### Exercício 11.3 – Sistema de Reservas de Quarto de Hotel

**Objetivo:** Implementar um sistema de reservas de quartos de hotel com funcionalidades básicas e criar testes unitários e de integração para repositórios, serviços e controladores.

### Parte 1: Configuração do Projeto no Spring Initializr

1. **Acessar o Spring Initializr:**
   * Vá ao site [Spring Initializr](https://start.spring.io/).
2. **Configurar o projeto:**
   * **Project:** Maven
   * **Language:** Java
   * **Spring Boot:** Última versão estável.
   * **Group:** com.example
   * **Artifact:** hotel
   * **Name:** hotel
   * **Package Name:** com.example.hotel
   * **Packaging:** Jar
   * **Java Version:** 21.
3. **Adicionar dependências:**
   * **Spring Web:** Para criar APIs REST.
   * **Spring Data JPA:** Para integração com a base de dados.
   * **H2 Database:** Para usar uma base de dados em memória nos testes.
   * **Spring Boot Starter Test:** Para suporte a testes com JUnit e Mockito.
4. **Gerar o projeto:**
   * Clique em **Generate** para descarregar o projeto como um ficheiro ZIP.
5. **Abrir o projeto no VS Code:**
   * Extraia o ficheiro ZIP.
   * No VS Code, clique em **File > Open Folder** e selecione a pasta do projeto.

### Parte 2: Estrutura do Projeto

1. **Criar as pastas necessárias:**
   * No VS Code, crie as seguintes pastas em src/main/java/com/example/hotel:
     + entity para a classe Reserva.
     + repository para o repositório ReservaRepository.
     + service para a lógica de negócios.
     + controller para os controladores REST.
   * Em src/test/java/com/example/hotel, crie:
     + repository para os testes do repositório.
     + service para os testes do serviço.
     + controller para os testes do controlador.
2. **A estrutura do projeto deverá ficar assim:**

src/  
├── main/  
│ ├── java/  
│ │ └── com.example.hotel/  
│ │ ├── HotelApplication.java  
│ │ ├── entity/  
│ │ ├── repository/  
│ │ ├── service/  
│ │ └── controller/  
├── test/  
│ └── java/  
│ └── com.example.hotel/  
│ ├── repository/  
│ ├── service/  
│ └── controller/

### Parte 3: Implementação do Modelo de Dados

**Criar a classe Reserva em entity:**

package com.example.hotel.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import java.time.LocalDate;  
  
@Entity  
public class Reserva {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String cliente;  
 private LocalDate dataEntrada;  
 private LocalDate dataSaida;  
  
 public Reserva() {}  
  
 public Reserva(String cliente, LocalDate dataEntrada, LocalDate dataSaida) {  
 this.cliente = cliente;  
 this.dataEntrada = dataEntrada;  
 this.dataSaida = dataSaida;  
 }  
  
 // Getters e setters...  
}

### Parte 4: Repositório Reserva

**Criar a interface ReservaRepository em repository:**

package com.example.hotel.repository;  
  
import com.example.hotel.entity.Reserva;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface ReservaRepository extends JpaRepository<Reserva, Long> {  
 List<Reserva> findByCliente(String cliente);  
}

### Parte 5: Serviço Reserva

**Criar a classe ReservaService em service:**

package com.example.hotel.service;  
  
import com.example.hotel.entity.Reserva;  
import com.example.hotel.repository.ReservaRepository;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class ReservaService {  
  
 private final ReservaRepository reservaRepository;  
  
 public ReservaService(ReservaRepository reservaRepository) {  
 this.reservaRepository = reservaRepository;  
 }  
  
 public List<Reserva> listarReservas() {  
 return reservaRepository.findAll();  
 }  
  
 public List<Reserva> buscarReservasPorCliente(String cliente) {  
 return reservaRepository.findByCliente(cliente);  
 }  
  
 public Reserva salvarReserva(Reserva reserva) {  
 return reservaRepository.save(reserva);  
 }  
}

### Parte 6: Controlador Reserva

**Criar a classe ReservaController em controller:**

package com.example.hotel.controller;  
  
import com.example.hotel.entity.Reserva;  
import com.example.hotel.service.ReservaService;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/reservas")  
public class ReservaController {  
  
 private final ReservaService reservaService;  
  
 public ReservaController(ReservaService reservaService) {  
 this.reservaService = reservaService;  
 }  
  
 @GetMapping  
 public List<Reserva> listarReservas() {  
 return reservaService.listarReservas();  
 }  
  
 @GetMapping("/cliente/{cliente}")  
 public List<Reserva> buscarPorCliente(@PathVariable String cliente) {  
 return reservaService.buscarReservasPorCliente(cliente);  
 }  
  
 @PostMapping  
 public Reserva salvarReserva(@RequestBody Reserva reserva) {  
 return reservaService.salvarReserva(reserva);  
 }  
}

### Parte 7: Configuração no application.properties

Adicione as configurações básicas para o H2 no ficheiro application.properties:

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=password  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

### **Parte 8: Testes**

#### 

#### **1. Estrutura dos Testes**

Certifique-se de que os testes estão organizados na seguinte estrutura:

src/  
├── test/  
│ ├── java/  
│ │ └── com.example.hotel/  
│ │ ├── controller/  
│ │ │ └── ReservaControllerTest.java  
│ │ ├── service/  
│ │ │ └── ReservaServiceTest.java  
│ │ └── repository/  
│ │ └── ReservaRepositoryTest.java

#### 

#### **2. Testes para o Repositório**

Os testes para o repositório validam a interação com a base de dados. Aqui está o exemplo completo de ReservaRepositoryTest:

package com.example.hotel.repository;  
  
import com.example.hotel.entity.Reserva;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;  
  
import java.time.LocalDate;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.assertEquals;  
import static org.junit.jupiter.api.Assertions.assertNotNull;  
  
@DataJpaTest  
public class ReservaRepositoryTest {  
  
 @Autowired  
 private ReservaRepository reservaRepository;  
  
 @Test  
 void testSaveAndFindByCliente() {  
 Reserva reserva = new Reserva("João Silva", LocalDate.now(), LocalDate.now().plusDays(2));  
 reservaRepository.save(reserva);  
  
 List<Reserva> reservas = reservaRepository.findByCliente("João Silva");  
 assertNotNull(reservas);  
 assertEquals(1, reservas.size());  
 assertEquals("João Silva", reservas.get(0).getCliente());  
 }  
}

#### **3. Testes para o Serviço**

Os testes de serviço validam a lógica de negócio isoladamente:

package com.example.hotel.service;  
  
import com.example.hotel.entity.Reserva;  
import com.example.hotel.repository.ReservaRepository;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import java.time.LocalDate;  
import java.util.Arrays;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.assertEquals;  
import static org.mockito.Mockito.when;  
  
public class ReservaServiceTest {  
  
 private final ReservaRepository reservaRepository = Mockito.mock(ReservaRepository.class);  
 private final ReservaService reservaService = new ReservaService(reservaRepository);  
  
 @Test  
 void testListarReservas() {  
 Reserva reserva1 = new Reserva("Ana", LocalDate.now(), LocalDate.now().plusDays(3));  
 Reserva reserva2 = new Reserva("Carlos", LocalDate.now(), LocalDate.now().plusDays(1));  
  
 when(reservaRepository.findAll()).thenReturn(Arrays.asList(reserva1, reserva2));  
  
 List<Reserva> reservas = reservaService.listarReservas();  
 assertEquals(2, reservas.size());  
 }  
  
 @Test  
 void testBuscarReservasPorCliente() {  
 Reserva reserva = new Reserva("Ana", LocalDate.now(), LocalDate.now().plusDays(3));  
  
 when(reservaRepository.findByCliente("Ana")).thenReturn(List.of(reserva));  
  
 List<Reserva> reservas = reservaService.buscarReservasPorCliente("Ana");  
 assertEquals(1, reservas.size());  
 assertEquals("Ana", reservas.get(0).getCliente());  
 }  
}

#### **4. Testes para o Controlador**

Os testes do controlador garantem que as requisições HTTP estão funcionando corretamente:

package com.example.hotel.controller;  
  
import com.example.hotel.entity.Reserva;  
import com.example.hotel.service.ReservaService;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.http.MediaType;  
import org.springframework.test.web.servlet.MockMvc;  
  
import java.time.LocalDate;  
import java.util.List;  
  
import static org.mockito.Mockito.when;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
  
@WebMvcTest(ReservaController.class)  
public class ReservaControllerTest {  
  
 @Autowired  
 private MockMvc mockMvc;  
  
 @MockBean  
 private ReservaService reservaService;  
  
 @Test  
 void testListarReservas() throws Exception {  
 Reserva reserva = new Reserva("Maria", LocalDate.now(), LocalDate.now().plusDays(1));  
  
 when(reservaService.listarReservas()).thenReturn(List.of(reserva));  
  
 mockMvc.perform(get("/api/reservas"))  
 .andExpect(status().isOk())  
 .andExpect(jsonPath("$[0].cliente").value("Maria"));  
 }  
  
 @Test  
 void testSalvarReserva() throws Exception {  
 Reserva reserva = new Reserva("José", LocalDate.now(), LocalDate.now().plusDays(3));  
  
 when(reservaService.salvarReserva(Mockito.any(Reserva.class))).thenReturn(reserva);  
  
 mockMvc.perform(post("/api/reservas")  
 .contentType(MediaType.APPLICATION\_JSON)  
 .content("{\"cliente\": \"José\", \"dataEntrada\": \"2025-01-14\", \"dataSaida\": \"2025-01-17\"}"))  
 .andExpect(status().isOk())  
 .andExpect(jsonPath("$.cliente").value("José"));  
 }  
}

### **Parte 9: Testar o Projeto**

#### **1. Executar o Projeto**

1. No terminal, navegue até a pasta raiz do projeto.
2. Execute o comando:

* mvn spring-boot:run

#### **2. Verificar Endpoints**

Com o servidor em execução, use ferramentas como Postman ou cURL para testar os endpoints:

* **GET /api/reservas**:
* curl -X GET http://localhost:8080/api/reservas
* **GET /api/reservas/cliente/{cliente}**:
* curl -X GET http://localhost:8080/api/reservas/cliente/João
* **POST /api/reservas**:
* curl -X POST http://localhost:8080/api/reservas \  
  -H "Content-Type: application/json" \  
  -d '{"cliente": "José", "dataEntrada": "2025-01-14", "dataSaida": "2025-01-17"}'

#### **3. Executar os Testes**

1. Para executar todos os testes, use:

* mvn test

1. Certifique-se de que todos os testes passam com sucesso antes de finalizar o desenvolvimento.