Crear Nuevo Proyecto Spring Web MVC

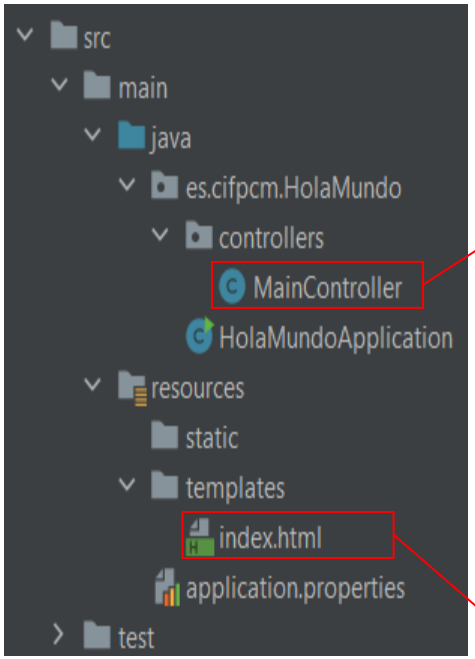
Para crear un proyecto web que soporte vistas, incluiremos las dependencias:

- Spring Web
- Thymeleaf

Generamos el archivo .zip que descomprimimos y abrimos con Intellij Idea

WAR

Spring Web MVC



MainController.java

```java
package es.cifpcm.HolaMundo.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class MainController {
    @GetMapping("/")
    public String saluda(){
        return "index";
    }
}
```

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hola Mundo</title>
</head>
<body>
    <h1>Hola a todo el Mundo</h1>
</body>
</html>
```

Spring Web MVC

Paso de datos a la vista

Para poder utilizar datos en la vista, que procedan del controlador, hay que realizar las siguientes acciones en los elementos implicados:

- Controlador:
  - Recibir por parámetro un objeto de tipo “Model”
  - Añadir un atributo al “Model” formado por un par clave-val


```
@GetMapping("/")
public String saluda(Model model){
    model.addAttribute( attributeName: "saludo", attributeValue: "Hola mi gente!");
    return "index";
}
```

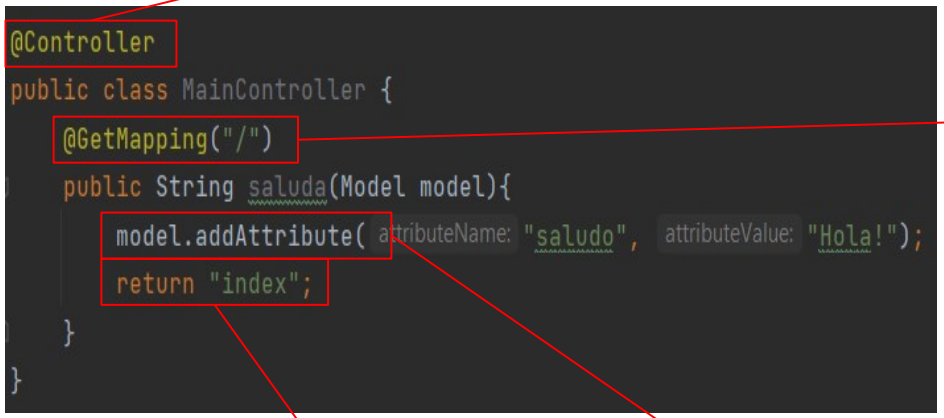
- Vista:
  - Añadir atributo (xmlns:th=”http://www.thymeleaf.org ”) en la etiqueta “html” de la plantilla
  - Incluir un atributo “th:text” en el elemento que queramos mostrar el dato

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Hola Mundo</title>
</head>
<body>
    <h1 th:text="${saludo}">Hola a todo el Mundo</h1>
</body>
</html>
```

Spring Web MVC : Controladores

Los controladores se componen principalmente de lo siguiente:

Anotación que transforma una clase simple en un controlador Spring



Anotación para métodos de un controlador que define:
- Tipo de método HTTP
- *Endpoint* de acceso

Anotaciones para otros métodos:
- GetMapping
- PostMapping
- PutMapping
- Etc.

Invocación de la vista.
Se realiza un “return” del nombre de la vista sin extensión.

Función para añadir atributos al modelo que será accesible desde la vista.
Utiliza un sistema clave-valor y pueden incluirse objetos.

Spring Web MVC : Vistas Parciales

Estilo Inclusivo

Para hacer uso de fragmentos usaremos los siguientes atributos.

- **th:fragment** : Para definir un elemento como fragmento y así poder ser utilizado en otras vistas

```
<nav th:fragment="menu">
  <!--Contenido del Nav-->
</nav>
```

- **th:replace** : El elemento que incluye este atributo se reemplaza por el fragmento referenciado. Se ha de especificar “directorio/archivo::nombre_fragmento”

```
<div th:replace="~{fragments/side::menu}"></div>
```

Spring Web MVC

Obtención de Datos de la URL: Desde un segmento de la URL

Como ya hemos visto, la URL puede presentar datos necesarios de obtener para poder realizar las acciones necesarias en los métodos de un controlador.

En este caso, el dato puede figurar tanto al final como en el medio de la URL. Ejemplo:

- /usuarios/{userId} → para buscar un determinado usuario por su Id
- /usuarios/{userId}/facturas → para buscar las facturas de un usuario en concreto

Utilizaremos la anotación @PathVariable para acceder a este dato.

La URL podría tener más de un dato a recuperar.

```
@GetMapping("/usuarios/{id}")
public String usuarios(@PathVariable String id, Model model) {
```

Spring Web MVC

Obtención de Datos de la URL: Mediante la cadena de consulta

La cadena de consulta será la parte de la URL que figura detrás del símbolo ?

Se compone por elementos clave-valor

Podrán figurar más de uno de estos pares, siempre concatenados por el símbolo &. Ejemplos:

- http://localhost:8080/?n1=v1
- http://localhost:8080/?n1=v1&n2=v2

Para recuperar estos valores utilizaremos la anotación @RequestParam de la siguiente forma:

```
localhost:8080/?name=Antonio
```

```
@GetMapping("/saludame")
public String sayHello(@RequestParam(value = "name", required=false, defaultValue = "amigo") String name, Model model) {
```

Spring Web MVC: Modelos

Enviar Modelo a la Vista

Para enviar un modelo a la vista, o un conjunto de ellos, utilizaremos la misma técnica antes vista, añadiendo un atributo con la instancia del objeto al elemento de tipo Model.

Creación de un objeto del modelo Persona

```
@GetMapping("/persona")
public String muestraPersona(Model model){

    Persona p1 = new Persona( id: 1,  name: "Antonio",  age: 24,  email: "antonio@gmail.com");
    model.addAttribute( attributeName: "persona", p1);

    return "index";
}
```

Inclusión del objeto de tipo persona en el Model

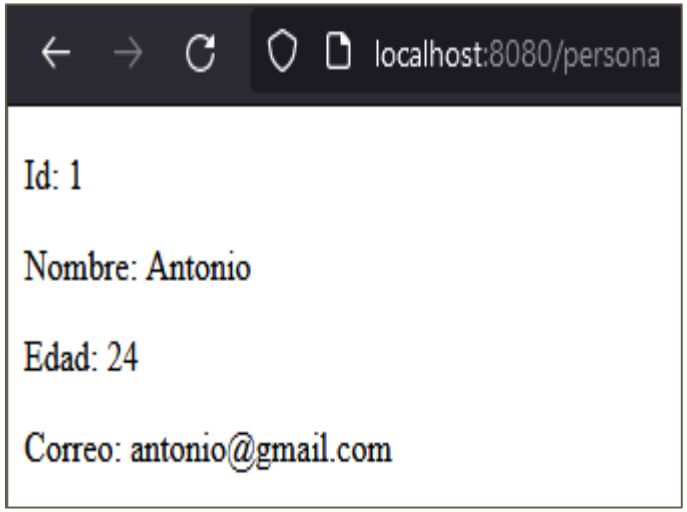
Spring Web MVC: Modelos

Mostrar Campos del Modelo en la Vista

Utilizaremos la misma técnica de ejemplos anteriores.

Para acceder a los atributos del objeto utilizaremos el operador punto (.)

```
<p th:text="${'Id: '+persona.id}"></p>
<p th:text="${'Nombre: '+persona.name}"></p>
<p th:text="${'Edad: '+persona.age}"></p>
<p th:text="${'Correo: '+persona.email}"></p>
```



Spring Web MVC: Modelos

Crear Nuevo Objeto del Modelo - 1ª Acción

Esta acción se encarga de cargar la vista con el formulario correspondiente.

Se pasará a la vista un objeto vacío de la clase del elemento a crear. De esta forma se emparejarán los campos del formulario con los de la entidad a crear.

```
@GetMapping("/persona/create")
public String crearPersona(Model model){
    model.addAttribute( attributeName: "persona", new Persona());
    return "persona/crear";
}
```

Envío del objeto vacío a la vista mediante un nuevo atributo en el Model utilizando un nombre determinado.

Se carga al vista “crear” del directorio “persona”

Spring Web MVC: Modelos

Crear Nuevo Objeto del Modelo - 1ª Acción

El formulario de la vista ha de definir principalmente:

- La acción a la que se llamará tras enviar el formulario
- Emparejar cada campo del formulario con un campo de la entidad a crear

```
<form action="#"
      th:action="@{/persona/create}"
      th:object="${persona}"
      method="post">

    <label for="name"></label>
    <input type="text" id="name" th:field="*{name}">

    <input type="submit" value="Crear">
</form>
```

Acción a invocar en el envío del formulario

Objeto a crear con el formulario y enviado en el Model

Atributo que empareja este *input* con un campo determinado de la entidad a crear

Spring Web MVC: Modelos

Crear Nuevo Objeto del Modelo - 2ª Acción

Esta acción recibe los datos del formulario en forma de modelo y realiza las acciones pertinentes en función de lo que se quiera conseguir.

En esta parte el método ha de ser de tipo Post

```
@PostMapping("/persona/create")
public String crearPersona(@ModelAttribute("persona") Persona persona){
    //En este punto ya tenemos la persona creada
    return "redirect:/";
}
```

Anotación que permite recuperar el objeto creado en el formulario

Tras realizar las acciones pertinentes con el objeto obtenido (almacenar en la BD, etc) podemos redirigir la navegación de esta forma a otra acción de algún controlador

Spring Web MVC: Modelos

Validación del Modelo en su Creación o Edición - Modelo

Como se comentó anteriormente, los campos de un modelo pueden ir acompañados de anotaciones de validación que definen restricciones a validar sobre cada uno de ellos. Pueden definir un mensaje de error personalizado.

Algunos ejemplos y uso de estas puede ser:

- @NotNull
- @NotEmpty
- @NotBlanc
- @Min y @Max
- @Size
- @Pattern
- @Email
- y hay más...

```
@NotBlank(message = "Name is mandatory")
private String name;

@NotBlank(message = "Email is mandatory")
private String email;
```

```
@NotBlank
@Size(min = 3, max = 12)
private String username;

@NotBlank
@Size(min = 6)
private String password;
```

Spring Web MVC: Modelos

Validación del Modelo en su Creación o Edición - Controlador

```
@PostMapping("/persona/create")
public String crearPersona(@Valid @ModelAttribute("persona") Persona persona,
                           BindingResult bindingResult){
    if(bindingResult.hasErrors()){
        //Modelo inválido
        return "persona/crear";
    }else{
        //Modelo válido
        return "redirect:<alguna_parte>";
    }
}
```

Spring Web MVC: Modelos

Validación del Modelo en su Creación o Edición - Formulario

Se adaptará el formulario para poder mostrar los posibles errores que fueran reportados por el controlador al validar el modelo recibido.

Utilizaremos atributos de Thymeleaf para modificar la apariencia del formulario para conseguir dos cosas:

- Añadir/quitar/omitir clases que definen estilos CSS en función de condiciones:
  - th:classappend=”${#fields.hasErrors(‘<fieldName>’)} ? ‘<clase1>’ : ‘<clase2>’”
  - Se puede usar ese operador ternario para evaluar valores de algún campo del modelo y añadir/quitar/omitir otros elementos de la vista, como los atributos. Se usaría el atributo *th:attrappend=”.....”*
- Mostrar determinados elementos de la vista para mostrar mensajes de error en caso de que exista un error con un determinado campo, usando dos atributos:
  - th:if=”${#fields.hasErrors(‘<fieldName>’)}” th:errors=”*{<fieldName>}”