Sistemas Distribuidos

Bloque II

Desarrollo de sistemas distribuidos y aplicaciones web.

Tema 2.2:

Spring Web y Mustache

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

- **Formulario web y Mustache**
- Condicionales y bucles
- Formulario con validación
- Operaciones CRUD sobre entidad

Formulario web y Mustache

- Página de perfil de usuario básica
- Crea una plantilla Mustache que muestre información básica de un usuario (como nombre, correo electrónico y una breve biografía).
- Pasa un objeto Usuario desde el controlador de Spring Boot a la vista Mustache y muestra sus propiedades.

Formulario web y Mustache

1. Configuración del proyecto:
   - Crea un nuevo proyecto Spring Boot con las dependencias de Spring Web y Mustache.
2. Creación del Modelo de usuario:
   - Define una clase Usuario con atributos como nombre, correo electrónico y biografía.
3. Controlador para mostrar el perfil:
   - Crea un controlador que pase un objeto Usuario a la vista.
4. Vista Mustache para el perfil de usuario:
   - Diseña una plantilla Mustache para mostrar la información del usuario.

Formulario web y Mustache

Usuario.java

```
public class Usuario {
    private String nombre;
    private String correo;
    private String biografia;
// Constructor, getters y setters
}
```

Los nombres es los atributos en la clase Usuario y los campos a los que accedemos desde Mustache tienen que tener el mismo nombre.

src/main/resources/templates/perfil.mustache

```
<!DOCTYPE html>
<html>
  <head> <title>Perfil de Usuario</title> </head>
  <body>
    <h1>Perfil de {{usuario.nombre}}</h1>
      <p>Correo: {{usuario.correo}}</p>
    <p>Biografía: {{usuario.biografia}}</p>
  </body>
</html>
```

Formulario web y Mustache

UsuarioControlador.java

```java
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class UsuarioControlador {
    @GetMapping("/perfil")
    public String perfil(Model model) {
        Usuario usuario = new Usuario("JD", "juan@example.com", "DevOps.");
        model.addAttribute("usuario", usuario);
        return "perfil";
    }
}
```

Una vez lanzada la aplicación, si accedemos al localhost:8080/perfil, deberíamos ver algo así:

Índice

- Formulario web y Mustache
- **Condicionales y bucles**
- Formulario con validación
- Operaciones CRUD sobre entidad

Condicionales y bucles

- Listado de Productos con Condicionales y Bucles:

- Utiliza Mustache para mostrar una lista de productos, cada uno con nombre, precio y disponibilidad (sí/no).
- Implementa condicionales en Mustache para mostrar un mensaje especial para productos que no estén disponibles.

Condicionales y bucles

1. Configuración del proyecto:
   - Crear un proyecto Spring Boot con las dependencias de Spring Web y Mustache.
2. Creación del Modelo de producto:
   - Definir una clase Producto con atributos como id, nombre, precio y un booleano para la disponibilidad.
3. Controlador para listar productos:
   - Crear un controlador con un método que devuelva una lista de productos, incluyendo algunos productos no disponibles.
4. Vista Mustache para mostrar productos:
   - Diseñar una plantilla Mustache para mostrar la lista de productos y usar condicionales para mostrar mensajes para productos no disponibles.

Condicionales y bucles

Producto.java

```
public class Producto {
    private Long id;
    private String nombre;
    private Double precio;
    private boolean disponible;

    //Getters, setters and constructive
}
```

ProductoControlador.java

```
@Controller
public class ProductoControlador {
    @GetMapping("/productos")
    public String listarProductos(Model model) {
        List<Producto> productos = Arrays.asList(
                new Producto(1L, "Producto 1", 10.0, true),
                new Producto(2L, "Producto 2", 15.0, false));
        model.addAttribute("productos", productos);
        return "productos";
    }
}
```

En este caso, en lugar de pasar un único elemento, se pasa una lista de ellos, en este caso de Producto

Condicionales y bucles

src/main/resources/templates/productos.mustache

```
<!DOCTYPE html>
<html>
<head> <title>Listado de Productos</title>
</head>
<body>
<h1>Productos</h1>
<ul>
  {{#productos}}
    <li>
      {{nombre}} - ${{precio}}
      {{#disponible}}
        (Disponible)
      {{/disponible}}
      {{^disponible}}
        (No disponible)
      {{/disponible}}
    </li>
  {{/productos}}
</ul>
</body>
</html>
```

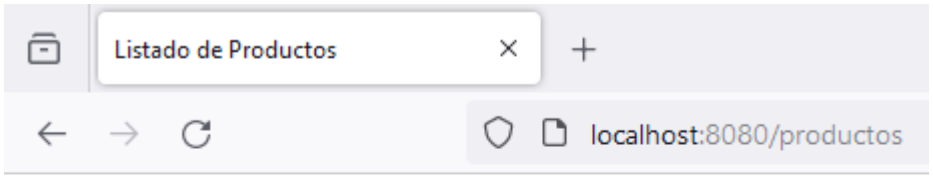
{{#productos}} {{/productos}} es una región de Mustache que funciona como un bucle. En este caso, nos mostrará todos los elementos en productos.

{{#disponible}} {{/disponible}} muestra (Disponible) si el valor es True.

{{^disponible}} es justo al revés

https://mustache.github.io/mustache.5.html

Salida:



Productos

- Producto 1 - $10.0 (Disponible)
- Producto 2 - $15.0 (No disponible)

Índice

- Formulario web y Mustache
- Condicionales y bucles
- **Formulario con validación**
- Operaciones CRUD sobre entidad

Formulario con validación

- Formulario de Contacto con Validación de Lado del Cliente:
- Crea una plantilla Mustache para un formulario de contacto con campos como nombre, correo electrónico y mensaje.
- Utiliza Mustache para generar mensajes de error o confirmación después de enviar el formulario, basándote en la respuesta del servidor.

Formulario con validación

1. Configuración del Proyecto:
   - Crear un proyecto Spring Boot con las dependencias de Spring Web y Mustache.
2. Creación de un Modelo de Mensaje:
   - Definir una clase Mensaje para representar los datos del formulario de contacto.
3. Controlador para el Formulario de Contacto:
   - Crear un controlador para manejar las solicitudes GET y POST del formulario de contacto.
4. Vista Mustache para el Formulario de Contacto:
   - Diseñar una plantilla Mustache para el formulario de contacto y agregar validación de lado del cliente.

Formulario con validación

ContactoControlador.java

```
@Controller
public class ContactoControlador {
    @GetMapping("/contacto")
    public String mostrarFormulario(Model model) {
        model.addAttribute("mensaje", new Mensaje());
        return "contacto";
    }
    @PostMapping("/contacto")
    public String procesarFormulario(Mensaje mensaje, Model model) {
        System.out.println("El mensaje de:" + mensaje.getCorreo() + " se ha mandado");
        model.addAttribute("exito", true);
        return "contacto"; }
}
```

Formulario con validación

src/main/resources/templates/contacto.mustache

```
<!DOCTYPE html>
<html>
<head>
    <title>Formulario de Contacto</title>
    <script>
        function validarFormulario() {
            var nombre = document.forms["formularioContacto"]["nombre"].value;
            var correo = document.forms["formularioContacto"]["correo"].value;
            var contenido = document.forms["formularioContacto"]["contenido"].value;
            if (nombre === "" || correo === "" || contenido === "") {
                alert("Todos los campos son obligatorios.");
                return false;
            }
            return true;
        }
    </script>
</head>
```

Mensaje.java

```
public class Mensaje {
    private String nombre;
    private String correo;
    private String contenido;
// Constructor, getters y setters
}
```

Formulario con validación

src/main/resources/templates/contacto.mustache

```
<body>
<h1>Formulario de Contacto</h1>
{{#exito}}
   <p>Mensaje enviado con éxito.</p>
{{/exito}}
<form name="formularioContacto" action="/contacto" method="post" onsubmit="return validarFormulario()">
   <label for="nombre">Nombre:</label>
   <input type="text" id="nombre" name="nombre"><br><br>
   <label for="correo">Correo electrónico:</label>
   <input type="email" id="correo" name="correo"><br><br>
   <label for="contenido">Mensaje:</label>
   <textarea id="contenido" name="contenido"></textarea><br><br>
   <input type="submit" value="Enviar">
</form>
</body>
</html>
```

Nota

- @PathVariable es una anotación utilizada en Spring.
- Sirve para extraer valores de las variables de ruta (path variables) de las URLs en las solicitudes HTTP
- Cuando defines una URL en tu controlador, puedes especificar partes de esta URL como variables de ruta. Por ejemplo, en una URL como /productos/{id}, {id} es una variable de ruta. Puedes utilizar @PathVariable para capturar el valor de {id} y usarlo en tu método del controlador.

```
@GetMapping("/productos/{id}")
public String getProducto(@PathVariable Long id, Model model) {
// Aquí, el valor de {id} en la URL se pasa al parámetro 'id' del método.
// Puedes usar 'id' para, por ejemplo, buscar un producto en una base de datos.}
```

Índice

- Formulario web y Mustache
- Condicionales y bucles
- Formulario con validación
- **Operaciones CRUD sobre entidad**

Operaciones CRUD sobre entidad

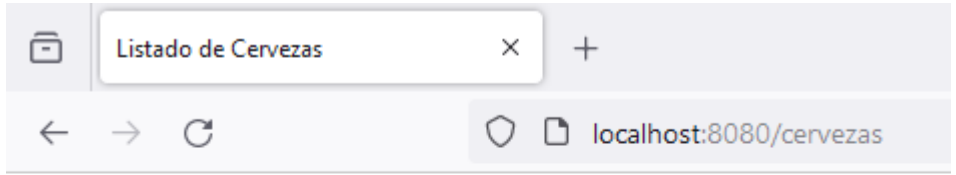
- Almacenamiento en Memoria:
  - Utilizar un mapa para almacenar las Cerveza.
- Agregar Cerveza:
  - Crear un formulario para añadir nuevas Cerveza.
  - Implementar un método en el controlador para procesar este formulario.
- Eliminar Cerveza:
  - Añadir un botón o enlace en la vista para eliminar Cerveza.
  - Implementar un método en el controlador para manejar la eliminación.
- Modificar Cerveza:
  - Crear un formulario para modificar Cerveza existentes.
  - Implementar un método en el controlador para actualizar una Cerveza.
- Mostrar todas las Cerveza
  - Mostrar un listado con todas los Cerveza (solo nombre)
  - Mostrar todas las características del Cerveza al hacer click sobre una

Operaciones CRUD sobre entidad

1. Configuración del Proyecto:
   - Crear un proyecto Spring Boot con las dependencias de Spring Web y Mustache.
2. Creación del Modelo de Cerveza:
   - Definir una clase Cerveza con atributos como id, nombre, tipo y alcohol.
3. Controlador para Operaciones CRUD de Cerveza:
   - Crear un controlador que maneje las operaciones CRUD: listar, agregar, modificar y eliminar Cerveza.
4. Vistas Mustache para Cerveza:
   - Diseñar plantillas Mustache para mostrar la lista de Cerveza y formularios para agregar y modificar

Operaciones CRUD sobre entidad

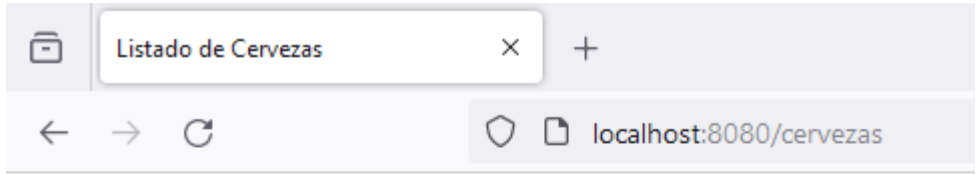
Listado de cervezas (vacío)



Cervezas Disponibles

Agregar Cerveza

Listado de cerveza

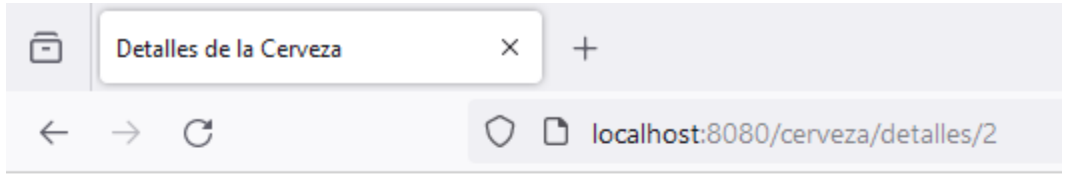


Cervezas Disponibles

- Mahou Session IPA
- Alhambra 1925

Agregar Cerveza

Detalles de la cerveza



Detalles de la Cerveza

Nombre: Alhambra 1925

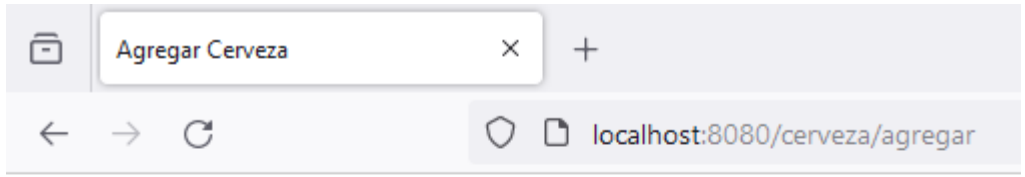
Tipo: Pilsener - Amber Lager

Alcohol: 6.4%

Editar Cerveza Eliminar Cerveza
Volver al listado

Operaciones CRUD sobre entidad

Agregar cerveza



Agregar Cerveza

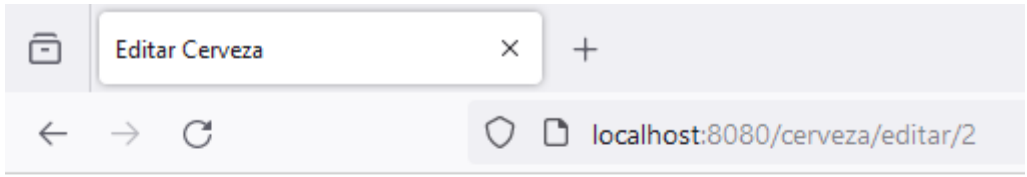
Nombre:

Tipo:

Alcohol (%):

Agregar

Editar cerveza



Editar Cerveza

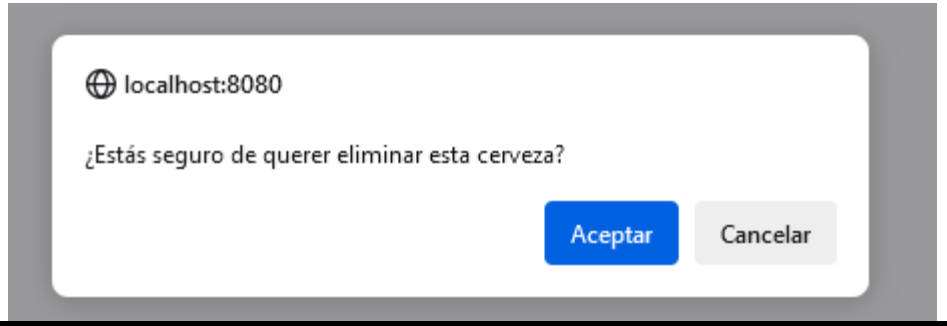
Nombre: Alhambra 1925

Tipo: Pilsener - Amber Lager

Alcohol (%): 6,4

Editar

Aviso para eliminar cerveza




```
<a href="/cerveza/eliminar/{{cerveza.id}}" onclick="return confirm('¿Estás seguro de querer eliminar esta cerveza?');">Eliminar Cerveza</a>
```

Sistemas Distribuidos

Bloque II

Desarrollo de sistemas distribuidos y aplicaciones web.

Tema 2.2:

Spring Web y Mustache

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es