Spring Boot

simplilearn

TECHNOLOGY

REST with Spring

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Identify the different annotations used in Spring Boot
- Describe Spring Boot
- Examine the important methods for HTTP
- Demonstrate how to initialize a RESTful web service
- Implement the concept of Spring Boot auto-configuration and Dispatcher Servlet microservices

A Day in the Life of a Full Stack Developer

You work as a developer for an organization that has assigned you to enterprise application development. Being a Java developer, you feel confident in building applications using Java-based frameworks.

The goal is to develop enterprise applications by implementing RESTful web services as a solution.

After considering various options, you decide to use Spring Boot and leverage its different annotations and methods to develop the application.

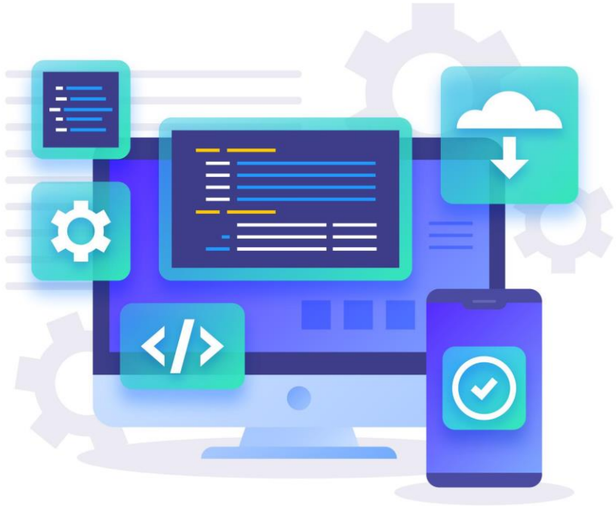
TECHNOLOGY

Spring Boot: Annotations

simplilearn

Spring Boot: Annotations

Spring Boot is a preferred framework for developing RESTful web services as a solution for enterprise applications.



To implement RESTful web services, configure the **pom.xml** file by adding the Spring Boot web dependency.

Spring Boot: Annotations

Syntax for adding dependencies in the **pom.xml** file:

```
<dependencies>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependencies>
```

Spring Boot: Annotations

Spring Boot utilizes annotation-based implementation for developing RESTful web services.

Rest Controller

Traditional methods often require up to ten months for single-stage development.

@RestController annotation can be used on a Java class to build RESTful web services.

@RestController allows sending both JSON or XML as a response from its web methods.

Rest Controller

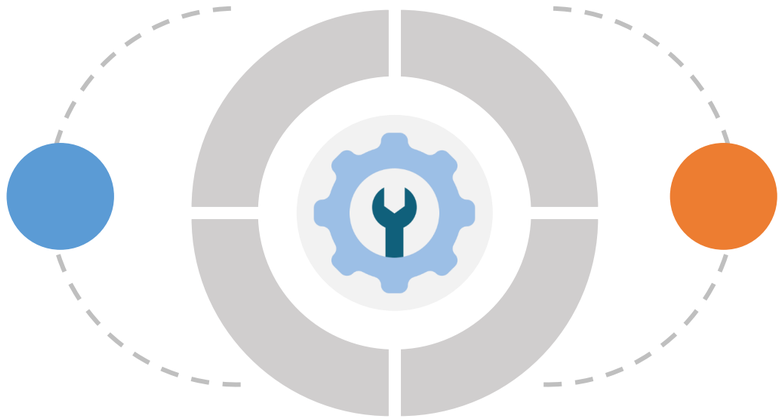
Syntax for **@RestController**:

```
@RestController
public class CategoriesController {
}
```

RequestMapping

The **@RequestMapping** annotation is used to define and access endpoints for a web service.

@RequestMapping is used on methods inside the class.

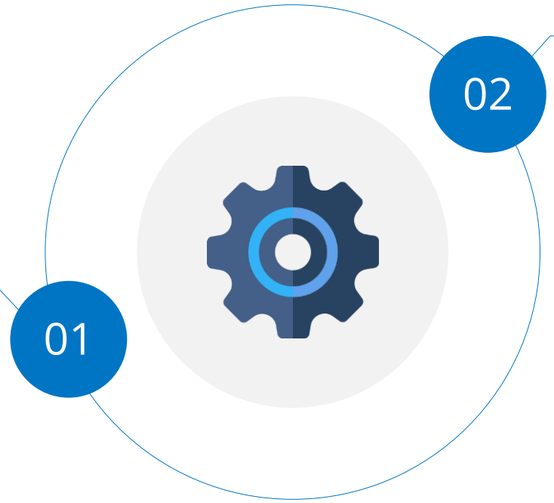


It helps declare the Request URI, allowing clients to access the REST endpoints.

This annotation plays a vital role in simplifying the development of web services by mapping HTTP requests to specific methods.

RequestMapping

A RequestMapping method helps to:



01 Consume a request

02 Produce a response

RequestMapping

Syntax for **@RequestMapping**:

```
@RequestMapping(value = "/categories")
public HashMap<Integer, Category> getCategories() {
    HashMap<Integer, Category> categoryMap = new HashMap<Integer, Category>();
    // Code to populate the categoryMap
    return categoryMap;
}
```

RequestBody

The **@RequestBody** annotation defines the content type of the request body in a RESTful API.

Syntax:

```
public ResponseEntity<User> createUser(@RequestBody
User user) {
    // Code to create the user
    // ...
}
```

This annotation specifies that the incoming HTTP request should be treated as the request body and be deserialized into the specified object type.

Path Variable

The **@PathVariable** annotation is used to extract and use values from the request URI in a Spring web application.

Syntax:

```
public ResponseEntity<User>
updateUser(@PathVariable("email") String email) {
    // Code to update user using the email
    // ...
}
```

It allows developers to define dynamic parts of the URI as variables and access their values within the method.

RequestParameter

The @RequestParameter annotation is used to:

- Capture the data which the client sends in the request object as parameters
- Set a default value in case no data comes from the client in the request

RequestParameter

Syntax for RequestParameter:

```
public ResponseEntity<Order> getUser(
   @RequestParam(value = "email", required = true,
defaultValue = "john@example.com") String name) {
    // Method implementation
}
```

Spring Boot: RESTful

Spring Boot: RESTful

Spring Boot is a preferred framework for developing RESTful web services as a solution for enterprise applications. It offers several key features and benefits:

Architectural Approach

Representational State Transfer (REST)

Utilizes HTTP Features

Spring Boot: RESTful

When it comes to data exchange in RESTful web services, there is no standardized format.

Note

JSON is a preferred format with REST due to its data representation.

Spring Boot: RESTful

The key abstraction in REST is resource access using a URI (Uniform Resource Identifier).

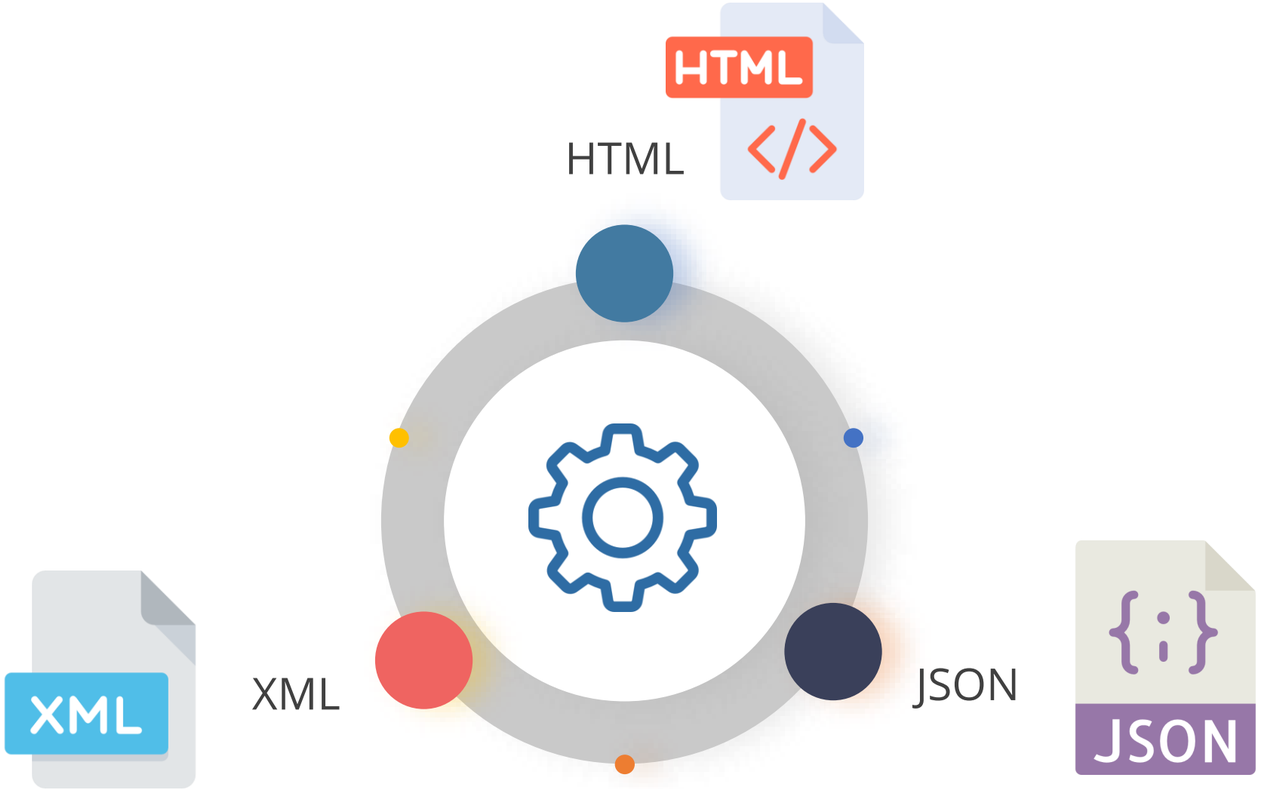
Example:

```
https://localhost:8080/news
```

Here, the URI represents a specific resource related to news. Clients can use this URI to access or manipulate the news resource through appropriate HTTP methods.

Spring Boot: RESTful

Resources can have different representations based on requirements:



Spring Boot: RESTful

When a resource is requested using its URI, the representation of the resource is provided.

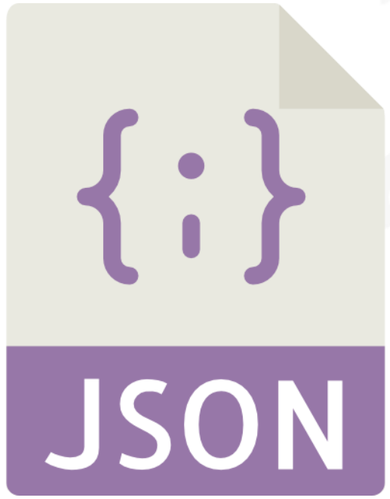
GET

PUT

POST

DELETE

Retrieves a resource and returns JSON



Spring Boot: RESTful

When a resource is requested using its URI, the representation of the resource is provided.

GET

PUT

POST

DELETE

Updates an existing resource with new data

Spring Boot: RESTful

When a resource is requested using its URI, the representation of the resource is provided.

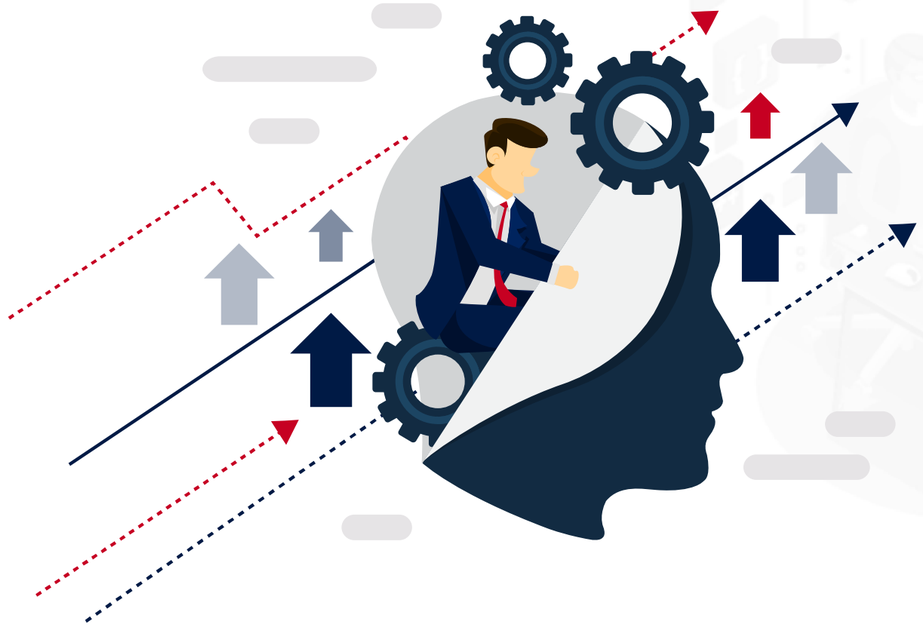
GET

PUT

POST

DELETE

Creates a new resource with data



Spring Boot: RESTful

When a resource is requested using its URI, the representation of the resource is provided.

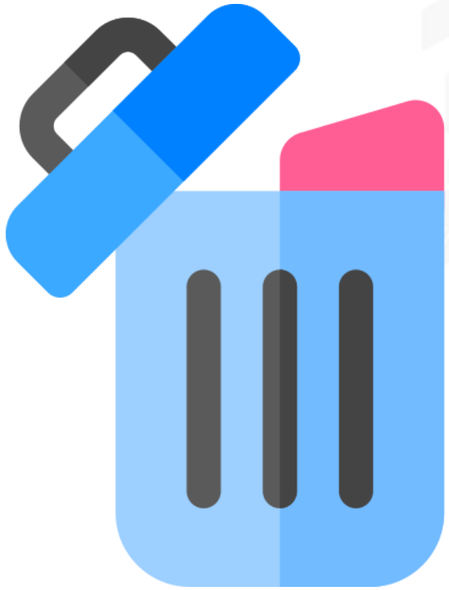
GET

PUT

POST

DELETE

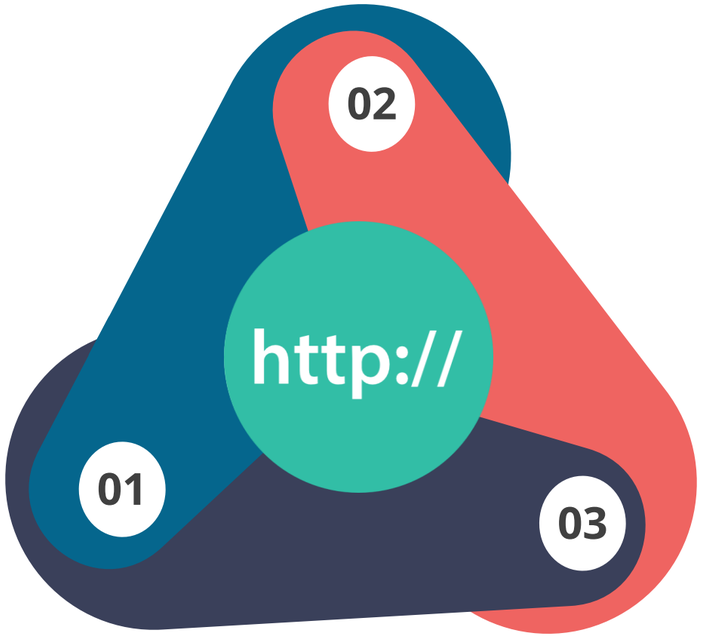
Deletes a resource based on an ID



Spring Boot: RESTful

Examples of using HTTP methods:

GET /products/{code}: Fetch a product based on the product code from the resource.



GET /products: Fetch all products from the resource.

POST /orders: Create a new order using the resource.

TECHNOLOGY

RESTful Web Services with Spring Boot

simplilearn

RESTful Web Services with Spring Boot

RESTful web services are easily consumed on the client side.

RESTful Web Services with Spring Boot

The following are the benefits of REST web services:

- Security
- Suitable actions (GET, POST, PUT, DELETE)
- Redirection and forwarding
- Caching

RESTful Web Services with Spring Boot

Critical factors for resilient RESTful services:

- Build the RESTful web service using Spring Boot
- Implement a simple web service that returns a string response, such as **Welcome to Spring Boot**, along with the current date and time stamp.

RESTful Web Services with Spring Boot

Build the service that will accept HTTP GET requests at http://localhost:8080/welcome

```
{
"code": 101,
"content": "Welcome to Spring Boot, Fionna.
It's 2022-12-20 20:00:00"
}
```

RESTful Web Services with Spring Boot

Customize with an optional name parameter in the query string:

```
http://localhost:8080/welcome?name=John
```

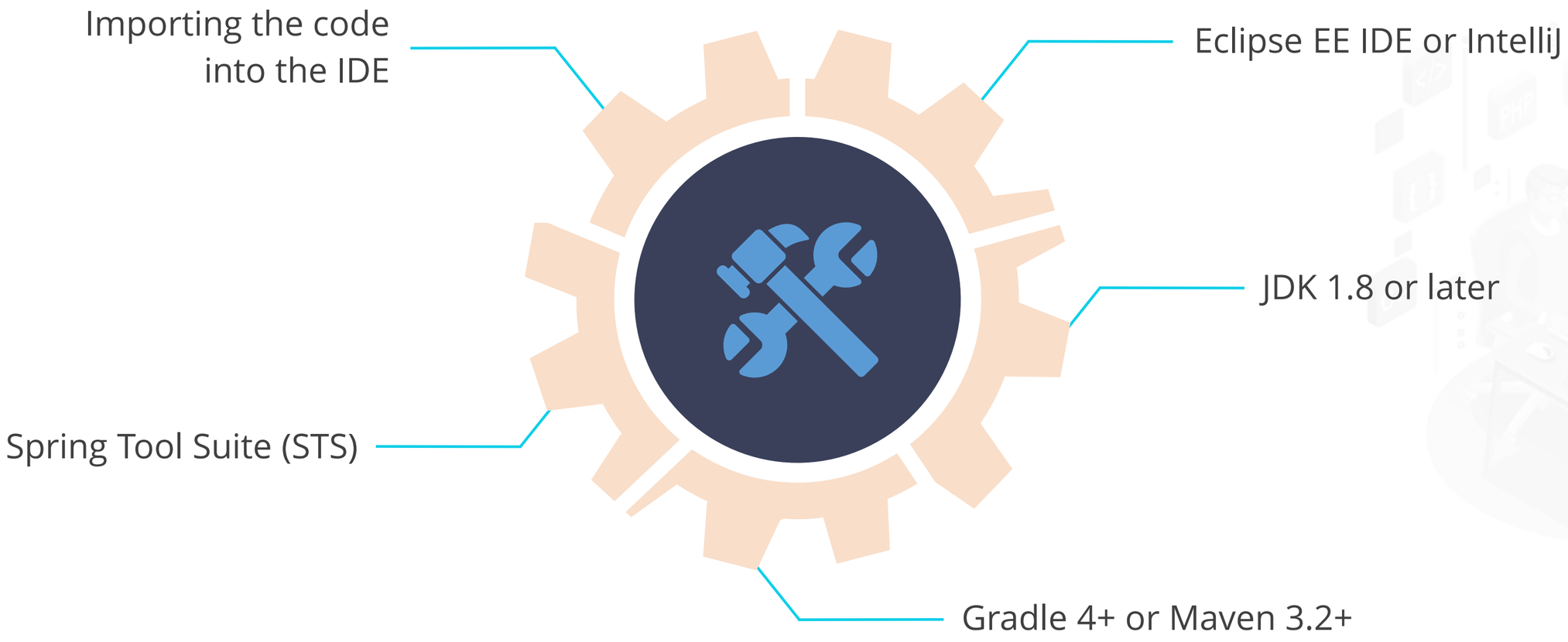
RESTful Web Services with Spring Boot

The name parameter uses the value **John** and will be reflected in the response:

```
{ “code:101,” “content”:”Welcome to Spring Boot, John.
Its 2022-20-20 20:00:00” }
```

RESTful Web Services with Spring Boot

The name parameter uses the value **John** and will be reflected in the response along with:



Initialize a RESTful Web Service

Initializing a RESTful Web Service

Web applications operate on the client-server architecture, which involves the following key aspects:

The application serves as the client, interacting with the server.

The application needs to communicate with the server or backend to store data.

The client can directly call services by sending HTTP requests to the server.

Communication between the client and server occurs using the HTTP protocol.

Initializing a RESTful Web Service

Web applications follow the client-server architecture. In this:

REST stands for Representational State Transfer

REST follows a conventional approach for building HTTP services.

Initializing a RESTful Web Service

HTTP protocols facilitate CRUD operations:

CREATE:
Creating new resources

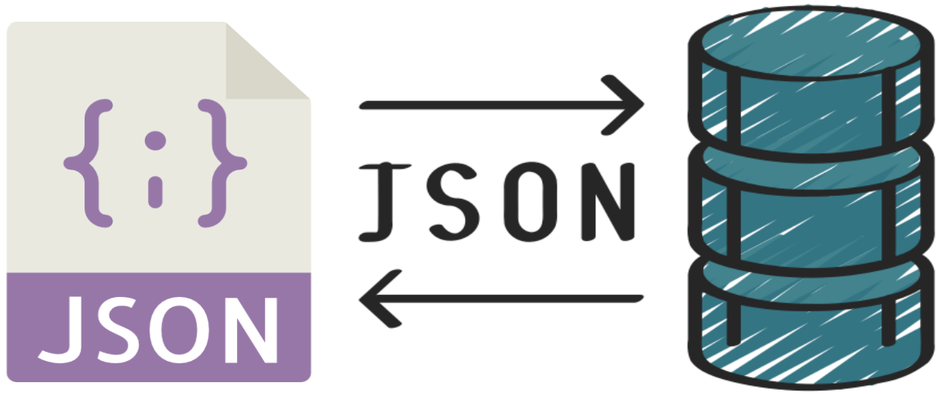
READ:
Retrieving existing resources

UPDATE:
Modifying existing resources

DELETE:
Removing resources

Initializing a RESTful Web Service

Create a Controller to return data in JSON format.



Initializing a RESTful Web Service

To manually initialize a project using Spring Initializr, follow these steps:

01 Navigate to https://start.spring.io

02 Choose either Gradle or Maven and Java as a programming language for development

03 Click on Dependencies and select the spring web

04 Click the **Generate** to create the project

05 Download the ZIP file, which contains the generated project files and folder structure

Initializing a RESTful Web Service

In Eclipse EE IDE, one can install the Spring STS plugin

Note

It will have the Spring Initializr integration.

Initializing a RESTful Web Service

Below are the steps to initialize a RESTful web service:

Start the process by thinking about the service interactions

Create the web service

Handle GET requests for the **/welcome** object

Initializing a RESTful Web Service

In response to the GET request, a JSON containing the 400 error code should be returned. The output will be:

```
{
  "code": "400",
  "content": "Welcome to Spring Boot, Fionna.
  It's 2022-20-20 20:00:00"
}
```

The code field is the unique identifier for the **welcome**, whereas the content is the textual representation.

Initializing a RESTful Web Service

The resource representation class is created to model the **welcome** representation.

```
src/main/java/com/example/restservice/Welcome.java shows.

package com.example.restservice;
public class Welcome  {
private final long code;
private final String content;
public Welcome (long code, String content) {
                  this.code = code;
                  this.content = content;
}
public long getCode() {
            return code;
}
public String getContent() {
            return content;
}
}
```

It provides the Java objects with fields, constructors, and accessors for the code and the content data.

Creating the Resource Controller

While building the RESTful web services using the Spring approach, the HTTP requests are handled by the controller.

```
src/main/java/com/example/restservice/WelcomeController.java.
```

Creating the Resource Controller

Handles GET requests for **/welcome** by returning a new instance of the Welcome class:

```
package com.example.restservice;
import java.util.Date;
import.java.util.concurrent.atomic.AtomicLong;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class WelcomeController  {
                    private static final String template =
“”Welcome to Spring Boot, %. Its ”+new Date();
                    private final AtomicLong counter = new
AtomicLong();
          @GetMapping(“/welcome”)
public Welcome welcome(@RequestParam(value = “name”,
defaultValue = “Fionna”) String name) {
return new Welcome(counter.incrementAndGet(),
String.format(template, name));
}
}
```

Creating the Resource Controller

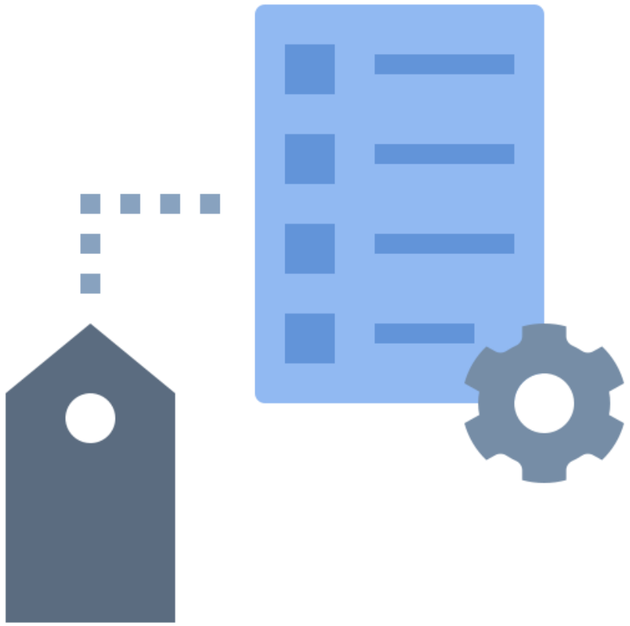
The **@GetMapping** annotation helps verify that the HTTP GET requests to **/welcome** are mapped to the **welcome()** method.

Creating the Resource Controller

@RequestParam helps to bind the value of the query string parameter name into the name parameter of the **welcome()** method.

Creating the Resource Controller

Implementation of the method body creates and returns the new **welcome** object with the code and the content attributes.

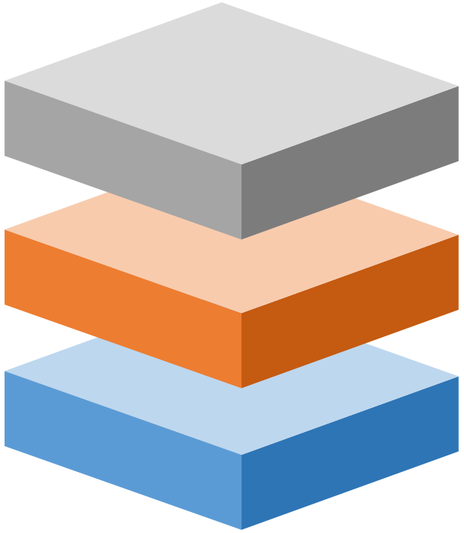


Creating the Resource Controller

Code uses the Spring **@RestController** annotation. In this:

It marks the classes as the controller.

It includes shorthand for including both **@Controller** and the **@ResponseBody**.



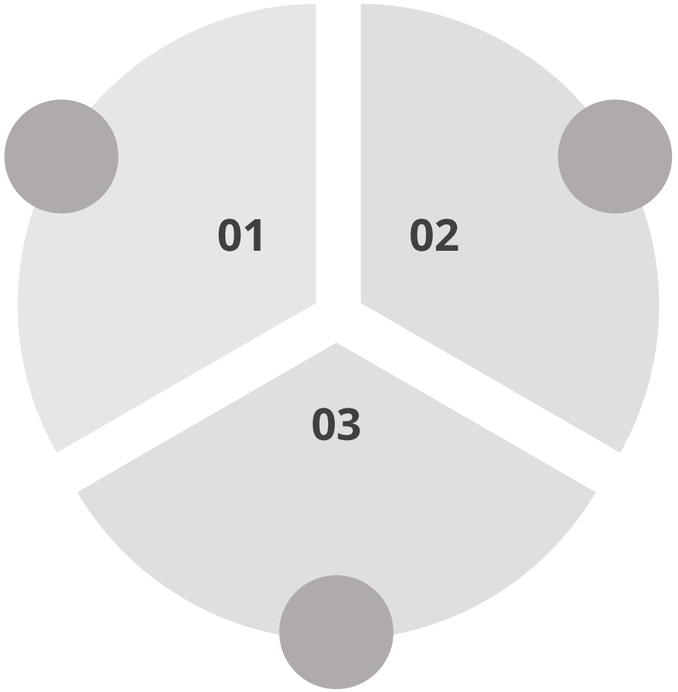
Every method returns the domain object instead of the view.

Creating the Resource Controller

Spring's HTTP message converter support helps convert the **welcome** object to JSON.

Creating the Resource Controller

@SpringBootApplication is the convenience annotation that adds:



Configuration

Tags the class as the source of bean definitions for the application context

EnableAutoConfiguration

Informs Spring Boot to add beans based on the classpath settings

ComponentScan

Informs Spring to explore other components, configurations, and services in the com/example package

Creating the Resource Controller

The web application is written entirely in Java, which saves time by avoiding the need for unnecessary configuration.

Note

There is not a single line of the XML or **web.xml** file.

Building an Execute JAR

The following tasks can be performed while building an execute JAR file:

A single executable JAR file that contains all the necessary dependencies, classes, and resources can be built.

Applications can be executed from the command line using Gradle or Maven.

Building an Execute JAR

Creating an executable JAR file increases the ease of shipping, versioning, and deploying the service as an application.

Note

If one uses Gradle, the application can run by using **./gradlew bootRun**.

Building an Execute JAR

The JAR file can be built using the **./gradlew** build.

Run the JAR file:

```
java -jar build/libs/welcome-rest-service-0.1.0.jar
```

Building an Execute JAR

The JAR file can be built with the **./mvnw** clean package. Run the JAR file:

```
java -jar target/welcome-rest-service-0.1.0.jar
```

Building an Execute JAR

Runnable JAR is created using these steps:

1. Building the classic WAR file
2. Logging output is displayed

Building an Execute JAR

Once the service is up, visit http://localhost:8080/welcome to see the following:

```
{
code:”101”,
“content”:” ”Welcome to Spring Boot, Fionna. Its 2022-20-20
20:00:00”
}
```

Building an Execute JAR

Add the name query string parameter by visiting http://localhost:8080/welcome?name=John

```
{
code:”101”,
“content”:” ”Welcome to Spring Boot, John. Its 2022-20-20
20:00:00”
}
```

It shows that the **@RequestParam** arrangement in the WelcomeController is working.

Building an Execute JAR

The name parameter takes the default value and can be overridden through the query string.

Building an Execute JAR

The code change indicates the same **WelcomeController** instance across multiple requests.

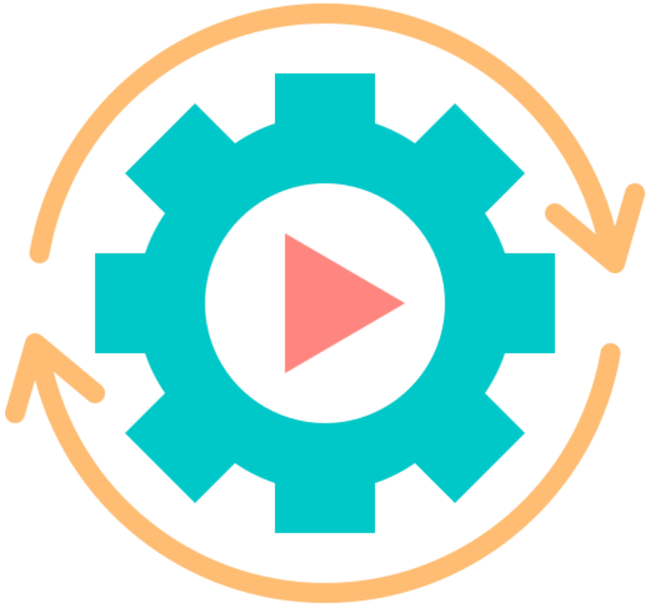
TECHNOLOGY

Spring Boot Auto Configuration and Dispatcher Servlet

simplilearn

Spring Boot Auto Configuration

Spring Boot configures Spring dependencies based on classpath presence.



Spring Boot Auto Configuration

The following are the characteristics of Spring Boot Auto Configuration. It:

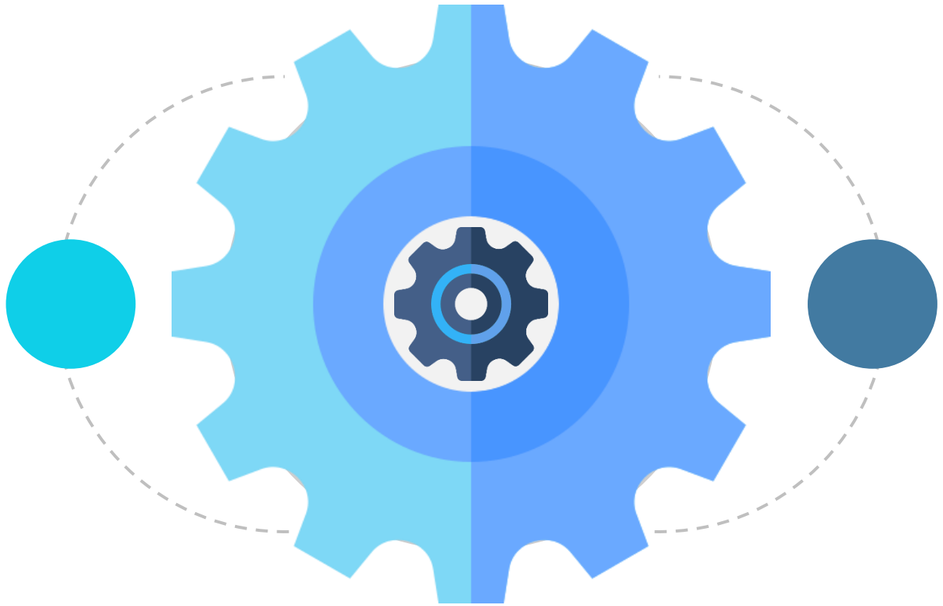
Makes development easier and faster as compared to others

Saves time by eliminating the need to define beans

Spring Boot Auto Configuration and Dispatcher Servlet

MVC database-driven Spring MVC application requires a lot of configuration, such as:

Dispatcher servlet



View resolver

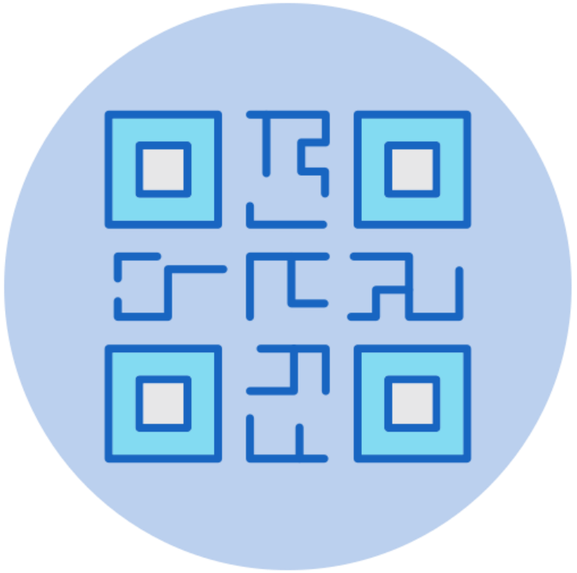
Spring Boot Auto Configuration and Dispatcher Servlet

Spring Boot auto-configures the dispatcher servlet if the Spring MVC JAR is on the classpath. It auto-configures:

- Jackson, if the Jackson jar is on the classpath
- Data source, if the hibernate jar is on the classpath
- Itself, by adding **@SpringBootApp** or **@EnableAutoConfiguration** annotation in the startup class

Spring Boot Auto Configuration and Dispatcher Servlet

It enables components scan, which is a feature of Spring to trigger an automatic scan for classes.



Spring Boot Auto Configuration and Dispatcher Servlet

Examples of the auto-configuration done by the Spring Boot:

- DispatcherServletAutoConfiguration
- DataSourceAutoConfiguration
- ErrorMvcAutoConfiguration
- JacksonAutoConfiguration

Spring Boot Auto Configuration and Dispatcher Servlet

To exclude the classes from the auto-configuration, add this configuration:

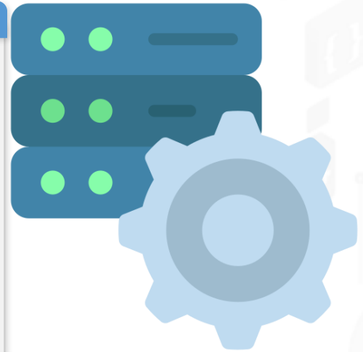
```
@SpringBootApplication(exclude={JacksonAutoConfiguration.class
,JmxAutoConfiguration.class})
```

```
spring.autoconfiguration.exclude=org.springframework.boot.auto
configure.jackson.JacksonAutoConfiguration
```

Spring Boot Auto Configuration and Dispatcher Servlet

Auto-configuration is generated by enabling the debug mode

```
logging.level.org.springframework=debug
```



Creating a RESTful Web Service with @RestControl

Problem Statement:

You have been asked to create a basic web service using Spring Boot to build and deploy RESTful APIs for various applications and use cases.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Implementing the web services that return a JSON file
2. Creating a Controller class
3. Implementing maps with key-value pair
4. Creating a News Model class

Key Takeaways

- Spring Boot is a preferred framework for developing RESTful web services as a solution for enterprise applications.
- Spring Boot uses Annotation-based implementation to develop RESTful web services.
- The **@GetMapping** annotation helps verify that the HTTP GET requests to /welcome are mapped to the **welcome()** method.
- **@RequestParam** helps to bind the value of the query string parameter name into the name parameter of the **welcome()** method.
- Spring Boot configures Spring dependencies based on classpath presence.

TECHNOLOGY

Thank You

simplilearn