



Bootcamp: Arquiteto(a) de Software

Desafio Final

Objetivos de Ensino

Exercitar os seguintes conceitos trabalhados nos Módulos:

1. Fundamentos de Arquitetura de Software.
2. Requisitos Arquiteturais e Modelagem Arquitetural.
3. Design Patterns, Estilos e Padrões Arquiteturais.
4. Principais Arquiteturas de Software da Atualidade.

Enunciado

Você é Arquiteto(a) de Software em uma grande empresa de vendas on-line. Você é responsável por construir e implantar uma solução que disponibilize publicamente dados de Cliente/Produto/Pedido (algum domínio) aos parceiros da empresa.

Para isso, você vai Projetar, Documentar e Implantar uma API REST, no padrão arquitetural MVC, que exponha um endpoint capaz de realizar um CRUD dos dados (e um pouco mais).

Bom trabalho!





Atividades

Criação de uma API REST com Arquitetura MVC

Objetivo:

O objetivo deste exercício é aplicar os conhecimentos de arquitetura de software, focando na implementação de uma API RESTful, seguindo o padrão MVC. A ideia é explorar práticas de design e construção de APIs, documentação de arquitetura e organização de código.

Requisitos do exercício:

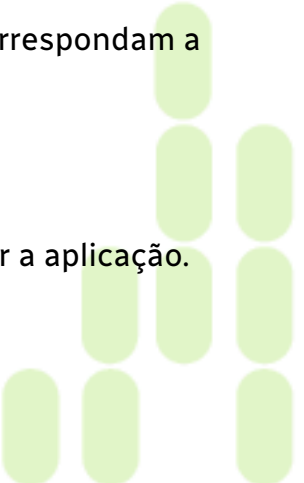
1. Escolha de Plataforma e Linguagem:

- o Utilize uma plataforma e linguagem de sua preferência (ex.: Java com Spring, Python com Flask, Node.js com Express etc.).

2. Funcionalidades da API:

- o Desenvolva uma API RESTful que implemente operações CRUD básicas sobre um domínio de sua escolha (ex.: Clientes, Produtos, Pedidos).
- o Implemente os seguintes métodos:
 - **CRUD:** Criação (Create), Leitura (Read), Atualização (Update) e Exclusão (Delete).
 - **Contagem:** Endpoint para retornar o número total de registros.
 - **Find All:** Endpoint para retornar todos os registros.
 - **Find By ID:** Endpoint para retornar um registro específico com base no ID.
 - **Find By Name:** Endpoint para retornar registros que correspondam a um nome específico.

3. Arquitetura:

- o Utilize o padrão MVC (Model-View-Controller) para estruturar a aplicação.
- 

- o Embora o uso de um banco de dados não seja obrigatório, incluir a persistência dos dados será considerado um diferencial.

4. Entrega do Exercício:

- o **Desenho arquitetural do software:**
 - Entregue o desenho da arquitetura utilizando UML e/ou C4 e/ou qualquer diagrama de sua preferência, utilizando ferramentas como o draw.io.
- o **Estrutura de pastas e explicação dos componentes:**
 - Apresente a estrutura de pastas do projeto, descrevendo brevemente o papel de cada componente, como o Controller, Model (Entidades) e Service.
- o **(Opcional) Entrega do Código:**
 - Se optar por entregar o código, faça o upload em um repositório GitHub ou GitLab e compartilhe o link.

Exemplo Passo a Passo usando Spring (Java)

1. Configuração inicial do projeto:

- o Crie um novo projeto Spring Boot, podendo utilizar o [Spring Initializr](#) para isso.
- o Inclua as dependências necessárias:
 - **Spring Web** para criação da API.
 - **Spring Data JPA e H2 Database** (ou outra) caso queira implementar a persistência.

2. Estrutura de pastas:

- o Organize o projeto seguindo a estrutura abaixo (sugestão):

```
└─ src/java
  └─ com/exemplo/api
    ├── controller      # Controladores que gerenciam as requisições
    ├── model           # Entidades de domínio (ex: Cliente)
    ├── repository      # Repositórios para interação com o banco de
    │                   dados
    ├── service         # Serviços que implementam a lógica de
    │                   negócios
    └─ ApiApplication   # Classe principal para iniciar a aplicação
```

3. Model – Entidade de Domínio (ex.: Cliente):

- o Crie uma classe Cliente na pasta model que representará a entidade de domínio:

```
@Entity
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String email;
    // Getters e Setters
}
```

4. Repository – Interface de Repositório:

- o Crie um repositório na pasta repository que estenda JpaRepository:

```
@Repository
public interface ClienteRepository extends JpaRepository<Cliente, Long> {
    List<Cliente> findByName(String nome);
}
```

5. Service – Lógica de negócios:

- o Na pasta service, crie uma classe *DominioService* que contenha a lógica de negócios:

```
@Service
public class ClienteService {
    @Autowired
    private ClienteRepository clienteRepository;

    public List<Cliente> listarTodos() { return
        clienteRepository.findAll(); }

    public Optional<Cliente> buscarPorId(Long id) { return
        clienteRepository.findById(id); }

    public List<Cliente> buscarPorNome(String nome) { return
        clienteRepository.findByName(nome); }

    public Cliente salvar(Cliente cliente) { return
        clienteRepository.save(cliente); }

    public void deletar(Long id) { clienteRepository.deleteById(id); }

    public long contarClientes() { return clienteRepository.count(); }
}
```

6. Controller – Controlador REST:

- o Na pasta controller, crie uma classe *DominioController* para mapear os endpoints:

```
@RestController
@RequestMapping("/clientes")
public class ClienteController {
    @Autowired
    private ClienteService clienteService;

    @GetMapping
    public List<Cliente> listarTodos() { return
        clienteService.listarTodos(); }

    @GetMapping("/{id}")
    public ResponseEntity<Cliente> buscarPorId(@PathVariable Long id) {
        return clienteService.buscarPorId(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }
}
```

```
}

@GetMapping("/nome/{nome}")
public List<Cliente> buscarPorNome(@PathVariable String nome) {
    return clienteService.buscarPorNome(nome); }

@GetMapping("/contar")
public long contarClientes() { return
    clienteService.contarClientes(); }

@PostMapping
public Cliente salvar(@RequestBody Cliente cliente) { return
    clienteService.salvar(cliente); }

@DeleteMapping("/{id}")
public void deletar(@PathVariable Long id) {
    clienteService.deletar(id); }
}
```

Observações para os alunos:

- **Organização do Código:** cada componente (Controller, Service, Model) deve ser bem documentado e isolado em suas respectivas responsabilidades.
- **Explicação da estrutura de pastas:** inclua uma breve explicação de cada pasta e o papel de cada componente no padrão MVC.
- **Desenho arquitetural:** entregue o diagrama UML e/ou C4 Model e/ou outro, mostrando a relação entre os componentes da aplicação.

Resumo dos Entregáveis:

1. Arquitetura do software (C4 Model/UML/Outro Diagrama no Draw.io).
2. Estrutura de pastas do projeto MVC.
3. Explicação da estrutura e dos elementos que comporão o código.
4. **Opcional** (código funcionando).
5. **Opcional** (persistência funcionando).

Divirta-se!