

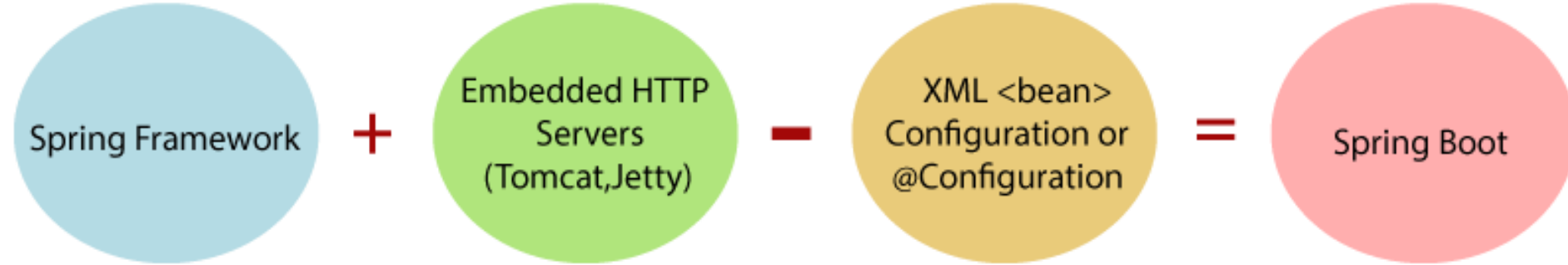
UNIT-4 Spring Boot

Introduction

- Spring Boot is a project that is built on the top of the Spring Framework.
- It provides an easier and faster way to set up, configure, and run both simple and web-based applications.
- It is a Spring module that provides the **RAD (*Rapid Application Development*)** feature to the Spring Framework.
- It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

Spring vs Spring Boot

- **Spring:** Spring Framework is the most popular application development framework of Java. The main feature of the Spring Framework is **dependency Injection** or **Inversion of Control (IoC)**. With the help of Spring Framework, we can develop a **loosely** coupled application. It is better to use if application type or characteristics are purely defined.
- **Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a stand alone Spring-based application or RESTful services.



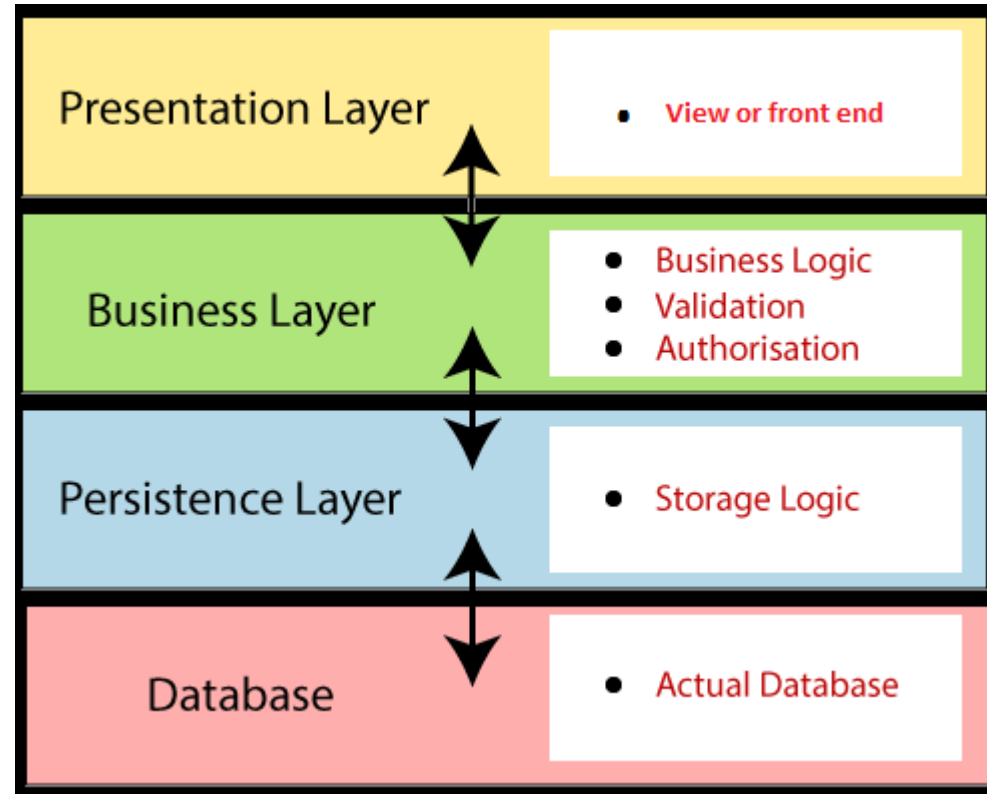
- In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.
- In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

Spring Boot Framework Uses

- The dependency injection approach is used in Spring Boot.
- It contains powerful database transaction management capabilities.
- It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
- It reduces the cost and development time of the application.

Spring Boot Architecture

- Spring Boot is a module of the Spring Framework. It is used to create stand-alone, production-grade Spring Based Applications with minimum efforts. It is developed on top of the core Spring Framework.
- Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.
- Before understanding the **Spring Boot Architecture**, we must know the different layers and classes present in it. There are **four** layers in Spring Boot are as follows:
 - **Presentation Layer**
 - **Business Layer**
 - **Persistence Layer**
 - **Database Layer**



- **Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.
- **Business Layer:** The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.
- **Persistence Layer:** The persistence layer contains all the **storage logic** and translates business objects from and to database rows.
- **Database Layer:** In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed.

Spring Boot Softwares

- Java 1.8
- Maven 3.0+
- An IDE (IntelliJ) is recommended.

First Project

- Open IntelliJ IDEA 2020.3. Navigate to File -> New -> Project
- Then choose maven tab and set
 - Group: com.example
 - Artifact: welcome-springboot
- Then click Finish
- Once the project is created, navigate to the `src/main/java/com/example/welcome` directory. Create the `WelcomeSpringbootApplication.java`
- Create the `pom.xml` file in the root directory of your project. Ensure it includes the necessary Spring Boot dependencies:

Example1

WelcomeSpringbootApplication.java

```
package com.example.welcome;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class WelcomeSpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(WelcomeSpringbootApplication.class, args);
    }

    @GetMapping("/")
    public String welcomeMessage() {
        return "Welcome to Spring Boot";
    }
}
```

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>welcomeproject</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>
```

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

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

Output:

Welcome to Spring Boot

- `@SpringBootApplication`: Marks this class as the entry point for the Spring Boot application.
- `@RestController`: Enables RESTful web services.
- `@GetMapping("/")`: Maps the root URL (/) to the `welcomeMessage()` method.

Annotations

- Annotations in Spring Boot are special forms of metadata that provide instructions or behavior to the framework. They simplify and reduce boilerplate code by allowing the developer to define functionality using declarative programming. Spring Boot uses a wide variety of annotations to manage beans, handle HTTP requests, define configuration, etc.

@SpringBootApplication

- @SpringBootApplication is a meta-annotation that combines three important Spring annotations:
 - @SpringBootConfiguration
 - @EnableAutoConfiguration
 - @ComponentScan
- It is placed on the main class of a Spring Boot application and acts as the entry point for the application. When this annotation is used, Spring Boot will automatically configure your application based on the dependencies added to your project and scan for components (beans) in the package where this class is located.

Ex:

```
@SpringBootApplication
public class MySpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(MySpringBootApplication.class, args);
    }
}
```

@RestController

- @RestController is a specialized version of the @Controller annotation. It is used in Spring to define a controller that handles HTTP requests and returns data directly as the response body rather than rendering a view.
- @RestController is a combination of:
 - @Controller: Marks the class as a web controller.
 - @ResponseBody: Indicates that the return value of the methods in this controller should be written directly to the HTTP response body, usually in JSON or XML format.
- @Controller is used to define a regular Spring MVC controller. Unlike @RestController, it is typically used when you want to return a view (e.g., JSP, Thymeleaf) instead of sending the response body directly.

Ex:

```
@RestController
```

```
public class MyRestController {  
    @GetMapping("/welcome")  
    public String welcomeMessage() {  
        return "Welcome to Spring Boot!";  
    }  
}
```

@GetMapping

- @GetMapping is a specialized annotation in Spring that handles HTTP GET requests. It is a shorthand for @RequestMapping(method = RequestMethod.GET) and is used to map HTTP GET requests to specific handler methods in a controller.
- It simplifies the mapping of URLs to methods that should be invoked when an HTTP GET request is received.

Ex:

```
@RestController
public class MyController {
    @GetMapping("/greet")
    public String greet() {
        return "Hello, World!";
    }
}
```

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>springboot-addition</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>
```

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

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

Pom.xml

- The pom.xml (Project Object Model) file is a core part of a Maven project and serves as the configuration file for Maven-based projects. It defines the project's dependencies, plugins, goals, build configuration, and other settings.
- When you run Maven commands, Maven refers to the pom.xml to know how to build the project and what dependencies and plugins it needs.
- Functions of pom.xml:
 - Dependency Management: Automatically handles libraries and their versions.
 - Build Configuration: Defines how the project is compiled, packaged, and deployed.
 - Project Metadata: Holds information about the project and versioning.
 - Plugins: Enhances the build process with specific tasks like running tests, generating reports, or packaging the application.

Key Elements of pom.xml

- **Project Metadata:** Defines essential project information such as the group ID, artifact ID, version, and packaging type.
 - `<groupId>com.example</groupId> <!-- Unique identifier for the project -->`
 - `<artifactId>my-app</artifactId> <!-- The name of the project -->`
 - `<version>1.0.0</version> <!-- The version of the project -->`
 - `<packaging>jar</packaging> <!-- Type of packaging: jar, war, etc. -->`

- **Dependencies:** Lists all the external libraries (dependencies) that the project needs to compile and run. Maven automatically downloads these libraries from remote repositories.

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.4.2</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
    <version>2.4.2</version>
  </dependency>
</dependencies>
```

- Parent Project: If your project is based on another project (like Spring Boot), the parent section can inherit certain configurations from that parent project. Spring Boot projects typically inherit from spring-boot-starter-parent.

```
<parent>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-parent</artifactId>  
  <version>2.4.2</version>  
</parent>
```


- **Plugins:** Defines Maven plugins that provide additional functionalities during the build process, such as compiling Java code, packaging, testing, etc.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Example2

- To explain the use of @Controller

WelcomeSpringbootApplication.java

```
package com.example.welcome;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@Controller
public class WelcomeSpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(WelcomeSpringbootApplication.class, args);
    }

    @GetMapping("/")
    public String welcomeMessage() {
        return "welcome";
    }
}
```

Welcome.html

```
<html>
<head>
  <title>Welcome Page</title>
</head>
<body>
<h1>Welcome to Spring Boot!</h1>
<p>This is a web page</p>
</body>
</html>
```

Pom.xml

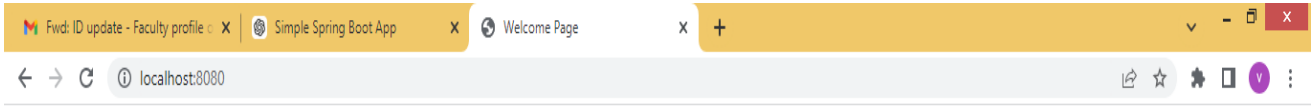
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>welcomeproject</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>
```

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

    <build>
      <plugins>
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>
  </project>
```



Welcome to Spring Boot!

This is a web page



Example3

- To explain the use of GetMapping with the input string
`@GetMapping("/print")`

WelcomeSpringbootApplication.java

```
package com.example.welcome;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class WelcomeSpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(WelcomeSpringbootApplication.class, args);
    }

    @GetMapping("/print")
    public String welcomeMessage() {
        return "Welcome to Spring Boot";
    }
}
```


Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>welcomeproject</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>
```

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

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

Output:

`http://localhost:8080/print`

Welcome to Spring Boot

Example4

- Spring boot application to display the square of the given number by the user

WelcomeSpringBootApplication.java

```
package com.example.springbootaddition;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootApplication.class, args);
    }
}
```

squareController.java

```
package com.example.springbootaddition;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestParam;
```

```
@Controller
```

```
public class squareController {
```

```
    @GetMapping("/")
```

```
    public String showForm() {
```

```
        return "num";
```

```
    }
```

```
@PostMapping("/square")
```

```
public String squareNumbers(@RequestParam("num1") int num1,
```

```
                             Model model) {
```

```
    int square = num1*num1;
```

```
    model.addAttribute("square", square);
```

```
    return "result";
```

```
}
```

```
}
```

num.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Squaring the Numbers</title>
</head>
<body>
<h1>Squaring the Number</h1>
<form action="/square" method="post">
  <label for="num1">Number 1:</label>
  <input type="text" id="num1" name="num1"><br>
  <input type="submit" value="square">
</form>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Result</title>
</head>
<body>
<h1>Result</h1>
<p>The square is: <span th:text="${square}"></span></p>
<a href="/">Square other numbers</a>
</body>
</html>
```

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>welcomeproject</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>
```

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

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```


Result

The square is: 16

[Square other numbers](#)

Example 5

- Practice the Spring boot web app for addition and subtraction of two numbers

SpringBootApplication.java

```
package com.example.springbootaddition;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootApplication.class, args);
    }
}
```

AdditionController.java

```
package com.example.springbootaddition;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class AdditionController {

    @GetMapping("/")
    public String showForm() {
        return "add";
    }

    @PostMapping("/add")
    public String addNumbers(@RequestParam("num1") int num1,
                             @RequestParam("num2") int num2,
                             Model model) {
        int sum = num1 + num2;
        model.addAttribute("sum", sum);
        return "result";
    }
}
```

add.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Add Two Numbers</title>
</head>
<body>
<h1>Add Two Numbers</h1>
<form action="/add" method="post">
  <label for="num1">Number 1:</label>
  <input type="text" id="num1" name="num1"><br>
  <label for="num2">Number 2:</label>
  <input type="text" id="num2" name="num2"><br><br>
  <input type="submit" value="Add">
</form>
</body>
</html>
```

Result.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Result</title>
</head>
<body>
<h1>Result</h1>
<p>The sum is: <span th:text="${sum}"></span></p>
<a href="/">Add more numbers</a>
</body>
</html>
```

SpringIO

- This is a web-based tool provided by Spring IO to quickly bootstrap Spring Boot projects. It allows us to generate a skeleton project by selecting dependencies, packaging types (Maven/Gradle), and Java versions. we can access it via the [Spring Initializr](https://start.spring.io) website.
- By using the URL start.spring.io, we can quickly generate the required file templates for the spring boot application.
 - If simple application, one java file will be created automatically without any code (Then user just have to type the code based on the application need). POM file will be fully created with all the dependencies automatically.
 - If web application, two java files will be created automatically without any code (Then user just have to type the code based on the application need). POM file will be fully created with all the required dependencies automatically.
 - If web application with DB connection (CRUD Operations based application), four java files (entity, main, controller, repository) will be created automatically without any code (Then user just have to type the code based on the application need). POM file and application.properties file will be fully created with all the required dependencies automatically.

Uses of SpringIO

- Project Setup – Quickly generates a Spring Boot project with pre-configured dependencies and build files (Maven).
- Dependency Management – Allows selecting required dependencies (e.g., web, JPA, security) for your application without manual configuration.
- Version Selection – Enables choosing the appropriate Spring Boot version for compatibility and feature set.
- Packaging Options – Provides options to create either a JAR or WAR file for deployment.
- Custom Group and Artifact – Helps define project metadata like Group ID, Artifact ID, and package structure.
- Language and Framework Choices – Supports Java, Kotlin, and Groovy as languages for the Spring Boot application.
- Downloadable Project Structure – Generates a fully functional, ready-to-run Spring Boot project, which can be downloaded as a zip file.


(44) WhatsApp xExtensions - G xZoho Creator - xInbox (2,693) - xChatGPT xSpring InitializNew Tab xNew Tab x+ - x

start.spring.io

☰

🕒

🐱

 **spring**initializr

Project

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ Maven

Language

☒ Java

☐ Kotlin

☐ Groovy

Spring Boot

☐ 3.4.0 (SNAPSHOT)

☐ 3.4.0 (M3)

☐ 3.3.5 (SNAPSHOT)

☒ 3.3.4

☐ 3.2.11 (SNAPSHOT)

☐ 3.2.10

Project Metadata

Group

com.example

Artifact

demo

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

demo.zip

Show all x

10:51 AM

9/24/2024

- Give any example spring boot application for SpringIO topic.

RDBMS CRUD

- In RDBMS (Relational Database Management System), CRUD operations represent the four basic functions that are essential for interacting with a database. These operations stand for Create, Read, Update, and Delete.
1. Create: The Create operation allows inserting new records into a table.
SQL Command:
INSERT INTO Example:sqlCopy code INSERT INTO students (id, name, age) VALUES (1, 'John', 22);
 2. Read: The Read operation retrieves records from a table.
SQL Command:
SELECT Example:sqlCopy code SELECT * FROM students WHERE id = 1;
 3. Update: The Update operation modifies existing records in a table.
SQL Command:
UPDATE Example:sqlCopy code UPDATE students SET name = 'Jane' WHERE id = 1;
 4. Delete: The Delete operation removes records from a table.
SQL Command:
DELETE FROM Example:sqlCopy code DELETE FROM students WHERE id = 1;

How to use CRUD Operations In Springboot

- To implement CRUD operations in Spring Boot with an HTML input/output page using Thymeleaf templates, and to configure application.properties for database connection, we can follow the steps listed below:

Step1: Set Up Your Spring Boot pom.xml

- Use Spring Initializer or pom.xml to create a new Spring Boot project with the following dependencies:
 - Spring Web
 - Spring Data JPA
 - Thymeleaf
 - H2 Database (or any other database)

- **Step2: Configure application.properties**
- Define the database connection and JPA settings in the application.properties file located in the src/main/resources folder.
 - Set database connection properties (H2/MySQL/etc.).
 - Enable H2 console (if using H2).
 - Configure Hibernate dialect, DDL auto settings, etc.

Step3: Create an Entity Class

- Define an entity class annotated with `@Entity`.
- Add fields that represent table columns.
- Use `@Id` and `@GeneratedValue` for the primary key.
- Use getters and setters for the fields.

- **Step4: Create a Repository Interface**

- Create a repository interface by extending `JpaRepository<EntityClass, ID>`.
- This interface will provide built-in methods for CRUD operations.

Step5: Create a Controller Class

- Create a controller class annotated with `@Controller`.
- Define methods mapped to HTTP endpoints (e.g., `/create`, `/read`, `/update`, `/delete`) using `@GetMapping` and `@PostMapping`.
- Use `Model` and `ModelAndView` to pass data to Thymeleaf templates.
- Handle form submissions (from the HTML input page) and return results (to the HTML output page).

- **Step 6: Create HTML Pages in src/main/resources/templates**
- Input Page: Create an HTML form (e.g., create.html) using Thymeleaf for accepting data input. The form should be connected to a controller method using th:action and th:object.
- Output Page: Create an output page (e.g., output.html) to display the results or details of the data (such as listing records from the database).

Example :CRUD operations or DB based applications with Controller annotations

Needed files

- Main class file
- Entity class file
- Repository interface file
- Controller file (If needed, include service file)
- Pom.xml file
- Applicaion.properties file
- Html files

Example

- Example to add student details (rollno, name) to the database

HTML files

“StudentForm.html”

```
<html>
<head>
  <meta charset="UTF-8">
  <title>Add Student</title>
</head>
<body>
<h1>Add Student</h1>
<form action="/add" method="post">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name" required>
  <button type="submit">Add Student</button>
</form>
<a href="/list">View Students</a>
</body>
</html>
```

“StudentList.html”

```
<html>
<head>
  <meta charset="UTF-8">
  <title>Student List</title>
</head>
<body>
<h1>Student List</h1>
<table border="1">
  <tr>
    <th>Roll No</th>
    <th>Name</th>
  </tr>
  <tbody>
    <tr th:each="student : ${students}">
      <td th:text="${student.rollNo}"></td>
      <td th:text="${student.name}"></td>
    </tr>
  </tbody>
</table>
<a href="/">Add More Students</a>
</body>
</html>
```

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>Welcomemessage</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
    <relativePath/>
  </parent>

  <dependencies>

    <!-- Spring Boot Starter for Web -->

    <dependency>

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

    </dependency>

    <!-- Spring Boot Starter for JPA -->

    <dependency>

      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
    <!-- H2 Database -->

    <dependency>

      <groupId>com.h2database</groupId>

      <artifactId>h2</artifactId>

      <scope>runtime</scope>

    </dependency>

    <!-- Spring Boot Starter Test -->

    <dependency>

      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>

      <scope>test</scope>

    </dependency>

    <dependency>

      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>

      </plugin>

    </plugins>

  </build>
</project>
```

Application.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

Main class

```
package com.example.studentmanagement;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(StudentManagementApplication.class, args);
    }
}
```

Entity class

```
package com.example.studentmanagement;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Student {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long rollNo;
```

```
    private String name;
```

```
// Getters and Setters
```

```
public Long getRollNo() {
```

```
    return rollNo;
```

```
}
```

```
public void setRollNo(Long rollNo) {
```

```
    this.rollNo = rollNo;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
}
```


Repository interface

```
package com.example.studentmanagement;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface StudentRepository extends JpaRepository<Student, Long> {  
}
```

Controller class

```
package com.example.studentmanagement;
```

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;
```

```
import java.util.List;
```

```
@Controller
```

```
public class StudentController {
```

```
@Autowired
```

```
private StudentRepository studentRepository;
```

```
@GetMapping("/")
```

```
public String showForm(Model model) {
```

```
    model.addAttribute("student", new Student());
```

```
    return "studentForm";
```

```
}
```

```
@PostMapping("/add")
```

```
public String addStudent(@RequestParam String name, Model model) {
```

```
    Student student = new Student();
```

```
    student.setName(name);
```

```
    studentRepository.save(student);
```

```
    return "redirect:/list";
```

```
}
```

```
@GetMapping("/list")
```

```
public String listStudents(Model model) {
```

```
    List<Student> students = studentRepository.findAll();
```

```
    model.addAttribute("students", students);
```

```
    return "studentList";
```

```
}
```

```
}
```

Student List

Roll No	Name
1	praveen

[Add More Students](#)



Postman

- Postman is a powerful tool used for testing, developing, and interacting with REST APIs. It allows developers to easily send HTTP requests, examine responses, automate tests, and manage APIs during development.

Features of Postman in REST API

- **Testing Building and Sending Requests:** Postman allows us to manually construct and send HTTP requests. We can specify the HTTP method (GET, POST, PUT, DELETE, etc.), the URL, headers, parameters, and body data (in formats like JSON, XML, etc.).
- This helps in quickly testing various API endpoints to check if they're functioning as expected.

- When we are checking the RestAPI (First install the postman), we have to set the following fields

In Header tab

- Key should be “content-type”
- value should be “application/json”

In Body tab

- Choose “raw” data type and “json”

Example :CRUD operations or DB based applications with RestController annotations (or) REST API

Needed files

- Main class file
- Entity class file
- Repository interface file
- Controller file (If needed, include service file)
- Pom.xml file
- Applicaion.properties file

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>Welcomemessage</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
    <relativePath/>
  </parent>

  <dependencies>

    <!-- Spring Boot Starter for Web -->

    <dependency>

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

    </dependency>

    <!-- Spring Boot Starter for JPA -->

    <dependency>

      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
    <!-- H2 Database -->

    <dependency>

      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>

    </dependency>

    <!-- Spring Boot Starter Test -->

    <dependency>

      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>

    </dependency>

    <dependency>

      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>

      </plugin>

    </plugins>

  </build>
</project>
```


Application.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

Main class

```
package com.example.studentmanagement;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(StudentManagementApplication.class, args);
    }
}
```

Entity class

```
package com.example.studentmanagement;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Student {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long rollNo;
```

```
    private String name;
```

```
// Getters and Setters
```

```
public Long getRollNo() {
```

```
    return rollNo;
```

```
}
```

```
public void setRollNo(Long rollNo) {
```

```
    this.rollNo = rollNo;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
}
```

Repository interface

```
package com.example.studentmanagement;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface StudentRepository extends JpaRepository<Student, Long> {  
}
```

Controller class

```
package com.example.studentmanagement;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

@RestController
@RequestMapping("/students")
public class StudentController {
    @Autowired
    private StudentRepository studentRepository;

    // Add a new student
    @PostMapping("/add")
    public Student addStudent(@RequestBody Student student) {    return studentRepository.save(student);    }

    // Get all students
    @GetMapping("/list")
    public List<Student> getStudents() {    return studentRepository.findAll();    }
}
```

Overview

GET http://localhost:80...



No Environment



http://localhost:8080/students/list



Save



GET



http://localhost:8080/students/list

Send



Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON



Beautify

```
1 {  
2   "name": "Vasu"  
3 }  
4
```

Body

Cookies

Headers (5)

Test Results



200 OK

702 ms

192 B

Save Response



Pretty

Raw

Preview

Visualize

JSON



```
1 [  
2   {  
3     "rollNo": 1,  
4     "name": "Vasu"  
5   }  
6 ]
```

Runner

