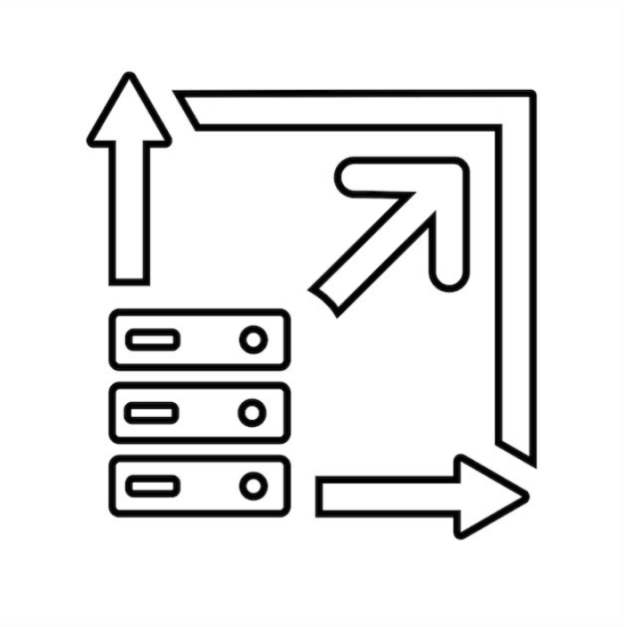
Antônio José Santos do Nascimento

Atividade I

Relatório Acadêmico: Padrões Arquiteturais com Model-View-Controller (MVC).

1. Introdução aos Padrões Arquiteturais Padrões arquiteturais são soluções estruturais e organizacionais testadas e comprovadas para problemas recorrentes no desenvolvimento de software. Eles fornecem um *blueprint* (projeto) para a estrutura de um sistema, definindo a maneira como os componentes interagem e se comunicam. A utilização desses padrões é crucial para:

- **Manutenibilidade:** Facilita a localização, modificação e correção de código.
- **Escalabilidade:** Permite que o sistema cresça e se adapte a novas funcionalidades e cargas de usuários.
- **Reutilização:** Promove a reutilização de código e lógica de design em diferentes partes do sistema.



Clareza: Torna o sistema mais fácil de entender por novos desenvolvedores.

2. O Padrão Model-View-Controller (MVC)O **Model-View-Controller (MVC)** é um padrão arquitetural que separa a aplicação em três componentes interligados. Ele foi originalmente concebido na década de 1970 por Trygve Reenskaug e se tornou proeminente em interfaces gráficas de usuário (GUIs) e, posteriormente, no desenvolvimento *web*.
O objetivo principal do MVC é a **separação de preocupações (*Separation of Concerns*)**, garantindo que as regras de negócio e a lógica de apresentação não estejam fortemente acopladas.

2.1. Componentes do MVC O padrão é composto por três partes principais:
ComponenteFunçãoExemplo em uma Aplicação WebModel (Modelo)Gerencia os **dados**, a **lógica de negócio** e as **regras** da aplicação. Não tem conhecimento sobre a *View* ou o

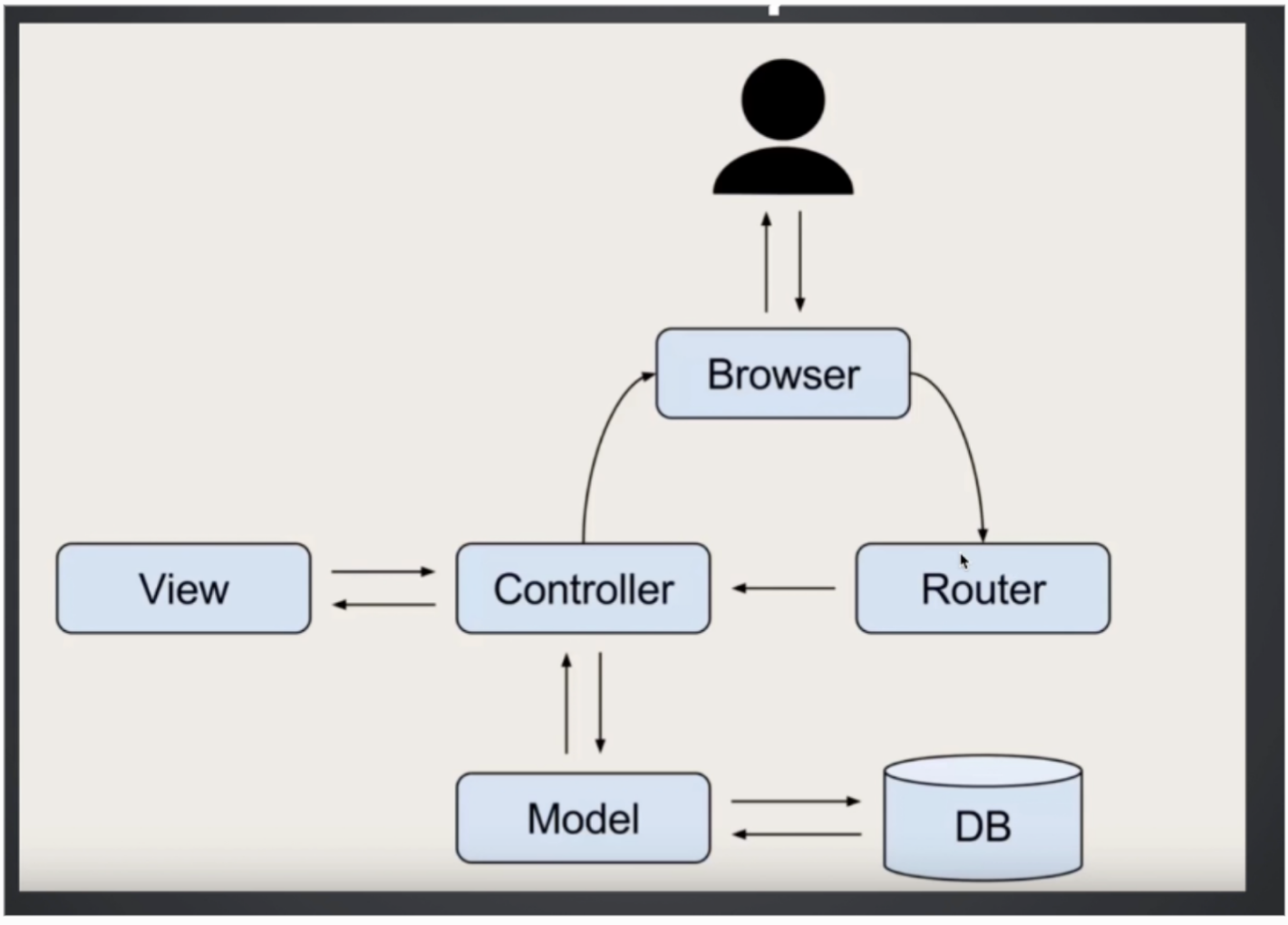
Controller.Classes que interagem com o banco de dados e manipulam objetos de dados (ex: User, Product).**View** (Visão)Responsável pela **interface do usuário**, apresentando os dados do *Model* e capturando a interação do usuário.Arquivos HTML, *templates* ou código de interface que exibe a lista de produtos.**Controller** (Controlador)Atua como **intermediário**. Recebe as entradas do usuário (*View*), aciona a lógica de negócio no *Model* e seleciona a *View* apropriada para exibir o resultado.Funções que recebem uma requisição HTTP, chamam o *Model* para obter dados e renderizam o *template* da *View*.**2.2. Fluxo de Interação**O fluxo de trabalho típico no padrão MVC é o seguinte:

1. O **Usuário** interage com a **View** (ex: clica em um botão).
2. A **View** envia a requisição de entrada para o **Controller**.
3. O **Controller** recebe a requisição, decide qual ação tomar e, se necessário, solicita a manipulação de dados ao **Model**.
4. O **Model** processa a lógica de negócio (ex: salva um novo registro no banco de dados) e notifica o **Controller** (em algumas variantes) ou a **View** (na variante *Observer*).
5. O **Controller** seleciona a **View** apropriada.
6. A **View** busca os dados atualizados no **Model** (ou o **Controller** os passa diretamente) e os apresenta ao usuário.

- **3. Vantagens e Desvantagens do MVC3.1. VantagensSeparação de Responsabilidades:** O principal benefício, tornando o código modular e mais fácil de testar (especialmente a lógica de negócio no *Model*).
- **Desenvolvimento Paralelo:** Permite que desenvolvedores de *frontend* (trabalhando na *View*) e

backend (trabalhando no *Model* e *Controller*) trabalhem simultaneamente com dependência mínima.

- **Suporte a Múltiplas Interfaces:** O mesmo *Model* pode ser reutilizado com diferentes *Views* (ex: uma interface *web* e uma interface móvel).
- **3.2. DesvantagensComplexidade para Aplicações Simples:** A separação pode introduzir uma sobrecarga de código e complexidade desnecessária para projetos pequenos.
- **Problemas com a *View*:** Em algumas implementações, pode haver uma dependência excessiva do *Controller* na *View*, levando a um acoplamento indesejado.
- **O "Controller Gordinho":** O *Controller* pode, inadvertidamente, começar a absorver muita lógica de negócio, transformando-se em um ponto de acoplamento e quebrando a separação de preocupações. É fundamental manter a lógica de negócio estritamente no *Model*.



Um exemplo prático ussando MVC:

model (TaskModel.php)

```
<?php
// TaskModel.php - Gerencia os dados e a lógica de negócio

class TaskModel {
    private $tasks = [
```

```
            'Comprar leite',
            'Estudar padrão MVC',
            'Pagar contas'
        ];

        /**
         * Retorna todas as tarefas.
         * @return array
         */
        public function getAllTasks() {
            return $this->tasks;
        }

        /**
         * Adiciona uma nova tarefa à lista (Lógica de negócio).
         * @param string $taskName
         */
        public function addTask($taskName) {
            // Lógica de validação ou persistência de dados real
(em um app real)
            $this->tasks[] = $taskName;
        }
}
```

O Controller (TaskController.php)

```
<?php
// TaskController.php - Intermediário entre a View e o Model
```

```
require_once 'TaskModel.php';

class TaskController {
    private $model;

    public function __construct() {
        $this->model = new TaskModel();
    }

    /**
     * Lida com a requisição de visualização da lista.
     */
    public function index() {
        // 1. O Controller solicita os dados ao Model.
        $tasks = $this->model->getAllTasks();

        // 2. O Controller carrega a View, passando os dados.
        // O array $tasks estará disponível dentro do arquivo
View.
        require 'TaskView.php';
    }
}
```

A View (TaskView.php)

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
```

```
    <title>Lista de Tarefas (MVC Simples)</title>
</head>
<body>

    <h1>Minhas Tarefas</h1>

    <?php if (!empty($tasks)): ?>
        <ul>
            <?php foreach ($tasks as $task): ?>
                <li><?php echo htmlspecialchars($task); ?></li>
            <?php endforeach; ?>
        </ul>
    <?php else: ?>
        <p>Nenhuma tarefa encontrada.</p>
    <?php endif; ?>

</body>
</html>
```

O Ponto de Entrada (index.php)

```
<?php
// index.php - Ponto de entrada (Roteador/Front Controller)

require_once 'TaskController.php';
```

```
// Instancia o Controller
$controller = new TaskController();

// Chama a ação/método padrão (index)
$controller->index();
```