Birzeit University
Faculty of Engineering and Technology
Electrical and Computer Engineering Department
Advance Computer Systems Engineering Lab ENCS515

EXP. No. 9. Spring Boot Part 1

1. Requirements

- ❖ Knowledge of the java programming language
- ❖ Basic Understanding of MVC (Model View Controller) architecture
- ❖ Basic Knowledge of Maven tool
- ❖ Basic Idea about Spring framework (preferably Spring MVC)
- ❖ Spring Tool Suite software (STS) (For windows 64-bit download this Link)
- ❖ Postman software (For windows 64-bit download this Link)

2. Objectives

- ❖ Understanding the fundamentals of Spring Boot framework
- ❖ Create rest services using Spring MVC

3. Introduction

Spring is an enterprise java framework which lets you write enterprise java applications, spring framework is widely used due to It's useful features, spring strongly introduces the concept of inversion of control specifically dependency injection which helps you wire your associations in a way which reduces the dependency between the entities.

Spring boot was introduced to make creating spring applications easier, spring boot was built above Spring MVC, Spring MVC is a java MVC framework which helps you build MVC architecture using defined features, It wires up all the MVC components in a nice way which helps developers develop flowless and easily tested scenarios.

Spring Boot makes it easy to create stand-alone, production-grade based Applications that you can simply run without even the need of servlet container, also Spring Boot contains default configuration which probably will be enough for most of the cases which makes it easy to start development without the need to diving into complex configurations (as in Spring , Spring MVC), also being stand-alone with embedded servlet container means the configuration of the server are the embedded with the application configurations, which makes it easier to deploy in different machines without worrying about configuring the servlet container.

Keep in mind that spring boot mainly bootstrap the development of spring MVC projects so actually spring MVC will be the one getting the job done.

4. Procedure

4.1. Downloading and running the STS software

After downloading the STS software, extract the downloaded file as in Figure 9.1, and run the software in

spring-tool-suite-3.9.8.RELEASE-e4.11.0-win32-x86_64>sts-bundle>sts-3.9.8.RELEASE>STS.exe

as shown in Figure 9.2. the software panels are shown as in Figure 9.3.

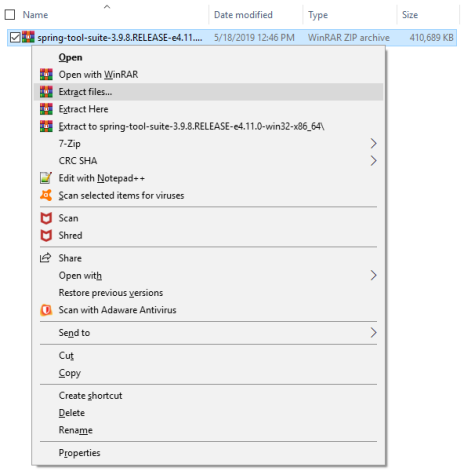

Figure 9.1 Extracting STS software

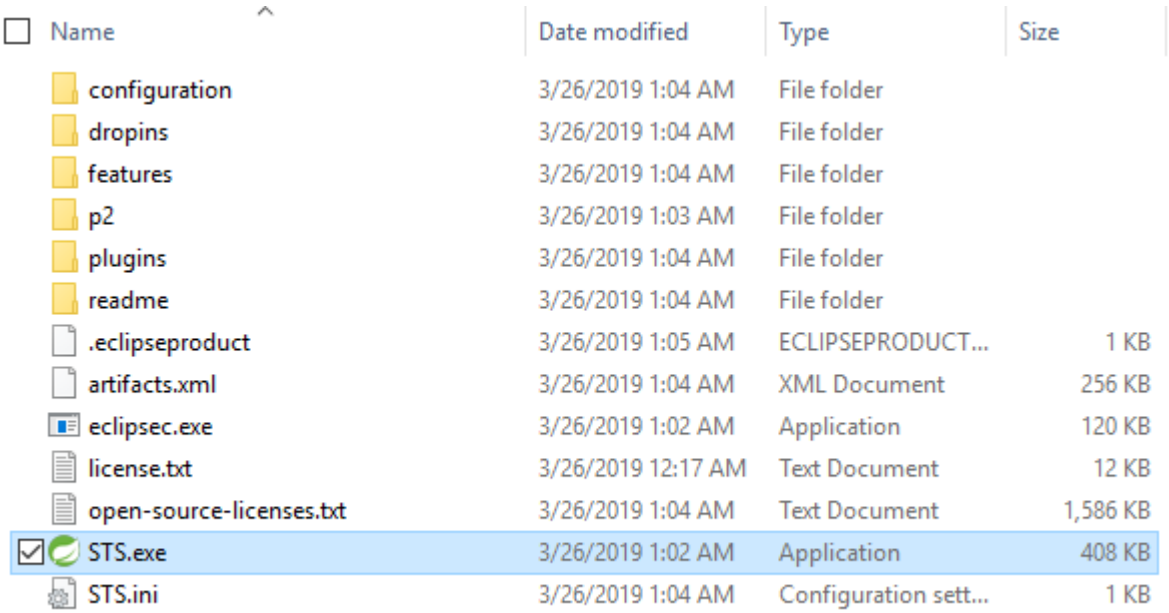

Figure 9.2 STS software

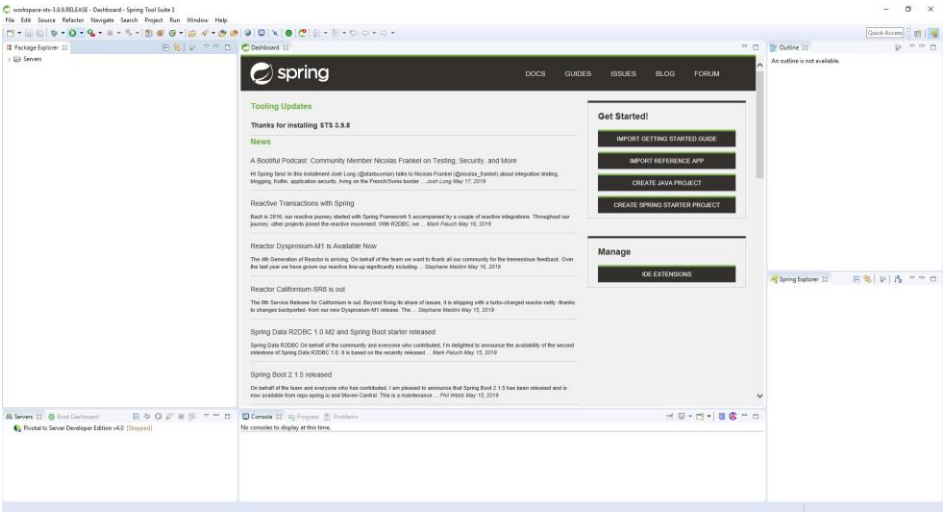

Figure 9.3 STS software panels screen

4.2. Setup Spring Boot Project

Spring boot aims to bootstrap the development of spring MVC application, spring project suffers of complex configurations which spring boot solves in a very nice way. There are many ways to create a spring boot project, we will introduce one way but keep in mind this is not the only way and not the best, it depends on the taste of developer and the environment you are working on.

- STS (Spring Tool Suite):

  You can use spring tool suite to create spring boot application in simple steps.

  - Creating new Spring Boot starter Project as shown in Figure 9.4.a and Figure 9.4.b.
  - After choosing Spring Boot starter Project a new screen as shown in Figure 9.5 will appear. Change the name of the application to “FirstApplication”. Make sure all other attributes are as shown in Figure 9.5.a The click next
  - Search for web in the search slot and add the web dependency for the application as shown in Figure 9.5.b. Then click finish.

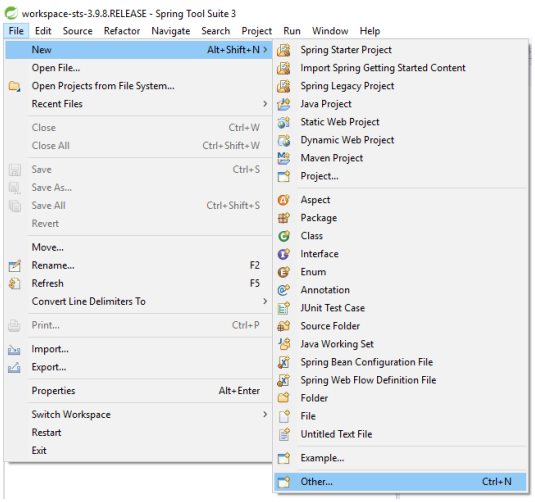

Figure 9.4.a New Other Project

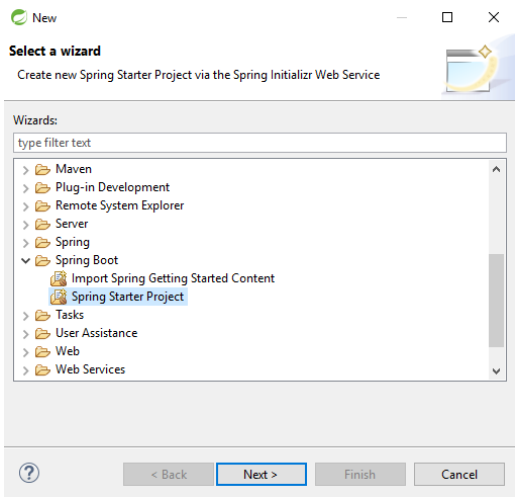

Figure 9.4.b New Spring Starter Project

Figure 9.4 Creating New SpringBoot Project

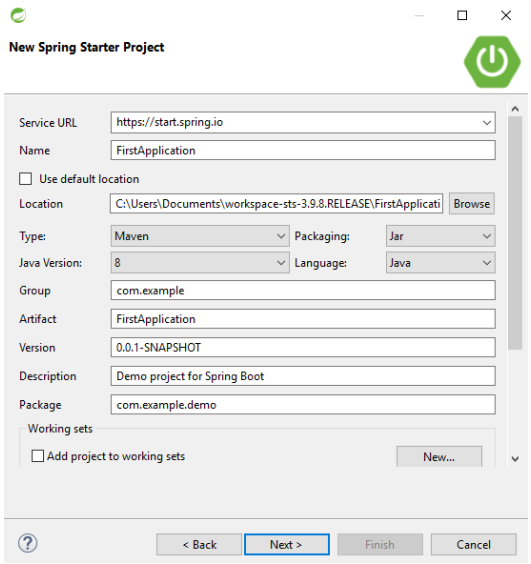

Figure 9.5.a Application name and other attributes

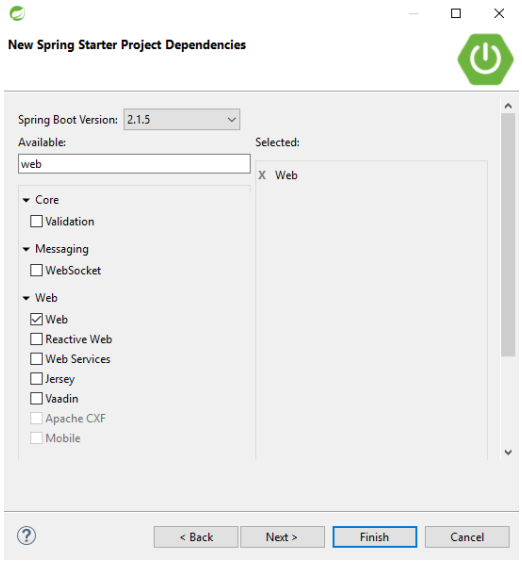

Figure 9.5.b Application dependencies

Figure 9.5 New Spring Starter Project Screen

Task: Do your research to figure out the purpose of the Group, Artifact, and version.

4.3. Create Project Structure

As we mentioned before spring boot is just an upper framework built above Spring MVC so spring MVC will do all the work, to start building our application we need to build a structure for our MVC components, there are many structures and trends to build MVC application, in this walk through we are aiming to build RESTful API's without worrying about the views.

REST (Representational state transfer): briefly is a standard of communication between client and server using textual representation of the resources using http stateless (each request does not know about the other request) requests, resources are presented using text wrapped inside http message, each resource is defined by URL and operations are defined through sub URL's and https methods **(Post for create new, Put for update, Get for getting resources, Delete for delete resources and others -search for them-)**. In Spring MVC you will have multi layers as:

- Controllers will play the role of the rest api's, business logic will not be included in controllers instead it will be wrapped in another layer called business services to keep the controllers layer as thin as possible, controllers will be mapped by URL's to access methods inside the controller.
- Services are the layer which contains most of the business logic, services will be injected inside controllers to be used when URL's mapped by controllers are accessed, services are by default singleton, which means whenever trying to instantiate the service you will get the same object, this is useful as the transfer of state between all the components is guaranteed, though it introduces a risk with multithreading systems which requires using thread safe operations.
- Models layer will contain our entities which represents the resources in our system.

You will create for packages under the demo package calling them (Controllers, Services and Models as shown in Figure 9.6.

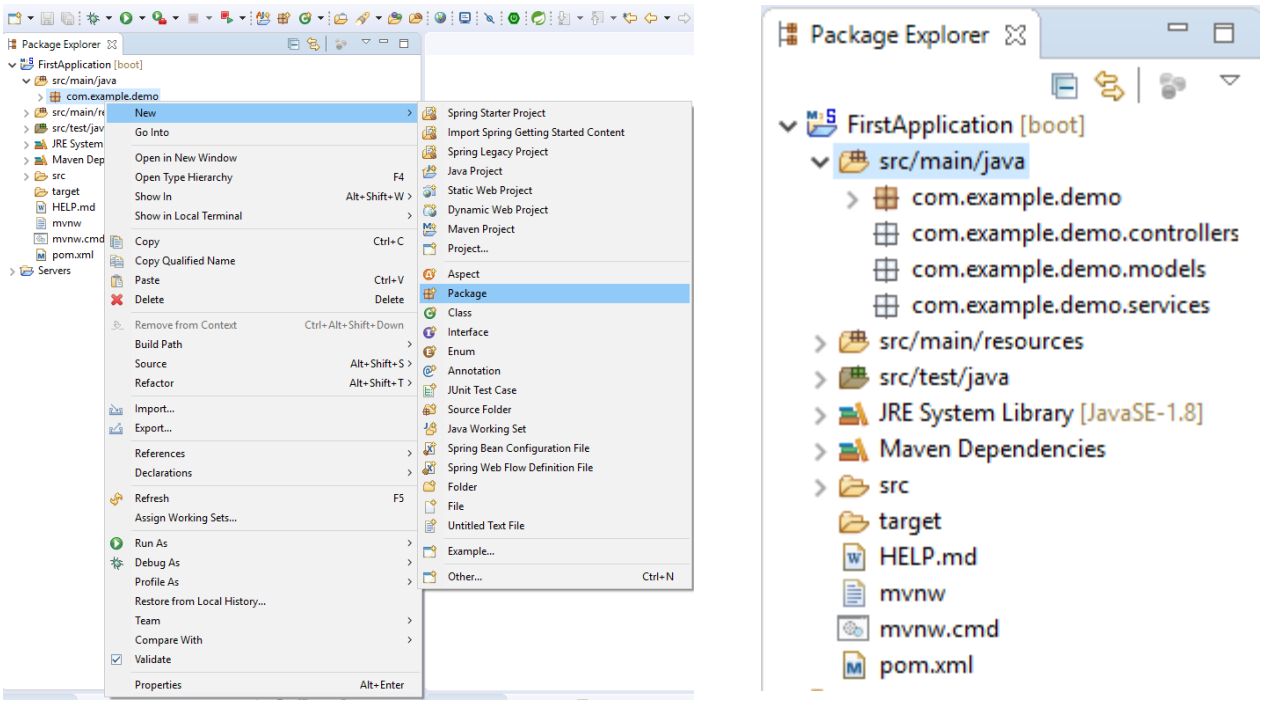

Figure 9.6 Adding Project Packages

- Models

Models are the entities of our application, in this walk through we will be implementing a User entity which has a name, a username and an email, models in spring boot basically are normal classes, so in models package go ahead and create a class User and add its attributes. Notice that all attributes should be private and should be accessed using accessors (getters and setters) to follow encapsulation standards.

```
package com.example.demo.models;

public class User {

	private Integer id;
	private String userName;

	public User() {
		super();
		// TODO Auto-generated constructor stub
	}

	public User(Integer id, String userName) {
		super();
		this.id = id;
		this.userName = userName;
	}

	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
	public String getUserName() {
		return userName;
	}
	public void setUserName(String userName) {
		this.userName = userName;
	}
	@Override
	public String toString() {
		return "User [id=" + id + ", userName=" + userName + "]";
	}

}
```

- Services

Services will take care of most of the job, they will be called by controllers and used to get or manipulate data, in services package create UserService class.

You can notify spring about service using "@Service" annotation, remember that spring will take care of initializing all predefined components so you will not need to initialize service manually or apply singleton constraints.

For now, let us create a static list containing users' data (later we will get this data from database), then create a method which returns this list. In addition, we will create a say hello method.

```
package com.example.demo.services;

import java.util.ArrayList;
import java.util.Arrays;
import org.springframework.stereotype.Service;
import com.example.demo.models.User;

@Service
public class UserService {

	private ArrayList<User> userList = new
ArrayList<User>(Arrays.asList(
				new User(1, "Ayham Hashesh"),
				new User(2, "Rajaie Imseeh")
				));

	public ArrayList<User> getUserList() {
		return this.userList;
	}

	public String sayHello() {
		return "hello";
	}

}
```

- Controllers

Controllers are the API's access when requesting a URL, to create a controller in packages controllers create a class UserController. To tell spring that your class is a controller you need to annotate it with "@RestController" annotation, when application starts spring will register classes annotated with this annotation as controller, this annotation in addition to define a controller tells spring that this controller is following rest standard which means a textual response will be returned instead of object. To direct the request made to a URL you need to map each URL to a method in a controller, this is done using annotation "@RequestMapping", by providing the URL to this annotation method will be executed and the returned object/List of objects will be parsed (by default) to JSON using a built in library (JACKSON).

In a normal way controllers use services so we need a reference for our user service in the controller, on the startup of the application spring will initialize the singleton service bean (if you do not know what beans are please do some reading about spring beans) so we do not need to initialize service, we can wire the service to our controller using "@Autowired" annotation which wires the service bean to the reference defined in our controller.

```
package com.example.demo.controllers;

import java.util.ArrayList;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.models.User;
import com.example.demo.services.UserService;

@RestController
public class UserController {
	@Autowired
	UserService userService;

	@RequestMapping("/hello")
	public String sayHello() {
		return userService.sayHello();
	}

	@RequestMapping("/users")
	public ArrayList<User> getAllUsers() {
		return userService.getUserList();
	}

}
```

The project packages and classes should look like as in Figure 9.7

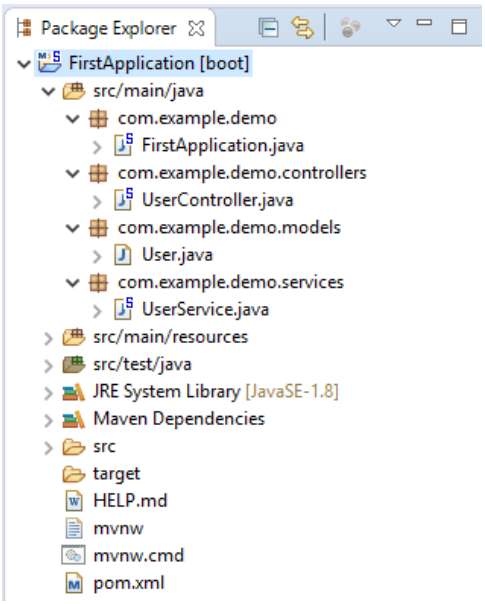

Figure 9.7 project packages and classes

Run the application by clicking right click on the main project package and choose "Run as" → "Spring Boot App" as shown in Figure 9.8.

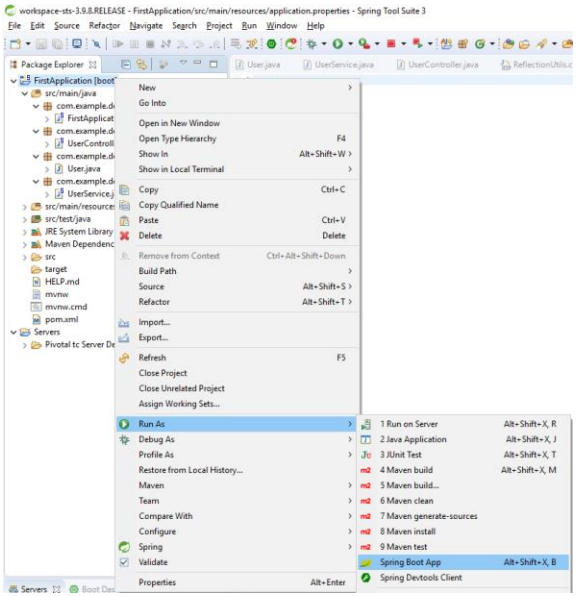

Figure 9.8 Run Spring Boot Application

When requesting localhost:8080/hello on any web browser this API will return hello (String will not be parsed to JSON for sure) as shown in Figure 9.9.

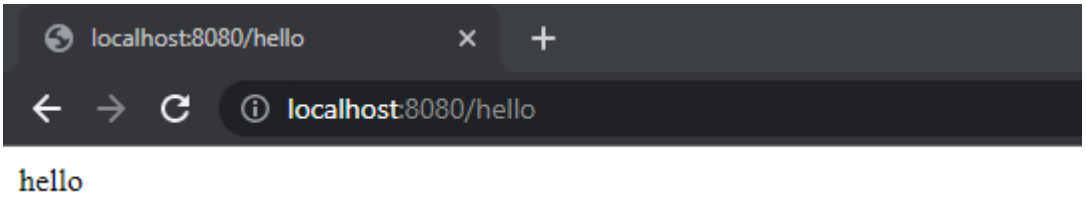

Figure 9.9 Web Browser request

Try to access localhost:8080/users and you will get a JSON array representing all the users as shown in Figure 9.10.

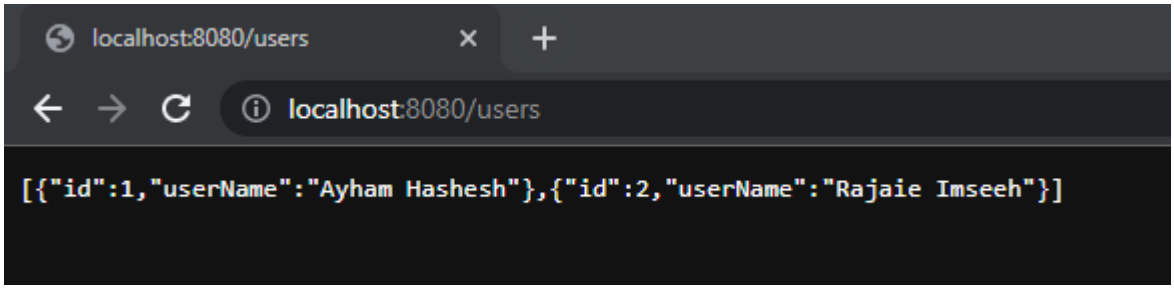

Figure 9.10 Web Browser Request for all users

This request is GET request, without specifying the HTTP method of a specific URL all requests will come to our getAllUsers() method, you can try to do a POST request using Postman Extension in chrome and you will get the same response.

To Specify a different method for POST, PUT you can pass a parameter to the "@RequestMapping" annotation defining the method using a predefined enumeration in spring.

```
@RequestMapping(method=RequestMethod.POST, value="/users")
public boolean addUser(@RequestBody User user) {

}
```

URL is passed use value keyword, to receive the object to be created you need to get it using "@RequestBody", this annotation will tell spring to map the payload JSON included in the request body to the Entity User and create object user.

To add the user to the array list which is in the UserService we create a method that adds a user to the arraylist in the UserService class.

```
public boolean addUserToUserList (User user) {
        return userList.add(user);
}
```

Then in the UserController class in tha addUser method we call this method that adds the user to the array list.

```
@RequestMapping(method=RequestMethod.POST, value="/users")
public boolean addUser(@RequestBody User user) {
        return userService.addUserToUserList(user);
}
```

After running the application as mentioned above we can test the application using the Postman software.

- Create a get method and add the "http://localhost:8080/users" URL to this request. This is for getting all users since it is a get method

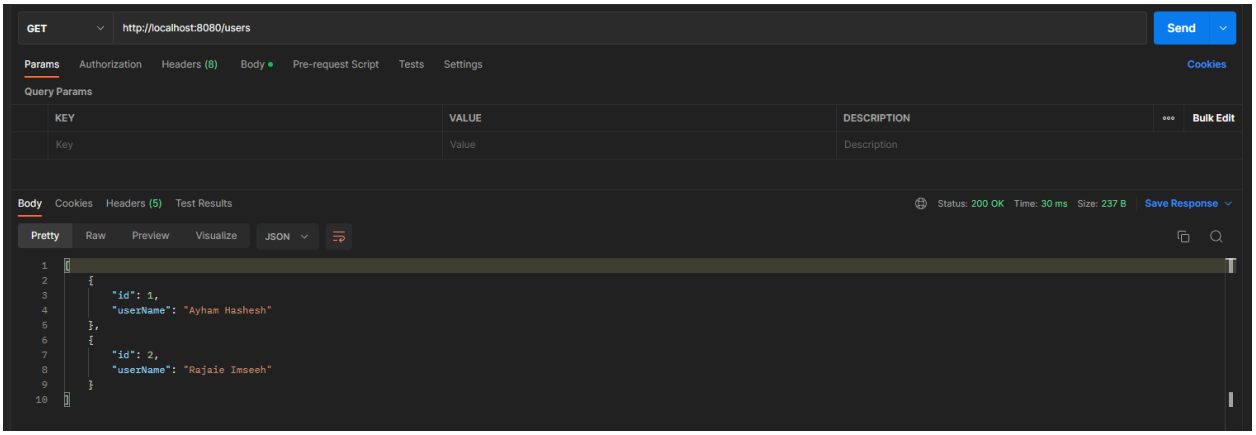

Figure 9.11 Postman Request for all users

- Create a Post Method that will add a user to the ArrayList using PostMan software, the user to be inserted will be in the body of the request. So click on the body button in the Postman Software and choose raw, change the input from text to JSON (application/json) and insert the new user we want to add as a JSON object, this is shown in Figure 9.12.

```
{
        "id": 3,
        "userName": "New User"
}
```

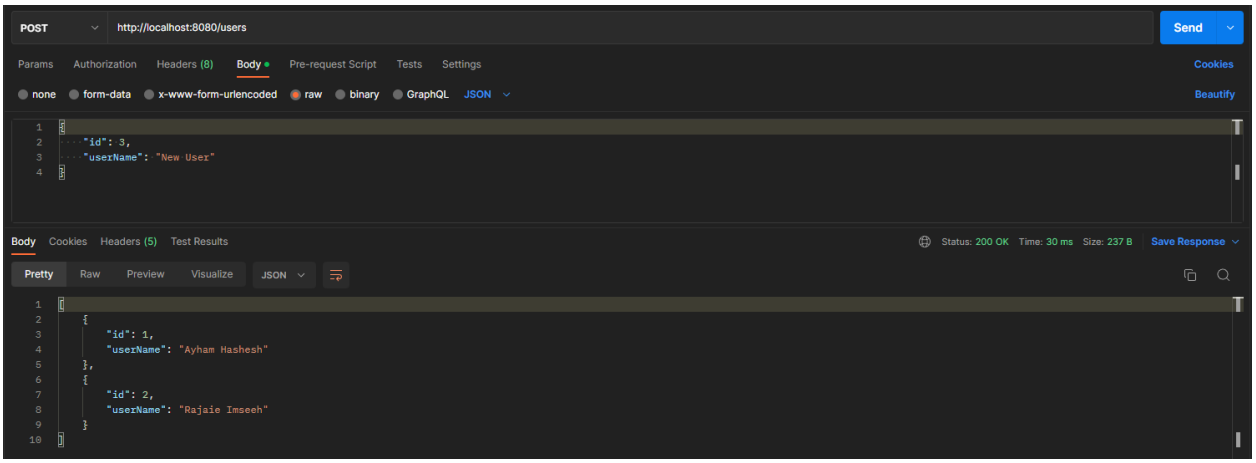

Figure 9.12 POST Method to Add User

After we click on Send a true value must be returned indicating that the user was added successfully as shown in. We can test if the user is added by calling the get method for getting all users as shown in Figure 9.14.

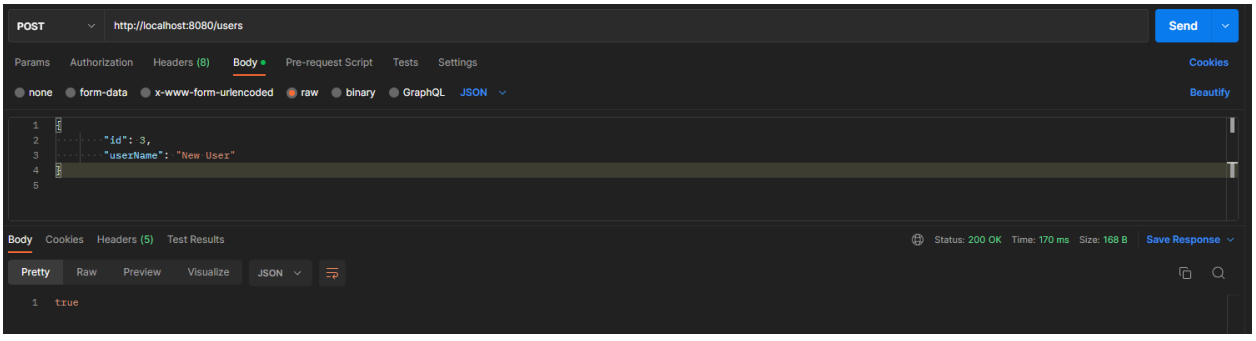

Figure 9.13 User was Added Successfully

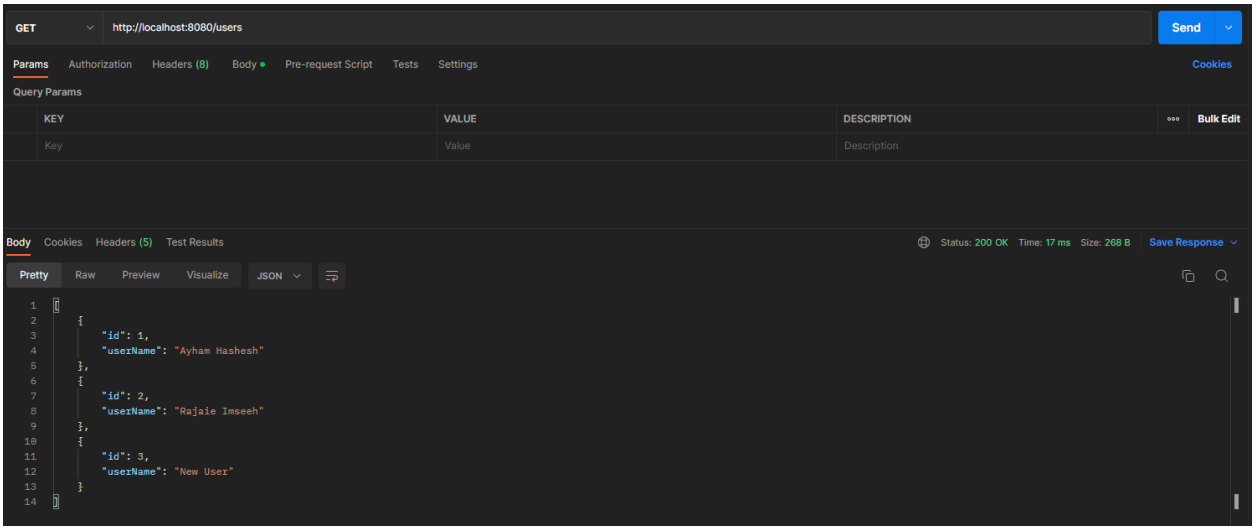

Figure 9.14 Get All Users with the New User

When delete a resource we usually specify the identifier of the resource to be deleted in the URL, for example if we want to remove an object and its identifier is "test" then the URL will be "localhost:8080/users/test" and the method will be DELETE, In our controller we can specify mapping for "users/test" but this will not be practical as we will need a mapping for each object we have, instead we can tell spring that "test" is a variable using "@PathVariable" annotation, value between {} will be stored in name variable.

```
@RequestMapping(method=RequestMethod.DELETE,value="/users/{id}")
        public boolean deleteUser(@PathVariable Integer id) {
        // Delete with id from the User Service Array List
}
```

To do a PUT (update) we usually send the updated object in the payload JSON and the identifier of the object as a path variable

```
@RequestMapping(method = RequestMethod.PUT, value="/users/{id}")
    public boolean updateUser(@RequestBody User user,@PathVariable
    Integer id) {
         // Update with id in the User Service Array List
    }
```

So for now we walked through the basic APIs in spring MVC and we saw how easily we built the application using spring boot, for now we can manipulate in memory data, data is not persisted yet which means if you restart the application all updated on data will be removed, in Next experiment we will walk through spring data JPA which is a framework to support persisting our java object in database so we will have a serialized data, also we will take a look at deployment the spring boot application using the built in servlet container and using an external servlet container (Tomcat).

5. Todo

This part will be given to you by the teacher assistant in the lab time.