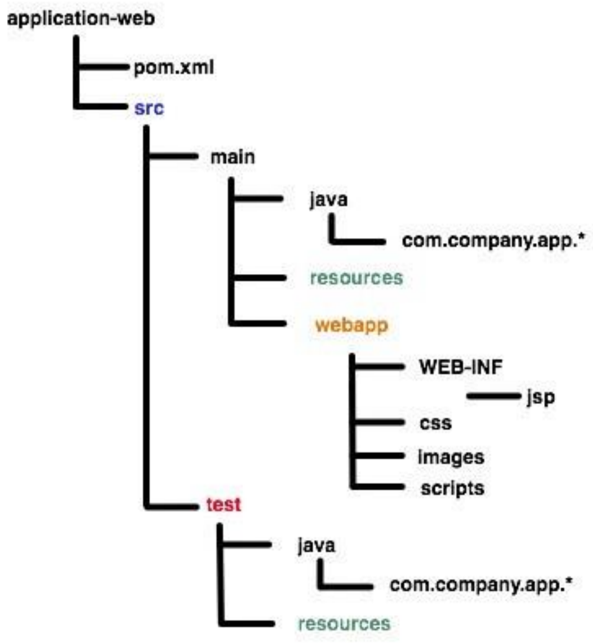
Spring MVC is one of the most popular Java framework used in developing web applications. It provides rich support for developing web applications.
Following is the structure of a Maven based project. In the root folder, there will be at least **pom.xml** (Project Object Model, which identifies the project as maven project) and *src* (Source directory). *Src* directory will contain 2 directories (*main*, *test*). *Main* will contain at least 2 (or 3 in some cases) directories (*java*, *resources*, *webapp*). *Java* will contain our packages, *resources* mostly contains property files (in some cases, static resources such as css, js, images, html files). In Spring MVC project, *webapp* directory contains mainly JSP and static resources. However, if it is a Spring Boot project, static resources and HTML files are kept in *resources* directory.



Maven Project File Structure

Open your **pom.xml** file. In *<dependencies></dependencies>* tag put the following dependencies.

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.3</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Above *<dependencies></dependencies>* tag we will include *<packaging>war</packaging>*, which means that we are creating web archive.
Below *<dependencies></dependencies>* put the following plugins.

```
<build>
   <plugins>
      <plugin>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.8.1</version>
         <configuration>
            <source>11</source>
            <target>11</target>
         </configuration>
      </plugin>
      <plugin>
         <artifactId>maven-war-plugin</artifactId>
         <version>3.2.3</version>
         <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
         </configuration>
      </plugin>
   </plugins>
</build>
```

Create a package in **src > main > java** folder. The package name will be **com.example.practice.config**. Let's create a Class **AppInitializer** which will implement **WebApplicationInitializer** class provided by Spring. We will implement **onStartup**(ServletContext) method and the code for this implementation will be like below.

```
  public class AppInitializer implements WebApplicationInitializer {

  public void onStartup(ServletContext servletCxt) {

    // ----------------- region RootContext creation and registration ---------------------
    AnnotationConfigWebApplicationContext rootContext = new AnnotationConfigWebApplicationContext();
    rootContext.register(RootConfig.class);
    rootContext.refresh();

    servletCxt.addListener(new ContextLoaderListener(rootContext));
    // ------------------ endregion RootContext creation and registration ---------------------

    // ----------------- region ServletContext creation and registration ---------------------
    AnnotationConfigWebApplicationContext servletRegisterer = new AnnotationConfigWebApplicationConte
xt();
    servletRegisterer.register(ServletConfig.class);
    ServletRegistration.Dynamic registration = servletCxt.addServlet("servlet",
        new DispatcherServlet(servletRegisterer));
    // ----------------- endregion ServletContext creation and registration ---------------------

    registration.setLoadOnStartup(1);
    registration.addMapping("/");
  }
}
```

In the above implementation we are doing 4 major things.
a) Defining and loading RootContext from RootConfig class (Which we will create a bit later).
b) Defining and loading ServletContext from ServletConfig class (Which we will create a bit later).
c) Telling the application that DispatcherServlet will load on application startup (Since, according to Spring's concept, it will have only one Servlet, DispatcherServlet to receive and process all requests.)
d) Mapping DispatcherServlet with "/" meaning that any request coming at application root URL will be received by DispatcherServlet.

Writing **RootConfig** class for components scanning for dependency injection.

```
// @ComponentScan(basePackages = {"com.example.practice.service"})
public class RootConfig { }
```

Note that currently we have no services and no package named with service, therefore our **RootConfig** class is empty and **@ComponentScan** is commented.

ServletConfig class will implement **WebMvcConfigurer** interface. This interface provides many helpful default methods. One of which is **configureViewResolvers(ViewResolverRegistry registry)**. We will override this method and introduce our jsp page folder locations (as prefix) and page extension (as suffix). We must annotate our class with **@EnableWebMvc** which will make sure that our class will be read as Web Mvc configuration class. The code is given below.

```
@EnableWebMvc
@Configuration
@ComponentScan(basePackages = {"com.example.practice.controllers"})
public class ServletConfig implements WebMvcConfigurer {

  // Configuration to render VIEWS
  public void configureViewResolvers(ViewResolverRegistry registry) {
    registry.jsp("/WEB-INF/views/", ".jsp");
  }
}
```

Create another package named **com.example.practice.controllers** and create our very first controller class there.

```
@Controller
public class RootController {

  @GetMapping("/")
  public String helloWorld(Model model) {
    model.addAttribute("name", "World");
    return "hello";
  }

  @GetMapping("/say-hello")
  public String helloName(Model model, @RequestParam(name = "name", defaultValue = "") String name) {
    model.addAttribute("name", name.isBlank() ? "World" : name);
    return "hello";
  }
}
```

A controller method must be annotated with **@Controller** annotation. Its mapped methods must be annotated with a convenient mapping annotation.

Create a **hello.jsp** page in **webapp > WEB-INF > views** directory. If there is no webapp directory, create it in **src > main** directory. In body tag of jsp page, put the following code.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Hello world</title>
</head>
<body>
<h1>Hello ${name}</h1>
</body>
</html>
```

Now you can import project in InteliJ IDEA and run it.