SENAI

Arquitetura MVC

Software Desenvolvido em Camadas

MVC

Model, View e Controller

Arquitetura MVC

História:

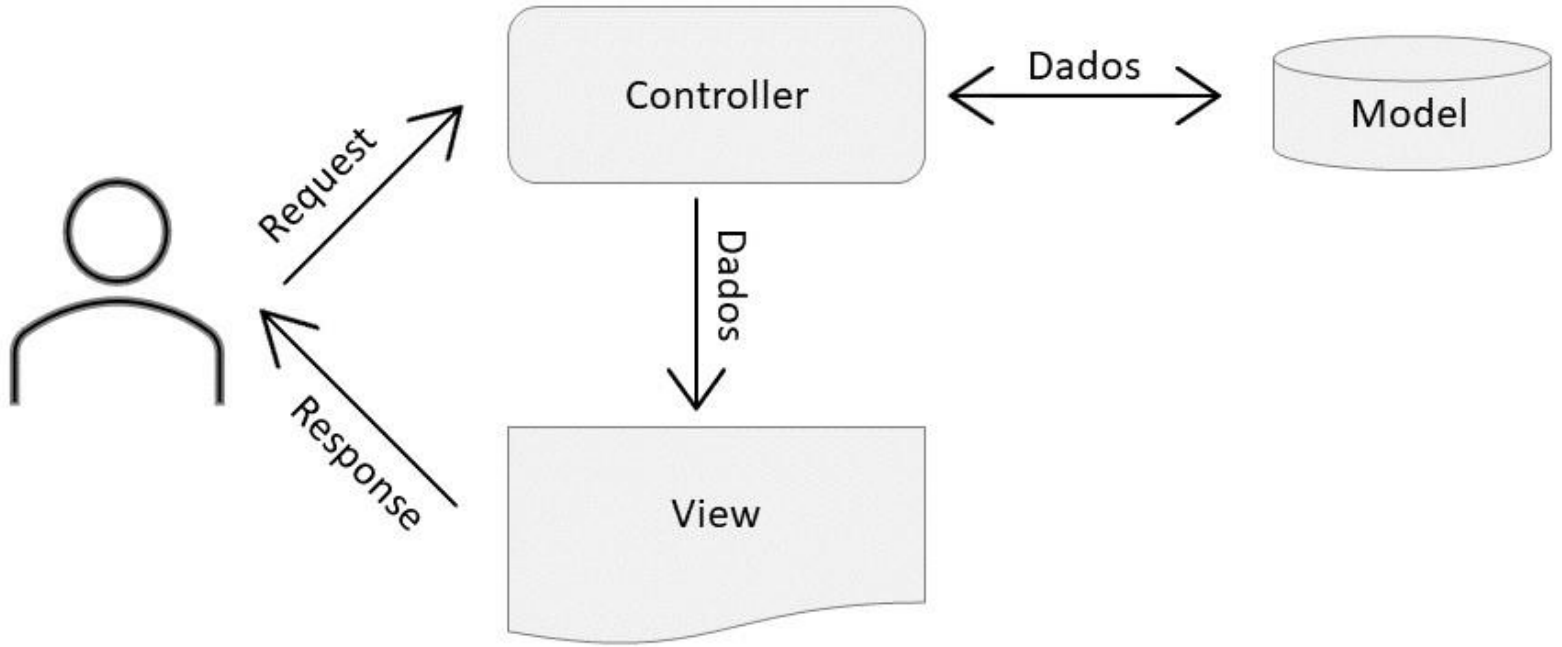
- ✓ Criada por **Trygve Reenskaug** em 1979 na Xerox Parc (GUI-interface gráfica, mouse, Ethernet etc.), uma ex-divisão de pesquisa da Xerox;
- ✓ Artigo (Steve Burbeck) que descreve a primeira versão da arquitetura e sua implementação: http://www.dgp.toronto.edu/~dwigdor/teaching/csc2524/2012_F/papers/mvc.pdf
- ✓ Inicialmente a arquitetura foi implementada na linguagem **Smalltalk** (a primeira linguagem orientada à objetos, enquanto outros dizem ser a SIMULA-68).

Arquitetura MVC

Conceito e definição:

- ✓ Padrão de Arquitetura de Software organizado em **Camadas**;
- ✓ A aplicação é dividida em **três camadas** principais:
  - Modelo (Model);
  - Visão (View);
  - Controle (Controller).
- ✓ Um dos principais objetivos é **separar a interface das regras e do modelo de negócios**;
- ✓ Cada camada assume **responsabilidades** no sistema.

Funcionamento



Model

Modelo

Model

Responsabilidades:

- ✓ Também conhecida como **Business Object Model** (Objeto Modelo de Negócio);
- ✓ Representa o **modelo** do negócio;
- ✓ Responsável pela **implementação** (tipos, tamanhos, comportamentos etc.), acesso e **manipulação da fonte de dados**;
- ✓ Recebe (às vezes com auxílio de algum padrão) os dados, faz o tratamento e valida.

View

Visão

View

Responsabilidades:

- ✓ Responsável pela interface apresentada ao usuário;
- ✓ Recebe instruções do **controller**, os dados do **model** e **apresenta** ao usuário;
- ✓ Nos frameworks são construídas em **HTML** com outras tecnologias de acordo com a linguagem, exemplos:
  - ASP.NET Core MVC: Razor, Blazor;
  - Java Spring Boot: Thymeleaf, JSP etc.;
  - PHP Laravel: Blade.

Controller

Controle

Controller

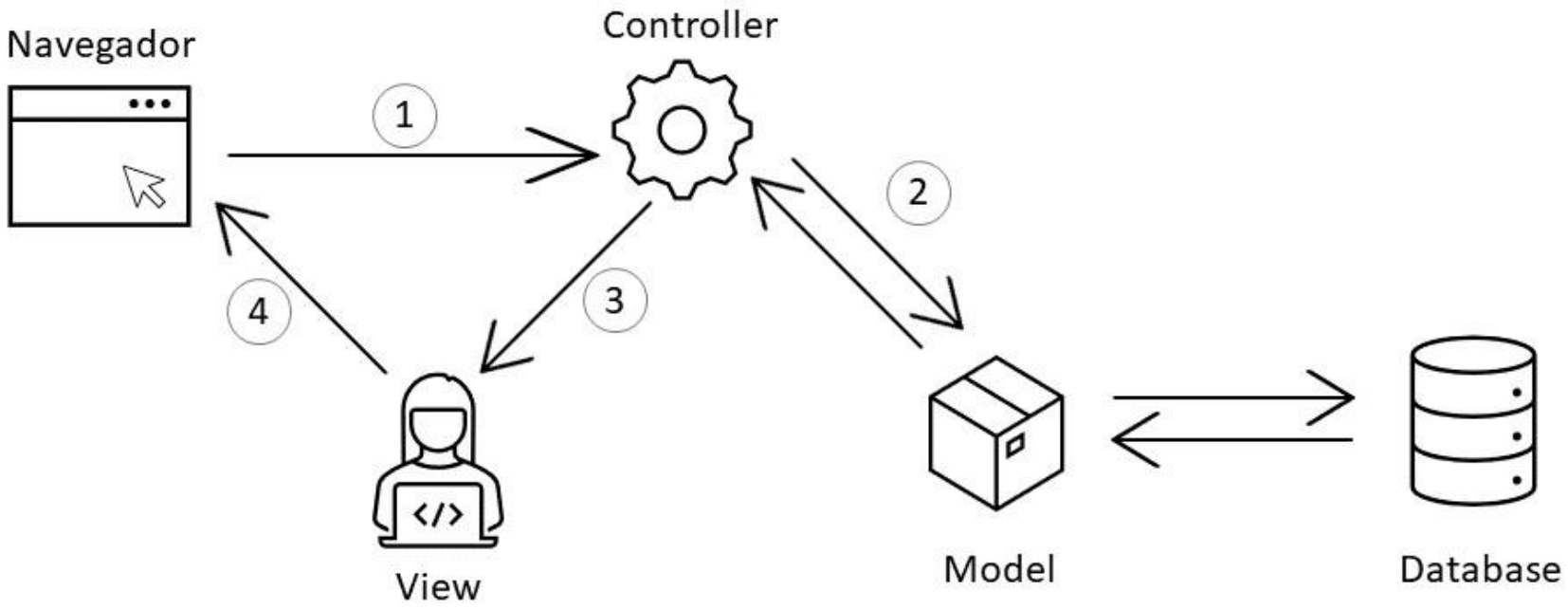
Responsabilidades:

- ✓ O Controller trabalha de acordo com as ações do usuários;
- ✓ É responsável pela lógica de negócios do sistema;
- ✓ Responsável também por intermediar as requisições e as respostas retornadas pelo Model;
- ✓ Processa os dados e direciona para outras camadas;
- ✓ Seu papel soa como uma espécie de “maestro” do sistema.

Rotas

Fluxo dos dados

Fluxo dos dados



Rotas

Fluxo dos dados e métodos:

- ✓ O fluxo das requisições em um sistema ocorre por meio de **rotas**;
- ✓ Podemos chamar as rotas de **urls amigáveis** que mapeiam as requisições;
- ✓ Essas requisições podem ser **endereçadas a um método/função** em um **controller** ou podem ser um direcionamento para outra página;
- ✓ Na maioria das linguagens as rotas são estabelecidas no **Controller** da aplicação. Mas em alguns casos, como por exemplo no Framework PHP Laravel, ela está em um arquivo específico.

Rotas

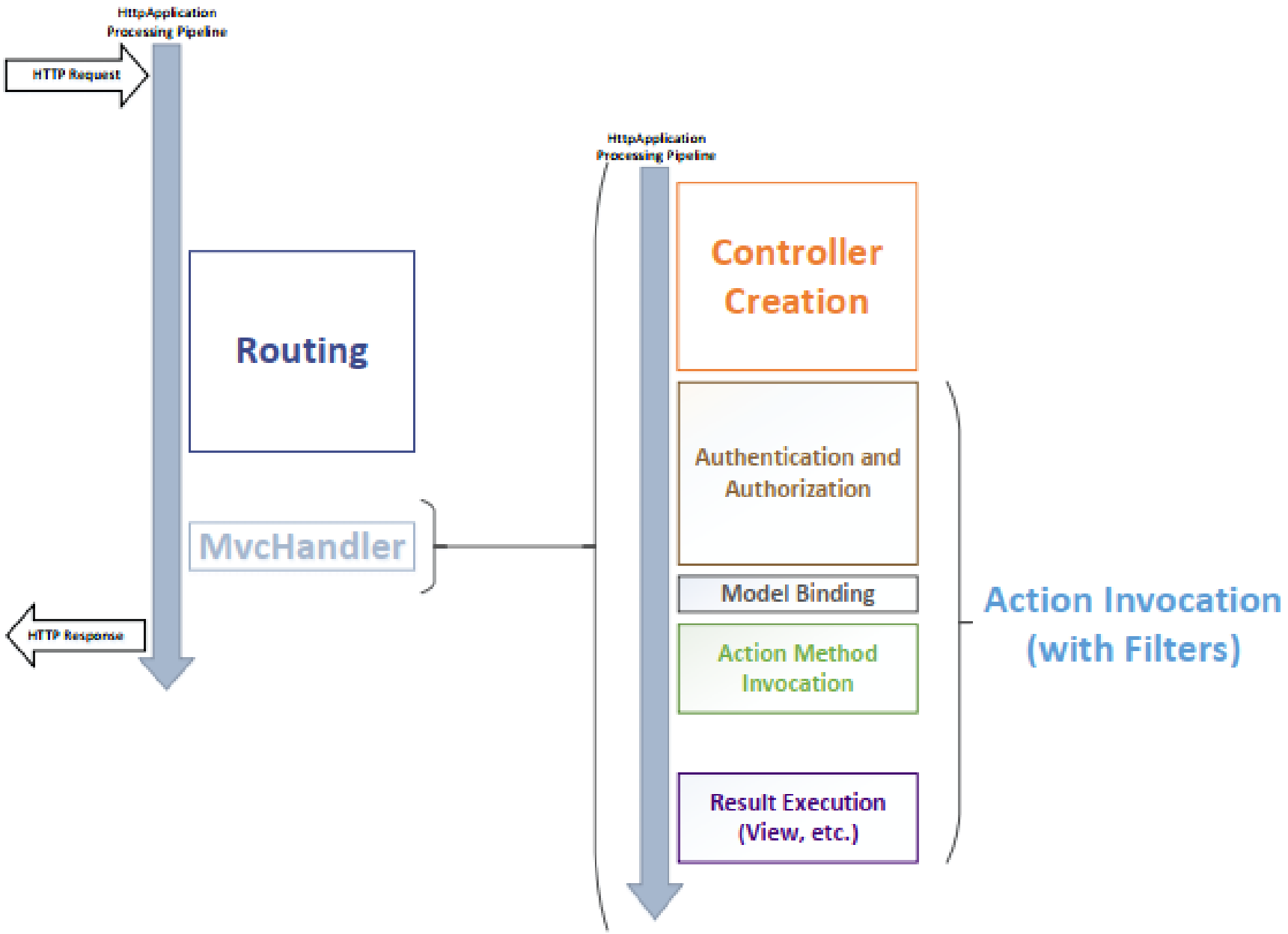
Fluxo dos dados e métodos:

✓ As rotas podem ser **simples** ou **com parâmetros**;

✓ As requisições são **métodos (verbos) HTTP**, os dois tipos mais utilizados são:

- **GET** → O método GET solicita a representação de um recurso específico. Requisições com o método GET devem **retornar** apenas dados.
- **POST** → O método POST é utilizado para **submeter** uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.

ASP.NET MVC


HttpApplication
Processing Pipeline
HTTP Request
Routing
MvcHandler
HTTP Response
HttpApplication
Processing Pipeline
Controller
Creation
Authentication and
Authorization
Model Binding
Action Method
Invocation
Result Execution
(View, etc.)
Action Invocation
(with Filters)

Principais prós e contras

Prós:

- ✓ Padronização da **comunicação** entre as camadas;
- ✓ **Coesão** (harmonia) entre as camadas;
- ✓ Desacoplamento dos elementos das camadas o que facilita a adição e reuso desses elementos;
- ✓ Pode ser implementado com **design patterns** (padrões de projeto) como o Repository, Observer etc.
- ✓ **Apresentação dos dados** de formas diferentes, pois pode-se trabalhar com **views** distintas;
- ✓ É mais seguro, pois os dados precisam passar pelo **controller (filtro)** antes de acessar a base de dados;

Principais prós e contras

- ✓ Uso de **APIs** sem precisar ficar modificando código;
- ✓ **Organização e eficiência** que permitem que o sistema seja “entendido” e modificado por diferentes profissionais (desenvolvedor front e back-end).

Contras:

- ✓ **Complexidade** na modelagem e implementação;
- ✓ **Dependência** de ferramentas frameworks que constroem as estruturas dos projetos;
- ✓ **Linguagens distintas** no desenvolvimento das **views**, o que pode atrasar um pouco as implementações com programadores que não conhecem a tecnologia;
- ✓ Possíveis desvios de finalidade, exemplo: lógica de negócio sendo implementada na view.

Referências

https://learn.microsoft.com/pt-br/aspnet/overview

https://learn.microsoft.com/pt-br/aspnet/mvc/overview/getting-started/lifecycle-of-an-aspnet-mvc-5-application

Livro: Gamma, Erich; Richard Helm, Ralph Johnson, John M. Vlissides - **Padrões de Projeto - Soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Editora Bookman, 2005.