UNIVERSIDADE DO VALE DO ITAJAI
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PADRÃO MVC EM JAVA

ITAJAÍ
2024

RESUMO

Esse relatório técnico tem como principal objetivo, mostrar o padrão de projeto MVC (Model-View-Controller) na linguagem de programação Java. O padrão MVC é normalmente utilizado para organizar estrutura de aplicações de software, oferecendo a separação das responsabilidades, facilitando a manutenção, atualização e reuso em projetos ou sistemas futuros. Portando esse relatório irá mostrar os conceitos fundamenteis do padrão MVC, mostrando suas vantagens e desvantagens, e exemplos, o primeiro sendo mostrado como funciona em uma aplicação web e o segundo sendo mostrado em uma aplicação desenvolvida em Java.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O padrão de projeto MVC (Model-View-Controller) é um dos mais conhecidos, utilizado na arquitetura de software no desenvolvimento de aplicações, principalmente em projetos webs e mobile, podendo também ser usadas em mais diversas outras aplicações. Sua principal característica é a separação da aplicação em três componentes principais: Model (Modelo), View (Visão) e Controller (Controlador). Está separação facilita a organização do código para o desenvolvedor e tornando-o fácil de manter o código estável e além de reduzir potenciais problemas. Neste relatório, vamos mostrar os fundamentos do padrão MVC, suas vantagens, desvantagens e uma aplicação prática no final do relatório em Java.

2 O QUE É MVC

O padrão MVC é uma abordagem de design de software que separa a aplicação em três componentes distintos:

- Model: Responsável pela lógica de negócios e gerenciamento dos dados da aplicação.
- View: Responsável pela apresentação dos dados ao usuário.
- Controller: Responsável por receber as entradas do usuário, processar as solicitações e determinar a resposta adequada.

Esta divisão permite uma separação clara das responsabilidades, facilitando a manutenção e evolução da aplicação.

Para exemplificar com mais clareza como seria como uma divisão ficaria em um projeto, vamos usar um exemplo que extraída do site TREINAWEB, que utiliza uma página web para exemplificar.

Primeiramente o projeto foi separado em três partes, facilitando o desenvolvimento da aplicação, as quais foram separadas em:

- Model (cliente.php): Contém as informações do cliente (nome, endereço, cidade etc.) e os métodos para manipular esses dados no banco de dados (inserir, alterar, excluir e listar clientes).
- View (página HTML): Mostra a lista de usuários cadastrados e o formulário para cadastro de novos usuários.
- Controller: Atua como intermediário entre o Model e a View, evitando um acoplamento direto entre eles. Ele faz a ligação entre os Models e Views.

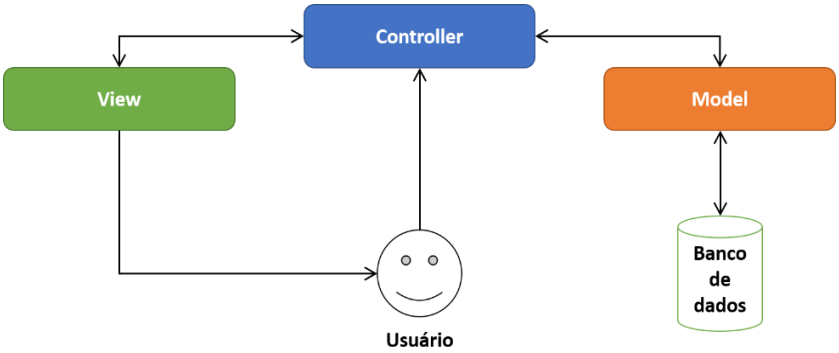

Figura 1 - Exemplo dos papéis de cada camada na implementação mencionada.

3 COMPONENTES DO PADRÃO MVC

3.1 MODEL (MODELO)

O Model representa a parte da aplicação que lida com a lógica de negócios e dados. Ele é responsável por recuperar, armazenar e manipular os dados, bem como garantir a integridade e consistência das regras de negócio. Em Java, os modelos são geralmente implementados como classes que contêm os atributos e métodos relacionados aos dados da aplicação.

3.2 VIEW (VISÃO)

A View é responsável pela apresentação dos dados ao usuário. Ela recupera os dados do Model e os exibe de forma adequada, seja em uma interface gráfica ou em uma interface web. Em Java, as Views podem ser implementadas usando frameworks como JavaFX, JSP (JavaServer Pages), ou bibliotecas de frontend como HTML e CSS em aplicações web.

3.3 CONTROLLER (CONTROLADOR)

O Controller atua como um intermediário entre o Model e a View. Ele recebe as entradas do usuário, processa estas solicitações e interage com o Model para recuperar ou atualizar os dados. Em seguida, determina a View adequada para apresentar a resposta ao usuário. Em Java, os Controllers são geralmente implementados como servlets ou classes de ação em frameworks como Spring MVC.

4 VANTAGENS E DESVANTAGENS

4.1 VANTAGENS

- Separação de responsabilidades: Facilita a manutenção e evolução da aplicação.
- Reutilização de código: Componentes podem ser reutilizados em diferentes partes da aplicação ou em diferentes projetos.
- Modularidade: Permite que diferentes desenvolvedores trabalhem em componentes distintos de forma independente.

4.2 DESVANTAGENS

- Complexidade inicial: Pode ser mais difícil de implementar e entender para desenvolvedores iniciantes.
- Sobrecarga: Em aplicações simples, o uso do padrão MVC pode introduzir uma complexidade desnecessária.

5 IMPLEMENTAÇÃO EM JAVA

Para a implementação em Java, primeiro vamos criar um problema, que o problema em questão é o seguinte: um proprietário está com problemas da criação de uma academia para treinos, crossfit e dança, ele não tem uma ideia de como criar um sistema para tal academia, portando ele pensou em contatar alunos da Univali, especificamente 2 alunos que já fizeram um sistema parecido para um outro proprietário de academia, os alunos tendo em vista que seria um outro sistema de academia, aceitou pois iria facilitar a vida deles tendo já feito algo parecido, por fim os alunos decidiram criarem um sistema usando o padrão de projeto MVC, que aprenderam faz pouco tempo e iriam projetar no serviço proposto pelo proprietário, tal implementação foi separada em 3 principais classes, a classe Aluno, que ficou com a parte do MODEL (MODELO), a classe Aluno View que ficou com a parte VIEW (VISÃO) e a classe AlunoController que ficou com CONTROLLER (CONTROLADOR), portando a implementação ficou nesse forma:

Model

```java
package Academia;

public class Aluno {

    // MODEL

    private String nome;
    private String matricula;
    private String email;
    private int idade;
    private Instrutor instrutor;

    public Aluno(String nome, String matricula, String email, int idade) {
        this.nome = nome;
        this.matricula = matricula;
        this.email = email;
        this.idade = idade;
    }
```

Figura 2

View

```java
package Academia;

public class AlunoView {

    // VIEW

    public void mostrarDetalhesAluno(String nome, String matricula, String email, int idade, Instrutor instrutor) {
        System.out.println(x:"Detalhes do Aluno:");
        System.out.println("Nome: " + nome);
        System.out.println("Matrícula: " + matricula);
        System.out.println("Email: " + email);
        System.out.println("Idade: " + idade);
        if (instrutor != null) {
            System.out.println("Instrutor: " + instrutor.getNome());
            System.out.println("Idade do Instrutor: " + instrutor.getIdade());
            System.out.println("Anos de Experiência do Instrutor: " + instrutor.getAnosExperiencia());
            if (instrutor instanceof InstrutorCrossFit) {
                System.out.println("Certificação do Instrutor de CrossFit: " + ((InstrutorCrossFit) instrutor).getCertificacao());
            } else if (instrutor instanceof InstrutorDanca) {
                System.out.println("Estilo de Dança do Instrutor de Dança: " + ((InstrutorDanca) instrutor).getEstiloDanca());
            }
        } else {
            System.out.println(x:"Instrutor: Nenhum");
        }
        System.out.println(x:"----------------------------------------");
    }
}
```

Figura 3

Controller

```java
package Academia;

public class AlunoController {

    // CONTROLLER

    private Aluno model;
    private AlunoView view;

    public AlunoController(Aluno model, AlunoView view) {
        this.model = model;
        this.view = view;
    }

    public void setNomeAluno(String nome) {
        model.setNome(nome);
    }

    public void setMatriculaAluno(String matricula) {
        model.setMatricula(matricula);
    }

    public void setEmailAluno(String email) {
        model.setEmail(email);
    }

    public void setIdadeAluno(int idade) {
        model.setIdade(idade);
    }

    public void setInstrutorAluno(Instrutor instrutor) {
        model.setInstrutor(instrutor);
    }

    public void atualizarView() {
        view.mostrarDetalhesAluno(model.getNome(), model.getMatricula(), model.getEmail(), model.getIdade(), model.getInstrutor());
    }
}
```

Figura 4

6 CONCLUSÃO

O padrão de projeto MVC representa uma forma robusta para estruturar aplicações de software de forma organizada. Ao separar claramente as responsabilidades entre modelagem de dados, lógica de negócios e apresentação, ele simplifica a manutenção, escalabilidade e evolução do código. Embora possa parecer complexo inicialmente, o MVC oferece benefícios substanciais, especialmente em projetos de grande porte. Em ambientes Java, sua implementação é amplamente suportada por vários frameworks, promovendo um desenvolvimento mais eficiente e estruturado.

REFERÊNCIAS

DEVMEDIA. Padrão MVC. Disponível em: https://www.devmedia.com.br/padrao-mvc-java-magazine/21995. Acesso em: 07 jun. 2024.

LEWAGON. O que é padrão MVC? Disponível em: https://blog.lewagon.com/pt-br/skills/o-que-e-padrao-mvc/. Acesso em: 07 jun. 2024.

DEVMEDIA. Java SE: Aprendendo o padrão MVC. Disponível em: https://www.devmedia.com.br/java-se-aprendendo-o-padrao-mvc/29546. Acesso em: 07 jun. 2024.

TREINAWEB. O que é MVC? Disponível em: https://www.treinaweb.com.br/blog/o-que-e-mvc. Acesso em: 07 jun. 2024.