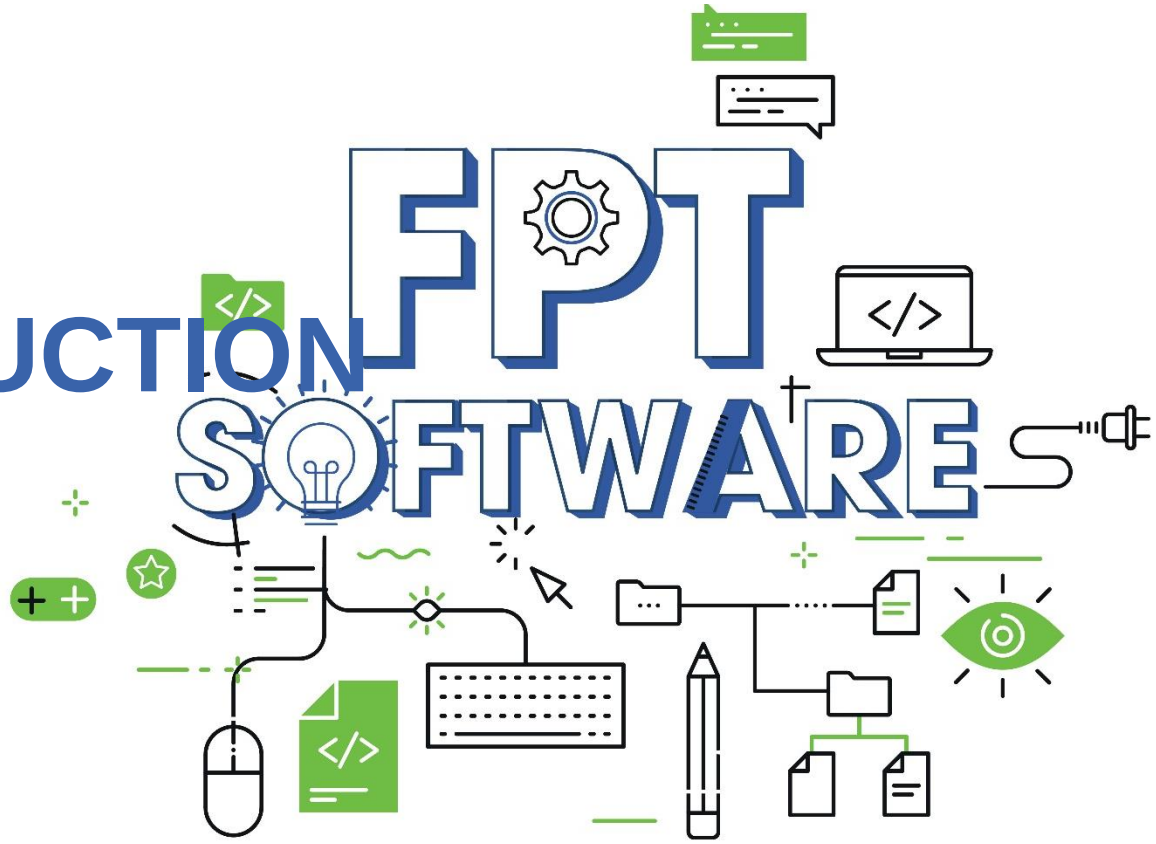
SPRING BOOT INTRODUCTION



Design by: DieuNT1

Agenda

1. Introduction and Starter templates
2. Bootstrap the Application
3. Spring Boot Annotations
4. Spring Boot Web Application
5. Question and Answer

Lesson Objectives

1. Understand Spring Boot Framework and its core technologies
2. Understand Spring Boot Starter templates
3. Able to Setting up a Spring Boot Project
3. Building and Running Spring Boot Applications

Section 1

Introduction

Introduction

Spring Boot is a Spring module that provides the **RAD** (Rapid Application Development) feature to the Spring framework.
The main motto of Spring Boot was to achieve the **Auto-Configuration feature.**

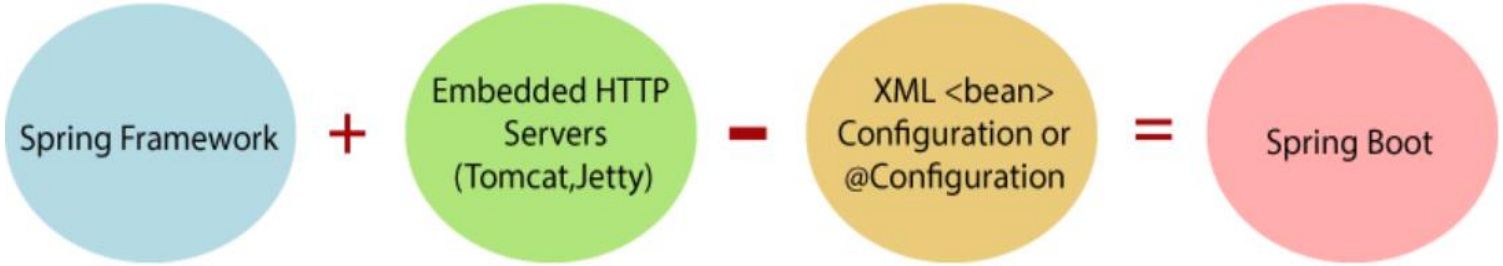
- **Spring Boot** is an **open source Java-based framework** used to create a micro Service.
  - ✓ Micro Service is an architecture that allows the developers to develop and deploy services independently.
  - ✓ Each service running has its own process and this achieves the lightweight model to support business applications.

*the Spring Boot framework was released in **April 2014** to overcome the cumbersome effort of manual configuration.*

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.

- **Spring Boot** offers the following advantages to its developers
  - ✓ Easy to understand and develop spring applications
  - ✓ Increases productivity
  - ✓ Reduces the development time



- Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.

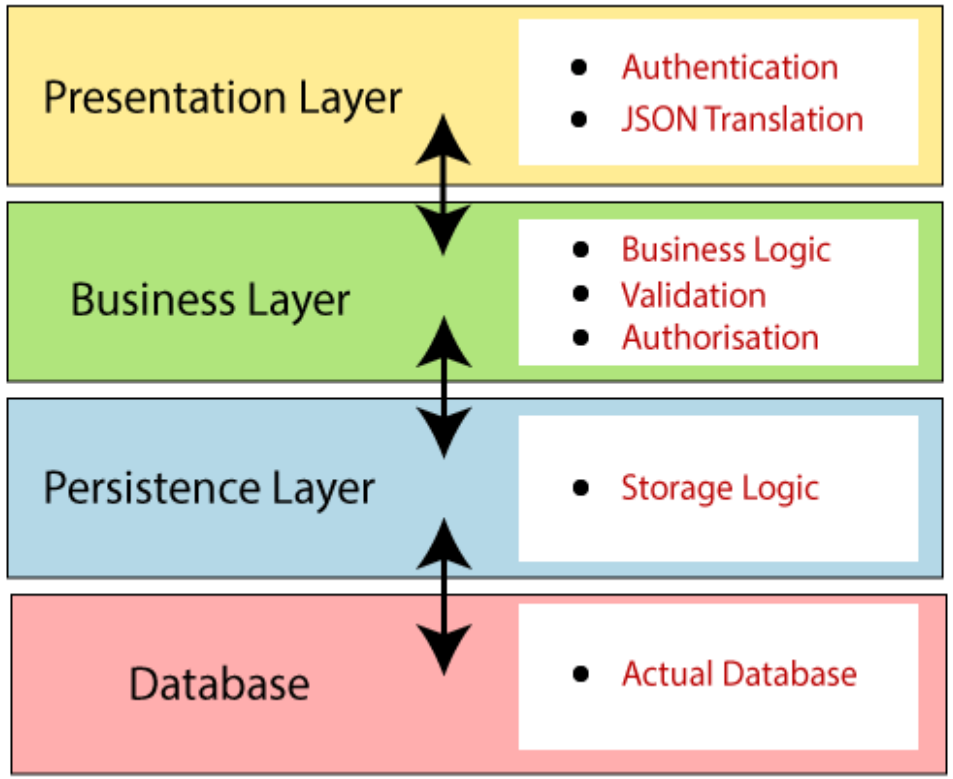
*We can use Spring **STS, IntelliJ IDEA, Eclipse IDE** or **Spring Initializr** to develop Spring Boot Java applications.*

Spring Boot Features

- Web Development
- SpringApplication
- Application events and listeners
- Admin features
- Externalized Configuration
- Properties Files
- YAML Support
- Type-safe Configuration
- Logging
- Security

Spring Boot Architecture

- **Layers in Spring Boot:** There are **four main layers** in Spring Boot:



Presentation Layer: as the name suggests, it consists of views (i.e. frontend part)

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**.

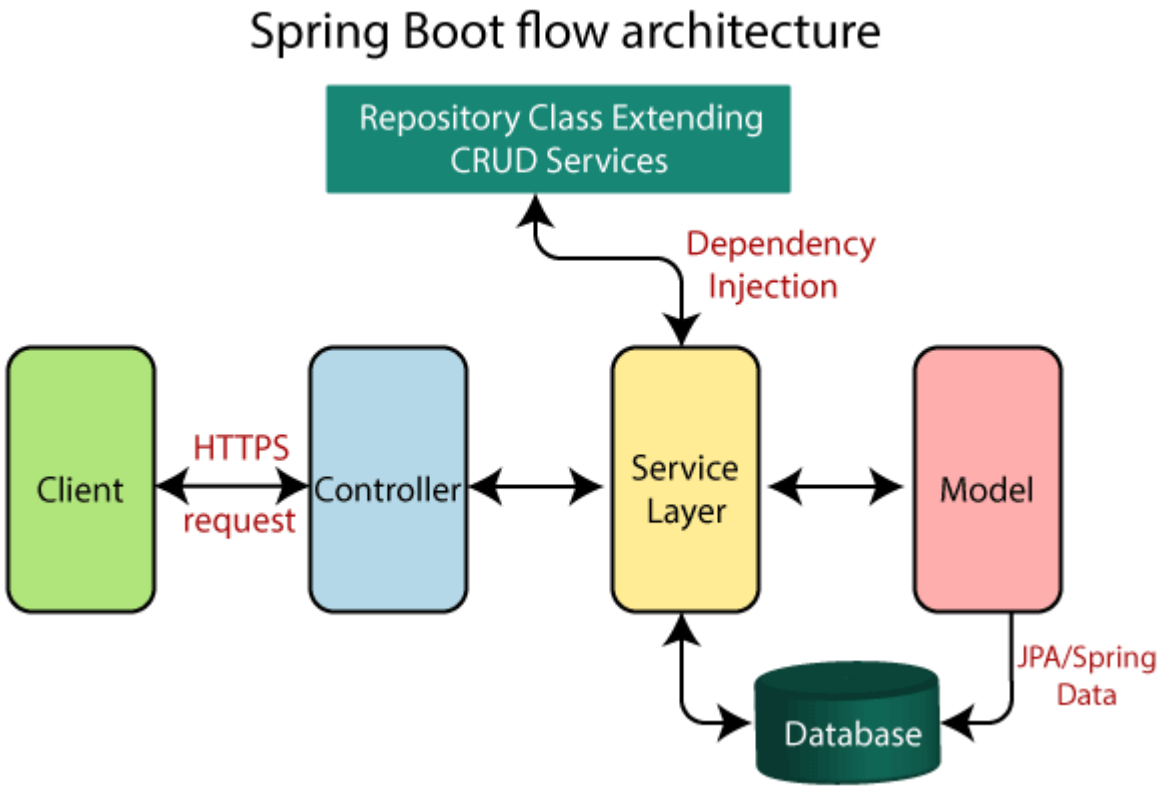
Persistence Layer: contains all the **storage logic** and translates business objects from and to database rows.

Database Layer: actual database, contains all the databases such as MySql, MongoDB, etc. **CRUD** (create, retrieve, update, delete) operations are performed.

- Then we have ***utility classes***, ***validator*** classes and ***view*** classes.

Spring Boot Architecture

- **Spring Boot flow architecture**:



- *Since Spring boot uses all the features/modules of spring-like Spring data, Spring MVC etc. so the architecture is almost the same as spring MVC, except for the fact that there is no need of **DAO** and **DAOImpl classes** in Spring boot.*
- *Creating a data access layer needs just a repository class instead which is implementing CRUD operation containing class.*

Section 2

Setup

Spring Boot Starters

- Handling dependency management is a **difficult task** for big projects.
- **What is starter template?**
  - ✓ Spring Boot starters are templates that contain a **collection of all the relevant transitive dependencies** that are needed to start a particular functionality.
  - ✓ **For example:**
    - If you want to create a Spring Web MVC application then in a traditional setup, you would have included all required dependencies yourself. It leaves the chances of **version conflict** which ultimately result in more **runtime exceptions**.

Note *that all Spring Boot starters follow the same naming pattern* ***spring-boot-starter-*** **, where * indicates that it is a type of the application.*

Spring Boot Starters

- With Spring boot, to create MVC application all you need to import is `spring-boot-starter-web` dependency.

```
<!-- Spring web brings all required dependencies to build web application. -->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Notes:

- ✓ *This dependency, internally imports all given dependencies and add to your project.*
- ✓ *Notice how some dependencies are direct, and some dependencies further refer to other starter templates which transitively downloads more dependencies.*

Spring Boot Starters

- Examples

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

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

Spring Boot Starter Parent

- The **spring-boot-starter-parent** dependency is the parent POM providing dependency and plugin management for Spring Boot-based applications.
- It contains the default *versions of Java to use*, the *default versions of dependencies* that Spring Boot uses, and the *default configuration* of the Maven plugins.

```
<!-- Parent pom is mandatory to control versions of child dependencies -->
<parent>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-parent</artifactId>
	<version>2.1.6.RELEASE</version>
	<relativePath /> <!-- lookup parent from repository -->
</parent>
```

Popular templates and their transitive dependencies

STARTER	DEPENDENCIES
spring-boot-starter	spring-boot, spring-context, spring-beans
spring-boot-starter-jersey	jersey-container-servlet-core, jersey-container-servlet, jersey-server
spring-boot-starter-actuator	spring-boot-actuator, micrometer-core
spring-boot-starter-aop	spring-aop, aspectjrt, aspectjweaver
spring-boot-starter-data-rest	spring-hateoas, spring-data-rest-webmvc
spring-boot-starter-hateoas	spring-hateoas
spring-boot-starter-logging	logback-classic, jcl-over-slf4j, jul-to-slf4j
spring-boot-starter-log4j2	log4j2, log4j-slf4j-impl
spring-boot-starter-security	spring-security-web, spring-security-config
spring-boot-starter-test	spring-test, spring-boot,junit,mockito, hamcrest-library, assertj, jsonassert, json-path
spring-boot-starter-web-services	spring-ws-core

Section 3

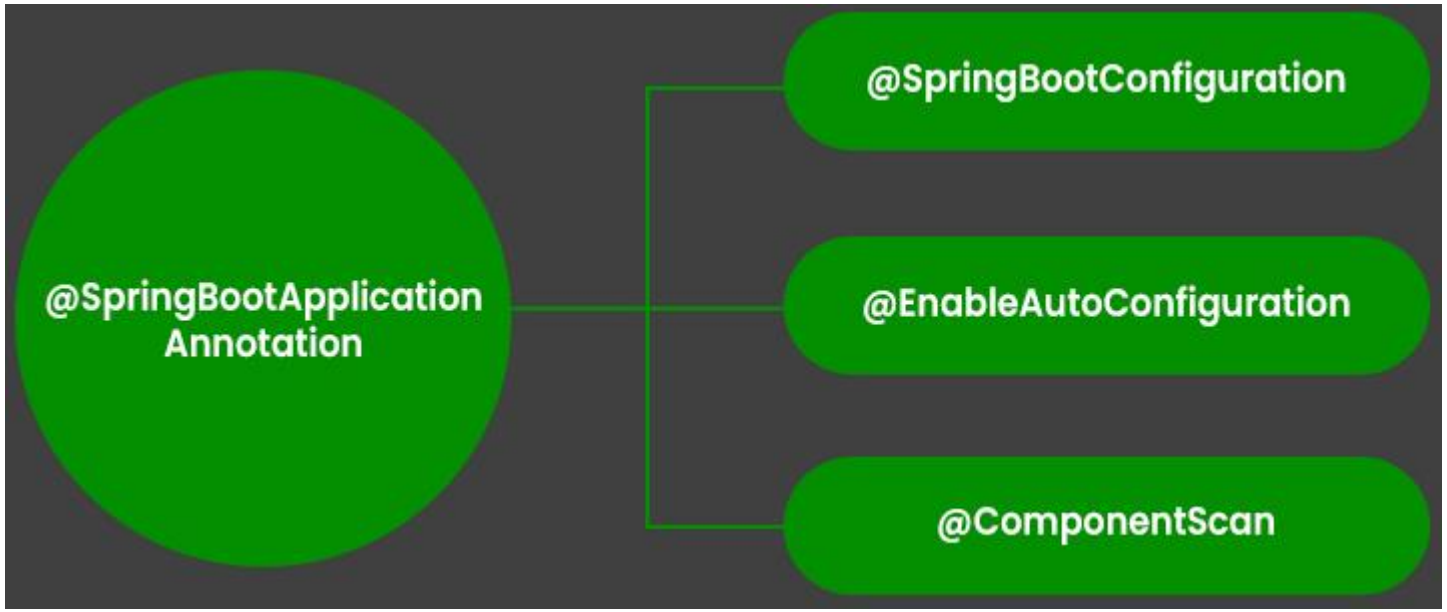
Spring Boot Annotations

Introduction

- The spring boot annotations are mostly placed in:
  - ✓ **org.springframework.boot.autoconfigure** and
  - ✓ **org.springframework.boot.autoconfigure.condition** packages.
- Some of the annotations that are available in this category are:
  - ✓ @SpringBootApplication
  - ✓ @SpringBootConfiguration
  - ✓ @EnableAutoConfiguration
  - ✓ @ComponentScan
  - ✓ Auto-Configuration Conditions

@SpringBootApplication annotation

- This annotation is used to mark the main class of a **Spring Boot application**.
- It encapsulates **@SpringBootConfiguration**, **@EnableAutoConfiguration**, and **@ComponentScan** annotations with their default attributes.



@SpringBootApplication annotation

- The Java class annotated with **@SpringBootApplication** is the main class of a Spring Boot application and application starts from here.

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

@SpringBootApplication
public class Application {

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

}
```

Bootstrap the Application

- To run the application, we need to use **@SpringBootApplication** annotation. Behind the scenes, that's equivalent[tươngđương] to **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan** together.
- It enables the **scanning of config classes**, **files** and **load** them into spring context.
- In below example, execution start with main() method. It start loading all the config files, configure them and bootstrap the application based on application properties in **application.properties** file in /resources folder.

MyApplication.java

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

application.properties

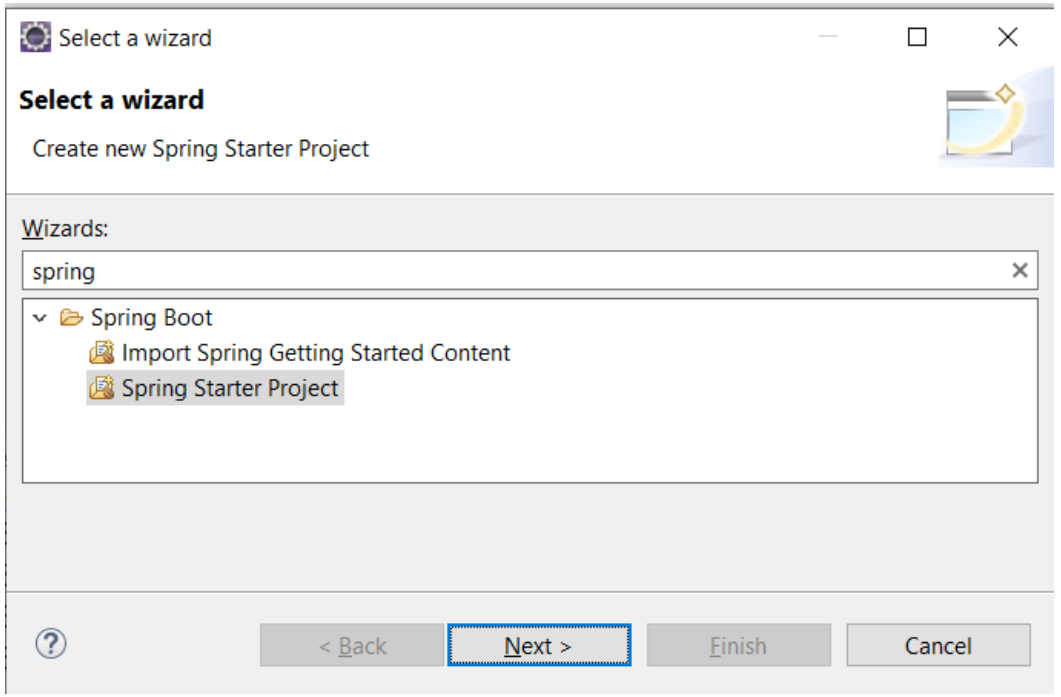
```
### Server port #########
server.port=8080

### Context root ########
server.contextPath=/home
```

Prerequisite of Spring Boot

- To create a Spring Boot application, following are the prerequisites:
  - ✓ Java 17
  - ✓ Maven 3.6+
  - ✓ Spring Boot 3+
  - ✓ Spring Framework 6+
  - ✓ IDE - Eclipse / STS / IntelliJ IDEA (or use Spring Initializr)

Create a new Project: use STS/Eclipse IDE

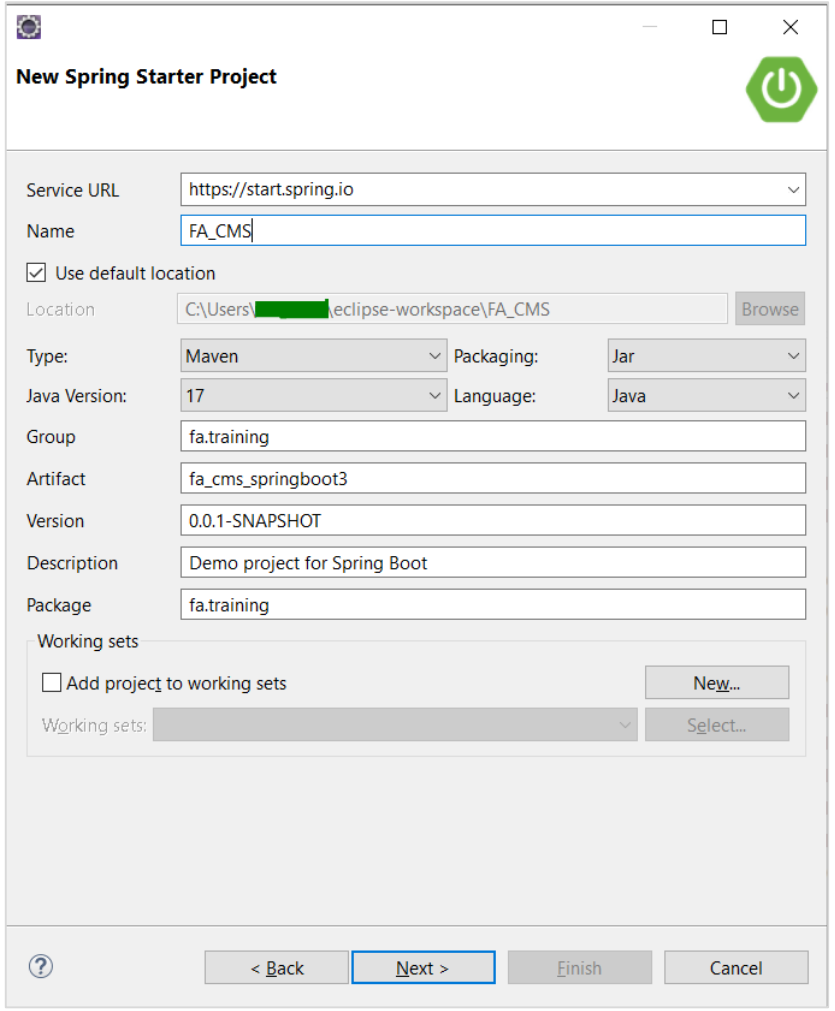
- **Step 1: Spring Starter Project:**

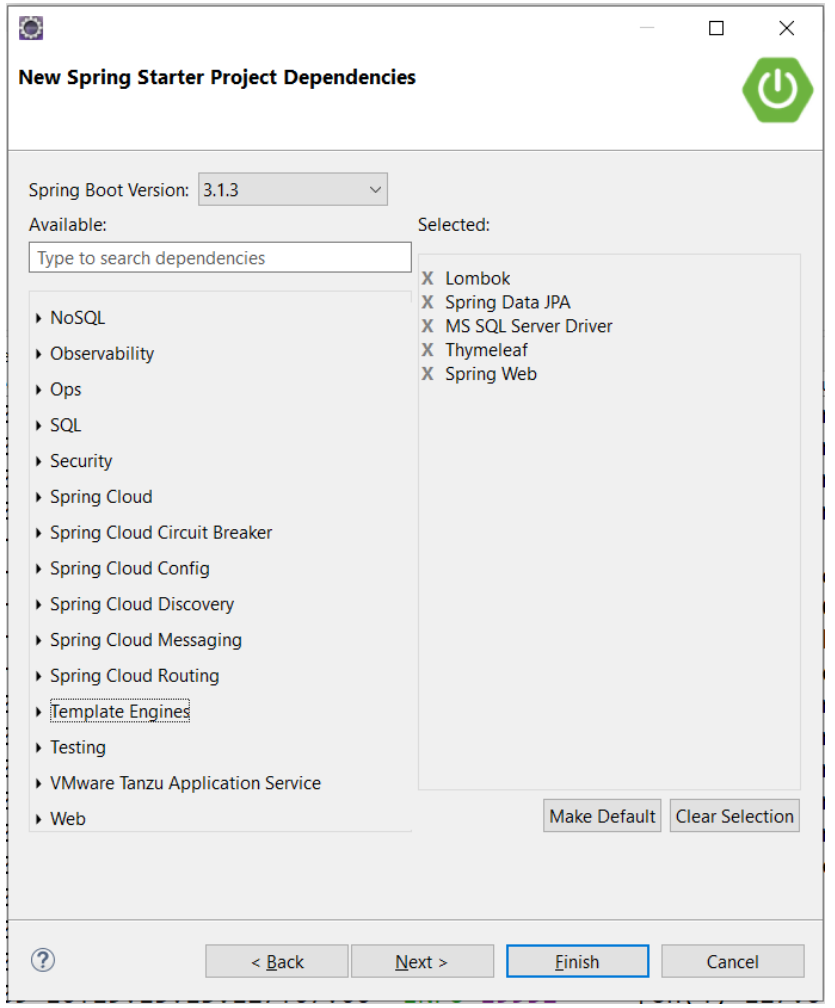


- **Step 2:** provide the name, group, and package of the project. We have provided:
  - ✓ Name: **fa_cms**
  - ✓ Group: **fa.training**
  - ✓ Package: **fa.training**

Create a new Project: use STS/Eclipse IDE

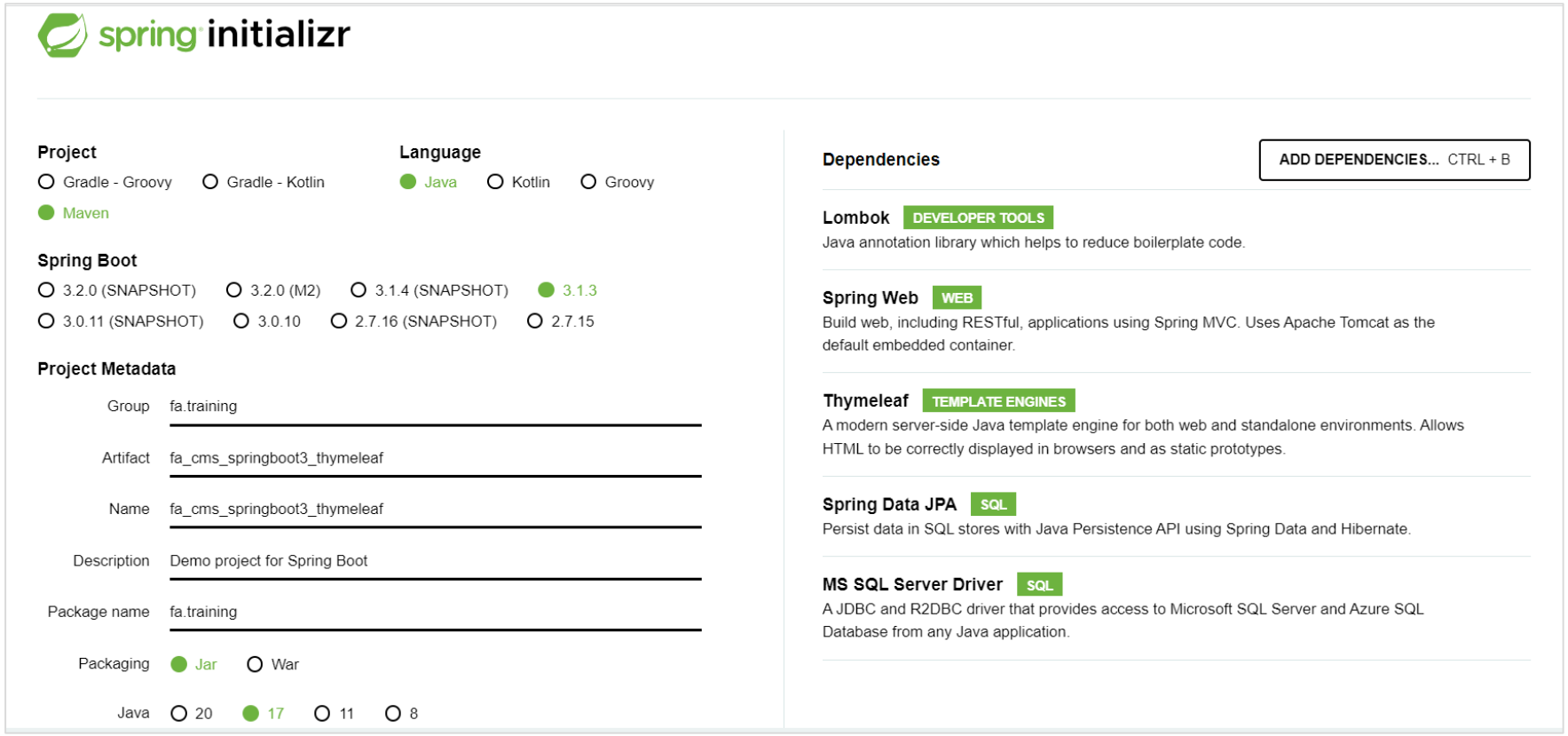
- Choose the Spring Boot Version **3.1.3** and technologies:





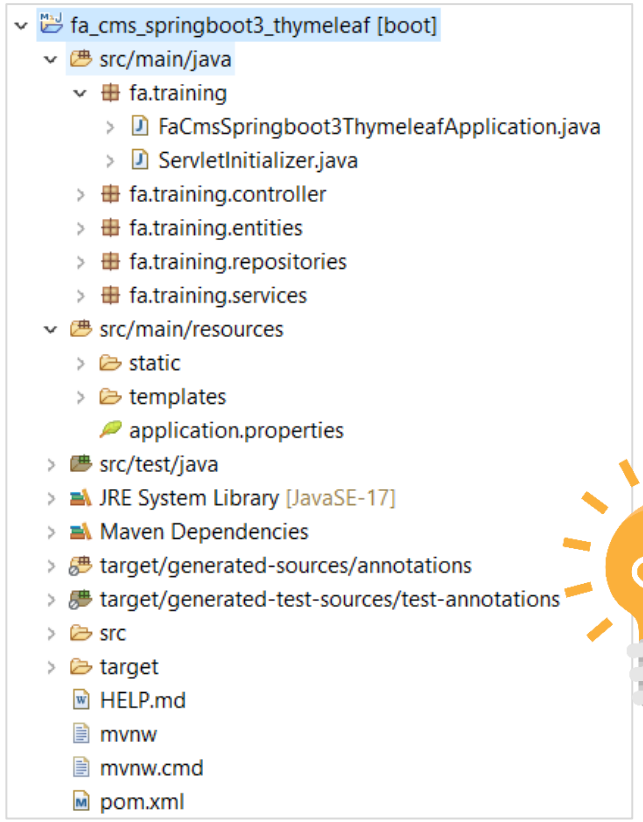
Use Spring Initializr

- The Initializr offers a fast way to pull in all the dependencies you need for an application and does a lot of the set up for you.
- This example needs the **Spring Web, Spring Data JPA**, and **MS SQL Server Driver** dependencies. The following image shows the Initializr set up for this sample project:



Project Structure

- **SpringBootApplication.java**




```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class FaCmsSpringboot3ThymeleafApplication {
public static void main(String[] args) {
        SpringApplication.run(
                FaCmsSpringboot3ThymeleafApplication.class, args);
}
}
```

Note:

- ✓ *Spring Boot auto-configuration tries to configure the beans automatically based on the dependencies added to the classpath.*
- ✓ *And since we have the JPA dependency on our classpath, Spring Boot tries to automatically configure a JPA DataSource*

Project Structure: Define the *DataSource*

- **application.properties**

```
spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.datasource.url= jdbc:sqlserver://localhost:1433;encrypt=true;
                       databaseName=FA_TRAINING;encrypt=false
spring.datasource.username= sa
spring.datasource.password= 12345678
```

- ***application.yml***:

```
spring:
        datasource:
                driverClassName: com.mysql.cj.jdbc.Driver
                url: jdbc:mysql://localhost:3306/myDb
                username: user1 password: pass
```

Project Structure: Define the *DataSource*

- We can define our data source programmatically, by **using the utility builder class *DataSourceBuilder*.**

```
@Configuration
public class DataSourceConfig {
@Bean
public DataSource getDataSource() {
        return DataSourceBuilder.create()
                .driverClassName("com.mysql.cj.jdbc.Driver")
                .url("jdbc:mysql://localhost:3306/myDb")
                .username("user1") .password("pass")
                .build();
}
}
```

Note:** if we're not yet ready to define our data source? Let's see how to **prevent Spring Boot from auto-configuring the data source.

```
@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
```

Run the launch application and check logs

- Let’s start running it with the simplest option–running as a Java application.
- In your IDE, right-click on the application class and run it as Java Application.
- For getting insight of registered beans, I have added modified the launch application as below.

```
. ____ _ __ _ _
/\\ / ___'_ __ _ _(_)_ __ __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
\\/ ___)| |_)| | | | | || (_| | ) ) ) )
' |____| .__|_| |_|_| |_\__, | / / / /
=========|_|==============|___/=/_/_/_/
[32m :: Spring Boot :: [39m [2m (v3.1.3)[0;39m
[2m2023-09-10T18:20:14.774+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mf.t.FaCmsSpringboot3ThymeleafApplication[0;39m [2m:[0;39m Starting FaCmsSpringboot3ThymeleafApplication using Java 17.0.5 wit
D:\FSOFT\Training\Training-Contents\Projects\fa_cms_springboot3_thymeleaf)
[2m2023-09-10T18:20:14.776+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mf.t.FaCmsSpringboot3ThymeleafApplication[0;39m [2m:[0;39m No active profile set, falling back to 1 default profile: "default"
[2m2023-09-10T18:20:15.200+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36m.s.d.r.c.RepositoryConfigurationDelegate[0;39m [2m:[0;39m Bootstrapping Spring Data JPA repositories in DEFAULT mode.
[2m2023-09-10T18:20:15.249+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36m.s.d.r.c.RepositoryConfigurationDelegate[0;39m [2m:[0;39m Finished Spring Data repository scanning in 42 ms. Found 2 JPA repo
[2m2023-09-10T18:20:15.596+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.s.b.w.embedded.tomcat.TomcatWebServer [0;39m [2m:[0;39m Tomcat initialized with port(s): 8080 (http)
[2m2023-09-10T18:20:15.603+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.apache.catalina.core.StandardService [0;39m [2m:[0;39m Starting service [Tomcat]
[2m2023-09-10T18:20:15.603+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.apache.catalina.core.StandardEngine [0;39m [2m:[0;39m Starting Servlet engine: [Apache Tomcat/10.1.12]
[2m2023-09-10T18:20:15.692+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.a.c.c.C.[Tomcat].[localhost].[/] [0;39m [2m:[0;39m Initializing Spring embedded WebApplicationContext
[2m2023-09-10T18:20:15.694+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mw.s.c.ServletWebServerApplicationContext[0;39m [2m:[0;39m Root WebApplicationContext: initialization completed in 877 ms
[2m2023-09-10T18:20:15.851+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.hibernate.jpa.internal.util.LogHelper [0;39m [2m:[0;39m HHH000204: Processing PersistenceUnitInfo [name: default]
[2m2023-09-10T18:20:15.888+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36morg.hibernate.Version [0;39m [2m:[0;39m HHH000412: Hibernate ORM core version 6.2.7.Final
[2m2023-09-10T18:20:15.891+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36morg.hibernate.cfg.Environment [0;39m [2m:[0;39m HHH000406: Using bytecode reflection optimizer
[2m2023-09-10T18:20:15.979+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.h.b.i.BytecodeProviderInitiator [0;39m [2m:[0;39m HHH000021: Bytecode provider name : bytebuddy
[2m2023-09-10T18:20:16.082+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.s.o.j.p.SpringPersistenceUnitInfo [0;39m [2m:[0;39m No LoadTimeWeaver setup: ignoring JPA class transformer
[2m2023-09-10T18:20:16.096+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mcom.zaxxer.hikari.HikariDataSource [0;39m [2m:[0;39m HikariPool-1 - Starting...
[2m2023-09-10T18:20:16.357+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mcom.zaxxer.hikari.pool.HikariPool [0;39m [2m:[0;39m HikariPool-1 - Added connection ConnectionID:1 ClientConnectionId: 4a2df
[2m2023-09-10T18:20:16.359+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mcom.zaxxer.hikari.HikariDataSource [0;39m [2m:[0;39m HikariPool-1 - Start completed.
[2m2023-09-10T18:20:16.377+07:00[0;39m [33m WARN[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36morg.hibernate.orm.deprecation [0;39m [2m:[0;39m HHH90000026: SQLServer2012Dialect has been deprecated; use org.hibernate.dial
[2m2023-09-10T18:20:16.537+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.h.b.i.BytecodeProviderInitiator [0;39m [2m:[0;39m HHH000021: Bytecode provider name : bytebuddy
[2m2023-09-10T18:20:17.086+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.h.e.t.j.p.i.JtaPlatformInitiator [0;39m [2m:[0;39m HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.trans
[2m2023-09-10T18:20:18.229+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mj.LocalContainerEntityManagerFactoryBean[0;39m [2m:[0;39m Initialized JPA EntityManagerFactory for persistence unit 'default'
[2m2023-09-10T18:20:18.421+07:00[0;39m [33m WARN[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mJpaBaseConfiguration$JpaWebConfiguration[0;39m [2m:[0;39m spring.jpa.open-in-view is enabled by default. Therefore, database
[2m2023-09-10T18:20:18.522+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.s.b.a.w.s.WelcomePageHandlerMapping [0;39m [2m:[0;39m Adding welcome page template: index
[2m2023-09-10T18:20:18.676+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mo.s.b.w.embedded.tomcat.TomcatWebServer [0;39m [2m:[0;39m Tomcat started on port(s): 8080 (http) with context path ''
[2m2023-09-10T18:20:18.683+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[ main][0;39m [36mf.t.FaCmsSpringboot3ThymeleafApplication[0;39m [2m:[0;39m Started FaCmsSpringboot3ThymeleafApplication in 4.191 seconds (proc
[2m2023-09-10T19:14:53.937+07:00[0;39m [33m WARN[0;39m [35m19952[0;39m [2m---[0;39m [2m[l-1 housekeeper][0;39m [36mcom.zaxxer.hikari.pool.HikariPool [0;39m [2m:[0;39m HikariPool-1 - Thread starvation or clock leap detected (housek
[2m2023-09-10T19:15:13.109+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[on(4)-127.0.0.1][0;39m [36minMXBeanRegistrar$SpringApplicationAdmin[0;39m [2m:[0;39m Application shutdown requested.
[2m2023-09-10T19:15:13.117+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[on(4)-127.0.0.1][0;39m [36mo.apache.catalina.core.StandardService [0;39m [2m:[0;39m Stopping service [Tomcat]
[2m2023-09-10T19:15:13.125+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[on(4)-127.0.0.1][0;39m [36mj.LocalContainerEntityManagerFactoryBean[0;39m [2m:[0;39m Closing JPA EntityManagerFactory for persistence unit 'de
[2m2023-09-10T19:15:13.127+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[on(4)-127.0.0.1][0;39m [36mcom.zaxxer.hikari.HikariDataSource [0;39m [2m:[0;39m HikariPool-1 - Shutdown initiated...
[2m2023-09-10T19:15:13.129+07:00[0;39m [32m INFO[0;39m [35m19952[0;39m [2m---[0;39m [2m[on(4)-127.0.0.1][0;39m [36mcom.zaxxer.hikari.HikariDataSource [0;39m [2m:[0;39m HikariPool-1 - Shutdown completed.
```

Section 4

Spring Boot Web App

What is starter template?

- Add a dependency to compile JSP files (if need):

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

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

<!-- To compile JSP files -->
<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
        <scope>provided</scope>
</dependency>

<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
</dependency>
```

What is starter template?

- Add the dependency thymeleaf if you use template engine:

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

```
<dependency>
	<groupId>org.thymeleaf.extras</groupId>
	<artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

Spring Boot JSP View Resolver

- To resolve **JSP** files location, you can have two approaches:
  - ✓ **Add entries in application.properties**

```
spring.mvc.view.prefix=/WEB-INF/view/
spring.mvc.view.suffix=.jsp

//For detailed logging during development

logging.level.org.springframework=TRACE
logging.level.com=TRACE
```

  - ✓ **Configure InternalResourceViewResolver to serve JSP pages**

```
@Configuration
@EnableWebMvc
@ComponentScan
public class MvcConfiguration implements WebMvcConfigurer {

  @Bean
  public ViewResolver viewResolver() {
    InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".jsp");
    viewResolver.setViewClass(JstlView.class);

    return viewResolver;
  }
```

Spring Boot JSP View Resolver

- **Configure to serve JSP pages:**

```
  @Override
  public void addResourceHandlers(ResourceHandlerRegistry registry) {
                  registry.addResourceHandler("/resources/**")
          .addResourceLocations("/resources/");
  }

}
```

Spring Boot Web Controller

- Create a controller class is the same as the controller class in Spring Web MVC: using **@Controller**, **@Get/@PostMapping**, **@Autowired**, ...

```
@Controller
public class InitController {

    @GetMapping("/")
    public String init(Model model) {

        model.addAttribute("user", new User());

        return "login";
    }

    @GetMapping("/index")
    public String initIndex(Model model) {

        return "index";
    }
}
```

Serving static content

- Spring Boot automatically adds static web resources located within any of the following directories:
  - ✓ /resources/
    - /static/
    - /templates/
    - /public/
- The directories are located in the classpath or in the root of the ServletContext.

❖ ***Example***: the following structure of a Web application which serves **index1.html**, **index2.html** and **index3.html** from three different locations:

```
src
├── main
│   ├── java
│   │   └── com
│   │       └── example
│   │           └── SimpleMvcSpringBootApplication.java
│   └── resources
│       ├── application.properties
│       ├── public
│       │   └── index2.html
│       ├── resources
│       │   └── index3.html
│       ├── static
│       │   └── index1.html
│       └── templates
└── test
    └── java
        └── com
            └── example
                └── SimpleMvcSpringBootApplicationTests.java
```

Serving static content

- **Using the View Controller to map URL with resources**
  - ✓ **ViewControllerRegistry** registers a View Controller.
  - ✓ It is used when we just need to map a URL with a view using **addViewController(String urlPath)**.

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
        WebMvcConfigurer.super.addViewControllers(registry);

        registry.addViewController("/savepassword").setViewName("savepassword.html");
        registry.addViewController("/login").setViewName("login.html");
        registry.addViewController("/").setViewName("loggedin/index.html");
}
```

Serving static content

- **Accessing Javascript and css in Spring Boot**
  - ✓ **CSS** and **Javascript** (.js) files are static resources and Spring Boot maps it by default in your /resources/static folder.
  - ✓ So for example, define a file **style.css** file in the folder **src/main/resources/static/css** with a minimal content:

```
h1 {
  background-color: green;
  color: red;
  text-align: center;
}
```

```
<html>
    <head>
    <link href="/css/style.css"

        rel="stylesheet">
    </head>
    <h1>Spring boot example</h1>
</html>
```

```
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── example
│   │   │           └── SimpleMvcSpringBootApplication.java
│   │   └── resources
│   │       ├── application.properties
│   │       ├── static
│   │       │   ├── css
│   │       │   │   └── style.css
│   │       │   └── index1.html
│   │       └── templates
```

Summary

- Introduction and Starter templates
- Bootstrap the Application
- Spring Boot Annotations
- Spring Boot Web App

THANK YOU!