Desarrollo de una Web App

SISTEMAS DISTRIBUIDOS

UNIVERSIDAD DE BURGOS

Jose Maria Santos Romero

Índice

Introducción

En el taller se ha desarrollado una pequeña web app usando las tecnologías Spring Boot, Bootstrap, JPA, Hibernate y Thymeleaf.

La web contiene una pantalla de inicio de sesión que se encarga de validar los datos introducidos, un controlador que recibe los datos y realiza solicitudes a una base de datos que contiene los datos de verificación de cada usuario.

Además también tiene un bloque de pantallas en la que se permite realizar el mantenimiento de las mascotas que están en la base de datos, esto es accesible únicamente si has realizado el login correctamente.

Básicamente se utiliza para la estructura de la aplicación el patrón Modelo-Vista-Controlador, donde el usuario a través del formulario de login envía sus credenciales que son recibidas por el controlador, este interactúa con la capa de servicio que se encarga de validar los datos, como por ejemplo que no haya ningún campo vacío y también para validar los datos, la capa de servicio interactúa con el repositorio para consultar y verificar los datos en la base de datos.

Conceptos teóricos y tecnologías utilizadas

Este proyecto es claramente un ejemplo de lo que es un sistema distribuido, ya que en este tipo de sistemas varias máquinas se comunican y coordinan sus acciones para constituir una única aplicación coherente, tal y como se está haciendo en este taller coordinando varias capas (Controlador, Servicio, Repositorio, etc.) para que el usuario pueda interactuar por ejemplo con pantalla login de manera consistente.

Se han utilizado varias tecnologías que facilitan la implementación y agilizan el desarrollo de la aplicación, estas son las tecnologías utilizadas:

- **Spring Boot**: Es un proyecto dentro del ecosistema Spring Framework que ayuda en el proceso de configuración, este detecta automáticamente las configuraciones necesarias basadas en las bibliotecas que están en el classpath, además tiene una serie de configuraciones por defecto que funcionan en la mayoría de casos y por otro lado tiene la capacidad de crear aplicaciones autónomas, que se pueden ejecutar como jar con un servidor como Tomcat.
  Se inicializa en "`LabMaven2Application.java`".

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LabMaven2Application {

    //Elementos para crear usuarios en inicio si nos interesa
    @Autowired
    private static UsersRepository usersRepository;

    @Autowired
    private static MascotasRepository mascotasRepository;

    public LabMaven2Application (UsersRepository usersRepository, MascotasRepository mascotasRepository){
        this.usersRepository = usersRepository;
        this.mascotasRepository = mascotasRepository;
    }
    public static void main(String[] args) {
        SpringApplication.run(LabMaven2Application.class, args);
        crear_usuarios();
        crear_mascotas();
    }
```

- **JPA e Hibernate**: Java Persistence API es lo que nos permite interactuar con la base de datos a través de objetos de Java y Hibernate es una implementación de JPA que ofrece varias características, como realizar consultas de datos en varios lenguajes, ORM (Mapeo Objeto-Relación) que sirve para mapear tablas de una base de datos a clases en Java y viceversa.
  La configuración de JPA e Hibernate se hace en el fichero "`application.properties`" donde se puede configurar la url de la base de datos, la estrategia DDL, las credenciales, etc. Spring Boot usa estas configuraciones para configurar automáticamente la conexión con la base de datos.

```
spring.application.name=lab_maven_2
#Data source
#Indica el driver/lib para conectar java a mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#Url donde esta el servicio de tu mysql y el nombre de la base de datos
spring.datasource.url=jdbc:mysql://localhost:53435/bbddtaller
#Usuario y contrasena para tu base de datos descrita en la linea anterior
spring.datasource.username=root
spring.datasource.password=nomelase

#[opcional]Imprime en tu consola las instrucciones hechas en tu base de datos.
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto=create-drop
server.port=8082
```

base de datos
credenciales
DDL

- **Thymeleaf**: Nos proporciona plantillas HTML para aplicaciones web desarrolladas con Java, se integra de forma natural con Spring Boot y también tiene multitud de expresiones que nos permite trabajar con objetos, manipular datos y gestionar la lógica de presentación. Thymeleaf se agrega como dependencia en nuestro fichero "`pom.xml`" y cómo se encuentra en el classpath Spring Boot lo configura automáticamente.

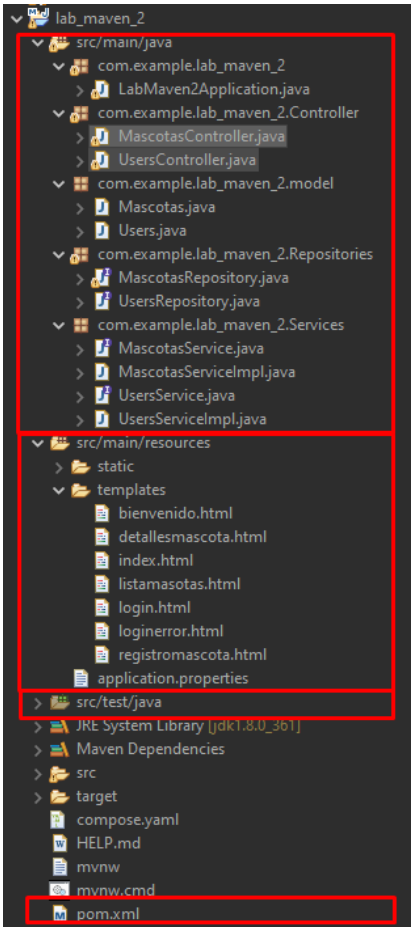
```
    <groupId>com.example</groupId>
    <artifactId>lab_maven_2</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>lab_maven_2</name>
    <description>lab_maven_2</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
```

Arquitectura del proyecto

En la construcción de la aplicación Spring MVC, estamos utilizando el patrón de diseño *Modelo-Vista-Controlador*, donde el **modelo** representa a la estructura de datos que en Spring son las entidades, las clases de servicio que trabajan sobre estos datos, las **vistas** son generadas usando Thymeleaf y Bootstrap y la parte del **controlador** es la que interactúa como intermediario entre la vista y el modelo, que en este proyecto están anotados con "`@Controller`".

En cuanto a la estructura del proyecto Maven ya la hemos estado viendo y trabajando con ella durante todo el curso, al utilizar Spring Initializer descargamos el proyecto con la estructura Maven ya creada, que podemos resumir en:

- **"`src/main/java`"**: Contiene los ficheros Java que contienen toda la lógica de la aplicación.
- **"`src/main/resources`"**: Contiene los recursos estáticos (`/static`) y las plantillas de Thymeleaf (`/templates`) y el fichero de configuración explicado anteriormente (`application.properties`).
- **"`src/test/java`"**: Contiene los tests unitarios.
- **Fichero "`pom.xml`"**: Fichero de configuración del Maven, que contiene las dependencias, plugins, etc.



En cuanto a la Base de Datos, se utiliza MySQL y en el esquema de nuestra base de datos se encuentran las tablas correspondientes a "`user`" y "`mascotas`", que podemos ver sus campos en las clases "`Users.java`" y "`Mascotas.java`".

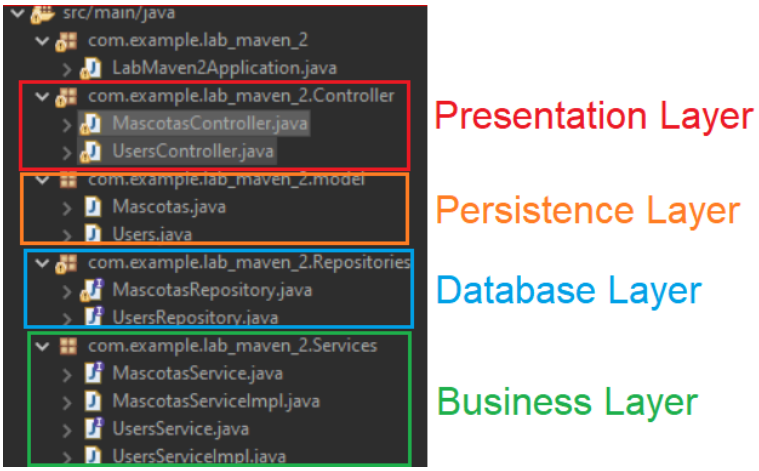
Por último la arquitectura del proyecto sigue una estructuctura típica de las aplicaciones web Spring Boot, que podemos dividirlas en 4 capas tal y como se ha visto en el taller:

- **Presentation Layer**: Esta capa se encarga de proporcionar las vistas, que han sido implementadas con Thymeleaf, también desde esta capa con los controladores anotados con "`@Controller`" en Spring Boot se encarga de realizar llamadas a la capa de servicio, que devuelven una vista (redirección a una url) o datos.

- **Business Layer**: Es conocida también como la capa de servicio, aquí se define la lógica principal de la aplicación, anotados con "`@Service`" en Spring Boot se encarga de encapsular la lógica del servicio, realizando validaciones u operaciones con los datos de los usuarios o mascotas.

- **Persistence Layer**: Esta capa está relacionada con los datos almacenados en la base de datos, maneja la lógica de almacenamiento, esta capa incluye las entidades que están anotadas con "`@Entity`" en Spring Boot, representan las tablas de la base de datos.

- **Database Layer**: Desde esta capa se realizan operaciones, los repositorios anotados con "`@Repository`" que extienden interfaces JpaRepository en Spring Boot, abstracta las operaciones CRUD (Create, Remove, Update, Delete).



Autentificación y CRUD

Para la **autenticación y gestión de usuarios** con Spring es muy sencillo, se realiza en el método "`validateAuthentication`" en "`UsersServiceImpl.java`", donde se verifica si el nombre de usuario y la contraseña proporcionados coinciden con algún registro en la base de datos, para ello la capa de servicio interactúa con la capa de base de datos, realizando una consulta en la base de datos con el método de la clase @Repository: "`userRepository.findUsersByNombreUsuarioAndPassword(username,password)`".

```
@Override
public boolean validateAuthentication(String username, String password){
    if (username == null || password == null) {
        return false;
    }
    Users user = userRepository.findUsersByNombreUsuarioAndPassword(username,password);

    return user != null;
}
```

Como se ha mencionado anteriormente un **CRUD** con site en las operaciones básicas que se aplican a la mayoría de recursos de la aplicación web, como lo son los usuarios o las mascotas en este proyecto.

Operación CRUD	Método de Petición HTTP (HTTP request method)
Create - Crear	POST
Read - Leer	GET
Update - Actualizar	PUT o PATCH
Delete - Eliminar	DELETE

Las operaciones son, pondré algunos ejemplos:

- **Crear**: A través del formulario en la página de login, el usuario puede registrarse, se hace en el método `saveUser` en `UsersServiceImpl` o `mascotasList` en `MascotasServiceImpl`.
- **Leer**: Obtienes datos de un usuario específico que se encuentra en la base de datos, se hace en métodos como `getUserByUsername`.
- **Actualizar**: Puedes modificar datos de por ejemplo de una mascota.
- **Eliminar**: Eliminar datos de la base de datos, esto se podría hacer por ejemplo en los usuarios creando un nuevo método para users o como ya se hace en `eliminarmascota` en `MascotasServiceImpl`, utiliza `deleteById`, que es una funcion del repositorio JPA.

Funcionalidad de los Sistemas Distribuidos

Para analizar cómo este proyecto con Spring Boot maneja la funcionalidad y la integración de los sistemas distribuidos, vamos a desarrollar distintas áreas que son, cómo se establece la conexión entre el cliente y servidor, cómo se interactúa con la base de datos y la seguridad y gestión de estados.

La **conexión entre el cliente y el servidor** en aplicaciones web desarrolladas con Spring Boot se realiza a través del protocolo HTTP, en el código se gestiona con los controladores anotados con @Controller que son puntos de entrada par solicitudes del cliente y se realizan en métodos con @GetMapping y @PostMapping que escuchan las solicitudes en URLs específicas, aquí un ejemplo.

```java
@GetMapping("/mascotas/listamascotas")
public String listamascotas(Model model)
{
    List<Mascotas> mascotasList = mascotasService.mascotasList();
    model.addAttribute("lista",mascotasList);
    return "listamasotas";
}
```

La **interacción con la base de datos** se realiza utilizando JPA con Hibernate como proveedor, en el código se gestiona con los repositorios anotados con @Repository que son interfaces que extienden de JpaRepository, que proporciona métodos CRUD predefinidos y tienen la capacidad de definir consultas personalizadas sin tener que definirlas manualmente, como se hace con “findUsersByNombreUsuarioAndPassword” por ejemplo, o los métodos como save, findAll o deleteById que están implementados automáticamente, aquí un ejemplo.

```java
@Repository
public interface  UsersRepository extends JpaRepository<Users, Long> {
    Users findUsersByNombreUsuario(String username);

    Users findUsersByNombreUsuarioAndPassword(String username, String password);

}
```

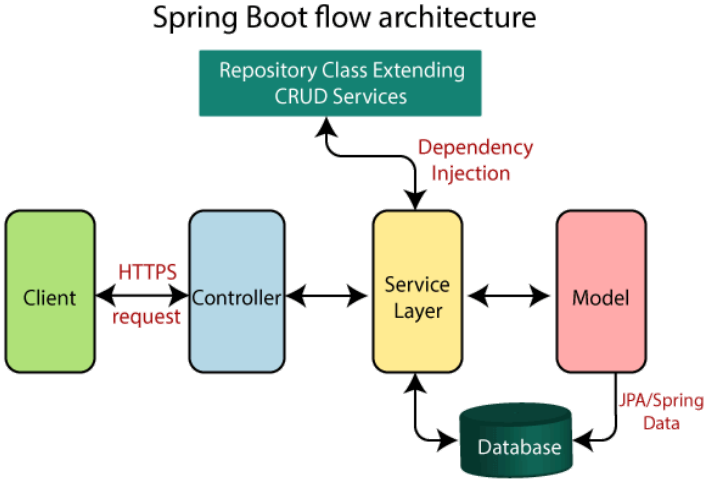
La **seguridad y gestión de estados** en una aplicación web con Spring Boot se basa en la autenticación y manejo de los estados de sesiones, que pueden ser gestionados usando sesiones HTTP o también con tokens de verificación que son enviados en cada solicitud, en el código el control de las sesiones es realizado en el controlador de usuarios, concretamente en el método `controlLogin`, aquí el ejemplo.

```java
@PostMapping("/login")
public String controlLogin(@RequestParam("username") String username,
                           @RequestParam("password") String password,
                           Model model){
    System.out.println("username: " + username);
    System.out.println("password: " + password);

    if ( usersService.validateAuthentication(username,password)){
        System.out.println("Valid username and password");
        return "binvenido";
    }
    else {
        System.out.println("Invalid username and password");
        return "loginerror";
    }

}
```

En el proyecto la integración y funcionalidad de sistemas distribuidos es correcta y está asegurada, ya que se usan patrones estándar desarrollados con Spring que incluye la arquitectura MVC y el uso de JPA con Hibernate.



Conclusión

Como conclusión me gustaría destacar el gran potencial que tiene el framework Spring Boot, ya que permite desarrollar tanto aplicaciones simples o como complejas.

La creación de este tipo de servicios cada vez está más a la orden del día y gracias a su enfoque de **desarrollo modular**, tiene una gran capacidad de escalar la aplicación según las necesidades que tenga el cliente, además también me parece una ventaja para el desarrollo de aplicaciones llevadas por empresas, ya que gracias a la **simplificación en el proceso de desarrollo** e implementación de funcionalidades no es necesario tener un gran equipo de trabajo, por lo que esto reduciría los costes de desarrollo de la aplicación.

Habiendo visto en este taller un ejemplo práctico del uso de esta herramienta, consideraría cada vez que sea posible usar Spring Boot, ya que **agiliza mucho el trabajo** a la hora de la configuración y despliegue de una aplicación, ya que viene con Tomcat embebido, multitud de métodos y configuraciones predeterminadas, **proporcionando así una base sólida sobre la que empezar a desarrollar la aplicación** con Java, teniéndote que preocupar únicamente por la lógica detrás del proyecto.

Bibliografía

Teams. Vídeos de las tutorías desarrollando la aplicación. Disponible en: https://teams.microsoft.com/l/meetup-join/19%3ameeting_NTZ…

Edgardo A. F. M. Sistemas Distribuidos. Disponible en: https://docencia.eafranco.com/materiales/sistemasoperativosii/09_Sistemas_Distribuidos.pdf

IBM. Java Spring Boot. Disponible en: https://www.ibm.com/es-es/topics/java-spring-boot

Arquitectura Java. Ejemplo de JPA. Disponible en: https://www.arquitecturajava.com/ejemplo-de-jpa/

YouTube. (*Tutorial*). Spring Boot Tutorial [Video]. Disponible en: https://www.youtube.com/watch?v=GRsBkK48NsI

OpenWebinars. ¿Qué es Thymeleaf?. Disponible en: https://openwebinars.net/blog/que-es-thymeleaf/

JavaTpoint. Spring Boot Architecture (*visto en clase*). Disponible en: https://www.javatpoint.com/spring-boot-architecture

freeCodeCamp. Operaciones CRUD: ¿Qué es CRUD?. Disponible en: https://www.freecodecamp.org/espanol/news/operaciones-crud-que-es-crud/

Tokio School. Spring Boot. Disponible en: https://www.tokioschool.com/noticias/spring-boot/