Spring Boot:Best Book

**Description**

**Requirements**

John has joined a new consultancy firm as a backend developer and has been asked to code a basic CRUD operation backend using SpringBoot with MySQL.

Entity Details:

The Books entity should have the following attributes:

* **bookId**: the unique id of the book (Integer)
* **bookName**: the name of the book (String)
* **author**: the author of the book (String)
* **bookCostPrice**: the price of the book (String)

Here is an example of a Book JSON object:

{

"bookId": 1,

"bookName": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"bookCostPrice": "10.99"

}

You are provided with the implementation of the models required for all the APIs. The task is to implement a set of REST services that exposes the endpoints and allows for adding, listing, and deleting the collection of book records in the following ways:

| API Route | API Type | Success Response Code | Validation Error Code |

|-------------------------|----------|------------------------|------------------------|

| /books | PUT | 200 | 400 |

| /books | GET | 200 | 404 |

| /books | DELETE | 200 | 404 |

Task 1: Service Layer Implementation in BooksService

Implement the logic in the service layer using BooksRepository

**Method Details**:

* **getAllBooks()**: Retrieve all book records from the repository using the findAll() method provided by BooksRepository
* **saveOrUpdate()**: Save a new book record or update an existing one using the save() method of BooksRepository
* **deleteAll()**: Delete all book records from the repository using the deleteAll() method of BooksRepository

Task 2: REST API Endpoints in BooksController

**PUT request to /books**

Save Book to Database

**HTTP Status Code**:

* 200 - For a successful response
* 400- If any of the parameters are empty, For booking, add another validation that the id cannot be less than 0 and more than 1000, for any invalid entry

**GET request to /books**

Get All Books

**Response Body**: JSON object representing the details of the book

**HTTP Status Code**:

* 200 - For a successful response
* 404 - If no books are found

**DELETE request to /books**

Delete All Books

**HTTP Status Code**:

* 200 - For a successful response
* 404 - For an unsuccessful response

Complete the given project to pass all the test cases when running the provided unit tests.

**Example Requests and Responses:**

PUT request to /books:

Request Body

{

"bookName": "Sapiens",

"author": "Noah Harrari",

"bookCostPrice": "99.99"

}

The response code is 200 and the response body, when converted to JSON, is as follows:

{

"bookId": 6,

"bookName": "Sapiens",

"author": "Noah Harrari",

"bookCostPrice": "99.99"

}

GET request to /books:

Response Body

[

{

"bookId": 2,

"bookName": "1984",

"author": "George Orwell",

"bookCostPrice": "8.99"

}

]

DELETE request to /books:

The response code is 200

BooksController.java

package com.example.book.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

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.RestController;

import com.example.book.entity.Books;

import com.example.book.service.BooksService;

@RestController

public class BooksController

{

    @Autowired

    BooksService booksService;

    @CrossOrigin

    @GetMapping("/books")

    private ResponseEntity<List<Books>> getAllBooks()

    {

         // code here ...

         return new ResponseEntity<>(null);

    }

    @CrossOrigin

    @DeleteMapping("/books")

    private ResponseEntity<String> deleteBook()

    {

         // code here ...

         return new ResponseEntity<>(null);

    }

    @CrossOrigin

    @PutMapping("/books")

    private ResponseEntity<Books> update(@RequestBody Books books)

    {

        // code here ...

        return new ResponseEntity<>(null);

    }

}

Books.java

package com.example.book.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import com.sun.istack.NotNull;

@Entity

@Table(name = "books")

public class Books

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column

    @NotNull

    private int bookId;

    @Column

    @NotNull

    private String bookName;

    @Column

    @NotNull

    private String author;

    @Column

    @NotNull

    private double bookCostPrice;

    public Books() {

    }

    public Books(String bookName, String author, double bookCostPrice) {

        this.bookName = bookName;

        this.author = author;

        this.bookCostPrice = bookCostPrice;

    }

    public int getBookId() {

        return bookId;

    }

    public void setBookId(int bookId) {

        this.bookId = bookId;

    }

    public String getBookName() {

        return bookName;

    }

    public void setBookName(String bookName) {

        this.bookName = bookName;

    }

    public String getAuthor() {

        return author;

    }

    public void setAuthor(String author) {

        this.author = author;

    }

    public double getBookCostPrice() {

        return bookCostPrice;

    }

    public void setBookCostPrice(double bookCostPrice) {

        this.bookCostPrice = bookCostPrice;

    }

    @Override

    public String toString() {

        return "Books [bookId=" + bookId + ", bookName=" + bookName + ", author=" + author + ", bookCostPrice="

                + bookCostPrice + "]";

    }

}

BooksRepository.java

package com.example.book.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.book.entity.Books;

//repository that extends JpaRepository

public interface BooksRepository extends JpaRepository<Books, Integer>

{

}

BooksService.java

package com.example.book.service;

import java.util.ArrayList;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.book.entity.Books;

import com.example.book.repository.BooksRepository;

@Service

public class BooksService

{

    @Autowired

    BooksRepository booksRepository;

    public List<Books> getAllBooks()

    {

        // code here ...

        return null;

    }

    //saving a specific record by using the method save() of CrudRepository

    public Books saveOrUpdate(Books books)

    {

        // code here ...

        return null;

    }

    //deleting all records

    public void deleteAll()

    {

        // code here ...

    }

}

AppTest.java

package com.example.book;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;

import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.nio.charset.Charset;

import java.util.ArrayList;

import java.util.List;

import javax.transaction.Transactional;

import static org.hamcrest.CoreMatchers.containsString;

import static org.hamcrest.CoreMatchers.is;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

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.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.annotation.DirtiesContext;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

import com.example.book.entity.Books;

import com.example.book.repository.BooksRepository;

import com.example.book.service.BooksService;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.ObjectWriter;

import com.fasterxml.jackson.databind.SerializationFeature;

@SpringBootTest()

@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE\_EACH\_TEST\_METHOD)

@AutoConfigureMockMvc

public class AppTest {

    public static final MediaType APPLICATION\_JSON\_UTF8 = new MediaType(MediaType.APPLICATION\_JSON.getType(), MediaType.APPLICATION\_JSON.getSubtype(), Charset.forName("utf8"));

    @Autowired

    private MockMvc mockMvc;

    @Autowired

    BooksService booksService;

    @Autowired

    BooksRepository booksRepository;

    @BeforeEach

    public void setUp() throws Exception {

        Books books1 = new Books();

        books1.setBookId(1);

        books1.setBookName("Merchant Of Venice");

        books1.setAuthor("Shakespeare");

        books1.setBookCostPrice(50.87);

        booksRepository.save(books1);

        Books books2 = new Books();

        books2.setBookId(2);

        books2.setBookName("Lost City");

        books2.setAuthor("Shawn Medes");

        books2.setBookCostPrice(97.99);

        booksRepository.save(books2);

    }

    @Test

    @Transactional

    @DisplayName("save Books details")

    public void SaveBooks() throws Exception {

        Books books3 = new Books();

        books3.setBookId(3);

        books3.setBookName("Smart City");

        books3.setAuthor("Sameer Joshi");

        books3.setBookCostPrice(80.99);

        booksRepository.save(books3);

        ObjectMapper mapper = new ObjectMapper();

        mapper.configure(SerializationFeature.WRAP\_ROOT\_VALUE, false);

        ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();

        String requestJson=ow.writeValueAsString(books3);

        mockMvc.perform(put("/books").contentType(APPLICATION\_JSON\_UTF8).content(requestJson))

                .andDo(print())

                .andExpect(status().isCreated())

                .andExpect(jsonPath("$.bookId", is(3)))

                .andExpect(jsonPath("$.bookName", is("Smart City")))

                .andExpect(jsonPath("$.author", is("Sameer Joshi")))

                .andExpect(jsonPath("$.bookCostPrice", is(80.99)));

    }

    @Test

    @DisplayName("save Books details for Bad Request")

    public void SaveBooksBadRequest() throws Exception {

        Books books3 = new Books();

        books3.setBookId(3);

        books3.setBookName("");

        books3.setAuthor("Sameer Joshi");

        books3.setBookCostPrice(80.99);

        booksRepository.save(books3);

        ObjectMapper mapper = new ObjectMapper();

        mapper.configure(SerializationFeature.WRAP\_ROOT\_VALUE, false);

        ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();

        String requestJson=ow.writeValueAsString(books3);

        mockMvc.perform(put("/books").contentType(APPLICATION\_JSON\_UTF8).content(requestJson))

                .andDo(print())

                .andExpect(status().isBadRequest());

    }

    @Test

    @DisplayName("Get Bad Request response for ivalid cost price")

    public void SaveBooksBadRequest\_2() throws Exception {

        Books books3 = new Books();

        books3.setBookId(3);

        books3.setBookName("Smart City");

        books3.setAuthor("Sameer Joshi");

        books3.setBookCostPrice(-4);

        booksRepository.save(books3);

        ObjectMapper mapper = new ObjectMapper();

        mapper.configure(SerializationFeature.WRAP\_ROOT\_VALUE, false);

        ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();

        String requestJson=ow.writeValueAsString(books3);

        mockMvc.perform(put("/books").contentType(APPLICATION\_JSON\_UTF8).content(requestJson))

                .andDo(print())

                .andExpect(status().isBadRequest());

    }

    @Test

    @Transactional

    @DisplayName("Update the book details")

    public void UpdateBooks() throws Exception {

        Books books3 = booksRepository.getById(2);

        books3.setBookName("Lost City");

        books3.setAuthor("Shawn Medes");

        books3.setBookCostPrice(20.99);

        booksRepository.save(books3);

        ObjectMapper mapper = new ObjectMapper();

        mapper.configure(SerializationFeature.WRAP\_ROOT\_VALUE, false);

        ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();

        String requestJson=ow.writeValueAsString(books3);

        mockMvc.perform(put("/books").contentType(APPLICATION\_JSON\_UTF8).content(requestJson))

                .andDo(print())

                .andExpect(status().isCreated())

                .andExpect(jsonPath("$.bookId", is(2)))

                .andExpect(jsonPath("$.bookName", is("Lost City")))

                .andExpect(jsonPath("$.author", is("Shawn Medes")))

                .andExpect(jsonPath("$.bookCostPrice", is(20.99)));

    }

    @Test

    @DisplayName("Get Books details")

    public void GetBooksOK() throws Exception {

        mockMvc.perform(get("/books").contentType(APPLICATION\_JSON\_UTF8))

                .andDo(print())

                .andExpect(status().isOk())

                .andExpect(jsonPath("$[0].bookId", is(1)))

                .andExpect(jsonPath("$[0].bookName", is("Merchant Of Venice")))

                .andExpect(jsonPath("$[0].author", is("Shakespeare")))

                .andExpect(jsonPath("$[0].bookCostPrice", is(50.87)))

                .andExpect(jsonPath("$[1].bookId", is(2)))

                .andExpect(jsonPath("$[1].bookName", is("Lost City")))

                .andExpect(jsonPath("$[1].author", is("Shawn Medes")))

                .andExpect(jsonPath("$[1].bookCostPrice", is(97.99)));

    }

    @Test

    @DisplayName("Get Books details Not Found")

    public void GetBooksNF() throws Exception {

        booksRepository.delete(booksRepository.getById(1));

        booksRepository.delete(booksRepository.getById(2));

        mockMvc.perform(get("/books").contentType(APPLICATION\_JSON\_UTF8))

                .andDo(print())

                .andExpect(status().isNotFound());

    }

    @Test

    @DisplayName("Delete Books details")

    public void DeleteBooksOK() throws Exception {

        mockMvc.perform(delete("/books").contentType(APPLICATION\_JSON\_UTF8))

                .andDo(print())

                .andExpect(status().isOk());

    }

    @Test

    @DisplayName("Books details Not Found")

    public void DeleteBooksNF() throws Exception {

        booksRepository.delete(booksRepository.getById(1));

        booksRepository.delete(booksRepository.getById(2));

        mockMvc.perform(delete("/books").contentType(APPLICATION\_JSON\_UTF8))

                .andDo(print())

                .andExpect(status().isNotFound());

    }

}