o caso de uso construída nesse projeto não foi diferente.

Rotas de autenticação e cadastro de usuário (/*login*), gerenciamento de dados de usuário (/*user*), cadastro e manipulação de produtos (/*admin*), fluxo de finalização de compra (/*checkout* e /*success*) foram construídas dentro de um padrão de geração estática com requisições ao *back-end* sob demanda.

Por exemplo, na aplicação *Fireshoes*, usuários podem ver, adicionar, editar e remover endereços e meios de pagamento na área do usuário (/*user*). Nenhuma informação já existente armazenada no banco de dados deve ficar disponíveis para indexação de motores de busca. Portanto, é perfeitamente justificado que esse pedaço tenha indisponibilidade de *SEO,* limitando-se a um cumprimento puramente funcional baseado em requisições do *front-end* pelo lado do cliente ao *back-end* via métodos *HTTP* para visualizar e manipular dados pessoais de um usuário.

Todavia, conforme salientado anteriormente, há seções da aplicação em que o *SEO* é muito valioso. Para atacar essa problemática, foi utilizada uma estratégia híbrida aliando caraterísticas de *SPA* e *MPA,* aproveitando os pontos fortes de cada uma delas. Pois, tanto o modelo de *SPA* quanto o de *MPA* com *AJAX* se utilizando de renderizações pelo lado do servidor, têm seus *trade-offs.*

Conforme Pedroni (2021) sintetiza, até esse momento as soluções de renderização de aplicações web caem basicamente em duas categorias: 1.) O servidor fornece um documento *HTML* pronto, usando especialmente renderização pelo lado do servidor onde a página vai ser devolvida ao cliente, sendo renderizado pelo servidor a cada requisição (*MPA* + *AJAX*). 2) O servidor devolve um documento mínimo, apenas com *HTML* básico e as conexões dos *scripts* necessários para depois construir a página e hidratar a aplicação de acordo com a interatividade do usuário (*SPA*).

“É importante destacar que existe uma desvantagem em ambas essas tecnologias vigentes comparadas com como a *Web 1.0* e os documentos estáticos funcionavam. Para qualquer requisição, é necessário que o servidor ou o cliente faça algum tipo de processamento adicional para que o documento ou página chegue

num estado final. Com isso em mente, a pergunta que muitos se fizeram foi: “Será que é possível ter as vantagens de performance de documentos prontos e estáticos com a possibilidade de se customizar o que vai ser entregue ao cliente?” E eis que o conceito de Geração Estática apareceu”. (PEDRONI, 2021).

Para compreender o conceito de Geração Estática é preciso antes retornar ao funcionamento do processo de renderização em aplicações *web*. Basicamente, trata-se do processo de conversão de código para a representação em documento *HTML* que compõe a interface do usuário. No modelo *MPA*, o mais comum é a renderização ocorrer pelo lado do servidor (*SSR*), enquanto em *SPA* o padrão é a renderização pelo lado do cliente (*CSR*).

A Geração Estática inverte a lógica de renderização em tempo de execução e se utiliza de um princípio de pré-renderização em tempo de compilação. Isto é, ao invés de o documento *HTML* ser construído a cada requisição por cliente ou servidor, ele já está disponibilizado pelo próprio servidor antes mesmo de ser requisitado. Trata-se de uma volta às origens da *Web 1.0,* onde havia menos camadas e, portanto, tornando o processo mais veloz, com uma melhoria clara trazida pelas tecnologias de hoje: a possibilidade de arquivos estáticos carregarem dados dinâmicos sob demanda.

Dentre as ferramentas em que se é possível fazer essa tratativa, se destaca o *meta-framework* de *React.js,* conhecido como *Next.js.* Ele oferece um ferramental que une as últimas duas décadas de desenvolvimento *web:* a simplicidade das requisições de cliente e servidor da *web 1.0*, a personalização e poder da renderização pelo lado do servidor da *web 2.0* e a alta experiência do usuário trazida pela performance das *SPA’s.*

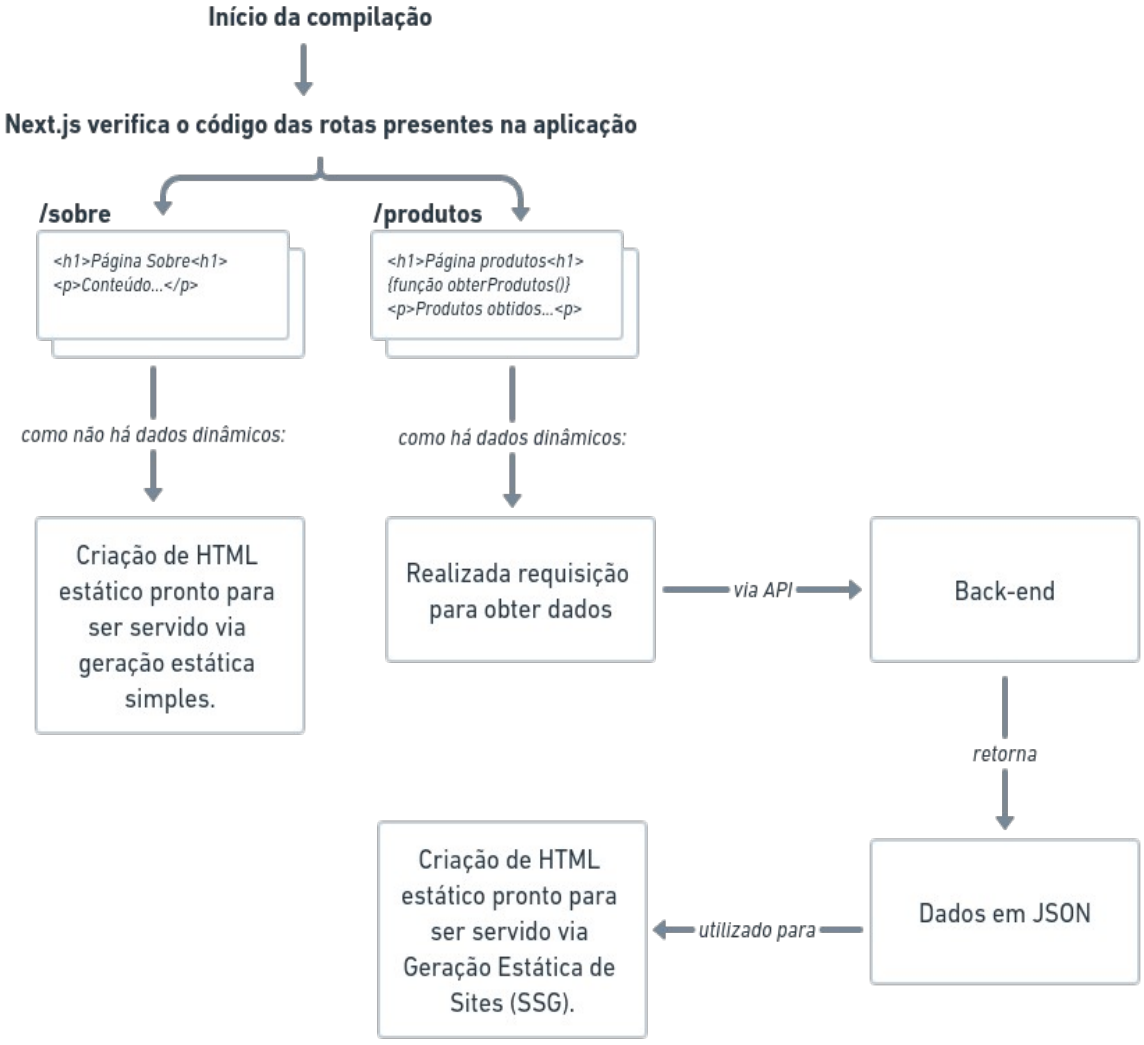
Sendo um *framework* construído a partir do *React.js* — uma biblioteca *JavaScript* para construir interfaces reativas, tida como a mais popular ferramenta para aplicação do modelo de *SPA* — o *Next.js* se aproveita do ecossistema e estrutura de desenvolvimento dessa biblioteca feita pelo time de desenvolvedores do *Facebook*.

> *Next.js* é um *framework React* que fornece blocos de construção para criar aplicações *web*. Por *framework*, queremos dizer que *Next.js* lida com as ferramentas e configurações necessárias para o *React* e fornece estrutura, recursos e otimizações adicionais para suas aplicações. Pode-se usar o *React* para criar a interface do usuário e adotar incrementalmente os recursos do *Next.js* para resolver requisitos comuns de aplicações *web* como roteamento, busca de dados, integrações — tudo isso enquanto melhora a experiência do desenvolvedor e do usuário final. (VERCEL, 2022).

Por padrão, o *Next.js* pré-renderiza cada rota de uma aplicação. Por pré-renderização significa que o *HTML* é gerado antecipadamente em um servidor, ao invés de ter tudo feito por *JavaScript* no dispositivo do usuário.

Dentre os tipos de pré-renderização oferecidos pelo *Next.js* há a geração estática simples (quando não há o consumo de dados dinâmicos), realizada de antemão no momento da compilação e antes de ser servido. Há também a Geração estática de *sites* (*SSG*), também construída da forma recém explanada, com o diferencial do consumo de dados externos.

Figura 06 - Exemplificação de fluxo de compilação do Next.js em páginas estáticas



Fonte: AUTOR, 2022

Ao pré-renderizar uma rota, o *Next.js* converte um código *React* em um *HTML* estático que pode ser armazenado em um servidor, permitindo que o mesmo arquivo possa ser retornado a cada requisição, distribuído em cache por *CDN's* (Redes de Fornecimento de Conteúdo) e entregue muito mais rápido aos usuários finais.

Isso ocorre porque navegadores não são exigidos para fazer um grande processamento inicial, especialmente por arquivos estáticos não possuírem scripts de alto custo computacional. Sem contar o ganho de performance pela diminuição da latência em razão do uso de *CDN's*.

Com elas, não há a necessidade de uma requisição aguardar a resposta de um servidor inteligente capaz de renderizar uma página. As solicitações são prontamente servidas por repositórios velozes e de baixo processamento localizados geograficamente próximos ao local de requisição.

"*CDN's* armazenam conteúdo estático (como *HTML* e arquivos de imagem) em vários locais ao redor do mundo e são colocados entre o cliente e o servidor de origem. Quando uma nova solicitação chega, o local *CDN* mais próximo do usuário pode responder com o resultado armazenado em cache" (VERCEL, 2022).

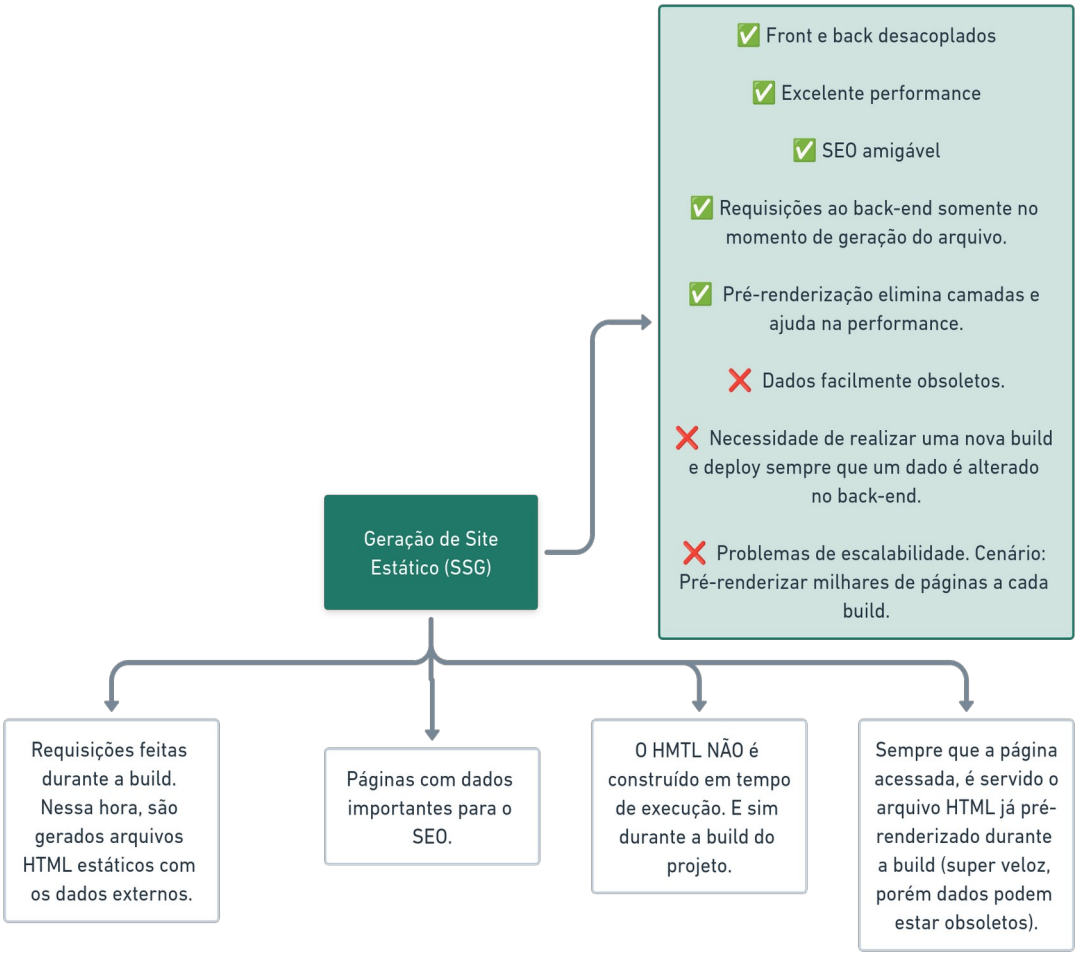
Comportamento esse, semelhante aos primeiros *sites* da *Web 1.0,* onde haviam transmissões simples de arquivos estáticos. Sem qualquer necessidade de interferência de um servidor inteligente em tempo de execução.

Porém, com a diferença da possibilidade de inserir dados externos durante a criação dos documentos *HTML.* Isso possibilita também uma diminuição de requisições ao *back-end*, banco de dados ou serviços terceiros, já que essas requisições são feitas uma única vez no momento da criação do documento.

Por fim, naturalmente, como não existe renderização pelo lado do cliente para construção da página não há qualquer prejuízo ao *SEO* e indexações por motores de busca.

Na Figura 07 pode-se observar um diagrama a respeito de páginas geradas com SSG, suas principais características e uma tabela com vantagens (simbolizadas por um sinal de checado) e desvantagens (sinalizados com um X).

Figura 07 - Diagrama de SSG



Fonte: AUTOR, 2022

Gerações estáticas são especialmente recomendadas em cenários em que não há mudanças recorrentes nos conteúdos necessários para composição da página. A justificativa é que os dados podem ficar facilmente obsoletos visto que eles são uma espécie de fotografia das informações que existiam na fonte dos dados no momento da geração do documento. Se a origem dos dados for alterada é necessária uma nova geração de páginas e publicação no servidor para que isso seja refletido no *site*.

“Se a sua aplicação trabalhar com dados em tempo real ou tem informações que atualizam constantemente, o *SGG* certamente não será uma solução prática uma vez que é necessário “compilar” seu projeto toda vez que desejar gerar um documento novo aos seus usuários”, (PEDRONI, 2011).

Entretanto, o *Next.js* oferece uma forma de lidar com o problema da limitação da vida útil dos dados trazidos pelo *SSG*. Trata-se da Regeneração Estática Incremental (*ISR*). Ela nada mais é do que *SSG* com uma pequena alteração: após

o primeiro usuário acessar a aplicação é disparado uma contagem regressiva no servidor com um período pré-determinado para que se gere uma nova versão da página com os dados mais atualizados.

> Você pode usar a Regeneração Estática Incremental para criar ou atualizar páginas estáticas depois de publicar o seu *site*. Isso significa que você não precisa recompilar toda a aplicação se seus dados forem alterados. Quando uma solicitação é feita para uma página que foi pré-renderizada em tempo de compilação, ela inicialmente mostrará a página em cache e após a janela de tempo pré-determinada, o *Next.js* aciona uma regeneração da página em segundo plano. Depois que a nova página for gerada com sucesso, o *Next.js* invalidará o cache e mostrará a página atualizada. Se a regeneração do plano de fundo falhar, a página antiga ainda permanecerá inalterada. (VERCEL, 2022).

Para evitar uma alta obsolescência de dados, a utilização de páginas geradas com *ISR* fez bastante sentido em algumas rotas da aplicação modelo, o *e-commerce Fireshoes*. Tanto a página inicial, quanto a de categorias, por exemplo, eram lugares que exigiam os benefícios oferecidos pela *SSG*, tais como *SEO* e baixo volume de consultas ao *back-end a*o mesmo tempo não poderiam ficar reféns de uma nova compilação e publicação do *site* sempre que uma informação fosse inserida ou atualizada no *back-end*.

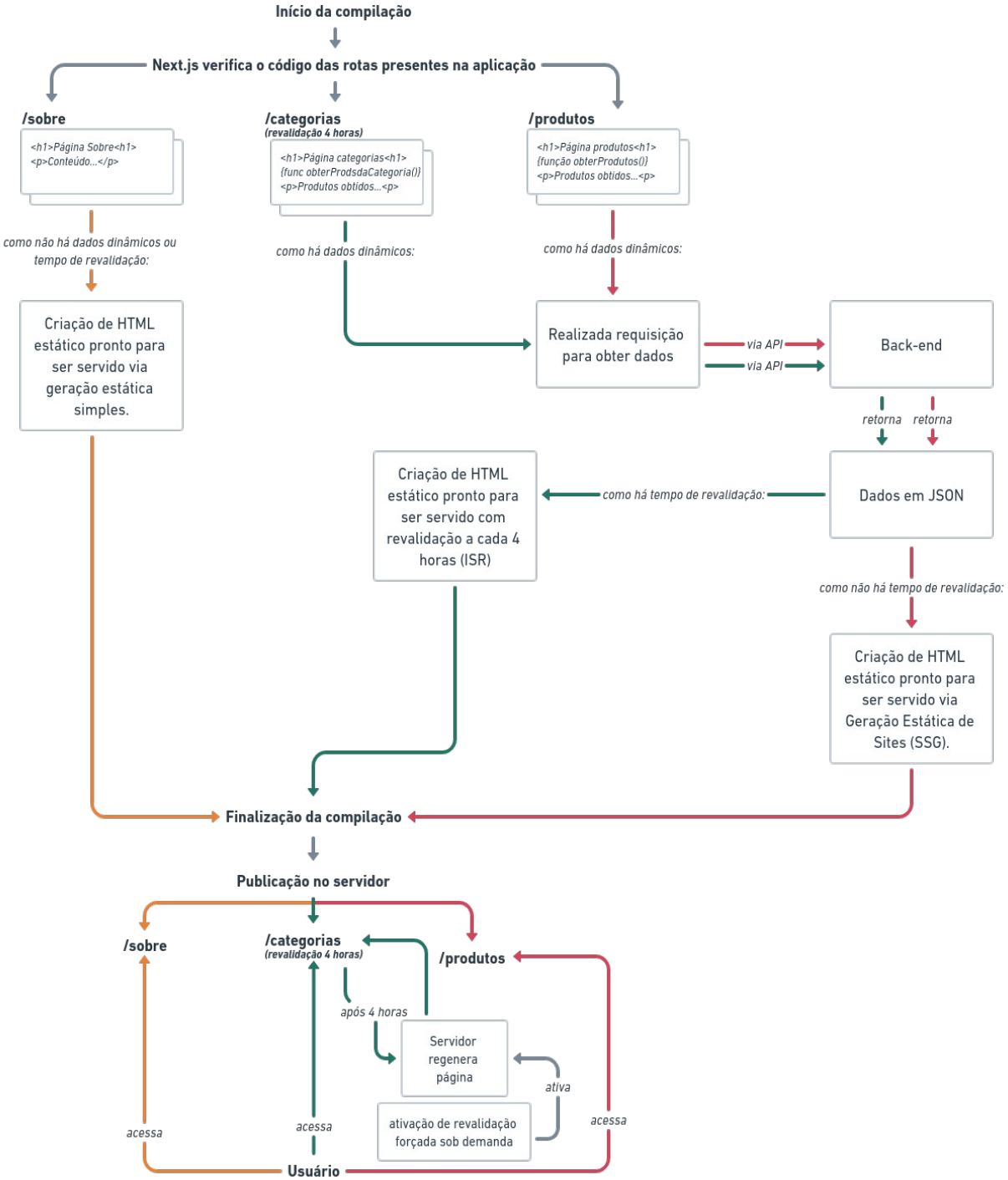
Nesse cenário, portanto, o emprego de *ISR* era ideal para manter os dados minimamente frescos, já que uma nova versão da página seria gerada após o tempo de revalidação ser disparado. Havia ainda, de toda a forma, um certo nível de fragilidade no que se refere a chance de apresentar dados obsoletos a um usuário. Ela ocorria justamente durante a janela de intervalo entre o tempo de revalidação pré-definido para a regeneração da página.

Exemplificando: Em um cenário em que a página de categorias tem uma revalidação a cada quatro horas, por exemplo, se o preço de um produto é alterado no banco de dados, ele seria exibido com o preço defasado na prateleira existente na página de categorias até que ela fosse regenerada após quatro horas. Em um caso real, essa situação poderia gerar um enorme prejuízo.

Como forma de resolver essa problemática, as páginas que utilizam *ISR* também possuem outro gatilho além da regeneração automática por tempo. Existem revalidações sob demanda através de um simples acesso a um *endpoint*. Na aplicação modelo, basta que se envie uma requisição a uma rota específica, tal

como “*dominio.com.br/api/revalidate-category*” que a rota descrita seria regenerada no mesmo instante, independente de janela temporal.

Figura 08 - Exemplificação de fluxo de compilação do Next.js em páginas estáticas com SSG e ISR



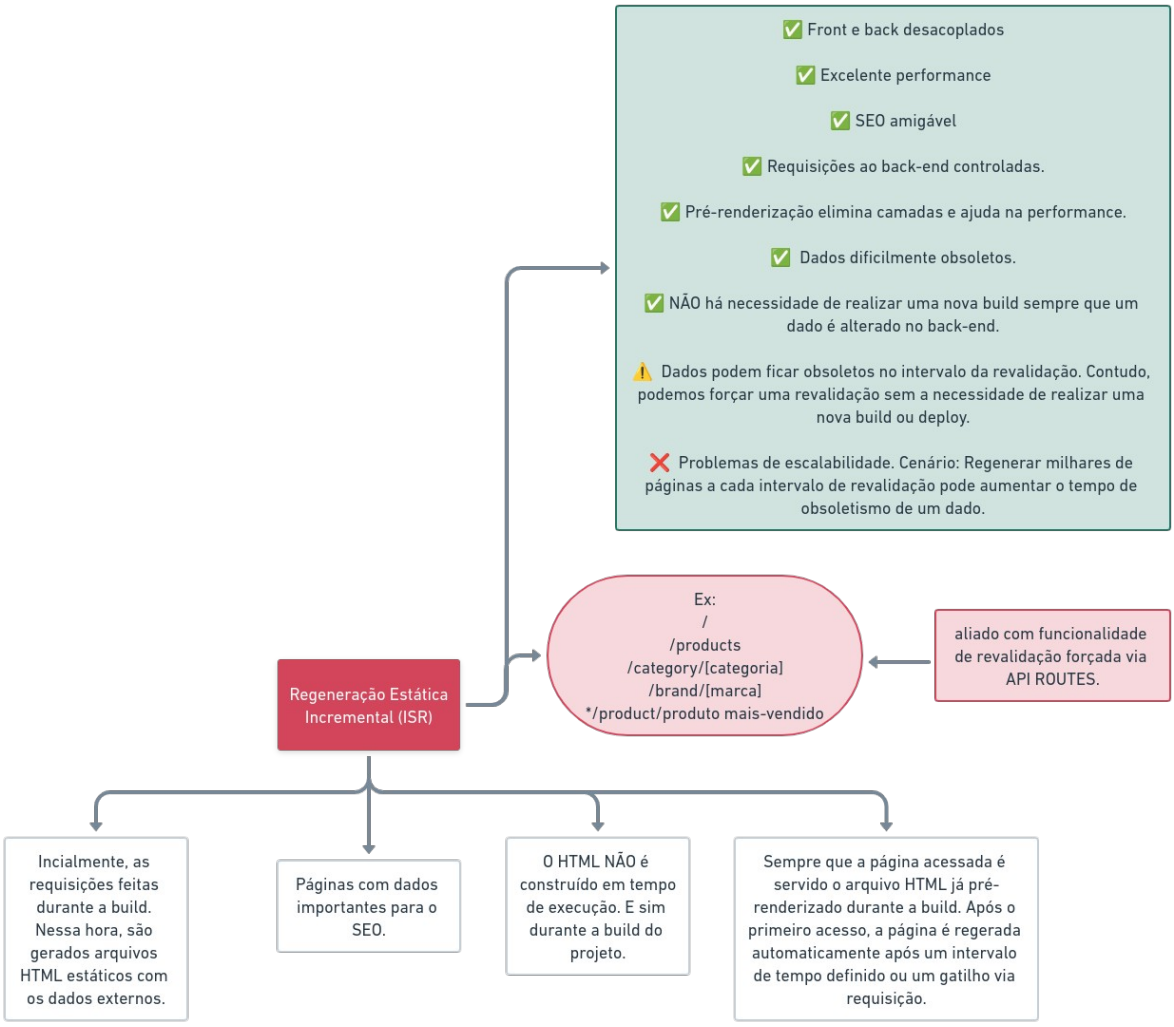
Fonte: AUTOR, 2022

Considerando que o *ISR* é praticamente um *SSG* com uma pequena regra de validação, à primeira vista, é possível inferir que o primeiro sempre vai ser uma

melhor opção que o segundo. Contudo, segundo Pedroni (2011), o *ISR* necessita de um servidor “inteligente” que possua uma linguagem de programação *back-end* capaz de fazer requisições à fonte de dados que vai fornecer as informações necessárias para gerar a nova versão do arquivo *HTML* estático. Já na hipótese do *SSG*, uma simples *CDN* que simplesmente devolve documentos atenderia o caso de uso.

A Figura 09 apresenta um diagrama que enumera as caraterísticas do modelo ISR (quadros inferiores), benefícios e prejuízos (quadro superior) e em quais rotas do estudo de caso foram utilizadas (círculo).

Figura 09 - Diagrama de ISR



Fonte: AUTOR, 2022

Outra utilização de *ISR* na aplicação modelo *Fireshoes* ocorre em rota de produtos específicos. Como será explorado mais abaixo, por padrão, no *e-*

commerce desenvolvido durante o projeto, as rotas de produtos utilizam a lógica de renderização pelo lado do servidor (*SSR*). Porém, a motivação para se ter produtos específicos com *ISR* se deu pela estratégia de possuir documentos estáticos para páginas muito acessadas.

A ideia é evitar um cenário de alto volume de requisições ao *back-end* ocasionada pelo intenso tráfego de um produto popular, por exemplo. Imaginando uma situação fictícia de promoção da loja, onde ocorresse um pico de acessos em produtos específicos, servir a mesma página estática para cada um dos acessos seria mais indicado do que sofrer com múltiplas consultas ao *back-end* para que este forneça a exata mesma resposta e cópias idênticas de uma página sejam construídas diversas vezes.

Utilizar *SSG* ou *ISR* nesse cenário resolveria problemas de alto volume de requisições ao mesmo tempo que melhoraria a própria experiência do usuário, uma vez que páginas estáticas são notadamente entregues com mais velocidade do quando precisam ser construídas em tempo de execução pelo servidor.

A motivação para se ter o comportamento padrão das rotas de produtos utilizando *SSR* se deu simplesmente para reduzir o risco de dados obsoletos, o que pode ocorrer em gerações estáticas.

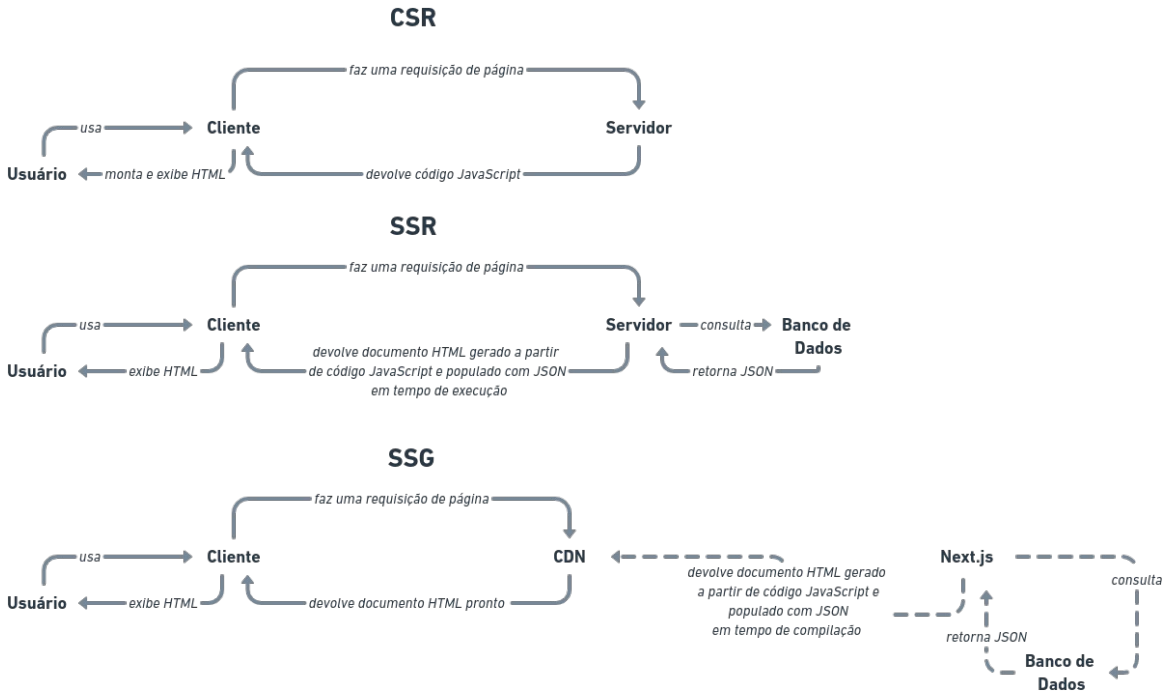
Quando qualquer página que utilize estratégia de *SSR* é acessada por cliente, seja pelo navegador, ou mesmo *crawlers* de buscadores, o arquivo *HTML* é prontamente construído no servidor e devolvido em tempo de execução. A construção não é feita pelo cliente e sim pelo servidor.

> Com a renderização do lado do servidor, o *HTML* da página é sempre gerado no servidor a cada acesso. O *HTML*, *JSON* e instruções *JavaScript* gerados são enviados ao cliente. No cliente, o *HTML* é usado para mostrar uma página não interativa rápida, enquanto o *React* usa os dados JSON e as instruções *JavaScript* para tornar os componentes interativos (por exemplo, anexando manipuladores de eventos a um botão). Esse processo é chamado de hidratação. (VERCEL, 2022).

Esse processo é comum e inclusive já utilizado anteriormente à existência de *CSR*, quando havia predominância de *MPA's*. Contudo, tecnologias modernas, como o *Next. js,* oferecem a possibilidade de unir o melhor de cada um dos modelos de desenvolvimento de aplicações web construídos nos últimos vinte anos trazendo uma abordagem moderna e híbrida. Nele, pode-se escolher o método de renderização mais apropriado para determinado caso de uso.

Na aplicação proposta por este projeto, fez sentido utilizar *SSR* para as páginas de produto a fim de garantir que os dados destes sejam sempre os mais atualizados possíveis. Isso porque sempre que uma página de produto for acessada será realizada uma consulta ao banco de dados, que por sua vez vai retornar as últimas informações relacionadas a ele (tais como preço, nome, etc), independentemente de qualquer tipo de *cache* ou documento previamente armazenado. Desta forma, ao mesmo tempo garante-se dados atualizados juntamente com todos os benefícios do *SEO* devido a resposta ser sempre um arquivo *HMTL* e não um código *JavaScript* como seria no caso de *CSR.*

Figura 10 - Fluxo CSR, SSR e SSG



Fonte: AUTOR, 2022

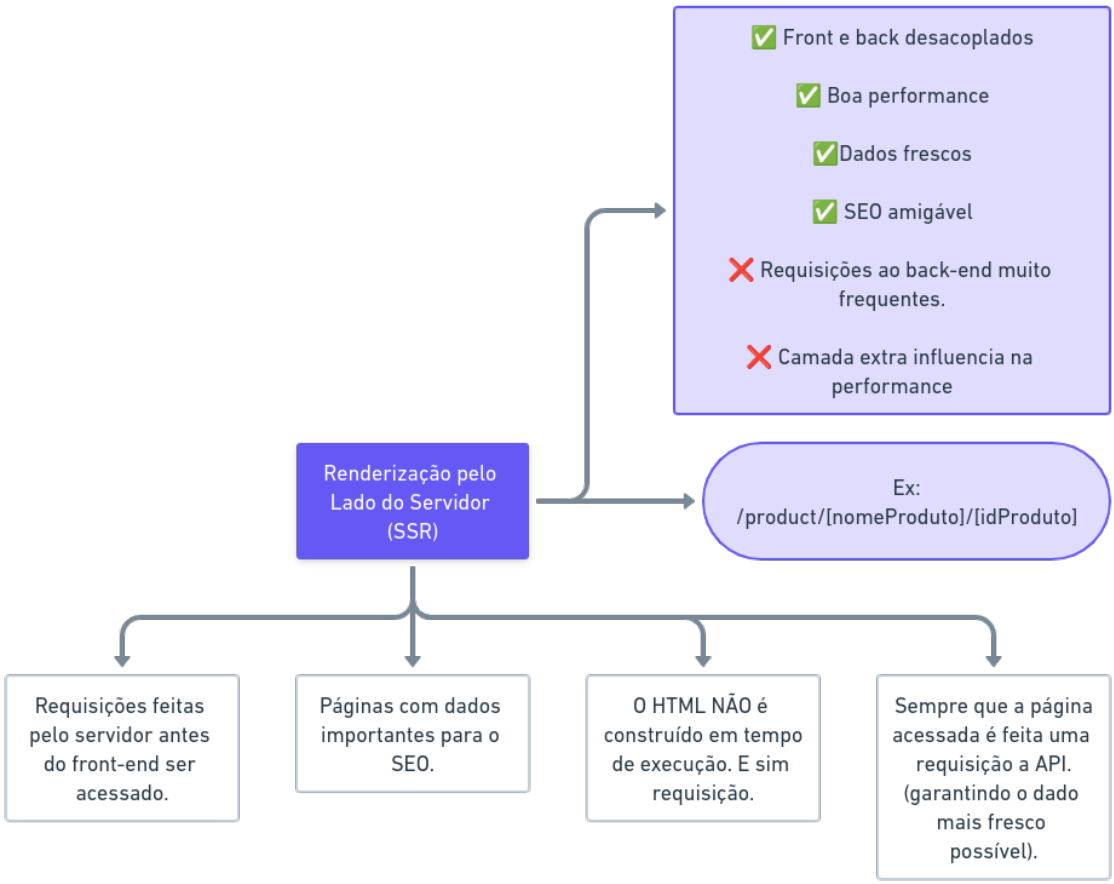
Na aplicação proposta por este projeto, fez sentido utilizar SSR para página de produtos a fim de garantir que os dados destes sejam sempre os mais atualizados possíveis, tendo em vista que sempre que suas rotas forem acessadas será realizada uma consulta ao banco de dados, que por sua vez vai retornar as últimas informações, independentemente de qualquer tipo de cache ou documento previamente armazenado. Ao mesmo tempo, lançando mão de todos os benefícios

trazidos pelo *SEO* devido a resposta do processo de *SSR* ser sempre um arquivo *HTML* e não um código *JavaScript* como seria no caso de *CSR.*

Na aplicação *Fireshoes*, quando um usuário acessa uma rota de produto, o servidor consulta o banco de dados utilizando o código de identificação passado como parâmetro na *URL* para obter as informações do produto em questão e a retorna em *JSON* para que o servidor possa renderizar o arquivo *HTML* já com as informações contidas no *JSON* devidamente populadas.

O diagrama presente na Figura 11 exibe algumas das particularidades da estratégia *SSR* (quadros inferiores), benefícios e prejuízos (quadro superior) e em quais rotas do estudo de caso foram utilizadas (círculo).

Figura 11 - Diagrama de SSR



Fonte: AUTOR, 2022

A utilização em conjunto de múltiplas estratégias que dialogam com diferentes épocas do desenvolvimento *web* aliada à construção de uma aplicação relativamente robusta e moderna foi o fio condutor deste projeto. Ele, sobretudo, se debruçou em variadas possibilidades de se atacar problemáticas dos dias atuais.

Por isso, o processo de estudar os relacionamentos entre cliente e servidor, métodos de renderização, maneiras de se obter páginas funcionais e performáticas foi especialmente desafiador. O projeto tinha como maior premissa aplicar diferentes conceitos técnicos no contexto de um caso de uso factível e fidedigno, como é uma aplicação de *e-commerce*.

A forma com que as pessoas se relacionam com a tecnologia, seja como usuários ou desenvolvedores vem ganhando cada vez novas camadas de peculiaridades. À medida que velhos problemas vão sendo superados, novos e mais complexos surgem a cada dia. Este trabalho buscou inserir-se justamente nesse cenário. Incorporar e vislumbrar planos de ação a fim de melhorar *e-commerces*, um tipo de aplicação que esteve em ampla ascensão nos últimos anos.

A proposta de discutir opções de renderização de páginas na web, arquitetura de *software,* encontrabilidade e interação entre as entidades cliente-servidor contidas neste projeto abarca temáticas importantes e contemporâneas do desenvolvimento web. Especialmente, à luz das novas responsabilidades que a camada de *front-end* ganhou com a maior capacidade dos navegadores modernos.

Por fim, o presente estudo exemplificou que existem diversas possibilidades de se construir aplicações performáticas utilizando tecnologias baseadas em bibliotecas e frameworks *JavaScript* para construção de interfaces sem renunciar a outros requisitos vitais, e por muitas vezes escanteados, como é o caso do *SEO* para lojas virtuais, por exemplo.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVARENGA, Darlan. **Com pandemia, comércio eletrônico tem salto em 2020 e dobra participação no varejo brasileiro**. G1. Disponível em: <https://g1.globo.com/economia/noticia/2021/02/26/com-pandemia-comercio-eletronico-tem-salto-em-2020-e-dobra-participacao-no-varejo-brasileiro.ghtml>. Primeiro acesso em: 22/09/2021.

ANDERSON, Shaun. **H*ow Fast Should A Website Load & How To Speed It Up***. Hobo. Disponível em: <https://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/>. Primeiro acesso em: 22/09/2021.

CAVALCANTE, Johnny. **Descomplicando SPA's**. Medium. Disponível em: <https://medium.com/trainingcenter/descomplicando-spas-caa8f57bdbf3>. Primeiro acesso em: 22/09/2021.

ECBR. **88% dos brasileiros compram online e dão preferência para roupas, alimentos e eletrodomésticos**. Disponível em: < https://www.ecommercebrasil.com.br/noticias/brasileiros-online-compras-roupas/ >. Primeiro acesso em: 22/09/2021.

EDITORIAL AELA.IO. **Wireframe: O Que é e Como Criar Um Para Seus Projetos de UX Design?** Medium. Disponível em: <https://medium.com/aela/wireframe-o-que-%C3%A9-e-como-criar-seu-primeiro-fab2fdecbb56>. Primeiro acesso em: 16/10/2021.

KHALIFA, Zenaib. ***Multi-page, Single-page, or a Hybrid?***. Medium. Disponível em: <https://medium.com/swlh/spa-mpa-or-a-hybrid-42fdf6b3415c />. Primeiro acesso em: 18/04/2022.

KINOSHITA, Marisa. **Nova pesquisa aponta como a tecnologia pode ajudar os e-commerces brasileiros a melhorar a experiência do usuário**. Think With Google. Disponível em: <https://www.thinkwithgoogle.com/intl/pt-br/tendencias-de-consumo/

jornada-do-consumidor/nova-pesquisa-aponta-como-a-tecnologia-pode-ajudar-os-e-commerces-brasileiros-a-melhorar-a-experiencia-do-usuario/>. Primeiro acesso em: 22/09/2021.

PEDRONI, Ricardo. **Server Side Rendering, Single Page Application, Static Site Generation, Incremental Static Regeneration e Next.js (pt. 1)**. Medium. Disponível em: <https://medium.com/reactbrasil/server-side-rendering-vs-static-site-generation-vs-single-page-application-vs-incremental-static-e91804ef9401>. Primeiro acesso em: 22/09/2021.

PEDRONI, Ricardo. **Server Side Rendering, Single Page Application, Static Site Generation, Incremental Static Regeneration e Next.js (pt. 2)**. Medium. Disponível em: <https://medium.com/reactbrasil/server-side-rendering-single-page-application-static-site-generation-incremental-static-f9b5d9b0bf8d>. Primeiro acesso em: 22/09/2021.

PEDRONI, Ricardo. **Server Side Rendering, Single Page Application, Static Site Generation, Incremental Static Regeneration e Next.js (pt. 3)**. Medium. Disponível em: <https://medium.com/reactbrasil/server-side-rendering-single-page-application-static-site-generation-incremental-static-1de36475d0c8>. Primeiro acesso em: 22/09/2021.

TEJADA, Zoiner. **Dados não relacionais e NoSQL.** Documentação Azure Microsoft. Disponível em: <https://docs.microsoft.com/pt-br/azure/architecture/data-guide/big-data/non-relational-data >. Primeiro acesso em: 16/10/2021.

VENTURA, Plínio. **O que é UML (Unified Modeling Language)**. Até o Momento. Disponível em: < https://www.ateomomento.com.br/diagramas-uml/ >. Primeiro acesso em: 08/11/2021.

VENTURA, Plínio. **Entendendo definitivamente o que é um Caso de Uso**. Até o Momento. Disponível em: < https://www.ateomomento.com.br/o-que-e-caso-de-uso/ >. Primeiro acesso em: 08/11/2021.

VERCEL**. Documentação Next Js.** Disponível em: <https://nextjs.org/docs/getting-started>. Primeiro acesso em: 22/09/2021.

APÊNDICE A – DIAGRAMA DE CASO DE USO

Figura 12 - Diagrama de Caso de Uso do ponto de vista de um cliente

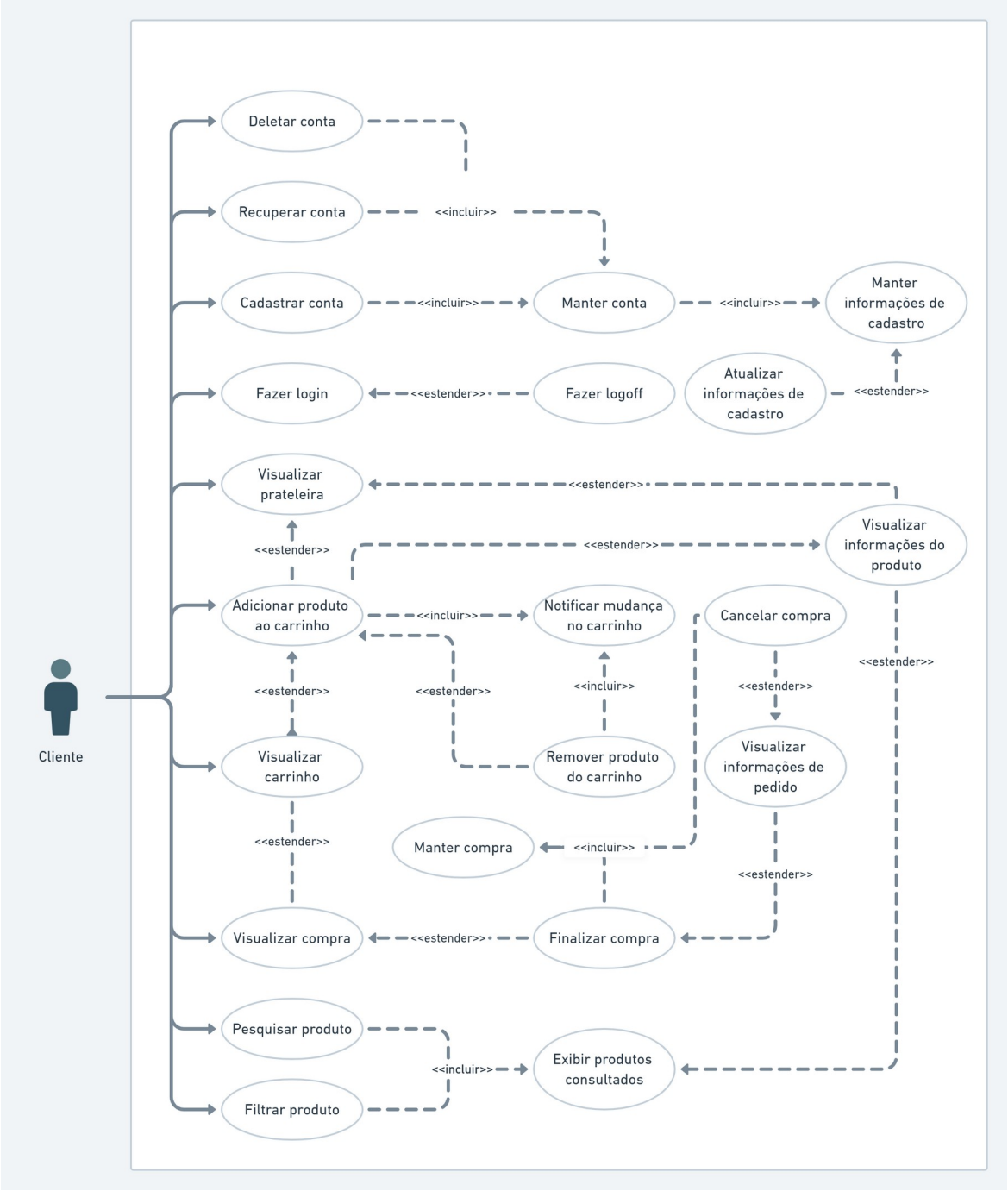
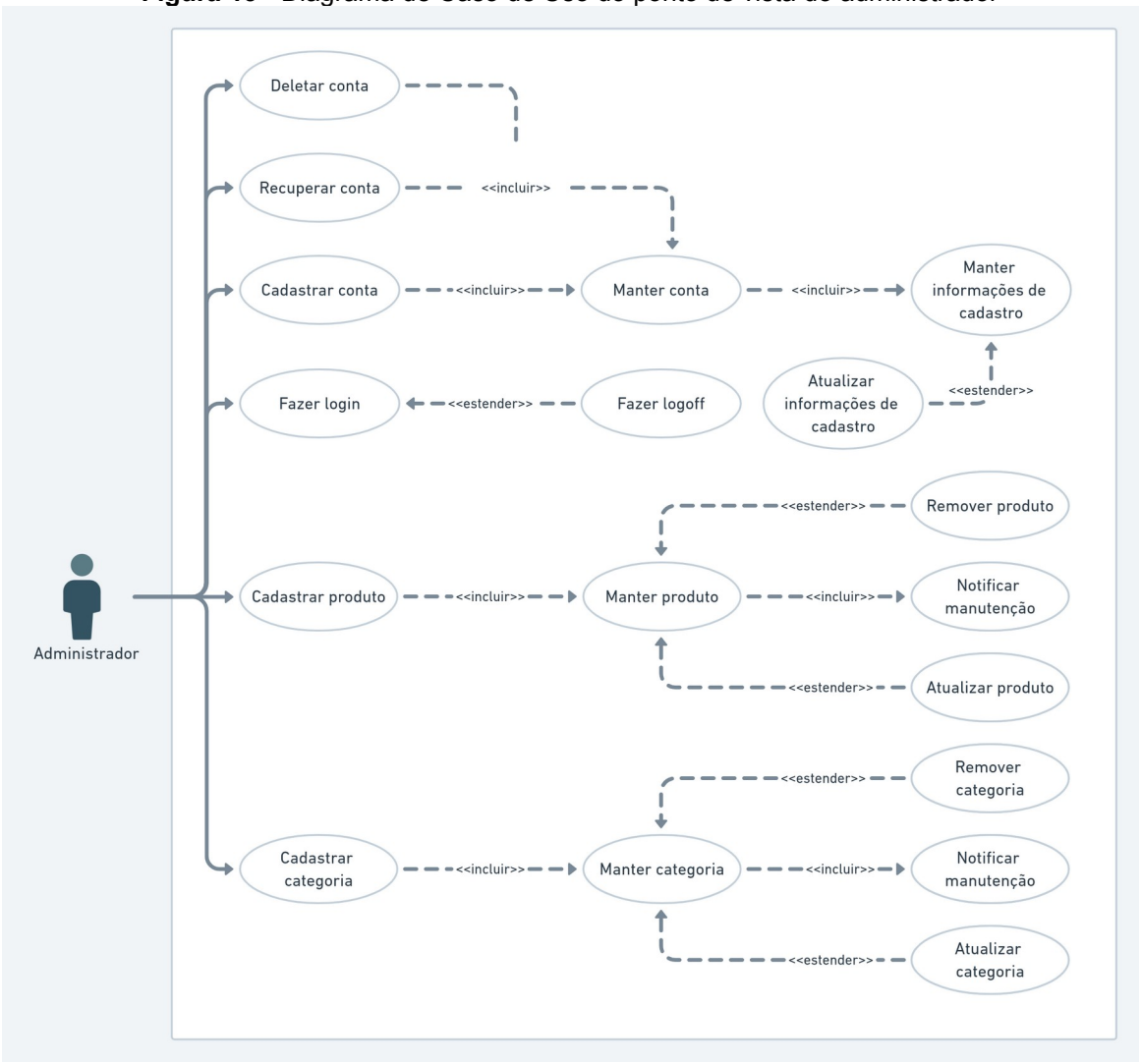


Figura 13 - Diagrama de Caso de Uso do ponto de vista do administrador



APÊNDICE B – DESCRIÇÃO DOS FLUXOS DE EVENTOS

Casos de uso: Cadastrar Conta
Descrição: Fluxo do cadastro feito por um cliente
Atores: Cliente ou Administrador
Pré-condições: 1 - O login a ser cadastrado não pode existir no sistema.
Pós-condições: 1 - O usuário deve estar cadastrado no banco de dados do sistema.
Fluxo principal: 1 - O usuário clica no botão de login realizado no cabeçalho da página. 2 - O usuário clica no botão ‘fazer cadastro’ para acessar o formulário de cadastro. 3 - O usuário preenche o formulário de cadastro. 4 - O usuário confirma o envio dos dados. 5 - O sistema valida os dados digitados no formulário. 6 - O sistema cadastra o usuário no banco de dados. 7 - O sistema notifica o sucesso do cadastro ao usuário.
Fluxo alternativo: 5a - O sistema identifica que os dados preenchidos estão inválidos. 5a.1 - O sistema notifica o usuário que os dados estão incorretos. 5b – O sistema identifica que já existe cadastro com o(s) dado(s) informados. 5b.1 - O sistema notifica o usuário que já existe um cadastro com os dados informados.

Casos de uso: Fazer Login com Senha
Descrição: Fluxo do login realizado por um usuário através de senha.
Atores: Cliente ou Administrador
Pré-condições: 1 - As informações preenchidas no formulário de login devem estar corretas. 2 - O usuário deve possuir uma conta cadastrada.

Pós-condições: 1 - O sistema permite a autenticação e o usuário é direcionado para a página inicial.
Fluxo principal: 1 - O usuário clica no botão de login realizado no cabeçalho da página. 2 - O usuário clica no botão ‘fazer login’ para acessar o formulário de login. 3 - O usuário preenche o formulário de login. 4 - O usuário confirma o envio dos dados. 5 - O sistema valida os dados digitados no formulário. 6 - O sistema realiza a autenticação do usuário. 7 - O sistema direciona o usuário para a página inicial.
Fluxo alternativo: 5a - O sistema identifica que os dados preenchidos estão inválidos. 5a.1 - O sistema notifica o usuário que os dados estão incorretos.

Casos de uso: Fazer login social
Descrição: Fluxo do login social realizado por um usuário.
Atores: Cliente
Pré-condições: 1 - O usuário deve possuir uma conta ativa no serviço/rede social que fará a autenticação via login social.
Pós-condições: 1 - O sistema permite a autenticação e o usuário é direcionado para a página inicial.
Fluxo principal: 1 - O usuário clica no botão de login realizado no cabeçalho da página. 2 - O usuário clica no botão ‘fazer login’ para acessar o formulário de login. 3 - O usuário clica em ‘login com ... (serviço/rede social)’ 4 - O sistema valida a autenticação. 5 - O sistema realiza a autenticação do usuário. 6 - O sistema direciona o usuário para a página inicial.
Fluxo alternativo: 4a - O usuário não possui uma conta no serviço/rede social que fará a autenticação via login social. 4a.1 - O usuário retorna para a página de login.

Casos de uso: Recuperar conta
Descrição: Fluxo de recuperação de conta realizado por um usuário.

Atores: Cliente
Pré-condições: 1 - O usuário deve possuir uma conta cadastrada.
Pós-condições: 1- O sistema atualiza as informações cadastrais de senha. 2- O sistema notifica que a conta foi recuperada com sucesso.
Fluxo principal: 1 - O usuário clica no botão de login realizado no cabeçalho da página. 2 - O usuário clica no botão ‘fazer login’ para acessar o formulário de login. 3 - O usuário clica em recuperar conta. 4 - O usuário digita o e-mail cadastrado para receber um link de recuperação de conta. 5 - O sistema envia um e-mail ao usuário com um link de recuperação de conta. 6 - O usuário acessa o link de recuperação de conta. 7 - O usuário digita a sua nova senha. 8 - O sistema atualiza as informações cadastrais de senha. 8 - O sistema notifica que a conta foi recuperada com sucesso.
Fluxo alternativo: 4a - O e-mail digitado é inválido. 4a.1 - O sistema notifica o usuário de que o e-mail digitado é inválido. 5a - O usuário não acessa o e-mail para a recuperação de conta em um intervalo de tempo pré-determinado. 5a.1 - A tentativa de recuperação de conta é expirada. 5a.2 - A operação não é concluída.

Casos de uso: Deletar conta
Descrição: Fluxo de recuperação de conta realizado por um usuário.
Atores: Cliente
Pré-condições: 1 - O usuário deve possuir uma conta cadastrada. 2 - O usuário deve estar logado.
Pós-condições: 1 - O sistema deleta uma conta cadastrada.
Fluxo principal:

1 - O usuário clica no botão da área do usuário. 2 - O usuário clica no botão deletar conta. 3 - O usuário confirma que deseja deletar a conta. 4 - O sistema realiza o logoff do usuário 5 - O sistema deleta a conta.
Fluxo alternativo: 3a - O usuário desiste de deletar a conta. 3a.1 - A operação de deletar conta não é realizada.

Casos de uso: Fazer logoff
Descrição: Fluxo do logoff realizado por um usuário.
Atores: Cliente ou Administrador
Pré-condições: 1 - O usuário deve estar logado.
Pós-condições: 1 - O usuário estará deslogado.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário clica no botão de logoff. 3 - O sistema realiza o logoff do usuário.

Casos de uso: Adicionar produto ao carrinho
Descrição: Fluxo do usuário para adicionar um produto ao carrinho.
Atores: Cliente
Pré-condições: 1 - O produto deve possuir estoque.
Pós-condições: 1 - O sistema adiciona o produto ao carrinho. 2 - O sistema diminui o estoque do produto. 3 - O usuário é notificado que o produto foi adicionado ao carrinho.

Fluxo principal: 1 - O usuário visualiza um produto. 2 - O usuário clica no botão adicionar ao carrinho. 3 - O sistema adiciona o produto ao carrinho. 4 - O sistema diminui o estoque do produto. 5- O usuário é notificado que o produto foi adicionado ao carrinho.

Casos de uso: Visualizar carrinho
Descrição: Fluxo do usuário para visualizar o carrinho.
Atores: Cliente
Pré-condições: -
Pós-condições: 1 - O sistema exibe o conteúdo do carrinho.
Fluxo principal: 1 - O usuário clica no botão do carrinho. 2 - O sistema exibe os itens do carrinho juntamente com as opções de acrescentar, diminuir, remover itens e ir para a página de checkout para visualizar a compra.
Fluxo alternativo: 2a - O sistema identifica que o carrinho está vazio. 2a.1 - O sistema exibe a mensagem de que o carrinho está vazio.

Casos de uso: Acrescentar quantidade de um produto no carrinho
Descrição: Fluxo do usuário para acrescentar a quantidade de um produto no carrinho.
Atores: Cliente
Pré-condições: 1 - O carrinho não pode estar vazio. 2 - O produto deve possuir estoque compatível com a adição.
Pós-condições: 1 - O sistema acrescenta a quantidade do produto no carrinho. 2 - O sistema diminui o estoque do produto. 3 - O usuário é notificado que o produto foi adicionado ao carrinho.

Fluxo principal: 1 - O usuário clica no botão do carrinho. 2 - O sistema exibe os itens do carrinho juntamente com as opções de acrescentar, diminuir, remover itens e ir para a página de checkout para visualizar a compra. 3 - O usuário clica no botão de '+' ao lado do produto. 4- O sistema acrescenta a quantidade do produto no carrinho. 5- O sistema diminui o estoque do produto. 6- O usuário é notificado que o produto foi adicionado ao carrinho.
Fluxo alternativo: 2a - O sistema identifica que não há estoque compatível com a adição. 2a.1 - O sistema não realiza a adição. 2a.2 - O sistema notifica que não há estoque suficiente.

Casos de uso: Diminuir quantidade de um produto no carrinho
Descrição: Fluxo do usuário para diminuir a quantidade de um produto no carrinho.
Atores: Cliente
Pré-condições: 1 - O carrinho não pode estar vazio. 2 - O carrinho deve possuir ao menos duas unidades de um mesmo produto.
Pós-condições: 1 - O sistema diminui a quantidade do produto no carrinho. 2 - O sistema 'devolve' o produto ao estoque. 3 - O usuário é notificado que o produto foi removido do carrinho.
Fluxo principal: 1 - O usuário clica no botão do carrinho. 2 - O sistema exibe os itens do carrinho juntamente com as opções de acrescentar, diminuir, remover itens e ir para a página de checkout para visualizar a compra. 3 - O usuário clica no botão de '—' ao lado do produto. 4 - O sistema diminui a quantidade do produto no carrinho. 5 - O sistema 'devolve' o produto ao estoque. 6 - O usuário é notificado que o produto foi removido do carrinho.

Casos de uso: Visualizar compra
Descrição: Fluxo do usuário para visualizar detalhes da compra a ser realizada.

Atores: Cliente
Pré-condições: 1 - O carrinho não pode estar vazio.
Pós-condições: 1 - O sistema exibe a página de checkout com a possibilidade de finalizar a compra.
Fluxo principal: 1 - O usuário clica no botão do carrinho. 2 - O sistema exibe os itens do carrinho juntamente com as opções de acrescentar, diminuir, remover itens e ir para a página de checkout para visualizar a compra. 3 - O usuário clica no botão de 'finalizar compra'. 4 - O sistema exibe a página de checkout para confirmar os dados de pagamento e oferece a possibilidade de finalizar a compra.
Fluxo alternativo: 2a - O sistema identifica que o carrinho está vazio. 2a.1 - O sistema exibe a mensagem de que o carrinho está vazio.

Casos de uso: Finalizar compra
Descrição: Fluxo do usuário para finalizar a compra.
Atores: Cliente
Pré-condições: 1 - O carrinho não pode estar vazio. 2 - O usuário deve estar logado. 3 - O usuário deve possuir um endereço de entrega cadastrado. 4 - O usuário deve possuir um meio de pagamento cadastrado.
Pós-condições: 1 - O usuário confirma a compra. 2 - O sistema registra a compra. 3 - O sistema exibe o sucesso da compra ao usuário.
Fluxo principal: 1 - O usuário clica no botão do carrinho. 2 - O sistema exibe os itens do carrinho juntamente com as opções de acrescentar, diminuir, remover itens e ir para a página de checkout para

visualizar a compra. 3 - O usuário clica no botão de ‘finalizar compra’. 4 - O sistema exibe a página de checkout para confirmar os dados de pagamento e oferece a possibilidade de finalizar a compra. 5 - O usuário confirma o endereço de entrega. 6 - O usuário confirma o meio de pagamento 7 - O usuário confirma a compra. 8 - O sistema registra a compra. 9 - O sistema exibe o sucesso da compra ao usuário.
Fluxo alternativo: 2a - O sistema identifica que o carrinho está vazio. 2a.1 - O sistema exibe a mensagem de que o carrinho está vazio. 5a - O sistema identifica que o usuário não realizou login. 5a.1 - O sistema exibe o botão ‘fazer login’. 5a.2 - O sistema direciona o usuário para realizar um login ou cadastrar uma conta. 5b - O sistema identifica que o usuário não possui um endereço de entrega cadastrado. 5b.1 - O sistema exibe botão de cadastrar um endereço de entrega. 5b.2 - O sistema direciona o usuário para cadastrar um endereço de entrega. 6a - O sistema identifica que o usuário não possui um meio de pagamento cadastrado. 6b.1 - O sistema exibe botão de cadastrar um meio de pagamento. 6b.2 - O sistema direciona o usuário para cadastrar um meio de pagamento.

Casos de uso: Cadastrar endereço de entrega
Descrição: Fluxo do usuário para cadastrar um endereço de entrega.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O endereço de entrega a ser cadastrado não pode possuir as mesmas informações de um endereço de entrega já cadastrado.
Pós-condições: 1 - O sistema cadastra um endereço de entrega. 2 - O sistema notifica o usuário que endereço de entrega foi cadastrado com sucesso.
Fluxo principal:

1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de endereços de entrega. 3 - O usuário clica no botão de cadastrar um endereço de entrega. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema cadastra um endereço de entrega. 8 - O sistema notifica o usuário que endereço de entrega foi cadastrado com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há um endereço de entrega cadastrado com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há um endereço de entrega cadastrado com as mesmas informações.

Casos de uso: Editar endereço de entrega
Descrição: Fluxo do usuário para editar um endereço de entrega.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O usuário deve possuir ao menos um endereço de entrega cadastrado. 4 - As novas informações do endereço de entrega a ser cadastrado não podem ser iguais às de um endereço de entrega já cadastrado.
Pós-condições: 1 - O sistema atualiza um endereço de entrega cadastrado. 2 - O sistema notifica o usuário que endereço de entrega foi editado com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de endereços de entrega. 3 - O usuário clica no botão de editar um endereço de entrega. 4 - O usuário preenche o formulário.

5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema edita um endereço de entrega cadastrado. 8 - O sistema notifica o usuário que endereço de entrega foi editado com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há um endereço de entrega cadastrado com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há um endereço de entrega cadastrado com as mesmas informações.

Casos de uso: Remover endereço de entrega
Descrição: Fluxo do usuário para remover um endereço de entrega.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - O usuário deve possuir ao menos dois endereços de entrega cadastrados.
Pós-condições: 1 - O sistema remove um endereço de entrega cadastrado. 2 - O sistema notifica o usuário que endereço de entrega foi removido com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de endereços de entrega. 3 - O usuário clica no botão de remover um endereço de entrega. 5 - O usuário confirma a remoção do endereço de entrega. 6 - O sistema verifica se há mais de um endereço cadastrado. 7 - O sistema remove um endereço de entrega cadastrado. 8 - O sistema notifica o usuário que o endereço de entrega foi removido com sucesso.
Fluxo alternativo: 6a - O sistema identifica que o usuário possui apenas um endereço de entrega cadastrado 6a.1 - O sistema não realiza a operação.

6a.2 - O sistema notifica que é preciso possuir ao menos um endereço de entrega cadastrado.

Casos de uso: Cadastrar meio de pagamento
Descrição: Fluxo do usuário para cadastrar um meio de pagamento.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O meio de pagamento a ser cadastrado não pode possuir as mesmas informações de um meio de pagamento já cadastrado.
Pós-condições: 1 - O sistema cadastra um meio de pagamento. 2 - O sistema notifica o usuário que o meio de pagamento foi cadastrado com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de meios de pagamento. 3 - O usuário clica no botão de cadastrar um meio de pagamento. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema cadastra um meio de pagamento. 8 - O sistema notifica o usuário que o meio de pagamento foi cadastrado com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há um meio de pagamento cadastrado com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há um meio de pagamento cadastrado com as mesmas informações.

Casos de uso: Editar meio de pagamento
Descrição: Fluxo do usuário para editar um meio de pagamento.

Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O usuário deve possuir ao menos um meio de pagamento cadastrado. 4 - As novas informações do meio de pagamento a ser cadastrado não podem ser iguais às de um meio de pagamento já cadastrado.
Pós-condições: 1 - O sistema atualiza um meio de pagamento cadastrado. 2 - O sistema notifica o usuário que o meio de pagamento foi editado com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de meios de pagamento. 3 - O usuário clica no botão de editar um meio de pagamento. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema edita um meio de pagamento cadastrado. 8 - O sistema notifica o usuário que o meio de pagamento foi editado com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há um meio de pagamento cadastrado com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há um meio de pagamento cadastrado com as mesmas informações.

Casos de uso: Remover meio de pagamento
Descrição: Fluxo do usuário para remover um meio de pagamento.
Atores: Cliente

Pré-condições: 1 - O usuário deve estar logado. 2 - O usuário deve possuir ao menos dois meios de pagamento cadastrados.
Pós-condições: 1 - O sistema remove um meio de pagamento cadastrado. 2- O sistema notifica o usuário que meio de pagamento foi removido com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de meios de pagamento. 3 - O usuário clica no botão de remover um meio de pagamento. 5 - O usuário confirma a remoção do meio de pagamento. 6 - O sistema verifica se há mais de um meio de pagamento cadastrado. 7 - O sistema remove um meio de pagamento cadastrado. 8 - O sistema notifica o usuário que o meio de pagamento foi removido com sucesso.
Fluxo alternativo: 6a - O sistema identifica que o usuário possui apenas um meio de pagamento cadastrado 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que deve haver pelo menos um meio de pagamento cadastrado.

Casos de uso: Editar dados cadastrais
Descrição: Fluxo do usuário para editar dados cadastrais.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas.
Pós-condições: 1 - O sistema atualiza os dados cadastrais. 2 - O sistema notifica o usuário que os dados cadastrais foram atualizados com sucesso.
Fluxo principal:

1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de dados cadastrais. 3 - O usuário clica no botão de editar dados cadastrais. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema edita os dados cadastrais. 8 - O sistema notifica o usuário que os dados cadastrais foram atualizados com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos.

Casos de uso: Visualizar histórico de compras
Descrição: Fluxo do usuário para visualizar o histórico de compras.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado. 2 - O usuário deverá ter finalizado ao menos uma compra.
Pós-condições: 1 - O sistema exibe o histórico de compras.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de compras. 3 - O sistema exibe o histórico de compras.
Fluxo alternativo: 3a - O sistema identifica que nenhuma compra foi realizada. 3a.1 - O sistema exibe a mensagem de que nenhuma compra foi realizada.

Casos de uso: Cancelar compra
Descrição: Fluxo do usuário para cancelar uma compra.
Atores: Cliente
Pré-condições: 1 - O usuário deve estar logado.

2 - O usuário deverá ter finalizado ao menos uma compra.
Pós-condições: 1 - O sistema cancela uma compra. 2 - O sistema atualiza o estoque dos produtos. 3 - O sistema atualiza o histórico de compras. 4 - O sistema notifica que a compra foi cancelada com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de compras. 3 - O sistema exibe o histórico de compras. 4 - O usuário clica em cancelar compra. 5 - O usuário confirma que deseja cancelar a compra. 6 - O sistema cancela a compra. 7 - O sistema atualiza o estoque dos produtos. 8 - O sistema atualiza o histórico de compras. 9 - O sistema notifica que a compra foi cancelada com sucesso.
Fluxo alternativo: 3a - O sistema identifica que nenhuma compra foi realizada. 3a.1 - O sistema exibe a mensagem de que nenhuma compra foi realizada. 5a - O usuário desiste de cancelar a compra. 5a.1 - A operação de cancelamento não é realizada.

Casos de uso: Pesquisar produto
Descrição: Fluxo do usuário para pesquisar um produto.
Atores: Cliente
Pré-condições: -
Pós-condições: 1 - O sistema direciona o usuário para a uma página de categorias com prateleiras que sejam condizentes com os parâmetros digitados na busca.
Fluxo principal: 1 - O usuário clica na barra de busca. 2 - O usuário digita um termo para a busca. 3 - O sistema direciona o usuário para uma página de categorias com

prateleiras que sejam condizentes com os parâmetros digitados na busca.
Fluxo alternativo: 3a - O sistema não encontra produtos baseados no parâmetro digitado na busca. 3a.1 - O sistema exibe uma mensagem dizendo ao usuário que não foram encontrados produtos.

Casos de uso: Filtrar produto
Descrição: Fluxo do usuário para filtrar produtos baseado em parâmetros selecionados por ele.
Atores: Cliente
Pré-condições: 1 - O usuário deve acessar a página de categoria.
Pós-condições: 1 - O sistema exibe prateleiras que sejam condizentes com os parâmetros filtrados.
Fluxo principal: 1 - O usuário acessa uma página de categorias via menu ou barra de busca. 2 - O usuário seleciona os filtros desejados. 3 - O sistema exibe prateleiras que sejam condizentes com os parâmetros filtrados
Fluxo alternativo: 3a - O sistema não encontra produtos baseados nos parâmetros selecionados nos filtros. 3a.1 - O sistema exibe uma mensagem dizendo ao usuário que não foram encontrados produtos.

Casos de uso: Visualizar informações do produto
Descrição: Fluxo do usuário para visualizar as informações de um produto
Atores: Cliente
Pré-condições: 1 - O produto precisa estar cadastrado

Pós-condições: 1 - O sistema exibe uma página de produto com todas as informações dele.
Fluxo principal: 1 - O usuário clica em um produto. 2 - O sistema exibe uma página de produto com todas as informações dele.

Casos de uso: Cadastrar categoria
Descrição: Fluxo do administrador para cadastrar uma categoria
Atores: Administrador
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - A categoria cadastrada não pode possuir as mesmas informações de uma categoria já cadastrada.
Pós-condições: 1 - O sistema cadastra uma categoria. 2 - O sistema notifica o usuário que a categoria foi cadastrada com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de categorias. 3 - O usuário clica no botão de cadastrar categoria. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema cadastra uma categoria. 8 - O sistema notifica o usuário que a categoria foi cadastrada com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há uma categoria cadastrada com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há uma categoria cadastrada com as mesmas informações.

Casos de uso: Editar categoria

Descrição: Fluxo do administrador para editar uma categoria
Atores: Administrador
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O sistema deve possuir ao menos uma categoria cadastrada.
Pós-condições: 1 - O sistema atualiza uma categoria. 2 - O sistema notifica o usuário que a categoria foi atualizada com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de categorias. 3 - O usuário clica no botão de editar categoria. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema atualiza uma categoria. 8 - O sistema notifica o usuário que a categoria foi atualizada com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há uma categoria cadastrada com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há uma categoria cadastrada com as mesmas informações.

Casos de uso: Deletar categoria
Descrição: Fluxo do administrador para deletar uma categoria
Atores: Administrador
Pré-condições: 1 - O usuário deve estar logado. 2 - O sistema deve possuir ao menos uma categoria cadastrada. 3 - Nenhum produto pode estar associado a categoria a ser deletada.

Pós-condições: 1 - O sistema deleta uma categoria. 2 - O sistema notifica o usuário que a categoria foi deletada com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de categorias. 3 - O usuário clica no botão de deletar categoria. 4 - O usuário confirma que deseja deletar a categoria. 5 - O sistema verifica se há algum produto associado à categoria a ser deletada. 6 - O sistema deleta a categoria. 7 - O sistema notifica o usuário que a categoria foi deletada com sucesso.
Fluxo alternativo: 4a - O usuário desiste de deletar a categoria. 4a.1 - A operação de deletar a categoria não é realizada. 5a - O sistema encontra um produto associado a categoria a ser deletada. 5a.1 - O sistema não realiza a operação 5a.2 - O sistema notifica o usuário que nenhum produto pode estar associado à categoria a ser deletada.

Casos de uso: Cadastrar produto
Descrição: Fluxo do administrador para cadastrar um produto
Atores: Administrador
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O produto cadastrado não pode possuir as mesmas informações de um produto já cadastrada. 4 - O sistema deve possuir ao menos uma categoria cadastrada.
Pós-condições: 1 - O sistema cadastra um produto. 2 - O sistema notifica o usuário que o produto foi cadastrado com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de produto. 3 - O usuário clica no botão de cadastrar produto. 4 - O usuário preenche o formulário.

5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema cadastra um produto. 8 - O sistema notifica o usuário que o produto foi cadastrado com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há um produto cadastrado com as mesmas informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há um produto cadastrado com as mesmas informações.

Casos de uso: Editar produto
Descrição: Fluxo do administrador para editar um produto
Atores: Administrador
Pré-condições: 1 - O usuário deve estar logado. 2 - As informações preenchidas no formulário devem estar corretas. 3 - O sistema deve possuir ao menos um produto cadastrado.
Pós-condições: 1 - O sistema atualiza um produto. 2 - O sistema notifica o usuário que o produto foi atualizado com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de produto. 3 - O usuário clica no botão de editar produto. 4 - O usuário preenche o formulário. 5 - O usuário confirma o envio dos dados. 6 - O sistema valida os dados digitados no formulário. 7 - O sistema atualiza um produto. 8 - O sistema notifica o usuário que o produto foi atualizado com sucesso.
Fluxo alternativo: 6a - O sistema identifica que os dados preenchidos estão inválidos. 6a.1 - O sistema não realiza a operação. 6a.2 - O sistema notifica o usuário que os dados estão incorretos. 6b - O sistema identifica que já há um produto cadastrado com as mesmas

informações. 6b.1 - O sistema não realiza a operação. 6b.2 - O sistema notifica o usuário que já há um produto cadastrado com as mesmas informações.

Casos de uso: Deletar produto
Descrição: Fluxo do administrador para deletar um produto
Atores: Administrador
Pré-condições: 1 - O usuário deve estar logado. 2 - O sistema deve possuir ao menos um produto cadastrado.
Pós-condições: 1 - O sistema deleta um produto. 2 - O sistema notifica o usuário que a produto foi deletado com sucesso.
Fluxo principal: 1 - O usuário clica no botão da área do usuário. 2 - O usuário vai até seção de produto. 3 - O usuário clica no botão de deletar produto. 4 - O usuário confirma que deseja deletar o produto. 5 - O sistema deleta o produto. 6 - O sistema notifica o usuário que o produto foi deletado com sucesso.
Fluxo alternativo: 4a - O usuário desiste de deletar o produto. 4a.1 - A operação de deletar o produto não é realizada.

APÊNDICE C – WIREFRAMES

Figura 14 - Wireframe da página inicial

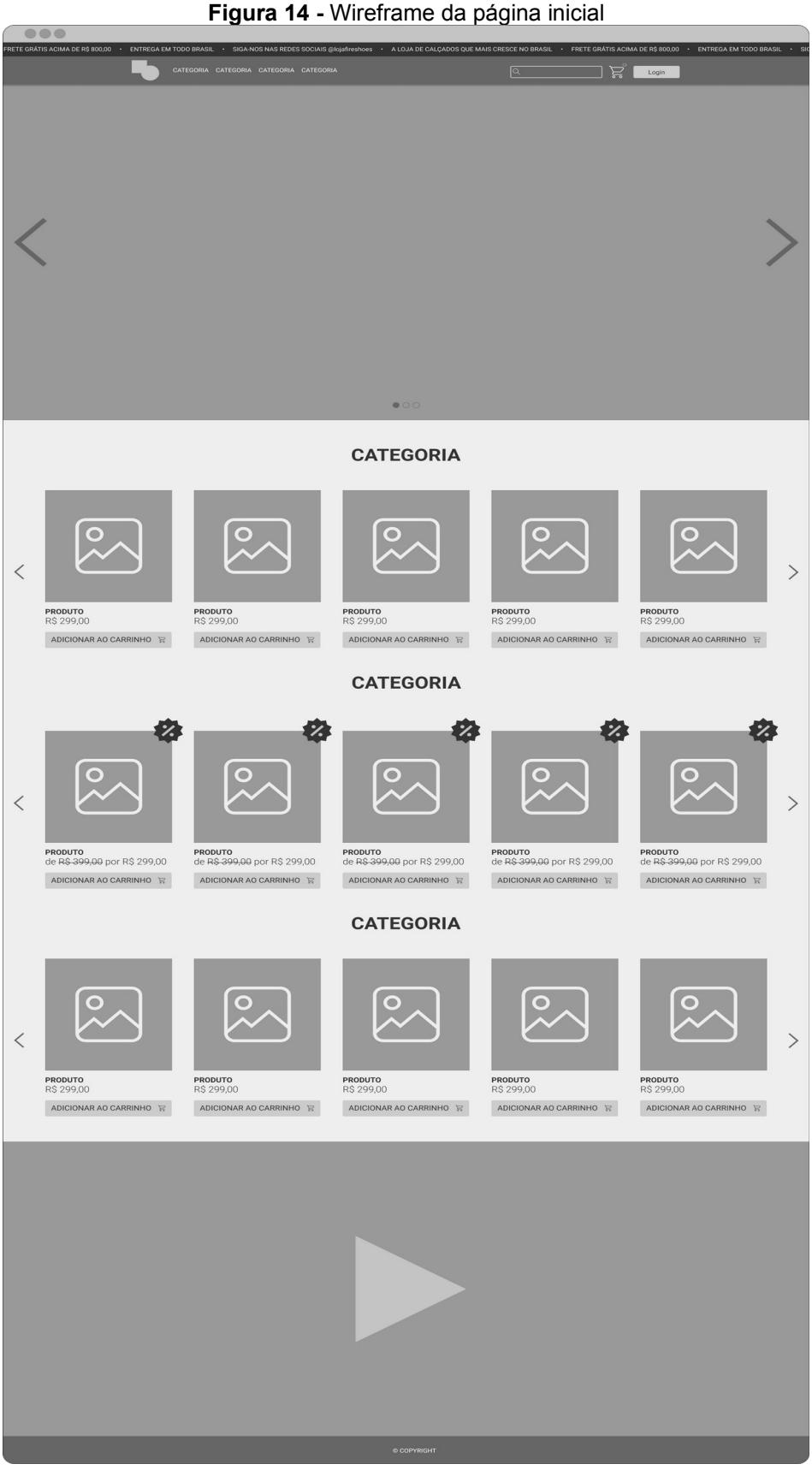


Figura 15 - Wireframe da área de login

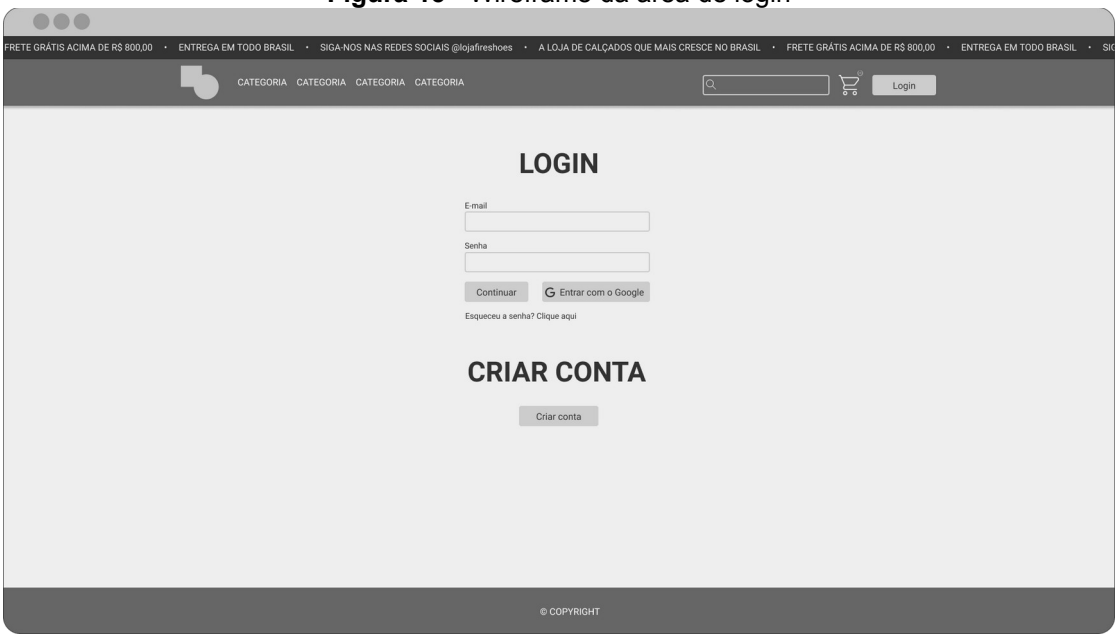


Figura 16 - Figura 15 - Wireframe da área de criação de conta

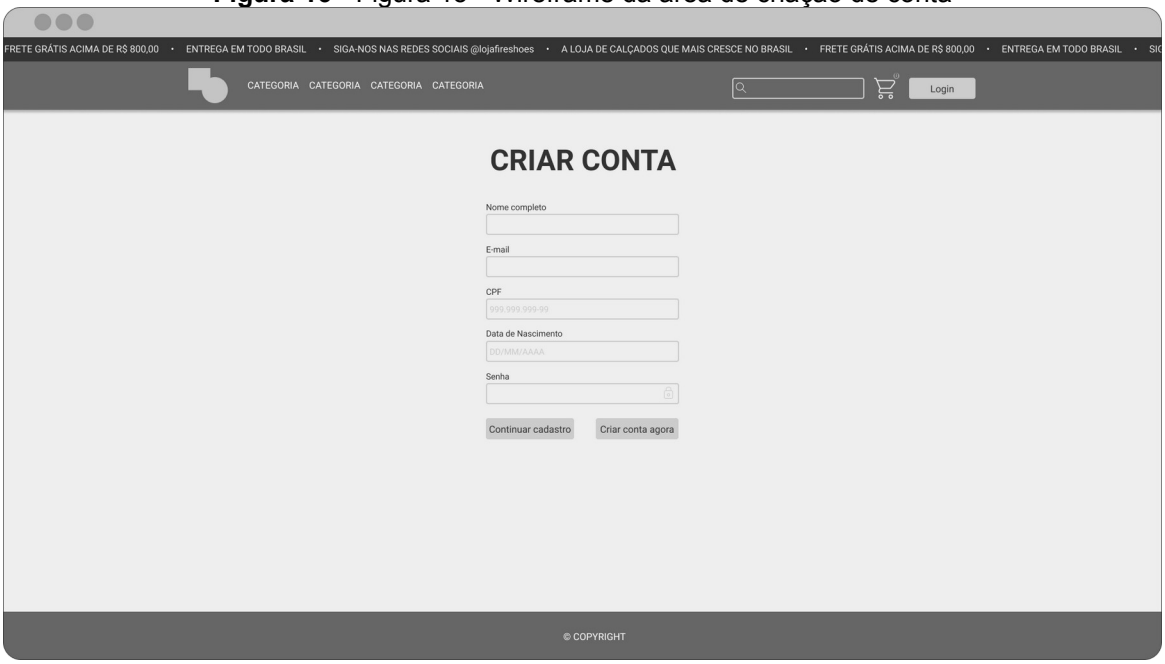


Figura 17 - Wireframe do comportamento de carrinho vazio

Figura 18 - Wireframe do comportamento de carrinho cheio

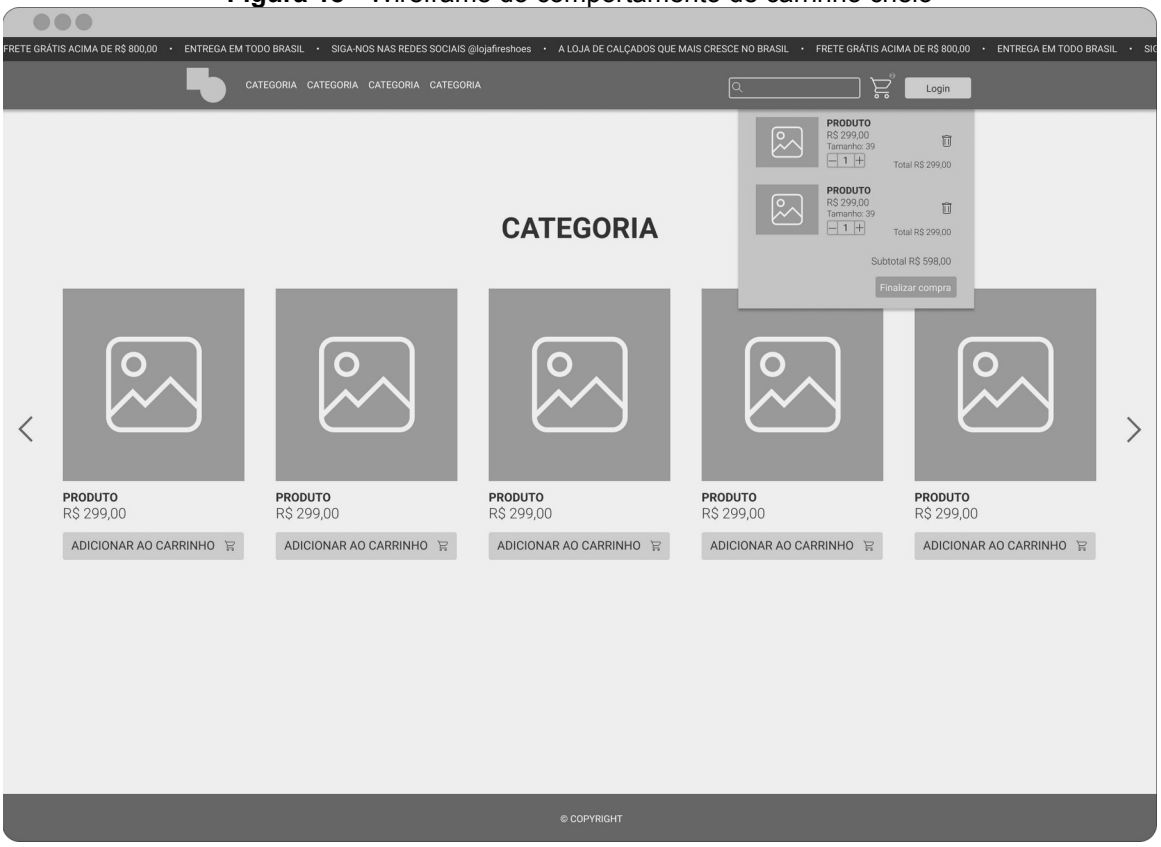


Figura 19 - Wireframe do fluxo de finalização de pedido. Etapa do carrinho

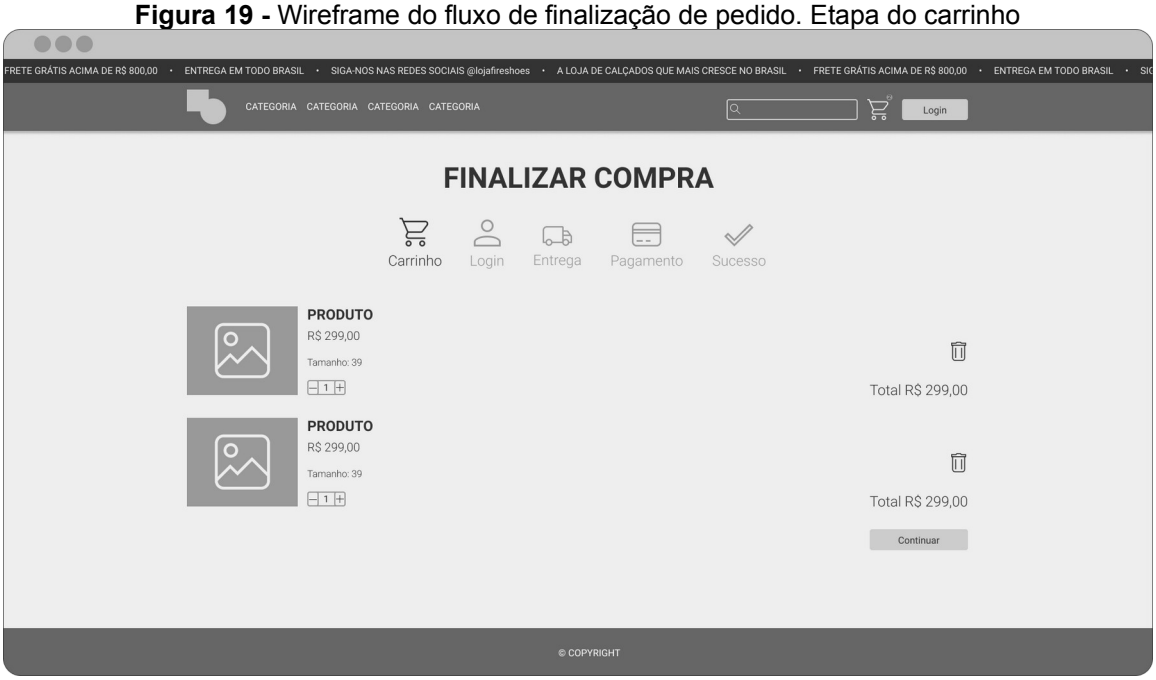


Figura 20 - Wireframe do fluxo de finalização de pedido. Etapa da entrega

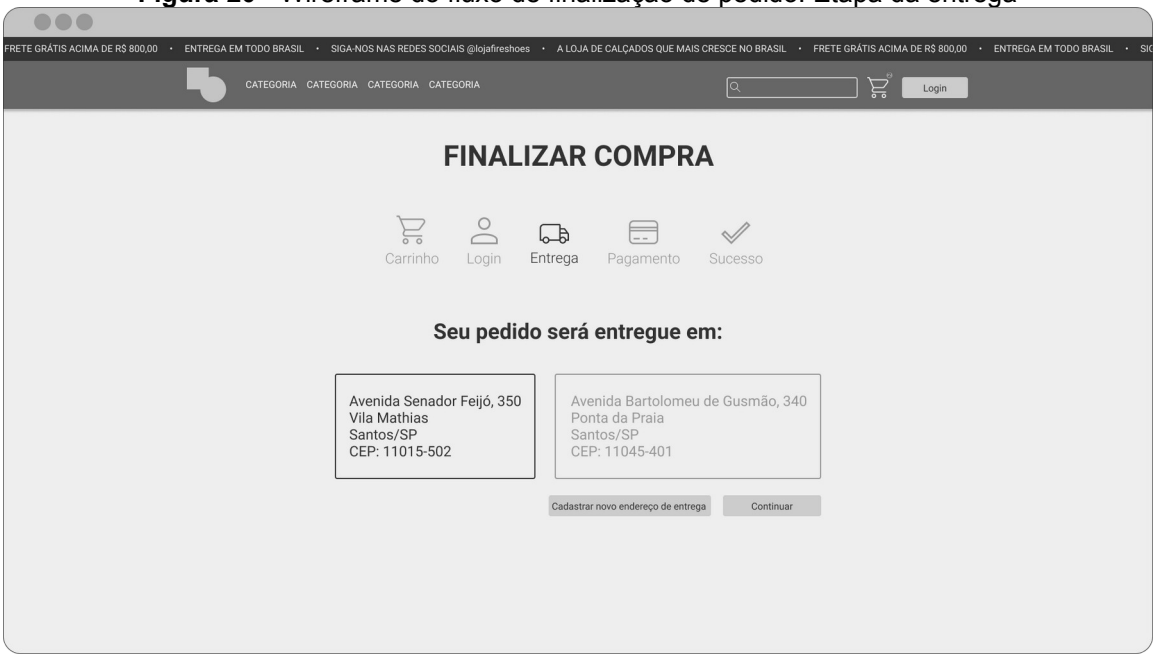


Figura 21 - Wireframe do fluxo de finalização de pedido. Etapa do pagamento

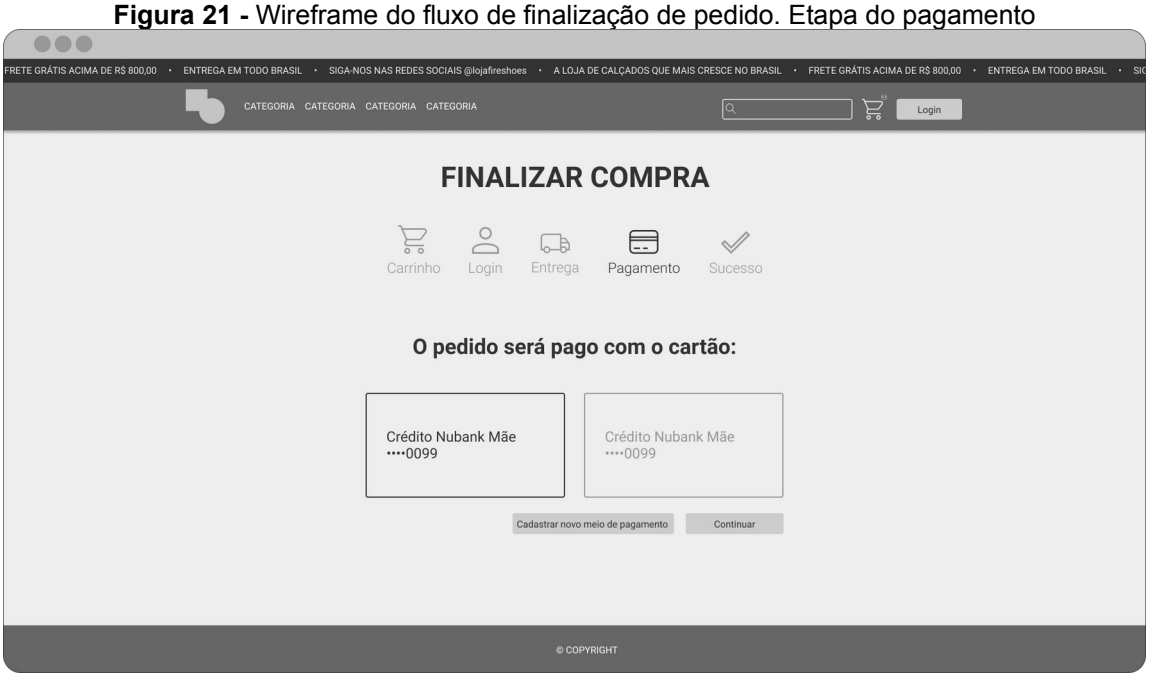


Figura 22 - Wireframe do fluxo de finalização de pedido. Etapa de confirmação

Figura 23 - Wireframe da página de produto

Figura 24 - Wireframe da página de categoria

Figura 25 - Wireframe da página usuário. Área de endereço de entrega

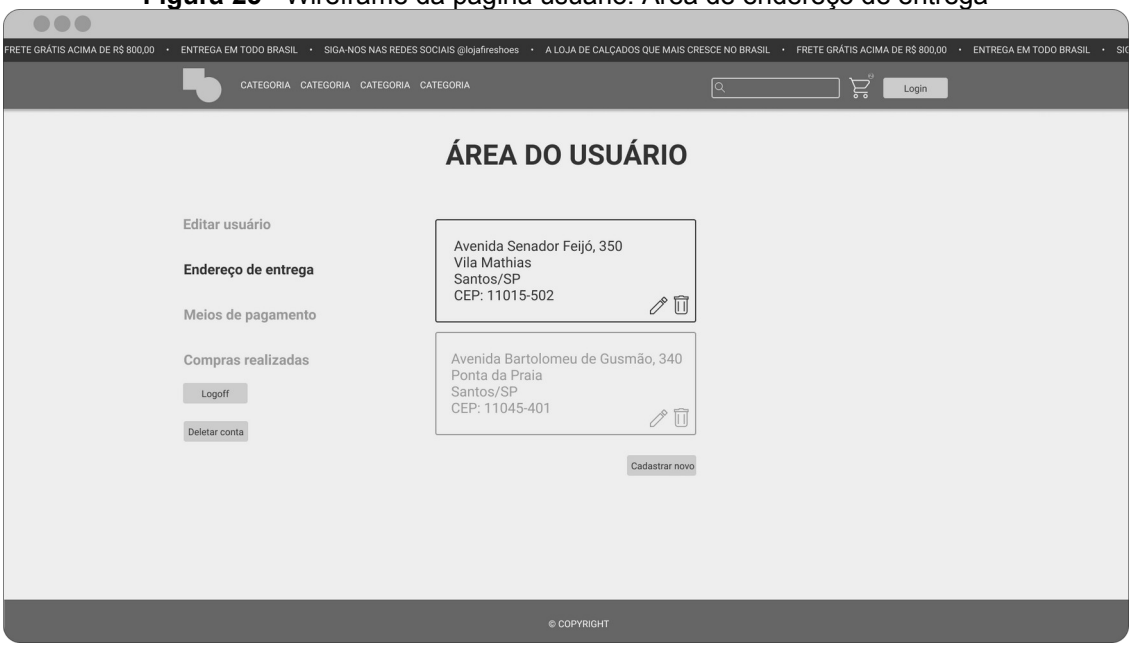


Figura 26 - Wireframe da página usuário. Modal de edição de endereço de entrega

Figura 27 - Wireframe da página usuário. Área de meio de pagamento

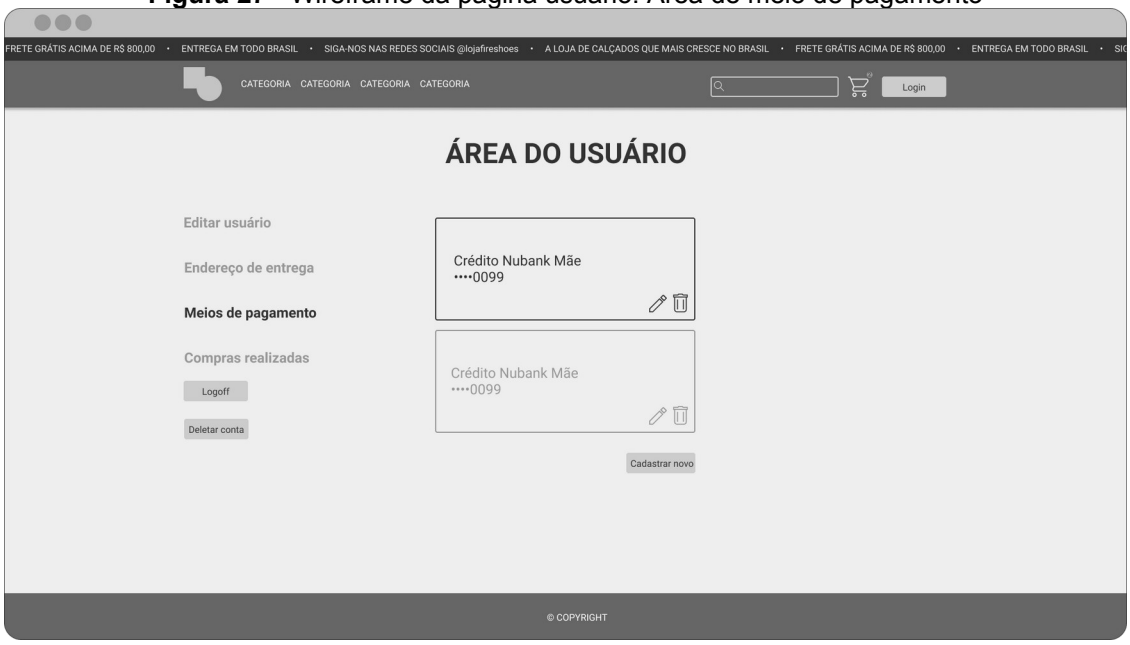


Figura 28 - Wireframe da página usuário. Modal de edição de meio de pagamento

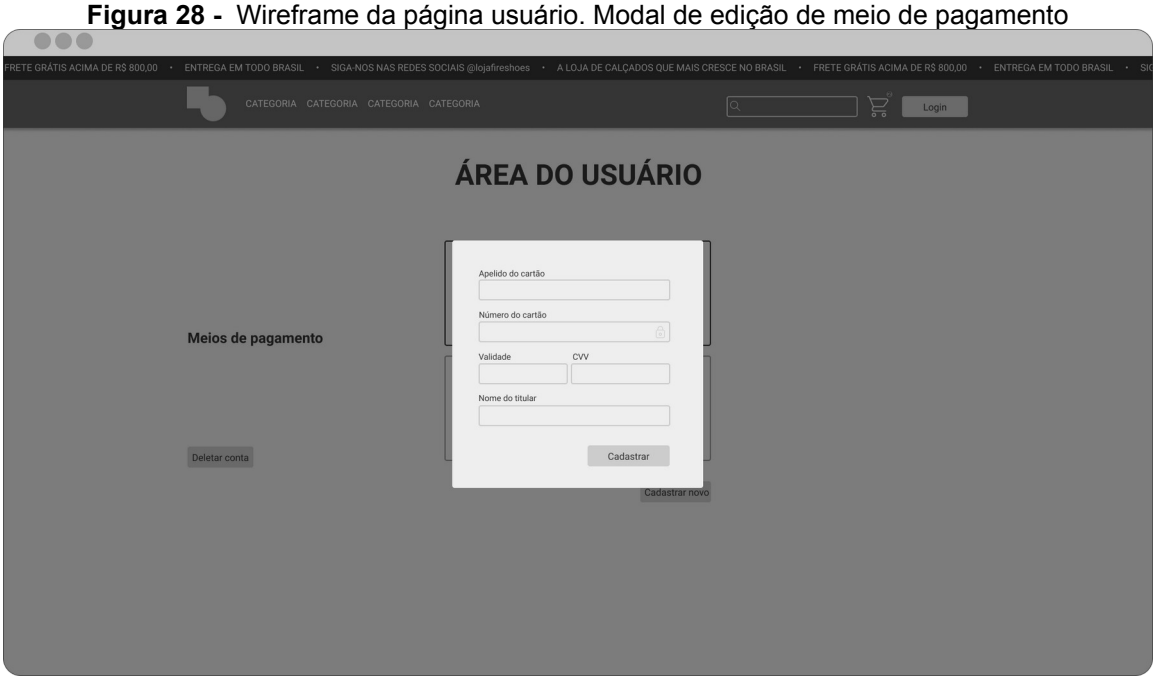
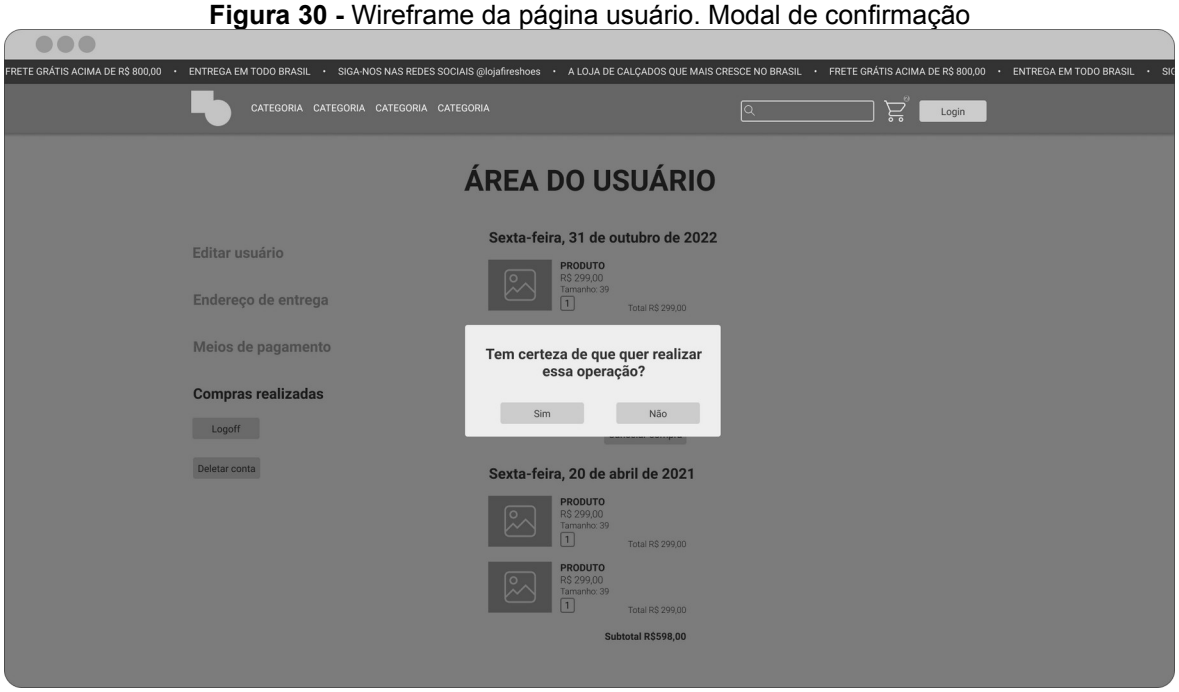


Figura 29 - Wireframe da página usuário. Área de compras realizadas

Figura 30 - Wireframe da página usuário. Modal de confirmação



APÊNDICE D – MAPAS MENTAIS

Figura 31 - Legenda dos mapas mentais

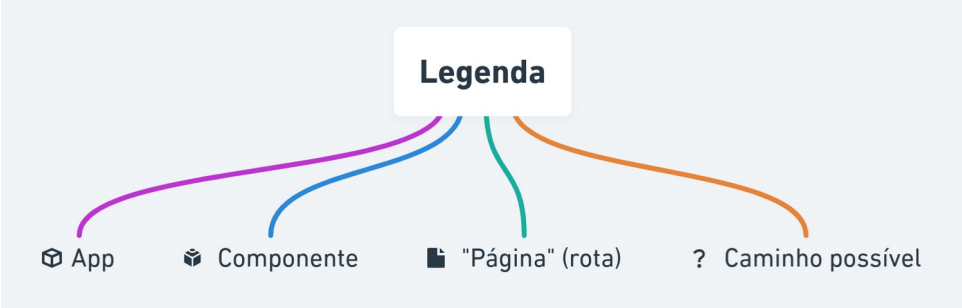
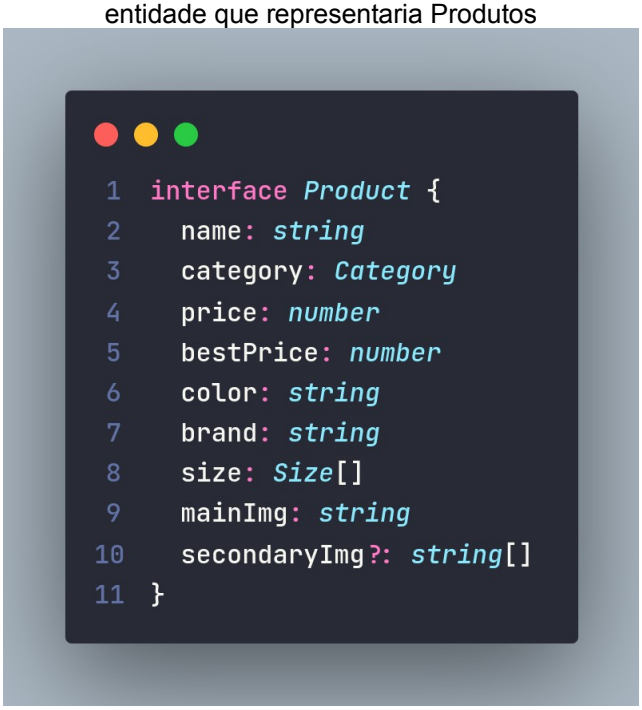


Figura 32 - Mapa mental para esboço da estrutura da aplicação

Figura 33 - Mapa mental para esboço das rotas

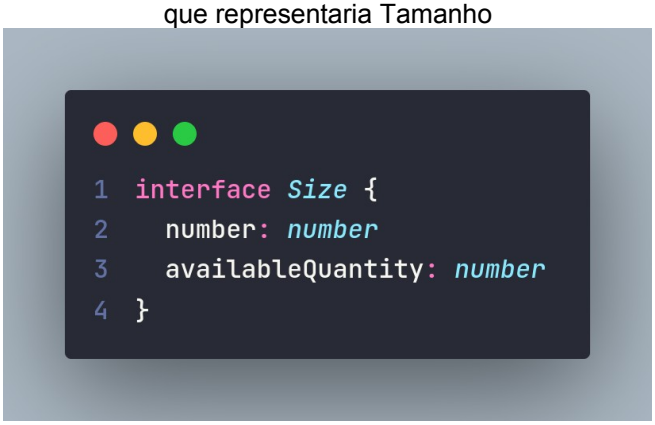
APÊNDICE E

Figura 34 - Planejamento da modelagem da entidade que representaria Produtos



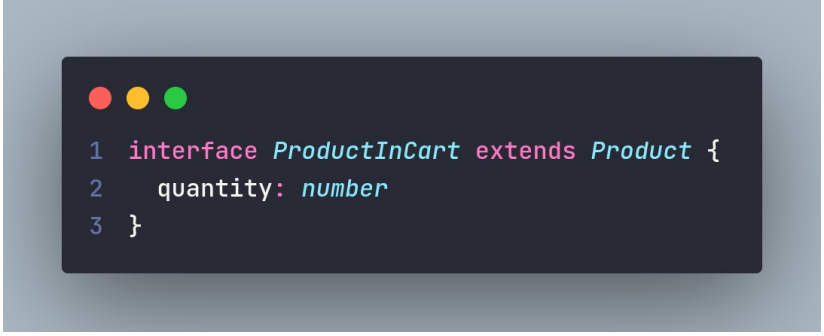
```
interface Product {
  name: string
  category: Category
  price: number
  bestPrice: number
  color: string
  brand: string
  size: Size[]
  mainImg: string
  secondaryImg?: string[]
}
```

Figura 35 - Planejamento da modelagem da entidade que representaria Tamanho



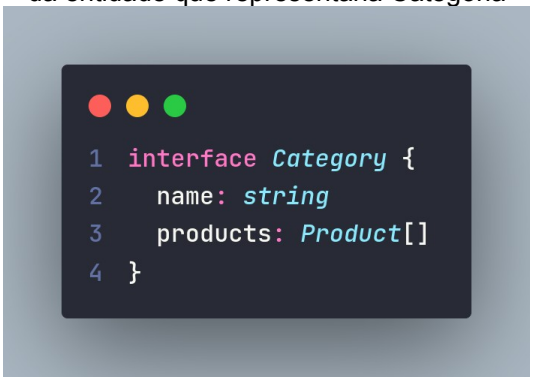
```
interface Size {
  number: number
  availableQuantity: number
}
```

Figura 36 - Planejamento da modelagem da entidade que representaria Produto no Carrinho



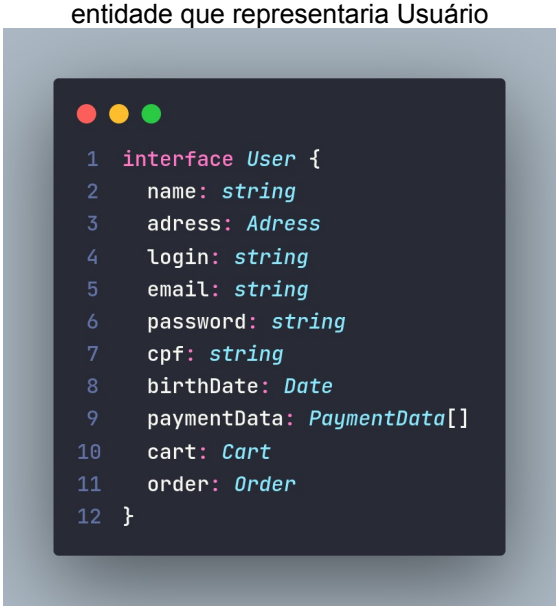
```
interface ProductInCart extends Product {
  quantity: number
}
```

Figura 37 - Planejamento da modelagem da entidade que representaria Categoria



```
interface Category {
  name: string
  products: Product[]
}
```

Figura 38 - Planejamento da modelagem da entidade que representaria Usuário

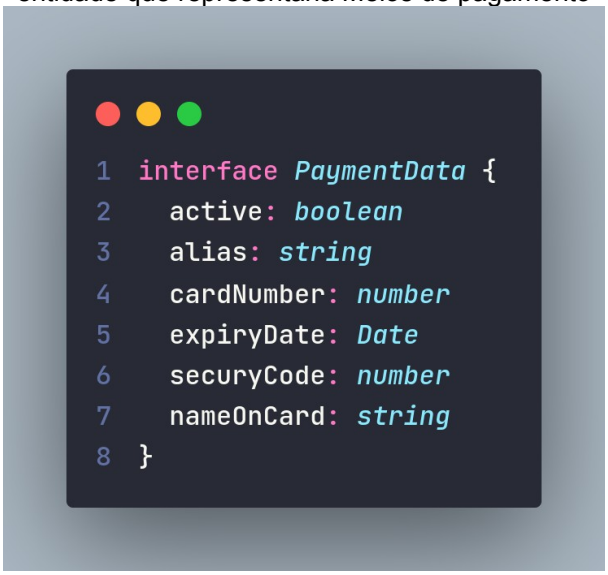


```
interface User {
  name: string
  adress: Adress
  login: string
  email: string
  password: string
  cpf: string
  birthDate: Date
  paymentData: PaymentData[]
  cart: Cart
  order: Order
}
```

Figura 39 - Planejamento da modelagem da entidade que representaria Endereço

```
interface Adress {
  active: boolean
  street: string
  streetNumber: string
  complement?: string
  zipCode: string
  district: string
  city: string
  state: string
}
```

Figura 40 - Planejamento da modelagem da entidade que representaria Meios de pagamento

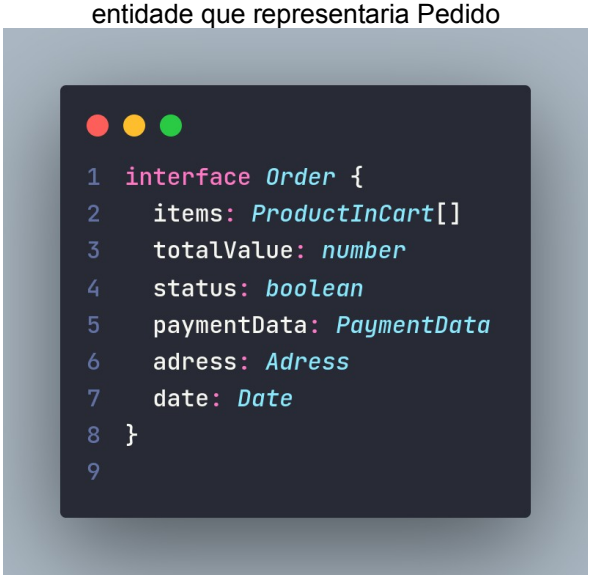


```
interface PaymentData {
  active: boolean
  alias: string
  cardNumber: number
  expiryDate: Date
  securyCode: number
  nameOnCard: string
}
```

Figura 41 - Planejamento da modelagem da entidade que representaria Carrinho

```
interface Cart {
  products: Product[]
  totalValue: number
}
```

Figura 42 - Planejamento da modelagem da entidade que representaria Pedido



```
interface Order {
  items: ProductInCart[]
  totalValue: number
  status: boolean
  paymentData: PaymentData
  adress: Adress
  date: Date
}

```