
FIAP

SOA - Java

Testes Unitários em Java — REST API (Projeto: restapi)

- Entender o papel dos testes unitários em uma API Spring Boot;
- Escrever testes do serviço (regras de negócio) isolando o repositório;
- Validar endpoints do controller com **WebMvcTest** + **MockMvc**;
- Executar e interpretar os resultados **(mvn test);**

Testes Unitários em Java — REST API (Projeto: restapi) - O que são Testes Unitários?

- Testam unidades pequenas do código (métodos/classes) de forma isolada
- Rápidos, determinísticos e independentes de infraestrutura externa
- Na API: foco na camada de serviço (regras) e controlador (contratos)

Testes Unitários em Java — REST API (Projeto: restapi) – Reflexão.....

- **MAS PORQUE EU DEVERIA FAZER ISSO?**
- **QUAIS SÃO OS BENEFÍCIOS;**
- **PORQUE TESTAR?**

Testes Unitários em Java — REST API (Projeto: restapi) - POM — dependências de testes (trecho)

- Controller: AlunoController — lista em memória + endpoints GET/POST/PUT/DELETE
- Service: AlunoService — regra de inclusão com validações (trim, obrigatórios, tamanho)
- Repository: AlunoRepository — JDBC (Oracle) via ConnectionFactory
- Model: Aluno (Lombok @Data/@AllArgsConstructor)

Testes Unitários em Java — REST API (Projeto: restapi) - AlunoController

- Valida o contrato HTTP do AlunoController sem subir toda a aplicação
- Cenários:
  - GET /api/alunos/1 (buscar) → retorna aluno existente
  - POST /api/alunos (incluir) → cria e retorna aluno usando Service mockado

Testes Unitários em Java — REST API (Projeto: restapi) - AlunoController

- @WebMvcTest(AlunoController.class): carrega apenas a camada web desse controller
- @Autowired MockMvc: cliente HTTP de teste (executa requests sem subir servidor)
- @MockBean AlunoService: substitui o bean real por um mock (isolando controller)

```java
@WebMvcTest(controllers = AlunoController.class)
class AlunoControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private AlunoService service;

    @Test
    void buscarDeveRetornarAlunoExistente() throws Exception {
        mockMvc.perform(get("/api/alunos/1"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id", is(1)))
                .andExpect(jsonPath("$.nome", is("Gabriel")));
    }

    @Test
    void incluirDeveRetornarAlunoCriado() throws Exception {
        Mockito.when(service.adicionar(anyString(), anyString()))
                .thenReturn(new Aluno(100L, "Sofia", "Nutrição"));

        String json = "{ \"nome\":\"Sofia\", \"curso\":\"Nutrição\" }";
        mockMvc.perform(post("/api/alunos")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(json))
                .andExpect(status().isOk()) // controller atual retorna 200
                .andExpect(jsonPath("$.id", is(100)))
                .andExpect(jsonPath("$.nome", is("Sofia")))
                .andExpect(jsonPath("$.curso", is("Nutrição")));
    }
```

Testes Unitários em Java — REST API (Projeto: restapi) – Por que MockMvc + @WebMvcTest?

- Velocidade: sobe apenas a camada MVC
- Isolamento: lógica de negócio é simulada via @MockBean
- Confiabilidade: valida rotas, headers, status e shape do JSON (jsonPath)

```java
@Test
@DisplayName("adicionar: deve criar aluno com nome/curso aparados e retornar com id")
void adicionarDeveCriarComSucesso() {
    when(repo.adicionar(any(Aluno.class))).thenAnswer(inv -> {
        Aluno a = inv.getArgument(0);
        return new Aluno(10L, a.getNome(), a.getCurso());
    });

    Aluno salvo = service.adicionar("  Gabriel  ", "  Engenharia  ");

    ArgumentCaptor<Aluno> captor = ArgumentCaptor.forClass(Aluno.class);
    verify(repo).adicionar(captor.capture());
    Aluno enviado = captor.getValue();
    assertEquals("Gabriel", enviado.getNome());
    assertEquals("Engenharia", enviado.getCurso());

    assertNotNull(salvo);
    assertEquals(10L, salvo.getId());
    assertEquals("Gabriel", salvo.getNome());
    assertEquals("Engenharia", salvo.getCurso());
}

@Test
@DisplayName("adicionar: deve falhar quando nome for vazio")
void adicionarDeveFalharNomeVazio() {
    assertThrows(IllegalArgumentException.class, () -> service.adicionar("   ", "Curso"));
    verifyNoInteractions(repo);
}
```

Testes Unitários em Java — REST API (Projeto: restapi) – Detalhes

- “quando repo.adicionar(...) for chamado com qualquer Aluno, devolva um novo Aluno com id=10 e os mesmos campos nome e curso que chegaram no argumento”. Assim, o teste não depende de banco; o repositório é simulado;

Testes Unitários em Java — REST API (Projeto: restapi) – Detalhes

- Pelo código do AlunoService, acontece:
  - validar(nome, curso) — checa obrigatoriedade e tamanho.
  - nome.trim() e curso.trim() — remove espaços das pontas.
  - Cria Aluno(null, "Gabriel", "Engenharia") e chama repo.adicionar(...).
  - O mock retorna Aluno(10L, "Gabriel", "Engenharia").

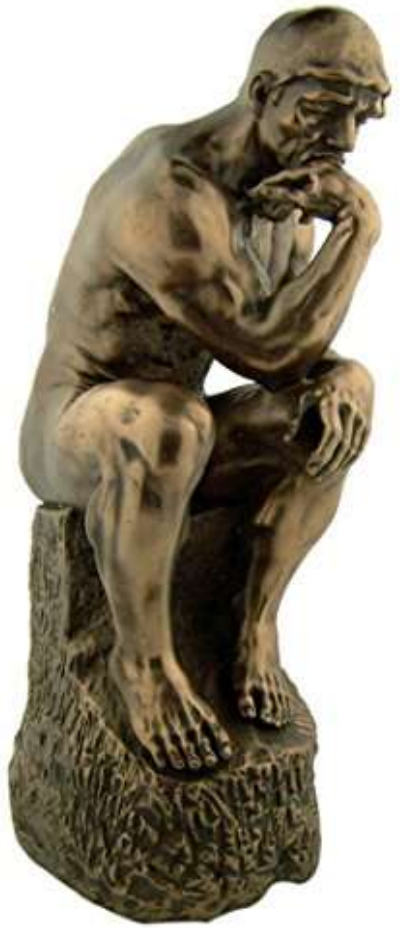
Testes Unitários em Java — REST API (Projeto: restapi) – Desafio

- Com base no projeto disponibilizado em sala, implementar os demais testes unitários nas classes AlunoController e AlunoService.
- Enviar o exercício ao final da aula para pela teams no privado, pois sua presença será contabilizada pela entrega do exercício!
- Exercício - **restapi-com-testes-buscar-incluir**

DÚVIDAS?

“A dúvida é o princípio da sabedoria.”

Aristóteles



OBRIGADO