SPRING BOOT BASICS

Instructor:

Table Content

Learning Goals

- **After the session, attendees will be able to:**

Understand Spring Boot Framework and its core technologies.

Section 1

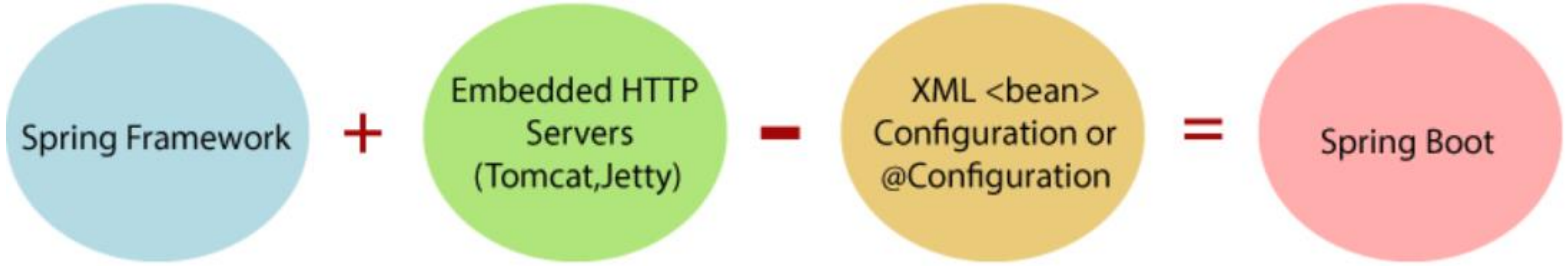
INTRODUCTION

Introduction

- **Spring Boot** is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring framework.
- **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.

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 Framework
+
Embedded HTTP Servers (Tomcat,Jetty)
-
XML <bean> Configuration or @Configuration
=
Spring Boot

- Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.
- We can use Spring **STS IDE** or **Spring Initializr** to develop Spring Boot Java applications.

Prerequisite of Spring Boot

- To create a Spring Boot application, following are the prerequisites. In this tutorial, we will use **Spring Tool Suite** (STS) IDE.
  - ✓ Java 1.8
  - ✓ Maven 3.0+
  - ✓ Spring Framework 5.0.0.BUILD-SNAPSHOT
  - ✓ An IDE (Spring Tool Suite) is recommended.
- **Spring Boot Features:**
  - ✓ Web Development
  - ✓ SpringApplication
  - ✓ Application events and listeners
  - ✓ Admin features
  - ✓ Externalized Configuration
  - ✓ Properties Files
  - ✓ YAML Support
  - ✓ Type-safe Configuration
  - ✓ Logging
  - ✓ Security

Section 2

SETUP

Spring Boot Starters

- First, let's use Spring Initializr to generate the base for our project.
- 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.

```xml
<!-- 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

```xml
<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.

```xml
<!-- 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.
- **@SpringBootApplication** annotation**:**
  - ✓ Spring boot is mostly about auto-configuration.
  - ✓ @SpringBootApplication annotation enable all able things in one step.
  - ✓ It enables the three features:
    - **@EnableAutoConfiguration**: enable auto-configuration mechanism.
    - **@ComponentScan**: enable @Component scan.
    - **@SpringBootConfiguration**: register extra beans in the context.

@SpringBootApplication annotation

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

```java
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

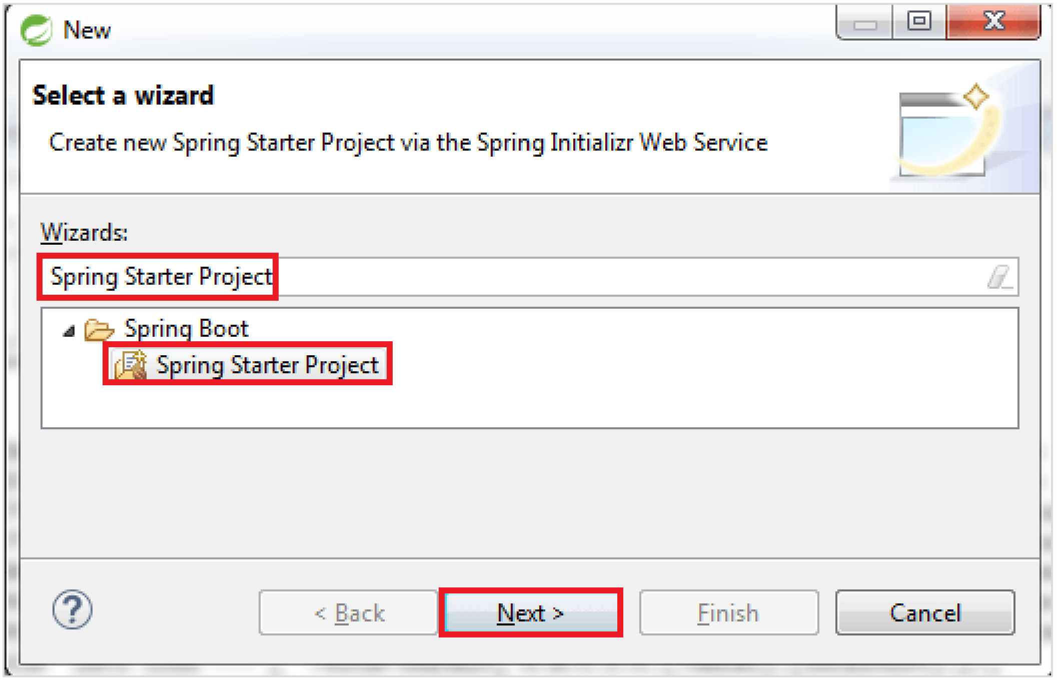
```
### Server port #########
server.port=8080

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

What You Need

- A favorite text editor or IDE
- JDK 1.8 or later
- Gradle 4+ or Maven 3.2+
- You can also import the code straight into your IDE:
  - ✓ Spring Tool Suite (STS)
  - ✓ IntelliJ IDEA
- For all Spring applications, you should start with the Spring Initializr: https://start.spring.io/

Use STS

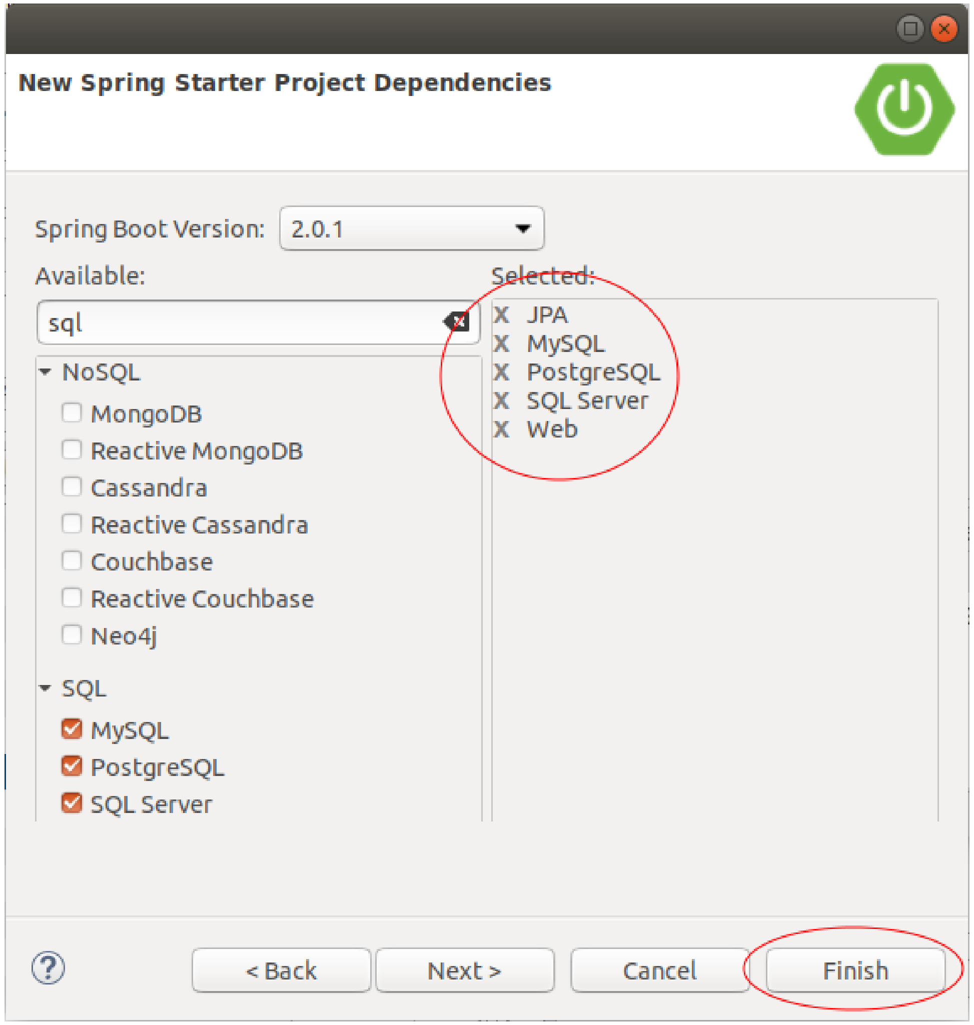
- **Step 1: Spring Starter Project:**



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

Use STS

- Choose the Spring Boot Version **2.0.1** and technologies:



New Spring Starter Project Dependencies

Spring Boot Version: 2.0.1

Available: sql

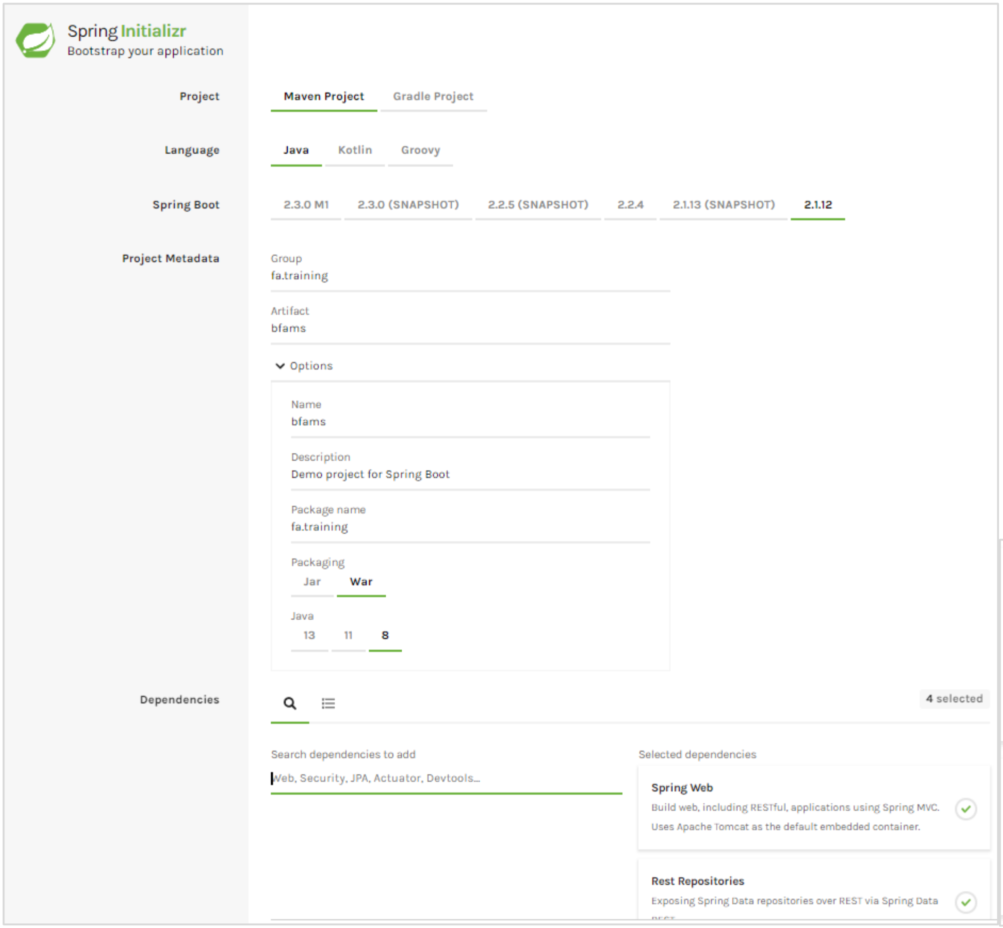
- NoSQL
  - [ ] MongoDB
  - [ ] Reactive MongoDB
  - [ ] Cassandra
  - [ ] Reactive Cassandra
  - [ ] Couchbase
  - [ ] Reactive Couchbase
  - [ ] Neo4j
- SQL
  - [x] MySQL
  - [x] PostgreSQL
  - [x] SQL Server

Selected:

X JPA
X MySQL
X PostgreSQL
X SQL Server
X Web

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 **Rest Repositories**, **Spring Data JPA**, and **H2** dependencies. The following image shows the Initializr set up for this sample project:



Spring Initializr
Bootstrap your application

Project: [✓ Maven Project] Gradle Project

Language: [✓ Java] Kotlin Groovy

Spring Boot: 2.3.0 M1 2.3.0 (SNAPSHOT) 2.2.5 (SNAPSHOT) 2.2.4 2.1.13 (SNAPSHOT) [✓ 2.1.12]

Project Metadata
Group: fa.training
Artifact: bfams
Options
Name: bfams
Description: Demo project for Spring Boot
Package name: fa.training
Packaging: Jar [✓ War]
Java: 13 11 [✓ 8]

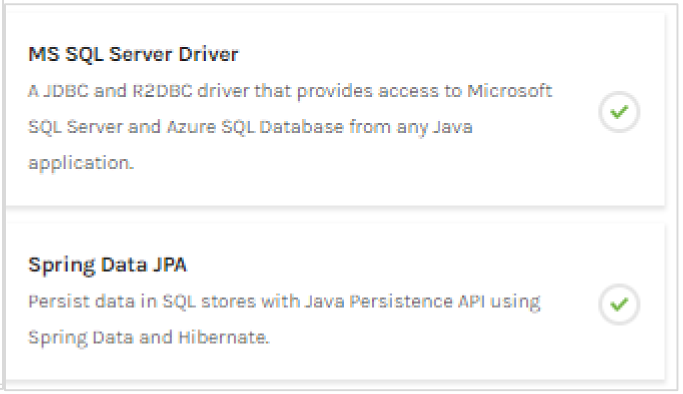
Dependencies — 4 selected

Search dependencies to add: Web, Security, JPA, Actuator, Devtools...

Selected dependencies

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

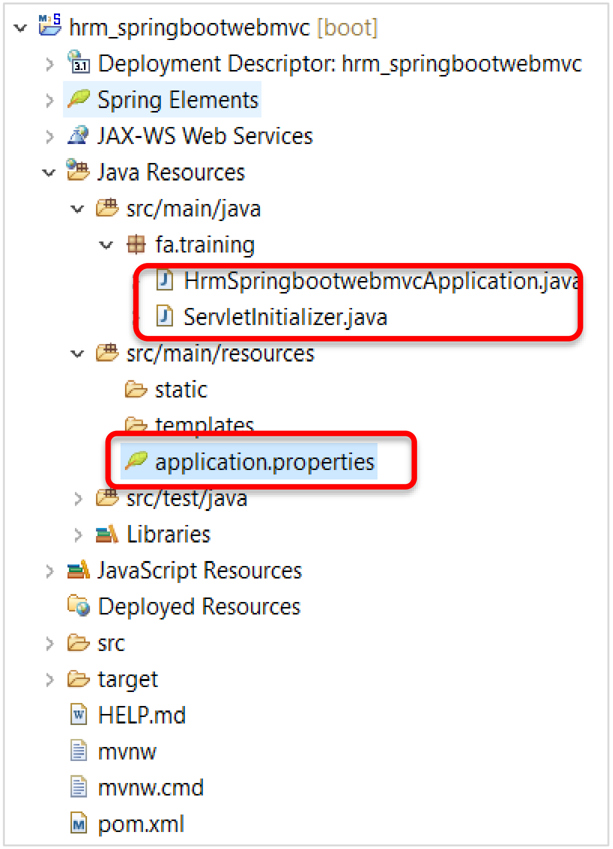
Rest Repositories
Exposing Spring Data repositories over REST via Spring Data REST.



MS SQL Server Driver
A JDBC and R2DBC driver that provides access to Microsoft SQL Server and Azure SQL Database from any Java application.

Spring Data JPA
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Project Structure



```
hrm_springbootwebmvc [boot]
  > Deployment Descriptor: hrm_springbootwebmvc
  > Spring Elements
  > JAX-WS Web Services
  v Java Resources
    v src/main/java
      v fa.training
          HrmSpringbootwebmvcApplication.java
          ServletInitializer.java
    v src/main/resources
        static
        templates
        application.properties
    > src/test/java
    > Libraries
  > JavaScript Resources
    Deployed Resources
  > src
  > target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

RestSpringBootApplication.java

```java
package fa.training;
import org.springframework.boot.SpringApplication;
Import org.springframework.boot.autoconfigure.
                                        SpringBootApplication;
@SpringBootApplication
public class Bfams2Application {
public static void main(String[] args) {         SpringApplication.
                       run(Bfams2Application.class, args);
}
}
```

application.properties

```
spring.datasource.url=jdbc:sqlserver://localhost;databaseName=FAMS1
spring.datasource.username=sa
spring.datasource.password=12345678
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.
                                                   SQLServerDriver
spring.jpa.show-sql=true
spring.jpa.hibernate.dialect=org.hibernate.dialect.SQLServer2012Dialect
spring.jpa.hibernate.ddl-auto =none
```

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.

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.0)

2020-11-20 16:46:40.403  INFO 920 --- [           main] fa.training.ServletInitializer           : Starting ServletInitializer v0.0.1-SNAPSHOT using Java 1.8.0_211 on LPP00065422A with PID 920 (C:\Users\dieunt1\ecli
2020-11-20 16:46:40.422  INFO 920 --- [           main] fa.training.ServletInitializer           : No active profile set, falling back to default profiles: default
2020-11-20 16:46:44.897  INFO 920 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2020-11-20 16:46:45.001  INFO 920 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 45 ms. Found 0 JPA repository interfaces.
2020-11-20 16:46:47.752  INFO 920 --- [           main] o.a.c.c.C.[.[.[/hrm_springbootwebmvc]    : Initializing Spring embedded WebApplicationContext
2020-11-20 16:46:47.753  INFO 920 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 7055 ms
2020-11-20 16:46:49.504  INFO 920 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2020-11-20 16:46:50.121  INFO 920 --- [         task-1] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-11-20 16:46:50.956  WARN 920 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. E
2020-11-20 16:46:51.281  INFO 920 --- [         task-1] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.23.Final
2020-11-20 16:46:55.186  INFO 920 --- [         task-1] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2020-11-20 16:46:55.747  INFO 920 --- [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: c770fe80-3a6f-4135-bcac-18dd4651db20

2020-11-20 16:46:57.510  INFO 920 --- [         task-1] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2020-11-20 16:46:57.703  INFO 920 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilte
org.springframework.security.web.authentication.logout.LogoutFilter@17c544e9, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@707172a8, org.springframework.security.web.authentic
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@146629c0, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@4bfc4be1, org.springframework.security.web.authenti
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@1fdc3182]
2020-11-20 16:46:58.289  INFO 920 --- [           main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories…
2020-11-20 16:46:58.291  INFO 920 --- [           main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2020-11-20 16:46:58.364  INFO 920 --- [           main] fa.training.ServletInitializer           : Started ServletInitializer in 20.812 seconds (JVM running for 50.503)
2020-11-20 16:46:58.950  INFO 920 --- [           main] org.apache.jasper.servlet.TldScanner     : At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete lis
2020-11-20 16:46:59.053  INFO 920 --- [           main] org.apache.coyote.ajp.AjpNioProtocol     : Starting ProtocolHandler ["ajp-nio-8009"]
2020-11-20 16:46:59.067  INFO 920 --- [           main] org.apache.catalina.startup.Catalina     : Server startup in [49,222] milliseconds
2020-11-20 16:47:01.131  INFO 920 --- [         task-1] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2020-11-20 16:47:01.803  INFO 920 --- [nio-8080-exec-2] o.a.c.c.C.[.[.[/hrm_springbootwebmvc]    : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-11-20 16:47:01.804  INFO 920 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-11-20 16:47:01.813  INFO 920 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 4 ms
2020-11-20 16:47:02.510  INFO 920 --- [         task-1] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.SQLServer2012Dialect
2020-11-20 16:47:05.947  INFO 920 --- [         task-1] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2020-11-20 16:47:06.035  INFO 920 --- [         task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
```

Section 4

SPRING BOOT WEB APP

What is starter template?

- Add a dependency to compile JSP files:

```
<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>
```

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**

```
package fa.training.controller;

@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/");
  }

}
```

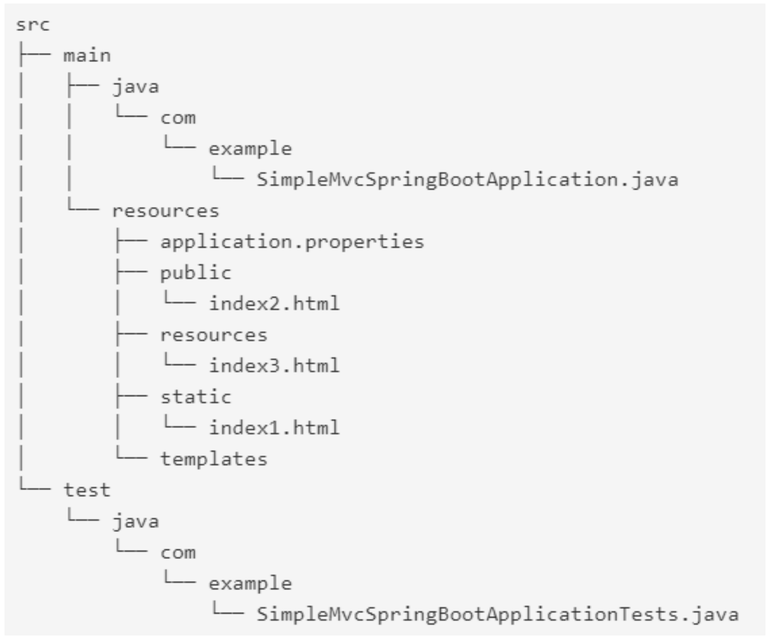
Spring Boot Web Controller

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

```java
@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:
  - ✓ /META-INF/resources/
  - ✓ /resources/
  - ✓ /static/
  - ✓ /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)**.

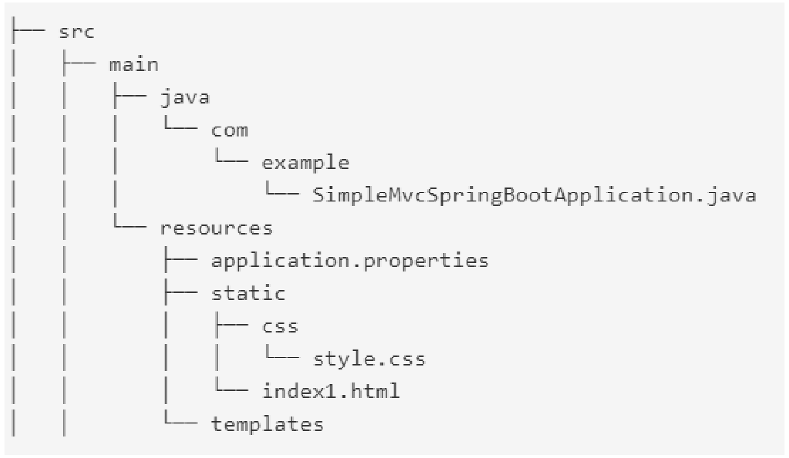
```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
        WebMvcConfigurer.super.addViewControllers(registry);

    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
```

Spring Boot Devtools

- The **spring-boot-devtools** dependency provided the auto-reload of UI in browser whenever there is change in some code.
- **pom.xml** file:

```xml
 <!-- devtools -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
```

- **Automatic UI refresh**
  - ✓ The spring-boot-devtools module includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed.
  - ✓ Precondition is that your browser should have supported extention for it.
  - ✓ By default, live reload is enabled

application.properties

```
spring.devtools.livereload.enabled  = false # Set false to disable live reload
```

SUMMAY

1
- Introduction

2
- Starter template

3
- Bootstrap the application

4
- Spring Boot Annotations

5
- Spring Boot JSP View Resolver

6
- Spring Boot Devtools

Thank you