1. O que é uma JPA

**JPA é uma tecnologia que simplifica o trabalho com bancos de dados em aplicações Java. Ela permite que você use classes Java para representar dados armazenados em um banco de dados. Em vez de escrever códigos complexos para lidar com o banco de dados, você pode definir classes que representam tabelas e usar métodos simples para salvar, buscar ou atualizar dados.**

**Com JPA, suas classes Java se tornam entidades que correspondem às tabelas do banco de dados. Isso torna as operações básicas de banco de dados, como inserir, atualizar, deletar e buscar dados, mais fáceis de realizar, reduzindo a complexidade do código e facilitando o trabalho do desenvolvedor.**

1. O que é um Controller ou classe controller

**Um controller é parte central de uma aplicação, atuando entre o usuário e o sistema. Ele recebe solicitações dos usuários, processa-as, interage com outros componentes (como bancos de dados), e retorna respostas apropriadas.**

**Os controllers:**

* **Recebem solicitações do cliente com informações sobre ações desejadas.**
* **Processam as solicitações e interagem com serviços para obter ou salvar dados.**
* **Geram respostas ao usuário, como páginas web ou dados JSON.**
* **Lidam com roteamento de URLs, direcionando cada URL para o método correspondente.**

**Em resumo, os controllers gerenciam a comunicação entre a interface do usuário e o backend, recebendo solicitações, processando-as e retornando respostas.**

1. O que é o que faz o Repository/DAO

* **Acesso ao banco de dados: Um repository ou DAO é responsável por estabelecer a conexão com o banco de dados e realizar operações de CRUD**
* **Abstração do banco de dados: Essa camada oferece uma interface para a aplicação interagir com o banco de dados sem precisar se preocupar com detalhes específicos da implementação**
* **Operações com dados: O repository fornece métodos prontos para realizar operações comuns com os dados. Por exemplo, um repository para uma entidade "Cliente" pode ter métodos como findAll para buscar todos os clientes, findById para buscar um cliente específico por ID**
* **Isolamento da lógica de negócio: O uso de repositories ajuda a manter a lógica de acesso aos dados separada da lógica de negócio da aplicação.**

**Em resumo, um repository ou DAO é uma camada que oferece uma interface para trabalhar com o banco de dados, executando operações de acesso a dados e isolando a lógica de negócio da lógica de persistência.**

1. O que é e o que faz o Service

**Um Service é uma camada que contém a lógica de negócio principal da aplicação. Atua como ponte entre as classes controller (que lidam com as solicitações do usuário) e os repositories (que interagem com o banco de dados).**

**Funções do Service:**

* **Lógica de negócio: Aqui estão as regras específicas da aplicação e as decisões sobre o processamento das solicitações.**
* **Interação com repositories: O Service faz chamadas a repositories para realizar operações de banco de dados.**
* **Encapsulamento da lógica: Mantém os controllers focados em solicitações do usuário, enquanto a lógica complexa é delegada para o Service.**
* **Coordenação de operações: O Service combina dados de diferentes fontes ou executa cálculos baseados nesses dados.**
* **Manutenção e escalabilidade: Facilita a organização e expansão da aplicação.**
* **Testabilidade: Permite testes de unidade isolados para métodos de service.**

**Em resumo, o Service coordena a lógica de negócio, auxiliando na organização e escalabilidade da aplicação.**

**Entidade= Tabela no banco**

**Json= “Formato” no qual o webservice apresenta os dados: {"nome": "João", "idade": 30 }**

**@**

**SpringBootApplication**

**Essa anotação inicia a aplicação Spring Boot. Ela habilita várias configurações automáticas, como a varredura de componentes.**

**@RestController**

**Indica que a classe é um controlador REST, ou seja, lida com solicitações HTTP e responde com JSON ou XML.**

**@RequestMapping**

**Mapeia URLs para métodos em um controlador. Pode ser usado em classes ou métodos para definir o caminho de uma requisição.**

**@GetMapping e @PostMapping**

**Mapeiam URLs para métodos específicos que lidam com requisições HTTP GET ou POST, respectivamente.**

**@PathVariable**

**Extrai um parâmetro de um método a partir da URL da requisição.**

**@RequestBody**

**Extrai um parâmetro de um método a partir do corpo da requisição, normalmente em formato JSON.**

**@Entity**

**Marca uma classe como uma entidade JPA, associando-a a uma tabela no banco de dados.**

**@Table**

**Especifica o nome da tabela no banco de dados para a entidade.**

**@Id**

**Marca um campo como a chave primária de uma tabela.**

**@GeneratedValue**

**Indica que o valor da chave primária é gerado automaticamente.**

**@Column**

**Especifica propriedades de uma coluna no banco de dados, como nome, tamanho e se é obrigatório.**

**@ManyToOne**

**Indica uma relação muitos-para-um entre entidades, como um agendamento relacionado a um médico.**

**@JoinColumn**

**Define qual coluna é usada para a relação entre entidades.**

**@Autowired:Injeta dependências automaticamente em campos ou construtores.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Projeto Application**

**Essa classe é responsável por iniciar a aplicação Spring Boot.**

**Ela tem o método main que é o ponto de entrada da aplicação Java.**

**Dentro desse método, SpringApplication.run é chamado para iniciar a aplicação Spring Boot.**

package br.com.projetoint.projeto;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication(scanBasePackages = "br.com.projetoint")

public class ProjetoApplication {

public static void main(String[] args) {

SpringApplication.run(ProjetoApplication.class, args);

}

}

**Medico**

**Esta é uma classe que representa o modelo de um médico na sua aplicação.**

**@Entity indica que esta classe é uma entidade JPA (Java Persistence API), ou seja, ela mapeia uma tabela no banco de dados.**

**A classe tem três propriedades:**

**idMedico: um número único para identificar cada médico (a chave primária).**

**nomeMedico: o nome do médico.**

**Possui métodos (get e set) para acessar e modificar cada propriedade.**

package br.com.projetoint.projeto.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "Medico")

public class Medico {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int idMedico;

@Column(name = "nomeMedico", nullable = false, length = 100)

private String nomeMedico;

//getters e setters...

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MedicoController**

**Essa classe é um controlador para manipular a entidade médico.**

**@RestController: diz ao Spring Boot que essa classe é um controlador REST.**

**@RequestMapping("/medico"): significa que todas as URLs que começam com /medico são mapeadas para os métodos dessa classe.**

**@GetMapping e @PostMapping: são métodos que respondem a requisições HTTP GET e POST, respectivamente.**

**O método obterMedicoPorId retorna um médico específico com base no ID fornecido.**

**listarMedicos retorna uma lista de todos os médicos.**

**criarMedico recebe um médico via requisição HTTP e o salva no banco de dados.**

package br.com.projetoint.projeto.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import br.com.projetoint.projeto.DAO.MedicoRepository;

import br.com.projetoint.projeto.model.Medico;

@RestController

@RequestMapping("/medico")

public class MedicoController {

@Autowired

private MedicoRepository medicoRepository;

@GetMapping("/{id}")

public Medico obterMedicoPorId(@PathVariable int id) {

return medicoRepository.findById(id).orElse(null);

}

@GetMapping

public List<Medico> listarMedicos() {

return medicoRepository.findAll();

}

@PostMapping

public Medico criarMedico(@RequestBody Medico medico) {

return medicoRepository.save(medico);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MedicoRepository**

**Essa é uma interface para manipular dados de médicos no banco de dados.**

**JpaRepository: é uma interface do Spring Data que oferece métodos para trabalhar com a base de dados, como findAll(), findById(), save(), etc.**

package br.com.projetoint.projeto.DAO;

import org.springframework.data.jpa.repository.JpaRepository;

import br.com.projetoint.projeto.model.Medico;

public interface MedicoRepository extends JpaRepository<Medico, Integer> {

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Agendamento**

**Esta classe representa um agendamento na sua aplicação.**

**@Entity: indica que essa classe é uma entidade JPA, mapeando uma tabela no banco de dados.**

**A classe tem várias propriedades, como idAgendamento (chave primária), nomePaciente, email, status, clinica, medico (relacionamento com a entidade Medico), dataHoraAgendamento, dataCadastro.**

**Possui métodos (get e set) para acessar e modificar cada propriedade.**

package br.com.projetoint.projeto.model;

import java.time.LocalDateTime;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.ManyToOne;

import jakarta.persistence.Table;

@Entity

@Table(name = "Agendamento")

public class Agendamento {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int idAgendamento;

@Column(name = "nomePaciente", nullable = false, length = 100)

private String nomePaciente;

@Column(name = "email", nullable = false, length = 100)

private String email;

@Column(name = "status", nullable = false, length = 20)

private String status;

@Column(name = "clinica", nullable = false, length = 50)

private String clinica;

@ManyToOne

@JoinColumn(name = "idMedico", nullable = false)

private Medico medico;

@Column(name = "dataHoraAgendamento", nullable = false)

private LocalDateTime dataHoraAgendamento;

@Column(name = "dataCadastro", nullable = false)

private LocalDateTime dataCadastro;

//getters e setters...

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AgendamentoController**

**Este controlador é para manipular a entidade agendamento.**

**@RestController: significa que a classe é um controlador REST.**

**@RequestMapping("/agendamento"): significa que todas as URLs que começam com /agendamento são mapeadas para os métodos desta classe.**

**@GetMapping e @PostMapping: são métodos que respondem a requisições HTTP GET e POST.**

**criarAgendamento recebe um agendamento via requisição HTTP e o salva no banco de dados.**

**obterAgendamento retorna um agendamento específico com base no ID fornecido.**

**listarAgendamentos retorna uma lista de todos os agendamentos.**

package br.com.projetoint.projeto.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import br.com.projetoint.Service.AgendamentoService;

import br.com.projetoint.projeto.model.Agendamento;

@RestController

@RequestMapping("/agendamento")

public class AgendamentoController {

@Autowired

private AgendamentoService agendamentoService;

@PostMapping

public Agendamento criarAgendamento(@RequestBody Agendamento agendamento) {

return agendamentoService.criarAgendamento(agendamento);

}

@GetMapping("/{id}")

public Agendamento obterAgendamento(@PathVariable int id) {

return agendamentoService.obterAgendamentoPorId(id);

}

@GetMapping

public List<Agendamento> listarAgendamentos() {

return agendamentoService.listarAgendamentos();

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AgendamentoRepository**

**Assim como MedicoRepository, essa é uma interface para manipular dados de agendamentos no banco de dados.**

**JpaRepository fornece métodos para trabalhar com a base de dados.**

package br.com.projetoint.projeto.DAO;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import br.com.projetoint.projeto.model.Agendamento;

@Repository

public interface AgendamentoRepository extends JpaRepository<Agendamento, Integer> {

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AgendamentoService**

**Este é o serviço para manipular agendamentos.**

**Ele usa AgendamentoRepository para realizar operações com agendamentos no banco de dados.**

**Oferece métodos como criarAgendamento, obterAgendamentoPorId e listarAgendamentos.**

package br.com.projetoint.Service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import br.com.projetoint.projeto.DAO.AgendamentoRepository;

import br.com.projetoint.projeto.model.Agendamento;

@Service

public class AgendamentoService {

@Autowired

private AgendamentoRepository agendamentoRepository;

public Agendamento criarAgendamento(Agendamento agendamento) {

return agendamentoRepository.save(agendamento);

}

public Agendamento obterAgendamentoPorId(int id) {

return agendamentoRepository.findById(id).orElse(null);

}

public List<Agendamento> listarAgendamentos() {

return agendamentoRepository.findAll();

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**application.properties**

**Esse arquivo contém configurações para a aplicação.**

**Ele configura o banco de dados, especificando o nome de usuário, senha, URL do banco de dados, entre outros detalhes.**

**Também define o server.port que determina a porta na qual a aplicação estará rodando.**

spring.datasource.username=root

spring.datasource.password=123456

spring.datasource.url=jdbc:mysql://localhost:3306/BD\_USUARIOS?useTimezone=true&serverTimezone=UTC

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

spring.profiles.active=development

server.port=8081