Modelo-Visão-Controlador (MVC) - Introdução

Objetivo:

I. Conceitos fundamentais do MVC
II. Modelo (Model)
III. Visão (View)
IV. Controlador (Controller)
V. Aplicação MVC em TypeScript
VI. Exercícios

Observação: antes de começar, crie um projeto para reproduzir os exemplos:

1) Crie a pasta exemplo (ou qualquer outro nome sem caracteres especiais) no local de sua preferência do computador;

2) Abra a pasta no VS Code e acesse o terminal do VS Code;

3) No terminal, execute o comando **npm init -y** para criar o arquivo fundamental de um projeto Node (arquivo package.json);

4) No terminal, execute o comando **npm i -D ts-node typescript** para instalar os pacotes ts-node e typescript como dependências de desenvolvimento;

5) No terminal, execute o comando **tsc --init** para criar o arquivo de opções e configurações para o compilador TS (arquivo tsconfig.json);

6) Crie a **pasta src** na raiz do projeto;

7) Crie o arquivo **index.ts** na **pasta src**;

8) Adicione na propriedade scripts, do package.json, o comando para executar o arquivo index.ts. Ao final o arquivo package.json terá o seguinte conteúdo:

```
{
  "name": "TPII_Aula02",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "index": "ts-node ./src/index",
    "ex01": "ts-node ./src/ex01",
    "ex02": "ts-node ./src/ex02",
    "ex03": "ts-node ./src/ex03",
    "ex04": "ts-node ./src/ex04",
    "ex05": "ts-node ./src/ex05",
    "teste": "ts-node ./src/teste",
    "teste1": "ts-node ./src/teste1",
    "teste2": "ts-node ./src/teste2"
  },
  "keywords": [],
  "author": "Prof.Henrique Louro",
  "license": "ISC",
```

```
  "devDependencies": {
    "ts-node": "^10.9.2",
    "typescript": "^5.4.2"
  }
}
```

Observação: se o comando `ts-node` não funcionar, então use o programa `npx` para executar o comando `tsnode`:

```
"scripts": {
"start": "npx ts-node ./src/index"
        },
```

I. Conceitos fundamentais do MVC

Model-View-Controller, comumente conhecido como MVC, é um padrão de design popular amplamente usado no desenvolvimento web. Este padrão divide o desenvolvimento de uma aplicação em três partes interligadas, visando separar as representações internas da informação das formas como a informação é apresentada ou aceita pelo usuário.

II. Modelo

É o que representa os dados e as regras que regem o acesso e as atualizações desses dados. Ele responde a solicitações de informações, processa instruções para alterar seu estado e se comunica com o banco de dados.

III. Visualizar

A Visualização é a interface do usuário — o que está sendo apresentado aos usuários e como os usuários interagem com o aplicativo. Representa a visualização dos dados que o modelo contém.

IV. Controlador

E ele que recebe informações do usuário e decide o que fazer com elas. É um sistema que gerencia o fluxo de dados no objeto do modelo e atualiza a visualização sempre que os dados mudam.

Essa divisão permite organização eficiente do código, alto nível de modularidade e flexibilidade.

V. Aplicando MVC em TypeScript

Vamos implementar uma aplicação simples utilizando o padrão MVC em TypeScript: um gerenciador de tarefas.

a. **Modelo (Model):** Nosso modelo será uma classe simples que representa uma tarefa com um ID, uma descrição e um booleano para verificar se a tarefa foi concluída.

```
class TarefaModel {
    id: number;
    descricao: string;
    concluida: boolean;
    constructor(id: number, descricao: string, concluida: boolean
= false) {
        this.id = id;
        this.descricao = descricao;
        this.concluida = concluida;
```

```
    }
}
```

b. **Visualizar (View):** A visualização será responsável pela interface do usuário, neste caso, o console registrando as tarefas.

```
class TarefaView {
    mostrar(tarefas: TarefaModel[]) {
        for (const tarefa of tarefas) {
            console.log(`${tarefa.id}: ${tarefa.descricao} -
${tarefa.concluida ? 'Concluída' : 'Pendente'}`);
        }
    }
}
```

c. **Controlador (Controller):** O controlador gerenciará a entrada do usuário, atualizando o modelo e a visualização conforme necessário.

```
class TarefaController {
    private modelo: TarefaModel[];
    private visao: TarefaView;

    constructor(modelo: TarefaModel[], visao: TarefaView) {
        this.modelo = modelo;
        this.visao = visao;
    }

    addTarefa(descricao: string) {
        const novaTarefa = new TarefaModel(this.modelo.length + 1,
descricao);
        this.modelo.push(novaTarefa);
        this.visao.mostrar(this.modelo);
    }

    completaTask(id: number) {
        const tarefa = this.modelo.find(tarefa => tarefa.id ===
id);
        if (tarefa) {
            tarefa.concluida = true;
        }
        this.visao.mostrar(this.modelo);
    }
}

const tarefas:TarefaModel[] = [];
const tarefav = new TarefaView();
const tarefac = new TarefaController(tarefas,tarefav);
tarefac.addTarefa("Fazer algo");
console.log("----------------------");
tarefac.addTarefa("Fazer outra coisa");
console.log("----------------------");
tarefac.addTarefa("Fazer mais alguma coisa");
console.log("----------------------");
tarefac.completaTask(1);
```

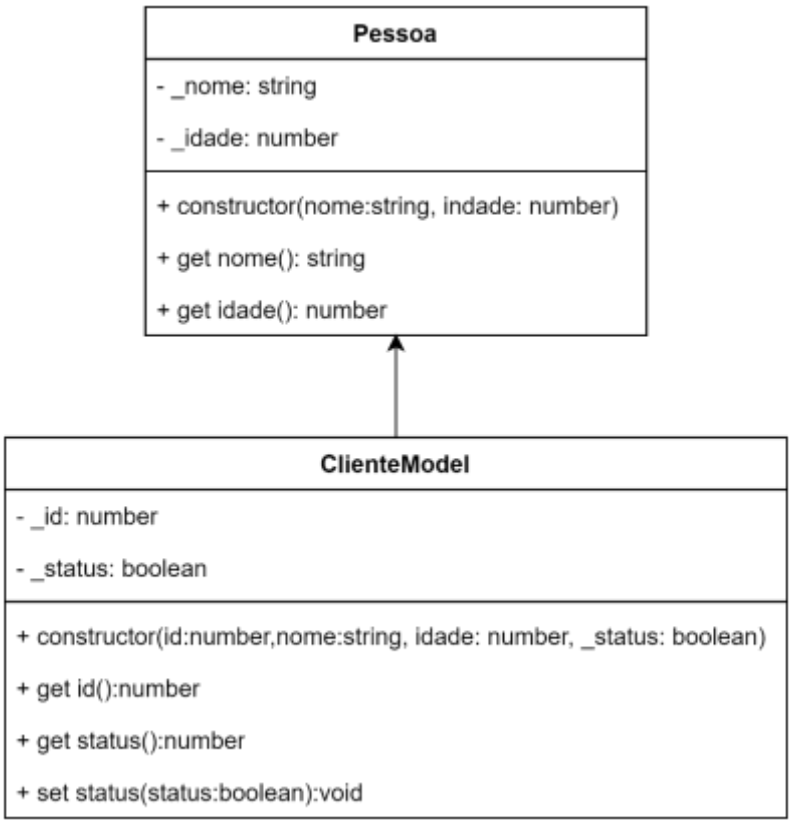
No exemplo acima, a classe **TarefaController** manipula as instâncias de **TarefaModel** adicionando uma nova tarefa ou marcando uma tarefa como concluída com base na entrada do usuário. Em seguida, ela se comunica com a instância da TarefaView para atualizar a exibição.

Conclusão

MVC é um padrão de design robusto que nos ajuda a separar funções em nossa aplicação. Ele fornece uma maneira eficiente de organizar o código, tornando-o mais fácil de manter, dimensionar e compreender. TypeScript, com seus tipos estáticos e recursos de POO, pode ajudar significativamente na estruturação de um design MVC em aplicações web. Apesar da sua simplicidade, o exemplo acima ilustra os princípios básicos de como implementar MVC, que podemos estender para cenários mais complexos, de acordo com as necessidades dos requisitos de um sistema.

VI. Exercícios

1) Dados os diagramas UML das Classes Pessoa e ClienteModel, implemente o Modelo, Visão e Controlador, conforme exemplo dado:



Tarefas de Implementação

- O Controlador deve ter métodos para adicionar Clientes e setar seus status para “Ativo (true)” e “Desativado (false)”;
- Para cada interação com os métodos do Controlador a View deve ser acionada e listar todos os clientes cadastrados e seus status;
- Cada Cliente cadastrado deve possuir um ID inteiro único e sequencial iniciado em 1 e status de Ativo;
- Instanciar os objetos para lista de clientes e view a serem passados a um objeto instanciado do Controlador;
- Cadastrar pelo menos 3 clientes.

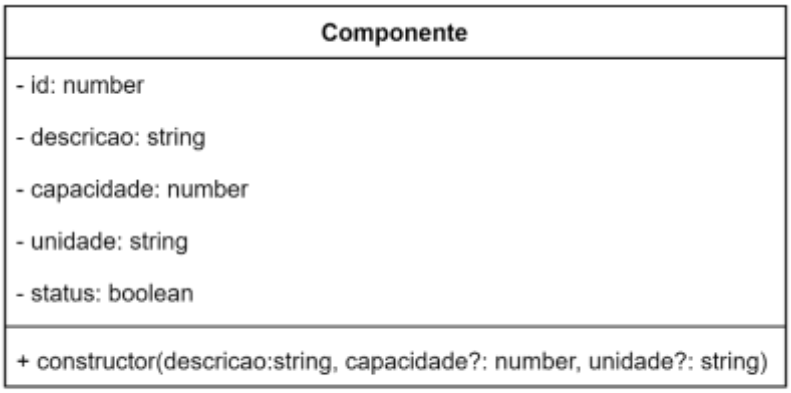
- Passar qualquer um dos Clientes cadastrados para Desativado.

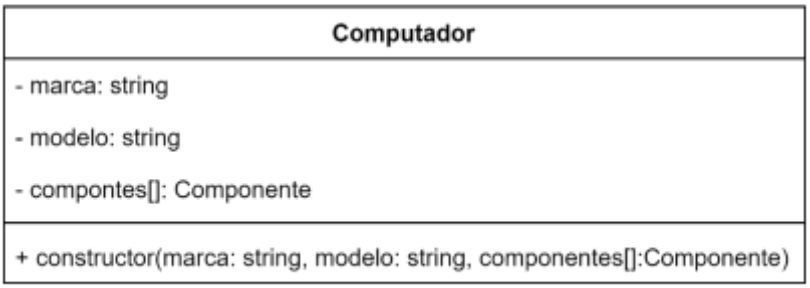
2) **Agenda de Contatos**: Crie uma classe "AgendaModel" que armazena informações de contatos (id,nome, telefone, e-mail):

Tarefas de Implementação:

- Crie uma classe "Contato" para representar cada contato e poder inseri-lo na Agenda.
- O Controlador deve ter métodos para adicionar Contatos e setar seus status para "Ativo (true)" e "Desativado (false)";
- Para cada interação com os métodos do Controlador a View deve ser acionada e listar todos os clientes cadastrados e seus status;
- Cada Contato cadastrado deve possuir um ID inteiro único e sequencial iniciado em 1;
- Instanciar os objetos para lista de contatos e view a serem passados a um objeto instanciado do Controlador;
- Cadastrar pelo menos 3 contatos.

3) **Computador e Componente**: Crie uma classe "Computador" que instancie internamente um array da classe "Componentes", conforme diagras UMLs a seguir:





Tarefas de Implementação:

- O Controlador deve ter métodos para adicionar Componentes na instância da Classe Computador e setar seus status de funcionamento para "Funcionando (true)" e "Com defeito (false)";
- Para cada interação com os métodos do Controlador a View deve ser acionada e listar os dados do computador e todos os componentes cadastrados e seus status;

- Cada Componente cadastrado deve possuir um ID inteiro único e sequencial iniciado em 1;
- Instanciar os objetos para lista de componentes e view a serem passados a um objeto instanciado do Controlador;
- Cadastrar pelo menos 3 componentes em um computador;
- Atentar que nem todo componente possuirá capacidade;
- Só precisará da unidade componentes que possuam capacidade. Por exemplo: Memória RAM (descrição), 4 (capacidade), Gb (unidade).