**Online Zee-tech Mobile Connection**

**Objective** :

***In this exercise, you will create RESTful webservices using Spring Boot for HTTP POST request and GET request.***

# 1.0 Functional Requirements

Online Zee-tech Mobile Connection provider wanted to use a webservice for obtaining the new connection , modifying the connection type, revoke the connection and viewing the connection details. Help Zee-tech to automate the above process by developing Rest Service using Maven to add and retrieve the connection details.

Your application should support the below services :

Get Request -->/SPS/findConnectionById/1 : This service should retrieve the connection object for the given connection number.

Get Request -->/SPS/findAllConnection : This service should retrieve all the connections.

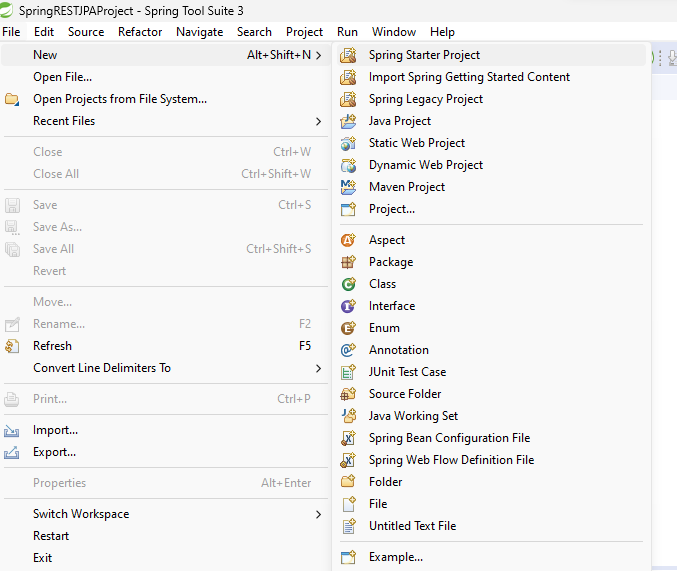
Post Request -->/SPS/addConnection: This service should add the connection object

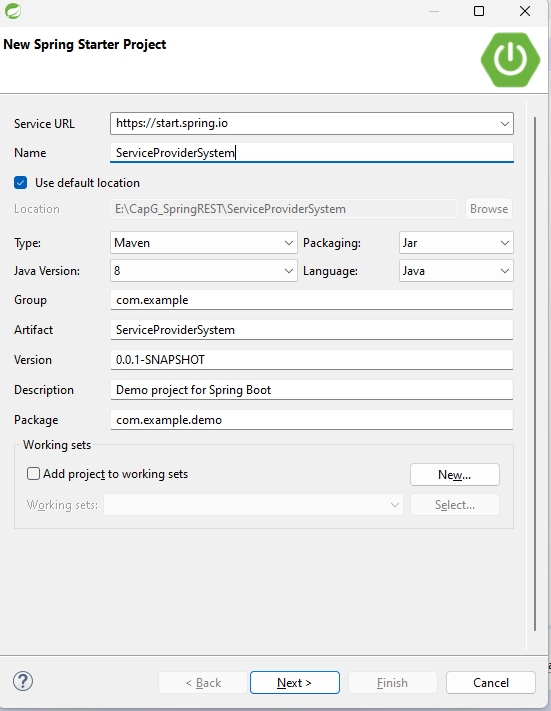
# 2.0 Technical Specifications

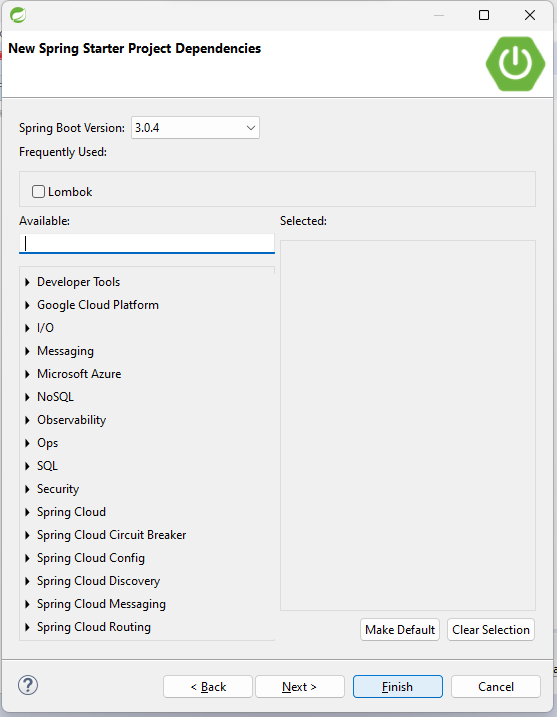
To start with this project, open STS, create a Spring Starter project and follow the instructions below.

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 and com.controller.

# Model class

You need to create a Model class, Connection with attributes

int connectionNum

String custName

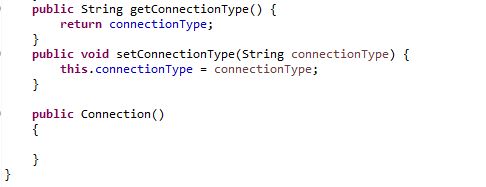
String planName

String connectionType

Write public getters and setters, a no argument constructor and a parameterized constructor with parameters in the order connectionNum, custName, planName and connectionType.

6. Create the class Connection in com.model package



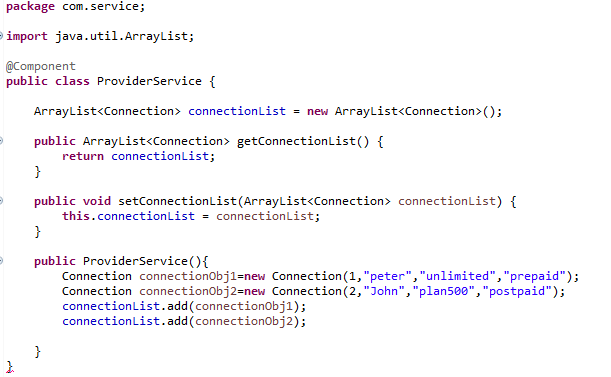


# Service class

Create a class ProviderService in com.service package.

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

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

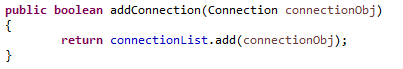


Note that we have written a constructor and added few Connection objects into connectionList. This is optional.

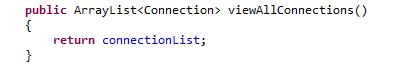
You can do this, so that you can test your application easily in Postman Client.

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

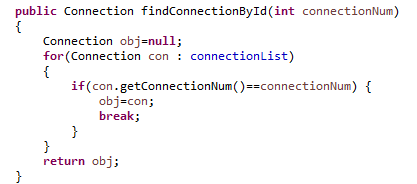
* **public boolean addConnection(Connection connectionObj)** – This method takes a Connection object as parameter. It should add the connection object to the connectionList.

****

* **public ArrayList<Connection> viewAllConnections()** – This method should retrieve all the Connection objects from the connectionList .



* **public Connection findConnectionById(int connectionNum)** – This method takes a connection number as parameter. It should retrieve the connection object for the given connection number. If no object exists in that number then return null.

****

# Controller class

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

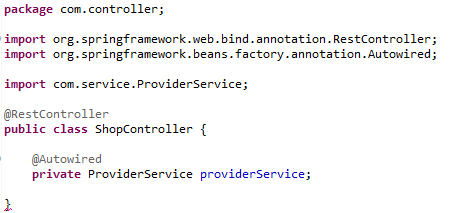
ShopController class should act as the RestController, wherein all the required services  are to be created to perform the ADD and RETRIEVE operations.

This controller class should inject the ProviderService 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 ***/SPS***

9. Create the ShopController 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.

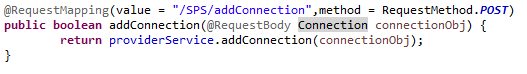
The class marked with @RestController performs CRUD operations and returns the output in JSON format to the user.

Also service is injected in controller.

10. Implement the below services in the ShopController class

* **Post Request -->/SPS/addConnection** – This service should add the Connection  object  by invoking the addConnection method in ProviderService.

To do this write the below code inside ShopController class



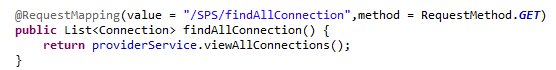
Also instead of



you can also use



* Get Request -->/SPS/findAllConnection – This service should invoke the invoke the viewAllConnections() in ProviderService and return all the connections..



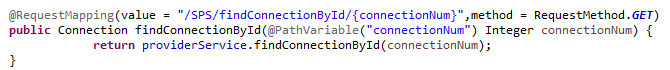
Instead of



you can also use



* **Get Request -->/SPS/findConnectionById/1** – This service should invoke the findConnectionById() method in ProviderService and return the connection object for the given connection number.



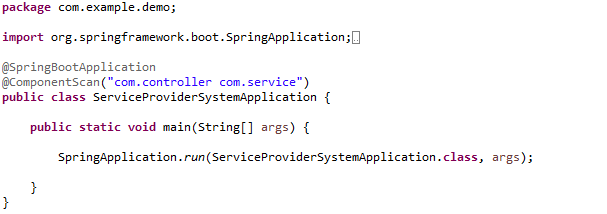
*@PathVariable*is used to handle template variables in the request URI mapping, and set them as method parameters.

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

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

11. In com.example.demo, you will have a class, ServiceProviderSystemApplication. 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.

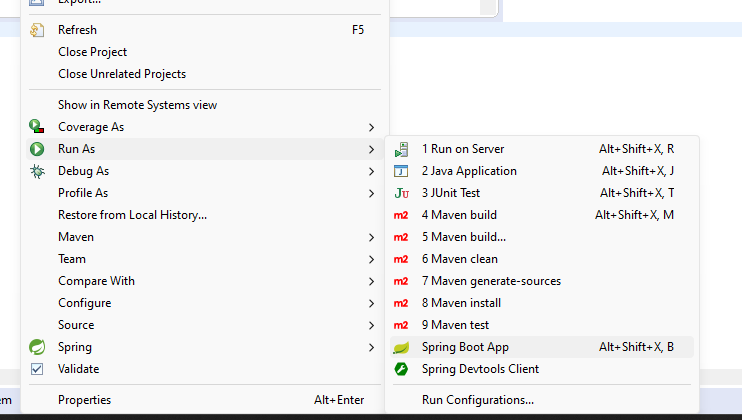
**@SpringBootApplication** annotation makes this class a configuration class.

**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.

12. 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

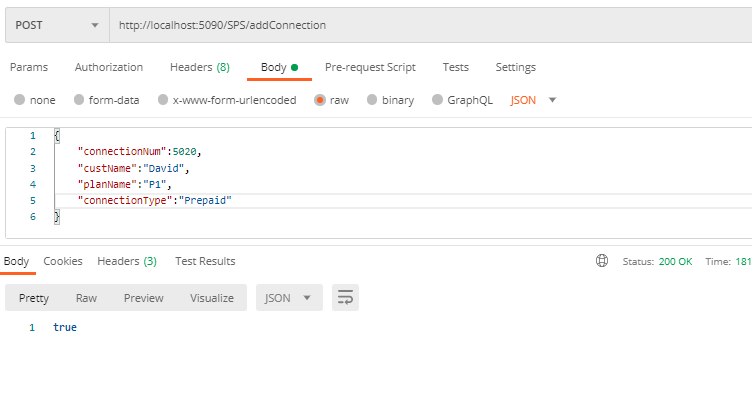


Comment the code in the java file ServiceProviderSystemApplicationTests available inside src/test/java folder.

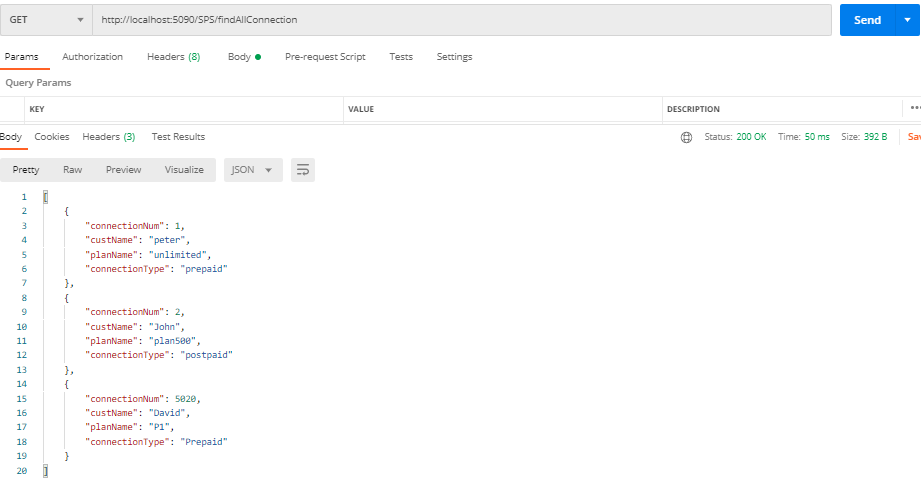
You will see the below code in the console



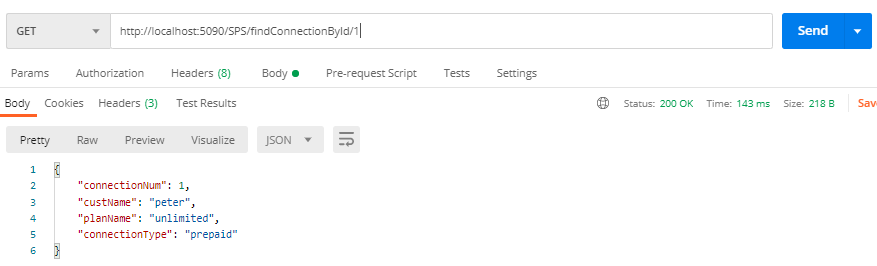
13. Open Postman Client and check the application as shown below



To check view all connection



To view connection by id



# 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 / attributename / methodname / 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 the exercise on building a Spring REST API to perform a service using HTTP Post and Get method !