Proyecto Controlador-Vista

¡Comencemos!

¿Qué aprenderás?

- Crear proyecto controlador-vista que permita el despliegue de contenido estático.

Introducción

Una vez realizada toda la primera unidad de Spring Framework y haber conocido los conceptos y funcionalidades que Spring posee, podremos generar nuestro propio proyecto que permite la integración de un controlador con una vista, ambas etapas importantes consideradas en el patrón de diseño modelo, vista y controlador (MVC).

Crearemos un proyecto que permita generar contenido estático en base a un controlador realizado en Java con Spring que envíe información hacia una página html.

¡Vamos con todo!

¿Qué es un MVC?

Un Modelo Vista Controlador es una forma de organizar un software (Arquitectura de Software) que separa los datos de la aplicación, las vistas de usuario o interfaz y la lógica del aplicativo.

Estructuras de Archivos

Los archivos que iremos creando en el desarrollo de nuestra solución serán organizados bajo el siguiente estándar:

- Vistas -> *./src/main/webapp/WEB-INF/views/*
- Contenido estático -> *./src/main/resources/static/*
- Controladores -> *./src/main/java/<nombre.del.paquete>/controller/*
- Modelos -> *./src/main/java/<nombre.del.paquete>/model/*

- Servicios -> *./src/main/java/<nombre.del.paquete>/service/*

Enunciado

Se necesita crear un proyecto web que liste información desde un archivo propuesto (*data.txt*), con una lista de nombres de productos.

data.txt:

```
PRODUCTO UNO
PRODUCTO DOS
PRODUCTO TRES
```

Solución

Creamos nuestro proyecto Spring e incorporamos la dependencia web y la dependencia *devtools* (opcional). Configuramos el archivo *pom.xml* agregando una nueva dependencia que nos permitirá manejar la conexión con los archivos html.

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

El archivo *POM.xml*, debe quedar así:

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.1.5.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.proyectoFinal.productos</groupId>
   <artifactId>productos</artifactId>
```

```
   <version>0.0.1-SNAPSHOT</version>
   <name>productos</name>
   <description>Proyecto que lista productos</description>
   <properties>
      <java.version>1.8</java.version>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-thymeleaf</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-devtools</artifactId>
         <scope>runtime</scope>
         <optional>true</optional>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
      </dependency>
   </dependencies>
   <build>
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
   </build>
</project>
```

Ahora, creamos un controlador, que no es más que una clase con notación *@Controller*. Este controlador se encargará de resolver las respuestas y entregar un resultado. El archivo quedará de la siguiente manera:

```
package com.proyectoFinal.productos;
import java.io.BufferedReader;
```

```
import java.io.FileReader;
import java.util.ArrayList;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HomeController {
   private final static Logger logger =
LoggerFactory.getLogger(HomeController.class);
   @RequestMapping("/")
   public String Main(Model modelo) {
      String nombre = "src/main/resources/static/data.txt";
      ArrayList<String> p = new ArrayList<String>();
      try {
         FileReader fr = new FileReader(nombre);
         BufferedReader br = new BufferedReader(fr);
           String data = br.readLine();
           while (data != null) {
              p.add(data);
               data = br.readLine();
           }
           br.close();
           fr.close();
      } catch (Exception e) {
         logger.error("Error leyendo el fichero "+ nombre + ": " + e);
      }
      modelo.addAttribute("nombre1",p.get(0));
      modelo.addAttribute("nombre2",p.get(1));
      modelo.addAttribute("nombre3",p.get(2));
      return "main";
   }
}
```

En el archivo HomeController, observamos:

- **@Controller:** Es la anotación que denomina a la clase como tipo controlador.
- **@RequestMapping("/"):** Es la anotación que resuelve la llamada desde el navegador, en este caso está referenciando a la raíz.

- **Main(Model modelo):** Corresponde al nombre que recibe el método de este controlador, el atributo Model, se encarga de agregar atributos al modelo.

- **Modelo.addAttribute("nombre",valor):** Corresponde al método que agrega un atributo al modelo.

- **Return "main":** Resuelve la respuesta que será enviada al navegador, en este caso corresponde al archivo main.html, que se encuentra en la carpeta *template*.

Luego, creamos el archivo main.html, este responderá la llamada return "main"; y nos devolverá como texto HTML un nuevo documento procesado. Este se encargará de nuestra vista.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Productos</title>
</head>
<body>
<h1>Página de productos</h1>
<p th:text="'Producto 1: ' + ${nombre1} + '!'" />
<p th:text="'Producto 2: ' + ${nombre2} + '!'" />
<p th:text="'Producto 3: ' + ${nombre3} + '!'" />
</body>
</html>
```

Se ve que el archivo *html*, utiliza una propiedad th:text, donde esta propiedad permite listar información en pantalla, haciendo referencia al nombre del atributo enviado desde el controlador creado en Java.

Finalmente, ejecutamos nuestra aplicación, desde el método *Main*:

```
package com.proyecto.productos;
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);
   }
}
```

Y obtenemos como resultado:

Imagen 1. Resultado de productos.
Fuente: Desafío Latam.

La estructura del directorio quedó finalmente de la siguiente manera:

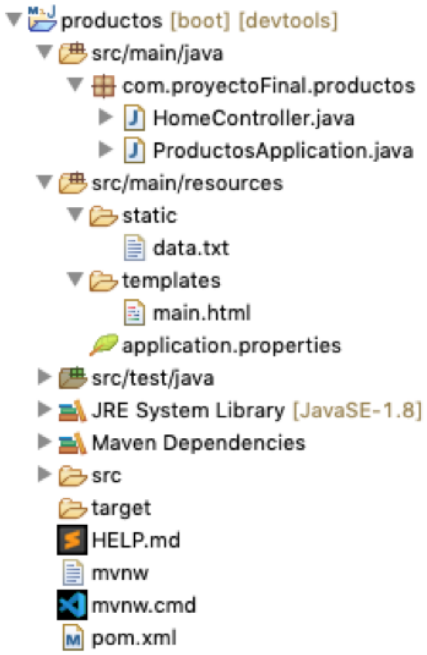

Imagen 2. Estructura final del proyecto.
Fuente: Desafío Latam.

Conclusión

Ya quedó nuestro primer proyecto web utilizando Spring. Nos dimos cuenta de que es necesario conocer varias funcionalidades nuevas que permiten que Spring maneje los modelos, las vistas y los controladores. Además de entender conceptos con los cuales se manejan ciertas tareas.