link código fonte:

https://github.com/HenriqueSilva7/sistema-controle-estoque

Diferença entre Padrão MVC e Arquitetura em Três Camadas

Embora o Padrão Model-View-Controller (MVC) e a Arquitetura em Três Camadas (3-Tier) sejam utilizados para organizar e estruturar aplicações, eles atuam em níveis de abstração diferentes e com propósitos distintos.

Padrão MVC (Model-View-Controller)

O MVC é um **padrão de design de software** usado para separar a lógica de uma aplicação em três componentes interconectados, facilitando a manutenção e a reutilização do código, especialmente em interfaces de usuário (UIs).

- **Model:** Representa os dados e a lógica de negócios da aplicação. O Model lida com o estado, o comportamento e a persistência dos dados. Ele notifica a View quando há mudanças.
- **View:** É a interface de usuário. A View exibe os dados do Model para o usuário e envia as requisições do usuário para o Controller.
- **Controller:** Atua como um intermediário. Ele recebe a entrada do usuário (requisições), manipula o Model e, em seguida, seleciona uma View para exibir a resposta.

O foco do MVC é na **separação de responsabilidades da interface de usuário**, garantindo que a lógica de negócio, a exibição e o controle de interação sejam independentes.

Arquitetura em Três Camadas (3-Tier)

A Arquitetura em Três Camadas é um **estilo de arquitetura de software** que divide a aplicação em três camadas lógicas e, frequentemente, físicas, separadas, que se comunicam de forma linear.

- **Camada de Apresentação (Presentation Tier):** É a camada superior, responsável pela interface do usuário. Ela envia as requisições para a camada de Lógica de Negócios e exibe os dados para o usuário.
- **Camada de Lógica de Negócios (Business Logic Tier):** É a camada intermediária que contém toda a lógica de negócio, processamento de dados e regras da aplicação. Ela atua como um "cérebro" da aplicação.

- **Camada de Dados (Data Tier):** É a camada mais baixa, responsável por gerenciar o armazenamento de dados, como bancos de dados, arquivos ou outros serviços de persistência. A Camada de Lógica de Negócios se comunica com esta camada.

O foco da Arquitetura em Três Camadas é na **distribuição física ou lógica dos componentes da aplicação** para melhorar a escalabilidade, segurança e gerenciamento de dependências.

Tabela Comparativa

Característica	Padrão MVC	Arquitetura em Três Camadas
Nível	Padrão de Design (foco na UI)	Estilo de Arquitetura (foco na distribuição)
Propósito	Separação da lógica da UI	Separação física/lógica de componentes
Comunicação	Interativa (Model notifica View, View envia para Controller)	Linear (Apresentação -> Lógica -> Dados)
Onde é usado	Aplicações com UI (Web, Desktop, Mobile)	Aplicações distribuídas, empresariais, cliente-servidor
Camadas	Model, View, Controller	Apresentação, Lógica de Negócios, Dados

Conclusão

A principal diferença é que o **MVC** é um padrão para organizar o código **dentro da camada de Apresentação** para separar a lógica da interface do usuário. Já a **Arquitetura em Três Camadas** é uma forma de organizar a aplicação **inteira**, geralmente em diferentes servidores ou camadas lógicas, para gerenciar as dependências e a escalabilidade da aplicação. É perfeitamente possível e comum ter uma arquitetura em Três Camadas onde a Camada de Apresentação é organizada usando o Padrão MVC.

Sistema de Controle de Estoque de Produtos

Trabalho Acadêmico - Arquitetura de Software e Programação

1. QUESTÕES TEÓRICAS

1.1 Diferença entre MVC e Arquitetura em Três Camadas

Padrão MVC (Model-View-Controller)

O padrão MVC é um padrão arquitetural que separa a aplicação em três componentes interdependentes:

- **Model (Modelo)**: Representa os dados e a lógica de negócio da aplicação
    - Gerencia o estado dos dados
    - Implementa regras de negócio
    - Notifica a View sobre mudanças nos dados
    - *Exemplo*: Classe `Produto` com métodos para validar preço, calcular valor total
- **View (Visão)**: Responsável pela apresentação dos dados ao usuário
    - Interface gráfica ou textual
    - Exibe informações do Model
    - Captura entrada do usuário
    - *Exemplo*: Formulário para cadastro de produtos, tela de listagem
- **Controller (Controlador)**: Gerencia a interação entre Model e View
    - Processa entrada do usuário
    - Atualiza o Model baseado nas ações
    - Seleciona a View apropriada
    - *Exemplo*: `ProdutoController` que recebe requisição de cadastro

Fluxo MVC:

Usuário → Controller → Model → Controller → View → Usuário

Arquitetura em Três Camadas (Three-Tier Architecture)

A arquitetura em três camadas separa a aplicação em camadas lógicas distintas:

- **Camada de Apresentação (Presentation Layer)**
    - Interface do usuário
    - Responsável pela interação com o usuário
    - *Exemplo*: Páginas web, aplicativo desktop
- **Camada de Lógica de Negócio (Business Logic Layer)**
    - Processamento de dados
    - Regras de negócio

    - Validações
    - *Exemplo*: Serviços que calculam estoque mínimo, aplicam descontos
- **Camada de Dados (Data Access Layer)**
    - Acesso ao banco de dados
    - Persistência de dados
    - Operações CRUD
    - *Exemplo*: Classes DAO/Repository para acesso aos produtos

Fluxo Três Camadas:

Apresentação → Lógica de Negócio → Dados → Lógica de Negócio → Apresentação

Principais Diferenças

Aspecto	MVC	Três Camadas
Foco	Separação de responsabilidades em aplicações interativas	Separação lógica e física da aplicação
Comunicação	Model notifica View diretamente	Comunicação sequencial entre camadas
Distribuição	Geralmente em uma única aplicação	Pode ser distribuído em diferentes servidores
Flexibilidade	Maior acoplamento entre componentes	Maior independência entre camadas
Uso Principal	Aplicações desktop e web	Aplicações empresariais distribuídas

1.2 Arquitetura Utilizada no Projeto

Para este sistema de controle de estoque, utilizaremos uma **combinação híbrida** das duas abordagens, aproveitando as vantagens de cada uma:

Estrutura Proposta:

1. **Camada de Apresentação** (inspirada na View do MVC)
    - Interface console ou web
    - Formulários de entrada
    - Exibição de relatórios
2. **Camada de Controle** (Controller do MVC)
    - `ProdutoController`: gerencia operações com produtos
    - `EstoqueController`: controla movimentações de estoque
    - Validação de entrada do usuário
3. **Camada de Negócio** (Model + Business Logic)
    - Classes `Produto`, `Estoque`
    - Regras de negócio (estoque mínimo, cálculos)
    - Validações de dados
4. **Camada de Dados** (Data Access)
    - Classes Repository/DAO
    - Conexão com banco de dados
    - Operações CRUD

Justificativa da Escolha:

- **Separação clara** de responsabilidades
- **Facilidade de manutenção** e teste
- **Escalabilidade** para futuras funcionalidades
- **Reutilização** de código entre diferentes interfaces
- **Testabilidade** independente de cada camada

---

2. DIAGRAMAS UML

2.1 Diagrama de Classes

classDiagram

class Produto {

-int idProduto

-string nome

-decimal preco

-int quantidade

+Produto(nome, preco)

```
    +getId() int
    +getNome() string
    +getPreco() decimal
    +getQuantidade() int
    +setQuantidade(int) void
    +calcularValorTotal() decimal
    +validarDados() boolean
}

class EstoqueService {
    -ProdutoRepository repository
    +cadastrarProduto(Produto) boolean
    +buscarProduto(int) Produto
    +listarProdutos() List~Produto~
    +incrementarEstoque(int, int) boolean
    +decrementarEstoque(int, int) boolean
    +verificarEstoqueMinimo(int) boolean
}

class ProdutoRepository {
    -Connection conexao
    +inserir(Produto) boolean
    +buscarPorId(int) Produto
    +listarTodos() List~Produto~
    +atualizar(Produto) boolean
    +deletar(int) boolean
```

```
    +atualizarQuantidade(int, int) boolean
}

class ProdutoController {
    -EstoqueService service
    +cadastrarProduto(dados) void
    +consultarProduto(id) void
    +listarProdutos() void
    +adicionarEstoque(id, quantidade) void
    +removerEstoque(id, quantidade) void
}

class EstoqueView {
    +exibirMenu() void
    +solicitarDadosProduto() Produto
    +exibirProduto(Produto) void
    +exibirListaProdutos(List~Produto~) void
    +exibirMensagem(string) void
}

ProdutoController --> EstoqueService
EstoqueService --> ProdutoRepository
EstoqueService --> Produto
ProdutoController --> EstoqueView
ProdutoRepository --> Produto
```

2.2 Diagrama de Sequência - Cadastro de Produto

```
sequenceDiagram
    participant U as Usuário
    participant V as EstoqueView
    participant C as ProdutoController
    participant S as EstoqueService
    participant R as ProdutoRepository
    participant BD as Banco de Dados

    U->>V: Seleciona "Cadastrar Produto"
    V->>U: Solicita dados (nome, preço)
    U->>V: Informa dados
    V->>C: cadastrarProduto(dados)
    C->>S: cadastrarProduto(produto)
    S->>S: validarDados(produto)
    alt Dados válidos
        S->>R: inserir(produto)
        R->>BD: INSERT INTO produto...
        BD->>R: Confirmação
        R->>S: true
        S->>C: true
        C->>V: exibirMensagem("Sucesso")
        V->>U: "Produto cadastrado!"
    else Dados inválidos
        S->>C: false
        C->>V: exibirMensagem("Erro")
```

```
    V->>U: "Dados inválidos!"
  end
```

2.3 Diagrama de Sequência - Movimentação de Estoque

```
sequenceDiagram
  participant U as Usuário
  participant V as EstoqueView
  participant C as ProdutoController
  participant S as EstoqueService
  participant R as ProdutoRepository
  participant BD as Banco de Dados

  U->>V: Seleciona "Alterar Estoque"
  V->>U: Solicita ID e quantidade
  U->>V: Informa dados
  V->>C: adicionarEstoque(id, qtd)
  C->>S: incrementarEstoque(id, qtd)
  S->>R: buscarPorId(id)
  R->>BD: SELECT * FROM produto WHERE id=?
  BD->>R: Dados do produto
  R->>S: produto
  alt Produto existe
    S->>R: atualizarQuantidade(id, novaQtd)
    R->>BD: UPDATE produto SET quantidade=?
    BD->>R: Confirmação
    R->>S: true
```

```
        S->>C: true
        C->>V: exibirMensagem("Estoque atualizado")
        V->>U: "Operação realizada!"
    else Produto não existe
        S->>C: false
        C->>V: exibirMensagem("Produto não encontrado")
        V->>U: "Erro: ID inválido!"
    end
```

---

3. ESPECIFICAÇÕES TÉCNICAS

3.1 Estrutura do Banco de Dados

```
CREATE DATABASE controle_estoque;

USE controle_estoque;


CREATE TABLE produto (
  id_produto INT NOT NULL AUTO_INCREMENT,
  nome VARCHAR(45) NOT NULL,
  preco DECIMAL(10,2) NOT NULL,
  quantidade INT DEFAULT 0,
  data_cadastro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  data_atualizacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (id_produto),
  INDEX idx_nome (nome)
);
```

```
-- Dados de teste

INSERT INTO produto (nome, preco, quantidade) VALUES

('Notebook Dell', 2500.00, 10),

('Mouse Logitech', 45.90, 25),

('Teclado Mecânico', 180.00, 8);
```

3.2 Funcionalidades do Sistema

1. **Cadastro de Produtos**
    - Inserir nome e preço
    - Validação de dados obrigatórios
    - Quantidade inicial zero
2. **Consulta de Produtos**
    - Buscar por ID
    - Listar todos os produtos
    - Exibir informações completas
3. **Controle de Estoque**
    - Incrementar quantidade
    - Decrementar quantidade
    - Validar estoque negativo
4. **Relatórios Básicos**
    - Produtos com estoque baixo
    - Valor total do estoque

3.3 Padrões e Boas Práticas

- **Repository Pattern**: Para acesso a dados
- **Service Layer**: Para lógica de negócio
- **Dependency Injection**: Para baixo acoplamento
- **Exception Handling**: Tratamento adequado de erros
- **Data Validation**: Validação em múltiplas camadas
- **SOLID Principles**: Especialmente SRP e DIP