

## 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.H2Dialect
```

Programmieren 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 testFindByTitleContainingIgnoreCase() {}

    @Test
    public void testFindAllContaining() {}
}
```

Klassen verwenden die H2 Datenbank. Damit wir solche benutzen können benötigen wir die folgende Maven

```
Dependency:
<dependency>
```

```
<groupId>com.h2database</groupId>
```

```
<artifactId>h2</artifactId>
```

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,
```

```

    &#34;title-&#34; + i, &#34;description-&#34; + i, &#34;publisher-&#34; + i);&#xA; } else {&#xA;
    localBook = new Book(&#34;isbn-&#34;.toUpperCase() + i, &#34;title-&#34;.toUpperCase() +
    i,&#xA; &#34;description-&#34;.toUpperCase() + i, &#34;publisher-&#34;.toUpperCase() +
    i);&#xA; }&#xA; this.bookRepository.save(localBook);&#xA; this.bookList.add(localBook);&#xA;
    }&#xA; }&#xA; }&#xA; @AfterEach&#xA; public void tearDown() {&#xA;
    this.bookRepository.deleteAllInBatch();&#xA; }&#xA; // ...&#xA;}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;&#xA;&#xA;import
org.springframework.beans.factory.annotation.Autowired;&#xA;import
org.springframework.http.ResponseEntity;&#xA;import
org.springframework.web.bind.annotation.DeleteMapping;&#xA;import
org.springframework.web.bind.annotation.GetMapping;&#xA;import
org.springframework.web.bind.annotation.PathVariable;&#xA;import
org.springframework.web.bind.annotation.PostMapping;&#xA;import
org.springframework.web.bind.annotation.PutMapping;&#xA;import
org.springframework.web.bind.annotation.RequestBody;&#xA;import
org.springframework.web.bind.annotation.RequestParam;&#xA;import
org.springframework.web.bind.annotation.RestController;&#xA;&#xA;import
ch.std.book.jpa.Book;&#xA;import
ch.std.book.repositories.BookRepository;&#xA;&#xA;@RestController&#xA;public class
BookController {&#xA; BookRepository bookRepository;&#xA;&#xA; public
BookController(BookRepository bookRepository) {&#xA; this.bookRepository =
bookRepository;&#xA; }&#xA;&#xA; @GetMapping(&#34;/rest/books&#34;)&#xA; public Book[]
getBooks(@RequestParam(value = &#34;value&#34;, required = false) String value) {&#xA;&#xA;
}&#xA;&#xA; @GetMapping(&#34;/rest/book/{id}&#34;)&#xA; public Book
getBookById(@PathVariable Long id) {&#xA;&#xA; }&#xA;&#xA;
@GetMapping(&#34;/rest/book&#34;)&#xA; public Book getBookByIsbn(@RequestParam String
isbn) {&#xA;&#xA; }&#xA;&#xA; @PostMapping(&#34;/rest/book&#34;)&#xA; public Book
createBook(@RequestBody Book book) {&#xA;&#xA; }&#xA;&#xA;
@PutMapping(&#34;/rest/book/{id}&#34;)&#xA; public Book updateBook(@RequestBody Book
book, @PathVariable Long id) {&#xA;&#xA; }&#xA;&#xA;
@DeleteMapping(&#34;/rest/book/{id}&#34;)&#xA; public ResponseEntity<?>
deleteBookById(@PathVariable Long id) {&#xA;&#xA; }&#xA;&#xA; public static class
BookNotFoundException extends RuntimeException {&#xA;&#xA; }&#xA;}

```

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&#xA;server.servlet.context-path=/book Starten 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;&#xA;&#xA;import java.util.ArrayList;&#xA;import
java.util.List;&#xA;&#xA;import org.junit.jupiter.api.BeforeEach;&#xA;import
org.junit.jupiter.api.Test;&#xA;import org.slf4j.Logger;&#xA;import
org.slf4j.LoggerFactory;&#xA;import
org.springframework.beans.factory.annotation.Autowired;&#xA;import
org.springframework.boot.test.context.SpringBootTest;&#xA;import
org.springframework.boot.test.context.SpringBootTest.WebEnvironment;&#xA;import
org.springframework.boot.test.web.client.TestRestTemplate;&#xA;import
org.springframework.boot.web.server.LocalServerPort;&#xA;import
org.springframework.http.HttpEntity;&#xA;import
org.springframework.test.context.ActiveProfiles;&#xA;&#xA;import
ch.std.book.jpa.Book;&#xA;&#xA;@SpringBootTest(webEnvironment =
WebEnvironment.RANDOM_PORT)&#xA;@ActiveProfiles(&#34;unittest&#34;)&#xA;public class

```

```
BookControllerTests {&#xA; &#xA; Logger logger =
LoggerFactory.getLogger(BookControllerTests.class);&#xA;&#xA; @Autowired&#xA; private
TestRestTemplate restTemplate;&#xA; private List&lt;Book&gt; bookList = new
ArrayList&lt;&gt;();&#xA;&#xA; @BeforeEach public void setup() { &#xA;
logger.info(&#34;BookControllerTests.setup&#34;);&#xA; this.bookList.clear();&#xA; // create
test book&#xA; String postUrl = &#34;/rest/book&#34;;&#xA; HttpEntity request = new
HttpEntity&lt;&gt;(new Book(&#34;1234567890&#34;, &#34;test&#34;, &#34;test&#34;,
&#34;test&#34;));&#xA; logger.info(&#34;BookControllerTests.setup, postUrl = &#34; +
postUrl);&#xA; Book testBook = this.restTemplate.postForObject(postUrl, request,
Book.class);&#xA; logger.info(&#34;BookControllerTests.setup, testBook = &#34; +
testBook);&#xA; this.bookList.add(testBook);&#xA; }&#xA;&#xA; @Test public void
testGetBooks() throws Exception { }&#xA;&#xA; @Test public void testGetAllBooks() throws
Exception { }&#xA;&#xA; @Test public void testGetBookById() throws Exception { }&#xA;}
```

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

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.  
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsausschluss.  
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

### Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

<https://www.simtech-ag.ch/tearDown>