**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.

**BookRepository.java**

package com.library.repository;

import com.library.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Book.java**

package com.library.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Book {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String title;

    private String author;

    // Constructors

    public Book() {}

    public Book(String title, String author) {

        this.title = title;

        this.author = author;

    }

    // Getters and Setters

    public Long getId() { return 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; }

}

**BookController.java**

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

    @Autowired

    private BookRepository bookRepository;

    // Create a book

    @PostMapping

    public Book createBook(@RequestBody Book book) {

        return bookRepository.save(book);

    }

    // Get all books

    @GetMapping

    public List<Book> getAllBooks() {

        return bookRepository.findAll();

    }

    // Get a book by ID

    @GetMapping("/{id}")

    public Book getBookById(@PathVariable Long id) {

        return bookRepository.findById(id)

                .orElseThrow(() -> new RuntimeException("Book not found with id " + id));

    }

    // Update a book

    @PutMapping("/{id}")

    public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

        Book book = bookRepository.findById(id)

                .orElseThrow(() -> new RuntimeException("Book not found with id " + id));

        book.setTitle(bookDetails.getTitle());

        book.setAuthor(bookDetails.getAuthor());

        return bookRepository.save(book);

    }

    // Delete a book

    @DeleteMapping("/{id}")

    public void deleteBook(@PathVariable Long id) {

        bookRepository.deleteById(id);

    }

}

**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("LIBRARY MANAGEMENT APPLICATION STARTED...");

    }

}

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

       xmlns:context="http://www.springframework.org/schema/context"

       xmlns:aop="http://www.springframework.org/schema/aop"

       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:schemaLocation="

         http://www.springframework.org/schema/beans

         http://www.springframework.org/schema/beans/spring-beans.xsd

         http://www.springframework.org/schema/context

         http://www.springframework.org/schema/context/spring-context.xsd

         http://www.springframework.org/schema/aop

         http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- Enable component scanning -->

    <context:component-scan base-package="com.library" />

    <!-- Enable Spring AOP auto-proxy -->

    <aop:aspectj-autoproxy />

</beans>

**Application.properties**

# H2 Database configuration

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Show SQL in logs

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

# Defer running data.sql until after Hibernate creates tables

spring.jpa.defer-datasource-initialization=true

# Enable H2 console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**data.sql**

INSERT INTO book (title, author) VALUES ('Spring Boot in Action', 'Craig Walls');

INSERT INTO book (title, author) VALUES ('Effective Java', 'Joshua Bloch');

INSERT INTO book (title, author) VALUES ('Clean Code', 'Robert C. Martin');

**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>3.2.5</version>

        <relativePath/> <!-- lookup parent from repository -->

    </parent>

    <groupId>com.library</groupId>

    <artifactId>library-management</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>Library Management</name>

    <description>Spring Boot Library Management Application</description>

    <packaging>jar</packaging>

    <properties>

        <java.version>17</java.version> <!-- Change to 11 or 8 if needed -->

    </properties>

    <dependencies>

      <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-actuator</artifactId>

      </dependency>

        <!-- Spring Web for REST APIs -->

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>

        </dependency>

        <!-- Spring Data JPA for ORM -->

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-data-jpa</artifactId>

        </dependency>

        <!-- H2 Database (In-memory DB for testing) -->

        <dependency>

            <groupId>com.h2database</groupId>

            <artifactId>h2</artifactId>

            <scope>runtime</scope>

        </dependency>

        <!-- Spring Boot DevTools (Optional for hot reload) -->

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-devtools</artifactId>

            <scope>runtime</scope>

        </dependency>

        <!-- Spring Boot Testing -->

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-test</artifactId>

            <scope>test</scope>

        </dependency>

        <dependency>

    <groupId>junit</groupId>

    <artifactId>junit</artifactId>

    <version>4.13.2</version>

    <scope>test</scope>

</dependency>

    </dependencies>

    <build>

        <plugins>

            <!-- Maven Compiler Plugin -->

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>3.11.0</version>

                <configuration>

                    <source>${java.version}</source>

                    <target>${java.version}</target>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>

**Output:**

