**Course Management System**

**Objective** :

***In this exercise, you will create RESTful webservices using Spring Boot for HTTP GET request and also implement exception handling for a RESTful Web Services developed using Spring Boot..***

# 1.0 Functional Requirements

ZEE Tech Software solutions has already implemented the Spring REST solution for course management. Client has a new requirement to create a REST web service along with exception. Help Zee-Tech to automate the above process by developing Rest Service using Maven.

Your application should support the below services :

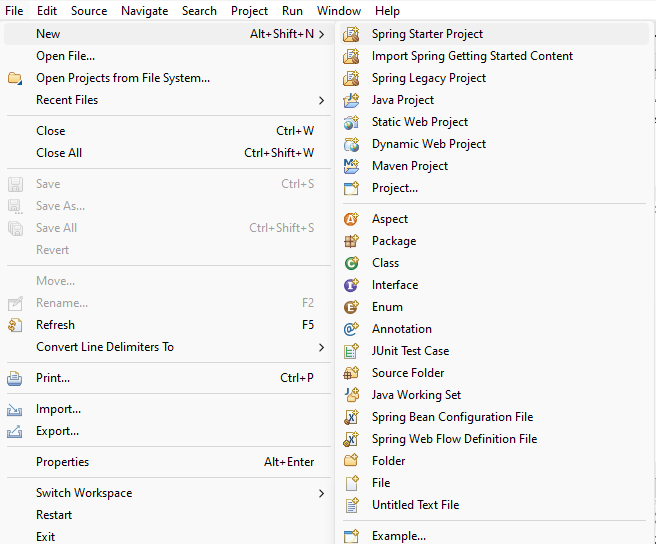
Get Request -->cms/find/101 : This service should retrieve the Course object for the given course id and also implement exception handling.

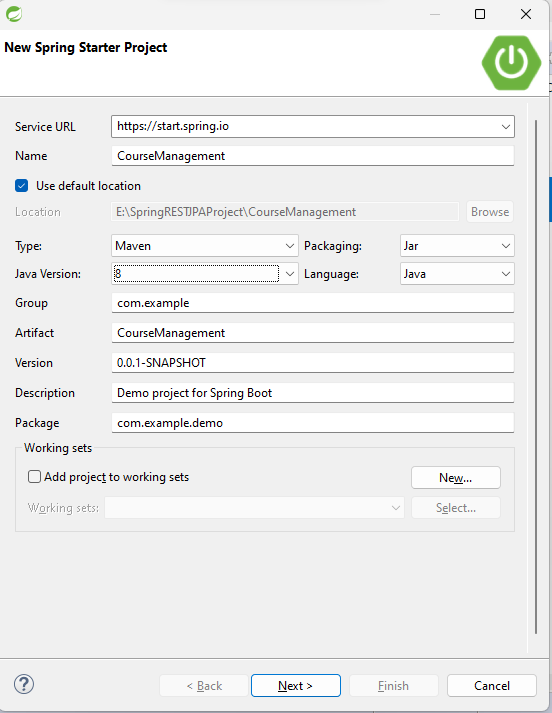
# 2.0 Technical Specifications

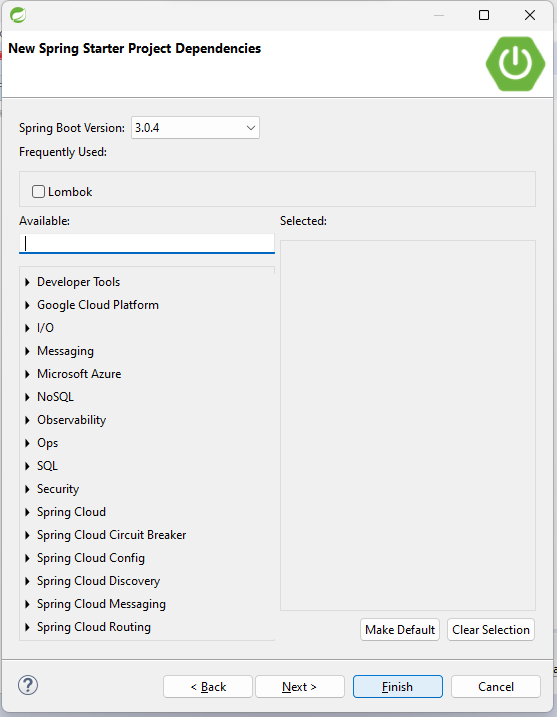
To start with this project, open STS, create a Spring Starter project and follow the instructions below.

1. Open IDE STS – Spring Tool Suite

2. Go to File ->Spring Starter Project

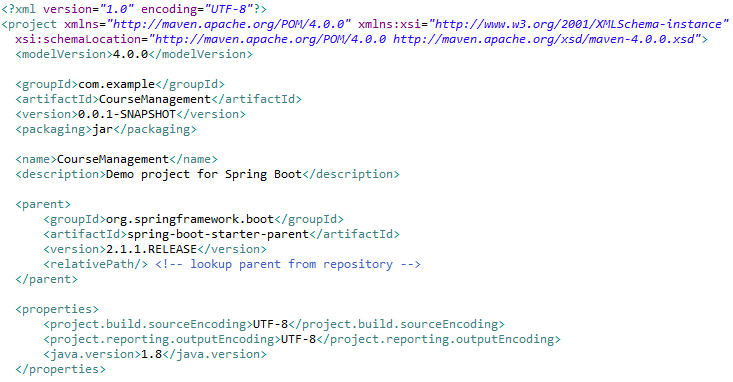




.

Click Finish.

3. Once the project is created, check / add the required dependency in pom.xml





4. Go to src/main/resources. Configure application.properties file

****

5. Go to src/main/java. Create packages com.model, com.service, com.exception and com.controller.

# Model class

You need to create a Model class, Course with attributes

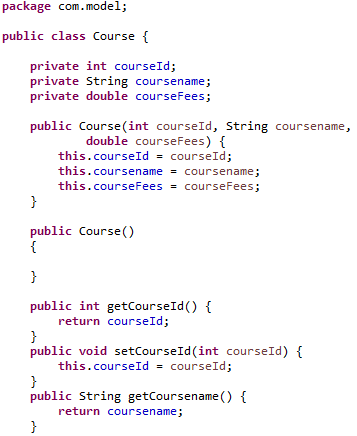
int courseId

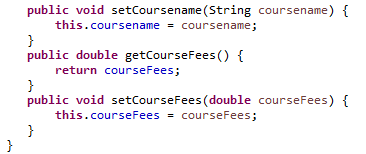
String coursename

double courseFees

Write public getters and setters, a no argument constructor and a parameterized constructor with parameters in the order courseId, coursename and courseFees.

6. Create the class Course in com.model package



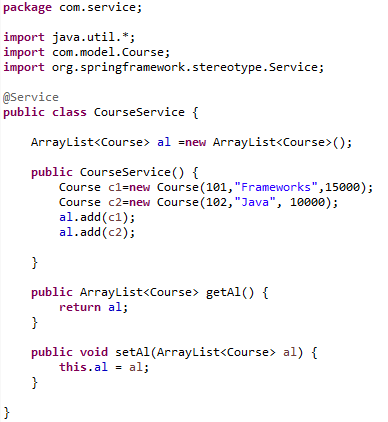


# Service class

Create a class CourseService in com.service package.

This class will have a list of Course objects as attribute. The methods in service should perform retrieval operations on Course objects in the list.

7. Create the CourseService class in com.service package with the attribute as shown below.

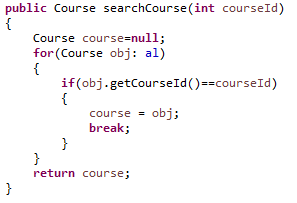


Note that we have written a constructor and added 2 Course objects into connectionList. This is optional.

You can do this, so that you can test your application easily in Postman Client.

**8. Implement the below methods in CourseService** **class**

* **public Course searchCourse(int courseId)** – This method takes a Course id as parameter. It should retrieve the course object for the given course id from the course list. If no object exists in that id then return null.

****

# Controller class

Create the REST API Controller in Spring Boot. For this, create a class CourseController in com.controller package.

CourseController class should act as the RestController, wherein the required service is to be created to perform the RETRIEVE operation.

This controller class should inject the CourseService class and invoke the methods in it.

The data returned from the Controller should be a JSON which is by default.

Any request to the service in this controller should start with ***/cms***

In the CourseController, create the service to view course by id.

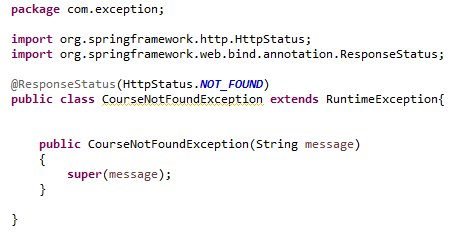
Get Request -->cms/find/101 : This service should retrieve the course object for the given course Id.

If the course for the given course is not found then a user defined exception , **CourseNotFoundException** with the message "**No such course id**" has to be thrown.

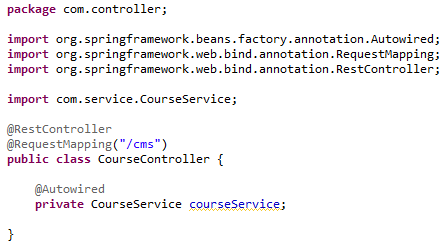
Also, instead of 500 error, the status code should be changed to 404. [**Hint**: Use the appropriate annotation above the CourseNotFoundException class with appropriate status code]

To do this create a class **CourseNotFoundException in com.exception** package. This class should inherit RuntimeException. Write a 1 argument constructor with String as argument.

9. Create the CourseNotFoundException class with the below code



10. Create the CourseController class with the below code



* **@RestController** makes the class as Spring Boot RESTful web service. It indicates that the data returned by each method in it will be written straight into the response body instead of rendering a template.

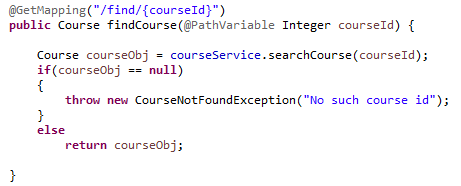
The class marked with @RestController performs CRUD operations and returns the output in JSON format to the user.

**@RequestMapping** specifies the base URI path for all request handling methods in this controller.

Also service is injected in controller.

11. Implement the below services in the CourseController class

* Get Request -->/find/1 – This service should invoke the searchCourse method in CourseService and return the Course object for the given course id.



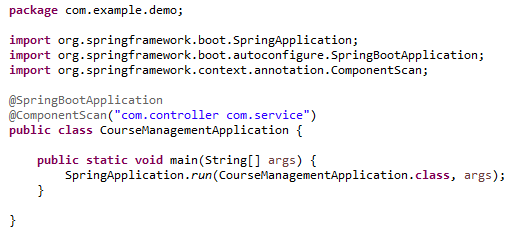
*@PathVariable*is used to handle template variables in the request URI mapping, and set them as method parameters.

**Create the Launch class for Spring Boot Application**

Every Spring Boot Application needs one launch class. This class is annotated with **@SpringBootApplication**.

12. In com.example.demo, you will have a class, CourseManagementApplication. Write the below code in this class.

Annotate this class with **@SpringBootApplication** and **@ComponentScan**. Call the static **method run of SpringApplication** class with the class name as parameter.



**@ComponentScan** indicates in which packages there are annotated classes which should be managed by Spring.

**@SpringBootApplication** annotation makes this class a configuration class.

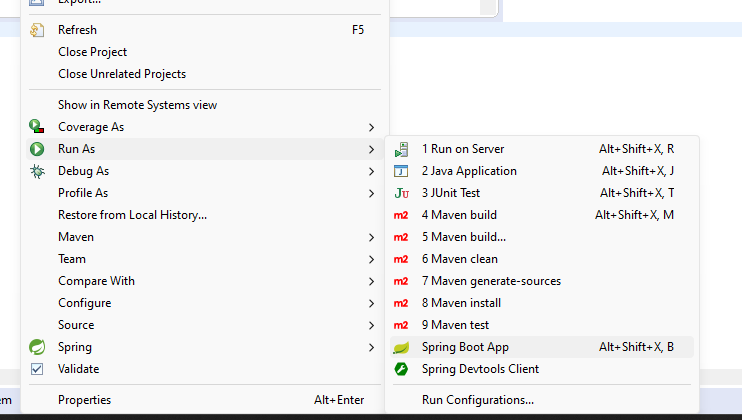
**SpringApplication** bootstraps and auto-configures our application. It also starts the embedded Tomcat server.

The class name is passed as an argument to run method to indicate that this is primary Spring Component for the project.

13. Comment the code in the java file CourseManagementApplicationTests available inside src/test/java folder.

14. Having completed the application, you can test the correctness by using Postman Client. To do this, Run the Application as Spring Boot App

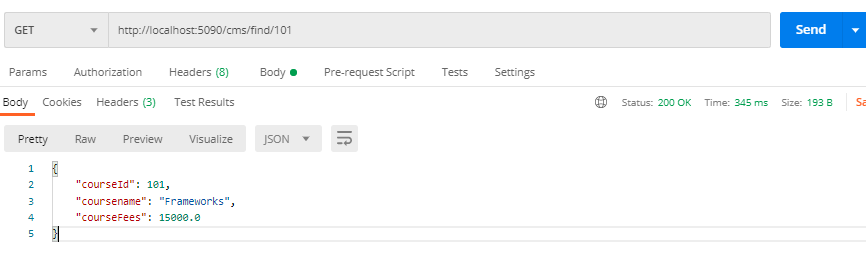
Right click the Project -> Run As -> Spring Boot App



You will see the below code in the console



15. Open Postman Client and check the application as shown below



3.0 Overall Design Constraints

When submitting the code to platform ensure the below points

1. **Check if the property name given in the application.properties files is same as given in the sample code. You can change the value if needed.**
2. **In the pom.xml ensure that you have provided only the dependencies provided in this sample. Don’t provide any additional dependency.**
3. **Use the service type and the service names as expected in the specification**
4. Adhere to the design specifications mentioned in the case study.
5. Ensure that you have provided all the classes / interface / attribute name / method name / return type / parameters as mentioned in the problem statement.
6. **Please make sure that your code does not have any compilation errors while submitting your case study solution.**

**Congratulations, you have successfully completed handling exception for a RESTful Web Services developed using Spring Boot. !**