UNIVERSIDADE POSITIVO

Breno Andrade da Silva
Guilherme Pontes Mendonça
Kelvin Matheus Lopes Cardoso

TRABALHO FINAL A1
DESENVOLVIMENTO DE SOFTWARE
PLATAFORMA DE DELIVERY

Professor
Fabio Inocencio Kravetz

CURITIBA

2025

SUMÁRIO

Projeto Final

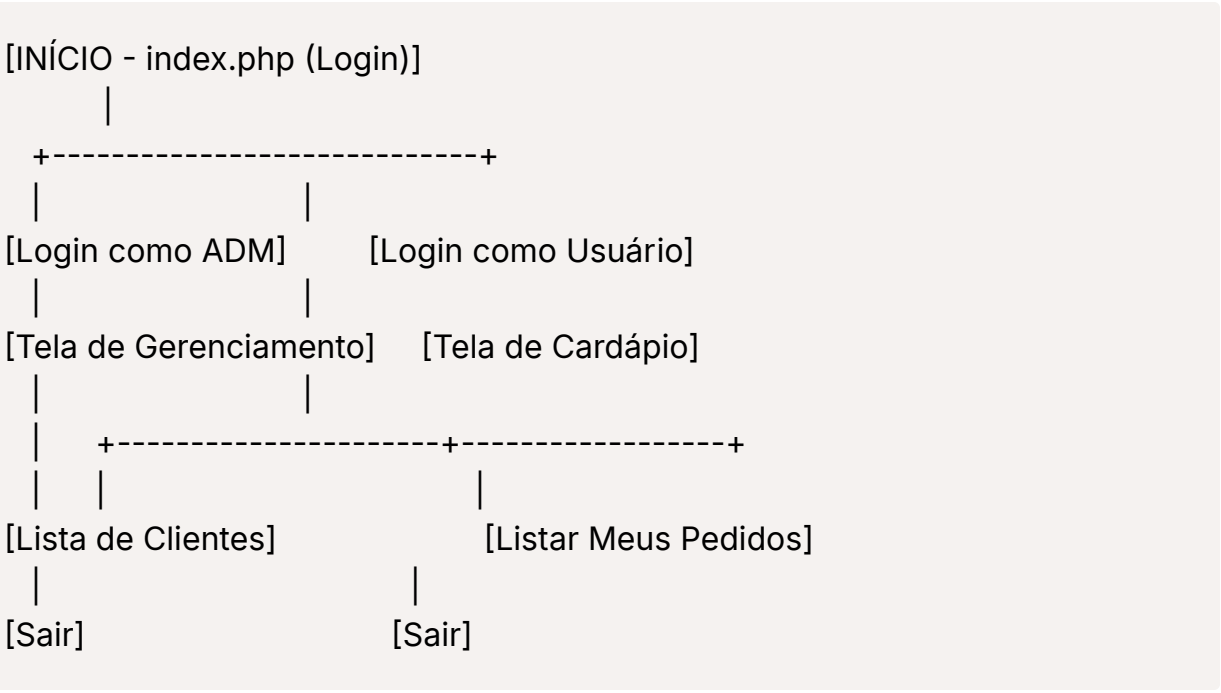
1 - Introdução

O projeto consiste no desenvolvimento de uma plataforma de delivery utilizando a linguagem PHP (versão 7 ou superior), estruturado com o padrão arquitetural **MVC (Model-View-Controller)** e integração ao banco de dados **MySQL** por meio da extensão **PDO**. O sistema é executado localmente via **XAMPP**, utilizando a porta padrão **3306**. Foram implementadas boas práticas de segurança, como o uso de **hash para criptografia de senhas**, **sessões** para controle de autenticação e **cookies** para funcionalidades adicionais. A aplicação abrange funcionalidades como **cadastro de usuários**, **listagem de produtos**, **realização de pedidos** e **gerenciamento administrativo**. Também foi implementado um **CRUD completo (Create, Read, Update e Delete)** para entidades principais do sistema, permitindo a manipulação total dos dados por meio de interfaces funcionais e seguras.

1.2 - Funcionalidades

- Cadastro de usuários
- Login usando criptografia hash bcrypt nativo do PDO
- Armazenamento em cookies e sessões de usuário
- Listagem de clientes, pedidos e produto
- Edição de cadastro de clientes
- Exclusão de cadastros de clientes
- Listagem dinamica de produtos tabelas com estruturas de controle PHP
- Validações front-end com JavaScript e back-end com PHP
- Estilização dinamica com css

2 - Paginas



2.2 Arvore de arquivos

```
APP-DELIVERY/
├── _public/
│   ├── css/
│   ├── img/
│   └── js/
├── config/
│   └── conexao.php
├── controllers/
│   ├── LoginController.php
│   ├── PedidoController.php
│   ├── ProdutoController.php
│   └── UsuarioController.php
├── models/
│   ├── ItemPedido.php
│   ├── Pedido.php
│   ├── Produto.php
│   └── Usuario.php
├── routes/
```

```
│   └── router.php
├── view/
│   ├── adm_pedidos.php
│   ├── cardapio.php
│   ├── clientes_pedido.php
│   ├── login.php
│   ├── template.php
│   ├── teste-pedido.php
│   └── usuarios.php
└── index.php
```

3 - Banco de dados

```
CREATE DATABASE delivery;
USE delivery;

-- Tabela de usuários (clientes e admins)
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    senha VARCHAR(255) NOT NULL,
    tipo ENUM('cliente', 'adm') NOT NULL
);

-- Tabela de produtos
CREATE TABLE produtos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    preco DECIMAL(10,2) NOT NULL,
    imagem VARCHAR(255) -- caminho da imagem (opcional)
);
```

```
-- Tabela de pedidos
CREATE TABLE pedidos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT,
    nome_cliente VARCHAR(100),
    endereco TEXT,
    telefone VARCHAR(20),
    forma_pagamento ENUM('Dinheiro', 'Cartão', 'Pix') NOT NULL,
    status ENUM('Pendente', 'Preparando', 'Saiu para entrega', 'Entregue') DEFAU
    data_pedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id) ON DELETE CASCADE
);

-- Itens de cada pedido (relacionamento muitos para muitos)
CREATE TABLE itens_pedido (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_pedido INT,
    id_produto INT,
    quantidade INT NOT NULL,
    preco_unitario DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (id_pedido) REFERENCES pedidos(id) ON DELETE CASCADE,
    FOREIGN KEY (id_produto) REFERENCES produtos(id)
);
```

O banco de dados `delivery` foi projetado para atender uma plataforma de entrega online, organizando de forma estruturada as informações dos usuários, produtos, pedidos e itens de pedidos. A utilização de **chaves primárias e estrangeiras** visa garantir a integridade referencial entre as tabelas e facilitar consultas relacionais.

A tabela `usuarios` armazena os dados de clientes e administradores, sendo identificados pelo campo `tipo`. A tabela `produtos` contém as opções disponíveis para compra, incluindo nome, descrição, preço e caminho da imagem. Já a tabela `pedidos` registra as compras feitas, relacionando o cliente (`id_usuario`), os dados para entrega e o status do pedido.

O relacionamento entre `pedidos` e `produtos` é feito por meio da tabela intermediária `itens_pedido`, que representa um relacionamento **muitos para muitos**, permitindo

que um pedido tenha vários produtos e cada produto possa pertencer a vários pedidos. Esta tabela também armazena a quantidade de itens e o preço unitário no momento da compra, garantindo o histórico correto mesmo que o preço do produto mude depois.

Com o uso de **chaves estrangeiras**, o banco assegura que os dados permaneçam consistentes. Por exemplo, ao excluir um usuário, seus pedidos também são removidos automaticamente, graças ao `ON DELETE CASCADE`. Esse modelo relacional favorece escalabilidade, facilidade de manutenção e precisão nos relatórios de vendas e gerenciamento de pedidos.

4 - Conexão

```
class Conexao{
    public static function conectar(){
        $host = 'localhost';
        $dbname = 'delivery';
        $user = 'root';
        $pass = '';
        $porta = 3306;

        try{
            // Inclua a porta na string DSN
            $pdo = new PDO("mysql:host=$host;port=$porta;dbname=$dbname;cha
            $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)
            return $pdo;
        }catch(PDOException $e){
            echo "Erro de conexão: " . $e->getMessage();
        }
    }
}
Conexao::conectar();
```

A conexão com o banco de dados no projeto foi implementada utilizando a extensão **PDO (PHP Data Objects)**, que oferece uma interface segura e flexível

para comunicação com o MySQL, rodando na porta padrão **3306**. Para facilitar a integração entre os diferentes ambientes de trabalho dos membros do grupo, foi criada uma classe chamada `Conexao` que centraliza a lógica de conexão, com variáveis separadas para **host**, **porta**, **nome do banco**, **usuário** e **senha**.

Essa abordagem facilita a **configuração individual** do ambiente, permitindo que cada integrante altere apenas os valores das variáveis conforme necessário, sem afetar o restante do sistema.

5 - Estrutura MVC

O projeto foi estruturado com base no padrão arquitetural **MVC (Model-View-Controller)**, que visa organizar o código de forma modular e facilitar a manutenção e escalabilidade do sistema. Na camada **Model**, estão as classes responsáveis pela representação dos dados e regras de negócio, como `Usuario`, `Produto`, `Pedido` e `ItemPedido`. A camada **Controller** atua como intermediária entre os dados e a interface do usuário, sendo composta por controladores específicos como `LoginController`, `PedidoController`, `ProdutoController` e `UsuarioController`, que lidam respectivamente com autenticação, gerenciamento de pedidos, produtos e usuários.

Na camada **View**, encontram-se os arquivos responsáveis pela interface com o usuário, como `cardapio.php`, `login.php`, `adm_pedidos.php`, `clientes_pedido.php`, `usuarios.php`, entre outros. A navegação entre essas páginas é controlada por meio do arquivo `router.php`, localizado na pasta `routes/`, que é incluído no `index.php`, o ponto de entrada principal do sistema. O roteador utiliza uma estrutura `switch` para identificar a página solicitada através da URL (`?pagina=nome_da_pagina`) e, com base nisso, chama as funções correspondentes, carregando em ordem o cabeçalho (`header`), o conteúdo da view e o rodapé (`footer`). Esse processo garante uma separação clara de responsabilidades e uma navegação dinâmica, mantendo a lógica de exibição consistente e organizada. A classe `Conexao` centraliza a configuração de acesso ao banco de dados via PDO, promovendo flexibilidade na conexão entre ambientes.

6 - Funcionalidades Principais

Os `controllers` recebem os dados da requisição, tratam a lógica de controle e, quando necessário, utilizam os `models` para processar ou manipular os dados.

6.1 - Página Login

6.1.1 - Cadastro de usuarios (POST)

- Funções utilizadas:

`controllers/LoginControllers.php` `cadastrarUsuario()` `linha 29:72`

`models/Usuario.php` `inserir()` `linha 11:31`

6.1.2 - Login (GET)

- Funções utilizadas:

`controllers/LoginControllers.php` `logarUsuario()` `linha 74:127`

`models/Usuario.php` `login()` `linha 33:46`

`logarUsuario()` utiliza a função `fetch(PDO::FETCH_ASSOC)` para criar um array, percorrer os usuarios do banco de dados para validar se realmente existe

6.2 - Página Catalogo

6.2.1 - Listar produtos (GET)

`model/Produto.php` `listarProdutos()` `linha 1:21`

`view/cardapio.php` `foreach()` `linha 15:38`

`model` cria a requisição do banco de dados e retorna um array que é iterado com `foreach()`

6.2.1 - Adicionar ao Carrinho via JSON

- **Funções utilizadas:**

`public/enviarPedido.js` `addToCart()` `linha 18:32`

`public/enviarPedido.js` `updateCartDisplay()` `linha 35:89`

A função `addToCart()` adiciona produtos ao array `cart` em JavaScript, simulando o carrinho de compras do usuário. Caso o item já exista no carrinho, a quantidade é incrementada. O conteúdo visual é atualizado pela função `updateCartDisplay()`, que exibe os itens, quantidades e total.

6.2.2 - Criar Pedido (POST)

- **Funções utilizadas:**

`controllers/PedidoController.php` `adicionarPedido()` `linha 1:60 aprox.`

`models/Pedido.php` `inserirPedido()` `linha 9:31 aprox.`

A função `enviarPedido()` no front-end empacota os dados do carrinho e do cliente em JSON e envia via `fetch()` para `index.php?pagina=pedido/adicionar`.

No back-end, a função `adicionarPedido()` valida os dados recebidos, autentica o usuário via sessão e salva o pedido no banco com `Pedido::inserirPedido()`. Retorna um JSON com sucesso ou erro.

6.3.1 - Listar usuarios e pedidos(GET)

- `models/Pedido.php`: Define a estrutura e operações no banco para os **pedidos**.
- `models/Usuario.php`: Define a estrutura e operações para os **usuários/clientes**.
- `controllers/PedidoController.php`: Controla as requisições relacionadas a **pedidos**.
- `controllers/UsuarioController.php`: Controla as requisições relacionadas a **usuários**.

Na camada **View**, são utilizadas **estruturas de controle** (como `if`, `else`) e **laços de repetição** (`foreach`, `for`) para percorrer os dados retornados pelos controllers e **montar dinamicamente as tabelas** com os clientes e pedidos, exibindo cada item de forma organizada na interface da aplicação.

6.3.2 - Editar Usuário (POST)

- Funções utilizadas:

`controllers/UsuarioController.php` `editar()` `linha 11:13`

`models/Usuario.php` `atualizar()` `linha 46:64`

A função `editar()` recebe os dados do formulário e os repassa ao model `atualizar()`, que atualiza o nome, email, tipo e opcionalmente a senha do usuário usando `UPDATE` com `bindParam`.

6.1.4 - Excluir Usuário (POST)

- Funções utilizadas:

`controllers/UsuarioController.php` `excluir()` `linha 15:17`

`models/Usuario.ph` `p` `excluir()` `linha 66:73`

A função `excluir()` chama `excluir()` no model, que executa o `DELETE FROM usuarios WHERE id = :id`, removendo o registro com segurança usando `prepare()` e `bindParam`.

7 - Cookies e Sessões

As iniciação das sessões estão centralizadas todas no arquivo `router.php`

7.1 - Sessões (`$_SESSION`)

- `controllers/LoginController.php`
  - Armazena dados do usuário logado: `$_SESSION['usuario']`
  - Mensagens de feedback: `$_SESSION['mensagem']`
  - Controle de permissões: `$_SESSION['permissao']`
- `controllers/PedidoController.php`
  - Mensagens de sucesso/erro: `$_SESSION['msg_sucesso']`, `$_SESSION['msg_erro']`
  - Verifica autenticação: `$_SESSION['usuario']['id']`
- `controllers/UsuarioController.php`
  - Mensagens de sucesso/erro: `$_SESSION['msg_sucesso']`, `$_SESSION['msg_erro']`
- `view/template.php`

    - Exibe nome do usuário logado: `$_SESSION['usuario']['nome']`
- `routes/router.php`
    - Inicializa a sessão com `session_start()`

7.2 - Cookies (`$_COOKIE`)

- `controllers/LoginController.php`
    - Armazena email e senha: `usuario_email`, `usuario_senha` (quando "lembrar de mim" está marcado)
    - Expiração configurada para 30 dias (`86400 * 30`)
- `view/login.php`
    - Preenche automaticamente os campos do formulário com os valores dos cookies
    - Marca checkbox "lembrar de mim" se os cookies existirem
- `controllers/LoginController.php`
    - Método `sair()` limpa as sessões e cookies ao deslogar

8 - CSS, HTML E JS

Estilo css, html e js inserido dinamicamente de acordo com o parametro `pagina=nome_da_pagina`

O que torna as opções do menu dinamicas com o tipo de usuario