Search...

Spring - REST Controller

Last Updated : 03 Sep, 2025

Spring Boot is built on top of the Spring Framework and comes with all of Spring's features plus additional capabilities that make development faster and easier. It is especially for building microservices and RESTful APIs because of its auto-configuration, embedded server support and minimal boilerplate code. When building REST APIs in Spring, the @RestController annotation plays a key role in handling HTTP requests and returning JSON responses.

REST Controller

A REST Controller in Spring Boot is a special class used to define RESTful web services. It is annotated with @RestController, which makes it capable of:

- Handling HTTP requests (GET, POST, PUT, DELETE).
- Returning Java objects as JSON/XML without extra configuration.
- Acting as an endpoint for client-server communication.

@RestController Annotation

Spring provides two main controller annotations:

- @Controller annotation is used for MVC-based applications where responses are typically HTML pages.
- @RestController annotation is a specialized version of @Controller that automatically serializes return objects into JSON/XML responses. It is equivalent to @Controller + @ResponseBody.

This annotation is used at the class level and allows the class to handle the requests made by the client. The RestController allows the class to handle REST API requests such as GET, POST, PUT and DELETE by automatically serializing responses to JSON.

Step-by-Step Implementation

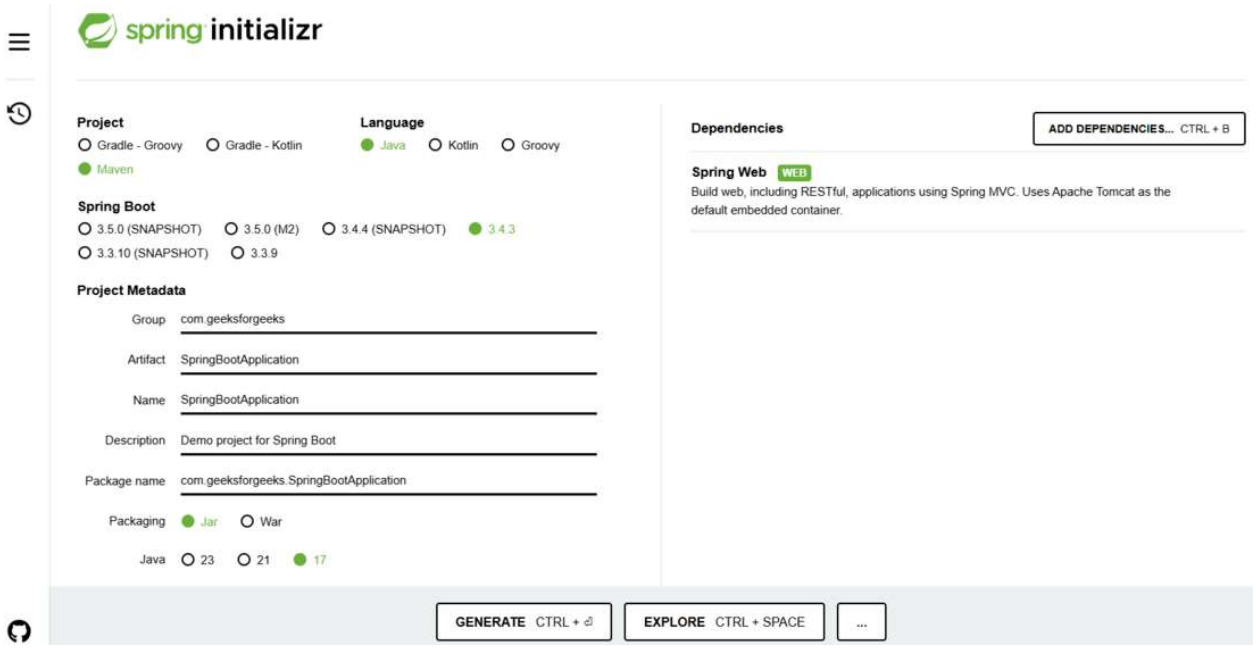
Step 1: Create a Spring Boot Project

Use Spring Initializr to generate a new Spring Boot project.

Project Configuration:

- **Project Type**: Maven
- **Language**: Java
- **Spring Boot Version**: 3.x (Latest Version)
- **Dependencies**: Spring Web
- **Java Version**: 17+

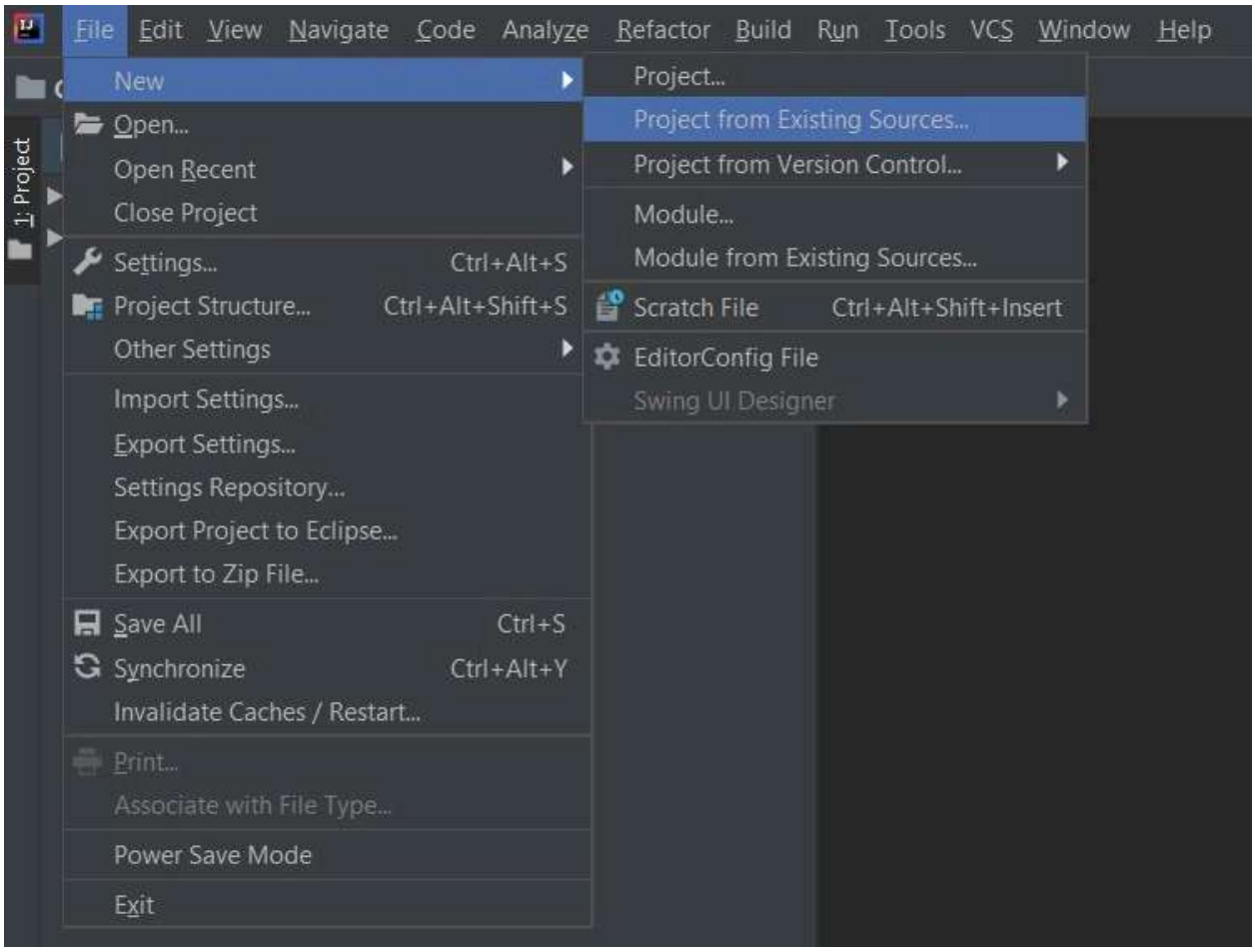
Click on Generate, which will download the starter project.



Step 2: Import the Project in IDE

- Open IntelliJ IDEA (or Eclipse/STS).

- Go to File -> New -> Project from Existing Sources and select pom.xml.
- Wait for Maven to download dependencies.



Note:

In the Import Project for Maven window, make sure you choose the same version of JDK which you selected while creating the project.

Step 3: Create the Model Class

Go to src -> main -> java, create a java class with the name Details.

Details.java:

```
public class Details {

    private int id;
    private String name;

    // Constructor
    public Details(int id, String name) {
        this.id = id;
```

```
        this.name = name;
    }

    // Getters and Setters
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Step 4: Create a REST Controller

Now create another Java class with the name Controller and add the annotation @RestController.

Controller.java:

```
import org.springframework.web.bind.annotation.*;
import java.util.*;

@RestController
@RequestMapping("/api")
public class Controller {

    private List<Details> detailsList = new ArrayList<>();

    // GET API → Fetch all details
    @GetMapping("/details")
    public List<Details> getAllDetails() {
        return detailsList;
    }

    // POST API → Add new detail
    @PostMapping("/details")
    public String addDetails(@RequestBody Details details) {
        detailsList.add(details);
        return "Data Inserted Successfully";
    }

    // PUT API → Update detail by ID
    @PutMapping("/details/{id}")
    public String updateDetails(@PathVariable int id, @RequestBody Details
updatedDetails) {
        for (Details details : detailsList) {
```

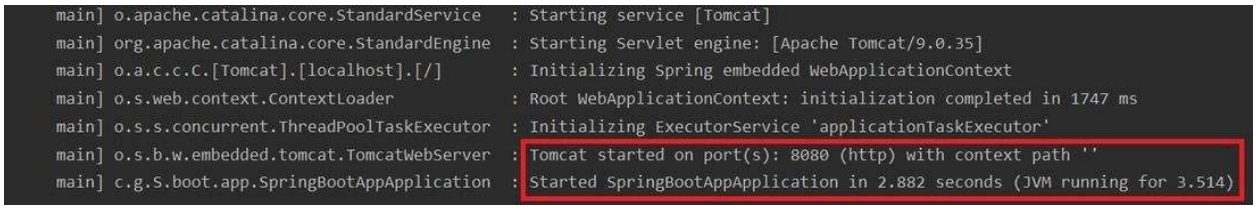
```
            if (details.getId() == id) {
                details.setName(updatedDetails.getName());
                return "Data Updated Successfully";
            }
        }
        return "Detail not found!";
    }

    // DELETE API → Remove detail by ID
    @DeleteMapping("/details/{id}")
    public String deleteDetails(@PathVariable int id) {
        detailsList.removeIf(details -> details.getId() == id);
        return "Data Deleted Successfully";
    }
}
```

This application is now ready to run.

Step 5: Run the Application

Run the SpringBootApplication class to start the embedded Tomcat server.



output

Note:

> *The default port of the Tomcat server is 8080 and can be changed in the application.properties file.*

Step 6: Test APIs using Postman

1. Add a New Detail (POST Request)

Endpoint: POST http://localhost:8080/api/details

Request Body:

```
{
    "id": 1,
    "name": "John Doe"
```

```
}
```

Response:

```
"Data Inserted Successfully"
```


2. Fetch All Details (GET Request)

Endpoint: GET http://localhost:8080/api/details

Response:

```
[
    {
        "id": 1,
        "name": "John Doe"
    }
]
```

3. Delete a Detail (DELETE Request)

Endpoint: DELETE http://localhost:8080/api/details/1

Response:

```
"Data Deleted Successfully"
```

To know how to use Postman, refer: How to test Rest APIs in Postman