Apresentação da Disciplina

Dia 9
Arquitetura e Padrões de Projeto

Centro de Alta Performance - SECITECI

SECITECI
SECRETARIA DE ESTADO DE CIÊNCIA E TECNOLOGIA E INOVAÇÃO

GOVERNO DE MATO GROSSO

<Fic_Dev>
Programador de Sistemas

Roteiro

1. Desafio
2. Objetivo da disciplina
3. Pré-requisitos
4. Conteúdo

<Fic_Dev>
Programador de Sistemas

Desafio

Criar um programa que rode no terminal em JavaScript para gerenciamento de tarefas, utilizando a arquitetura MVC e que tenha os seguintes requisitos: tenha um menu onde o usuário pode escolher (criar tarefa, listar tarefas, deletar tarefa e sair do programa), seu programa deve atender todas essas funcionalidades.

Objetivo da disciplina

Entender os padrões de projetos mais utilizados.

Entender e praticar a arquitetura de software MVC no JavaScript.

Pré-requisitos

- Sólidos conhecimentos em Lógica
- Alto poder de abstração
- Muita Força de Vontade, Persistência e Atenção
- Ser autodidata ;-)

<Fic_Dev>
Programador de Sistemas

Conteúdo

- **Padrões de Projeto** (Design Patterns)

Conteúdo

Padrões de projeto e arquitetura são **conceitos distintos**, mas estão relacionados no desenvolvimento de software.

São soluções comuns para problemas específicos de projeto de software.

São soluções de nível de código que descrevem como componentes individuais do sistema devem ser projetados e organizados para atender a requisitos específicos.

Alguns exemplos de padrões de projetos:

<Fic_Dev>
Programador de Sistemas

Conteúdo

- **Padrão de Módulo** (Module Pattern):

Conteúdo

Ele permite que você encapsule o código em um escopo, ocultando as variáveis e funções internas do escopo global. Isso ajuda a prevenir conflitos de nomes e torna o código mais organizado e fácil de ler.

```
1  // Module Pattern
2  class User {
3    // Variável privada
4    #nome;
5
6    // Encapsulamento
7    setarNome(value) {
8      this.#nome = value;
9    }
10
11   pegarNome() {
12     return this.#nome;
13   }
14 }
15
16 const user = new User();
17
18 console.log(user.#nome); // Erro: pois a váriavel é privada
19
20 user.setarNome("Fulano de Tal")
21 console.log(user.pegarNome()); // Fulano de Tal
```

<Fic_Dev>
Programador de Sistemas

Conteúdo

- **Padrão de Fábrica** (Factory Pattern):

Conteúdo

É usado quando você precisa criar objetos com base em um conjunto comum de propriedades e métodos. Ele encapsula a lógica de criação em uma função de fábrica que retorna uma instância do objeto.

```
// Factory Pattern

class Usuario {
  nome = 'Fulano';
}

// Função Factory: Retorna a instância do objeto,
// Com isso se eu precisar mudar a classe Usuario
// não preciso alterar em todos os arquivos a instância,
// apenas nessa função
function  funcaoFactory() {
  return new Usuario();
}

const usuario = funcaoFactory();
console.log(usuario.nome); // Fulano
```

<Fic_Dev>
Programador de Sistemas

Conteúdo

Alguns outros padrões de projetos para estudarem depois:

- **Padrão de Observador** (Observer Pattern):
- **Padrão de Instância Única** (Singleton Pattern):
- **Padrão de Injeção de Dependência** (D.I. Pattern):

Conteúdo

- **Arquitetura**

Conteúdo

A Arquitetura esta mais voltada para a **estrutura geral do sistema**, que inclui a divisão em componentes, suas interações, bem como a estratégia geral de implementação.

A arquitetura de software também pode incluir a **seleção de tecnologias**, **padrões de design** e **decisões de alto nível** que afetam o sistema como um todo.

Ela serve para fornecer uma **estrutura sólida** e **organizada** para o sistema, garantindo que ele atenda aos requisitos funcionais e não funcionais, e que seja **fácil de entender**, **manter** e **evoluir** ao longo do tempo.

<Fic_Dev>
Programador de Sistemas

Conteúdo

- **MVC:** (Model-View-Controller)

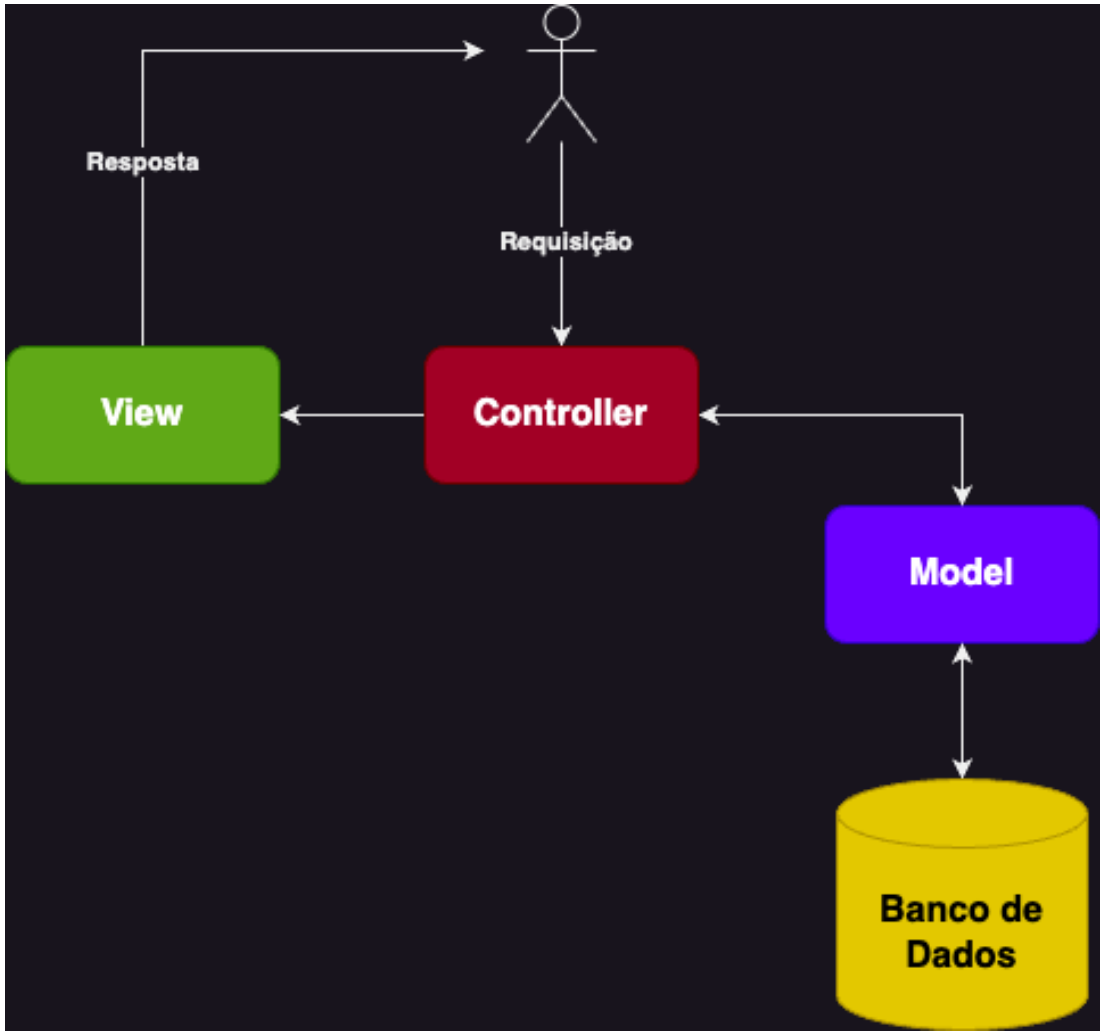
Existem vários tipos de arquiteturas de software e suas subdivisões, mas vamos abordar a mais comum o **MVC**.

É uma **arquitetura em camadas**, onde possuímos 3 camadas: **Model**, **View**, **Controller**.

Conteúdo

- **Model:** lida com a lógica de negócios, o acesso a banco de dados e a validação de dados;
- **View:** trata da interação com o usuário. É responsável por exibir os dados para o usuário em um formato adequado e interativo;
- **Controller:** atua como intermediário entre o **Model** e o **View**.

Conteúdo



Conteúdo

- **Exemplo:**

Conteúdo

Vamos ver um exemplo de um programa onde podemos criar e listar tarefas.

```
// Model
class TarefaModel {
    constructor(id, descricao) {
        this.id = id;
        this.descricao = descricao;
    }
}
```

<Fic_Dev>
Programador de Sistemas

Conteúdo

```
// View
class TarefaView {
    static render(tarefa) {
        console.log(`ID: ${tarefa.id} - Descrição: ${tarefa.descricao}`);
    }
}
```

Conteúdo

```
// Controller
class TarefaController {
    constructor() {
        this.tarefas = [];
        this.idAtual = 1;
    }
    criarTarefa(descricao) {
        const tarefa = new TarefaModel(this.idAtual, descricao);
        this.idAtual++;
        this.tarefas.push(tarefa);
        return tarefa;
    }
    listarTarefas() {
        this.tarefas.forEach(tarefa => TarefaView.render(tarefa));
    }
}
```

Conteúdo

```
// Executando:
const tarefaController = new TarefaController();
// Criando tarefas
const tarefas = ['Estudar', 'Programar', 'Ler', 'Exercitar'];
for (const tarefa of tarefas) {
    tarefaController.criarTarefa(tarefa);
}
// Listar tarefas
tarefaController.listarTarefas();
```

Questionamentos?

Muito Obrigado !

<Fic_Dev>
Programador de Sistemas