

Documentação do Projeto de Banco de Dados

Aplicação para o restaurante japonês



Professora/orientadora:

- Gabrielle Karine Canalle

Responsáveis

- Adriana Rodrigues
- Pedro Villas Boas
- Vinicius Ventura

01. APRESENTAÇÃO DO PROJETO

Esta documentação descreve o projeto de desenvolvimento do site de vendas para o **Restaurante Japonês Portal Temaki**, da qual foram utilizadas diversas tecnologias e práticas com o uso destas.

O projeto envolveu o uso de SQL da linguagem de SQL e o MySQL como o sistema de gerenciamento de banco de dados relacional, que possui três lojas (Recife, Olinda e Paulista) e realiza vendas tanto nas lojas físicas quanto online.

Para os **dados**, foi definido SQL como a linguagem de programação e o MySQL como o sistema de gerenciamento de banco de dados relacional. Para o **backend**, foi utilizado Java como linguagem de programação. E para o **frontend**, foi utilizado HTML, CSS, Bootstrap e Javascript. E spring boot para conexão entre estes.

Uma etapa fundamental do projeto foi a realização da entrevista com a cliente. Essa entrevista teve como objetivo entender detalhadamente as necessidades e expectativas da cliente, permitindo a construção de um minimundo que represente os requisitos do sistemas. A partir desta entrevista, foram retiradas as principais funcionalidades e dados necessários para o desenvolvimento do banco de dados e do site.

01.1. Requisitos Identificados

- **Funcionalidades Principais**
 - Cadastro de funcionários e dependentes
 - Cadastro de clientes
 - Autenticação e login de usuários
 - Cadastro do cardápio do menu
 - Gerenciamento (cadastro e, alteração)
 - Sistema de pedidos e controle de atendimento do pedido

- **Outros Requisitos**
 - Interface amigável
 - Segurança dos dados

01.2. Repositório

A aplicação encontra-se disponível no repositório abaixo:

Github: <https://github.com/Dricalucia/BD--Projeto>

02. MINIMUNDO

O **Portal Temaki** é uma rede um restaurante com três unidades (uma matriz e duas filiais) e gostaria de ter um sistema de gerenciamento da operação de vendas on-line. Cada unidade possui as seguintes informações: CNPJ, IE, endereço, telefone, ponto de referência.

Cada loja tem seu time de funcionários, que podem ser: atendentes, ajudantes de cozinha, auxiliares de serviços gerais, supervisor e chefs. Cada funcionário é identificado por um número único de matrícula, nome, função, data contratação, dependentes, salário e horário de trabalho.

Os clientes, na maioria, são pessoas físicas, mas também atende pedidos de pessoas jurídicas. As informações registradas são: nome, endereço, telefone, email, contato, data cadastro, ponto referencia, observação. Os clientes que possuem cadastro, atualmente, são os que realizam pedidos por aplicativo.

Cada pedido realizado pelo cliente é gerado um número único, com data e hora do pedido, o tipo de pedido,, o número do cliente (para clientes cadastrados) e o status do pedido.

Através do site, os clientes têm acesso ao cardápio de menu, onde ele poderá visualizar os itens que deseja inserir no carrinho de compra. O pedido só pode ser confirmado se o cliente possuir cadastro no aplicativo e estiver logado.

Os itens do menu representam os itens disponíveis no cardápio do restaurante, como sushis, sashimis, tempurás, yakisobas, entre outros. Cada item de menu possui um código único, nome, descrição, preço e categoria.

Diariamente um produto do cardápio é colocado em promoção no site. O prazo de validade desta promoção é de 24 horas.

Deseja uma aplicação que tenha uma interface amigável para que os clientes possam utilizar sem dificuldade. É fundamental a segurança dos dados que estão sendo gerados e armazenados no banco de dados.

03. MODELAGEM

A modelagem do banco de dados foi dividida em três etapas principais: **conceitual, lógica e física.**

Utilizamos diagramas ER(Entidade-Relacionamento) para visualizar as entidades, atributos e relacionamentos entre elas

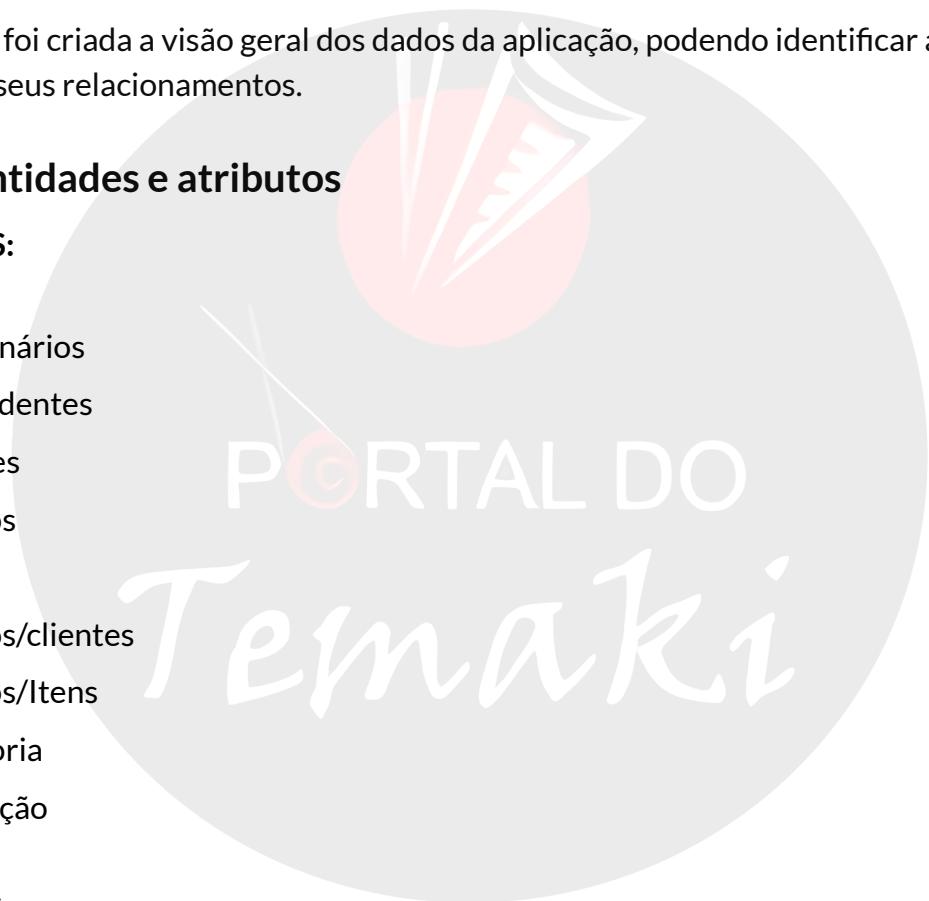
03.1. Modelagem Conceitual

Nessa etapa foi criada a visão geral dos dados da aplicação, podendo identificar as principais entidades e seus relacionamentos.

03.1.1. Entidades e atributos

ENTIDADES:

- Lojas
- Funcionários
- Dependentes
- Clientes
- Pedidos
- Itens
- Pedidos/clientes
- Pedidos/Itens
- Categoria
- Promoção

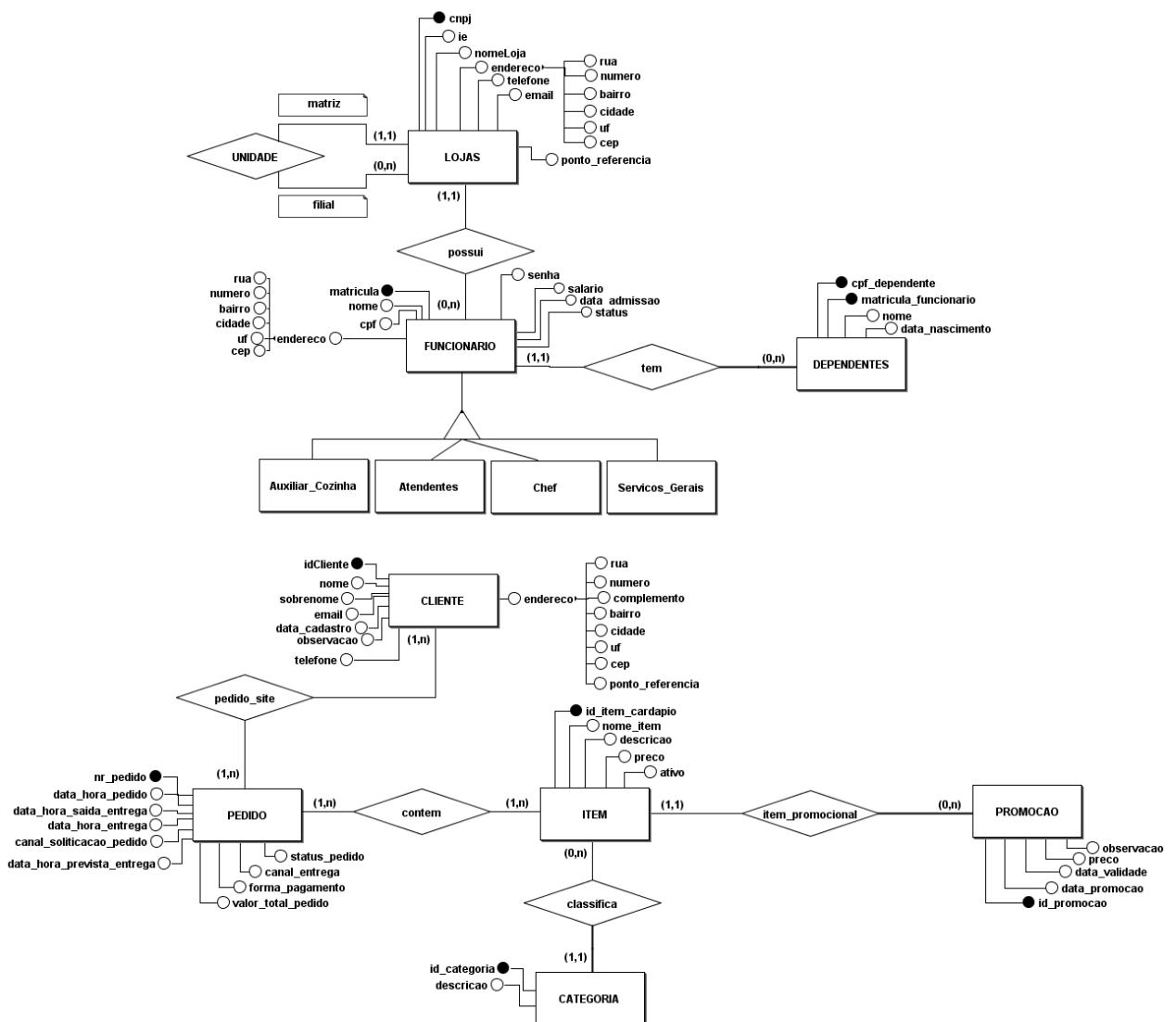


ATRIBUTOS:

- **Filial:** CNPJ, IE, endereço, telefone, contato, email
- **Funcionários:** Matrícula(chave única serial), nome, CPF, RG, dt_nascimento, endereço, telefone, email, salário e horário de trabalho
- **Dependentes:** IdDependentes (chave única serial), nome, dt_nascimento, CPF
- **Clientes:** IDCliente(chave única serial), nome, endereço, telefone, email, ponto referencia, observação
- **Menu (itens):** IdMenu(chave única serial), nome, descrição, preço, categoria (multivvalorado)

- Pedidos: IDPedido(chave única serial), IDCliente, dtPedido, hrPedido, TipoPedido (multivvalorado), IdItensMenu, quantidade, preço, status

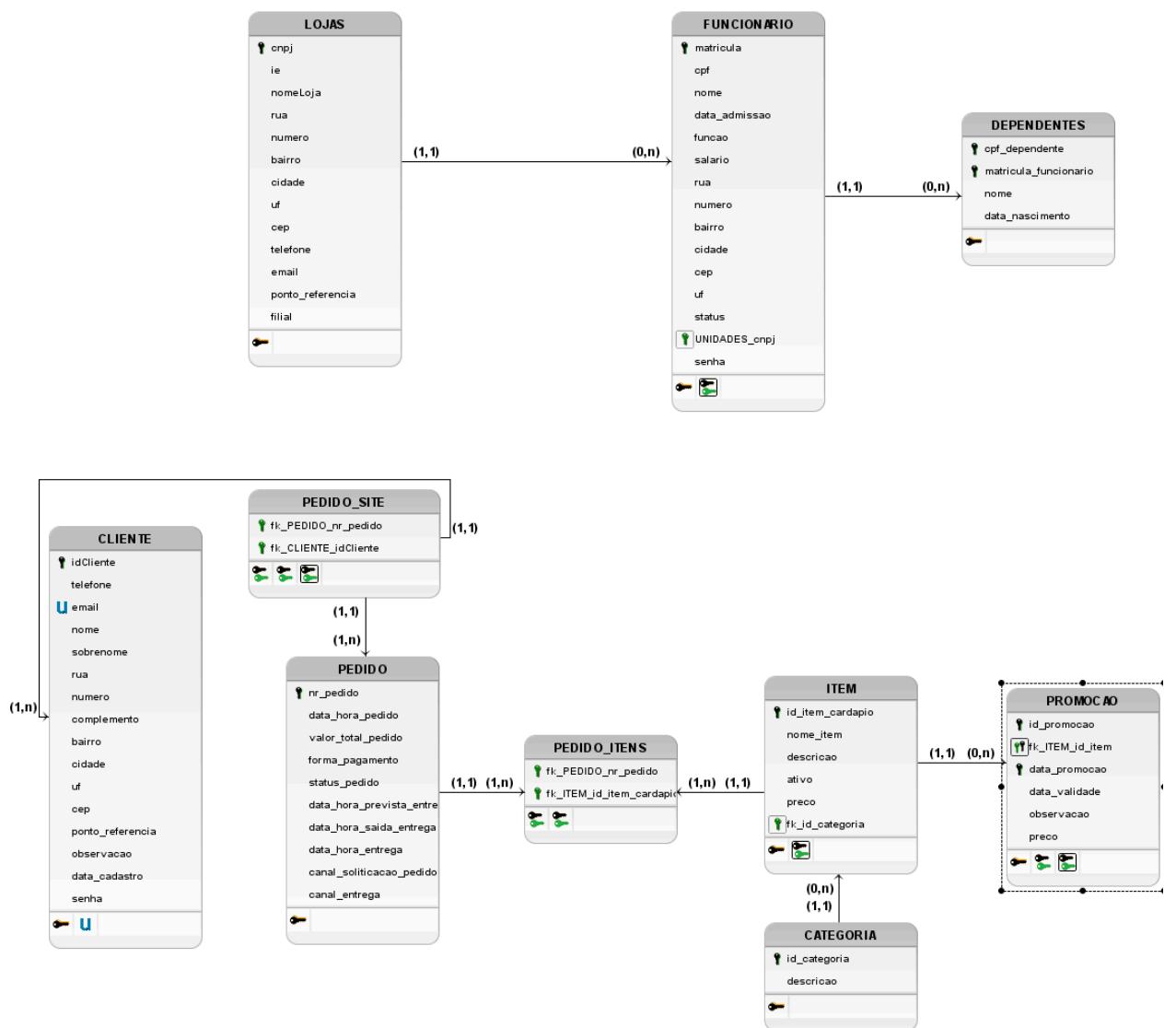
03.1.2. Diagrama Entidade-Relacionamento (DER)



03.2. Modelagem Lógica

Com base no modelo conceitual, foi elaborado o modelo lógico, que incluiu a definição das tabelas, colunas, tipos de dados e relacionamento entre as tabelas. Essa fase visou garantir que o banco de dados atendesse aos requisitos de armazenamento e recuperação de dados de maneira eficiente.

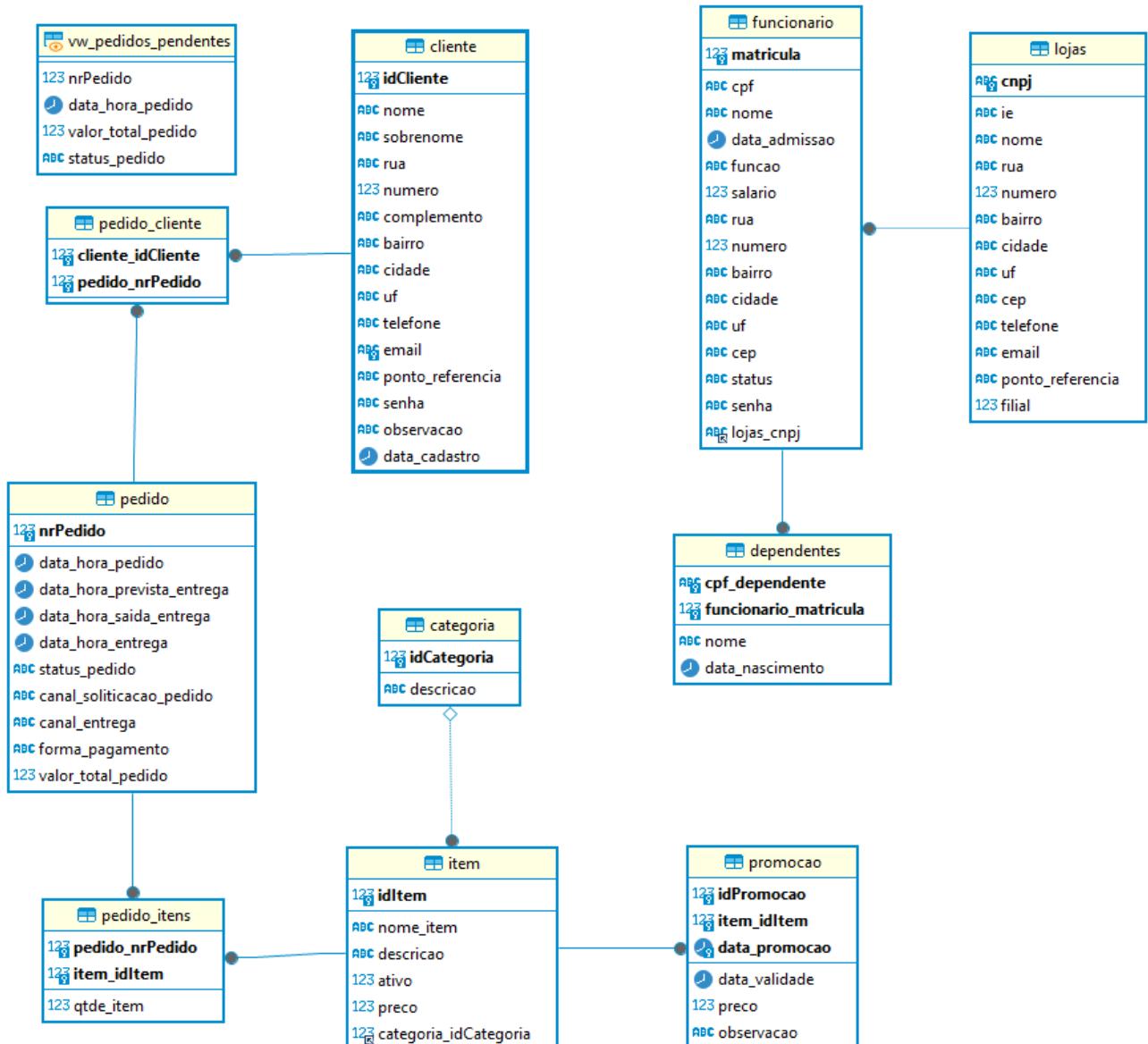
03.2.1. Diagrama Entidade-Relacionamento (DER)



03.3. Modelagem Física

A modelagem física traduziu o modelo lógico na implementação real no SQL. Inclui a criação de scripts SQL para criação das tabelas e outros objetos do banco de dados

03.3.1. Diagrama Entidade-Relacionamento (DER)



03.3.2. Script SQL

```

/* Projeto BD Restaurante */
create database restaurante;
use restaurante;
create table lojas (
  cnpj varchar(15) primary key,
  ie varchar(15) not null,
  nome varchar(10) not null,
  rua varchar(40) not null,
  numero integer,
  bairro varchar(15) not null,
  cidade varchar(15) not null,
  uf varchar(2) not null,
  cep varchar(8) not null,
  telefone varchar(11),
  email varchar(50),
  ponto_referencia text,
  filial boolean
);

create table funcionario (
  matricula integer primary key auto_increment,
  cpf varchar(11) not null,
  nome varchar(50) not null,
  data_admissao datetime not null default current_timestamp,
  funcao varchar(20) not null,
  salario decimal(10,2),
  rua varchar(50) not null,
  numero integer,
  bairro varchar(20) not null,
  cidade varchar(20) not null,
  uf varchar(2) not null,
  cep varchar(8) not null,
  status char(1) not null,
  lojas_cnpj varchar(15) not null,
  constraint fk_lojas_cnpj foreign key (lojas_cnpj) references lojas(cnpj)
);

create table dependentes (
  cpf_dependente varchar(11),
  funcionario_matricula integer,
  nome varchar(50) not null,
  data_nascimento date not null,
  constraint fk_funcionario_matricula foreign key (funcionario_matricula) references funcionario(matricula),
  primary key (cpf_dependente, funcionario_matricula)
);

create table cliente (
  idCliente integer primary key auto_increment,
  nome varchar(10) not null,
  sobrenome varchar(10) not null,

```

```

rua varchar(40) not null,
numero integer,
complemento varchar(30),
bairro varchar(15) not null,
cidade varchar(15) not null,
uf varchar(2) not null,
telefone varchar(11) unique not null,
email varchar(50),
ponto_referencia text not null,
observacao text,
data_cadastro datetime not null default current_timestamp
);

create table item (                                # temaki salmão, urumaki, jôjô
  idItem integer primary key auto_increment,
  nome_item varchar(100),
  descricao varchar(255),
  ativo boolean,
  preco decimal(10,2),
  categoria_idCategoria integer,
  constraint fk_categoria_idCategoria foreign key (categoria_idCategoria) references categoria(idCategoria)
);
create table categoria (                         # Bebidas, Proteinas, Temperos, Carboidrato
  idCategoria integer primary key auto_increment,
  descricao varchar(20)
);
create table promocao (
  idPromocao integer auto_increment,
  item_idItem integer,
  data_promocao datetime not null default current_timestamp,
  data_validade datetime not null,
  preco decimal(10,2),
  observacao varchar(255),
  constraint fk_item_idItem_promocao foreign key (item_idItem) references item(idItem),
  primary key (idPromocao, item_idItem, data_promocao)
);

create table pedido (
  nrPedido integer primary key auto_increment,
  data_hora_pedido datetime not null default current_timestamp,
  data_hora_prevista_entrega datetime not null,
  data_hora_saida_entrega datetime,
  data_hora_entrega datetime,
  status_pedido varchar(20),
  canal_solicitacao_pedido varchar(20) default 'site', #Padrão será site
  canal_entrega varchar(20) default 'domicilio',           #em domicilio, retirada na loja
  forma_pagamento varchar(20) default 'cartao credito',    #cartão credito, cartão débito, vale alimentação, dinheiro
  valor_total_pedido decimal(10,2)
);

create table pedido_cliente (
  cliente_idCliente integer,

```

```

pedido_nrPedido integer,
constraint fk_cliente_idCliente_pedidos foreign key (cliente_idCliente) references cliente(idCliente),
constraint fk_pedido_nrPedido_pedidos foreign key (pedido_nrPedido) references pedido(nrPedido),
primary key (cliente_idCliente, pedido_nrPedido)
);
create table pedido_itens (
  pedido_nrPedido integer,
  item_idItem integer,
  qtde_item integer,
  constraint fk_pedido_nrPedido_itens foreign key (pedido_nrPedido) references pedido(nrPedido),
  constraint fk_item_idItem_itens foreign key (item_idItem) references item(idItem),
  primary key(pedido_nrPedido, item_idItem)
);

```

/ / /

```

/* Scripts SQL Avançado - RELATORIOS */
/* View para listar todos os pedidos que estão com status 'em aberto' */
/* para ser usado na tela de funcionários */
create view vw_pedidos_pendentes as
select
  nrPedido,
  data_hora_pedido,
  valor_total_pedido,
  status_pedido
from pedido
where status_pedido <> 'Concluido' -- definir os níveis de status: Em aberto, Em preparo, Saída para entrega, Concluído
order by nrPedido;

select * from vw_pedidos_pendentes; -- chamada da view

```

```

/* Visualizar os detalhes do pedido */
select
  p.nrPedido,
  p.data_hora_pedido,
  p.valor_total_pedido,
  c.nome as nome_cliente,
  c.sobrenome as sobrenome_cliente,
  i.nome_item as item,
  pit.qtde_item as qtde,
  i.preco as Preco,
  p.data_hora_prevista_entrega,
  p.data_hora_saida_entrega,
  p.data_hora_entrega
from pedido p
join pedido_cliente pc on p.nrPedido = pc.pedido_nrPedido
join cliente c on pc.cliente_idCliente = c.idCliente
join pedido_itens pit on p.nrPedido = pit.pedido_nrPedido
join item i on pit.item_idItem = i.idItem
where p.nrPedido = ?; -- substituir '?' pelo parâmetro do nrpedido, que será passado na lista de pedido

```

```

/* Função para exibir a lista de pedidos por determinado período
* data inicial e final devem ser passados como parâmetros */
delimiter $$
```

```

create function fn_pedidos_por_periodo(@data_inicial date, @data_final date)
returns table
as
begin
  return (
    select
      p.nrPedido as Nr_pedido,
      p.data_hora_pedido as Data_Hora_Pedido,
      p.valor_total_pedido as Total_Pedido,
      c.nome as nome_cliente,
      c.sobrenome as sobrenome_cliente
    from pedido p
    join pedido_cliente pc on p.nrPedido = pc.pedido_nrPedido
      join cliente c on pc.cliente_idCliente = c.idCliente
    where p.data_hora_pedido between @data_inicial and @data_final
  );
end $$
delimiter ;

select *
from fn_pedidos_por_periodo(?, ?); -- substituir '?' por variaveis que tenha a data inicial e final da seleção de pedidos

/* Função para exibir a lista de pedidos por cliente e determinado periodo
 * data inicial e final devem ser passados como parametros */
delimiter $$
create function fn_pedidos_cliente_por_periodo(@data_inicial date, @data_final date, @cliente_id int)
returns table
as
begin
  return (
    select
      p.nrPedido as nr_Pedido,
      p.data_hora_pedido as data_hora_pedido,
      p.valor_total_pedido as total_pedido,
      c.nome as nome_cliente,
      c.sobrenome as sobrenome_cliente
    from pedido p
    join pedido_cliente pc on p.nrPedido = pc.pedido_nrPedido
      join cliente c on pc.cliente_idCliente = c.idCliente
    where p.data_hora_pedido between @data_inicial and @data_final
      and c.idCliente = @cliente_id
  );
end $$
delimiter ;

-- chamada da função
select *
from fn_pedidos_cliente_por_periodo(?, ?, ?); -- substituir '?' por variaveis que tenha a data inicial e final e o codigo do cliente

/* listar o pedido de maior valor realizado em determinado periodo */

```

```

select p.data_hora_pedido, pc pedido_nrPedido, c.nome as nome_cliente, c.sobrenome as sobrenome_cliente,
p.valor_total_pedido
from pedido_cliente pc
join cliente c on pc.cliente_idCliente = c.idCliente
join pedido p on pc.pedido_nrPedido = p.nrPedido
where p.valor_total_pedido =
  select max(valor_total_pedido)
  from pedido
  where data_hora_pedido between ? and ?      -- substituir '?' por variaveis que tenha a data inicial e final
)
limit 1;

/* Listar por cliente a quantidade de pedidos realizados em determinado periodo */
select c.nome as nome_cliente, c.sobrenome as sobrenome_cliente, count(pc.pedido_nrPedido) as quantidade_pedidos
from pedido_cliente pc
join cliente c on pc.cliente_idCliente = c.idCliente
join pedido p on pc.pedido_nrPedido = p.nrPedido
where p.data_hora_pedido between ? and ?      -- substituir '?' por variaveis que tenha a data inicial e final
group by c.nome;

/* Listar o total de vendas por item determinado periodo */
select pit.item_idItem, i.nome_item, sum(pit.qtde_item) as total_vendas
from pedido_itens pit
join item i on pit.item_idItem = i.idItem
join pedido p on pit.pedido_nrPedido = p.nrPedido
where p.data_hora_pedido between ? and ?      -- substituir '?' por variaveis que tenha a data inicial e final
group by pit.item_idItem, i.nome_item;

-- Populando tabela lojas
INSERT INTO lojas (cnpj, ie, nome, rua, numero, bairro, cidade, uf, cep, telefone, email, ponto_referencia, filial)
VALUES
('', '', '1', '', '', '', '', '', '', TRUE);

-- Populando tabela funcionario
INSERT INTO funcionario (matricula, cpf, nome, data_admissao, funcao, salario, rua, numero, bairro, cidade, uf, cep,
status, senha, lojas_cnpj)
VALUES
(1, '', NOW(), '1.00', 1, '', '', '', '', '', '');

-- Populando tabela dependentes
INSERT INTO dependentes (cpf_dependente, funcionario_matricula, nome, data_nascimento)
VALUES
('1', '1990-01-01');

-- Populando tabela cliente
INSERT INTO cliente (idCliente, nome, sobrenome, rua, numero, complemento, bairro, cidade, uf, telefone, email,
ponto_referencia, senha, observacao, dataCadastro)
VALUES
(1, '', '1', '', '', '', '', '', '', '', NOW());

-- Populando tabela pedido
INSERT INTO pedido (nrPedido, data_hora_pedido, data_hora_prevista_entrega, data_hora_saida_entrega,
data_hora_entrega, status_pedido, canal_solicicacao_pedido, canal_entrega, forma_pagamento, valor_total_pedido)

```

VALUES

```
(1, NOW(), DATE_ADD(NOW(), INTERVAL 1 HOUR), NOW(), NOW(), '', '', '', '', 0.00);
```

-- Populando tabela categoria

```
INSERT INTO categoria (idCategoria, descricao)
```

VALUES

```
(1, '');
```

-- Populando tabela item

```
INSERT INTO item (idItem, nome_item, descricao, ativo, preco, categoria_idCategoria)
```

VALUES

```
(1, "", TRUE, 0.00, 1);
```

-- Populando tabela promocao

```
INSERT INTO promocao (idPromocao, item_idItem, data_promocao, data_validade, preco, observacao)
```

VALUES

```
(1, 1, NOW(), NOW(), 0.00, '');
```

-- Populando tabela pedido_itens

```
INSERT INTO pedido_itens (pedido_nrPedido, item_idItem)
```

VALUES

```
(1, 1);
```

-- Populando tabela pedido_cliente

```
INSERT INTO pedido_cliente (cliente_idCliente, pedido_nrPedido, funcionario_matricula)
```

VALUES

```
(1, 1, 1);
```

03.4. Normalização

Para garantir a integridade e eficiência do banco de dados, as tabelas foram normalizadas até a terceira forma norma (3FN). A normalização envolveu a eliminação de redundância e a organização das tabelas em formas normais, garantindo a consistência do banco de dados.

03.5. Dicionário de Dados

- Tabela: Loja

Field	Type	Null	Key	Default	Extra
cnpj	varchar(15)	NO	PRI		
ie	varchar(15)	NO			
nome	varchar(10)	NO			
rua	varchar(40)	NO			
numero	int	YES			
bairro	varchar(15)	NO			
cidade	varchar(15)	NO			
uf	varchar(2)	NO			
cep	varchar(8)	NO			
telefone	varchar(11)	YES			
email	varchar(50)	YES			
ponto_referencia	text	YES			
filial	tinyint(1)	YES			

- Tabela: Funcionário

Field	Type	Null	Key	Default	Extra
matricula	int	NO	PRI		auto_increment
cpf	varchar(11)	NO			
nome	varchar(50)	NO			
data_admissao	datetime	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
funcao	varchar(20)	NO			
salario	decimal(10,2)	YES			
rua	varchar(50)	NO			
numero	int	YES			
bairro	varchar(20)	NO			
cidade	varchar(20)	NO			
uf	varchar(2)	NO			

cep	varchar(8)	NO			
status	char(1)	NO			
senha	varchar(20)	NO			
lojas_cnpj	varchar(15)	NO	MUL		

- Tabela: **Dependente**

Field	Type	Null	Key	Default	Extra
cpf_dependente	varchar(11)	NO	PRI		
funcionario_matricula	int	NO	PRI		
nome	varchar(50)	NO			
data_nascimento	date	NO			

- Tabela: **Cliente**

Field	Type	Null	Key	Default	Extra
idCliente	int	NO	PRI		auto_increment
nome	varchar(10)	NO			
sobrenome	varchar(10)	NO			
rua	varchar(40)	NO			
numero	int	YES			
complemento	varchar(30)	YES			
bairro	varchar(15)	NO			
cidade	varchar(15)	NO			
uf	varchar(2)	NO			
telefone	varchar(11)	NO			
email	varchar(50)	NO	UNI		
ponto_referencia	text	NO			
senha	varchar(20)	NO			
observacao	text	YES			
dataCadastro	datetime	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED

- Tabela: Pedido

Field	Type	Null	Key	Default	Extra
nrPedido	int	NO	PRI		auto_increment
data_hora_pedido	datetime	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
data_hora_prevista_entrega	datetime	NO			
data_hora_saida_entrega	datetime	YES			
data_hora_entrega	datetime	YES			
status_pedido	varchar(20)	YES			
canal_solicitacao_pedido	varchar(20)	YES		site	
canal_entrega	varchar(20)	YES		domicílio	
forma_pagamento	varchar(20)	YES		cartao credito	
valor_total_pedido	decimal(10,2)	YES			

- Tabela: Pedido_cliente

Field	Type	Null	Key	Default	Extra
cliente_idCliente	int	NO	PRI		
pedido_nrPedido	int	NO	PRI		

- Tabela: Pedido_item

Field	Type	Null	Key	Default	Extra
pedido_nrPedido	int	NO	PRI		
item_idItem	int	NO	PRI		
qtde_item	int	YES			

- Tabela: Categoria

Field	Type	Null	Key	Default	Extra
idCategoria	int	NO	PRI		auto_increment
descricao	varchar(20)	YES			

- Tabela: **Item**

Field	Type	Null	Key	Default	Extra
idItem	int	NO	PRI		auto_increment
nome_item	varchar(100)	YES			
descricao	varchar(255)	YES			
ativo	tinyint(1)	YES			
preco	decimal(10,2)	YES			
categoria_idCategoria	int	YES	MUL		

- Tabela: **Promoção**

Field	Type	Null	Key	Default	Extra
idPromocao	int	NO	PRI		auto_increment
item_idItem	int	NO	PRI		
data_promocao	datetime	NO	PRI	CURRENT_TIMESTAMP	DEFAULT_GENERATED
data_validade	datetime	NO			
preco	decimal(10,2)	YES			
observacao	varchar(255)	YES			

04. Desenvolvimento do Backend

O backend foi desenvolvido utilizando a linguagem Java e o framework Spring Boot, que facilitou a criação de APIs RESTful e a integração com o banco de dados MySQL.

Principais Componentes:

- Controladores (Controllers): Gerenciam as requisições HTTP.
- Serviços (Services): Contêm a lógica de negócios.
- Repository (Repositories): Interagem com o banco de dados.

05. Desenvolvimento do Frontend

O frontend foi desenvolvido utilizando HTML, CSS, Bootstrap, JavaScript, jQuery e AJAX, garantindo uma interface responsiva e amigável.

Estrutura:

- HTML: Estrutura básica das páginas.
- CSS: Estilos personalizados.
- Bootstrap: Layout responsivo e componentes prontos.
- JavaScript: Funcionalidade dinâmica e interativa.
- jQuery: Simplificação de manipulação do DOM e efeitos visuais.
- AJAX: Comunicação assíncrona com o servidor para atualização de dados sem recarregar a página

06. Banco de dados

Para o banco de dados foi utilizado a linguagem de programa SQL e o MySQL como o sistema de gerenciamento do banco de dados (SGBD).

07. Desenvolvimento

===== Tecnologias Utilizadas: =====

- FrontEnd:

- HTML, CSS, Bootstrap;
- JavaScript, jQuery, AJAX;

- BackEnd:

- Java;

- Banco de Dados:

- MySQL;

- Comunicação:

- REST API, SpringBoot;

===== Login: =====

- Foi dividido em 2 partes, LoginFuncionario e LoginCliente, pois cada Login é feito de uma maneira. Cliente é com Email e Senha. Funcionario com CPF e Senha.

===== Funcionario: =====

- Status: (FEITO)

- O código checa se o usuário existe no banco de dados e só depois efetua o login.

- Se não encontrar o usuário ou digitar algo errado, não é feito login e apresenta mensagem de erro nos campos.

===== Cliente: =====

- Status: (FEITO)

- O código checa se o usuário existe no banco de dados e só depois efetua o login.

- Se não encontrar o usuário ou digitar algo errado, não é feito login e apresenta mensagem de erro nos campos.

===== Cadastro: =====

- A página de cadastro é utilizada apenas para Clientes, pois o cadastro de funcionários vai ser feito direto dentro das páginas de funcionário.

===== Cadastro Cliente: =====

- Status: (FEITO)

- O código atualiza o banco de dados com o novo usuário cadastrado;

- Quando o cadastro é finalizado, é mostrado uma model confirmando o cadastro e com um botão para levar para a pagina de login.

===== Paginas do Cliente: =====

- CardapioCliente

- Status: (FEITO)

- O que vai fazer:

- Deve possuir em cima do Panel, que irá conter os itens disponíveis no banco de dados, um campo de filtro. O campo de filtro deve poder filtrar por: Categoria de Item, Nome do Item e preço.

- Deve possuir dentro do Panel todos os itens que estão no banco de dados, separados por sua categoria.

- Os itens que estão dentro do panel, deverão conter, dentro de seu card, um botão que ao ser clicado deverá adicionar o item a sacola do cliente.

- Ao clicar no botão para adicionar a sacola, deverá abrir um modal com:

- O nome do item;

- Descrição do item;

- Valor do item;

- Campo para o cliente adicionar uma observação do item;

- Um botão para aumentar e diminuir a quantidade que o cliente quer pedir do item;

- Um botão para adicionar o(s) item(ns) a sacola do cliente. No botão deve conter o nome "Adicionar" a esquerda e o valor total do item que o cliente vai adicionar a sacola. Se o cliente aumentou a quantidade do item que ele está pedindo para um valor maior que um, o valor total que está presente dentro do botão deve ser atualizado.

- Ao clicar no botão dentro do model para adicionar deve:

- Adicionar a quantidade do item que foi pedido pelo cliente a sacola;

- Verificar se houve alguma observação digitada no campo de observação do pedido e, caso sim, enviar essa observação para a sacola do cliente.

- configurações:

- Status: (FEITO)

- O que vai fazer:

- Deve possuir 4 botões que serão:

- Botão Alterar E-mail: Ao clicar deve aparecer um modal com campos de:

- E-mail atual;

- Novo e-mail;

- Senha do cliente;

- Botão para confirmar;

- Ao apertar no botão para confirmar, deve-se confirmar no banco de dados se o email do campo email atual é o mesmo do usuário que está logado, se for

continua para a verificação da senha. Se a senha também estiver igual ao do usuário logado, a alteração do email é feita.

- Ao ser confirmada a mudança do email, deve fechar o modal de mudança de email e aparecer um novo modal;

- O novo modal deve conter um aviso avisando que o usuário será deslogado da plataforma e que ele deve fazer login com o novo email. O novo modal também deve ter um botão que o usuário confirma para então ser deslogado e levado a página de login.

- Ao apertar o botão, o usuário então é deslogado e levado de volta a página de login.

- Botão Alterar Senha:

- Ao clicar deve aparecer um modal com campos de:

- E-mail atual;

- Senha atual do cliente;

- Senha Nova;

- Botão para confirmar;

- Ao apertar no botão para confirmar, deve-se confirmar no banco de dados se o email do campo email atual é o mesmo do usuário que está logado, se for continua para a verificação da senha.

- Se a senha do campo senha atual do cliente for a mesma do usuário logado, a alteração da senha é feita.

- Ao ser confirmada a mudança da senha, deve fechar o modal de mudança da senha e aparecer um novo modal;

- O novo modal deve conter um aviso avisando que o usuário será deslogado da plataforma e que ele deve fazer login com a nova senha. O novo modal também deve ter um botão que o usuário confirma para então ser deslogado e levado a página de login.

- Ao apertar o botão, o usuário então é deslogado e levado de volta a página de login.

- Botão Alterar Endereço:

- Ao clicar deve aparecer um modal com campos de:

- Novo CEP;

- Nova rua;

- Novo numero;

- Novo bairro;

- Novo complemento;

- Novo Ponto de referencia;

- O novo modal deve ter um botão de confirmação; Ao

clicar no botão, o endereço será atualizado no banco com o novo endereço do usuário que está logado.

- Botão Excluir Conta:

- Ao clicar deve aparecer um modal com campos de:
 - Uma mensagem perguntando se o usuário tem certeza que deseja apagar sua conta e avisar que isso é irreversível;
 - Um campo pedindo para digitar o email atual;
 - Um campo pedindo para digitar a senha atual;
 - Um botão para cancelar, que ao clicar irá limpar todos os campos do modal e fechar o modal, voltando para a página de configurações;
 - Um botão para confirmar e quando clicado, deve-se confirmar no banco de dados se o email do campo email atual é o do email do usuário logado, se for igual, vai para verificar a senha;
 - Se a senha do campo senha atual do cliente for a mesma do usuário logado, deve-se aparecer outro modal confirmando que a conta foi apagada com sucesso;
 - O novo modal deverá ter um botão que quando clicado irá apagar o usuário do banco de dados e levar o usuário para a página de login.

- PedidosFeitosCliente:

- Status: (FEITO)

- O que vai fazer:

- Deve conter um container com todos os pedidos que o cliente já fez de forma resumida.

- Os pedidos estarão organizados como uma lista/tabela, um embaixo do outro, em que no título terão os campos:

- Número do pedido;
- Itens do pedido;
- Status do pedido;
- Total

- Cada linha contendo as informações de um pedido;

- No fim da linha de um pedido, tem um botão que abre um modal de mais informações do pedido específico;

- O modal deverá conter:

- Um título com as informações "Informações Pedido Nº - \$", onde o \$ é o número do pedido;

- Abaixo do título, uma tabela com as informações:

- Itens, Quantidade, Status, Total, Data e Hora, Previsão de

Entrega e Forma de Pagamento;

- Abaixo da tabela, terá o Endereço de Entrega do cliente;

- Abaixo do endereço de entrega, terá as observações que o cliente fez no pedido;

- SacolaCliente:

- Status: (FEITO)

- O que vai fazer:

- Deve conter todos os itens que o cliente adicionou pela pagina do cardapio;

- Os itens adicionados devem estar listados um abaixo do outro, como uma lista;

- Cada item individualmente deve conter as informações:

- Nome do item;

- Valor unitario do item;

- Quantidade que foi adicionada pela aba do cardapio;

- Ao lado da quantidade deve conter dois botões, um a

esquerda para diminuir a quantidade e um a direita para aumentar a quantidade do item;

- Preço total do item, ou seja o valor unitario x a quantidade que tem na sacola;

- A observação que foi colocada pela aba do cardapio;

- Um botão a no fim do item a direta, para remover o item da sacola;

- Deve conter um botão para confirmar o envio do pedido;

- Deve conter o valor total da sacola;

- Ao clicar no botão de confirmar, deve aparecer um modal mostrando o endereço do cliente como cadastrado no banco de dados, para que ele confirme o endereço;

- No modal de confirmação de endereço deve conter o botão para confirmar o endereço e um botão de ajuste de endereço;

- O botão de ajuste de endereço deve redirecionar o usuário para a tela de configurações do cliente;

- O botão de confirmar o endereço deve fechar o modal de confirmação de endereço e abrir um modal de pagamento;

- O modal de pagamento será para escolher a forma de pagamento;

- O modal de pagamento deve conter um campo dropdown que seleciona apenas a forma de pagamento do cliente (Dinheiro, cartão ou pix);

- Após selecionar a forma de pagamento, o cliente pode clicar no botão de confirmar pedido dentro do modal de pagamento, que então será de fato confirmado e enviado o pedido;

- Ao clicar no botão de confirmar no modal de pagamento, o pedido do cliente deve ser enviado para o banco de dados para que o novo pedido apareça na pagina de "PedidosEmAberto" do funcionário e "PedidosFeitosCliente" do cliente;

- Ao clicar no botão de confirmar e depois do pedido ter sido enviado, deve aparecer um modal com um alert dentro do modal, informando que o pedido foi enviado com sucesso.

- O pedido enviado agora deve aparecer tanto na pagina "PedidosEmAberto" do funcionário quanto na pagina "PedidosFeitosCliente".

Existem também paginas do funcionario, mas no momento eu estou focado em fazer a pagina "CardapioCliente" no momento. Me ajude com ela.

===== Paginas do Funcionario: =====

- Dashboard:

- Status: (FEITO)

- O que vai fazer:

- Vai possuir 3 botões, cada botão é responsável por realizar uma chamada a API para exibir algo diferente dentro do container da tela. Os botões são:

- Itens vendidos por Periodo;

- Aqui vai aparecer um modal que vai conter 2 campos:

- Data inicial

- Data final

- Ambos os campos podem ser do tipo date.

- Ambos os campos precisam ter um label com seus

respectivos nomes;

- Após digitar as datas e apertar no botão de confirmar,

localizado em baixo dos campos:

- Vai aparecer a resposta da API.

- Aqui a resposta da API vai ser:

- [

- {

- "item_idItem": 1,

- "nome_item": "Abacaxi com Hortelã",

- "total_vendas": 3

- },

-]

- O endpoint a ser usado é GET

'<http://localhost:8080/itens/vendidos-entre/{dataInicial}/{dataFinal}>'

- O retorno serão todos os itens vendidos durante esse periodo, com suas quantidades.

- Aqui o resultado da API deve aparecer uma tabela com os campos:

- "item_idItem", "nome_item" e

"total_vendas";

- Pedido de maior valor por Periodo;

- Aqui vai aparecer um modal que vai conter 2 campos:

- Data inicial

- Data final

- Ambos os campos podem ser do tipo date.
 - Ambos os campos precisam ter um label com seus respectivos nomes;

- Após digitar as datas e apertar no botão de confirmar, localizado em baixo dos campos:

- Vai aparecer a resposta da API.
- Aqui a resposta da API vai ser:

```
[  
 {  
 "data_hora_pedido": "2024-05-26T12:50:07",  
 "pedido_nrPedido": 58,  
 "nome_cliente": "Pedro",  
 "sobrenome_cliente": "Vilas Boas",  
 "valor_total_pedido": 472.1  
 }  
 ]
```

- O endpoint a ser usado é GET

'<http://localhost:8080/pedido-cliente/total-max-entre-datas/{dataInicial}/{dataFinal}>'

- O retorno será o pedido de maior valor.
- Aqui o resultado da API deve aparecer uma tabela

com os campos:

"pedido_nrPedido", "nome_cliente",
 "sobrenome_cliente", "data_hora_pedido" e "valor_total_pedido";
 - N° Pedido por Cliente;
 - Aqui vai aparecer um modal que vai conter 2 campos:
 - Data inicial
 - Data final
 - Ambos os campos podem ser do tipo date.
 - Ambos os campos precisam ter um label com seus respectivos nomes;

- Após digitar as datas e apertar no botão de confirmar, localizado em baixo dos campos:

- Vai aparecer a resposta da API.
- Aqui a resposta da API vai ser:

```
[  
 {  
 "nome_cliente": "Daniel",  
 "sobrenome_cliente": "Rodrigues",  
 "quantidade_pedidos": 1  
 }  
 ]
```

- O endpoint a ser usado é GET

'<http://localhost:8080/pedido-cliente/count-entre-datas/{dataInicial}/{dataFinal}>'

- O retorno serão todos os clientes e suas quantidades de pedido.

- Aqui o resultado da API deve aparecer uma tabela com os campos:

- "nome_cliente", "sobrenome_cliente" e
"quantidade_pedidos";

- Configurações:

- Status: (FEITO)

- O que vai fazer:

- Vai possuir 5 botões dentro de um container. São eles:

- Cadastrar Funcionário. Ao clicar nesse botão, vai abrir um modal que:

- Vai possuir um modalHeader: h5: Texto: Cadastro de Novo

Funcionario

- Vai possuir um modalBody que vai possuir um formulario com campos e seus respectivos tipos:

{

"matricula": 0,

- O campo matricula é do tipo autoincrement no banco, logo ele pode conter o valor 0, ser do tipo hidden e disabled;

"cpf": "string",

"nome": "string",

"dataAdmissao": "2024-05-27T00:42:24.366Z",

- Deve ser pego a data e a hora do momento que foi feito o cadastro do novo funcionario;

"funcao": "string",

- Deve ser um botão do tipo dropdown com as opções: Gerente, Nutricionista, Chef, Atendente e Assistente de Cozinha;

"salario": 0,

"cep": "string",

- Ao digitar o CEP e clicar fora do campo, deve preencher os campos de: Rua, bairro, cidade e UF. Se não encontrar o cep, mostrar o campo como CEP invalido.

"rua": "string",

"numero": 0,

"bairro": "string",

"cidade": "string",

"uf": "string",

"status": "string",

- Deve ser automaticamente colocado como

"A" e o campo ficar como disabled;

```

    "senha": "string",
    "lojasCnpj": "string"
}
```

- Esse campo sempre vai ter o valor padrão

"12345678901234" e ser do tipo hidden;

```
}
```

- Deve-se usar o endpoint POST

"<http://localhost:8080/funcionarios>" ao apertar o botão de salvar funcionario;

- Deve ter um campo com um check perguntando se "Possui Dependente?"

- Se o check for marcado deve aparecer os campos:

```
{
```

```

    "cpfDependente": "string",
    "matriculaFuncionario": 0,
```

- Aqui deve ser GARANTIDO que vai ter a mesma matricula do funcionario que está sendo criado no momento que o botão de salvar for apertado.

- Como vamos utilizar essa tabela de relacionamento fraco, devemos primeiro fazer o POST do funcionario e então o POST do dependente no endpoint "<http://localhost:8080/dependentes>"

```

    "nome": "string",
    "dataNascimento": "2024-05-27T02:18:33.520Z"
```

- Pode ser um campo do tipo de calendario

aqui;

```
}
```

- Vai possuir um modalFooter que vai possuir 2 botões: 1 a direita para salvar o funcionario e 1 a direita para cancelar o cadastro.

- Editar Funcionário. Ao clicar nesse botão, vai abrir um modal que:

- Vai possuir um modalHeader: h5: Texto: Editar Funcionario
- Em que primeiro vai ser solicitado a matricula do

funcionario que deseja editar.

- Em baixo vai possuir um botão que é para avançar na edição do funcionario;

- Ao apertar no botão deve-se usar o endpoint GET

"[http://localhost:8080/funcionarios/{matricula}](http://localhost:8080/funcionarios/{matricula)" para verificar e buscar os dados do funcionario;

- Se não existir fecha o modal e apresenta um alert dizendo que o funcionario não foi encontrado.

- Se existir o funcionario com a matricula, continua para a edição de funcionario no mesmo modal.

- Vai possuir um modalBody que vai possuir um formulario com campos e seus respectivos tipos:

```
{
```

```
    "matricula": 0,
```

- Não deve ser possivel editar, mas deve

mostrar o campo e o valor da matricula do funcionario; Campo deve ser disabled;

```
    "cpf": "string",
```

```
    "nome": "string",
```

```
    "dataAdmissao": "2024-05-27T00:42:24.366Z",
```

- Não deve ser possivel editar, mas deve

mostrar o campo e o valor da data em que foi cadastrado no banco; Campo deve ser disabled;

```
    "funcao": "string",
```

- Deve ser um botão do tipo dropdown com

as opções: Gerente, Nutricionista, Chef, Atendente e Assistente de Cozinha;

```
    "salario": 0,
```

```
    "cep": "string",
```

- Ao digitar o CEP e clicar fora do campo,

deve preencher os campos de: Rua, bairro, cidade e UF. Se não encontrar o cep, mostrar o campo como CEP invalido.

```
    "rua": "string",
```

```
    "numero": 0,
```

```
    "bairro": "string",
```

```
    "cidade": "string",
```

```
    "uf": "string",
```

```
    "status": "string",
```

- Deve ser um botão do tipo dropdown com

as opções: 'Ativo' (Deve-se enviar a letra "A" para o banco), 'Desligado' (Deve-se enviar a letra "D" para o banco), 'Férias' (Deve-se enviar a letra "F" para o banco);

```
    "senha": "string",
```

```
    "lojasCnpj": "string"
```

- Esse campo sempre vai ter o valor padrão

"12345678901234" e ser do tipo hidden;

```
}
```

- Vai possuir um modalFooter que vai possuir 2 botões: 1 a direita para salvar o funcionario e 1 a direita para cancelar o cadastro.

- Deve-se usar o endpoint PUT

"<http://localhost:8080/funcionarios/{matricula}>"

- Cadastrar Item. Ao clicar nesse botão, vai abrir um modal que:

- Vai possuir um modalHeader: h5: Texto: Cadastro de Novo Item

- Vai possuir um modalBody que vai possuir um formulario com

campos e seus respectivos tipos:

```
{
  "idItem": 0,
```

- O campo idItem é do tipo autoincrement no banco, logo ele pode conter o valor 0, ser do tipo hidden e disabled;

```
  "nomeItem": "string",
```

```
  "descricaoItem": "string",
```

```
  "itemAtivo": true,
```

- Deve ser um botão do tipo dropdown com 2

opções: Ativo (Que manda como "true" para o banco) ou Desabilitado (Que manda como "false" para o banco). Por padrão vai ser Ativo.

```
  "precoItem": 0,
```

```
  "idCategoria": 0
```

- Deve ser um botão do tipo dropdown que:

- Deve conter todas as opções de categorias

que existem no banco e mostrar seus nomes nas opções do botão dropdown.

- Deve-se usar o endpoint GET

'<http://localhost:8080/categorias>' para buscar todas as categorias.

- Esse endpoint retorna:

```
"string"
```

```
[{"idCategoria": 0, "descricao":}
```

```
]
```

```
}
```

- Vai possuir um modalFooter que vai possuir 2 botões: 1 a direita

para salvar o item e 1 a direita para cancelar o cadastro.

- Deve-se usar o endpoint POST

'<http://localhost:8080/itens/{idItem}>' ao apertar o botão de salvar funcionario para salvar o item no banco;

- Editar Item. Ao clicar nesse botão, vai abrir um modal que:

- Vai possuir um modalHeader: h5: Texto: Editar Item

- Em que primeiro vai ser solicitado o idItem do item que

deseja editar.

- Em baixo vai possuir um botão que é para avançar na edição do funcionario;

- Ao apertar no botão deve-se usar o endpoint GET

'<http://localhost:8080/itens/{idItem}>' para verificar e buscar os dados do item;

- Se não existir fecha o modal e apresenta um

alert dizendo que o item não foi encontrado.

- Se existir o item com o idItem, continua para a edição do item no mesmo modal.

- Vai possuir um modalBody que vai possuir um formulario com campos e seus respectivos tipos:

```
{
  "idItem": 0,
```

- Não deve ser possivel editar, mas deve

mostrar o campo e o valor do idItem do item; Campo deve ser disabled;

```
  "nomeItem": "string",
  "descricaoItem": "string",
  "itemAtivo": true,
```

- Deve ser um botão do tipo dropdown com 2 opções: Ativo (Que manda como "true" para o banco) ou Desabilitado (Que manda como "false" para o banco). Por padrão vai ser Ativo.

```
  "precoItem": 0,
  "idCategoria": 0
```

- Deve ser um botão do tipo dropdown que:
 - Deve conter todas as opções de categorias

que existem no banco e mostrar seus nomes nas opções do botão dropdown.

- Deve-se usar o endpoint GET

'<http://localhost:8080/categorias>' para buscar todas as categorias.

- Esse endpoint retorna:

```
[
  {
    "idCategoria": 0,
    "descricao": "string"
  }
]
```

- Vai possuir um modalFooter que vai possuir 2 botões: 1 a direita para salvar o funcionario e 1 a direita para cancelar o cadastro.

- Deve-se usar o endpoint PUT

"<http://localhost:8080/itens/{idItem}>"

- Clientes. Ao clicar nesse botão, vai abrir um modal que:

- Listar todos os clientes em uma tabela;
- Essa tabela vai ser dividida em uma tabela com os campos

header:

```
"idCliente": 0,
  "nome": "string",
```

- Deve mostrar o id dos clientes;

- Deve mostrar o nome dos clientes;
`"sobrenome": "string",`
- Deve mostrar o sobrenome dos clientes;
`"telefone": "string",`
- Deve mostrar o telefone do cliente;
`"email": "string",`
- Deve mostrar o email do cliente;
`"dataCadastro": "2024-05-27T01:47:42.276Z"`
- Deve mostrar a data de cadastro do cliente;
- Abaixo da tabela, vai ser mostrado o endereço do cliente como:
 - h5: Texto: Endereço de {nome} + {sobrenome}
 - Abaixo do h5, vai ser mostrado o endereço do cliente:
 - Rua, número, bairro, cidade, UF;
 - Abaixo dos campos de rua, número etc, vai ter:
 - Complemento;
 - Abaixo do campo de complemento, vai ter:
 - Ponto de Referencia;
 - No modalFooter deve ter um botão para fechar o modal.
 - Deve-se usar o endpoint GET '<http://localhost:8080/clientes>'

- AreaDoFuncionario:

- Status: (FEITO)

- O que vai fazer:

- Listar todos os funcionários em uma tabela;

- Essa tabela vai ser dividida em uma tabela com os campos:

- Matrícula

- Nome

- Função

- Salário

- Status

- Ver Mais: Nessa coluna, vão existir apenas botões que irão abrir um modal.

- Ao clicar no botão para abrir o modal, deve-se:

- Abrir o modal que vai conter uma tabela com as mesmas informações que tinham na tabela principal e mais alguns campos, só que agora apenas para o funcionário que foi selecionado.

- h5: Texto: {nome do funcionario}

- Os campos que devem existir nessa primeira tabela são:

- Matrícula

- Função

- Salário

- Status

- CPF
- Data de admissão
- Abaixo dessa tabela deve conter:
 - h5: Texto: Endereço do Funcionario
 - Rua, número, bairro, cidade, UF
 - CEP
- Abaixo do endereço do funcionario, deve ter o seguinte h5:
 - h5: Texto: Dependentes
 - Se o funcionario tiver pelo menos um dependente, outra tabela

com os campos:

- Nome Dependente
- CPF Dependente
- Data de Nascimento Dependente
- Se o funcionario não tiver nenhum dependente, deve ser escrito embaixo do "h5: Dependentes" o texto: "Nenhum dependente."

- PedidosEmAberto:

- Status: (FEITO)
- O que vai fazer:
 - Listar todos os pedidos que ainda não foram aceitos;
 - Os pedidos vão estar organizados em Cards, dentro de um container;
 - Os Cards terão as informações:
 - Nº do pedido no header do card;
 - No corpo do Card:
 - Data em que o pedido foi feito;
 - Quantidade de cada item e o nome do item;
 - Ícone de uma pessoa e ao lado o nome do cliente;
 - Valor total do pedido;
 - Dois botões, um na esquerda para aceitar o pedido e um na direita para negar o pedido;

- Se o pedido foi aceito deve-se:

- Mudar o Status do pedido para: "Em

Preparo". O pedido foi aceito na pagina de pedidos em aberto e NÃO está mais na pagina de pedidos aberto e ESTÁ na pagina de pedidos aceitos. O pedido acabou de ser aceitado. Logo, só precisa mudar o status do pedido. O restante dos campos ficam IDENTICOS como estavam no banco de dados;

- Se o pedido foi negado deve-se:

- Mudar o Status do pedido para: "Negado". O pedido foi negado na pagina de pedidos em aberto. Apenas o campo de "statusPedido" foi mudado para "Negado". E o botão para trocar o status deve estar "disabled".

- Se os itens do pedido forem maior que 50 caracteres, deve-se colocar um "..." no fim do pedido e aparecer um botão no canto direito do header com "..." no texto;

- O botão ao ser pressionado vai:
 - Abrir um modal com as informações:
 - Container com borda preta e rounded. Dentro do container:
 - No header: Texto: Detalhes do Pedido Nº {nrPedido};
 - No corpo:
 - h5: Texto: Itens;
 - Listar quantidade e itens um em baixo do outro;
 - Icone de uma pessoa e ao lado o nome do cliente;
 - No footer:
 - h3: Texto: Total: R\$ {valorTotalPedido}
 - Dois botões, um na esquerda para aceitar o pedido e um na direita para negar o pedido;
 - No topo do modal, deve ter um botão "X" para fechar o modal.

Curl do endpoint "http://localhost:8080/pedidos/{nrPedido}":

```
{
  "nrPedido": 0,
  "dataHoraPedido": "2024-05-26T16:07:49.140Z",
  "dataHoraPrevistaEntrega": "2024-05-26T16:07:49.140Z",
  "dataHoraSaidaEntrega": "2024-05-26T16:07:49.140Z",
  "dataHoraEntrega": "2024-05-26T16:07:49.140Z",
  "statusPedido": "string",
  "canalSolicitacaoPedido": "string",
  "canalEntrega": "string",
  "formaPagamento": "string",
  "valorTotalPedido": 0
}
```

- PedidosAceitos:
 - Status: (FEITO)
 - O que vai fazer:
 - O que deve ser mudado no pedido, quando o Status do pedido for:
 - "Em Aberto": O pedido ainda está na pagina de pedidos aberto e na pagina de pedidos aceitos. Mas não foi aceito ainda. Logo nada precisa mudar.
 - "Em Preparo": O pedido NÃO está mais na pagina de pedidos aberto e

ESTÁ na pagina de pedidos aceitos. O pedido acabou de ser aceitado. Logo, só precisa mudar o status do pedido. O restante dos campos ficam IDENTICOS como estavam no banco de dados.

- "Em Rota": O pedido saiu para entrega, quando for colocado esse Status, deve-se mudar o campo de statusPedido para "Em Rota" e preencher o campo "dataHoraSaidaEntrega" com a data e hora do momento em que foi mudado o Status. O restante dos dados, ficam IDENTICOS como estavam no banco de dados.

- "Concluído": O pedido foi entregue. Logo, deve-se mudar o campo de statusPedido para "Concluído" e precisa mudar o campo "dataHoraEntrega" com a data e hora do momento em que foi mudado o Status. O restante dos campos ficam IDENTICOS como estavam no banco de dados.

- "Negado": O pedido foi negado na pagina de pedidos aberto. Apenas o campo de "statusPedido" foi mudado para "Negado". E o botão para trocar o status deve estar "disabled".

===== Utils: =====

- HeaderCliente:

- Status: (FEITO)
- O que vai fazer:

- HeaderFuncionario:

- Status: (FEITO)
- O que vai fazer:

08. Conclusão

O projeto resultou numa aplicação funcional, com uma interface amigável e um sistema de gerenciamento de dados.

A entrevista com o cliente foi essencial para entendermos as necessidades e desenvolver um sistema que atendeu as expectativas.

A modelagem conceitual, lógica e física garantiu um banco de dados bem estruturado e normalizado. A utilização do Spring Boot facilitou a integração e desenvolvimento da aplicação.

Este documento serve como um registro completo das atividades e decisões tomadas durante o projeto, garantindo a transparência e facilitando futuras manutenções e evoluções do sistema.

Este projeto foi um exemplo da aplicação das boas práticas em desenvolvimento de software e gerenciamento de banco de dados.



PORTAL DO
Temaki