Übung Book Service Teil 2

BookRepository

Erstellen Sie das BookRepository für den Spring Data Zugriff auf die Book Entity z.B. wie folgt:package ch.std.book.repositories;

import java.util.List;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import ch.std.book.jpa.Book;

public interface BookRepository extends JpaRepository<Book, Long> {
 Optional<Book> findById(Long id);
 Optional<Book> findByIsbn(String isbn);
 List<Book> findByTitle(String text);
 List<Book> findByTitleContaining(String text);
 List<Book> findByTitleContainingIgnoreCase(String text);
 List<Book> findByTitleStartingWith(String text);
 List<Book> findByTitleEndingWith(String text);
 @Query("SELECT b FROM Book b WHERE b.isbn LIKE %:text% OR b.title LIKE %:text% OR b.description LIKE %:text% OR b.publisher LIKE %:text%")
 List<Book> findAllContaining(@Param("text") String text);
 // Experimentieren Sie mit weiteren Varianten
}Wichtig: Starten Sie nach jeder Anpassung der Repositories die Anwendung und verifizieren Sie ob solche korrekt startet.

BookRepositoryJPATest

Wir beginnen mit der Programmierung des ersten Unit Tests für dieses Projekt. Hierzu arbeiten wir mit dem Profile "unittest" und erstellen die Datei application-unittest.properties im test/resources Verzeichnis wie folgt:book.scheduler.initial.enabled=false

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.database-platform=org.hibernate.dialect.H2DialectProgrammieren Sie im test/java-Folder die Klasse ch.std.book.repositories.BookRepositoryJPATest:package ch.std.book.repositories;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@ExtendWith(SpringExtension.class)
@DataJpaTest
@ActiveProfiles("unittest")
public class BookRepositoryJPATest {
 @BeforeEach
 public void setup() {
 }
 @Test
 public void testFindById() {
 }
 @Test
 public void testFindByIsbn() {
 }
 @Test
 public void testFindAll() {
 }
 @Test
 public void testFindAllSorted() {
 }
 @Test
 public void testFindByTitle() {
 }
 @Test
 public void testFindByTitleContaining() {
 }
 @Test
 public void testFindByTitleContainingIgnoringCase() {
 }
 @Test
 public void testFindAllContaining() {
 }
}@DataJpaTest Klassen verwenden die H2 Datenbank. Damit wir solche benutzen können benötigen wir die folgende Maven Dependency:<dependency>
 <groupId>com.h2database</groupId>
 <artifactId>h2</artifactId>
</dependency>Das schwierigste an der Programmierung der Unit Tests ist das Aufsetzen der Testdaten. Solche werden in der Regel über die Setup-Methode und die Annotation @BeforeEach aufgebaut und je nach Use Case mit der Methode markiert mit @AfterEach wieder abgebaut. Das folgende Listing zeigt den möglichen Setup für die Testdaten:...
@ExtendWith(SpringExtension.class)
@DataJpaTest
@ActiveProfiles("unittest") // alternativ kann mit der @TestPropertySource Annotation gearbeitet werden
public class BookRepositoryJPATest {

 @Autowired
 private BookRepository bookRepository;

 private Book book;
 private List<Book> bookList;

 @BeforeEach
 public void setup() {
 book = new Book("978-1617292545", "this is the title", "that's the description", "my publisher");
 this.bookRepository.save(book);
 // insert more books
 this.bookList = new ArrayList<>();
 for (int i = 0; i < 10; i++) {
 Book localBook = null;
 if (i % 2 == 0) {
 localBook = new Book("isbn-" + i,

"title-" + i, "description-" + i, "publisher-" + i);
  } else {
 localBook = new Book("isbn-".toUpperCase() + i, "title-".toUpperCase() + i,
  "description-".toUpperCase() + i, "publisher-".toUpperCase() + i);
  }
  this.bookRepository.save(localBook);
  this.bookList.add(localBook);
 }
 }

 @AfterEach
 public void tearDown() {
 this.bookRepository.deleteAllInBatch();
 }
 // ...
}Das Beispiel erwartet weitere Konstruktoren in der Klasse Book.Programmieren Sie die Unit Tests so aus, dass solche Sinn und Zweck erfüllen. Testen Sie auch die restlichen BookRepository Methoden, sofern Sie solche programmiert haben.Eine Musterlösung finden Sie unter BookRepositoryJPATest.java.

BookController

Erstellen Sie die Klasse BookController als Rest Service und implementieren Sie die Methoden gemäss der Vorlage:package ch.std.book.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import ch.std.book.jpa.Book;
import ch.std.book.repositories.BookRepository;

@RestController
public class BookController {
  BookRepository bookRepository;

  public BookController(BookRepository bookRepository) {
  this.bookRepository = bookRepository;
  }

  @GetMapping("/rest/books")
  public Book[] getBooks(@RequestParam(value = "value", required = false) String value) {

 }

  @GetMapping("/rest/book/{id}")
  public Book getBookById(@PathVariable Long id) {

  }

 @GetMapping("/rest/book")
  public Book getBookByIsbn(@RequestParam String isbn) {

  }

  @PostMapping("/rest/book")
  public Book createBook(@RequestBody Book book) {

  }

 @PutMapping("/rest/book/{id}")
  public Book updateBook(@RequestBody Book book, @PathVariable Long id) {

  }

 @DeleteMapping("/rest/book/{id}")
  public ResponseEntity<?> deleteBookById(@PathVariable Long id) {

  }

  public static class BookNotFoundException extends RuntimeException {

  }
}Programmieren Sie die Methoden so aus, dass die Daten aus der Datenbank gelesen und geschrieben werden gemäss Anforderung. Definieren Sie noch den Server Context inkl. Port in der Datei application.properties z.B. wie folgt:server.port=8080
server.servlet.context-path=/bookStarten Sie die Applikation und testen Sie ob die Bücher geladen werden:

BookControllerTests

Erstellen Sie die Klasse BookControllerTest und implementieren Sie die Methoden gemäss der Vorlage:package ch.std.book.integration;

import java.util.ArrayList;
import java.util.List;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.web.server.LocalServerPort;
import org.springframework.http.HttpEntity;
import org.springframework.test.context.ActiveProfiles;

import ch.std.book.jpa.Book;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("unittest")
public class

BookControllerTests {

  Logger logger = LoggerFactory.getLogger(BookControllerTests.class);

  @Autowired
  private TestRestTemplate restTemplate;
  private List<Book> bookList = new ArrayList<>();

  @BeforeEach public void setup() {
 logger.info("BookControllerTests.setup");
    this.bookList.clear();
    // create test book
    String postUrl = "/rest/book";
    HttpEntity request = new HttpEntity<>(new Book("1234567890","test", "test", "test"));
    logger.info("BookControllerTests.setup, postUrl = " + postUrl);
    Book testBook = this.restTemplate.postForObject(postUrl, request, Book.class);
    logger.info("BookControllerTests.setup, testBook = " + testBook);
    this.bookList.add(testBook);
  }

  @Test public void testGetBooks() throws Exception { }

  @Test public void testGetAllBooks() throws Exception { }

  @Test public void testGetBookById() throws Exception { }
}Sie können sich an der Klasse CityControllerTests orientieren.

Lösung

Eine mögliche Lösung finden Sie als Maven Projekt bookservice2.zip

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Copyright

https://www.simtech-ag.ch/übung-book-service-teil-2