

Igor Gomes da Silva

Guilherme Cordeiro Rodrigues

Erick Ribeiro de Santana

Rafael Santos Mauro de Menezes

Kauã Silva Dias

Gustavo Nunes de Oliveira

B2B Smart

Sistema de e-commerce entre fornecedores e clientes

Sumário

1.	Introdução.....	3
2.	Visão geral.....	3
2.1.	Descrição.....	3
2.2.	O projeto e a contribuição para a comunidade.....	3
2.3.	ESG e as ODS contempladas no projeto.....	3
2.4.	Justificativa.....	4
2.5.	Objetivo.....	4
3.	Escopo do projeto.....	4
3.1.	Matriz de papéis e responsabilidades.....	5
4.	Regras de negócio.....	6
5.	Descrição de requisitos.....	7
5.1.	Requisitos funcionais.....	7
5.2.	Requisitos não funcionais.....	8
6.	Modelos de casos de uso.....	9
6.1.	Identificação dos autores e suas responsabilidades.....	9
6.2.	Descrição de prioridades no desenvolvimento.....	9
6.3.	Diagrama de caso de uso.....	10
6.4.	Descritivo de caso de uso.....	11
7.	Diagramas.....	13
7.1.	Diagrama de atividades.....	13
7.2.	Diagrama de sequência.....	14
7.3.	Diagrama de entidade e relacionamento.....	15
7.4.	Diagrama de classes.....	16
8.	Metodologia.....	17
9.	Melhorias futuras.....	33
10.	Referência Bibliográfica.....	34

1. Introdução

Esta documentação visa detalhar os requisitos funcionais e não funcionais, as regras de negócio e os modelos de casos de uso associados ao sistema B2BSmart, um sistema de comércio eletrônico projetado para facilitar transações entre fornecedores (produtores) e clientes (compradores) que residem e atuam em toda a região Sul do Brasil, sendo Paraná, Santa Catarina e Rio Grande do Sul. O objetivo é fornecer uma visão abrangente e técnica das funcionalidades, interações e processos operacionais que serão implementados para garantir a eficácia, segurança e conformidade do sistema no contexto de um ambiente B2B.

2. Visão geral

O sistema B2B Smart é um sistema para desktop de comércio eletrônico B2B (Business-to-Business) projetado para facilitar transações entre fornecedores e clientes de pequeno, médio e grande porte. Este sistema busca otimizar e automatizar o processo de compra e venda, proporcionando uma plataforma integrada e eficiente para ambas as partes, além de fornecer mais opções de negócios para ambos os lados, buscando a maior ascensão de ambas as empresas.

2.1 Descrição

O sistema B2B Smart oferece uma interface intuitiva e segura onde os fornecedores podem listar seus produtos, definir preços e monitorar as vendas. Por outro lado, os clientes têm acesso a um catálogo diversificado de produtos, com informações detalhadas, opções de compra simplificadas e ferramentas de gestão de pedidos e pagamentos.

2.2 O projeto e a contribuição para a comunidade

O projeto B2B Smart contribui significativamente para a comunidade empresarial, oferecendo uma solução tecnológica que promove a eficiência operacional, a expansão de mercados e a sustentabilidade econômica. Ao facilitar e automatizar o comércio entre fornecedores e clientes de pequeno e médio porte, dessa forma, permitindo que ambos usuários tenham possibilidade de ascender financeiramente.

2.3 ESG e as ODS contempladas no projeto

O projeto B2B Smart está alinhado com os princípios de ESG (Environmental, Social, and Governance) ao promover práticas comerciais sustentáveis, inclusivas e éticas.

Em relação aos Objetivos de Desenvolvimento Sustentável (ODS) da ONU, o projeto contribui principalmente para o ODS 8 (Trabalho Decente e Crescimento Econômico) ao fomentar o desenvolvimento de pequenas empresas e impulsionar o emprego para estes

estabelecimentos, e ODS 9 (Indústria, Inovação e Infraestrutura) ao impulsionar a digitalização e a modernização do setor B2B.

2.4 Justificativa

A necessidade de um sistema como o B2B Smart surgiu da identificação dos desafios enfrentados pelos fornecedores e clientes de pequeno e médio porte no processo de comércio B2B, além da grande dificuldade de ascender financeiramente. A falta de plataformas integradas, eficientes e seguras para facilitar essas transações resulta em ineficiências operacionais, oportunidades perdidas e barreiras ao crescimento. Portanto, a implementação deste sistema visa preencher essa lacuna, oferecendo uma solução abrangente e adaptada às necessidades específicas do mercado B2B.

2.5 Objetivo

O objetivo principal do projeto B2B Smart é desenvolver, implementar e operacionalizar um sistema de comércio eletrônico B2B simplificado, seguro e escalável que atenda às necessidades dos fornecedores e clientes. Fazendo com que tanto os fornecedores vendem mais os seus produtos em lotes, quanto o cliente tenha uma maior diversidade de produtos em seu estabelecimento, nosso principal objetivo é facilitar seus negócios.

3. Escopo do projeto

O escopo do projeto B2B Smart engloba o desenvolvimento de um sistema de comércio eletrônico B2B com uma arquitetura orientada a serviços.

No front-end, serão empregadas as linguagens e tecnologias HTML, CSS, JavaScript e React para conceber uma interface de usuário intuitiva e responsiva. As funcionalidades front-end abrange áreas como registro, login, catálogo de produtos, carrinho de compras e painéis de controle específicos para fornecedores e clientes.

No back-end, a implementação será realizada utilizando Java com Spring Boot para o desenvolvimento de APIs RESTful. Essas APIs serão responsáveis pela autenticação, autorização, lógica de negócios e integração com o banco de dados SQL, que será utilizado para armazenar informações relativas a usuários, produtos e pedidos.

3.1 Matriz de papéis e responsabilidades

Igor Gomes da Silva - responsável pelo desenvolvimento das APIs RESTful essenciais para o sistema B2B Smart, abrangendo autenticação, autorização e integração com o banco de dados SQL.

Guilherme Cordeiro Rodrigues - desempenha o papel de gestor técnico da equipe e assume a responsabilidade pela coordenação estratégica, gestão de recursos e elaboração da documentação técnica detalhada do projeto B2B Smart.

Erick Ribeiro de Santana - é o responsável pela concepção e desenvolvimento do design criativo das interfaces do sistema B2B Smart, focando na usabilidade, estética e experiência do usuário (UX/UI).

Kauã Silva Dias - é o responsável pelo desenvolvimento front-end do projeto B2B Smart, utilizando HTML, CSS, JavaScript e React para criar uma interface de usuário intuitiva e responsiva.

Gustavo Nunes de Oliveira - atua como um desenvolvedor full-stack, fornecendo suporte e auxílio tanto no desenvolvimento back-end quanto front-end do projeto B2B Smart, colaborando estreitamente com os desenvolvedores especializados em cada área para garantir a integração e a coesão das funcionalidades do sistema.

Rafael Santos Mauro de Menezes - desempenha um papel multifuncional no projeto B2B Smart. Ele contribui ativamente no design das interfaces, garantindo uma estética e usabilidade otimizadas. Além disso, ele é responsável pela realização de testes abrangentes, identificando e tratando possíveis exceções e erros nos diferentes processos de desenvolvimento.

4. Regras de negócio

RN01 - Pré-requisito para acesso às funcionalidades do sistema

- É necessário que o usuário esteja conectado ao sistema para que ele utilize de todas as ferramentas

RN02 - Pré-requisito para cadastrar produtos dentro do sistema

- Apenas usuários tipados como fornecedores têm a possibilidade de realizar o cadastro dos produtos

RN03 - Pré-requisito para realizar a compras dentro do sistema

- Apenas usuários tipados como clientes podem realizar a compra dos produtos com os fornecedores

RN04 - Pré-requisito para atualizar status do pedido

- Apenas usuários tipados como fornecedores podem cancelar ou aprovar pedidos e, caso seja aprovado, o cliente pode confirmar seu pedido a fim de garantir que tenha certeza do pedido.
- Cada solicitação dura no máximo 72 horas, ou seja, o fornecedor deve aceitar ou recusar esse pedido dentro desse tempo e, caso não haja resposta, será cancelado. Já o cliente também terá essas mesmas 72 horas para confirmar seu pedido e, caso não haja resposta, será aprovado.

RN05 - Pré-requisito para cancelar a compra efetuada

- Apenas usuários administradores podem cancelar compras que foram aceitas pelo fornecedor por engano.

5. Descrição de requisitos

5.1 Requisitos funcionais

[RF001] Cadastro usuário fornecedor e cliente.

O Sistema permite ao usuário realizar o cadastro de 2 formas, sendo uma cliente e outra fornecedor (clicando em uma opção que disponibilizamos no cadastro do usuário), isso definirá sua função no sistema.

[RF002] Editar cadastro de usuários.

O sistema permite editar as informações do usuário que foi criado caso tenha alguma informação errada.

[RF003] Realizar pedidos - Cliente.

O sistema permite ao cliente realizar pedidos com base em suas necessidades para os fornecedores, definindo o fornecedor e escolhendo os produtos.

[RF004] Aprovar ou recusar pedidos - Fornecedor.

O sistema permite ao fornecedor aceitar ou cancelar o pedido feito por um cliente.

[RF005] Cadastro de produtos - Fornecedor.

O sistema permite que o fornecedor cadastre novos produtos no sistema para que sejam comprados.

[RF006] Verificação de ganhos e gastos - fornecedor e cliente respectivamente.

O sistema permite que o fornecedor verifique seus ganhos com base em suas vendas e que o cliente verifique seus gastos.

[RF007] Tela de suporte e informações.

O usuário terá acesso a uma tela onde será apresentado nossas informações e contato para possíveis requisições.

5.2 Requisitos não funcionais

[RNF001] Desempenho de alta velocidade.

O sistema deverá suportar 1000 transações por segundo com uma resposta das ações menor que 3 segundos.

[RNF002] Segurança.

O sistema deverá garantir que não haja o vazamento de informações de seus usuários e os dados serão guardados por criptografias.

[RNF003] Compatibilidade.

O sistema deverá funcionar em todos os tipos de navegadores (Safari,Firefox,Opera,Google,etc).

[RNF004] Conformidade.

O sistema deve estar em conformidade com as regulamentações de privacidade de dados, como o GDPR.

[RNF005] Usabilidade.

O sistema deve ser de fácil uso e sem complexidade para qualquer tipo de usuário conseguir registrar seu caso de forma rápida e fácil.

[RNF006] Disponibilidade.

O sistema deve estar disponível, ao mínimo, 99% do ano, para garantir que a satisfação do usuário seja atendida.

[RNF007] Implementação.

O sistema será desenvolvido usando as linguagens HTML, CSS, JAVASCRIPT, REACT, JAVA SPRING BOOT, SQL.

[RNF007] Manutenção.

O código-fonte deve ser bem documentado.

6. Modelos de casos de uso

6.1 Identificação dos atores e suas responsabilidades

Usuário fornecedor: responsável por inserir seus produtos dentro do sistema, aceitar ou recusar os pedidos efetuados pelo cliente, consultar seu ganho e solicitar o cancelamento de pedidos.

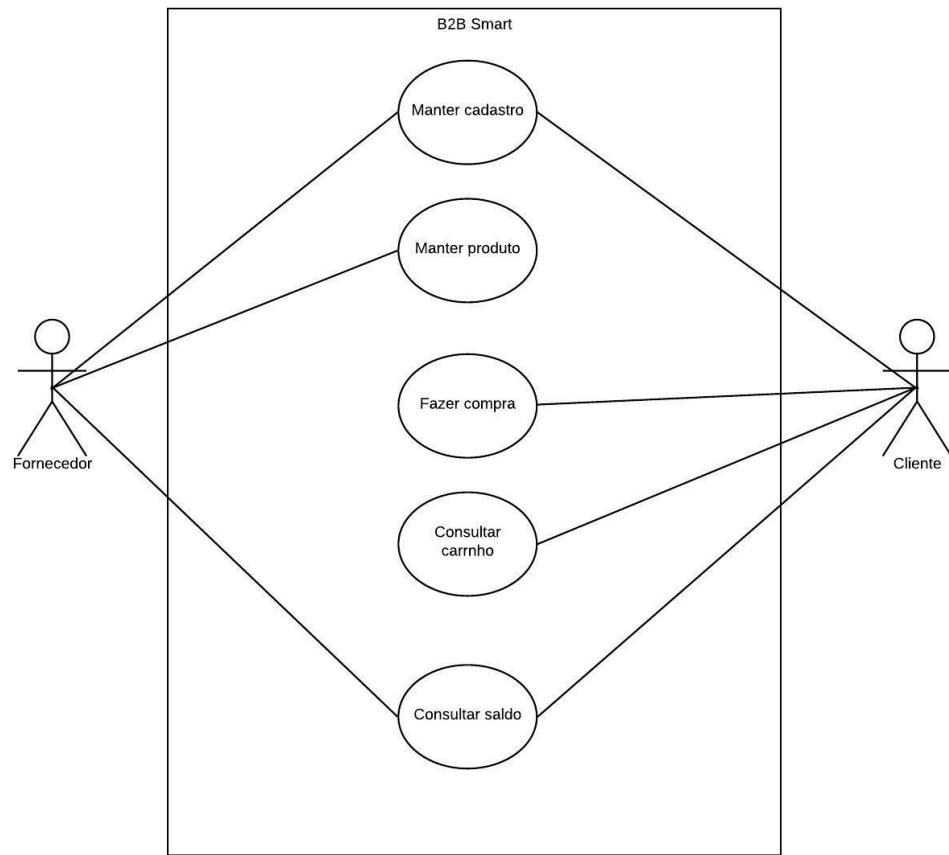
Usuário cliente: responsável por realizar os pedidos dos produtos de maneira direta com os fornecedores e consultar seus gastos e solicitar o cancelamento de pedidos.

Administrador: Responsável por realizar o cancelamento de pedidos de acordo com a solicitação dos usuários e dar orientação a novos usuários do sistema.

6.2 Definição de prioridade no desenvolvimento dos casos de uso

Número	Nome do caso de uso	Prioridade	Justificativa	Aluno responsável
UC01	Manter cadastro	Alta	Processo primário de negócio	Igor
UC02	Manter produto	Alta	Processo primário de negócio	Igor
UC03	Fazer compra	Alta	Processo primário de negócio	Igor
UC04	Consultar carrinho	Alta	Processo primário de negócio	Igor
UC05	Consultar saldo	Média	Processo primário de negócio	Gustavo
UC06	Cancelar compra	Alta	Processo primário de negócio	Kauã

6.3 Diagrama de caso de uso



6.4 Descritivo de caso de uso

Descritivo de caso de uso: UC01 Manter cadastro

Breve descrição

Este caso de uso tem por objetivo permitir que o usuário possa realizar seu cadastro no sistema.

Atores principais

Usuário Cliente, Usuário Fornecedor.

Fluxo básico

1. O usuário acessa a tela principal de login, com a possibilidade de realizar o cadastro.
2. O usuário preenche a informação de sua razão social.
3. O usuário preenche a informação de seu apelido.
4. O usuário preenche a informação de seu CNPJ.
5. O usuário preenche a informação de uma senha.
6. O usuário preenche a informação de seu endereço.
7. O usuário clica no botão para confirmar o cadastro.
8. Fim da execução do caso de uso.

Descritivo de caso de uso: UC02 Manter produto

Breve descrição

Este caso de uso tem por objetivo permitir que o usuário Fornecedor possa realizar cadastro de seus produtos no sistema.

Atores principais

Usuário Fornecedor.

Fluxo básico

1. O usuário Fornecedor acessa a tela de cadastro de produtos a partir da tela principal.
2. O usuário preenche a informação do nome do produto.
3. O usuário preenche a informação do valor unitário do produto.
4. O usuário preenche a informação da marca do produto.
5. O usuário clica no botão para confirmar o cadastro.
6. Fim da execução do caso de uso.

Descritivo de caso de uso: UC03 Fazer compra**Breve descrição**

Este caso de uso tem por objetivo permitir que o usuário Cliente possa realizar a compra dos produtos.

Atores principais

Usuário Cliente.

Fluxo básico

1. O usuário clica em um botão para abrir a tela de compra.
2. O usuário é apresentado à lista de fornecedores.
3. O usuário escolhe o fornecedor desejado.
4. O usuário seleciona os produtos do fornecedor escolhido.
5. O usuário indica a quantidade desejada dos produtos escolhidos.
6. Após o usuário selecionar os produtos ao clicar em um botão é enviado ao carrinho.
7. Fim da execução do caso de uso.

Descritivo de caso de uso: UC04 Cancelar compra**Breve descrição**

Este caso de uso tem por objetivo permitir que o Usuário Cliente ou Usuário Fornecedor possa solicitar o cancelamento de uma transação.

Atores principais

Usuário Cliente.

Fluxo básico

1. O usuário clica em um botão com o ícone de um carrinho para acessar a lista de produtos selecionados.
2. O usuário indica a forma de pagamento desejada.
3. O usuário ao clicar em um botão finaliza a venda.
4. Fim da execução do caso de uso.

Descritivo de caso de uso: UC05 Consultar saldo**Breve descrição**

Este caso de uso tem por objetivo permitir que o usuário Cliente e usuário fornecedor possa consultar o saldo de suas transações

Atores principais

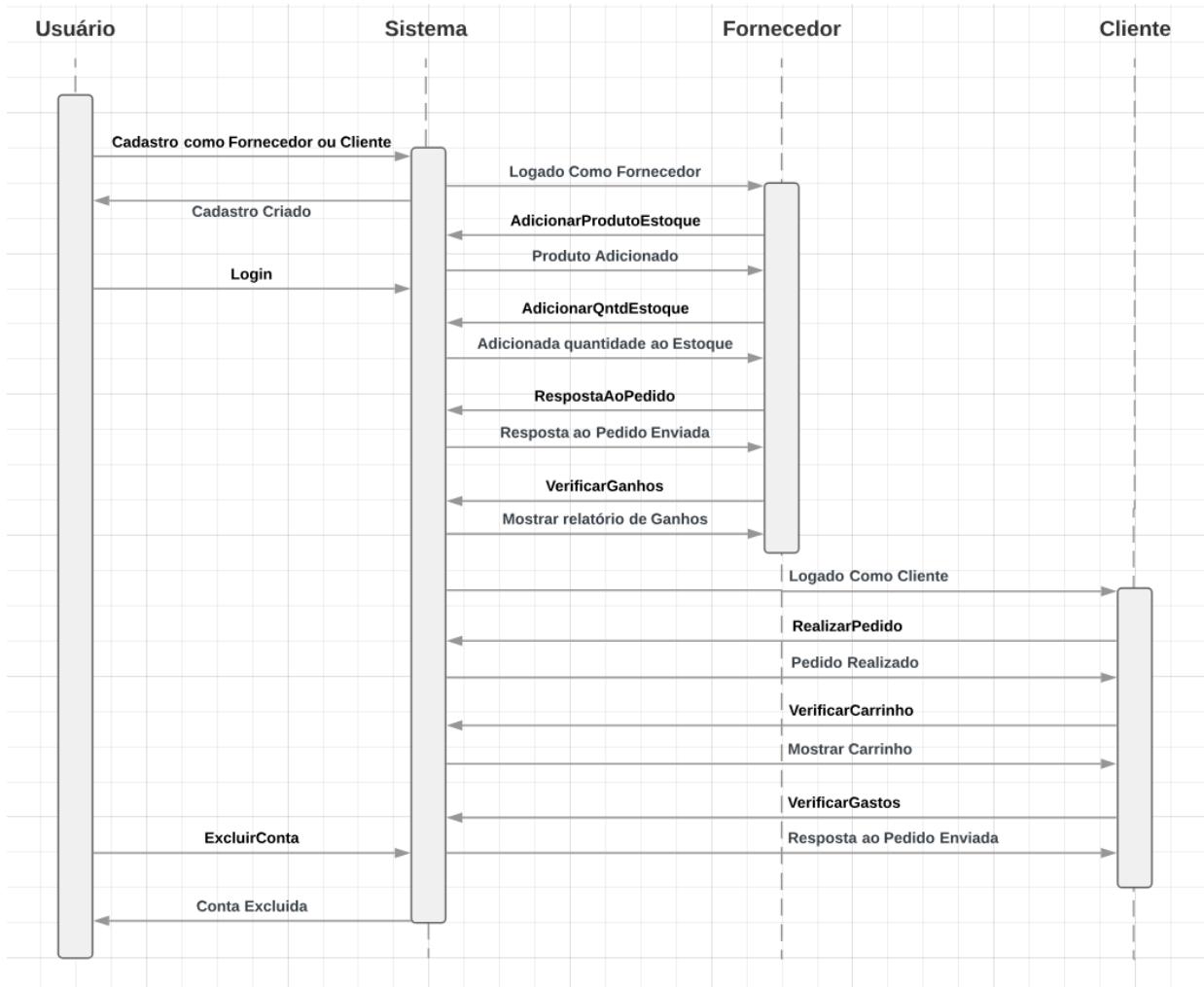
Usuário Cliente, Usuário fornecedor

Fluxo básico

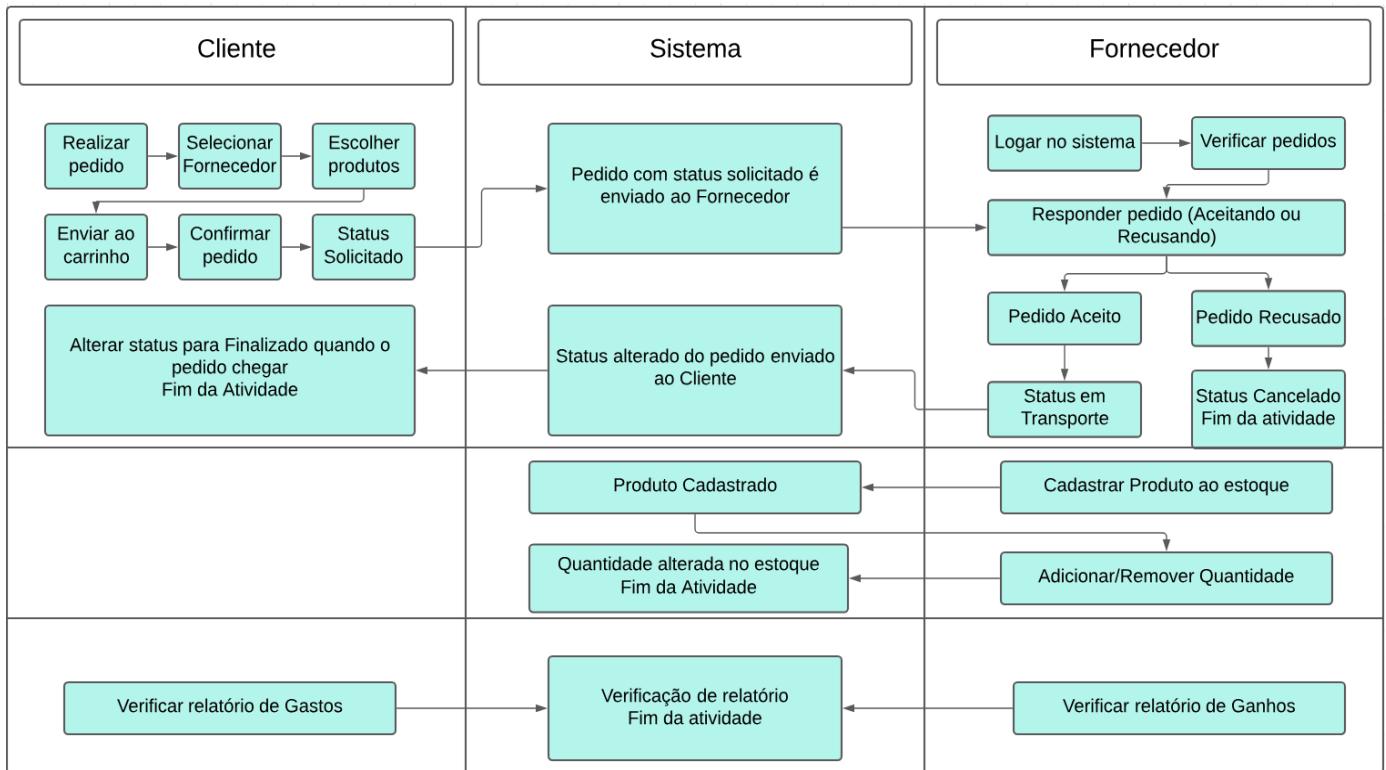
1. O usuário clica em um botão de nome “consultar saldo”
2. Abre-se a tela onde é apresentado as transações feitas pelo usuário
3. Fim da execução do caso de uso.

7. Diagramas

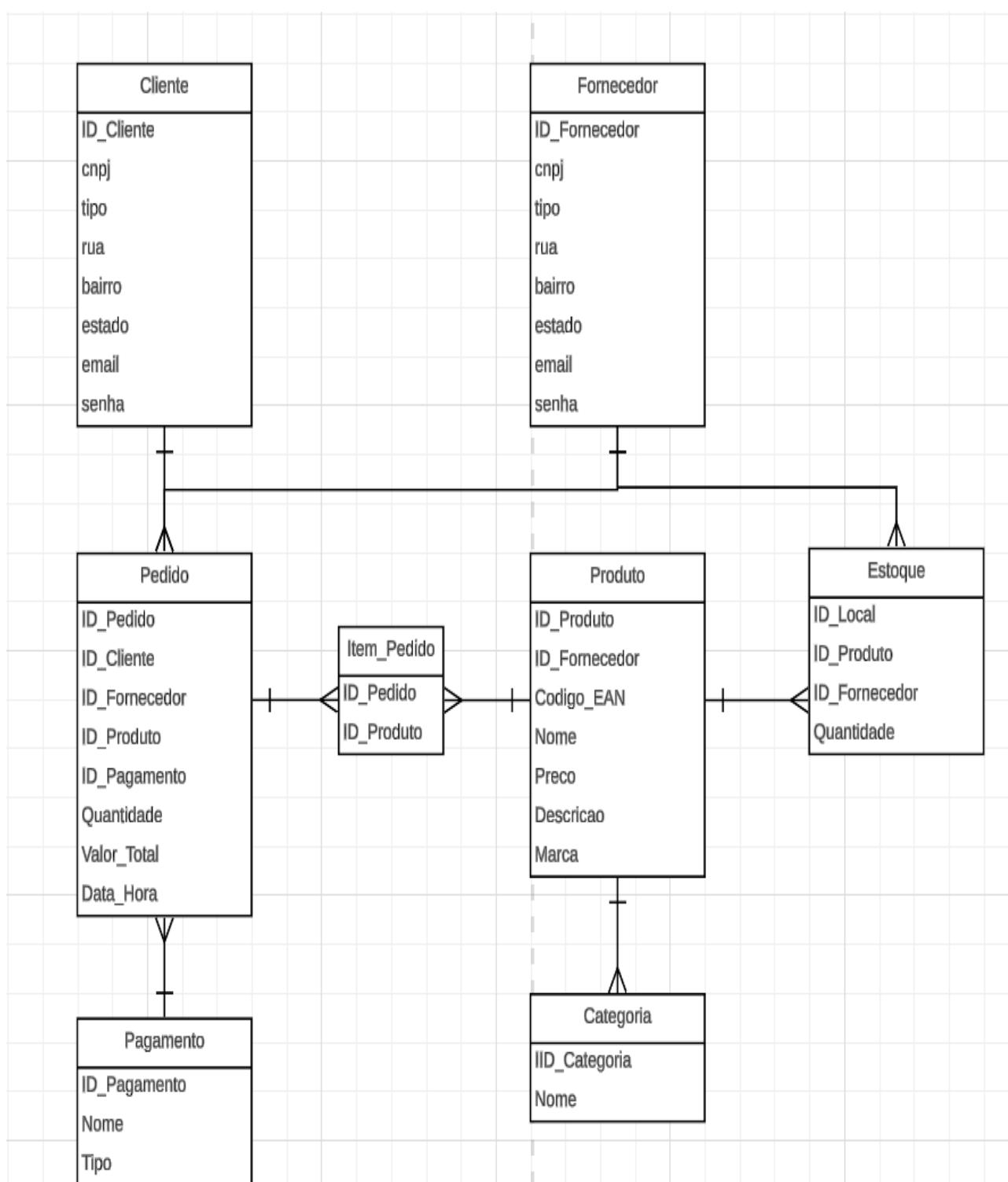
7.1 Diagrama de Sequência



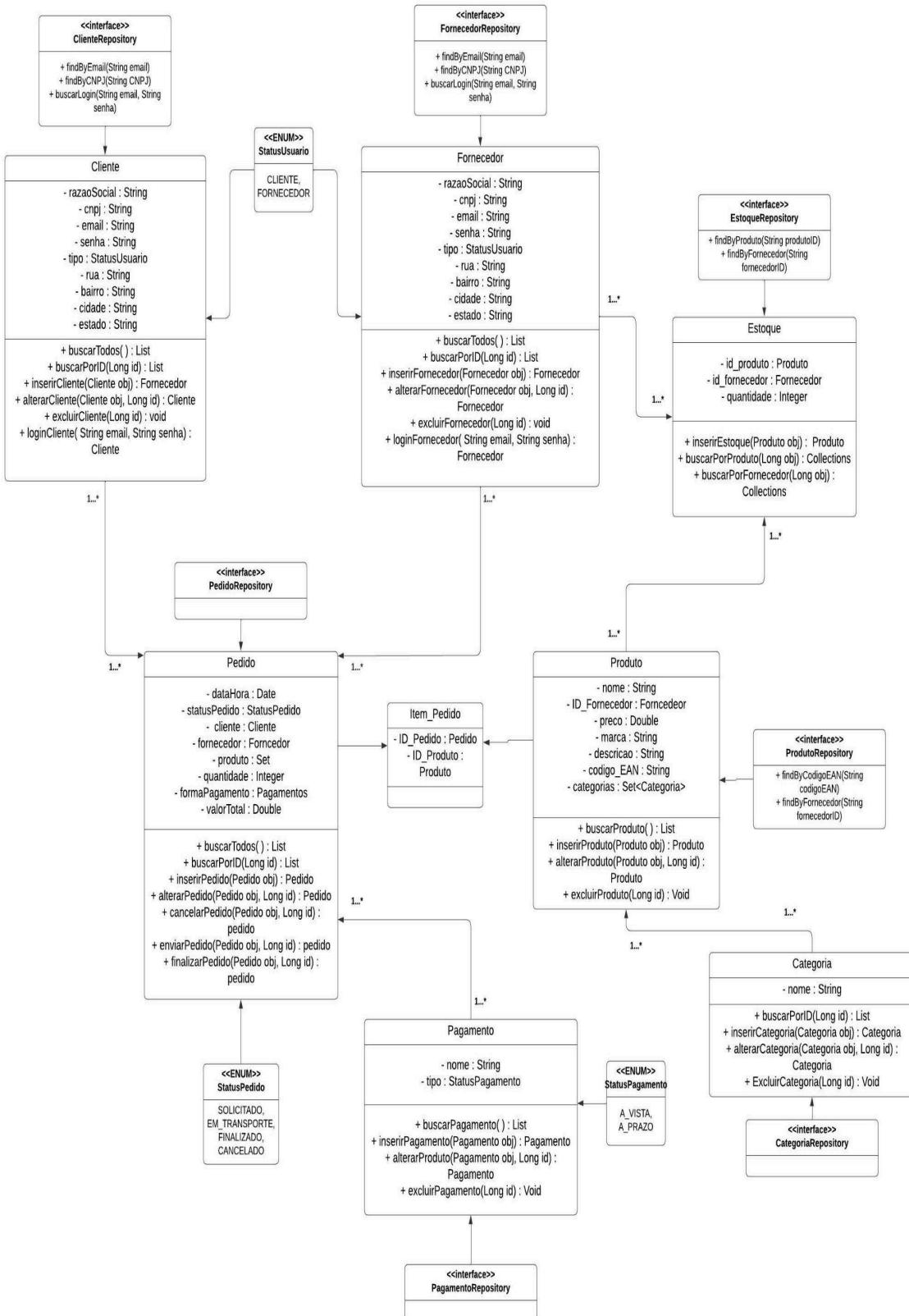
7.2 Diagrama de Atividades



7.3 Diagrama de Entidade e relacionamento



7.4 Diagrama de Classes



8. Metodologia

No desenvolvimento do trabalho, a equipe optou por dividir as tarefas para acelerar o desenvolvimento do front-end, back-end e documentação. Inicialmente, foi proposta a criação de um sistema de e-commerce, que foi aprimorado por toda a equipe responsável pelo desenvolvimento.

Adotamos uma metodologia baseada em etapas, onde cada fase é concluída antes de iniciar a próxima, com o trabalho sendo dividido em três blocos: front-end, back-end e documentação.

Inicialmente, o time se reuniu para definir todas as funcionalidades e processos que fariam parte do escopo do projeto. O front-end começou com a criação de uma sequência de telas no Figma, que foram desenvolvidas utilizando HTML, CSS e JavaScript, já a codificação foi realizada com suporte mútuo e sob supervisão para assegurar a integração com o back-end. As principais telas desenvolvidas seguem nas imagens abaixo.

Cadastro (onde será cadastrado o fornecedor ou cliente):

The screenshot shows a registration form titled 'Cadastro'. It includes fields for Razão Social, CNPJ, Email, Senha (password), Confirmar Senha (confirm password), and a radio button for 'Cliente' (Client). There is also a file upload field labeled 'Escolher arquivo' (Select file) with the note 'Nenhum arquivo escolhido' (No file selected). Below these are fields for Rua e Número, Cidade, Selecionar o Estado (Select state), and Bairro. At the bottom is a blue 'Cadastrar' (Register) button.

Login (onde o fornecedor ou cliente fará o login):

The screenshot shows a login form titled 'Login'. It has fields for Email and Senha (password), and a blue 'Entrar' (Enter) button at the bottom. Below the button is a link 'Não possui uma conta? Cadastre-se' (Don't have an account? Register).

Tela inicial (tela que irá abrir após login feito no sistema):

B2B Smart

Página Inicial | Lista de Produtos | Estoque de Produtos | Relatório de Lucro | Pedidos Solicitados | Carrinho

Bem-vindo à B2B Smart

Aqui você encontra os melhores produtos para o seu negócio. Navegue pelo nosso catálogo e descubra tudo que podemos oferecer para a sua empresa.

Contate-nos

© 2024 B2B Smart. Todos os direitos reservados.

Cadastro de produtos (produto novo a ser cadastrado | Exemplo fornecedor):

B2B Smart

Página Inicial | Lista de Produtos | Estoque de Produtos | Relatório de Lucro | Pedidos Solicitados | Carrinho

Lista de Produtos

Cadastrar Produto

Nome:

Marca:

Descrição:

Preço: R\$

EAN: 3242

Editor

Salvar

Editar produtos no estoque (editar quantidade de produtos no estoque | Exemplo fornecedor):

B2B Smart

Página Inicial | Lista de Produtos | Estoque de Produtos | Relatório de Lucro | Pedidos Solicitados | Carrinho

Estoque de Produtos

sucrilhos	bola	chuteira
Quantidade: 60 Editar	Quantidade em Estoque: 60 Salvar	Quantidade em Estoque: 0 Editar
bolacha	Quantidade em Estoque: 3 Editar	Quantidade em Estoque: 0 Editar
bolacha	Quantidade em Estoque: 0 Editar	

Editar Produto no Estoque

Nome: sucrilhos
Quantidade em Estoque: 60

Salvar

Estoque (produtos e suas quantidades cadastradas previamente | Exemplo fornecedor):

sucrilhos Quantidade em Estoque: 60 Editar	bola Quantidade em Estoque: 5 Editar	chuteira Quantidade em Estoque: 6 Editar
bolacha Quantidade em Estoque: 3 Editar	bolacha Quantidade em Estoque: 0 Editar	bolacha Quantidade em Estoque: 0 Editar

Listagem de produtos (produtos que foram cadastrados no estoque e suas informações completas | Exemplo fornecedor) :

bola Marca: penalty Descrição: bola redonda Preço: R\$4 EAN: 324234532 Editar Excluir Fazer Pedido	chuteira Marca: penalty Descrição: para chutar a bola Preço: R\$200 EAN: 345325 Editar Excluir Fazer Pedido	bolacha Marca: bolacha Descrição: 1 Preço: R\$1 EAN: 1 Editar Excluir Fazer Pedido
--	---	--

Relatório de valores (relatório de lucro para fornecedor e relatório de gastos para cliente):

Relatório de Lucro	
Relatório de Lucro	
bola Quantidade Vendida: 5 Lucro: R\$20.00	
chuteira Quantidade Vendida: 3 Lucro: R\$600.00	
Total Geral Quantidade Total Vendida: 8 Lucro Total: R\$620.00	

Pedidos solicitados (aba onde poderá verificar os pedidos que estão em andamento):

B2B Smart

Página Inicial | Lista de Produtos | Estoque de Produtos | Relatório de Lucro | Pedidos Solicitados | Carrinho

Pedidos Solicitados

bola Marca: penalty Descrição: bola redonda Preço: R\$4 EAN: 324234532 Quantidade: 5	chuteira Marca: penalty Descrição: para chutar a bola Preço: R\$200 EAN: 345325 Quantidade: 3
--	---

Carrinho (aba onde estarão os produtos selecionados no carrinho | Exemplo cliente):

B2B Smart

Página Inicial | Lista de Produtos | Estoque de Produtos | Relatório de Lucro | Pedidos Solicitados | Carrinho

Carrinho

bolacha Marca: bolacha Descrição: 1 Preço: R\$1 EAN: 1 Quantidade: 2
--

[Finalizar Compra](#)

Simultaneamente, o desenvolvimento do back-end foi iniciado com a criação de métodos e processos utilizando Java Spring Boot, dessa forma, a equipe trabalhou na modelagem e implementação das funcionalidades necessárias para o sistema. A codificação das principais funcionalidades e processos dentro do sistema seguem abaixo em blocos de códigos.

Dependências utilizadas:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>dom4j</groupId>
    <artifactId>dom4j</artifactId>
    <version>1.6.1</version>
</dependency>
```

Conexão com o banco de dados mysql:

```
1 spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/B2B
2 spring.datasource.username=root
3 spring.datasource.password=root
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.show-sql=true
6 spring.jpa.hibernate.ddl-auto=update
7 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
8
```

Serviço de Cadastro de Cliente e alguns tratamentos de repetição de dados:

```
// Método para inserir um novo usuário
public Cliente inserirUsuario(Cliente obj) throws Exception {
    try {
        // Verifica se já existe um usuário com o mesmo email
        if (repository.findByEmail(obj.getEmail()) != null) {
            throw new EmailExistsException("Já existe um email cadastrado para " + obj.getEmail() + " ");
        }
        // Verifica se já existe um usuário com o mesmo CNPJ
        if (repository.findByCNPJ(obj.getCNPJ()) != null) {
            throw new CnpjExistsException("Já existe um CNPJ cadastrado para " + obj.getCNPJ() + " ");
        }
        // Criptografa a senha antes de salvar
        obj.setSenha(Util.md5(obj.getSenha()));

    } catch (NoSuchAlgorithmException e) {
        // Exceção se houver erro na criptografia da senha
        throw new CriptoExistsException("Erro na criptografia da senha");
    }

    Cliente user = repository.saveAndFlush(obj);
    return user;
}
```

Controlador do método para cadastro de cliente:

```
// Endpoint para cadastrar um novo usuário
@PostMapping(value = "/cadastrar")
public ResponseEntity<String> inserirUsuario(@RequestBody Cliente obj) throws Exception {
    // Chama o serviço para inserir um novo usuário e retorna uma mensagem de sucesso
    obj = userService.inserirUsuario(obj);
    return ResponseEntity.ok("Usuario cadastrado com sucesso!");
}
```

Método de criptografia de senha:

```
public class Util {

    // Método voltado para realizar a criptografia da senha dentro do banco de dados
    public static String md5(String senha) throws NoSuchAlgorithmException {

        MessageDigest messagedig = MessageDigest.getInstance("MD5");
        BigInteger hash = new BigInteger(1, messagedig.digest(senha.getBytes()));
        return hash.toString(16);
    }
}
```

Atributos do objeto Pedido e suas relações:

```

@Entity
@Table(name = "pedido")
public class Pedido implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss'Z'", timezone = "GMT")
    private Instant dataHora;

    private StatusPedido statusPedido;

    @ManyToOne
    @JoinColumn(name = "id_cliente")
    private Cliente cliente;

    @ManyToOne
    @JoinColumn(name = "id_fornecedor")
    private Fornecedor fornecedor;

    @ManyToMany
    @JoinTable(name = "item_pedido",
    joinColumns = @JoinColumn(name = "Id_pedido"),
    inverseJoinColumns = @JoinColumn(name = "Id_produto"))
    private Set<Produto> produto = new HashSet<>();

    private Integer quantidade;

    @ManyToOne
    @JoinColumn(name = "id_pagamento")
    private Pagamento pagamento;

    private Double valorTotal;
}

```

Método de login de usuário:

```

public Cliente loginUsuario(String email, String senha) throws ServiceExc {
    // Busca o usuário pelo email e senha fornecidos
    Cliente userLogin = repository.buscarLogin(email, senha);
    return userLogin;
}

```

Métodos e Querys no repositório do cliente, para filtros específicos:

```

public interface ClienteRespository extends JpaRepository<Cliente, Long> {

    // Método para filtrar os usuários do tipo cliente pelo email na tabela User
    @Query("select i from Cliente i where i.email = :email")
    public Cliente findByEmail(String email);

    // Método para filtrar os usuários do tipo cliente pelo CNPJ na tabela User
    @Query("select j from Cliente j where j.CNPJ = :cnpj")
    public Cliente findByCNPJ(@Param("cnpj") String cnpj);

    // Método para buscar o login do usuário do tipo cliente baseado no email e senha fornecidos
    @Query("select k from Cliente k where k.email = :email and k.senha = :senha")
    public Cliente buscarLogin(@Param("email") String email, @Param("senha") String senha);
}

```

Método para alterar o status do pedido de “SOLICITADO” para “EM_TRÂNSITO” e alguns tratamentos, e ajuste de saldo no estoque:

```
public Pedido enviarPedido(Pedido obj, Long id) throws Exception {
    try {
        Pedido pedido = Repository.getReferenceById(id);

        if (pedido == null) {
            throw new ResourceNotFoundException("Pedido não encontrado para o ID fornecido: " + id);
        }

        if (pedido.getStatusPedido() == StatusPedido.EM_TRANSPORTE) {
            throw new StatusEnviadoException("Pedido já em transporte!");
        } else if (pedido.getStatusPedido() == StatusPedido.FINALIZADO) {
            throw new StatusEnviadoException("Pedido já finalizado!");
        } else if (pedido.getStatusPedido() == StatusPedido.CANCELADO) {
            throw new StatusEnviadoException("Pedido já cancelado");
        }

        // Aqui, em vez de passar o ID fornecido, buscamos o ID do produto no pedido
        Estoque estoque = estoqueRepository.findByProduto(objEstoque.getId_produto().getId());

        if (estoque == null) {
            throw new ResourceNotFoundException("Estoque não encontrado para o produto com ID: " + objEstoque.getId_produto().getId());
        }
        diminuirEstoque(estoque, obj);
        estoqueRepository.save(estoque);

        pedido.setStatusPedido(StatusPedido.EM_TRANSPORTE);
        Pedido pedidoEnviado = Repository.save(pedido);

        return pedidoEnviado;
    } catch (EntityNotFoundException e) {
        throw new ResourceNotFoundException("Pedido não encontrado para o ID fornecido: " + id);
    }
}
```

Classe ENUM Status do Pedido:

```
public enum StatusPedido {
    FINALIZADO("Finalizado", 1),
    CANCELADO("Cancelado", 2),
    EM_TRANSPORTE("Em transporte", 3),
    SOLICITADO("Solicitado", 4);

    private final String descricao;
    private final int valorInteiro;

    StatusPedido(String descricao, int valorInteiro) {
        this.descricao = descricao;
        this.valorInteiro = valorInteiro;
    }

    public String getStatus() {
        return descricao;
    }

    public int getValorInteiro() {
        return valorInteiro;
    }

    // Método para encontrar o enum a partir de um valor inteiro
    public static StatusPedido getByValorInteiro(int valorInteiro) {
        for (StatusPedido status : StatusPedido.values()) {
            if (status.valorInteiro == valorInteiro) {
                return status;
            }
        }
        throw new IllegalArgumentException("Nenhum StatusPedido encontrado com o valor inteiro: " + valorInteiro);
    }
}
```

Métodos que trabalham em conjunto para alterar o cadastro do usuário do tipo fornecedor:

```
// Método para alterar um usuário existente
public Fornecedor alterarUsuario(Fornecedor obj, Long id) throws Exception {
    Fornecedor usuario;
    try {
        // Busca o usuário pelo ID
        usuario = repository.findById(id).orElseThrow(() -> new ResourceNotFoundException(id));
        // Criptografa a senha antes de salvar
        obj.setSenha(Util.md5(obj.getSenha()));
        // Atualiza os dados do usuário
        updateData(usuario, obj);

    } catch (NoSuchAlgorithmException e) {
        // Exceção se houver erro na criptografia da senha
        throw new CriptoExistsException("Erro na criptografia da senha");
    }
    return repository.save(usuario);
}

// Método para atualizar os dados de um usuário
public void updateData(Fornecedor entity, Fornecedor obj) {
    entity.setRazaoSocial(obj.getRazaoSocial());
    entity.setCNPJ(obj.getCNPJ());
    entity.setEmail(obj.getEmail());
    entity.setSenha(obj.getSenha());
    entity.setTipo(obj.getTipo());
    entity.setRua(obj.getRua());
    entity.setEstado(obj.getEstado());
    entity.setBairro(obj.getBairro());
}
```

Objeto cliente:

```
private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
private String razaoSocial;
private String CNPJ;
private String email;
private String senha;
// Usando em ENUM para definir o tipo de usuário, ou FORNECEDOR ou CLIENTE
private StatusUsuario tipo;
private String rua;
private String estado;
private String bairro;

@JsonIgnore
@OneToMany(mappedBy = "cliente")
private List<Pedido> pedidos = new ArrayList<>();

public Cliente() {
}

public Cliente(Long id, String razaoSocial, String cnpj, String email, String senha, StatusUsuario tipo, String rua,
               String estado, String bairro, Date dataCriacao, Date dataAtualizacao) {
    super();
    this.id = id;
    this.razaoSocial = razaoSocial;
    this.CNPJ = cnpj;
    this.email = email;
    this.tipo = tipo;
    this.senha = senha;
    this.rua = rua;
    this.estado = estado;
    this.bairro = bairro;
}
```

Serviço para cadastro de um novo cliente com verificações de existência de dados:

```
// Método para inserir um novo usuário
public Cliente inserirUsuario(Cliente obj) throws Exception {
    try {
        // Verifica se já existe um usuário com o mesmo email
        if (repository.findByEmail(obj.getEmail()) != null) {
            throw new EmailExistsException("Já existe um email cadastrado para " + obj.getEmail() + " ");
        }
        // Verifica se já existe um usuário com o mesmo CNPJ
        if (repository.findByCNPJ(obj.getCPF()) != null) {
            throw new CnpjExistsException("Já existe um CNPJ cadastrado para " + obj.getCPF() + " ");
        }
        // Criptografa a senha antes de salvar
        obj.setSenha(Util.md5(obj.getSenha()));

    } catch (NoSuchAlgorithmException e) {
        // Exceção se houver erro na criptografia da senha
        throw new CriptoExistsException("Erro na criptografia da senha");
    }

    Cliente user = repository.saveAndFlush(obj);
    return user;
}
```

Método controller com a URL e parâmetros para o login do cliente:

```
// Endpoint para realizar login
@PostMapping(value = "/login")
public ResponseEntity<String> login(@Validated @RequestBody Cliente usuario, BindingResult br, HttpSession session)
    throws NoSuchAlgorithmException, ServiceExc {
    // Verifica se houve erros de validação nos campos do objeto Cliente
    if (br.hasErrors()) {
        // Retorna uma resposta com status 400 e uma mensagem de erro se houver erros de validação
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Erro de validação");
    }

    // Verifica se a senha do cliente não é nula
    if (usuario.getSenha() != null) {
        // Tenta realizar o login do cliente e retorna uma mensagem de sucesso se bem-sucedido
        Cliente userLogin = userService.loginUsuario(usuario.getEmail(), Util.md5(usuario.getSenha()));
        if (userLogin == null) {
            // Retorna uma resposta com status 404 se o cliente não for encontrado
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Usuário não encontrado. Tente novamente");
        } else {
            // Define o cliente logado na sessão e retorna uma mensagem de sucesso
            session.setAttribute("Usuário Logado", userLogin);
            return ResponseEntity.ok("Login realizado com sucesso");
        }
    } else {
        // Retorna uma resposta com status 400 se a senha do cliente for nula
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Senha não pode ser nula");
    }
}
```

Método responsável por realizar a criptografia da senha dentro do BD, visando segurança e integridade de informações:

```
// Método voltado para realizar a criptografia da senha dentro do banco de dados
public static String md5(String senha) throws NoSuchAlgorithmException {

    MessageDigest messagedig = MessageDigest.getInstance("MD5");
    BigInteger hash = new BigInteger(1, messagedig.digest(senha.getBytes()));
    return hash.toString(16);
}
```

Classe ENUM para gravar o tipo do usuário:

```
public enum StatusUsuario {
    //Criando os tipos pre determinados para realizar o cadastro
    CLIENTE("cliente"),
    FORNECEDOR("fornecedor");

    private String role;

    //Construtor para inserir as roles nas demais classes
    StatusUsuario(String role) {
        this.role = role;
    }

    //Metodo utilizado para pegar a role indicada
    public String getRole() {
        return role;
    }
}
```

Objeto responsável pela classe Produto com anotação associativa entre categoria e produto enviando os IDs de ambos para uma tabela "Categoria_Produto":

```
//anotation voltado para gerar o ID do produto
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

private String nome;
private BigDecimal preco;
private String marca;
private String descricao;
private String codigo_EAN;

@ManyToOne
@JoinColumn(name = "id_fornecedor")
private Fornecedor fornecedor;

@ManyToMany
@JoinTable(name = "CategoriaProduto",
joinColumns = @JoinColumn(name = "IdProduto"),
inverseJoinColumns = @JoinColumn(name = "IdCategoria"))
private Set<Categoria> categorias = new HashSet<>();

public Produto() {
}

public Produto(Long id, String nome, BigDecimal preco, String marca, String descricao,
Integer quantidadeInicial, Date dataCriacao, String codigo_EAN, Date dataAtualizacao, Fornecedor fornecedor) {
super();
this.id = id;
this.nome = nome;
this.preco = preco;
this.marca = marca;
this.descricao = descricao;
this.codigo_EAN = codigo_EAN;
this.fornecedor = fornecedor;
}
```

Método de cadastro de produto, com processo responsável por registrar o produto cadastrado na tabela de estoque e setar sua quantidade em 0:

```
// Método voltado para cadastros de novos produtos no BD
public Produto inserirProduto(Produto obj) throws Exception {
    Produto produtoNovo = productRepository.saveAndFlush(obj);

    // Criando um novo registro de estoque
    Estoque estoque = new Estoque();
    estoque.setid_produto(produtoNovo);
    estoque.setid_fornecedor(produtoNovo.getFornecedor()); // Supondo que o fornecedor do produto seja o mesmo do estoque
    estoque.setQuantidade(0); // Define a quantidade inicial como 0

    // Salvando o registro de estoque
    estoqueRepository.save(estoque);

    return produtoNovo;
}
```

Atributos do Objeto Pedido e suas associações:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss'Z'", timezone = "GMT")
private Instant dataHora;

private StatusPedido statusPedido;

@ManyToOne
@JoinColumn(name = "id_cliente")
private Cliente cliente;

@ManyToOne
@JoinColumn(name = "id_fornecedor")
private Fornecedor fornecedor;

@ManyToOne
@JoinColumn(name = "id_pagamento")
private Pagamento pagamento;

@OneToMany(mappedBy = "idPedido", cascade = CascadeType.ALL, orphanRemoval = true)
private Set<ItemPedido> itens = new HashSet<>();

@JoinColumn(name = "total_venda")
private BigDecimal totalVenda;
```

Método para inserir um novo pedido e fazer os respectivos cálculos de total dos itens e total da venda:

```
@Transactional
public Pedido inserirPedido(Pedido pedido) throws Exception {
    // Associar cada item ao pedido antes de salvar
    pedido.setStatusPedido(StatusPedido.SOLICITADO);
    for (ItemPedido item : pedido.getItens()) {
        item.setIdPedido(pedido);
    }

    // Calcular o valor total dos itens
    calcularValorTotalItens(pedido);

    // Calcular o total da venda
    calcularTotalVenda(pedido);

    // Salvar o pedido, que também salva os itens devido ao cascade
    Pedido pedidoNovo = pedidoRepository.save(pedido);

    return pedidoNovo;
}
```

Método responsável para fazer a troca dos itens dentro de um pedido já efetuado e fazer a troca de valores:

```
// Método para atualizar os itens de pedido
private void atualizarItensPedido(Pedido entity, Pedido obj) {
    // Lista de itens atualizados
    Set<ItemPedido> itensAtualizados = new HashSet<>();

    // Percorre os itens do objeto atualizado
    for (ItemPedido itemAtualizado : obj.getItens()) {
        // Busca pelo item correspondente no pedido original
        ItemPedido itemExistente = entity.getItens().stream()
            .filter(item -> item.getIdLocal().equals(itemAtualizado.getIdLocal())).findFirst().orElse(null);

        if (itemExistente != null) {
            // Atualiza os dados do item existente
            itemExistente.setQuantidade(itemAtualizado.getQuantidade());
            itemExistente.setValorTotal(itemAtualizado.getValorTotal());
            itensAtualizados.add(itemExistente);
        } else {
            // Se não encontrar o item, adiciona o novo item
            itemAtualizado.setIdPedido(entity);
            itensAtualizados.add(itemAtualizado);
        }
    }

    // Remove os itens que não estão mais presentes no objeto atualizado
    entity.getItens().removeIf(item -> !itensAtualizados.contains(item));
    entity.getItens().addAll(itensAtualizados);
}
```

Método responsável para enviar um pedido com o status "SOLICITADO", tratamentos de status para trocar para "EM_TRANSPORTE" apenas pedidos com o status "SOLICITADO" e verificar a quantidade do produto dentro do estoque, caso a quantidade presente no estoque seja menor que a quantidade dentro do pedido, o status não é alterado e é alertado a falta de produto dentro do estoque:

```
public Pedido enviarPedido(Pedido obj, Long id) throws Exception {
    try {
        // Obtém a referência do pedido pelo ID
        Pedido pedido = Repository.getReferenceById(id);

        // Verifica se o pedido existe
        if (pedido == null) {
            // Lança uma exceção se o pedido não for encontrado
            throw new ResourceNotFoundException("Pedido não encontrado para o ID fornecido: " + id);
        }
        // Verifica o status atual do pedido
        if (pedido.getStatusPedido() == StatusPedido.EM_TRANSPORTE) {
            // Lança uma exceção se o pedido já estiver em transporte
            throw new StatusEnviadoException("Pedido já em transporte!");
        } else if (pedido.getStatusPedido() == StatusPedido.FINALIZADO) {
            // Lança uma exceção se o pedido já estiver finalizado
            throw new StatusEnviadoException("Pedido já finalizado!");
        } else if (pedido.getStatusPedido() == StatusPedido.CANCELADO) {
            // Lança uma exceção se o pedido já estiver cancelado
            throw new StatusEnviadoException("Pedido já cancelado");
        }
        // Atualiza o estoque antes de enviar o pedido
        for (ItemPedido itemPedido : pedido.getItens()) {
            // Busca o estoque do produto
            Estoque estoque = estoqueRespository.findByProduto(itemPedido.getIdProduto().getId());
            // Verifica se há quantidade suficiente no estoque
            if (itemPedido.getQuantidade() > estoque.getQuantidade()) {
                // Lança uma exceção se o estoque for insuficiente
                throw new Exception(
                    "Estoque insuficiente para o produto com ID: " + itemPedido.getIdProduto().getId());
            }
            // Atualiza o estoque com a quantidade do item pedido
            estoqueService.atualizarEstoque(itemPedido.getIdProduto().getId(), itemPedido.getQuantidade());
        }
        // Define o status do pedido para "EM TRANSPORTE"
        pedido.setStatusPedido(StatusPedido.EM_TRANSPORTE);
        // Salva o pedido atualizado
        Pedido pedidoEnviado = Repository.save(pedido);
        // Retorna o pedido enviado
        return pedidoEnviado;
    } catch (EntityNotFoundException e) {
        // Lança uma exceção se o pedido não for encontrado
        throw new ResourceNotFoundException("Pedido não encontrado para o ID fornecido: " + id);
    }
}
```

Objeto ItemPedido, uma espécie de carrinho que guarda as informações do produto a ser atributo dentro do pedido, inserindo sua quantidade e calculando o seu valor total mediante ao valor unitário cadastrado dentro do objeto produto:

```
@Id  
@GeneratedValue(strategy = GenerationType.AUTO)  
private Long idLocal;  
  
@ManyToOne  
@JoinColumn(name = "id_pedido")  
@JsonIgnore  
private Pedido idPedido;  
  
@ManyToOne  
@JoinColumn(name = "id_produto")  
private Produto idProduto;  
  
private Integer quantidade;  
  
@Column(precision = 8, scale = 2)  
private BigDecimal valorTotal;  
  
public ItemPedido() {  
}  
  
public ItemPedido(Long idLocal, Pedido idPedido, Produto idProduto, Integer quantidade, BigDecimal valorTotal) {  
    super();  
    this.idLocal = idLocal;  
    this.idPedido = idPedido;  
    this.idProduto = idProduto;  
    this.quantidade = quantidade;  
    this.valorTotal = valorTotal;  
}
```

Objeto estoque e suas associações:

```
@Component  
@Entity  
@Table(name = "estoque")  
public class Estoque {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id_local;  
  
    @ManyToOne  
    private Produto id_produto;  
    @ManyToOne  
    private Fornecedor id_fornecedor;  
  
    private Integer quantidade;  
  
    public Estoque() {  
    }  
  
    public Estoque(Long id_local, Produto id_produto, Fornecedor id_fornecedor, Integer quantidade) {  
        super();  
        this.id_local = id_local;  
        this.id_produto = id_produto;  
        this.id_fornecedor = id_fornecedor;  
        this.quantidade = quantidade;  
    }  
}
```

Métodos que trabalham em conjunto para atualizar a quantidade do produto no estoque:

```
// Método para inserir um novo estoque ou atualizar a quantidade do estoque
public Estoque inserirEstoque(Long id, Estoque obj) throws Exception{
    Estoque estoque;
    try {
        estoque = estoqueRepository.findByProduto(id);
        updateData(estoque, obj);
    } catch (EntityNotFoundException e) {
        throw new ResourceNotFoundException(id);
    }
    return estoqueRepository.save(estoque);
}

// Método para atualizar a quantidade do estoque
public void updateData(Estoque entity, Estoque obj) {
    entity.setQuantidade(entity.getQuantidade() + obj.getQuantidade());
}

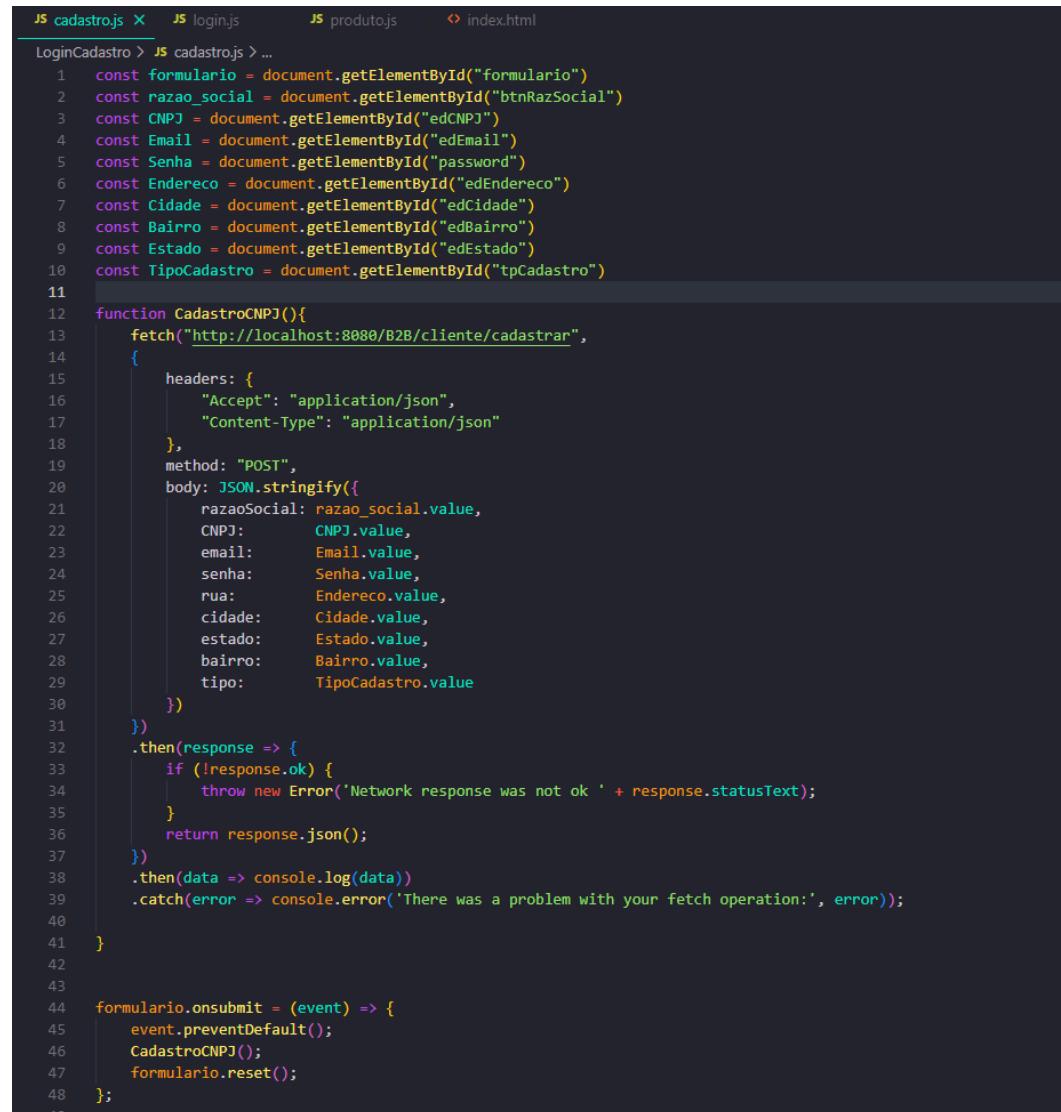
public void atualizarEstoque(Long idProduto, Integer quantidade) {
    // Busca o produto no estoque pelo id
    Estoque estoque = estoqueRepository.findByProduto(idProduto);

    // Atualiza a quantidade do produto no estoque
    estoque.setQuantidade(estoque.getQuantidade() - quantidade);

    // Salva as alterações no banco de dados
    estoqueRepository.save(estoque);
}
```

Partindo para a conexão entre front end e back end, abaixo seguem a codificação da conexão das telas de cadastro e login.

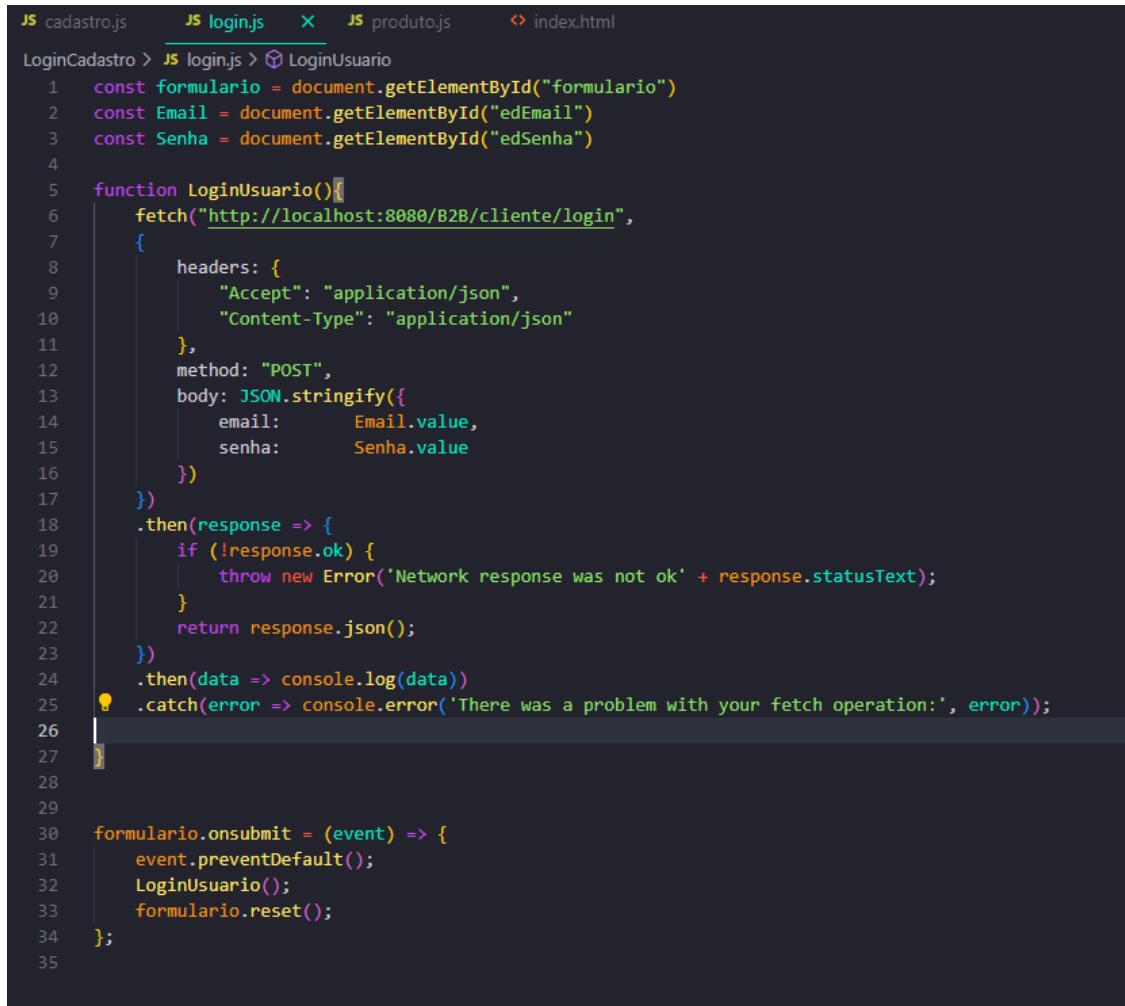
Cadastro:



```
JS cadastro.js X JS login.js JS produto.js index.html
LoginCadastro > JS cadastro.js > ...
1 const formulario = document.getElementById("formulario")
2 const razao_social = document.getElementById("btnRazSocial")
3 const CNPJ = document.getElementById("edCNPJ")
4 const Email = document.getElementById("edEmail")
5 const Senha = document.getElementById("password")
6 const Endereco = document.getElementById("edEndereco")
7 const Cidade = document.getElementById("edCidade")
8 const Bairro = document.getElementById("edBairro")
9 const Estado = document.getElementById("edEstado")
10 const TipoCadastro = document.getElementById("tpCadastro")

11
12 function CadastroCNPJ(){
13     fetch("http://localhost:8080/B2B/cliente/cadastrar",
14     {
15         headers: {
16             "Accept": "application/json",
17             "Content-Type": "application/json"
18         },
19         method: "POST",
20         body: JSON.stringify({
21             razaoSocial: razao_social.value,
22             CNPJ: CNPJ.value,
23             email: Email.value,
24             senha: Senha.value,
25             rua: Endereco.value,
26             cidade: Cidade.value,
27             estado: Estado.value,
28             bairro: Bairro.value,
29             tipo: TipoCadastro.value
30         })
31     })
32     .then(response => {
33         if (!response.ok) {
34             throw new Error('Network response was not ok ' + response.statusText);
35         }
36         return response.json();
37     })
38     .then(data => console.log(data))
39     .catch(error => console.error('There was a problem with your fetch operation:', error));
40
41 }
42
43
44 formulario.onsubmit = (event) => {
45     event.preventDefault();
46     CadastroCNPJ();
47     formulario.reset();
48 };
49
```

Login:



```
JS cadastro.js      JS login.js  X  JS produto.js    index.html
LoginCadastro > JS login.js > LoginUsuario
1  const formulario = document.getElementById("formulario")
2  const Email = document.getElementById("edEmail")
3  const Senha = document.getElementById("edSenha")
4
5  function LoginUsuario(){
6      fetch("http://localhost:8080/B2B/cliente/login",
7      {
8          headers: {
9              "Accept": "application/json",
10             "Content-Type": "application/json"
11         },
12         method: "POST",
13         body: JSON.stringify({
14             email: Email.value,
15             senha: Senha.value
16         })
17     })
18     .then(response => {
19         if (!response.ok) {
20             throw new Error('Network response was not ok' + response.statusText);
21         }
22         return response.json();
23     })
24     .then(data => console.log(data))
25     .catch(error => console.error('There was a problem with your fetch operation:', error));
26
27 }
28
29
30 formulario.onsubmit = (event) => {
31     event.preventDefault();
32     LoginUsuario();
33     formulario.reset();
34 };
35
```

A documentação do projeto também começou juntamente com os outros processos. Foi realizada uma série de reuniões para definir e refinar os requisitos do sistema, garantindo que todas as necessidades fossem bem especificadas e documentadas.

9. Projetos futuros

Sobre nosso sistema, pensamos em fazer inovações no futuro, porém por conta da falta de tempo e da demora do nosso grupo para mapear todos os processos e funcionalidades que contemplariam o trabalho de, não conseguimos aplicá-las nesse projeto.

Algumas das futuras adições e alterações que faremos são:

Criação de uma aba de estoque para os clientes. (Atualmente só temos para os fornecedores);

Criação de uma API que direciona um usuário a se comunicar com outro colaborador por meio de um botão com a logo do outlook.

Um relatório que mostra os produtos mais vendidos por fornecedor e outro para os mais comprados por cliente.

10. Referências bibliográficas

Para criação do front end:

PAIXÃO, Danilo. Crie um site para Ecommerce com HTML CSS E JS| Iniciante, <https://www.youtube.com/watch?v=gB1Rvs1OPIc>, <https://stackoverflow.com>, março de 2024.

BATTISTI, Matheus. 20+ Projetos de JavaScript - Aprenda HTML, CSS e JavaScript, <https://iqvia.udemy.com/course/20-projetos-de-javascript-aprenda-html-css-e-javascript/>, março de 2024.

Para criação do back end:

ALVES, Nélio. Java COMPLETO Programação Orientada Objetos + Projetos. <https://www.udemy.com/course/java-curso-completo/?couponCode=LEADERSALE24A>, abril de 2024.

ALVES, Nélio. Microsserviços Java com Spring Boot e Spring Cloud. <https://www.udemy.com/course/microsservicos-java-spring-cloud/?couponCode=LEADERSALE24A>, abril de 2024.

CARDOSO, Frank. Desenvolvimento de uma loja virtual com Spring, React e Next js Apresentação, https://www.youtube.com/watch?v=K8sh4xOZ42M&list=PLoIFXY7ye0lO33-laf_bH-NyuvRQt4l0H, abril de 2024.

EXCHANGE, Stack. StackOverflow, <https://stackoverflow.com>, abril de 2024.

AI, Open. ChatGPT, chat.openai.com, março de 2024.

Para conexão entre ambos:

MOTO, Code. Projeto Web API Java Spring Boot - 7 (Conectando Front/Back). <https://www.youtube.com/watch?v=hsewph3Xphw>, abril de 2024.

ANGELO, Lucas. Aula 15 - Criando frontend HTML e JS da aplicação e buscando dados da API Spring Boot (Java Web), <https://www.youtube.com/watch?v=gnslbzh10Bw>, abril de 2024.

AWARI. Exemplo De Frontend Com Spring Boot: Aprenda Na Prática, https://awari.com.br/exemplo-de-frontend-com-spring-boot-aprenda-na-pratica/?utm_source=blog&utm_campaign=projeto+blog&utm_medium=Exemplo%20De%20Frontend%20Com%20Spring%20Boot:%20Aprenda%20Na%20Prática, abril de 2024.