Turma do Café

Especificação de Software com Disciplined Agile Delivery

Projeto de Bloco: Engenharia Disciplinada de Softwares

Instituto Infnet - ESTI - Engenharia de Software

Christian Rocha

Prof. Armênio Torres Santiago Cardoso

16/10/2023

Índice

Construção

Testes

Implantação

Extra

Requisitos

Visão do Projeto

Escopo

O projeto Turma do Café (referenciado como “projeto” neste documento) visa criar um e-commerce de café. Conectando apreciadores de café que buscam grãos de qualidade, com torras diferentes e sabores diversificados, com a venda dos melhores grãos de cafés do mundo em uma só loja virtual com total segurança da qualidade e autenticidade de cada produto, experiência de compra diferenciada e simplificada.

Este projeto se insere no mercado de café, um setor em crescimento constante, com uma base sólida e crescente de apreciadores de café em busca de novas experiências. O público-alvo deste projeto são os amantes de café que buscam experiências autênticas e variadas, podendo ser consumidores domésticos que possuem os aparatos de barista em casa, como consumidores que frequentam coffee shops ou participam de eventos relacionados ao tema. No geral, este mercado do café abrange diversos nichos, públicos e classes sociais.

Algumas limitações do projeto são o curto prazo de seis meses para desenvolvimento da plataforma como um todo, orçamento inicial reduzido para manter uma infraestrutura em nuvem pública ou adquirir um servidor on premise, escolha e negociação de taxa com empresas para pagamento online, por conta do baixo volume de vendas iniciais e orçamento baixo para marketing inicial.

As partes interessadas são o dono do negócio, a equipe de desenvolvimento, posteriormente logística e clientes.

Existe um computador com recursos intermediários para desenvolvimento, porém para de fato executar o site do ecommerce, será necessário levá-lo para nuvem pública ou servidor on premise. Conta-se também para o projeto com as plataformas gratuitas de inteligência artificial para as mais diversas tarefas.

Toda a parte dos interessados são conectadas, ou seja, um atraso no desenvolvimento irá consequentemente atrasar o lançamento do site, o que por sua vez pode deixar de aproveitar alguma data comemorativa ou determinado período maior de vendas de um ano.

Da mesma forma, exteriormente uma mudança climática pode afetar de forma direta o negócio, pois se uma safra de café é perdida por conta de qualquer tipo de imprevisto, seja climático, logística isso influenciará diretamente na venda para o consumidor final.

Funcionalidades principais e restrições

O planejamento inicial para a primeira versão do projeto envolve a criação de um catálogo de produtos com grãos de café disponíveis para venda imediata. Além disso, terá um carrinho de compras para que os clientes possam adicionar produtos e concluir suas compras. Também haverá uma página para verificação de pedidos e outras informações institucionais.

É importante ressaltar que não foi incluída a autenticação nesta fase inicial. Portanto, o cliente precisará preencher manualmente suas informações pessoais e de pagamento. Além disso, a parte de pagamento foi abstraída no desenvolvimento local, pois ainda não existe um gateway de pagamento implementado e disponível.

Durante o desenvolvimento, enfrentam-se restrições de tempo, como prazos e possíveis impasses no código e nos testes. Além disso, considera-se restrições orçamentárias, como limitações no orçamento. Por fim, a usabilidade e compatibilidade com dispositivos móveis serão fatores cruciais para o sucesso do projeto.

Solução desejada

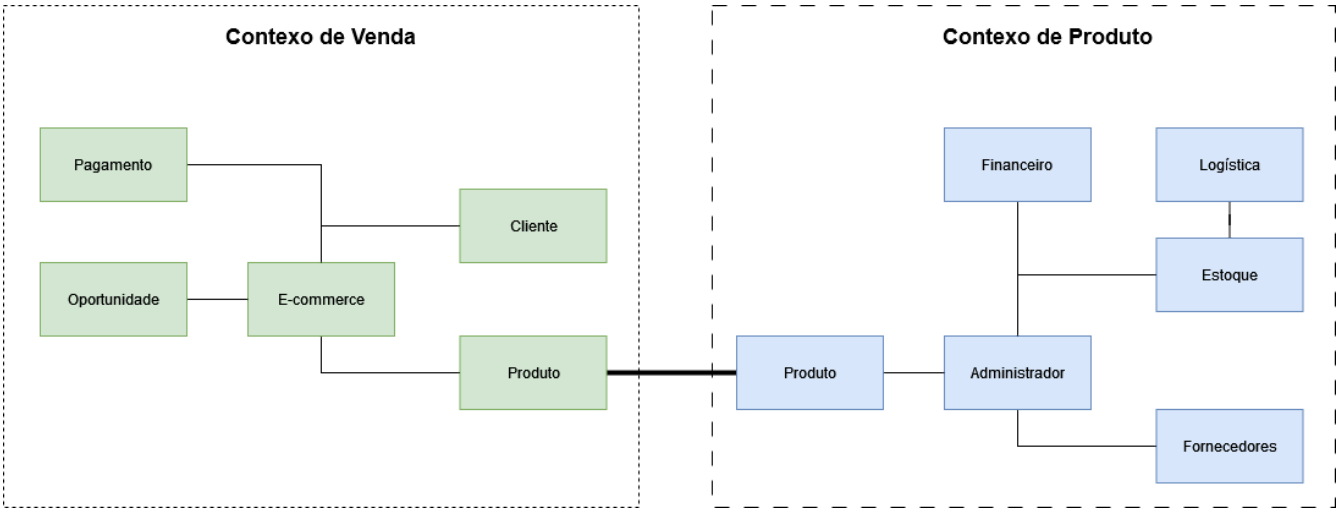
O objetivo geral do projeto é melhorar o e-commerce de café já explicado anteriormente, estabelecendo um vínculo sólido e direto com o comprador e apreciador de café. Espera-se que com a implementação da loja virtual os compradores possam ter acesso a grãos de café diversificados, com entrega rápida e compra simples e direta. Que possa-se o e-commerce se tornar referência na venda dos grãos de cafés mais renomados e exclusivos.

Matriz de Requisitos Funcionais

Para uma boa implementação e experiência de usuário, espera-se que os requisitos abaixo sejam cumpridos:

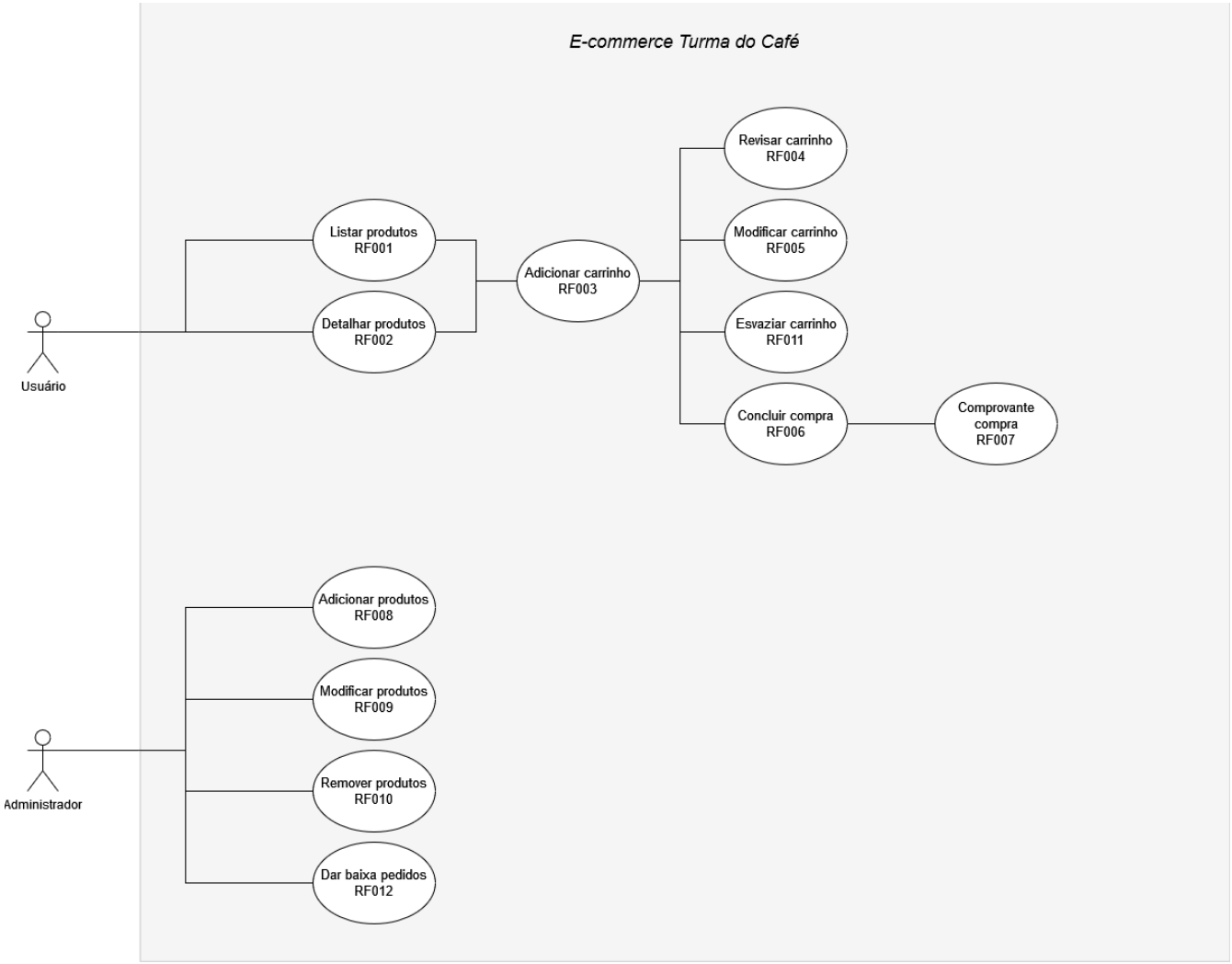
ID	Descrição	Escala	Prioridade
RF001	O sistema deve permitir ao usuário navegar pelos produtos disponíveis no e-commerce.	2	Essencial
RF002	O sistema deve exibir informações detalhadas sobre cada produto, incluindo preço e descrição.	1	Importante
RF003	O sistema deve permitir aos usuários adicionar produtos ao carrinho de compras.	2	Essencial
RF004	O sistema deve permitir aos usuários revisar o conteúdo do carrinho de compras.	1	Importante
RF005	O sistema deve permitir aos usuários modificar o conteúdo do carrinho de compras.	2	Importante
RF006	O sistema deve permitir aos usuários concluir a compra do conteúdo do carrinho de compras.	4	Essencial
RF007	O sistema deve gerar um comprovante de compra após a conclusão do pagamento.	2	Importante
RF008	O sistema deve permitir aos administradores adicionar produtos ao catálogo.	2	Essencial
RF009	O sistema deve permitir aos administradores modificar produtos do catálogo.	2	Essencial
RF010	O sistema deve permitir aos administradores remover produtos do catálogo.	2	Essencial
RF011	O sistema deve permitir o usuário esvaziar o carrinho de compras	2	Desejável
RF012	O sistema deve permitir o administrador dar baixa em pedidos	2	Desejável

Relação de Domínios de Negócio



Análise

Diagrama de Casos de Uso



Descrição dos Casos de Uso do MVP

RF001 - *O sistema deve permitir ao usuário navegar pelos produtos disponíveis no e-commerce*.

RF002 - *O sistema deve exibir informações detalhadas sobre cada produto, incluindo preço e descrição.*

Descrição

O administrador ou cliente visita a página de detalhamento de um produto.

Atores

Administrador e cliente.

Pré-condições

Precisa ter ao menos um produto cadastrado no banco de dados.

Pós-condições

O produto será detalhado se for carregado corretamente do banco de dados.

Fluxo principal

1. Administrador ou cliente acessam o detalhamento do produto específico.
2. O sistema busca pelo ID do produto no banco de dados.
3. O sistema retorna as informações e exibe a página do detalhamento do mesmo.

Fluxo alternativo

Passo 3 - Se for administrador.

1. O sistema mostra as opções para editar ou deletar o produto.
2. Caso opção seja editar:
   2.1. O sistema redireciona o administrador para a página de edição.
3. Caso opção seja deletar:
   3.1. O sistema redireciona o administrador para a página de deleção.

RF008 - *O sistema deve permitir aos administradores adicionar produtos ao catálogo.*

Descrição

O administrador realiza o processo para adicionar um novo produto.

Atores

Administrador.

Pré-condições

O usuário precisa ser administrador.

Pós-condições

O produto será adicionado se não existir outro produto igual e os dados forem válidos.

Fluxo principal

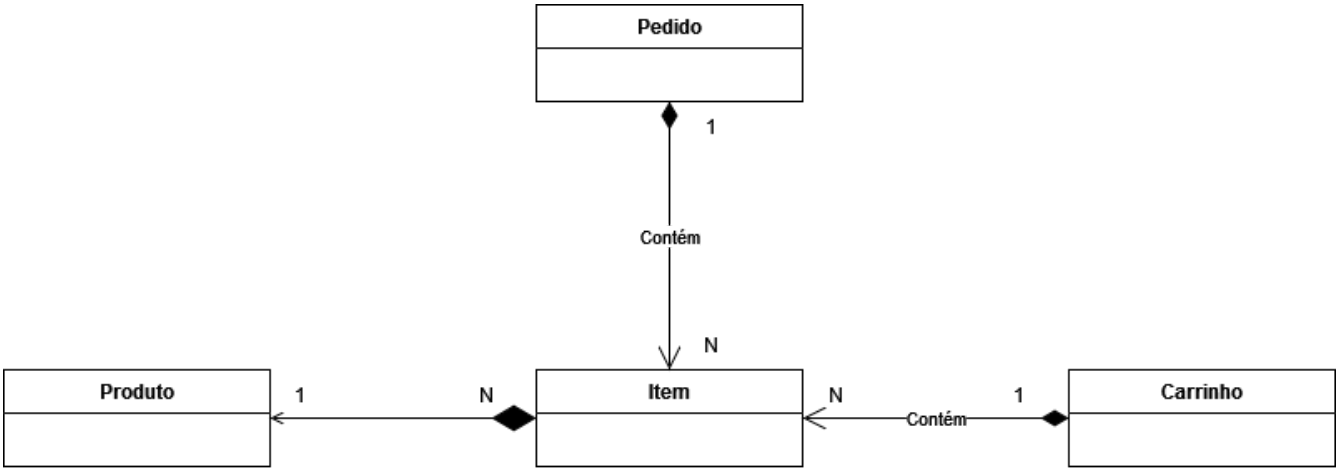
4. Dados do produto são registrados no formulário pelo administrador.
5. Administrador solicita enviar o formulário.
6. O sistema adiciona o novo produto ao banco de dados.
7. O sistema redireciona o administrador para a página de produtos.
8. O sistema lista todos os produtos registrados no banco de dados.

Fluxo alternativo

Passo 2 - Se o formulário estiver incompleto ou com dado em formato incorreto em um ou mais campos.

1. O sistema bloqueia o envio.
2. O sistema mostra mensagem de erro no campo que necessita correção.
3. O administrador corrige o campo incorreto.
4. O administrador submete novamente o formulário:
    - 4.1. Caso ainda tenha inconsistência.
        - 4.1.1. O sistema retorna para o passo 1 do fluxo alternativo.
    - 4.2. Caso tenha um formulário válido.
        - 4.2.1. Retorno ao Passo 3 do fluxo principal.

Diagrama de Domínio



Relacionamentos

Produto - Item

Um Item somente séra criado ao adicionar um Produto ao Carrinho, o Item é uma agregação por composição, onde um Produto pode estar associado a um ou mais Item, da mesma forma que um ou mais Item pode estar associado a um único produto.

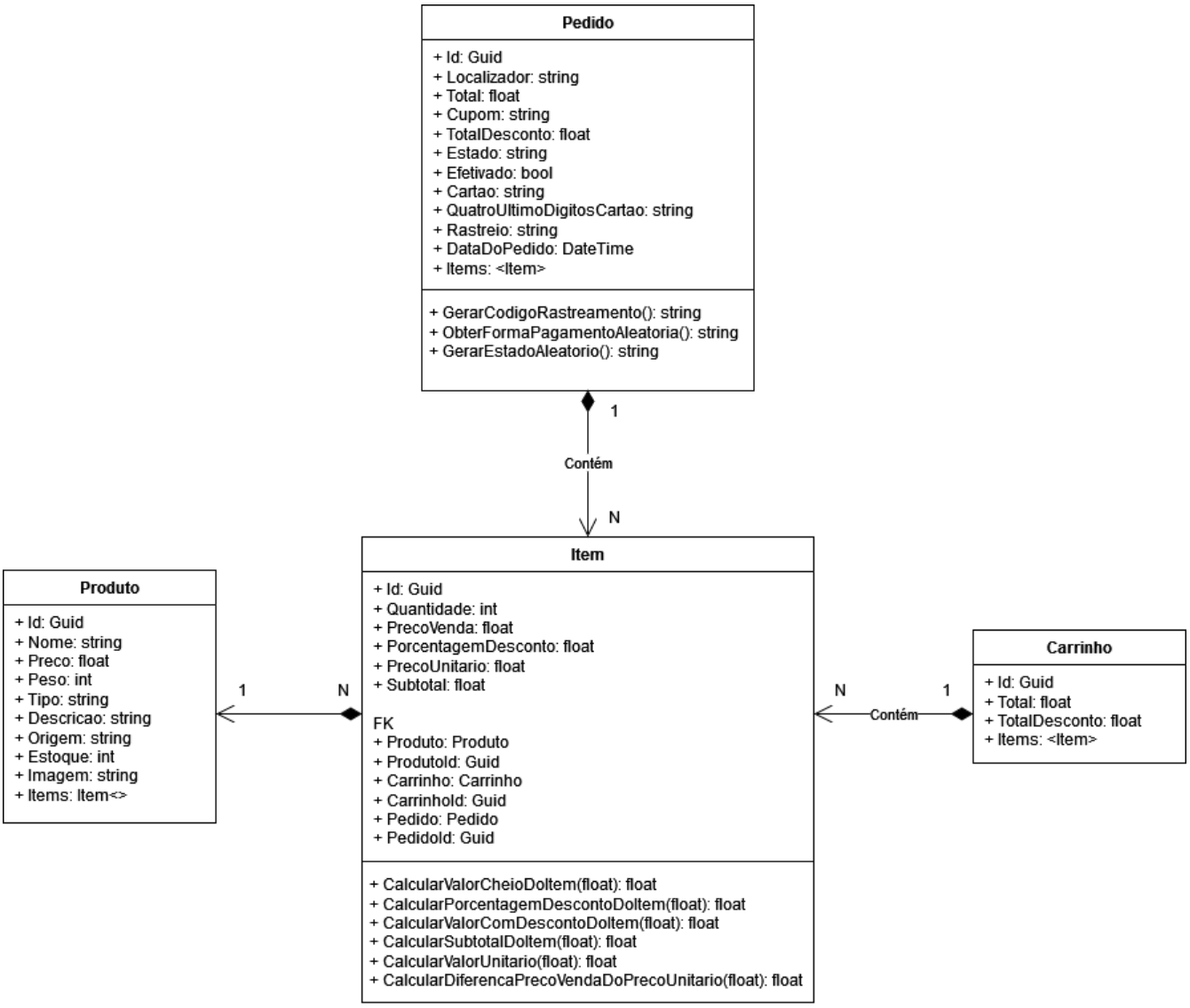
Item - Carrinho

Um Carrinho pode conter um ou mais Item, da mesma forma que um ou mais Item pode estar contido em um único Carrinho. Foi abstraído o cenário do Carrinho estar vazio, pois o Carrinho só é criado quando um Produto é adicionado ao Carrinho, assim ele se torna Item e de fato estará presente no Carrinho.

Item - Pedido

Um Pedido pode conter um ou mais Item, da mesma forma que um ou mais Item pode estar contido em um único Pedido. O Pedido deve ter no mínimo um Item para ser gerado.

Diagrama de Classes

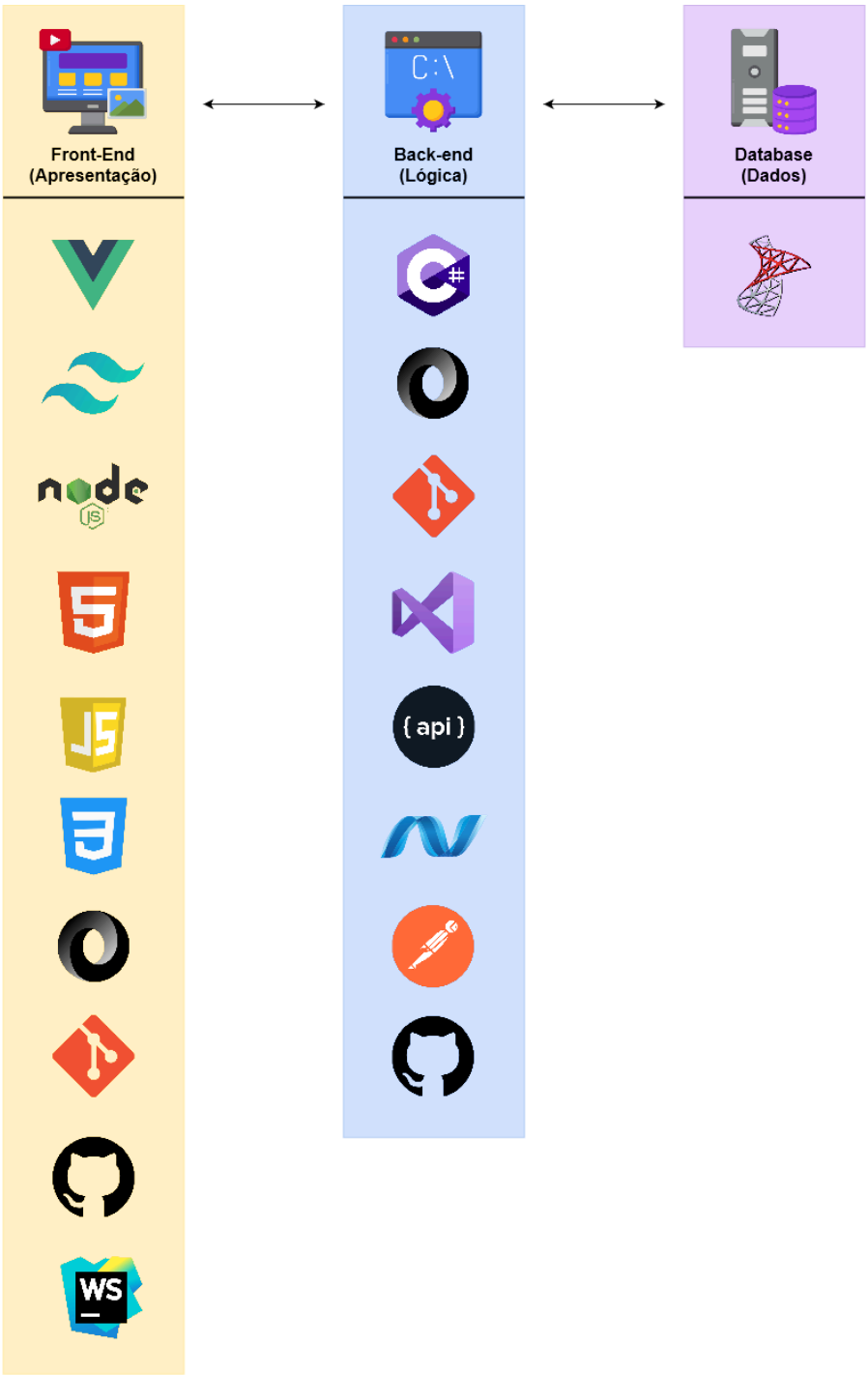


Arquitetura

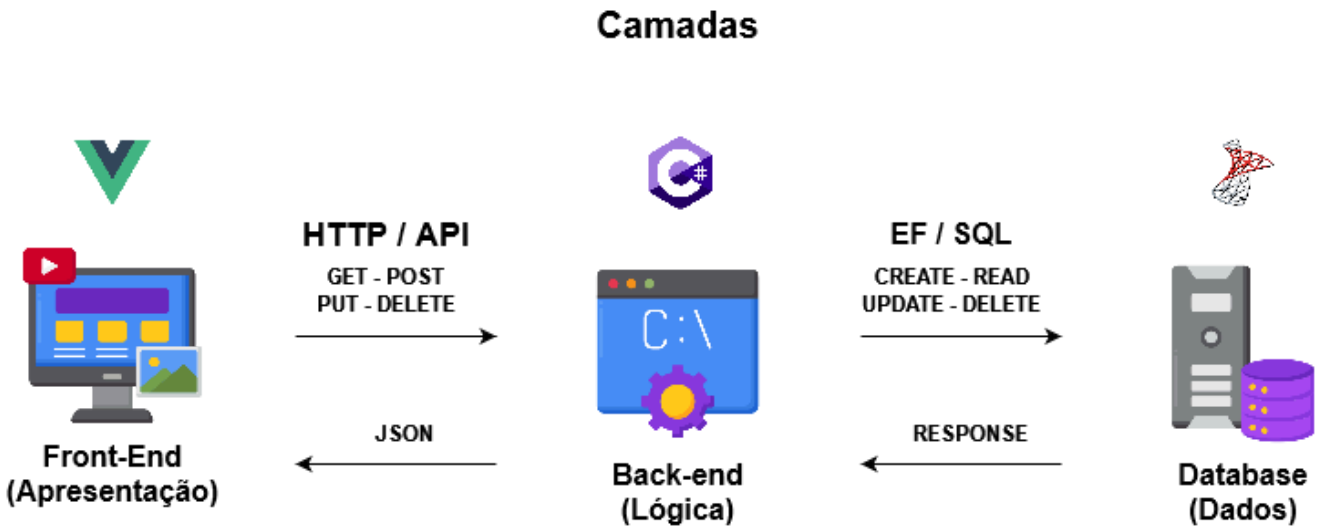
Matriz de Requisitos Não-Funcionais

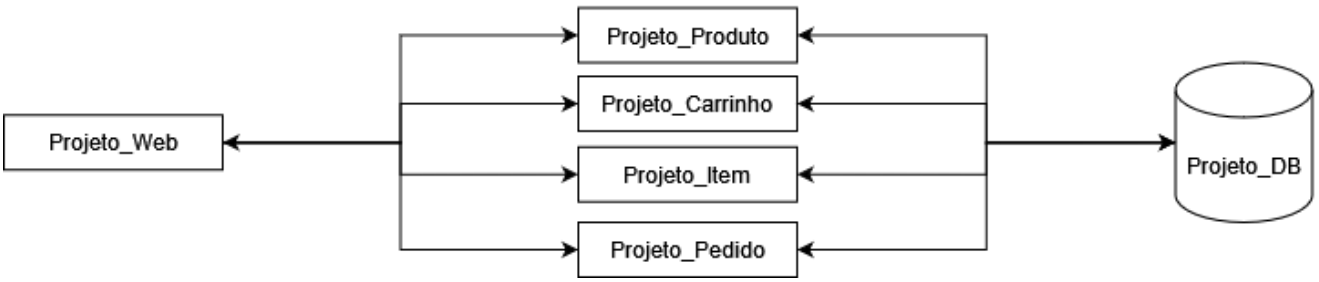
ID	Contexto	Descrição
RNF001	Confiabilidade	Garantir uma alta disponibilidade do site, minimizando períodos de inatividade para não impactar nas vendas.
RNF002	Eficiência	O tempo de carregamento da página do catálogo e do processo de checkout não deve exceder 5 segundos para garantir uma experiência do usuário ágil.
RNF003	Usabilidade	A interface do usuário deve ser intuitiva e fácil de navegar, proporcionando uma experiência agradável para usuários de diferentes níveis de habilidade.
RNF004	Manutenibilidade	Garantir a capacidade de realizar atualizações contínuas no sistema, incorporando novos produtos e recursos sem impactar negativamente a disponibilidade.
RNF005	Confiabilidade	Implementar servidores redundantes para garantir alta disponibilidade, minimizando o impacto de falhas de hardware.
RNF006	Usabilidade	O sistema deve ser compatível com dispositivos móveis para uma experiência de compra responsiva.

Stack de Tecnologias

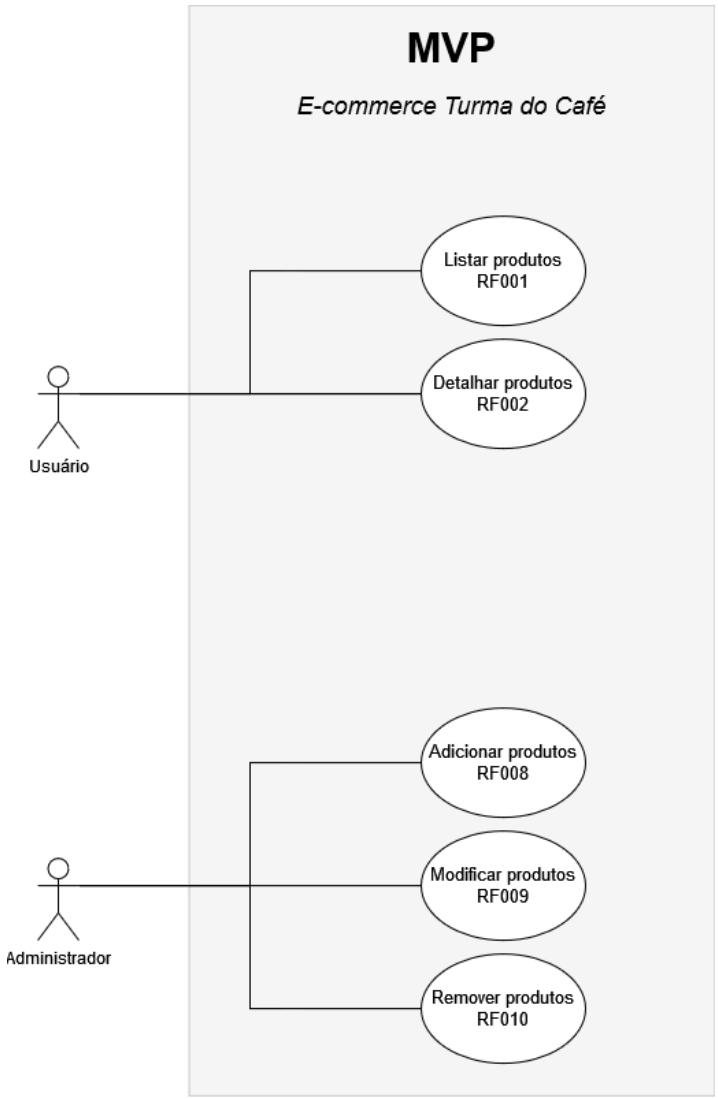


Desenho da Arquitetura da Aplicação





Descrição do Mínimo Produto Viável (MVP)



Com base no Diagrama de Casos de Uso e na funcionalidade básica de um e-commerce, o primeiro MVP, após a criação do sistema básico funcional (front-end e back-end), modelagem das classes e implementação do banco de dados, seria:

CRUD (Create - Read - Update - Delete) inicial de produtos no e-commerce.

RF001 - *O sistema deve permitir ao usuário navegar pelos produtos disponíveis no e-commerce*.

RF002 - *O sistema deve exibir informações detalhadas sobre cada produto, incluindo preço e descrição.*

Para testar se está funcionando corretamente as operações implementadas abaixo, e o que de fato o usuário final irá ver no catálogo de produtos e comprar, apresenta-se ao cliente o catálogo de produtos inicial e opção para detalhamento dos mesmos.

RF008 - *O sistema deve permitir aos administradores adicionar produtos ao catálogo.*

O primeiro passo de um e-commerce são os produtos adicionados ao catálogo. Sem produtos, sem vendas.

RF009 - *O sistema deve permitir aos administradores modificar produtos do catálogo.*

O segundo passo é implementar a modificação do mesmo para corrigir possíveis erros ou atualizar estoque.

RF010 - *O sistema deve permitir aos administradores remover produtos do catálogo.*

Caso o produto esgote ou saia de comercialização, será necessário removê-lo do e-commerce.

Projeto

Diagrama de classes completo

Caso de Uso RF002
O sistema deve permitir aos usuários pesquisar e navegar pelos produtos disponíveis no e-commerce.

Diagrama de Pacotes

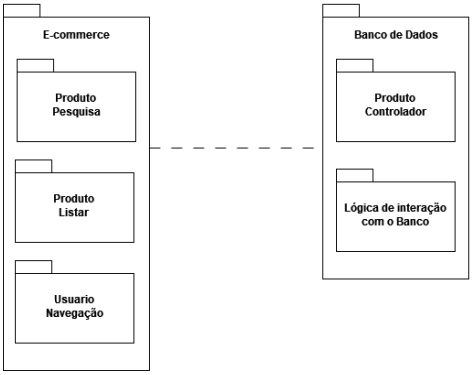
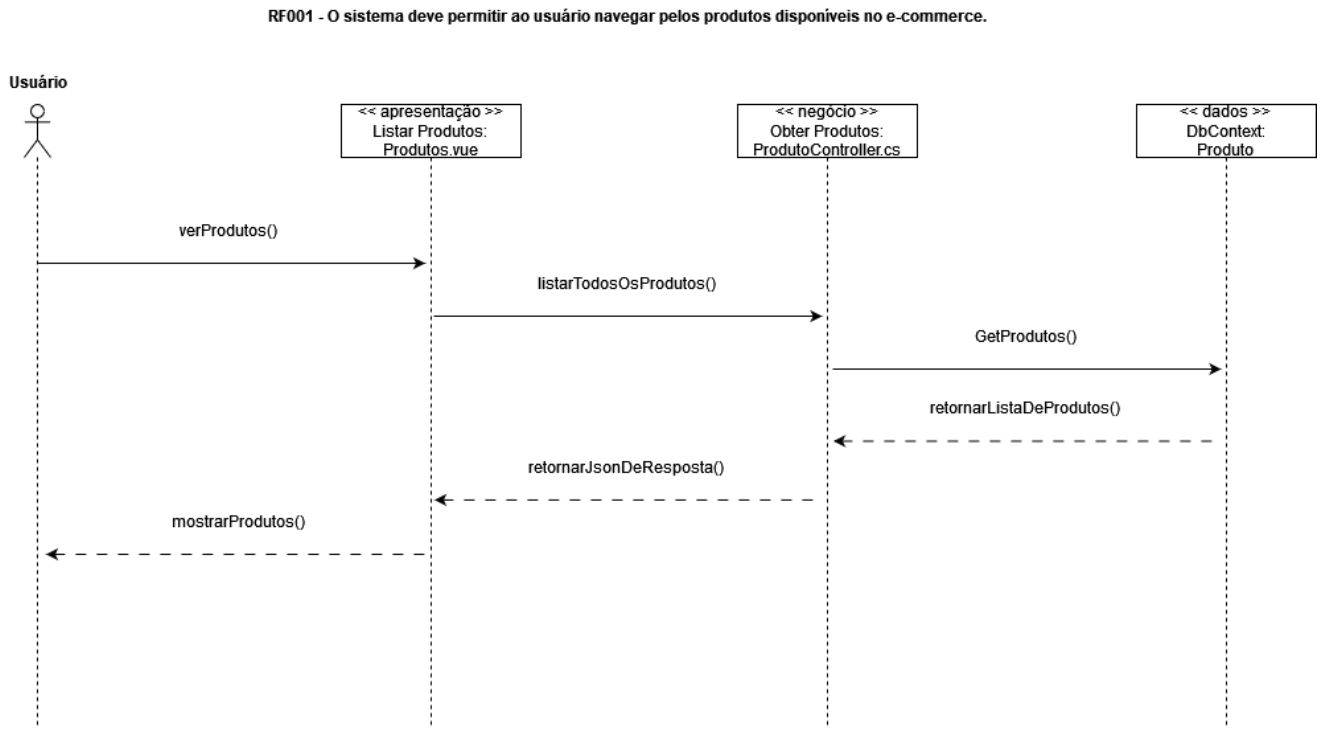


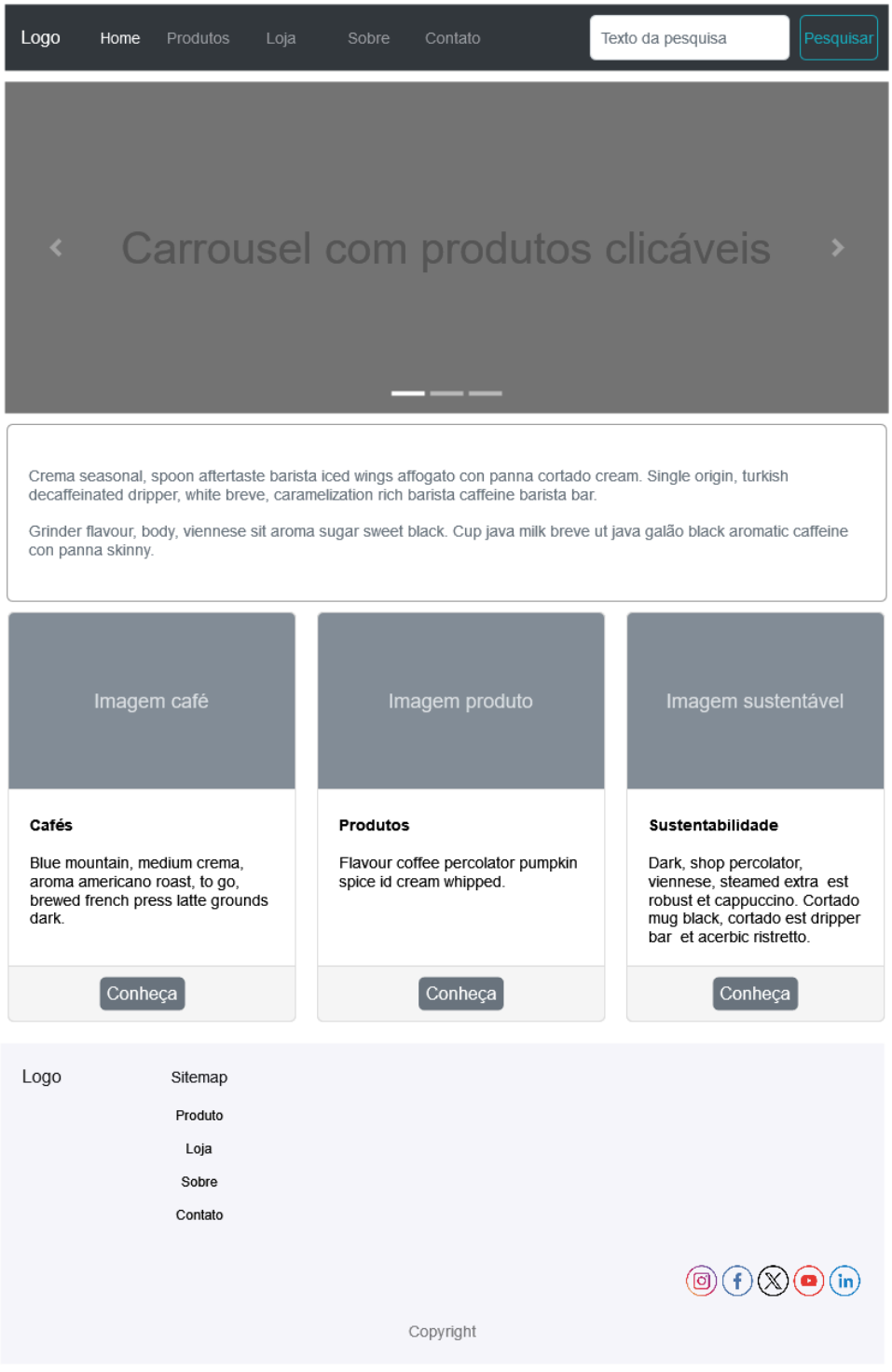
Diagrama de Relacionamento

Diagrama de Sequência

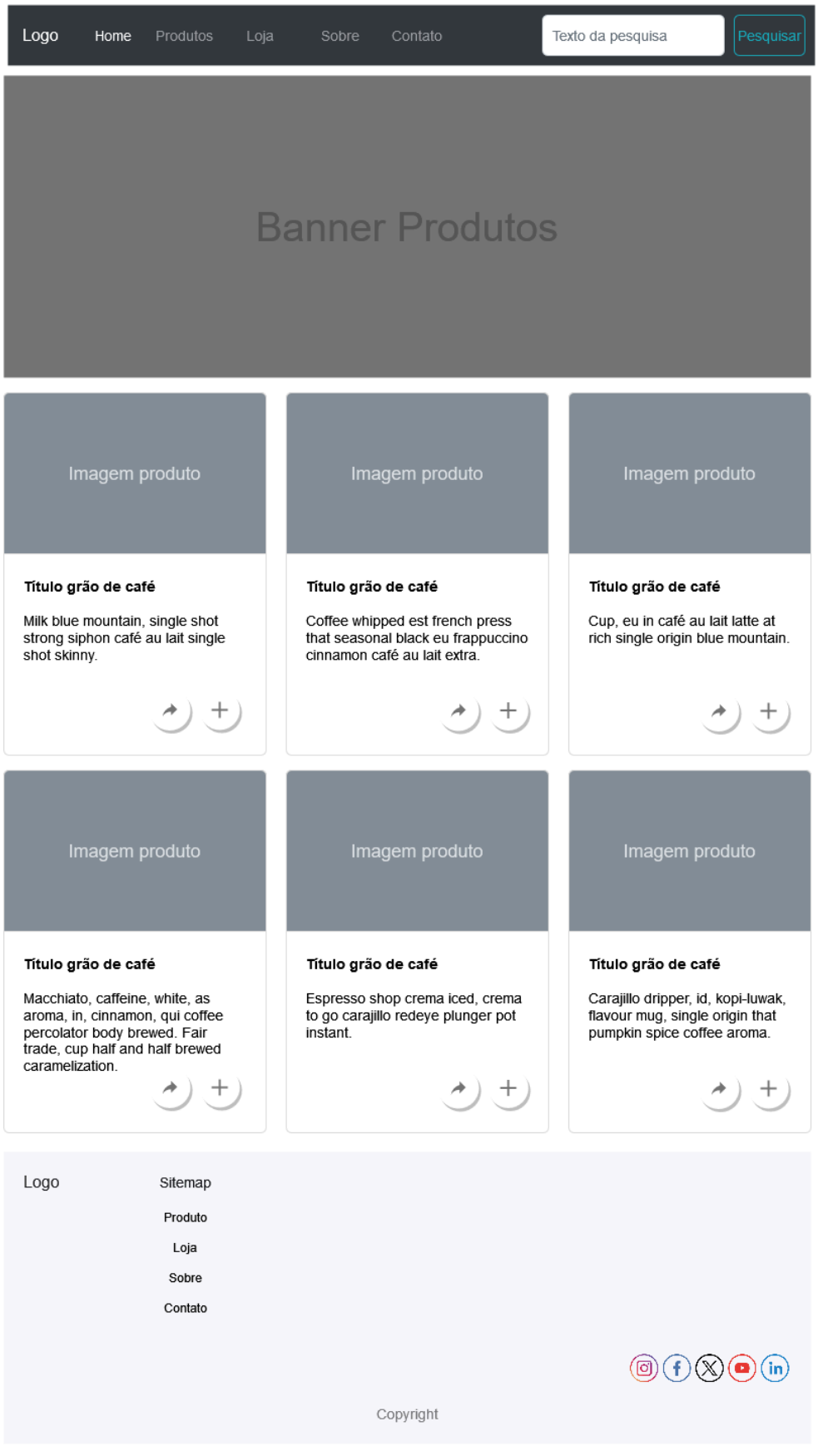


Protótipo de telas

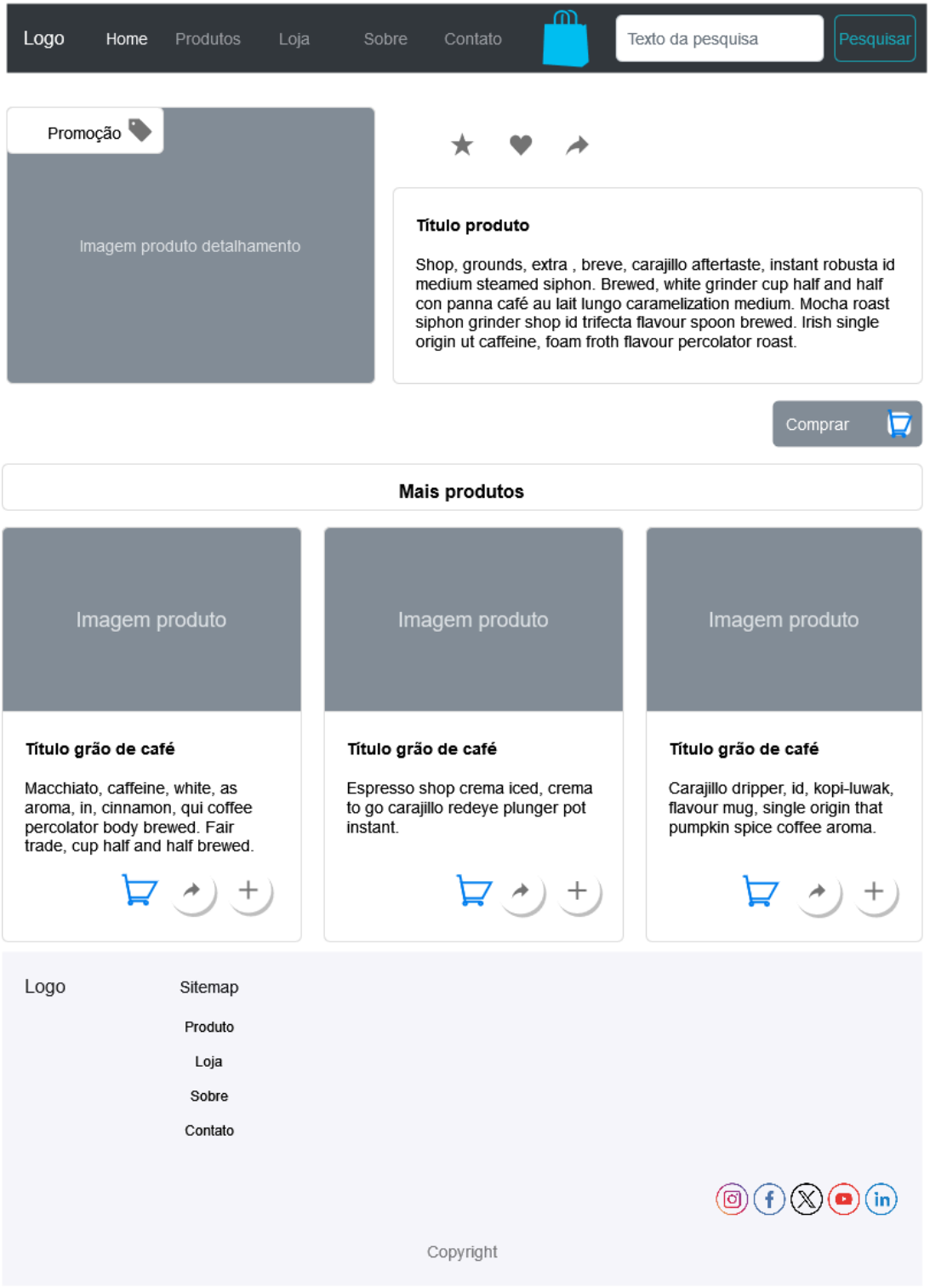
Home



Produtos


Logo
Home
Produtos
Loja
Sobre
Contato
Texto da pesquisa
Pesquisar
Banner Produtos
Imagem produto
Título grão de café
Milk blue mountain, single shot strong siphon café au lait single shot skinny.
Imagem produto
Título grão de café
Coffee whipped est french press that seasonal black eu frappuccino cinnamon café au lait extra.
Imagem produto
Título grão de café
Cup, eu in café au lait latte at rich single origin blue mountain.
Imagem produto
Título grão de café
Macchiato, caffeine, white, as aroma, in, cinnamon, qui coffee percolator body brewed. Fair trade, cup half and half brewed caramelization.
Imagem produto
Título grão de café
Espresso shop crema iced, crema to go carajillo redeye plunger pot instant.
Imagem produto
Título grão de café
Carajillo dripper, id, kopi-luwak, flavour mug, single origin that pumpkin spice coffee aroma.
Logo
Sitemap
Produto
Loja
Sobre
Contato
Copyright

Detalhamento



Construção

Plano do Projeto

Ator - Caso de Uso	NC	Iteração
Usuário - Listar produtos	1	7
Usuário - Detalhar produtos	2	7
Usuário - Adicionar carrinho	3	8
Usuário - Revisar carrinho	2	8
Usuário - Esvaziar carrinho	2	8
Usuário - Concluir compra	4	9
Usuário - Comprovante compra	2	9
Administrador - Adicionar produtos	3	7
Administrador - Modificar produtos	2	7
Administrador - Remover produtos	2	7
Administrador - Dar baixa pedidos	1	9

Repositório

- Front-end
- Back-end

Branches

Front-end

- main: Produção (estável / rollback).
- mvp-tp5: MVP do TP5.
- test-mvp: Teste/desenvolvimento do site partindo do MVP da branch *mvp-tp5*, futuramente irá ser migrada para *dev*.
- unit-tests-vitest: Desenvolvimento dos testes unitários em Vitest, futuramente irá ser migrada para *dev*.
- dev: Desenvolvimento da aplicação para testes e validação, estando o commit seguro e estável, é feito após um merge para a *main*.

Back-end

- main: Produção (estável / rollback).
- unit-tests-xunit: Desenvolvimento dos testes unitários em xUnit, futuramente irá ser migrada para *dev*.
- dev: Desenvolvimento corrente da aplicação, estando o commit seguro e estável, é feito após um merge para a *main*.
- modelagem-entidades: Testes iniciais falhos para relacionamento entre entidades, branch descontinuada.
- novo-teste-modelagem: Utilizado no desenvolvimento final, com entidades relacionadas e interligadas corretamente. Já migrada para *dev* e posteriormente *main*.

TDD

Já havia trabalhado com testes unitários, conheci os do contexto exploratório durante este desenvolvimento. Como o conceito obsoleto e em desuso de que o front-end apenas mostra dados e o mais importante é o back-end, descobri os testes para o front-end como um artefato que nos ajuda a garantir que as informações passadas e mostradas para o usuário podem ser válidas e garantirem que a informação está coesa com o esperado. Por parte do back-end consolidei melhor o conceito de testes, agora trabalhando com requisições. No front-end implementei a parte de adição de produtos com as validações de preenchimento e valores do formulário e de requisição POST para o back-end. Por outro lado, no back-end foram desenvolvidos testes unitários e exploratórios no contexto do CRUD da entidade de Produtos, foi conhecido os testes exploratórios e aplicados para lidar com cenários de que um produto tivesse contexto nulo, se existir um mesmo UUID já na tabela Produto no banco de dados, realizar concorrência em requisições POST para adicionar novos produtos ao banco de dados, aferimento do tempo médio de resposta de uma requisição POST para adicionar produtos e verificar a persistência de um produto adicionado ao banco de dados.

Testes

No contexto de testes no projeto, foi tomado por base a explicação gerada pela inteligência artificial Copilot, que mostrou um resumo interessante do que será tratado nesta seção:

> *"Os testes unitários ajudam a garantir que cada componente do seu sistema esteja funcionando corretamente, enquanto os testes exploratórios ajudam a identificar problemas que podem surgir quando o sistema é usado de maneiras inesperadas ou não previstas."*

A pasta *Tests* do projeto back-end organiza os testes de diferentes tipos do seguinte modo:

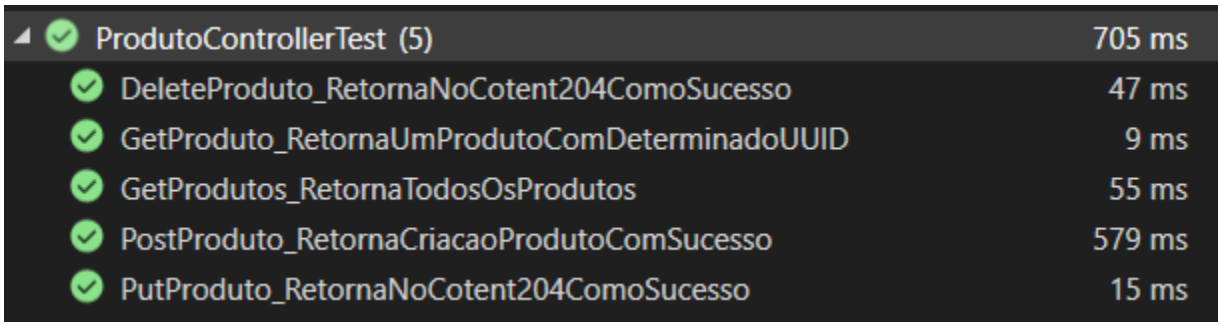
- Testes unitários: arquivos terminados com *ControllerTest*. Estes testes verificam o comportamento individual de cada controlador da aplicação.

- Testes exploratórios: arquivos terminados com *ControllerExploratoryTest*. Estes testes são realizados de forma manual e exploratória, com o objetivo de encontrar falhas que não foram detectadas pelos testes unitários.

Já a pasta *__tests__* do projeto front-end tem cada arquivo sendo especificado por uma função/componente do projeto sendo testado.

Testes unitários orientados ao TDD

Produto (ProdutoController) - Back-end API - CRUD (xUnit):



Teste GetProdutos_RetornaTodosOsProdutos

Verifica Produto(s) já adicionado(s) retorna(m) corretamente.

Assert Quantidade elementos da resposta é igual a quantidade adicionada.

Motivo Objetos adicionados são recuperados do banco de dados.

Teste GetProduto_RetornaUmProdutoComDeterminadoUUID

Verifica UUID passado retorna o objeto correto do banco de dados.

Assert UUID passado é igual ao retornado.

Motivo Operação sendo executada e o objeto sendo retornado corretamente.

Teste PostProduto_RetornaCriacaoProdutoComSucesso

Verifica Produto sendo adicionado no banco de dados corretamente.

Assert UUID passado é igual ao retornado.

Motivo Objeto está sendo adicionado ao banco de dados corretamente.

Teste PutProduto_RetornaNoCotent204ComoSucesso

Verifica Produto sendo alterado corretamente no banco de dados.

Assert Resposta HTTP 204 (No Content) e Produto.Nome atualizado.

Motivo Objeto está sendo alterado no banco de dados corretamente.

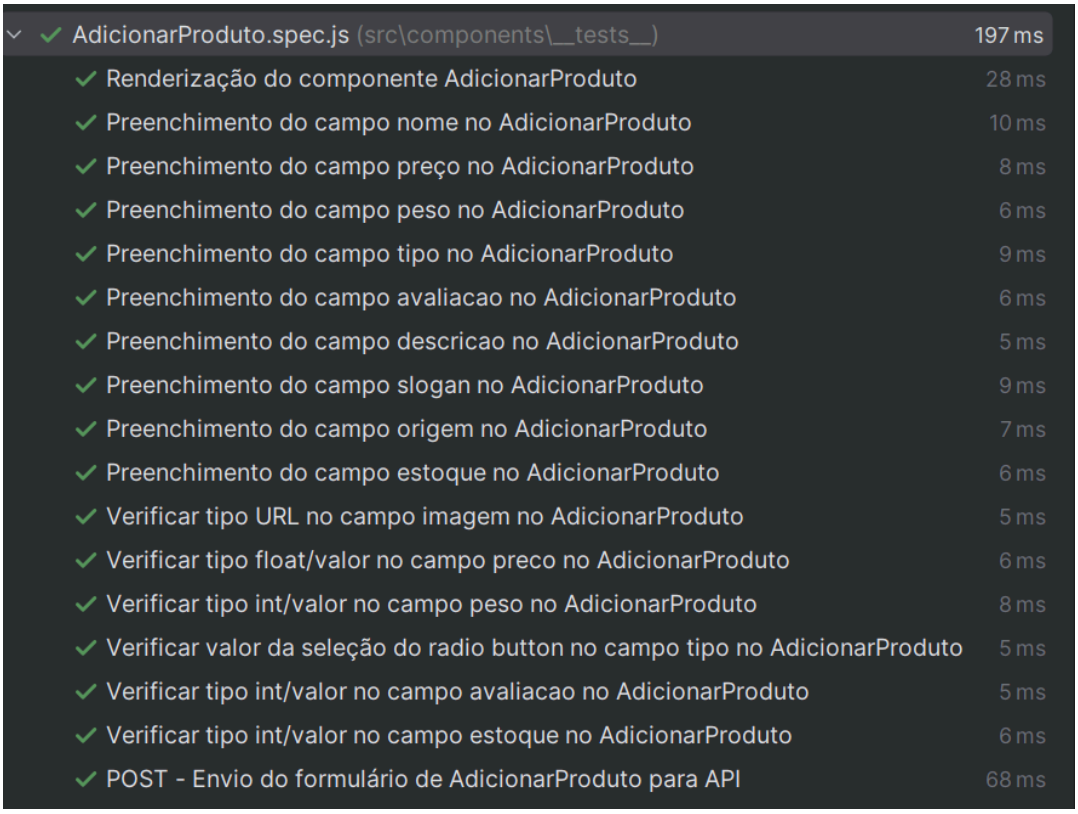
Teste DeleteProduto_RetornaNoCotent204ComoSucesso

Verifica Produto sendo deletado corretamente no banco de dados.

Assert Resposta HTTP 204 (No Content) e objeto nulo no banco de dados.

Motivo Objeto está sendo deletado no banco de dados corretamente.

Produto (AdicionarProduto) - Front-end - Vue (vitest):



Teste Renderização do componente AdicionarProduto

Verifica Componente AdicionarProduto foi renderizado.

Assert Verifica se ele existe (true) usando o wrapper.

Motivo Verificar se o componente existe para dar base a aplicação.

Teste Preenchimento do campo nome no AdicionarProduto

Verifica Preenchimento do campo Nome do produto está funcionando.

Assert Valida se o valor do campo Nome do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo preco no AdicionarProduto

Verifica Preenchimento do campo Preço do produto está funcionando.

Assert Valida se o valor do campo Preço do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo peso no AdicionarProduto

Verifica Preenchimento do campo Peso do produto está funcionando.

Assert Valida se o valor do campo Peso do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo tipo no AdicionarProduto

Verifica Preenchimento do campo Tipo do produto está funcionando.

Assert Valida se o valor do campo Tipo do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo avaliacao no AdicionarProduto

Verifica Preenchimento do campo Avaliação do produto está funcionando.

Assert Valida se o valor do campo Avaliação do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo descricao no AdicionarProduto

Verifica Preenchimento do campo Descrição do produto está funcionando.

Assert Valida se o valor do campo Descrição do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo slogan no AdicionarProduto

Verifica Preenchimento do campo Slogan do produto está funcionando.

Assert Valida se o valor do campo Slogan do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo origem no AdicionarProduto

Verifica Preenchimento do campo Origem do produto está funcionando.

Assert Valida se o valor do campo Origem do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Preenchimento do campo estoque no AdicionarProduto

Verifica Preenchimento do campo Estoque do produto está funcionando.

Assert Valida se o valor do campo Estoque do produto é igual ao preenchido.

Motivo Verificar se o preenchimento do campo está funcionando corretamente.

Teste Verificar tipo URL no campo imagem no AdicionarProduto

Verifica “type” URL do campo imagem se tem um padrão de URL.

Assert Valida se o valor do campo imagem possui um padrão URL válido.

Motivo Verificar se o “type” do campo está funcionando corretamente.

Teste Verificar tipo float/valor no campo preco no AdicionarProduto

Verifica “type” number do campo preco se tem um padrão de número/float.

Assert Valida se o valor do campo preco possui um padrão número float/válido.

Motivo Verificar se o “type” do campo está funcionando corretamente.

Teste Verificar tipo int/valor no campo peso no AdicionarProduto

Verifica “type” number do campo peso se tem um padrão de número/int.

Assert Valida se o valor do campo peso possui um padrão número/int válido.

Motivo Verificar se o “type” do campo está funcionando corretamente.

Teste Verificar valor da seleção do radio button no campo tipo no AdicionarProduto

Verifica valor da seleção do radio button condiz com o selecionado.

Assert Valida se o valor selecionado é igual ao esperado.

Motivo Verificar se o item escolhido é igual ao que está no código.

Teste Verificar tipo int/valor no campo avaliacao no AdicionarProduto

Verifica “type” number do campo avaliacao se tem um padrão de número/int.

Assert Valida se o valor do campo avaliacao possui um padrão número/int válido.

Motivo Verificar se o “type” e intervalo do campo estão funcionando.

Teste Verificar tipo int/valor no campo estoque no AdicionarProduto

Verifica “type” number do campo estoque se tem um padrão de número/int.

Assert Valida se o valor do campo estoque possui um padrão número/int válido.

Motivo Verificar se o “type” e valor mínimo do campo estão funcionando.

Teste POST - Envio do formulário de AdicionarProduto para API

Verifica Captação dos dados do formulário, montagem em JSON e envio para API.

Assert Resposta da API com o JSON enviado.

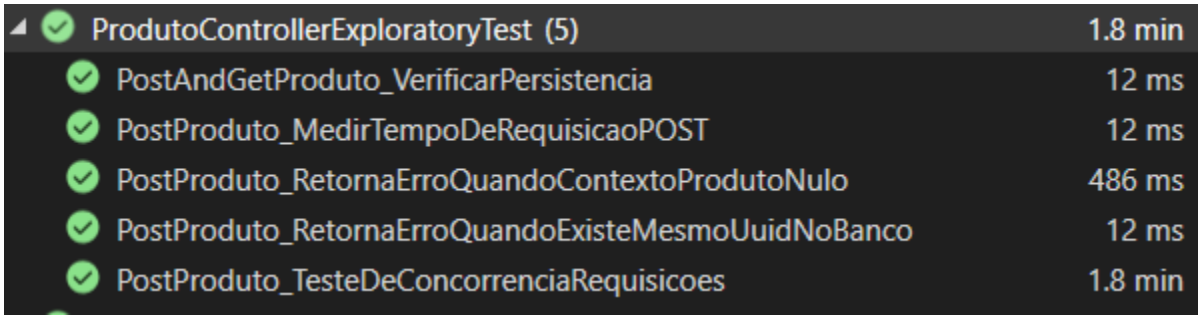
Motivo Verificar se o formulário capta os dados, monta o JSON e faz a requisição.

Plano de testes exploratórios

No contexto de testes exploratórios, o caso de uso escolhido para realizar os testes foi:

RF008 - O sistema deve permitir aos administradores adicionar produtos ao catálogo.

Produto - **Back-end API - CRUD (xUnit):**



Teste PostProduto_RetornaErroQuandoContextoProdutoNulo

Explora Comportamento da API com produto enviado com contexto nulo.

Assert Resposta HTTP 400 (Bad Request) e mensagem de erro.

Motivo Comportamento da API em caso de falha (nula) de objeto.

Teste PostProduto_RetornaErroQuandoExisteMesmoUuidNoBanco

Explora Comportamento da API com UUID enviado já existente no banco de dados.

Assert ArgumentException e mensagem de erro.

Motivo Comportamento da API em caso de envio com UUID existente.

Teste PostProduto_TesteDeConcorrenciaRequisicoes

Explora Comportamento da API com concorrência em requisições POST (10000).

Assert Contagem de entradas no banco de dados ser igual a 10000 (~ 1,8 minutos).

Motivo Comportamento da API em caso de múltiplas requisições simultâneas.

Teste PostProduto_MedirTempoDeRequisicaoPOST

Explora Tempo médio para executar requisição POST.

Assert Atributo Id (UUID) passado é igual ao retornado.

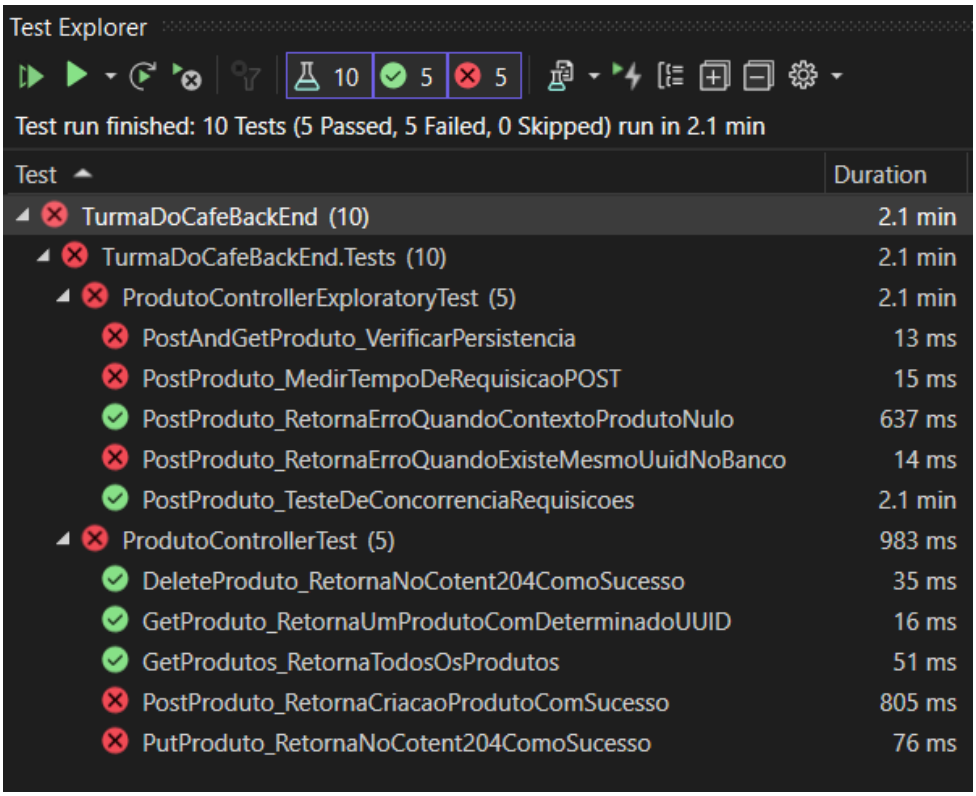
Motivo Verificar tempo médio e desempenho de requisição POST.

Teste PostAndGetProduto_VerificarPersistencia

Explora Capacidade da API em persistir dados no banco e retorná-los.

Assert Respostas HTTP 200/201 (Created/OK) e dados de envio e retorno iguais.

Motivo Garantir dados adicionados e recuperados com sucesso e coesão.



Depois da atualização do relacionamento das entidades no back-end para continuar o desenvolvimento do sistema, metade do conjunto de testes falharam. No desenvolvimento de software isso é esperado quando ocorre uma alteração no código.

Implantação

Plano de Implantação

Resumidamente e de forma técnica, o projeto se consiste em:

- Front-end: Vue.js com Vite.
- Back-end: C# com Entity Framework em modelo Web API.
- Banco de dados: SQL Server no ExpressSQL no Visual Studio.
- Comunicação: Requisições HTTP (JSON).

Versões recomendadas:

- Front-end
    - Vue: 3.3.11
    - Vite: 5.0.8
    - Node: 21.4.0
    - Demais dependências encontram-se no arquivo *package.json* no diretório raiz do projeto.

- Back-end
    - Windows Server 2022.
    - .NET: 6 LTS.
    - Demais dependências encontram-se ao dar um duplo clique no projeto dentro do ambiente de desenvolvimento Visual Studio.

- Banco de dados
    - SQL Server: 15.0.4153.

Na implantação abaixo, foi escolhido usar a Google Cloud Platform (GCP) pela sua interface mais simples, escalabilidade e por ter os recursos necessários para o funcionamento do e-commerce mais próximo ao cliente dentro da zona de São Paulo (*southamerica-east1*).

Este processo abstrai a aquisição de um domínio e a etapa de verificação de domínio para criação do bucket no Cloud Storage. Para colocar o e-commerce na web, será necessário o domínio e a verificação do mesmo junto ao Google para criar um bucket com o domínio de um website e torná-lo acessível como um website normal. É necessário realizar esta parte para concluir a etapa 10 abaixo.

O processo consiste em dez etapas, subdivididas por tarefas:

1. Configuração da GCP

1.1. Primeiro acesso

Acesse a plataforma e crie uma conta.

1.2. Configuração

Configure o projeto, orçamento e escolha a zona citada acima.

2. Habilitacao de APIs

2.1. APIs e Serviços

Ative as seguintes APIs: Cloud SQL, Cloud Storage e App Engine na seção citada.

3. Criação das instâncias de máquina virtual (back-end e banco de dados)

3.1. Criação das instâncias

Na seção Compute Engine, crie duas instâncias de máquinas virtuais.

- Back-end: Sistema operacional windows com .NET instalado.
- Banco de dados: SQL Server com sistema operacional Windows e SQL Server instalado.

3.2. Recomendação

Escolher um tipo de máquina simples para ir escalando conforme necessário.

4. Configuração do Cloud SQL (banco de dados)

4.1. Criação da instância

Na seção de Cloud SQL, crie uma instância de banco de dados SQL Server.

4.2. Versão e configuração

Selecione a versão do SQL Server e configure o banco de dados.

4.3. Conexão

Conecte a instância da máquina virtual do back-end com o banco de dados.

5. Configuração do Cloud Storage (front-end)

5.1. Criação do bucket

Crie um bucket na região única recomendada com classe padrão para armazenamento de arquivos.

5.2. Permissões

Defina as permissões de acesso para todos os usuários como leitura de objetos apenas.

6. Publicar a API Web

6.1. Publicação

Publique a API Web como aplicativo .NET Core.

6.2. Recomendação

Utilizar o Visual Studio ou CLI do .NET para realizar a operação.

7. Configuração das variáveis de ambiente

7.1. Console do App Engine

Dentro do console do App Engine, configure as variáveis de ambiente para API Web, incluindo a string de conexão com o banco de dados.

8. Implantação do front-end

8.1. Compilação

Compile o projeto do front-end no ambiente de desenvolvimento integrado com o comando no terminal *vite build* para gerar o código de produção.

8.2. Upload do front-end

Na pasta *build* gerada dentro do projeto, faça o upload dela para o bucket criado no Cloud Storage.

8.3. Arquivos padrões

Defina nas propriedades do bucket os arquivos responsáveis pela rota raíz e 404.

9. Testar e integrar front-end com back-end

9.1. Teste a comunicação

Utilizando um software de requisições HTTP, como o Postman para testar a API Web do back-end.

9.2. Comunicação front-end com back-end

Verifique se o front-end consegue se comunicar com o back-end por meio das requisições HTTP.

9.3. Caso tenha problemas

Recomenda-se utilizar os logs do App Engine e Cloud SQL para identificar possíveis causas e resolvê-las.

10. Lançamento

10.1. Definição de domínio e configuração DNS

Com domínio registrado, configurar a DNS para apontar para o IP da instância do App Engine.

10.2. Abrir o acesso ao público geral

Altere as configurações do App Engine para permitir acesso público.

10.3. Monitoramento de desempenho

Recomenda-se utilizar o Cloud Monitoring para verificar o desempenho da API e do banco de dados, para assim realizar otimizações e melhorar o tempo de resposta e escalabilidade.

Extra

Refatoração

Front-end

Durante o desenvolvimento do projeto, foi percebido que o framework Vue é muito mais rápido no quesito de carregamento e renderização dos dados da página em relação ao tempo de resposta da requisição para preenchimento dos componentes com os dados vindo do back-end, assim em alguns casos foi utilizado *watcher,* a função *setTimeOut* do JavaScript e/ou uma variável booleana de controle (*ok* e/ou *ok2*) *para* tentar atrasar a renderização do componente e receber e processar o JSON de resposta do back-end, evitando assim que o desse erro no DOM e a página não fosse renderizada. Alguns casos tiveram êxito, onde foi conseguido contornar com esta abordagem.

É percebido no código do front-end na parte de <*script*> de componentes Vue que existe repetição de código do mesmo método em vários componentes diferentes, foi tentado criar um arquivo global de métodos para requisições, aproveitando assim a reutilização de código, porém por conta do problema explicado no parágrafo anterior, mesmo com uso de métodos assíncronos ou uso do *await* nada parecia ser respeitado pelo framework e o tempo de requisição era maior do que se o método da requisição estivesse dentro do componente, o que gerava erros no carregamento do componente e impossibilitava o uso global de métodos.

Outro ponto foi que por algum motivo, os dados recebidos das requisições parecem não serem armazenados nas constantes de referência do JavaScript, o que precisou ser remediado com o maior uso de variáveis locais em cada componente. Isso acabou afetando o ponto de simplificação do código.

Back-end

Por ter uma série de fatores na transição de um produto para item, ou seja, quando ele é adicionado ao Carrinho, a lógica da criação de Carrinho, Item e Pedido ficaram no mesmo escopo, o que poderia ser refatorado para mais métodos e cada um dentro da responsabilidade do seu controller. Por questão de tempo e com o foco na entrega do sistema funcionando, foi optado por deixar assim, mas com essa abertura para futura refatoração.

Front-end - Back-end

Na junção dos dois contextos, foi preciso passar de forma não aconselhável dados pelo cabeçalho da requisição no front-end para o back-end, os dados são menos sensíveis mas em produção isso não é recomendado. Poderia ser utilizado o conceito abordado no projeto de DTO, onde escolheria os dados daquele DTO específico para cada tipo de requisição.

Observação

Utilização da GCP para armazenar imagens de Produto

Foi utilizado para armazenar as imagens do Produtos um container de arquivos estáticos na GCP com o Cloud Storage, optou-se por utilizar esta abordagem por conta de inúmeras tentativas de salvar a imagem do Produto no banco em formato blob com conversão da imagem para blob no front-end e decodificação no retorno para exibição, mas não foi possível isso, então ficou esta parte manualmente com uma string da URL pública do arquivo na GCP.

Rodando localmente front-end e back-end com política de CORS

Após build do back-end e front-end, recomenda-se para o back-end no Visual Studio ou IDE de preferência, acessar a classe *Program.cs* localizado na raíz do projeto e adicionar o IP e porta local que o front-end está sendo executado na máquina, isso pode ser feito adicionando no bloco de política de IPs do CORS, conforme abaixo. Garantindo que o back-end permita requisições do front-end que está no IP local na porta 8080 ou 8081, dependendo do tipo de *npm script* que está sendo executado. Caso não faça isso, poderá ter o problema de CORS. Recomenda-se utilizar o front-end no http://localhost:8080 ou 8081, URLs já configuradas.

```
builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "MyPolicy",
        policy =>
        {
            policy.AllowAnyHeader().AllowAnyMethod();
            policy.WithOrigins(
                "http://localhost:8080",
                "http://192.168.15.111:8080",
                "http://192.168.3.192:8080",
                "http://192.168.154.1:8080",
                "http://192.168.242.1:8080",
                "http://192.168.3.178:8080",
                "http://192.168.5.190:8080",

                "http://localhost:8081",
                "http://192.168.15.111:8081",
                "http://192.168.3.192:8081",
                "http://192.168.154.1:8081",
                "http://192.168.242.1:8081",
                "http://192.168.3.178:8081",
                "http://192.168.5.190:8081"
            );  // mudar os IPs caso os IPs do computador mudar - ver no ipconfig na CMD
        });
});
```

Changelog

TP1 para TP3

- Adição da logo a capa do documento.
- Revisão do Diagrama de Casos de Uso.
- Revisão de Prioridades da Matriz de Requisitos Funcionais.
- Adição do Diagrama de Sequência com alguns casos de uso.
- Adição do Diagrama de Classes.
- Adição das Responsabilidades GRASP e GoF.
- Atualização do Apêndice.
- Reformatação do documento.
- Adição dos Changelogs.

TP3 para TP5

- Reformulação do índice para o modelo estipulado.
- Adição dos Requisitos Não-Funcionais, da Stack de Tecnologias, do Desenho da Arquitetura da Aplicação, dos links dos Repositório, da Relação de Domínios de Negócio e Diagrama de classes completo.
- Revisão da tabela da Matriz de Requisitos Funcionais.
- Refatoração do documento para atender ao novo índice.

TP5 para TP7

- Migração do Requisito de Responsividade de Funcional para Não-funcional.
- Remoção do caso de uso de Responsividade do Diagrama de Casos de Uso.
- Correção do link do GitHub para o projeto do Back-end.
- Adição dos testes unitários para front-end (vitest) e back-end (xUnit)
- Adição dos testes exploratórios para o back-end
- Adição da sub-seção “Branches” na seção Repositório
- Alteração das páginas no Sumário

TP7 para TP9

- Atualização da parte de branches em Repositório.
- Adição de conteúdo na parte de Visão do Projeto.
- Revisão do Sumário da documentação.
- Renomeação de Introdução para Visão do Projeto.
- Adição na seção de Análise do tópico Descrição dos Casos de Uso do MVP.
- Revisão do Plano de Projeto.
- Remoção da seção de perguntas Apêndice 1 - Introdução ao DAD.
- Revisão da seção de Testes.

- Adição do conteúdo para Implantação do Projeto.
- Revisão do Diagrama de Classes.
- Modificação das branches e conteúdo na seção de Repositório.
- Atualização dos resultados dos testes unitários e exploratórios do back-end.
- Refatoração do espaçamento deste documento.

TP9 para ENTREGÁVEL

- Atualização do Diagrama de Classes.
- Atualização do Diagrama de Domínio.
- Atualização do Diagrama de Casos de Uso.
- Atualização do Desenho da Arquitetura da Aplicação.
- Adição da seção de Observação.
- Adição da seção de Refatoração.
- Renomeação de Sumário para Índice.
- Renomeação de itens do Índice.