

```
# Java Spring-Course-Project
```

Project is about starting a simple spring application.

****Project Steps:****

1. Go to web and search Spring boot start web maven central
2. Select latest

Java Spring-Course-Project

This project is about starting a simple spring application.

Project Steps:

1. Go to web and search Spring boot start web maven central

The screenshot shows the Maven Repository website. The top navigation bar includes the 'MVN REPOSITORY' logo and a search bar. The left sidebar contains a graph of 'Indexed Artifacts (39.8M)' and a list of 'Popular Categories' such as 'Testing Frameworks & Tools', 'Android Packages', 'Logging Frameworks', 'Java Specifications', 'JVM Languages', 'JSON Libraries', 'Language Runtime', 'Core Utilities', 'Mocking', 'Web Assets', and 'Annotation Libraries'. The main content area displays the details for the 'Spring Boot Starter Web' artifact. It includes a description: 'Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container'. Below this, there are tabs for 'License' (Apache 2.0), 'Categories' (Web Frameworks), 'Tags' (spring, framework, web, starter), 'Ranking' (#49 in MvnRepository, #1 in Web Frameworks), and 'Used By' (12,963 artifacts). A table lists various repositories and their counts: Central (225), Spring Milestones (93), Redhat GA (1), Alfresco (1), Evolveum (1), Gradle Releases (1), Grails Core (2), Kylogence Public (2), and ICM (8). At the bottom, a table shows the version history for the artifact, with columns for Version, Vulnerabilities, Repository, Usages, and Date.

Version	Vulnerabilities	Repository	Usages	Date
3.3.2		Central	118	Jul 18, 2024
3.3.x		Central	540	Jun 20, 2024
3.3.0		Central	692	May 23, 2024
3.3.0		Central	74	Jul 18, 2024

2. Click on the latest version, then copy the Gradle text.

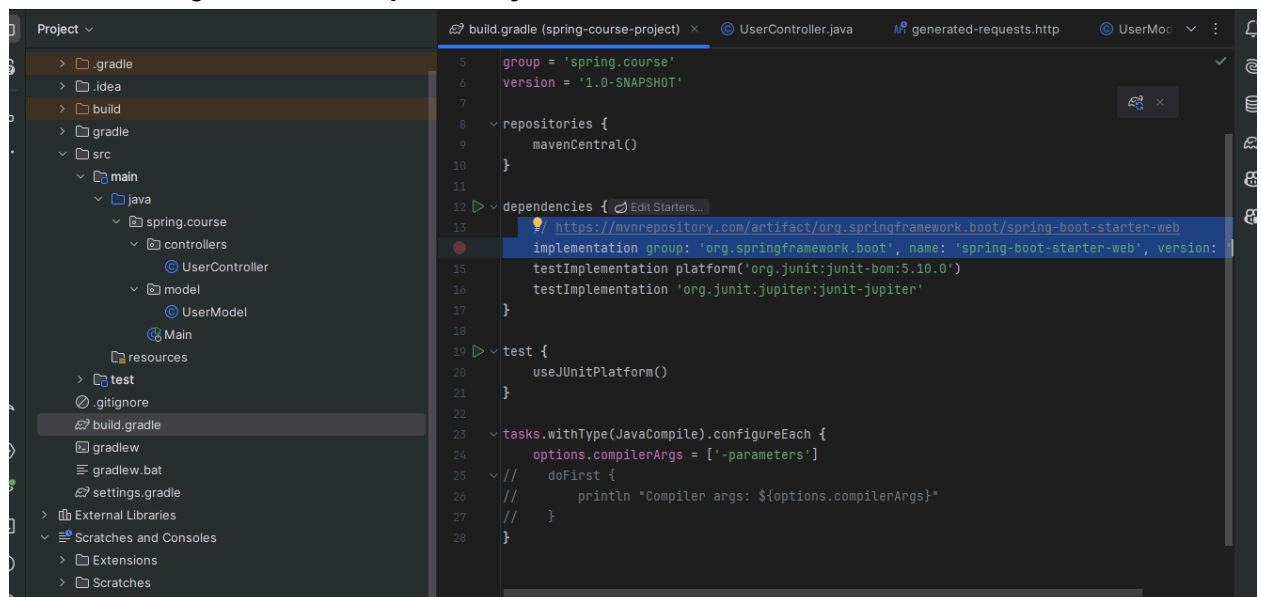
Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container

License	Apache 2.0
Categories	Web Frameworks
Tags	spring framework web starter
Organization	VMware, Inc.
HomePage	https://spring.io/projects/spring-boot
Date	Jul 18, 2024
Files	pom (2 KB) jar (4 KB) View All
Repositories	Central
Ranking	#49 in MvnRepository (See Top Artifacts) #1 in Web Frameworks
Used By	12,963 artifacts

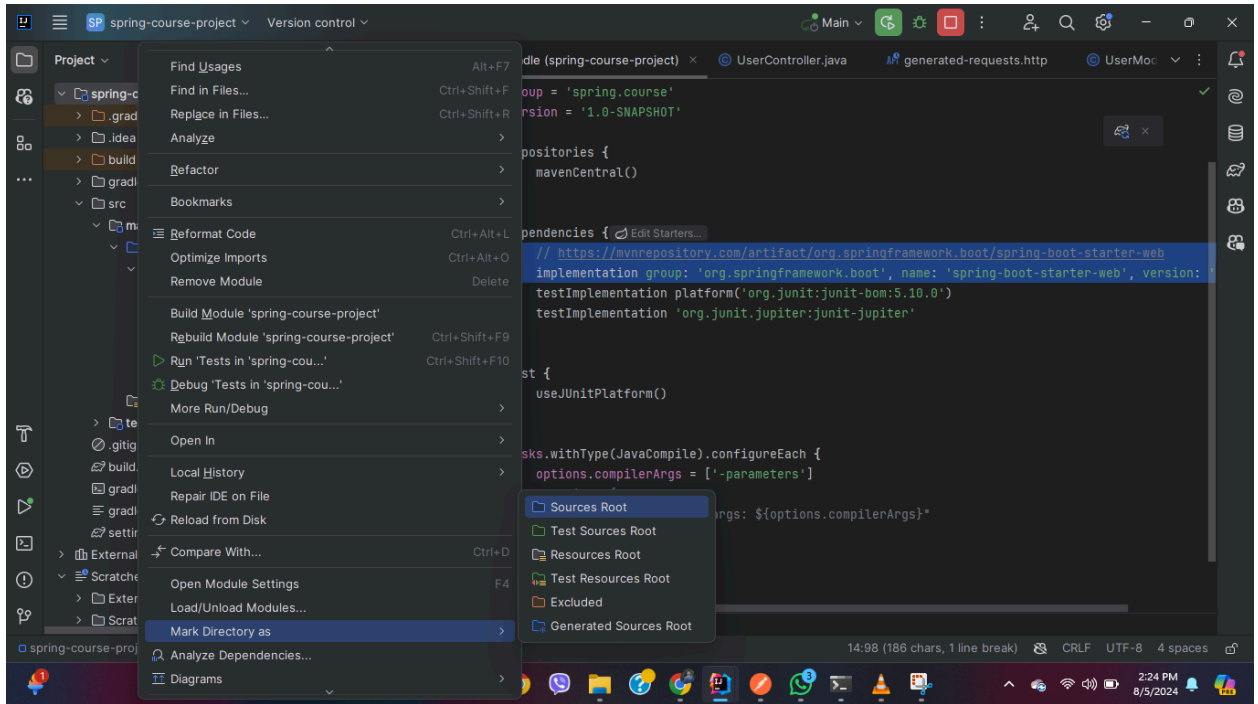
Maven
Gradle
Gradle (Short)
Gradle (Kotlin)
SBT
Ivy
Grape
Leiningen
Buildr

```
// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web
implementation group: 'org.springframework.boot', name: 'spring-boot-starter-web', version: '3.3.2'
```

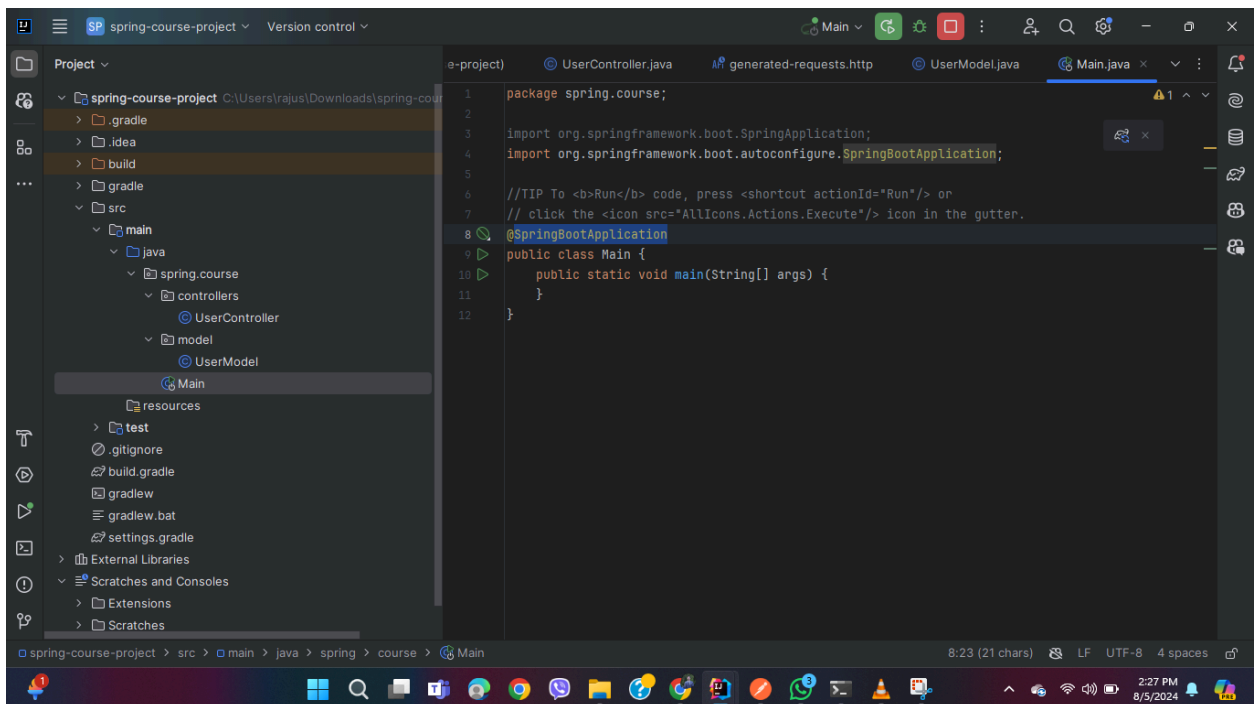
3. Paste in build.gradle file in **dependency** section:



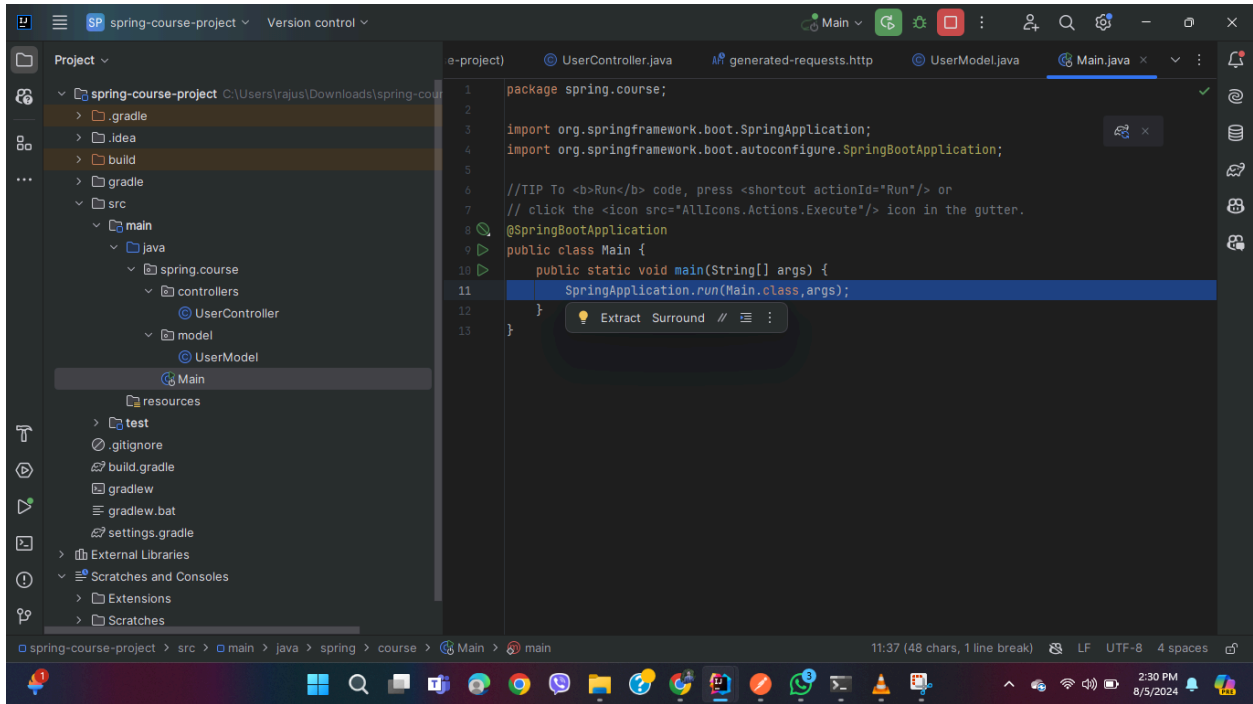
4. Mark the directory as source root



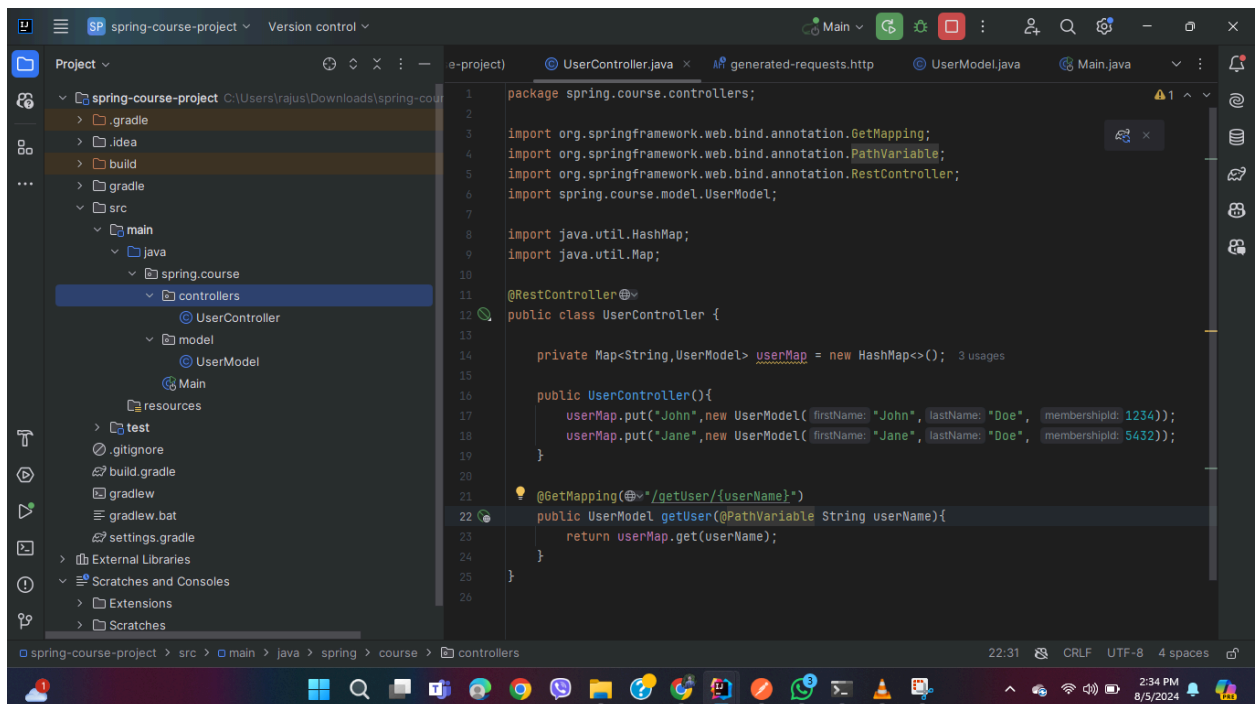
- Go to main.java and write this annotation: `@SpringBootApplication`. -> It is like an entry point for spring boot app.



- Go inside body of main class and write `SpringApplication.run(Main.class,args);` -> It starts the application.



7. Create a package called controller and create class called **UserController**.
When clients are requesting for data in the app, it requests hit the controller first.



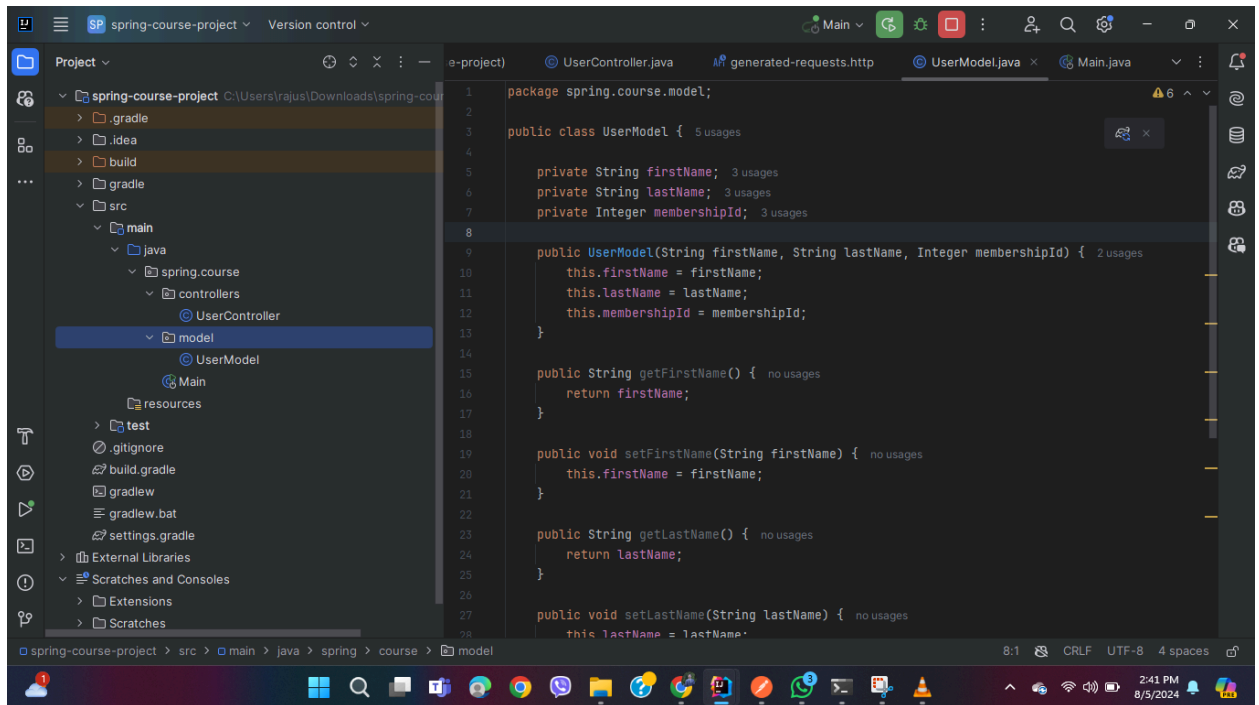
@RestController annotation should be used first as done before the **UserController** constructor.

@GetMapping annotation is used for get operations.

The `@GetMapping("/getuser/{userName}")` gives us the format of the api requests the client sends. Here

The client can send get requests like `localhost:8080/getuser/John` and we can get answer accordingly.

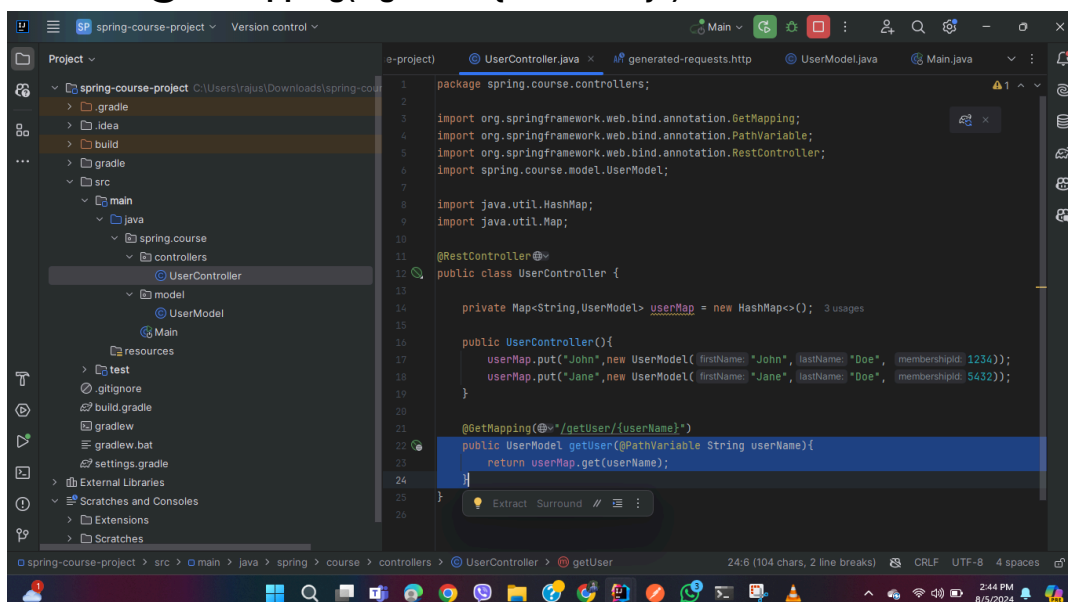
8. Create a new package called model and create a class called UserModel. Then, write the properties for UserModel, then put constructor, getter, setter.



The screenshot shows an IDE with the project structure on the left. The package `spring.course.model` is selected, and the `UserModel` class is being created. The code in the editor is as follows:

```
1 package spring.course.model;
2
3 public class UserModel {
4
5     private String firstName;
6     private String lastName;
7     private Integer membershipId;
8
9     public UserModel(String firstName, String lastName, Integer membershipId) {
10         this.firstName = firstName;
11         this.lastName = lastName;
12         this.membershipId = membershipId;
13     }
14
15     public String getFirstName() {
16         return firstName;
17     }
18
19     public void setFirstName(String firstName) {
20         this.firstName = firstName;
21     }
22
23     public String getLastName() {
24         return lastName;
25     }
26
27     public void setLastName(String lastName) {
28         this.lastName = lastName;
29     }
30 }
```

9. Then ,make a function that returns UserModel object given the userName . Put this annotation `@GetMapping("/getuser/{userName}")` before that function.



The screenshot shows the `UserController` class in the `spring.course.controllers` package. The code is as follows:

```
1 package spring.course.controllers;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RestController;
6 import spring.course.model.UserModel;
7
8 import java.util.HashMap;
9 import java.util.Map;
10
11 @RestController
12 public class UserController {
13
14     private Map<String, UserModel> userMap = new HashMap<>();
15
16     public UserController() {
17         userMap.put("John", new UserModel("John", "Doe", 1234));
18         userMap.put("Jane", new UserModel("Jane", "Doe", 5432));
19     }
20
21     @GetMapping("/getuser/{userName}")
22     public UserModel getUser(@PathVariable String userName) {
23         return userMap.get(userName);
24     }
25 }
```

As we said before, controller is the the first point of access to the application.
So, in controller we write api methods like

@GetMapping("/getUser/username") // @GetMapping is the annotation we need.
After that we make a function that returns UserModel object given the username.

10. Then, call the controller using the api pattern we made in controller.
i.e.

