Faculdade Municipal de Palhoça
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Programação I
Professor: Rafael Novo da Rosa
Alunos: Lucas Schutz e Vitor Emanuel

Banco Fmp

Palhoça
2025

1. Introdução

Banco Fmp

Este trabalho apresenta o BancoFMP, um sistema web de gestão bancária desenvolvido em PHP utilizando a arquitetura MVC e banco de dados MySQL. O nosso objetivo é documentar a criação de uma aplicação funcional que realiza operações financeiras essenciais, tais como cadastro de usuários, autenticação e transferências de valores.

A relevância deste projeto está em demonstrar na prática como a Programação Orientada a Objetos e a organização em camadas contribuem para um código mais estruturado e fácil de manter. Somado a isso, o uso do MySQL assegura a integridade e a segurança das informações tratadas. O texto a seguir descreve a trajetória do desenvolvimento e abrange desde as decisões técnicas iniciais até os testes de validação do sistema.

2. Fundamentação teórica

2.1 Programação Orientada a Objetos (POO)

A POO organiza o software em torno de classes (modelos abstratos que definem atributos e métodos) e objetos (instâncias com dados próprios). Os quatro pilares mais relevantes para este projeto são:

- Encapsulamento: protege o estado interno das classes e expõe apenas interfaces controladas (getters/setters ou métodos específicos), reduzindo efeitos colaterais.
- Herança: permite que classes derivadas reutilizem e especializem comportamento de classes base.
- Polimorfismo: possibilita que a mesma interface seja implementada por diferentes classes, facilitando substituição e extensão.
- Abstração: favorece modelar entidades do domínio (Usuário, Conta, Transação) de forma direta.

Esses conceitos tornam o código mais legível, testável e fácil de manter, o que é essencial em aplicações que manipulam valores financeiros.

2.2 Padrão MVC (Model-View-Controller)

O padrão MVC separa a aplicação em três camadas:

- Model: representa os dados e regras de negócio; encapsula acesso ao banco e lógica de domínio.
- View: responsável pela apresentação (HTML/PHP), formulários e templates.
- Controller: recebe requisições, valida dados, invoca Models e decide qual View exibir.

A separação em camadas facilita localizar e corrigir erros, permite desenvolvimento paralelo e melhora a organização geral do projeto.

2.3 MySQL e Integração com PHP

Bancos relacionais como MySQL são adequados para sistemas que exigem integridade e transações (ACID). Em PHP, a integração pode ser feita via PDO (PHP Data Objects) ou MySQLi; neste projeto foi adotado PDO para permitir tratamento de exceções, prepared statements e portabilidade. A utilização de transações (beginTransaction, commit, rollBack) é essencial para operações financeiras que requerem atomicidade.

3. Metodologia

3.1 Arquitetura do sistema

O sistema utiliza um padrão de Front Controller. Todas as requisições são direcionadas para o arquivo public/index.php, que atua como um roteador central. Ele verifica o parâmetro ?route= na URL, instancia o Controller adequado (ex: UsuarioController ou ContaController) e chama o método correspondente. Isso centraliza a gestão de dependências e o início da sessão (session_start), garantindo que nenhuma página seja acessada sem passar pelas configurações iniciais.

A organização de diretórios foi definida para reforçar a separação:

- /config — classes de configuração e conexão com banco (ex.: Database.php).
- /model — classes de domínio (ex.: Usuario.php, Conta.php, Transacao.php).
- /controller — classes que tratam requisições (ex.: UsuarioController.php, ContaController.php).
- /view — arquivos de interface (templates, formulários).
- /public — arquivos públicos, assets, index.php de entrada.

3.2 Implementação

Estrutura de Pastas: A organização física dos arquivos reflete a separação de responsabilidades:

- config/: Database.php (Conexão Singleton PDO).
- controller/: Lógica de controle (ContaController.php, UsuarioController.php).
- model/: Regras de negócio e acesso a dados (Conta.php, Transacao.php, Usuario.php).

- view/: Interfaces HTML divididas em subpastas (admin, conta, template, usuario).
- public/: Ponto de entrada (index.php) e ativos (css/style.css).

Trechos de Código:

- Model (Destaque para a Transação ACID): A classe Conta.php demonstra o uso de transações bancárias seguras. O método realizarTransferencia garante a atomicidade: ou o débito e o crédito ocorrem juntos, ou nada acontece.

Código:

```
public function realizarTransferencia($id_origem, $id_destino, $valor, $taxa, $metodo) {
    try {
        $this->db->beginTransaction();
        $saldo_origem = $this->getSaldo($id_origem);
        $this->updateSaldo($id_origem, $saldo_origem - ($valor + $taxa));
        $saldo_destino = $this->getSaldo($id_destino);
        $this->updateSaldo($id_destino, $saldo_destino + $valor);
        $this->db->commit();
        return true;
    } catch (Exception $e) {
        $this->db->rollBack();
        return $e->getMessage();
    }
}
```

Além das transações complexas, o sistema implementa operações de leitura (Read) essenciais para o dashboard, como a consulta de saldo:

Código:

```
public function getSaldo($id_usuario) {
    $stmt = $this->db->prepare("SELECT saldo FROM contas WHERE id_usuario = ?");
```

```
    $stmt->execute([$id_usuario]);
    $resultado = $stmt->fetch();
    return $resultado ? $resultado['saldo'] : 0;
}
```

Para contemplar as operações de criação (Create) do CRUD, o sistema utiliza instruções INSERT protegidas por prepared statements. Abaixo, o método utilizado para registrar uma nova transação no histórico:

Código:

```
// Exemplo de Criação (Create) em Transacao.php
public function registrar($id_origem, $id_destino, $valor, $taxa, $metodo) {
    $sql = "INSERT INTO transacoes (id_usuario_origem, id_usuario_destino, valor, taxa, metodo) VALUES (?, ?, ?, ?, ?)";
    $stmt = $this->db->prepare($sql);
    return $stmt->execute([$id_origem, $id_destino, $valor, $taxa, $metodo]);
}
```

- Controller (Destaque para Validação): O ContaController.php valida as regras de negócio antes de chamar o Model, impedindo transferências inválidas.

Código:

```
if ($valor <= 0) {
    $erro = "O valor da transferência deve ser positivo.";
} elseif ($usuario_destino['id'] == $id_origem) {
    $erro = "Você não pode transferir para si mesmo.";
} elseif ($saldo_origem < $valor_total_debito) {
    $erro = "Saldo insuficiente.";
}
```

- Estrutura do Banco de Dados (SQL): O banco Bancophp foi estruturado com chaves estrangeiras (FOREIGN KEY) para garantir a integridade referencial entre contas, usuários e transações.

Código:

```
CREATE TABLE `transacoes` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `id_usuario_origem` INT NOT NULL,
  `id_usuario_destino` INT NOT NULL,
  `valor` DECIMAL(10, 2) NOT NULL,
  `taxa` DECIMAL(10, 2) NOT NULL DEFAULT 0.00,
  `metodo` VARCHAR(20) NOT NULL,
  `data` DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (`id_usuario_origem`) REFERENCES `usuarios`(`id`),
  FOREIGN KEY (`id_usuario_destino`) REFERENCES `usuarios`(`id`)
);
```

3.3 Conexão com banco de dados

A conexão entre o PHP e o MySQL foi desenvolvida utilizando a biblioteca PDO (PHP Data Objects), que oferece uma interface mais segura, flexível e moderna para interação com bancos de dados. No arquivo Database.php, foi adotado o padrão Singleton (ou uma variação estática), garantindo que toda a aplicação utilize apenas uma instância de conexão durante sua execução. Isso evita aberturas repetidas e desnecessárias de conexão, reduzindo custo de processamento e aumentando a eficiência do sistema.

A configuração da conexão também define o modo de erro como PDO::ERRMODE_EXCEPTION, o que faz com que qualquer falha ocorrida nas operações com o banco de dados seja lançada como uma exceção. Esse comportamento facilita a depuração, já que o erro pode ser capturado e tratado de forma controlada, permitindo que o sistema responda de maneira mais clara e segura em situações inesperadas — como falhas de consulta, perda de conexão ou dados inválidos.

4. Considerações finais

O desenvolvimento do BancoFMP evidenciou a importância de aplicar boas práticas de engenharia de software: encapsulamento das regras de negócio em models, roteamento centralizado em controllers e apresentação limpa em views. As principais decisões de projeto (uso de PDO, transações, prepared statements e validações em múltiplas camadas) visaram garantir segurança e integridade dos dados.

4.1 Desafios e soluções

- Integridade de transações: resolvido com o uso de transações PDO (beginTransaction / commit / rollBack).
- Autoload e roteamento sem framework: foi implementado um autoloader e um roteador simples no index.php para evitar requires dispersos.
- Validação e segurança: uso de password_hash para senhas, prepared statements para evitar SQL Injection e validação front/back-end.

4.2 Testes e validação

A validação dos dados de entrada ocorre em três níveis de segurança:

1. Front-end (HTML5): Uso de atributos como type="email", pattern="\d{6}" e maxlength="6" nos formulários de login e transferência para garantir que a senha possua exatamente 6 dígitos numéricos antes do envio.
2. Back-end (Controller): O PHP realiza a sanitização e verificação lógica. Exemplo: filter_var($email, FILTER_VALIDATE_EMAIL) para garantir formato de e-mail e is_numeric($senha) para bloquear injeção de caracteres não numéricos na senha.
3. Segurança de Senha: As senhas nunca são salvas em texto plano. O sistema utiliza password_hash() no registro e password_verify() no login, garantindo criptografia robusta.

5. Conclusão

O projeto BancoFMP cumpriu com êxito os objetivos propostos, resultando em um sistema bancário funcional que aplica de forma coerente os pilares da Programação Orientada a Objetos, a arquitetura MVC e a persistência de dados em MySQL. As operações essenciais — cadastro, autenticação e transferências — foram implementadas com sucesso, garantindo a integridade financeira através do uso rigoroso de transações ACID.

O desenvolvimento deste trabalho permitiu consolidar o conhecimento teórico na prática, especialmente no que tange à segurança da informação com o uso de prepared statements e criptografia de senhas, além de reforçar a importância da separação de responsabilidades para a manutenção do código.

Como melhorias futuras para a evolução do sistema, sugere-se a implementação de autenticação em dois fatores (2FA) para aumentar a segurança, a geração de comprovantes em PDF e a criação de relatórios gerenciais mais detalhados no painel administrativo. Além disso, a interface poderia ser migrada para um framework front-end moderno, proporcionando uma experiência de usuário ainda mais fluida.

6. Referências

SILVA, Cleuton. Os 4 Pilares da Programação Orientada a Objetos. Disponível em: https://www.dio.me/articles/os-4-pilares-da-programacao-orientada-a-objetos-SSU4Q9. Último acesso em: 21 de novembro de 2025.

GASPAROTTO, Henrique. POO: Os 4 pilares da Programação Orientada a Objetos. Disponível em: https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264. Último acesso em: 21 de novembro de 2025.

SOUZA, Ângelo. Arquitetura MVC: Entendendo o Modelo-Visão-Controlador. Disponível em: https://www.dio.me/articles/arquitetura-mvc-entendendo-o-modelo-visao-controlador. Último acesso em: 21 de novembro de 2025.

MEDEIROS, Higor. Introdução ao Padrão MVC: Primeiros passos na Arquitetura MVC. Disponível em: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308. Último acesso em: 21 de novembro de 2025.