**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

Solution:

**LibraryManagementApplication.java:**

package com.library;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class LibraryManagementApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(LibraryManagementApplication.class, args);  
 System.*out*.println("=== Spring Boot Library Management Application Started ===");  
 System.*out*.println("Access H2 Database Console: http://localhost:8080/h2-console");  
 System.*out*.println("API Base URL: http://localhost:8080/api/books");  
 System.*out*.println("========================================================");  
 }  
}

**BookController.java:**

package com.library.controller;  
  
import com.library.entity.Book;  
import com.library.service.BookService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.validation.annotation.Validated;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.validation.Valid;  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/books")  
@Validated  
@CrossOrigin(origins = "\*")  
public class BookController {  
  
 private final BookService bookService;  
  
 @Autowired  
 public BookController(BookService bookService) {  
 this.bookService = bookService;  
 }  
  
   
 @GetMapping  
 public ResponseEntity<List<Book>> getAllBooks() {  
 List<Book> books = bookService.getAllBooks();  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @GetMapping("/{id}")  
 public ResponseEntity<Book> getBookById(@PathVariable Long id) {  
 Optional<Book> book = bookService.getBookById(id);  
 return book.map(ResponseEntity::*ok*)  
 .orElse(ResponseEntity.*notFound*().build());  
 }  
  
 @GetMapping("/isbn/{isbn}")  
 public ResponseEntity<Book> getBookByIsbn(@PathVariable String isbn) {  
 Optional<Book> book = bookService.getBookByIsbn(isbn);  
 return book.map(ResponseEntity::*ok*)  
 .orElse(ResponseEntity.*notFound*().build());  
 }  
  
 @PostMapping  
 public ResponseEntity<Book> createBook(@Valid @RequestBody Book book) {  
 try {  
 Book createdBook = bookService.createBook(book);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(createdBook);  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().build();  
 }  
 }  
  
   
 @PutMapping("/{id}")  
 public ResponseEntity<Book> updateBook(@PathVariable Long id, @Valid @RequestBody Book bookDetails) {  
 try {  
 Book updatedBook = bookService.updateBook(id, bookDetails);  
 return ResponseEntity.*ok*(updatedBook);  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteBook(@PathVariable Long id) {  
 try {  
 bookService.deleteBook(id);  
 return ResponseEntity.*noContent*().build();  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
  
 @GetMapping("/search")  
 public ResponseEntity<List<Book>> searchBooks(@RequestParam String keyword) {  
 List<Book> books = bookService.searchBooks(keyword);  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @GetMapping("/search/title")  
 public ResponseEntity<List<Book>> searchBooksByTitle(@RequestParam String title) {  
 List<Book> books = bookService.searchBooksByTitle(title);  
 return ResponseEntity.*ok*(books);  
 }  
  
   
 @GetMapping("/search/author")  
 public ResponseEntity<List<Book>> searchBooksByAuthor(@RequestParam String author) {  
 List<Book> books = bookService.searchBooksByAuthor(author);  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @GetMapping("/search/genre")  
 public ResponseEntity<List<Book>> searchBooksByGenre(@RequestParam String genre) {  
 List<Book> books = bookService.searchBooksByGenre(genre);  
 return ResponseEntity.*ok*(books);  
 }  
  
 @GetMapping("/available")  
 public ResponseEntity<List<Book>> getAvailableBooks() {  
 List<Book> books = bookService.getAvailableBooks();  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @GetMapping("/unavailable")  
 public ResponseEntity<List<Book>> getUnavailableBooks() {  
 List<Book> books = bookService.getUnavailableBooks();  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @PatchMapping("/{id}/availability")  
 public ResponseEntity<Book> updateBookAvailability(@PathVariable Long id, @RequestParam Boolean available) {  
 try {  
 Book updatedBook = bookService.updateBookAvailability(id, available);  
 return ResponseEntity.*ok*(updatedBook);  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
  
 @GetMapping("/year/{year}")  
 public ResponseEntity<List<Book>> getBooksByPublicationYear(@PathVariable Integer year) {  
 List<Book> books = bookService.getBooksByPublicationYear(year);  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @GetMapping("/published/after/{year}")  
 public ResponseEntity<List<Book>> getBooksPublishedAfter(@PathVariable Integer year) {  
 List<Book> books = bookService.getBooksPublishedAfter(year);  
 return ResponseEntity.*ok*(books);  
 }  
  
 @GetMapping("/published/before/{year}")  
 public ResponseEntity<List<Book>> getBooksPublishedBefore(@PathVariable Integer year) {  
 List<Book> books = bookService.getBooksPublishedBefore(year);  
 return ResponseEntity.*ok*(books);  
 }  
  
  
 @GetMapping("/stats")  
 public ResponseEntity<BookService.LibraryStats> getLibraryStats() {  
 BookService.LibraryStats stats = bookService.getLibraryStats();  
 return ResponseEntity.*ok*(stats);  
 }  
  
  
 @ExceptionHandler(IllegalArgumentException.class)  
 public ResponseEntity<String> handleIllegalArgumentException(IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body("Error: " + e.getMessage());  
 }  
}

**Book.java:**

package com.library.entity;  
  
import javax.persistence.\*;  
import javax.validation.constraints.NotBlank;  
import javax.validation.constraints.NotNull;  
import javax.validation.constraints.Pattern;  
import javax.validation.constraints.Size;  
import java.time.LocalDateTime;  
  
@Entity  
@Table(name = "books")  
public class Book {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "book\_id")  
 private Long id;  
  
 @NotBlank(message = "Title is required")  
 @Size(min = 1, max = 255, message = "Title must be between 1 and 255 characters")  
 @Column(name = "title", nullable = false)  
 private String title;  
  
 @NotBlank(message = "Author is required")  
 @Size(min = 1, max = 255, message = "Author must be between 1 and 255 characters")  
 @Column(name = "author", nullable = false)  
 private String author;  
  
 @NotBlank(message = "ISBN is required")  
 @Pattern(regexp = "^(?:ISBN(?:-1[03])?:?\\s\*)?(?=[0-9X]{10}$|(?=(?:[0-9]+[-\\s]){3})[-\\s0-9X]{13}$|97[89][0-9]{10}$|(?=(?:[0-9]+[-\\s]){4})[-\\s0-9]{17}$)(?:97[89][-\\s]?)?[0-9]{1,5}[-\\s]?[0-9]+[-\\s]?[0-9]+[-\\s]?[0-9X]$",  
 message = "Invalid ISBN format")  
 @Column(name = "isbn", unique = true, nullable = false)  
 private String isbn;  
  
 @Column(name = "genre")  
 private String genre;  
  
 @Column(name = "publication\_year")  
 private Integer publicationYear;  
  
 @Column(name = "available", nullable = false)  
 private Boolean available = true;  
  
 @Column(name = "created\_at", nullable = false, updatable = false)  
 private LocalDateTime createdAt;  
  
 @Column(name = "updated\_at")  
 private LocalDateTime updatedAt;  
  
  
 public Book() {  
 this.createdAt = LocalDateTime.*now*();  
 this.updatedAt = LocalDateTime.*now*();  
 }  
  
 public Book(String title, String author, String isbn) {  
 this();  
 this.title = title;  
 this.author = author;  
 this.isbn = isbn;  
 }  
  
 public Book(String title, String author, String isbn, String genre, Integer publicationYear) {  
 this(title, author, isbn);  
 this.genre = genre;  
 this.publicationYear = publicationYear;  
 }  
  
   
 @PrePersist  
 protected void onCreate() {  
 this.createdAt = LocalDateTime.*now*();  
 this.updatedAt = LocalDateTime.*now*();  
 }  
  
 @PreUpdate  
 protected void onUpdate() {  
 this.updatedAt = LocalDateTime.*now*();  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public String getAuthor() {  
 return author;  
 }  
  
 public void setAuthor(String author) {  
 this.author = author;  
 }  
  
 public String getIsbn() {  
 return isbn;  
 }  
  
 public void setIsbn(String isbn) {  
 this.isbn = isbn;  
 }  
  
 public String getGenre() {  
 return genre;  
 }  
  
 public void setGenre(String genre) {  
 this.genre = genre;  
 }  
  
 public Integer getPublicationYear() {  
 return publicationYear;  
 }  
  
 public void setPublicationYear(Integer publicationYear) {  
 this.publicationYear = publicationYear;  
 }  
  
 public Boolean getAvailable() {  
 return available;  
 }  
  
 public void setAvailable(Boolean available) {  
 this.available = available;  
 }  
  
 public LocalDateTime getCreatedAt() {  
 return createdAt;  
 }  
  
 public void setCreatedAt(LocalDateTime createdAt) {  
 this.createdAt = createdAt;  
 }  
  
 public LocalDateTime getUpdatedAt() {  
 return updatedAt;  
 }  
  
 public void setUpdatedAt(LocalDateTime updatedAt) {  
 this.updatedAt = updatedAt;  
 }  
  
  
 @Override  
 public String toString() {  
 return "Book{" +  
 "id=" + id +  
 ", title='" + title + '\'' +  
 ", author='" + author + '\'' +  
 ", isbn='" + isbn + '\'' +  
 ", genre='" + genre + '\'' +  
 ", publicationYear=" + publicationYear +  
 ", available=" + available +  
 ", createdAt=" + createdAt +  
 ", updatedAt=" + updatedAt +  
 '}';  
 }  
  
   
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Book book = (Book) o;  
 return isbn != null && isbn.equals(book.isbn);  
 }  
  
 @Override  
 public int hashCode() {  
 return isbn != null ? isbn.hashCode() : 0;  
 }  
}

**BookRepository.java:**

package com.library.repository;  
  
import com.library.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Repository  
public interface BookRepository extends JpaRepository<Book, Long> {  
  
  
 List<Book> findByTitleContainingIgnoreCase(String title);  
  
  
 List<Book> findByAuthorContainingIgnoreCase(String author);  
  
  
 List<Book> findByGenreContainingIgnoreCase(String genre);  
  
   
 Optional<Book> findByIsbn(String isbn);  
  
  
 boolean existsByIsbn(String isbn);  
  
  
 List<Book> findByAvailable(Boolean available);  
  
  
 List<Book> findByPublicationYear(Integer publicationYear);  
  
  
 List<Book> findByPublicationYearGreaterThan(Integer year);  
  
  
 List<Book> findByPublicationYearLessThan(Integer year);  
  
   
 @Query("SELECT b FROM Book b WHERE LOWER(b.title) LIKE LOWER(CONCAT('%', :keyword, '%')) OR LOWER(b.author) LIKE LOWER(CONCAT('%', :keyword, '%'))")  
 List<Book> findByTitleOrAuthorContainingIgnoreCase(@Param("keyword") String keyword);  
  
  
 @Query("SELECT b.genre, COUNT(b) FROM Book b GROUP BY b.genre")  
 List<Object[]> countBooksByGenre();  
  
   
 @Query("SELECT COUNT(b) FROM Book b WHERE b.available = true")  
 long countAvailableBooks();  
  
  
 @Query("SELECT COUNT(b) FROM Book b")  
 long countTotalBooks();  
  
   
 @Query("SELECT b FROM Book b WHERE " +  
 "(:title IS NULL OR LOWER(b.title) LIKE LOWER(CONCAT('%', :title, '%'))) AND " +  
 "(:author IS NULL OR LOWER(b.author) LIKE LOWER(CONCAT('%', :author, '%'))) AND " +  
 "(:genre IS NULL OR LOWER(b.genre) LIKE LOWER(CONCAT('%', :genre, '%'))) AND " +  
 "(:available IS NULL OR b.available = :available)")  
 List<Book> findBooksByCriteria(@Param("title") String title,  
 @Param("author") String author,  
 @Param("genre") String genre,  
 @Param("available") Boolean available);  
}

**BookService.java:**

package com.library.service;  
  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional  
public class BookService {  
  
 private final BookRepository bookRepository;  
  
 @Autowired  
 public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
  
 public Book createBook(Book book) {  
 if (bookRepository.existsByIsbn(book.getIsbn())) {  
 throw new IllegalArgumentException("Book with ISBN " + book.getIsbn() + " already exists");  
 }  
 return bookRepository.save(book);  
 }  
  
  
 @Transactional(readOnly = true)  
 public List<Book> getAllBooks() {  
 return bookRepository.findAll();  
 }  
  
  
 @Transactional(readOnly = true)  
 public Optional<Book> getBookById(Long id) {  
 return bookRepository.findById(id);  
 }  
  
  
 @Transactional(readOnly = true)  
 public Optional<Book> getBookByIsbn(String isbn) {  
 return bookRepository.findByIsbn(isbn);  
 }  
  
  
 public Book updateBook(Long id, Book bookDetails) {  
 Book book = bookRepository.findById(id)  
 .orElseThrow(() -> new IllegalArgumentException("Book not found with id: " + id));  
  
  
 if (!book.getIsbn().equals(bookDetails.getIsbn()) &&  
 bookRepository.existsByIsbn(bookDetails.getIsbn())) {  
 throw new IllegalArgumentException("Book with ISBN " + bookDetails.getIsbn() + " already exists");  
 }  
  
 book.setTitle(bookDetails.getTitle());  
 book.setAuthor(bookDetails.getAuthor());  
 book.setIsbn(bookDetails.getIsbn());  
 book.setGenre(bookDetails.getGenre());  
 book.setPublicationYear(bookDetails.getPublicationYear());  
 book.setAvailable(bookDetails.getAvailable());  
  
 return bookRepository.save(book);  
 }  
  
   
 public void deleteBook(Long id) {  
 if (!bookRepository.existsById(id)) {  
 throw new IllegalArgumentException("Book not found with id: " + id);  
 }  
 bookRepository.deleteById(id);  
 }  
  
   
 @Transactional(readOnly = true)  
 public List<Book> searchBooksByTitle(String title) {  
 return bookRepository.findByTitleContainingIgnoreCase(title);  
 }  
  
  
 @Transactional(readOnly = true)  
 public List<Book> searchBooksByAuthor(String author) {  
 return bookRepository.findByAuthorContainingIgnoreCase(author);  
 }  
  
  
 @Transactional(readOnly = true)  
 public List<Book> searchBooksByGenre(String genre) {  
 return bookRepository.findByGenreContainingIgnoreCase(genre);  
 }  
  
   
 @Transactional(readOnly = true)  
 public List<Book> searchBooks(String keyword) {  
 return bookRepository.findByTitleOrAuthorContainingIgnoreCase(keyword);  
 }  
  
  
 @Transactional(readOnly = true)  
 public List<Book> getAvailableBooks() {  
 return bookRepository.findByAvailable(true);  
 }  
  
   
 @Transactional(readOnly = true)  
 public List<Book> getUnavailableBooks() {  
 return bookRepository.findByAvailable(false);  
 }  
  
  
 public Book updateBookAvailability(Long id, Boolean available) {  
 Book book = bookRepository.findById(id)  
 .orElseThrow(() -> new IllegalArgumentException("Book not found with id: " + id));  
  
 book.setAvailable(available);  
 return bookRepository.save(book);  
 }  
  
  
 @Transactional(readOnly = true)  
 public List<Book> getBooksByPublicationYear(Integer year) {  
 return bookRepository.findByPublicationYear(year);  
 }  
  
 @Transactional(readOnly = true)  
 public List<Book> getBooksPublishedAfter(Integer year) {  
 return bookRepository.findByPublicationYearGreaterThan(year);  
 }  
  
  
 @Transactional(readOnly = true)  
 public List<Book> getBooksPublishedBefore(Integer year) {  
 return bookRepository.findByPublicationYearLessThan(year);  
 }  
  
   
 @Transactional(readOnly = true)  
 public LibraryStats getLibraryStats() {  
 long totalBooks = bookRepository.countTotalBooks();  
 long availableBooks = bookRepository.countAvailableBooks();  
 long unavailableBooks = totalBooks - availableBooks;  
  
 return new LibraryStats(totalBooks, availableBooks, unavailableBooks);  
 }  
  
  
 public static class LibraryStats {  
 private final long totalBooks;  
 private final long availableBooks;  
 private final long unavailableBooks;  
  
 public LibraryStats(long totalBooks, long availableBooks, long unavailableBooks) {  
 this.totalBooks = totalBooks;  
 this.availableBooks = availableBooks;  
 this.unavailableBooks = unavailableBooks;  
 }  
  
 public long getTotalBooks() {  
 return totalBooks;  
 }  
  
 public long getAvailableBooks() {  
 return availableBooks;  
 }  
  
 public long getUnavailableBooks() {  
 return unavailableBooks;  
 }  
  
 @Override  
 public String toString() {  
 return "LibraryStats{" +  
 "totalBooks=" + totalBooks +  
 ", availableBooks=" + availableBooks +  
 ", unavailableBooks=" + unavailableBooks +  
 '}';  
 }  
 }  
}

**pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.7.0</version>  
 <relativePath/>  
 </parent>  
  
 <groupId>com.library</groupId>  
 <artifactId>library-management</artifactId>  
 <version>1.0.0</version>  
 <name>LibraryManagement</name>  
 <description>Spring Boot Library Management System</description>  
  
 <properties>  
 <java.version>8</java.version>  
 <maven.compiler.source>8</maven.compiler.source>  
 <maven.compiler.target>8</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Boot Starter Web -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- Spring Boot Starter Data JPA -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <!-- H2 Database -->  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <!-- Spring Boot Starter Validation -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-validation</artifactId>  
 </dependency>  
  
 <!-- Spring Boot Starter Test -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**DataLoader.java:**

package com.library.config;  
  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.stereotype.Component;  
  
@Component  
public class DataLoader implements CommandLineRunner {  
  
 private final BookRepository bookRepository;  
  
 @Autowired  
 public DataLoader(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 // Check if data already exists  
 if (bookRepository.count() == 0) {  
 loadSampleData();  
 }  
 }  
  
 private void loadSampleData() {  
 System.*out*.println("Loading sample data...");  
  
   
 Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "978-0-7432-7356-5");  
 book1.setGenre("Classic Literature");  
 book1.setPublicationYear(1925);  
 book1.setAvailable(true);  
  
 Book book2 = new Book("To Kill a Mockingbird", "Harper Lee", "978-0-06-112008-4");  
 book2.setGenre("Classic Literature");  
 book2.setPublicationYear(1960);  
 book2.setAvailable(true);  
  
 Book book3 = new Book("1984", "George Orwell", "978-0-452-28423-4");  
 book3.setGenre("Dystopian Fiction");  
 book3.setPublicationYear(1949);  
 book3.setAvailable(false);  
  
 Book book4 = new Book("Pride and Prejudice", "Jane Austen", "978-0-14-143951-8");  
 book4.setGenre("Romance");  
 book4.setPublicationYear(1813);  
 book4.setAvailable(true);  
  
 Book book5 = new Book("The Catcher in the Rye", "J.D. Salinger", "978-0-316-76948-0");  
 book5.setGenre("Coming-of-age Fiction");  
 book5.setPublicationYear(1951);  
 book5.setAvailable(true);  
  
 Book book6 = new Book("Java: The Complete Reference", "Herbert Schildt", "978-0-07-180855-9");  
 book6.setGenre("Programming");  
 book6.setPublicationYear(2020);  
 book6.setAvailable(true);  
  
 Book book7 = new Book("Spring Boot in Action", "Craig Walls", "978-1-617-29254-5");  
 book7.setGenre("Programming");  
 book7.setPublicationYear(2015);  
 book7.setAvailable(true);  
  
 Book book8 = new Book("Clean Code", "Robert C. Martin", "978-0-13-235088-4");  
 book8.setGenre("Programming");  
 book8.setPublicationYear(2008);  
 book8.setAvailable(false);  
  
 Book book9 = new Book("The Lord of the Rings", "J.R.R. Tolkien", "978-0-544-00341-5");  
 book9.setGenre("Fantasy");  
 book9.setPublicationYear(1954);  
 book9.setAvailable(true);  
  
 Book book10 = new Book("Harry Potter and the Philosopher's Stone", "J.K. Rowling", "978-0-7475-3269-9");  
 book10.setGenre("Fantasy");  
 book10.setPublicationYear(1997);  
 book10.setAvailable(true);  
  
 // Save all books  
 bookRepository.save(book1);  
 bookRepository.save(book2);  
 bookRepository.save(book3);  
 bookRepository.save(book4);  
 bookRepository.save(book5);  
 bookRepository.save(book6);  
 bookRepository.save(book7);  
 bookRepository.save(book8);  
 bookRepository.save(book9);  
 bookRepository.save(book10);  
  
 System.*out*.println("Sample data loaded successfully!");  
 System.*out*.println("Total books in library: " + bookRepository.count());  
 }}

**A screen shot of a computer screen

AI-generated content may be incorrect.Output**:

A screen shot of a computer screen

AI-generated content may be incorrect.