**Case Study: Building a RESTful API for a Library Management System without Spring Boot**

**Requirements:**

* Develop a REST API to manage books.
* Allow basic CRUD operations (Create, Read, Update, Delete) on book data.
* Use traditional Spring without Boot.

**Steps:**

**1. Project Setup**

You’ll need to set up a Maven project with the following dependencies in your pom.xml:

xml

Copy code

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.27</version>

</dependency>

<!-- Spring Web (for REST API) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.3.27</version>

</dependency>

<!-- Jackson (for JSON handling) -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.15.2</version>

</dependency>

<!-- Servlet API -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

</dependency>

<!-- Logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</version>

</dependency>

</dependencies>

**2. Spring Configuration: XML and Annotations**

Since we are not using Spring Boot, we need to configure Spring manually, typically using XML for dependency injection and annotations for REST API configuration.

**web.xml (DispatcherServlet Configuration)**

This file is the entry point for the Spring Web MVC framework. It tells the application to use the DispatcherServlet.

xml

Copy code

<web-app>

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring-config.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

**spring-config.xml (Spring Configuration)**

This XML file will be used to configure the Spring beans (controllers, services, repositories, etc.).

xml

Copy code

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

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

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

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

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/mvc

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

<!-- Enable annotation-based configuration -->

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

<!-- Enable Spring MVC -->

<mvc:annotation-driven />

</beans>

**3. Model Class: Book**

The model will represent the data for a book in the library.

java

Copy code

package com.example.library.model;

public class Book {

private String id;

private String title;

private String author;

private double price;

// Getters and setters

public String getId() {

return id;

}

public void setId(String 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 double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

**4. Controller: BookController**

The controller handles incoming HTTP requests and returns responses.

java

Copy code

package com.example.library.controller;

import com.example.library.model.Book;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private static List<Book> bookList = new ArrayList<>();

static {

bookList.add(new Book("1", "Effective Java", "Joshua Bloch", 45.00));

bookList.add(new Book("2", "Clean Code", "Robert C. Martin", 40.00));

}

// Get all books

@GetMapping

public List<Book> getAllBooks() {

return bookList;

}

// Get a book by ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable("id") String id) {

return bookList.stream().filter(book -> book.getId().equals(id)).findFirst().orElse(null);

}

// Create a new book

@PostMapping

public String addBook(@RequestBody Book book) {

bookList.add(book);

return "Book added successfully!";

}

// Update a book

@PutMapping("/{id}")

public String updateBook(@PathVariable("id") String id, @RequestBody Book updatedBook) {

for (Book book : bookList) {

if (book.getId().equals(id)) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

return "Book updated successfully!";

}

}

return "Book not found!";

}

// Delete a book

@DeleteMapping("/{id}")

public String deleteBook(@PathVariable("id") String id) {

bookList.removeIf(book -> book.getId().equals(id));

return "Book deleted successfully!";

}

}

**5. Dispatcher Configuration**

This is needed to map your controllers properly and resolve views (though in a REST API, views are typically not needed).

java

Copy code

package com.example.library.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration

@EnableWebMvc

public class WebConfig implements WebMvcConfigurer {

@Bean

public InternalResourceViewResolver viewResolver() {

InternalResourceViewResolver resolver = new InternalResourceViewResolver();

resolver.setPrefix("/WEB-INF/views/");

resolver.setSuffix(".jsp");

return resolver;

}

}

**6. Running the Application**

You can deploy the application to any Java servlet container (like Tomcat or Jetty). Package the application as a WAR file using Maven and deploy it to your server.

In the absence of Spring Boot, the manual configuration allows more flexibility but requires additional effort for setup, such as setting up the web.xml, spring-config.xml, and ensuring the correct packaging.

This setup provides the basic foundation for creating a Spring Web REST API without Spring Boot. If you need additional features like database integration (using Spring Data or JPA), you can continue to build on this foundation by manually configuring additional Spring components.