Spring

Model and View

Spring: Model

- Un Model es una clase que se usa para representar la lógica de negocio a implementar.
- Como siempre toda clase debe ser nombrada en CamelCase.
- Por el momento no decoraremos con ninguna anotación. Pero cuando pasemos a la persistencia, tendremos que decorar con algunas anotaciones.

```
public class Usuario{

}
```

Spring: Model

- Todo modelo de Spring sigue el formato de clase POJO (Plain old Java Object).
- Esto quiere decir que el modelo, se define a través sus datos almacenados en atributos.
- Y el acceso a estos atributos es definido por medio de los métodos get a set.

```
public class Usuario{

    private String nombre;
    private String apellido;

    public void
setNombre(String nombre){
        this.nombre = nombre;
}

}
```

Spring: Model and View

- Podemos estos modelos pueden ser comunicados a través de la aplicación, y una parte en donde se utilizaran mucho es la vista.
- Podemos enviar el objeto completo hacia la vista con las instrucciones que ya hemos vistos para enviar datos hacia la vista.

```
@Controller
@RequestMapping("/site")
public class SiteController{

    @GetMapping("/usuario")
    public String usuario(Model
model){
        Usuario usuario = new
Usuario();
usuario.setNombre("Juan");
 model.addAttribute("usuario",
usuario);
        return "index";
    }
}
```

Spring: Model and View

- Esto datos puede ser manipulados en la vista gracias a las directivas de thymeleaf.
- Para acceder a los datos ocupamos ${}
- En este caso debemos acceder al objeto y luego a su atributo para poder operarlo o mostrarlo Ejemplo: ${usuario.nombre}

```
@Controller
@RequestMapping("/site")
public class SiteController{

    @GetMapping("/usuario")
    public String usuario(Model model){
        Usuario usuario = new Usuario();
usuario.setNombre("Juan");
 model.addAttribute("usuario", usuario);
        return "index";
    }
}
```

Spring: Model and View

- También existen otras directivas thymeleaf, para operar los datos.
- th:if=”{}”, Sirve para condicionar una acción.
- Ejemplo: <span th:if=”{usuario.apellido == null}” > no tiene apellido

```
public class Usuario{

    private String nombre;
    private String apellido;

    public void
setNombre(String nombre){
        this.nombre = nombre;
}

}
```

Spring: Model and View

- También existen otras directivas thymeleaf, para operar los datos.
- th:each=”objecto : ${lista}”, Sirve para realizar una acción por cada elemento de una lista de objetos.
- Ejemplo: Podemos crear una tabla.

```
<table>
<tr th:each=”usuario : ${usuarios}”>
    <td th:text=”usuario.nombre”> </td>
    <td th:text=”usuario.apellido”></td>
</tr>
</table>
```

```
public class Usuario{

    private String nombre;
    private String apellido;

    public void
setNombre(String nombre){
        this.nombre = nombre;
}

}
```

Spring: View and Model

- Y desde la vista la modelo, podemos enviar objetos. ¡Claro que si!

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

    <input type=”text”
th:field=”*{nombre}” />

    <input type=”text”
th:field=”*{apellido}” />

</form>
```

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

    @GetMapping(“/agregar”)
    public String
agregar(@ModelAttribute
Usuario usuario, Model model){
 model.addAttribute(“usuario”,
usuario);
        return “index”;
    }
}
```