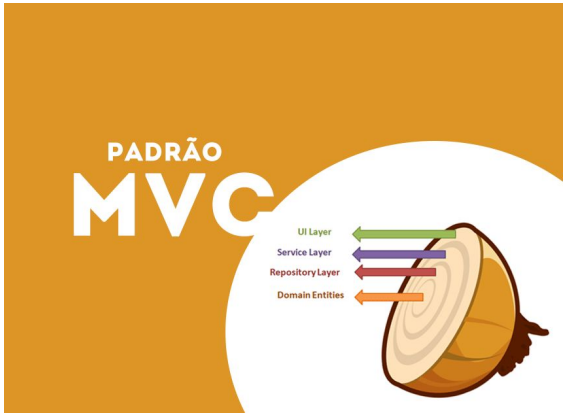
Padrões arquiteturais(MVC)

José Glauber UFCG 2024.1

Problemática

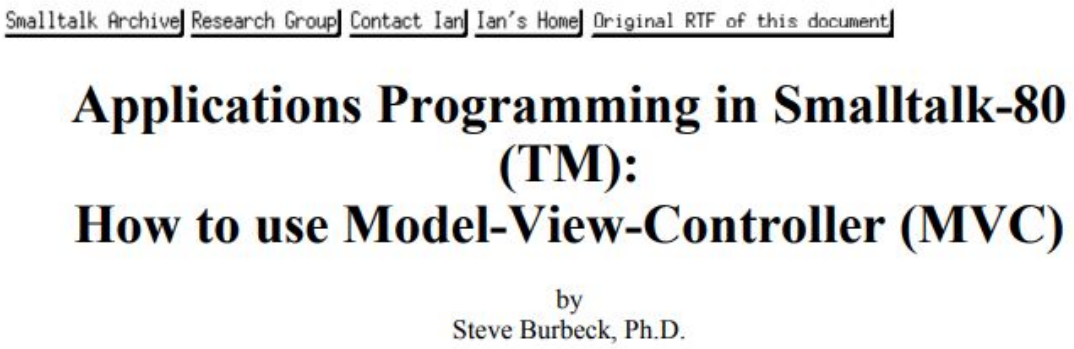
- Arquitetura de Software é um processo delicado;
  - como os componentes são organizados e quais tecnologias serão utilizadas;
  - experiência prática e colaboração em equipe;

- Padrões arquiteturais já foram pensados para solucionar problemas corriqueiros;

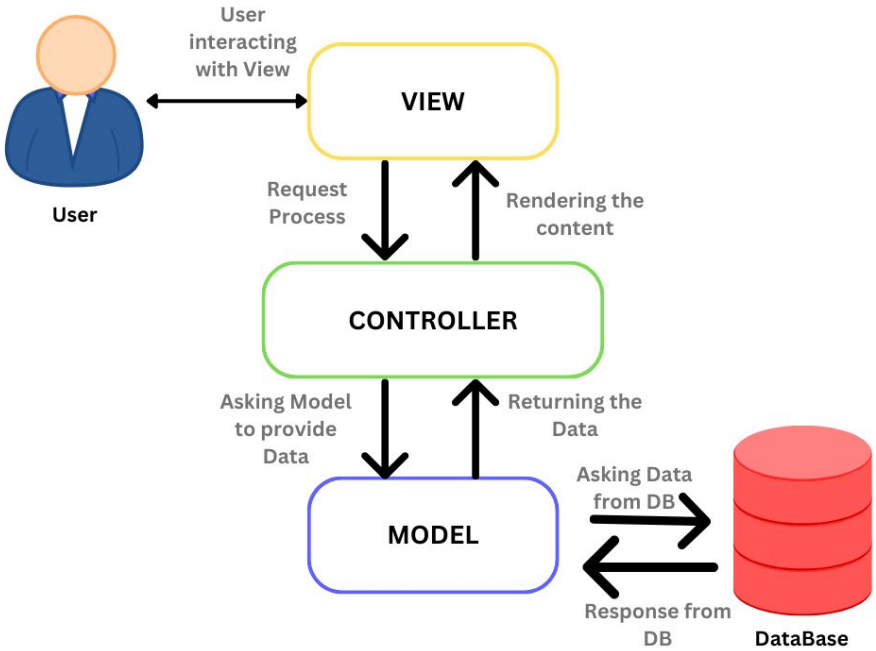


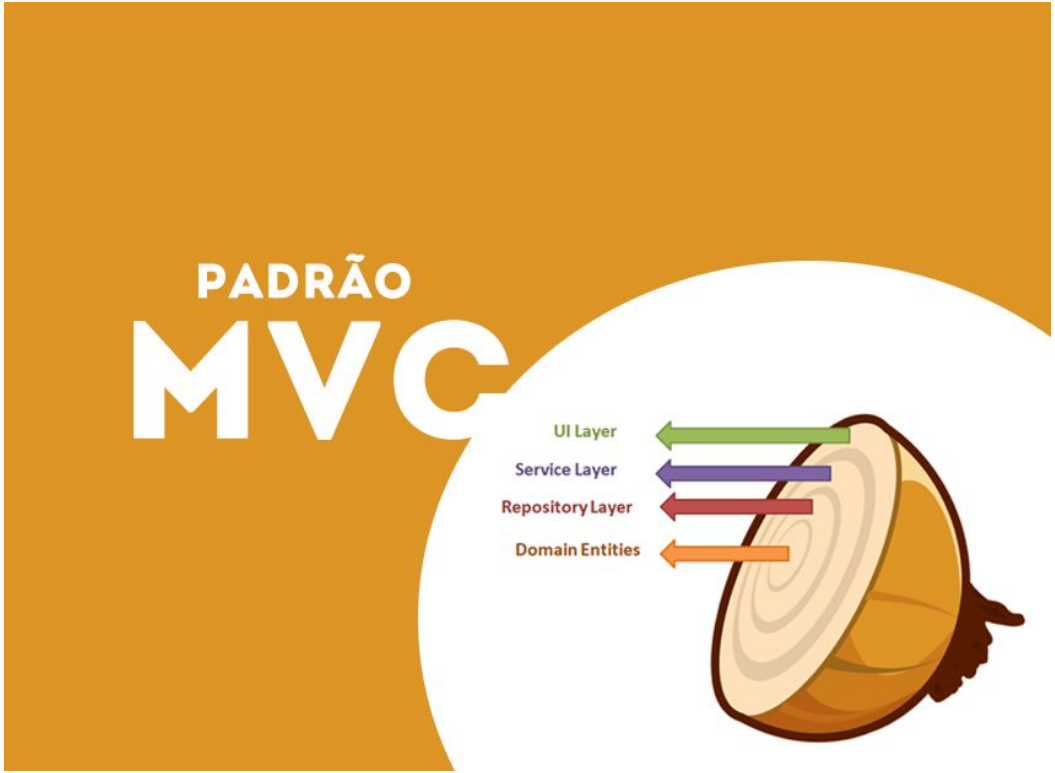
Model-View-Controller. Como surgiu?

- Formulado na década de 1970 o principal objetivo desse padrão é separar a apresentação dos dados e interação dos usuários dos métodos (**frontend**) que interagem com o banco de dados (**backend**).



Model-View-Controller





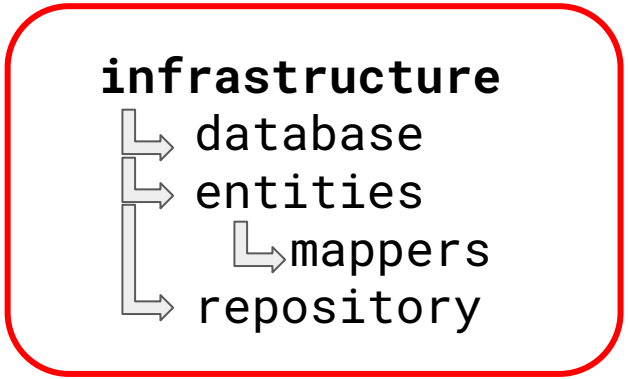
- Infraestrutura
- Aplicação
- UI

Camada de infraestrutura

Essa é a camada de acesso aos dados. Fornece os mecanismos necessários para ler, gravar, atualizar e deletar os dados do sistema.

Responsabilidades gerais:

- Consultas SQL;
- Mapeamento objeto relacional;
- Operações de armazenamento e recuperação de dados;



Infra - Database

Guarda documentos relacionados a conexão com banco de dados

- Migrations
- Configurações de conexão
- Scripts popular banco de dados

Infra - Entidades

- As entidades são objetos de domínio que possuem uma identidade única ao longo do ciclo de vida da aplicação. Elas representam objetos reais ou conceituais no sistema.
- Mapeado diretamente para o banco de dados e são responsáveis por encapsular dados e comportamentos associados a esse objeto.

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/database');

class Order extends Model {}

Order.init({
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },
  customerId: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
  totalAmount: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  status: {
    type: DataTypes.STRING(50),
    allowNull: false,
  },
}, {
  sequelize,
  modelName: 'Order',
  tableName: 'orders',
});

module.exports = Order;
```

Infra - Entidades (responsabilidades)

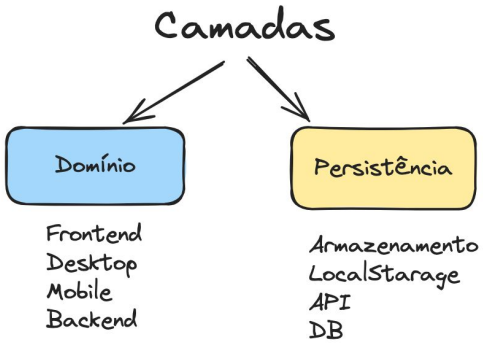
- Identidade
- Persistência
- Autonomia

Objetos que têm uma identidade distinta, independentemente de suas características ou valores. A identidade é mais importante que os dados que compõem a entidade.

Evans (2003)

Infra - Mappers

Um Data Mapper é um **padrão de design** que separa a lógica de negócios da lógica de persistência, mapeando objetos em uma aplicação para a estrutura de um banco de dados relacional sem que a aplicação precise saber como isso é feito. Fowler (2002)



Infra - Repositories

Classes que encapsulam a lógica necessária para acessarmos uma fonte de dados, desacoplando a camada de acesso aos dados da camada de domínio.

- Permitir a troca do banco de dados sem afetar o sistema como um todo;
- Diminui o acoplamento entre as classes;
- Impulsiona o uso da injeção de dependência;
- Facilita a criação de testes unitários;

Camada de Aplicação

Coordena a interação entre a camada de apresentação (UI) e a de persistência (infraestrutura).

Responsabilidades gerais:

- Gerenciar lógica de negócio;
- Promover comunicação entre diferentes partes do sistema;

application
- service
- domain
  - models
  - validator

Domínio - Modelos

São representações dos dados que serão enviados ou recebidos pela API. Eles são usados para definir a estrutura das requisições e respostas e geralmente mapeiam para objetos de transferência de dados (DTOs)

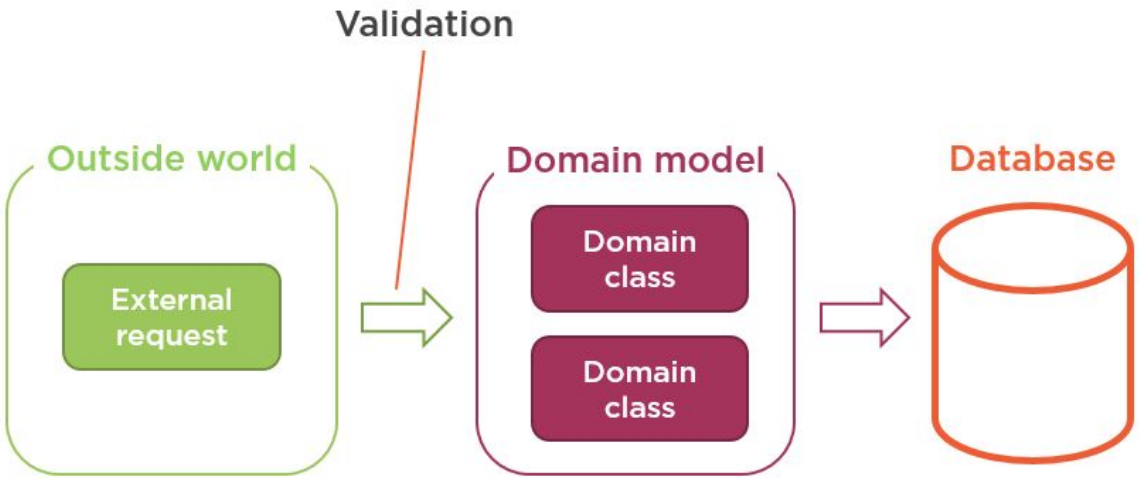
home > joseglauber > Documentos > UFCG > ProgWeb > aulas > example2.js > ...

```
1  const Joi = require('joi');
2
3  // Definindo o esquema de validação para um Pedido
4  const OrderModel = Joi.object({
5    id: Joi.number().integer().min(1),
6    customerId: Joi.number().integer().min(1).required(),
7    totalAmount: Joi.number().precision(2).required(),
8    status: Joi.string().max(50).required(),
9  });
10
11 module.exports = OrderModel;
```

- **Transferência de dados**
- **Desacoplamento**
- **Validação**

Domínio - Validações

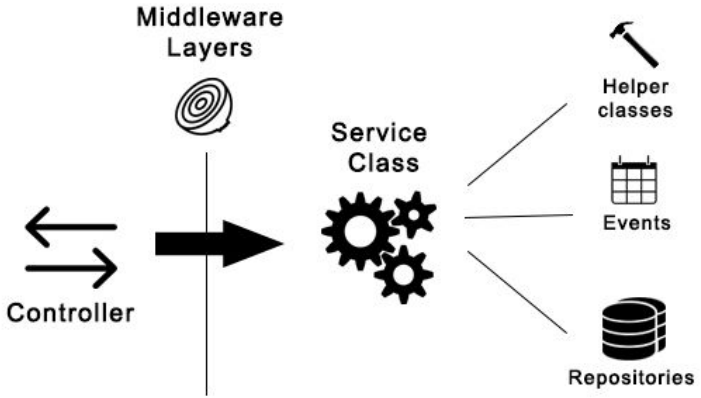
Regras de negócios e restrições que garantem que os dados estejam corretos e consistentes.



- **Campos obrigatórios**
- **Atendeu a regra de negócio**
- **Campos corretos..**

Serviço

Responsável pela lógica de negócios da aplicação. Recebe as entradas vindas da camada de apresentação, processa os dados e retorna o resultado.



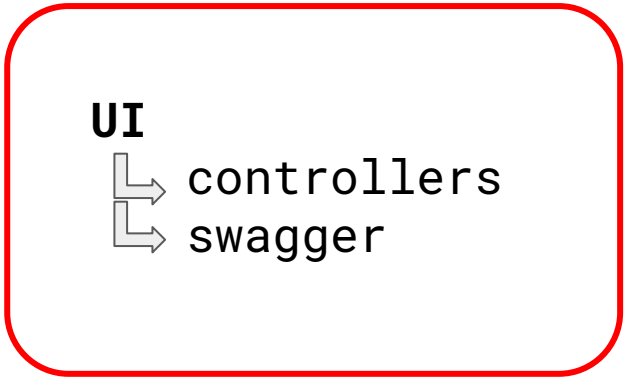
- Responsável pela lógica de implementação
- Fluxo de validações
- Acessar bancos de dados

Camada UI

Envolve principalmente a gestão de entradas e saídas de dados através de APIs e outros pontos de interação

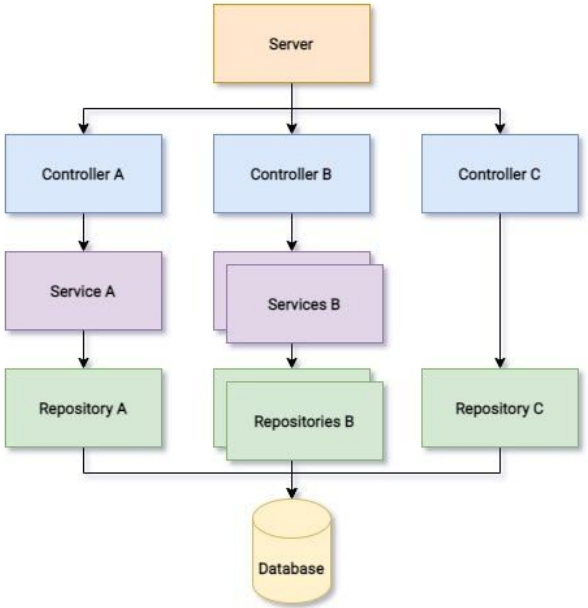
Responsabilidades gerais:

- Comunicação direta com o frontend
- Recebimento dos dados vindos do cliente



Camada UI - Controllers

Receber e processar as requisições HTTP dos clientes. Se conecta diretamente com os serviços.



Cada controller geralmente é associado a uma rota específica da API e pode lidar com diferentes tipos de operações HTTP.

Ainda podem..

- Validar entradas dos usuários;
- Manipular erros;
- Formatar respostas antes do envio para o cliente;

Camada UI - Swagger

- Fornece uma interface gráfica interativa que permite visualizar e testar os endpoints da API.
- Ajuda desenvolvedores e consumidores da API a entender como interagir com os serviços disponíveis, quais parâmetros são necessários e quais respostas podem ser esperadas.

Referências

- Design Patterns: Elements of Reusable Object-Oriented Software - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Applications Programming in Smalltalk-80: How to use Model-View-Controller - S Burbeck - **Smalltalk-80** v2, 1992
- Patterns of Enterprise Application Architecture - Martin Fowler
- Architectural Patterns: A Guide to Modern Software Architecture
- https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2
- https://www.linkedin.com/pulse/data-mapper-pattern-frontend-itallo-s%C3%A1-vieira-ricxf/
- https://renicius-pagotto.medium.com/entendendo-o-repository-pattern-fcdd0c36b63b