Bravi Distribuidora

Thomaz Lima, Sofia Saraiva

Departamento de Ciência da Computação- Centro de Estudos e Sistemas Avançados do Recife - C.E.S.A.R - Recife - PE - Brasil.

{trl, spscl}@cesar.school

Sumário:

Minimundo

Este sistema é projetado para gerenciar as operações comerciais e administrativas de uma empresa atuante na distribuição de produtos. Ele abrange diversas entidades que interagem entre si, com foco nos funcionários, clientes, fornecedores, produtos, estoque, compras e fornecimentos. A seguir, detalhamos as entidades e seus respectivos relacionamentos.

Entidades

Funcionários

Os funcionários são cadastrados com seus dados pessoais e informações de trabalho. Cada funcionário está vinculado a um setor específico e pode ser registrado como usuário do sistema. O CPF do funcionário é utilizado como nome de usuário, e o nome serve como senha inicial. Seguem os atributos:

ID único (id_funcionario), CPF, Nome, Data de Nascimento, Endereço (Rua, Bairro, CEP, Número), Cargo, Setor, CPF do gerente (caso o funcionário tenha um gerente)

Relacionamentos

- Cada funcionário pertence a um único setor, registrado na tabela Setor.
- Cada funcionário pode ter um gerente, representado pelo CPF do gerente na tabela Funcionario.
- Cada funcionário é um usuário do sistema, registrado na tabela Usuario.

Usuários

Os funcionários do sistema são registrados como usuários, com atributos como:

id (identificador único), usuário, senha, fk_Funcionario_CPF (referência ao funcionário), isGerente (indica se o usuário é um gerente).

Relacionamentos:

- Cada usuário está vinculado a um único funcionário, através do CPF. A tabela Usuario permite identificar gerentes e outros colaboradores.

Fornecedores

A Bravi lida com três tipos de fornecedores, classificados conforme a natureza da parceria comercial. Seguem os atributos:

ID único (id_fornecedor), Categoria (Diferente da Categoria de Produto), CNPJ, Inscrição Estadual, Endereço (Rua, Bairro, CEP, Número), Razão Social.

Fornecimentos

Os fornecimentos de produtos pelos fornecedores são registrados para controle de estoque e movimentações financeiras. Os atributos são:

- fk_Produto_NSM, fk_Fornecedor_CNPJ, id, valor.

Relacionamentos:

- Cada fornecimento está associado a um fornecedor, nota e a um produto específico.

Clientes

A Bravi mantém um cadastro detalhado de seus clientes. A tabela de clientes possui os seguintes atributos:

CNPJ, Nome, Rua, Bairro, CEP, Número, Inscrição Estadual, Razão Social.

Relacionamentos:

- Cada cliente pode realizar compras, que são registradas na tabela _Compra.

Compras

As compras realizadas pelos clientes são registradas para controle de vendas e estoque. Os atributos são:

fk_Cliente_CNPJ, fk_Produto_NSM, id, valor.

Relacionamentos:

- As compras estão associadas aos clientes na tabela Cliente.
- As compras estão associadas aos produtos através da tabela _Compra e as datas através da tabela notas.

Notas Fiscais

As notas fiscais registram todas as transações comerciais, sendo usadas para controle financeiro, contábil e tributário. Com a junção das tabelas Nota_in e Nota_out, o registro agora é feito na tabela Nota, com os seguintes atributos:

id único, fk_Compra_id (para registros de entrada), fk_Fornece_id (para registros de saída), is_in (indica se é uma entrada ou saída)

Relacionamentos

- As notas fiscais estão associadas a compras (tabela Nota, com is_in = TRUE) e fornecimentos (tabela Nota, com is_in = FALSE), controlando os movimentos de estoque e financeiros.

Produtos

A Bravi mantém um catálogo de produtos com informações detalhadas para controle de estoque e vendas. Cada produto possui uma categoria associada. Seguem os atributos:

NSM (Número de Série do Produto), Nome, Descrição, e fk Categoria_id (chave estrangeira que referencia a categoria).

Relacionamentos:

- Cada produto está associado a uma categoria através da tabela Categoria Produto.
- Os produtos podem ser comprados ou fornecidos, como indicado nas tabelas Compra e Fornece.
- Os produtos são associados a tabela estoque, que diz onde estão armazenados e quantos ainda restam.

Categorias de Produto

Os produtos são classificados em categorias para facilitar a organização e o gerenciamento. Seguem os atributos:

fk_Produto_NSM, fk_Categoria_id.

Relacionamentos:

- Uma categoria pode ter vários produtos, representado pela tabela Categoria Produto.

Estoque

O estoque é gerido com base nas entradas e saídas e romaneios de carga. Cada produto está vinculado a entradas e saídas financeiras. Seguem os atributos:

Fk de Setor, quantidade, Fk de Produto.

Relacionamentos:

- Cada produto no estoque está vinculado a movimentações de contas a pagar e a receber, através das tabelas Estoque e Nota (tanto de entrada quanto de saída).
- Cada Estoque está relacionado a um setor.

Setor

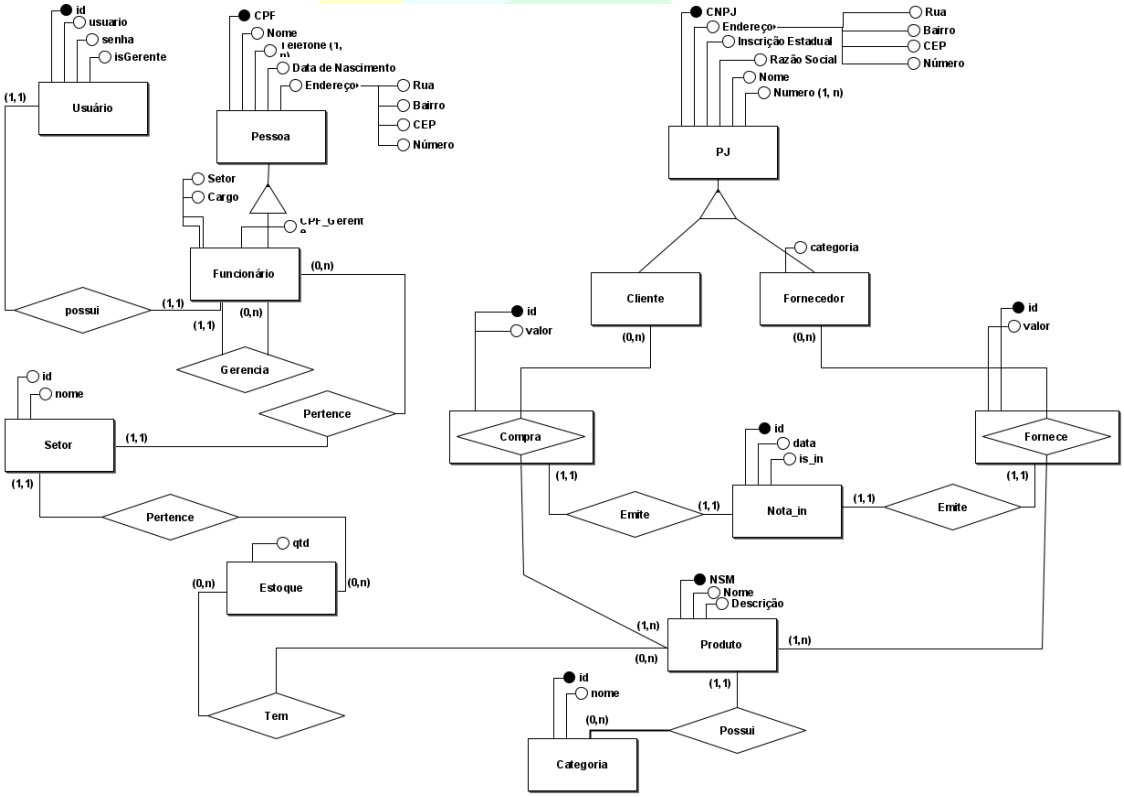
O setor representa uma divisão interna dentro da empresa, responsável por agrupar os funcionários de acordo com suas funções específicas. Cada funcionário está vinculado a um setor, e esse vínculo é essencial para o controle organizacional, administrativo e operacional da empresa. A tabela Setor armazena informações sobre cada divisão, como o nome do setor. A seguir, detalhamos os atributos e relacionamentos envolvidos:

ID único, Nome

Relacionamentos:

- Cada funcionário está vinculado a um único setor, representado pela chave estrangeira fk_Setor_id na tabela Funcionario.
- Cada produto no estoque pode estar vinculado a um setor específico, registrado na tabela Estoque. A tabela Estoque gerencia as quantidades de produtos disponíveis em cada setor, permitindo um controle eficiente de inventário e movimentações de estoque.

Modelo Conceitual



O modelo conceitual do banco foi elaborado para mapear os elementos essenciais e os relacionamentos que suportam a gestão de produtos de higiene, limpeza, descartáveis, maquinários e EPIs, incluindo uma linha sustentável. Utilizando um Diagrama Entidade-Relacionamento (DER), foram identificadas entidades como clientes, produtos, pedidos e fornecedores, junto com seus atributos e relacionamentos, visando refletir as operações da empresa.

Para realizar esse modelo, fizemos o levantamento de requisitos por meio de entrevistas e análise dos processos internos; identificação de entidades e atributos relevantes, como descrição de produtos e valores de pedidos; e definição de relacionamentos, incluindo vendas e reposições. O diagrama foi validado com usuários para garantir aderência às necessidades do negócio e normalizado para evitar redundâncias e inconsistências.

Requisitos de BD

Ter no mínimo 8 entidades:
Cumprido:

- Existem 12 tabelas no banco de dados: Categoria, Funcionario, Fornecedor, Cliente, Produto, _Fornece, _Compra, Nota, Categoria_Produto, Estoque, Setor e Usuario.

Atributo composto:
Cumprido:

- O endereço é um atributo composto nas tabelas Funcionario, Fornecedor e Cliente, composto por Rua, Bairro, CEP, Número.

```sql
CREATE TABLE IF NOT EXISTS Funcionario (
    fk_Setor_id INTEGER,
    Cargo VARCHAR(50),
    CPF VARCHAR(14) PRIMARY KEY,
    Nome VARCHAR(50),
    Data_de_Nascimento DATE,
    Rua VARCHAR(50),
    Bairro VARCHAR(50),
    CEP VARCHAR(10),
    Numero VARCHAR(50),
    CPF_GERENTE VARCHAR(14),
);
```

Atributo multivalorado:
Cumprido:

- O atributo Numero (Telefone) é multivalorado nas tabelas Cliente e Fornecedor.

```sql
CREATE TABLE IF NOT EXISTS Fornecedor (
    categoria VARCHAR(50),
    CNPJ VARCHAR(20) PRIMARY KEY,
    Nome VARCHAR(50),
    Rua VARCHAR(50),
    Bairro VARCHAR(50),
    CEP VARCHAR(10),
    Numero INTEGER,
    Numero2 INTEGER,
    Inscricao_Estadual VARCHAR(50),
    Razao_Social VARCHAR(50)
);
```

```sql
CREATE TABLE IF NOT EXISTS Cliente (
    CNPJ VARCHAR(20) PRIMARY KEY,
    Nome VARCHAR(50),
    Rua VARCHAR(50),
    Bairro VARCHAR(50),
    CEP VARCHAR(10),
    Numero INTEGER,
    Numero2 INTEGER,
    Inscricao_Estadual INTEGER,
    Razao_Social VARCHAR(50)
);
```

Cardinalidades diversas:
Cumprido:

- Relacionamento entre Fornecedor e Produto (muitos para muitos via Fornece).
- Relacionamento entre Cliente e Produto (muitos para muitos via Compra).
- Relacionamento entre Funcionário e Funcionário (auto-relacionamento, gerente e subordinado).

Atributo de relacionamento:
Cumprido: A tabela _Fornece ilustra um relacionamento entre as entidades Produto e Fornecedor. Ela possui duas chaves estrangeiras (fk_Produto_NSM e fk_Fornecedor_CNPJ) que referenciam, respectivamente, as tabelas Produto e Fornecedor.

```sql
CREATE TABLE IF NOT EXISTS _Fornece (
    fk_Produto_NSM INTEGER,
    fk_Fornecedor_CNPJ VARCHAR(20),
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    valor DECIMAL(10, 2),
    FOREIGN KEY (fk_Produto_NSM) REFERENCES Produto (NSM),
    FOREIGN KEY (fk_Fornecedor_CNPJ) REFERENCES Fornecedor (CNPJ)
);
```

Auto-relacionamento:
Cumprido:

- A tabela Funcionario tem um auto-relacionamento, onde um Funcionario pode ser gerente de outro Funcionario (via CPF_GERENTE).

```
CPF_GERENTE VARCHAR(14),
FOREIGN KEY (fk_Setor_id) REFERENCES Setor(id),
FOREIGN KEY (CPF_GERENTE) REFERENCES Funcionario(CPF)
```

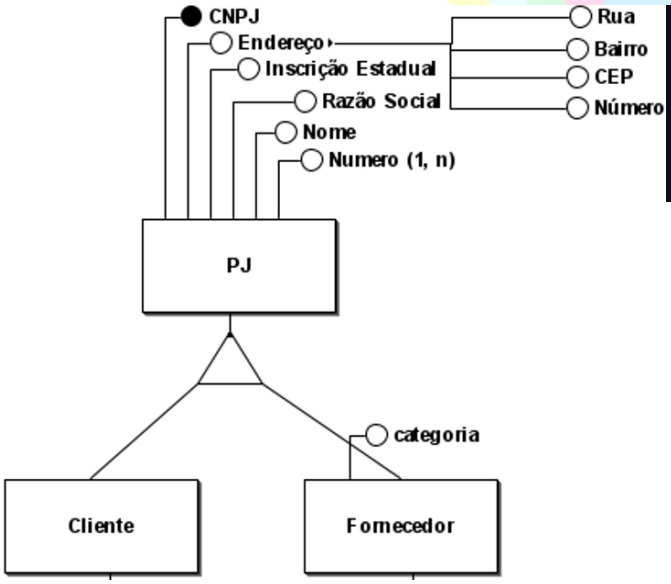
Agregação ou ternário:
Cumprido:

- A relação Compra possui atributos próprios, como id e valor, indicando que essa relação encapsula informações específicas da transação, além de conectar Cliente e Fornecedor.

Entidade fraca:
Cumprido:

- A categoria é uma entidade dependente, ou seja, ela depende do produto para ser identificada e não pode existir sem um produto associado. Nesse contexto, a categoria de um produto é uma entidade fraca, pois sua identificação e existência estão diretamente vinculadas ao produto, formando uma relação de dependência entre elas.

Herança:
Cumprido: As tabelas Fornecedor e Cliente herdam de PJ e a tabela funcionário herda de pessoa.




```
public class Fornecedor extends PJ {

    private String categoria;
    private String inscricaoEstadual;
```

Trigger:
Cumprido:

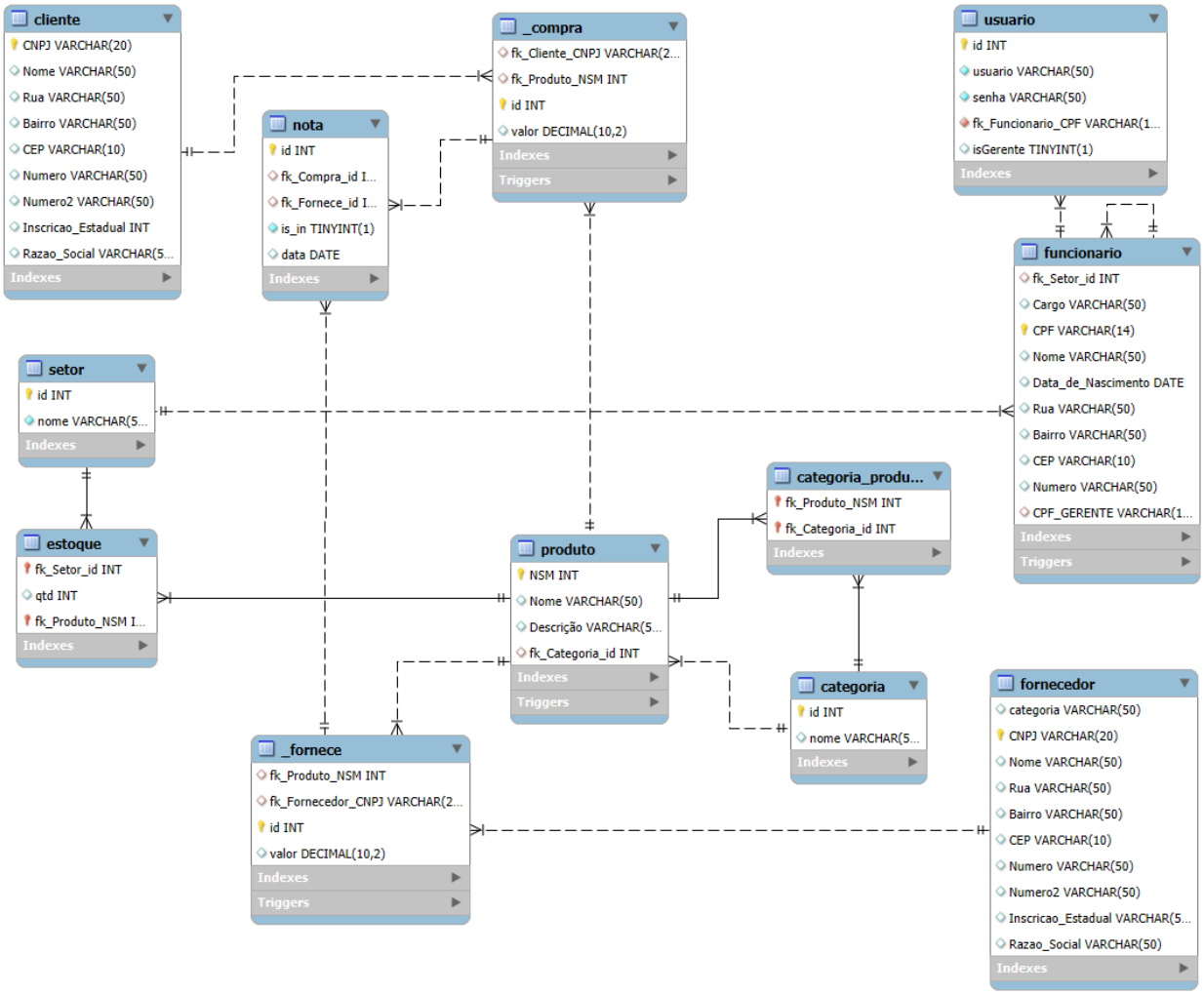
- O objetivo principal do trigger é automatizar a criação de um usuário na tabela Usuario sempre que um novo Funcionario é registrado. Isso é útil para garantir que todos os funcionários que entram no sistema também tenham uma conta de acesso associada automaticamente, com seus dados de CPF e Nome como credenciais de login.

```sql
DELIMITER //
CREATE TRIGGER after_funcionario_insert
AFTER INSERT ON Funcionario
FOR EACH ROW
BEGIN
    DECLARE cargoGerente BOOLEAN DEFAULT FALSE;

    IF NEW.Cargo = 'Gerente' THEN
        SET cargoGerente = TRUE;
    END IF;

    INSERT INTO Usuario (usuario, senha, fk_Funcionario_CPF, isGerente)
    VALUES (NEW.CPF, NEW.Nome, NEW.CPF, cargoGerente);
END //
DELIMITER ;

DELIMITER //

CREATE TRIGGER after_compra_insert
AFTER INSERT ON _Compra
FOR EACH ROW
BEGIN
    DECLARE produto_existe INT;

    SET produto_existe = 0;

    SELECT COUNT(*) INTO produto_existe
    FROM Estoque
    WHERE fk_Produto_NSM = NEW.fk_Produto_NSM;

    IF produto_existe > 0 THEN
        UPDATE Estoque
        SET qtd = qtd - 1
        WHERE fk_Produto_NSM = NEW.fk_Produto_NSM;
    ELSE
        INSERT INTO Estoque (qtd, fk_Produto_NSM, fk_Setor_id)
        VALUES (-1, NEW.fk_Produto_NSM, 1);
    END IF;

    INSERT INTO Nota (fk_Compra_id, is_in, data)
    VALUES (NEW.id, TRUE, NOW());
END //
```

Extra:

- O conceito de DEFAULT foi utilizado na coluna isGerente da tabela Usuario, com o valor padrão definido como FALSE.

Modelo Lógico

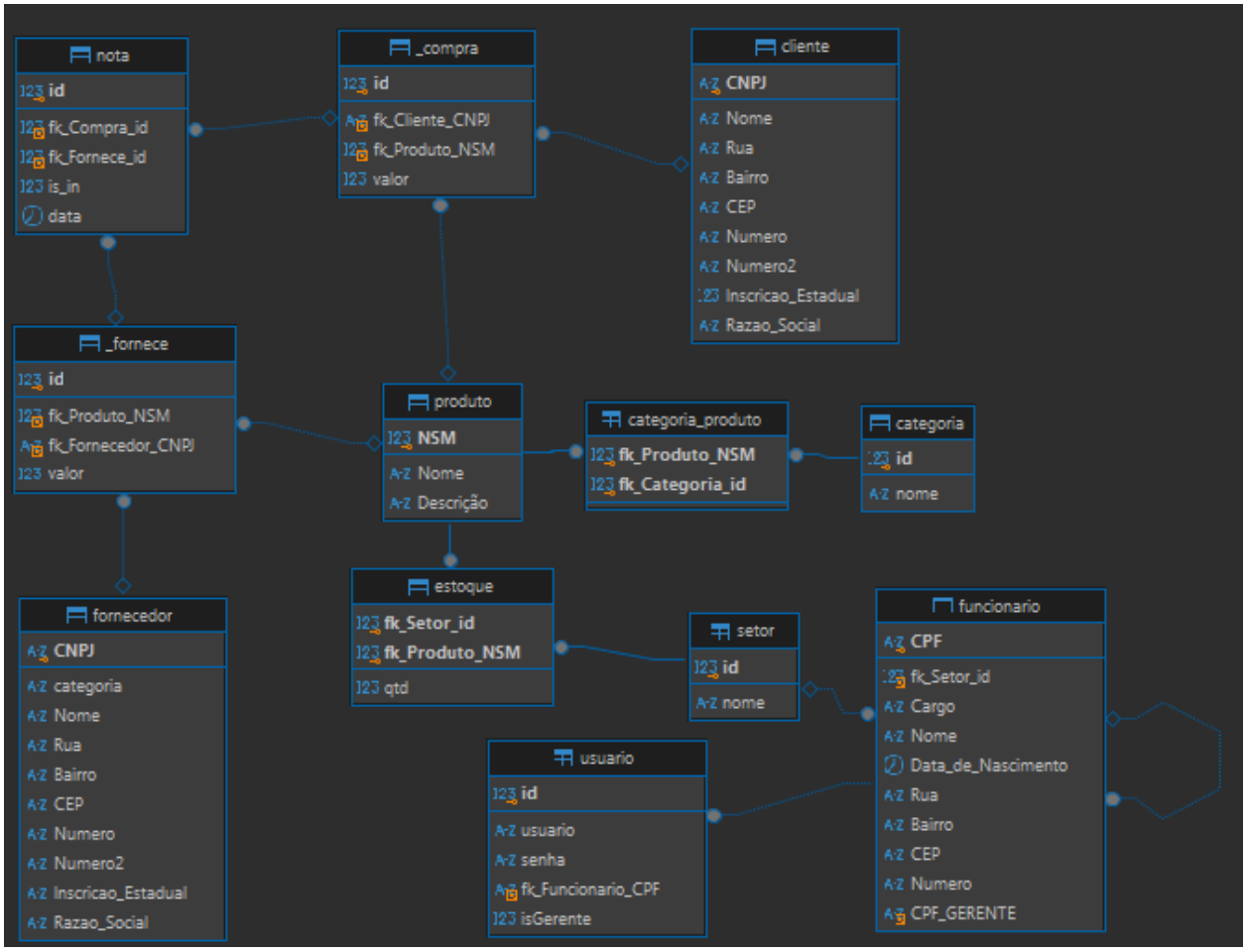


O modelo lógico desenvolvido para a Bravi organiza os dados da empresa, estruturando informações sobre clientes, compras, produtos, fornecedores, funcionários e categorias. Ele estabelece relações entre tabelas principais, como Cliente, que identifica os clientes pelo CNPJ; Compra e Nota, responsáveis pelo gerenciamento de transações; e Produto, vinculado a categorias e fornecedores para o controle de estoque e custos. Além disso, contempla a gestão interna por meio das tabelas Funcionário e Setor, proporcionando uma visão integrada das operações comerciais e administrativas.

Para promover a escalabilidade do sistema, optamos por separar as entidades Funcionário e Usuário em tabelas distintas, mesmo que a relação entre elas seja de 1:1. Essa decisão foi estratégica para acomodar requisitos futuros e simplificar a gestão de acessos. Inicialmente, cada funcionário teria um acesso específico baseado em seu cargo, como no caso de gerentes, que possuem permissões diferenciadas. A separação permite expandir as funcionalidades relacionadas a usuários sem impactar diretamente a tabela de funcionários. Por exemplo, seria

possível adicionar novos tipos de usuários no futuro (não vinculados a funcionários) ou gerenciar credenciais e permissões de forma independente.

Modelo Físico



O modelo físico do banco de dados da Bravi Distribuidora é gerado automaticamente ao rodar o script SQL, criando todas as tabelas necessárias, como cliente, fornecedor, funcionário, usuário e produto, além de estabelecer seus relacionamentos por chaves primárias e estrangeiras. Ele define, por exemplo, a conexão entre funcionários e seus usuários associados, o vínculo de gerentes com subordinados e a associação de produtos às categorias e fornecedores

Consultas SQL Avançadas

Cálculo do Lucro

```java
public Map<String, Object> getLucro() {
    String sql = "SELECT (SELECT SUM(valor) FROM _Compra) - " +
            "(SELECT SUM(f.valor) FROM _Fornece f JOIN _Compra fc ON f.fk_Produto_NSM = fc.fk_Produto_NSM) AS lucro";
    return jdbcTemplate.queryForMap(sql);
}
```

O cálculo do lucro é feito subtraindo o valor total das compras pelo valor total pago aos fornecedores. A primeira subconsulta soma todos os valores registrados na tabela Compra, representando o total das compras realizadas. A segunda subconsulta soma os valores pagos aos fornecedores, realizando um JOIN entre as tabelas Fornece e Compra para associar os pagamentos aos produtos adquiridos.

Evolução das Vendas Anuais

```java
@Override
public List<Map<String, Object>> getEvolucaoVendas() {
    String sql = "SELECT YEAR(n.data) AS ano, SUM(c.valor) AS total_vendas " +
            "FROM Nota n " +
            "JOIN _Compra c ON n.fk_Compra_id = c.id " +
            "GROUP BY YEAR(n.data) " +
            "ORDER BY ano";
    return jdbcTemplate.queryForList(sql);
}
```

A consulta tem como objetivo calcular a evolução das vendas ao longo dos anos. Ela extrai o ano da data da nota fiscal e soma os valores das compras associadas a essas notas. A consulta faz um JOIN entre as tabelas Nota e Compra usando a chave estrangeira que vincula uma nota a uma compra. Em seguida, agrupa os resultados por ano e os ordena de forma crescente. O resultado final é uma lista contendo o total de vendas por ano.

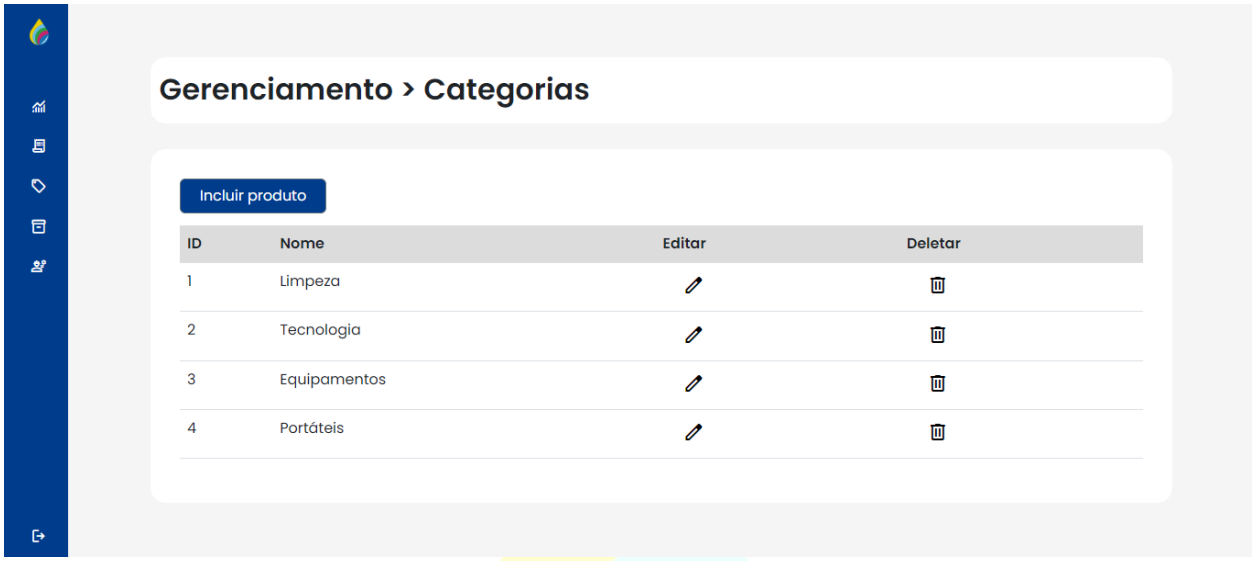
Maiores Compradores

```java
public Map<String, Object> getMaioresCompradores() {
    String sql = "SELECT c.Nome, SUM(v.valor) AS total_compras FROM _Compra v JOIN Cliente c " +
            "ON v.fk_Cliente_CNPJ = c.CNPJ GROUP BY c.Nome ORDER BY total_compras DESC LIMIT 5";
    List<Map<String, Object>> result = jdbcTemplate.queryForList(sql);
    Map<String, Object> resultMap = new HashMap<>();
    resultMap.put("maiores_compradores", result);
    return resultMap;
}
```

A consulta retorna os cinco maiores compradores somando o valor total de suas compras. Ela utiliza um JOIN entre Compra e Cliente, agrupando os resultados por nome do cliente com `GROUP BY`, ordenando os totais em ordem decrescente com ORDER BY e limitando a saída a cinco registros com LIMIT 5. O resultado é armazenado em um mapa para retorno.

Requisitos de Interface

Listar

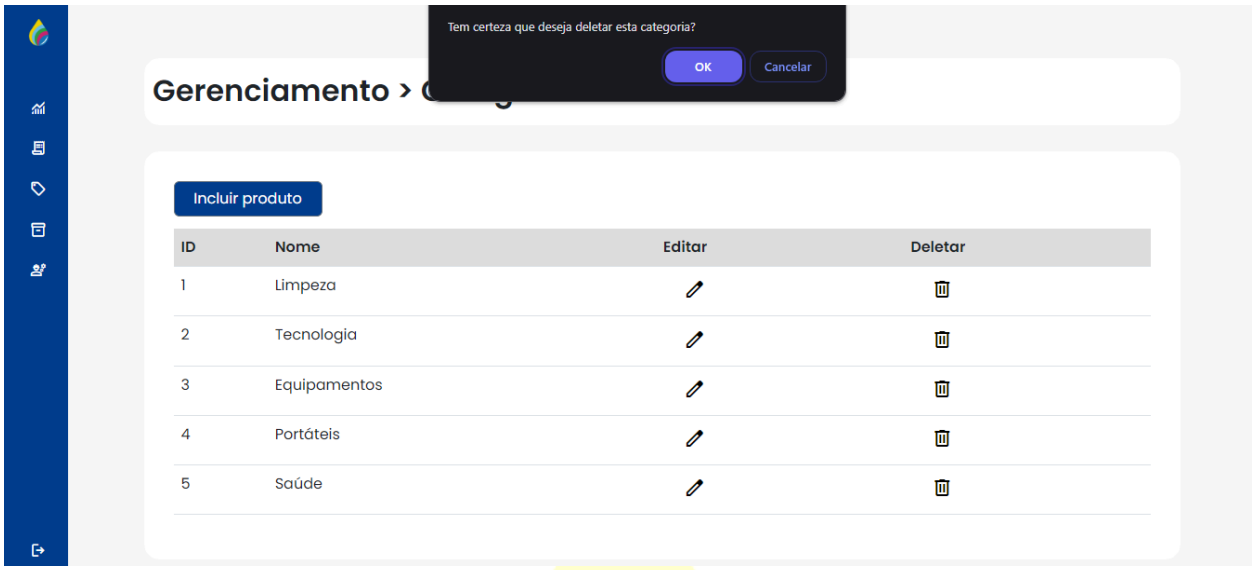


A listagem agora é exibida corretamente, com o sistema configurado para apresentar as informações da tabela desejada na interface de forma adequada.

Inserção

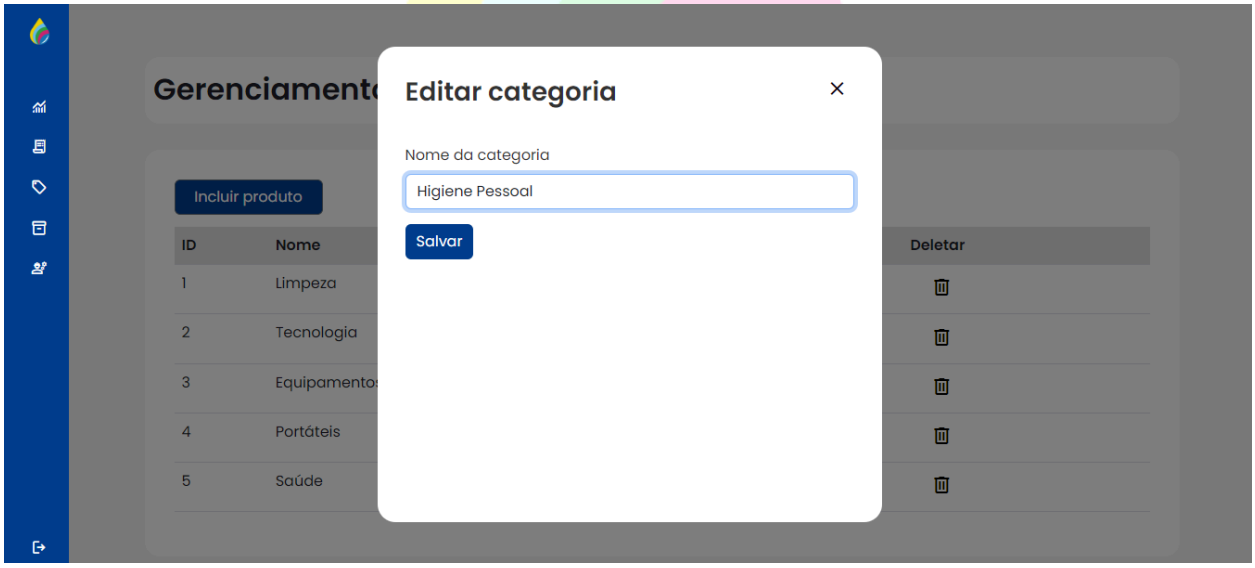
A funcionalidade de inserção está operando corretamente, permitindo que o usuário adicione novos dados nas tabelas apropriadas através da interface.

Deleção



A deleção está funcionando corretamente, possibilitando que o usuário exclua registros das tabelas desejadas diretamente pela interface.

Alteração



A alteração está operando como esperado, permitindo que o usuário modifique informações nas tabelas específicas através da interface.

Gerar relatórios

A escolha dos gráficos apresentados no dashboard foi feita com o objetivo de fornecer uma visualização clara e objetiva das principais métricas de desempenho da empresa. O gráfico de barras, que exibe o lucro por ano, foi selecionado por sua capacidade de evidenciar comparações diretas entre períodos, destacando o crescimento ou a oscilação dos resultados financeiros ao longo do tempo. Esse formato permite identificar rapidamente padrões, como anos de maior ou menor desempenho, facilitando análises históricas e planejamento estratégico.

Já o gráfico de linha, utilizado para mostrar a evolução das vendas por ano, foi escolhido por ser ideal para representar tendências ao longo do tempo. Esse tipo de gráfico permite visualizar a progressão contínua do volume de vendas, destacando momentos de alta ou baixa no desempenho, e ajuda na identificação de sazonalidades ou impactos de ações específicas no mercado.

As tabelas de maiores compradores e categorias foram incluídas para destacar os clientes mais relevantes e detalhar a diversificação do estoque. Essas informações auxiliam na fidelização de clientes-chave, personalização de ofertas e na análise dos segmentos mais representativos, orientando decisões sobre investimentos e expansão de produtos.

Por fim, os indicadores de faturamento, lucro e meta foram inseridos para oferecer uma visão consolidada das métricas financeiras mais relevantes, garantindo que os dados críticos estejam sempre acessíveis de forma clara e direta. Esses elementos são fundamentais para monitorar o desempenho em relação aos objetivos traçados e facilitar ajustes nas estratégias para alcançar os resultados desejados.

Ferramentas Utilizadas

BR_MODELO

O BR Modelo foi essencial na fase inicial do projeto para a construção dos modelos conceitual, lógico e físico. Ele nos proporcionou:

- Modelagem Conceitual: a ferramenta foi utilizada para criar o diagrama conceitual, representando as entidades, atributos e os relacionamentos de forma clara e didática. Essa etapa garantiu uma visão global e inicial do sistema de banco de dados.

- Protótipo do Modelo Lógico e Físico: após a modelagem conceitual, o BR Modelo foi usado para prototipar o modelo lógico, transformando conceitos em estruturas de tabelas e relações. Também foi possível gerar a versão inicial do modelo físico, permitindo uma visão prévia da implementação real no banco de dados.

MYSQL WORKBENCH

Para consolidar e refinar o modelo lógico e físico, migramos para o MySQL Workbench, que oferece ferramentas avançadas para modelagem e execução. As principais atividades realizadas incluem:

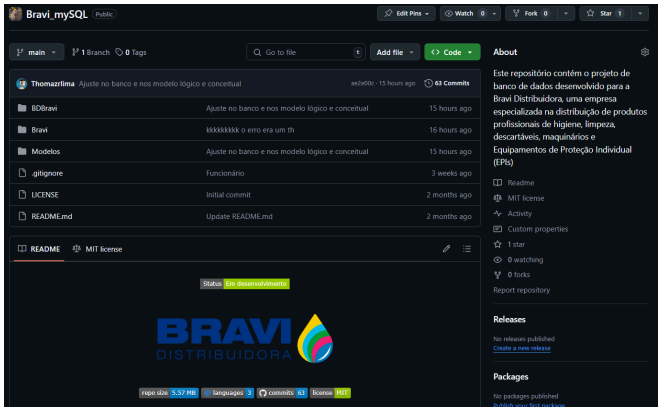
- Ajustes no Modelo Lógico: O modelo lógico foi detalhado e finalizado, garantindo uma estrutura consistente e aderente às regras de negócios.

DISCORD

- Utilizado para nossas reuniões sobre o projeto. Realizamos encontros regulares no Discord para discutir o andamento do projeto, alinhar as tarefas, e resolver dúvidas ou problemas técnicos

GIT / GITHUB

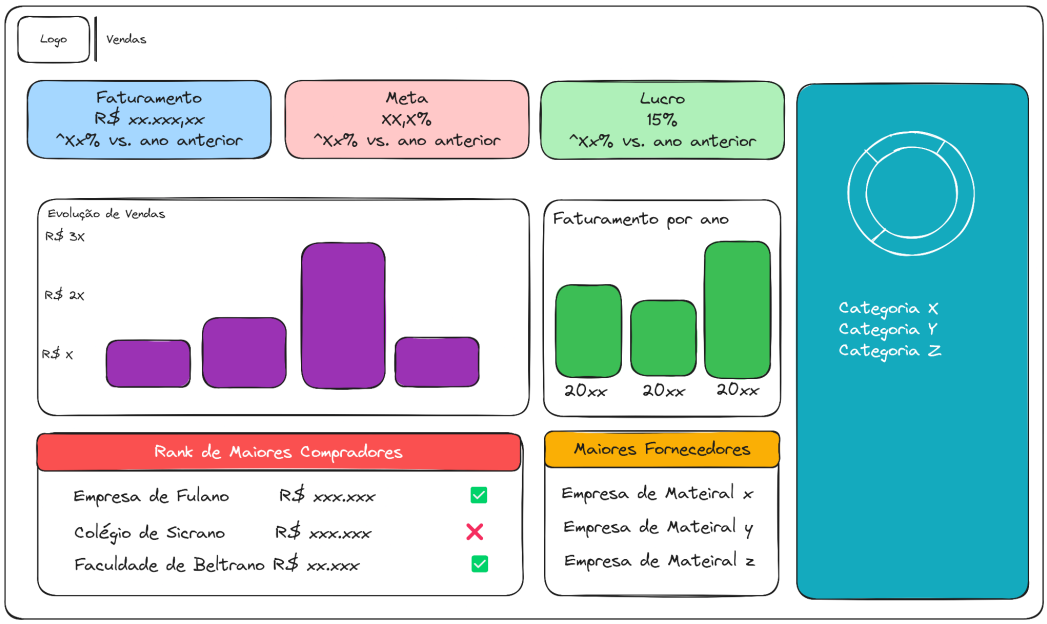
- Esses sistemas nos permitiram controlar mudanças no código, compartilhar contribuições de forma eficiente e garantir a integridade do trabalho.



Para acessar o nosso github: https://github.com/P-E-N-T-E-S/Bravi_mySQL

Excalidraw
Esboço de Interfaces: Desenhamos protótipos de telas e fluxos de interação de forma simples e ágil.

Diagramas de Fluxo: Elaboramos diagramas que representavam processos e interações entre os componentes do sistema.



IDEAs
Durante o desenvolvimento deste projeto, utilizamos ferramentas robustas da JetBrains, como o IntelliJ IDEA e o DataGrip, para facilitar e otimizar nosso fluxo de trabalho.

1. **IntelliJ IDEA**
   - O IntelliJ IDEA foi a principal ferramenta empregada para o desenvolvimento da aplicação. Este ambiente de desenvolvimento integrado (IDE) ofereceu funcionalidades avançadas que aceleraram o processo de codificação e depuração.

2. **DataGrip**
   - O DataGrip foi essencial para gerenciar e interagir com o banco de dados utilizado no projeto. Esta ferramenta facilitou as operações de manipulação de dados e manutenção do esquema do banco.

Postman
- O Postman foi uma ferramenta utilizada para o desenvolvimento, teste e validação das APIs RESTful implementadas neste projeto. Ele proporcionou um ambiente completo para simular requisições e verificar o comportamento do sistema, garantindo que as funcionalidades atendam aos requisitos especificados.

Dependências

Spring

O Spring Framework foi adotado como a base para o desenvolvimento da aplicação, graças à sua modularidade, robustez e integração com diversas ferramentas. A configuração simplificada e a inicialização rápida facilitaram o desenvolvimento ágil, permitindo a redução da complexidade inicial do projeto. Além disso, a fácil integração com outros módulos do Spring garantiu que as funcionalidades fossem adicionadas de forma eficiente e sem contratempos.

Thymeleaf

Para a construção das interfaces web, utilizamos o Thymeleaf como mecanismo de template. A escolha se deu pela sua integração nativa com o Spring Boot, que proporcionou um desenvolvimento mais fluido e dinâmico. O Thymeleaf permite a construção de páginas HTML de maneira simples e intuitiva, o que agilizou a implementação das funcionalidades do projeto.

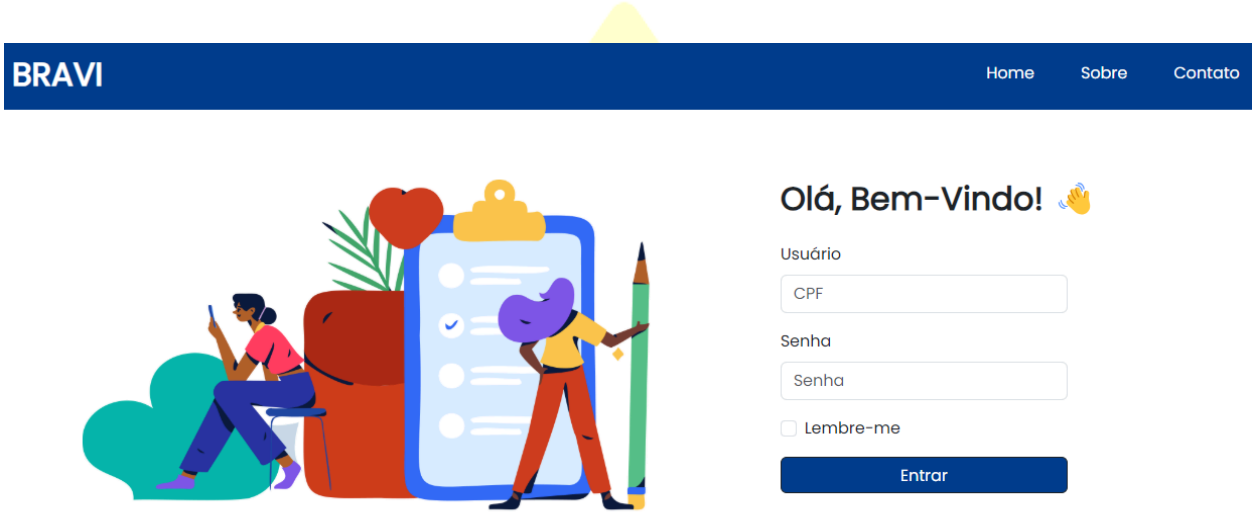


Chart.js

O Chart.js foi selecionado para a criação de gráficos e visualizações, principalmente devido à sua simplicidade e flexibilidade. Com suporte a diversos tipos de gráficos interativos, a biblioteca permitiu uma apresentação clara e eficiente dos dados, além de garantir a interatividade necessária para uma experiência de usuário mais rica. A utilização do Chart.js agregou valor ao projeto, proporcionando gráficos dinâmicos e visualmente atraentes, essenciais para a análise de dados.

Como Rodar

Pré-requisitos

- **Java 21 ou superior**
- **JDK Instalado**
- **Node.js**
- **Banco de Dados MySQL configurado e rodando localmente ou em um servidor remoto.**

1. **Clone a aplicação no GitHub**
   Primeiro, clone o repositório do projeto para o seu ambiente local:
   https://github.com/P-E-N-T-E-S/Bravi_mySQL

```
git clone <Url do repositório>
cd <diretório do projeto>
```

2. **Configurar o banco de dados**

   Você precisa de um banco de dados configurado para rodar a aplicação. Caso ainda não tenha configurado um banco de dados, siga os passos abaixo:

   2. 1 Abra o Pasta BDBravi.
   Ao acessar a pasta BDBravi, você encontrará dois arquivos. O primeiro, denominado “codigoBanco”, deve ser executado para criar a estrutura do banco de dados. Em seguida, execute o arquivo “povoamentoBanco” para inserir os dados iniciais e povoar as tabelas criadas.

3. **Executando a Aplicação**

   3. 1 Criando o Arquivo .env
   É essencial que o arquivo .env esteja localizado na pasta Bravi para garantir o funcionamento correto do sistema. Esse arquivo contém variáveis de configuração sensíveis, como as credenciais do banco de dados e outras configurações importantes, que são utilizadas pelo sistema durante a execução. A ausência ou posicionamento incorreto do .env pode causar falhas na conexão com o banco de dados ou outros erros críticos.
   Crie o arquivo **.env** na pasta **Bravi** (que contém o src) com o seguinte conteúdo:

```
DATABASE_URL=jdbc:mysql://localhost:3306/BDBravi
DATABASE_USERNAME={Seu Usuário}
DATABASE_PASSWORD={Sua Senha}
```

3. 2 Rodando no Terminal

Certifique-se de que o terminal esteja na pasta Bravi antes de rodar os comandos. Isso é fundamental, pois a execução depende da estrutura de arquivos e configurações específicas presentes nesta pasta. Caso contrário, o sistema não será iniciado corretamente.

3. 2 .1 MacOs ou Linux

Se o seu sistema operacional for MacOS ou Linux, é necessário preparar o script para execução. Para isso, execute o seguinte comando antes de prosseguir:

```
chmod +x mvnw
```

Para instalar as dependências do projeto, execute o seguinte comando

```
npm install
./mvnw clean install
```

Após garantir que o arquivo mvnw possui permissões de execução, é necessário executar o seguinte comando para iniciar a aplicação:

```
./mvnw spring-boot:run
```

3 .3 Rodando no Intelij

Se você estiver utilizando o IntelliJ IDEA, rodar a aplicação é ainda mais simples. Após abrir o projeto na IDE e clique no ícone de play.

A IDE cuidará de compilar e executar a aplicação automaticamente, sem a necessidade de comandos adicionais no terminal. Certifique-se de que todas as configurações do projeto estão corretas, como o arquivo .env na pasta Bravi, para evitar problemas durante a execução.

4. Acessando a aplicação

Para acessar a aplicação, abra o navegador e digite o seguinte endereço na barra de pesquisa: http://localhost:8080/login

5. Login:

Ao acessar o login, insira o **CPF** como nome de usuário e o **Nome** como a senha. Aqui estão alguns exemplos de login:

1. **Usuário**: 44444444444, **Senha**: João Silva
2. **Usuário**: 55555555555, **Senha**: Maria Oliveira
3. **Usuário**: 66666666666, **Senha**: Carlos Souza
4. **Usuário**: 77777777777, **Senha**: Ana Costa
5. **Usuário**: 88888888888, **Senha**: Pedro Santos
6. **Usuário:** 99999999999, **Senha:** Julia Martins

Processo de desenvolvimento

1. **Planejamento e Análise Inicial:**

O desenvolvimento iniciou com o mapeamento das necessidades do cliente, a Bravi Distribuidora, um cliente real no ramo de distribuição de produtos. A análise envolveu entrevistas com a equipe da Bravi para entender suas dores e requisitos específicos. Essa etapa resultou na definição clara dos objetivos e no escopo inicial do projeto, preparando o terreno para a modelagem e estruturação do banco de dados.

2. **Desenvolvimento do Modelo Conceitual:**

Com base nas necessidades levantadas na fase de planejamento, foi criado o modelo conceitual do banco de dados. Esse modelo refletiu a estrutura do sistema, considerando as principais entidades e seus relacionamentos, como clientes, fornecedores, produtos, funcionários e estoque. A modelagem foi feita de forma a garantir que todas as operações do sistema fossem contempladas de forma eficaz.

3. **Desenvolvimento do Modelo Lógico:**

O próximo passo foi transformar o modelo conceitual em um modelo lógico, adaptando-o para o ambiente de banco de dados relacional. Durante essa fase, foram refinadas as cardinalidades, chaves primárias e estrangeiras, garantindo a integridade referencial entre as tabelas. A validação das decisões do modelo lógico foi realizada de forma contínua, com feedback de professores e ajustes nas definições.

4. **Codificação:**

A codificação foi iniciada com a configuração de um projeto Spring Boot, utilizando Apache Maven como gerenciador de dependências. Foi feita a integração com o banco de dados MySQL, configurando o JDBC para comunicação eficiente entre a aplicação e o banco.

4.1 Desenvolvimento Inicial: O projeto foi configurado para incluir dependências essenciais, como Spring Boot, e foi iniciado o processo de organização e definição de rotinas de desenvolvimento. O foco inicial foi na estruturação do back-end.

4.2 Criação dos Modelos (Model) e API REST: Foram criados os modelos (abstrações de tabelas) em Java, com seus respectivos métodos getter, setter e construtores. Cada modelo foi organizado de acordo com o tipo de entidade, como cliente, fornecedor e produto, sendo testado com ferramentas como Insomnia para garantir a comunicação com a API REST.

4.3 Criação dos Repositórios (Repository): Para cada modelo criado, foi desenvolvido um repositório responsável pela interação com o banco de dados. Cada repositório contava com consultas SQL específicas para acessar e manipular os dados.

4.4 Criação dos Controladores (Controller): Os controladores foram implementados para definir as rotas e as funções de cada endpoint da API, garantindo a aplicação das regras de negócios e o correto funcionamento da comunicação entre front-end e back-end.

4.5 Protótipo de Interface: Utilizando Excalidraw, foi elaborado um protótipo visual que guiou o desenvolvimento da interface do sistema, garantindo que a experiência do usuário fosse intuitiva e visualmente agradável.

4.6 Na fase de desenvolvimento do front-end, optamos por utilizar HTML e CSS puro para a construção das páginas do sistema

4.7 Desenvolvimento das Páginas: Seguindo o protótipo desenvolvido no Excalidraw, foram criadas as páginas do sistema. Começamos com a página de Login, por ser mais simples.

4.8 Desenvolvimento da Página Home: A página principal foi elaborada com várias requisições à API, utilizando dados de diferentes modelos.

5. **Finalização e Entrega do Projeto:**

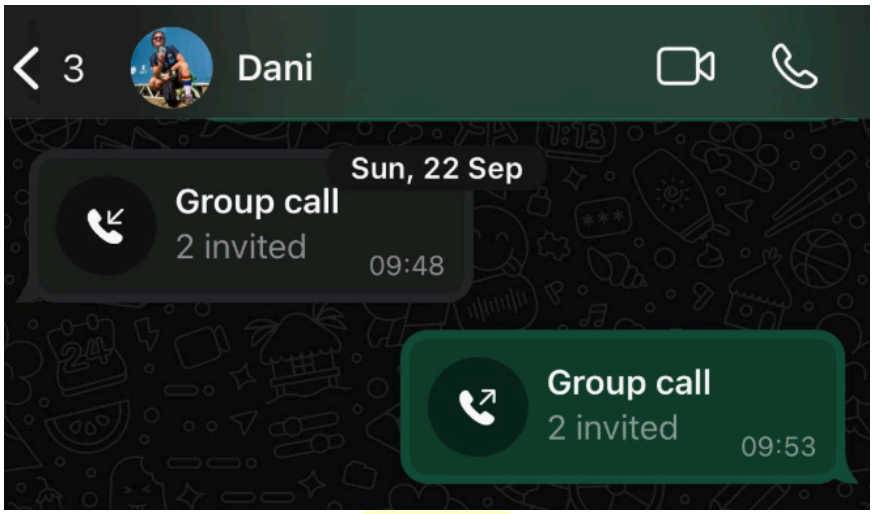
Após a conclusão do desenvolvimento, o projeto passou pela fase de documentação, onde todas as etapas e decisões de desenvolvimento foram descritas detalhadamente. Foi realizada também a preparação dos slides para apresentação, detalhando as funcionalidades, a arquitetura do sistema e os resultados obtidos. A entrega final foi realizada com a documentação completa, código-fonte e o sistema operacional.

Validação do Cliente

A autenticidade do projeto e sua realização em parceria com a Bravi Distribuidora podem ser comprovadas pelos seguintes pontos:

1. **Contato Direto com a Proprietária da Bravi:** Tivemos interação direta com Daniella, proprietária da Bravi Distribuidora, para validar as necessidades e compreender as demandas do cliente. Vale ressaltar que Daniella é tia de Sofia, reforçando a relação de confiança no desenvolvimento do projeto**.**

2. **Evidências de Comunicação:** Dispomos de registros, como prints de conversas e históricos de chamadas realizadas com Daniella, detalhando o processo de coleta de requisitos e a validação das funcionalidades do projeto.



3. **Provas da Associação de Daniella com a Bravi:**
   Publicações no Instagram da Bravi contendo fotos de Daniella.
   Criação do mascote da empresa inspirado em Daniella, evidenciando seu vínculo com a marca.