TALLER DE CREACIÓN DE UNA PEQUEÑA WEB APP USANDO TECNOLOGÍAS SPRING BOOT, JPA, HIBERNATE Y THYMELEAF

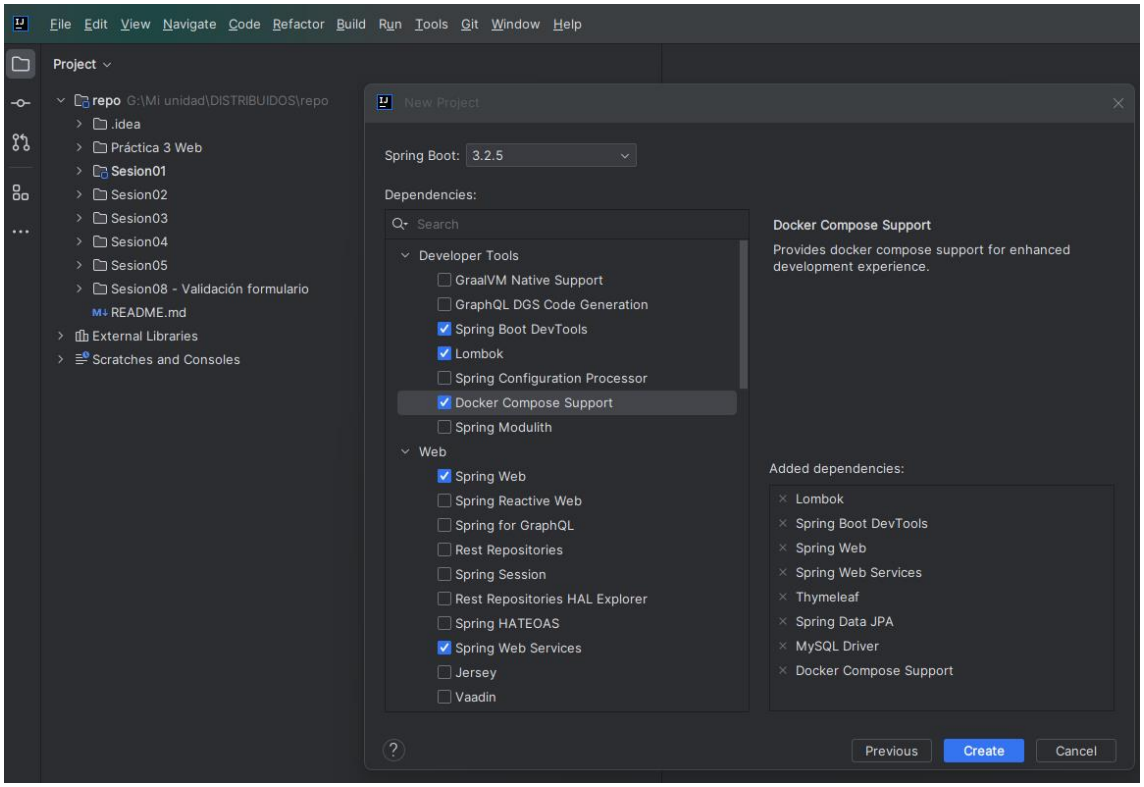
He ***creado un nuevo proyecto*** en paralelo al hecho en las tutorías para comprobar de manera personal cómo se realiza una aplicación web sencilla y así darme cuenta de los problemas que pueden surgir en la creación de éste, en vez de copiar y descifrar sin más el resultado de un proyecto dado. Además, creo que se asemeja más a la ***idea original del taller***, antes de recortar su alcance por problemas de tiempo.

Para ello he usado un modelo sencillo que en vez de mascotas ***gestionará Coches*** y que se ejecutará en Docker con las mismas premisas que el creado en clase. Como siempre, dicho proyecto estará en el repositorio: https://github.com/fps1001/SistemasDistribuidos

PASOS PARA LA CREACIÓN DEL PROYECTO

CREACIÓN DE PROYECTO BASE DESDE INTELLIJ

A diferencia de cómo construí el proyecto de la práctica 3 esta vez he decidido usar ***la interfaz de Intellij en vez de usar la web initializr***, puesto que al obtener el mismo resultado me evita tener que importar el proyecto saltándome un paso. Siendo como es, mi editor preferido para Java, Intellij facilita la creación del proyecto en el que elegiremos las siguientes dependencias:



Lombok: librería de inyección de código. Se pueden añadir anotaciones al código que generan en tiempo de compilación funciones como setters y getters evitando en exceso de código repetitivo y mejorando la legibilidad de las clases del modelo.

Spring Web / Spring Web Services: añade funcionalidades a las aplicaciones web como por ejemplo SOAP (protocolo simple de acceso a objetos) que permite no tener que implementar DAO's típicos de manera explícita.

Thymeleaf: motor de plantillas para crear vistas HTML sencillas y dinámicas que veremos a continuación.

Spring Data JPA: proporciona abstracción sobre el acceso persistente.

MySQL Driver: como datasource se usa esta base de datos.

Docker Compose Support: viene con configuración para desplegar la aplicación a través de imágenes y contenedores de Docker.

Además, usando esta modalidad tenemos acceso rápido a guías de uso de cada componente en el archivo HELP.md.

CONFIGURAR compose.yaml y APPLICATION.PROPERTIES

A continuación, configuraremos las opciones generales del proyecto modificando estos dos archivos. En primer lugar, cambiaremos el puerto externo para que coincida con las propiedades del proyecto dejando libre el puerto más comúnmente usado.

Spring.jpa.show-sql= true: muestra las sentencias SQL ejecutadas por consola, lo que ayudará a comprobar que operaciones de creación de registros y tablas está realizando el proyecto desde la consola de intellij.

Spring.jpa.hibernate.dll-auto: create/create-drop/update: determina si la base de datos eliminará las tablas al desplegar el proyecto o solo actualizará los campos de las tablas si así se determina.

En cuanto a docker hemos configurado las variables de la base de datos bajo el epígrafe environment como pueden ser la contraseña. El puerto en el que desplegará la base de datos del lado del usuario también se ha cambiado de igual manera que se hizo con el servidor de la aplicación para evitar los puertos más comunes a la hora de generar aplicaciones.

INCLUSIÓN DE PÁGINA HTML CON PLANTILLA BOOTSTRAP

Para evitar la creación y diseño manual de una página de login y de inicio del proyecto se copia una plantilla Bootstrap del mismo modo que se hizo en el taller:

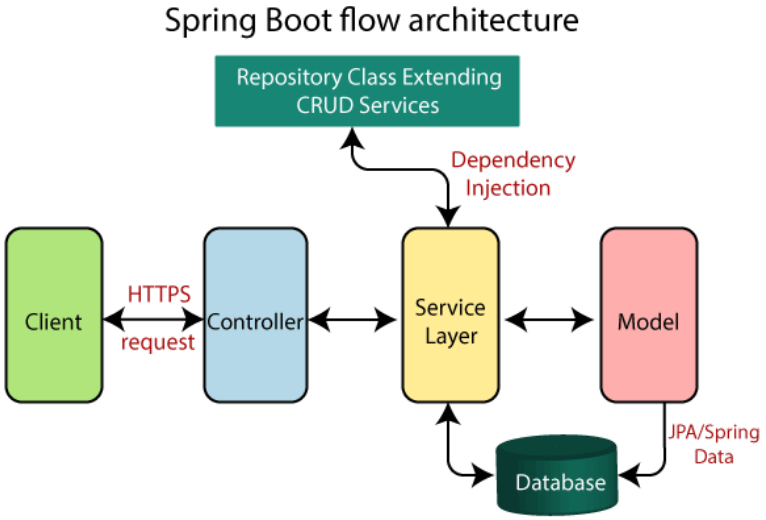
Esta será la pantalla index. Modificaré el menú superior navbar para enviar a la pantalla de login y detalle de coches, como podrá ver en el vídeo demostración de la ejecución del proyecto.

CREAR MODELO DE DATOS

He querido ir generando un proyecto en paralelo al creado en el taller, que igualmente tendrá un usuario, pero en vez de mascotas serán Coches (por aquello de ser un taller). En el gestor de clases de intellij tendría el aspecto de la imagen de la izquierda. Como ***última modificación mejoré la relación de las dos clases al incluir una relación de clase OneToMany***: un usuario puede tener n coches. Realicé la relación entre las clases modificando las vistas puesto que me parecía un diseño más realista de lo que una aplicación en desarrollo sería.

ESTRUCTURA DEL PROYECTO / INTEGRACIÓN DE CAPAS

A continuación, veremos como hemos ido construyendo las diferentes capas de las que se compone la aplicación a nivel de abstracción:

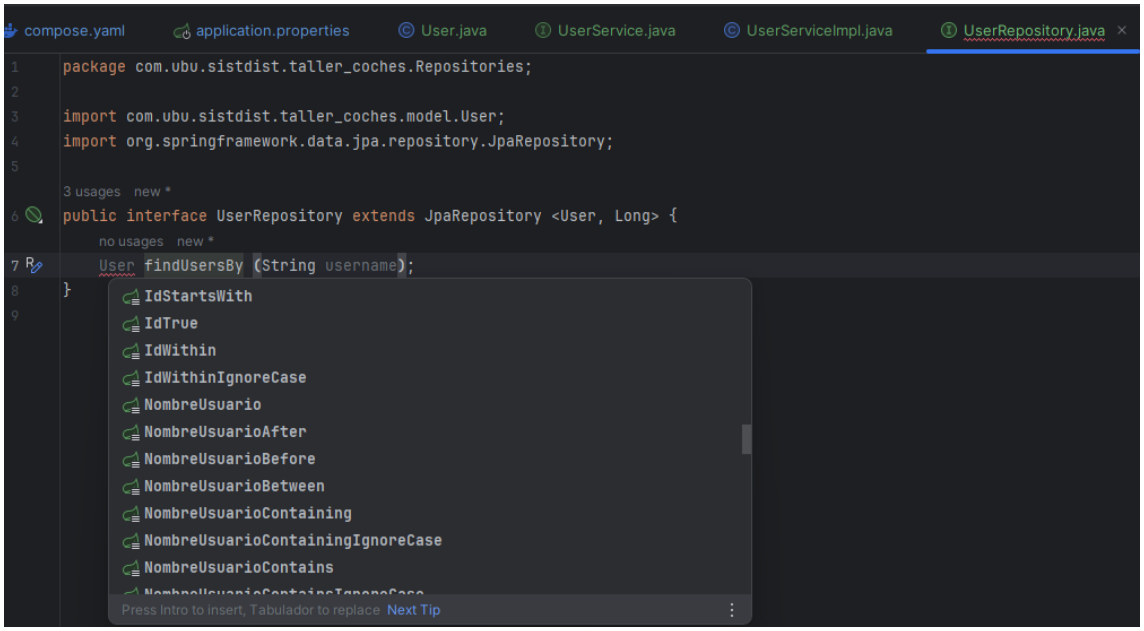


Fuente: [Spring Boot Architecture - javatpoint]

REPOSITORIO

Es la capa que accede a la base de datos basándose en la herencia de la clase JpaRepository, que es una clase genérica a la que se debe pasar una clase que queremos acceder y su clave. Esto permite acceder a base de datos sin utilizar DAOS por ejemplo y compone un set de funciones que podremos usar de manera automática como por ejemplo ***findUsersByNombreUsuarioAndPassword.***

#Indica el driver/lib para conectar java a mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver



En un futuro, al ser un sistema tan poco acoplado permite que, si se cambia el Dataset, el repositorio seguirá accediendo a los datos persistentes de la misma manera hay un que pe

El sistema ha cargado en la caché todos los campos que tiene el modelo y por debajo hará la query.

En el momento que se pone JPARepository se elimina la necesidad de DAO.

CAPA DE SERVICIO

Se encarga de la lógica de negocio: valida operaciones y es intermediaria entre las peticiones HTML de la capa de presentación y los datos, persistencia o base de datos. Es decir, a nivel de abstracción es la capa que indica ***"he interpretado todos los elementos y te voy a mandar datos".***

La capa de servicio recibe las solicitudes de la capa de presentación (por ejemplo, al loguearse un usuario) y llama a los repositorios de usuario en este caso para acceder a los datos. Luego, procesa estos datos según la lógica de negocio y devuelve el resultado a la capa de presentación.

CONTROLADOR

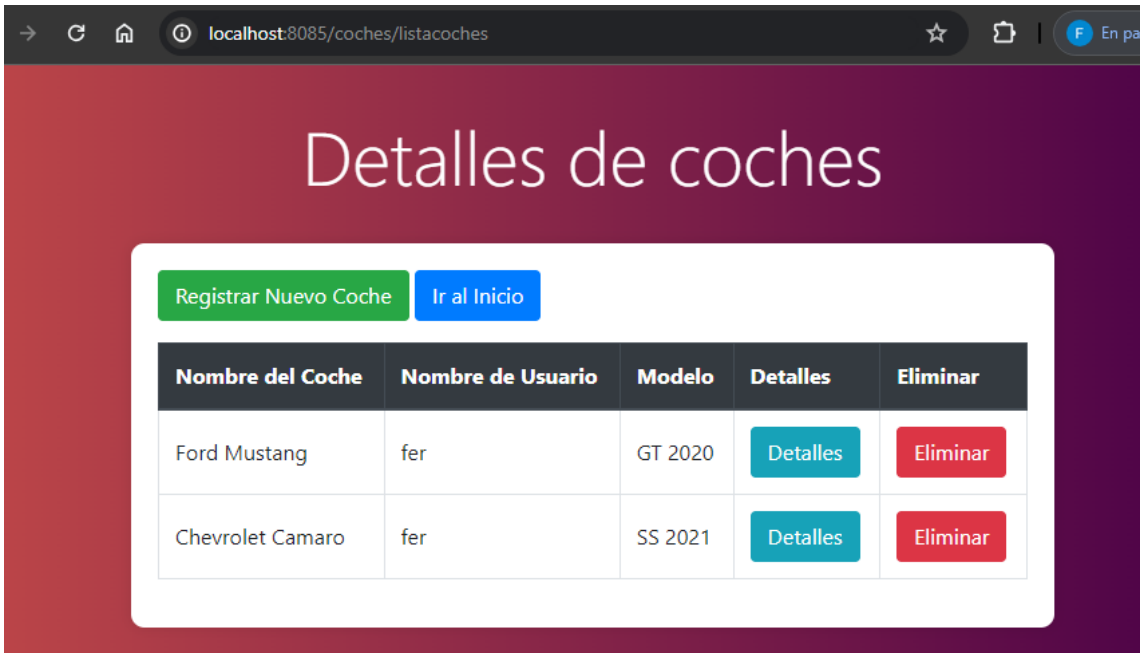
Va a recibir peticiones a través de URL's usando anotaciones @GetMapping o PostMapping o en función del tipo de petición HTML realizada por el cliente.

Por lo general el usuario va a solicitar un Get que mostrará el modelo de la pantalla que quiere acceder y tras actuar contra la página generará una petición PostMapping ***que tomará los valores y se los pasará a la capa de servicio*** que tras su tratamiento (guardado, actualización, borrado, en definitiva, operaciones CRUD): devolverá al usuario a otra vista, para lo cual también se podrá redirigir a otra pantalla.

Aclaración acerca del Modelo que se pasa como parámetro en las funciones: se trata de una estructura interna que utiliza el sistema para enviar la información que renderiza la página html que luego muestra las vistas y que vamos a usar con Thymeleaf para generar los componentes dinámicos que se mostrarán en pantalla. Por tanto, cuando en un controlador se devuelve una cadena de texto, lo que se está haciendo por debajo es enviar una página html. También puede suceder que indique un redirect: return "redirect:/usuarios" en cuyo caso en vez de a la página html se enviará a la dirección de controlador.

GENERADOR DE VISTAS: RESOURCES: STATIC/TEMPLATES/THYMELEAF

Para mejorar la parte visual de interacción con el usuario usé para la pantalla de inicio una plantilla Bootstrap y para el resto usé una serie de estilos que usan las clases de nuevo de Bootstrap usando material y que van incluidos en los archivos html (lo suyo sería dividir funcionalidades en los archivos css), pero como el alcance de la práctica así lo permite lo he dejado en los mismos archivos.



CARGA INICIAL DE DATOS EN EL ARRANQUE

Para llevarla a cabo haremos lo siguiente: como la opción comentada en el taller parece muy razonable como para no hacerla, aunque sea una aplicación en desarrollo, ***vamos a generar una variable de entorno*** en el archivo de configuración que le indique al proyecto durante el arranque si queremos añadir datos a la base de datos o no: en aplication.properties:

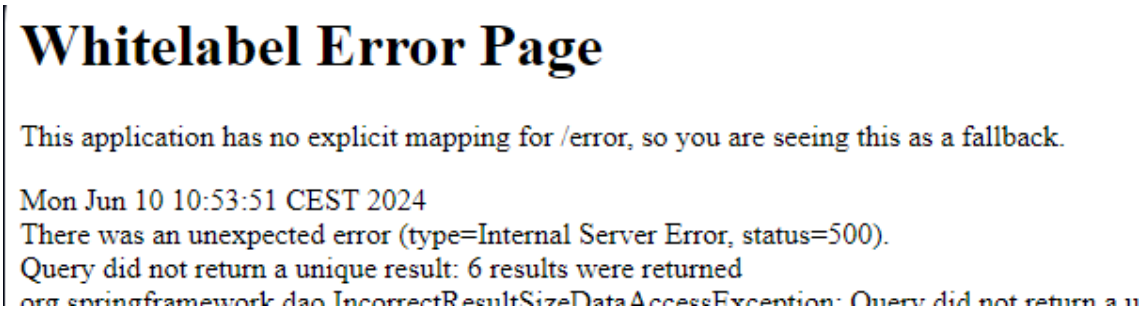
```
# Indicará si carga los valores de base de datos iniciales. 1 = carga de inicio.
execution.mode = 1
```

Después de usar la variable env como indicaba en el taller, descubrí que hay una ***anotación de spring que permite acceder al valor*** y es la siguiente:

```
@Value("${execution.mode}")
private int executionMode;
```

De tal manera que simplemente a la hora de arrancar la aplicación comprobará en un if su valor y ejecutará las funciones que graban en base de datos los valores si está a 1 dicho valor.

Es importante que spring.jpa.hibernate.ddl-auto=create-drop no esté puesto en update porque sino puede pasar lo que me ocurría a mi:

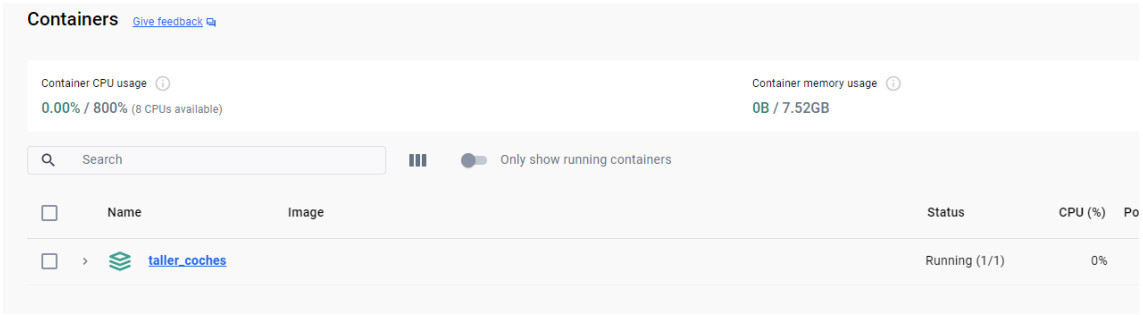


Al no haber puesto create-drop en application.properties, los valores se añadían a la base de datos sin haber borrado los datos anteriores, por lo que encontraba múltiples usuarios con la combinación usuario-contraseña y al no estar contemplada dicha opción saltaba la excepción.

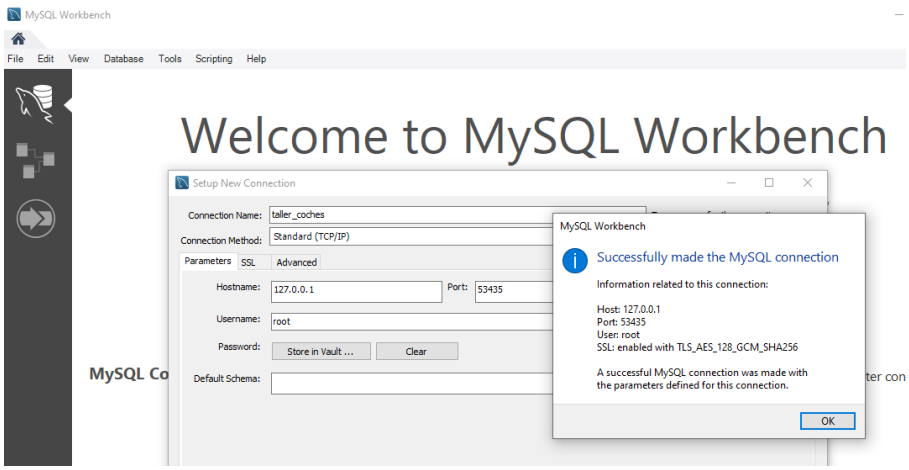
DESPLIEGUE DE LA APLICACIÓN Y COMPROBACIÓN DE FUNCIONAMIENTO

Desplegamos la aplicación y vemos en el navegador el resultado de la página de login como si de la práctica del formulario JAVA se tratase:

Para que la página se haya ejecutado se ha descargado una imagen Docker y generado un contenedor que está funcionando para dar servicio al despliegue que estamos haciendo:



Podemos también comprobar visitando el gestor de base de datos si la base de datos está levantada en el puerto indicado y si tiene grabada los datos por defecto que hemos introducido:



De hecho, en la ***práctica de Chat Web usé la pantalla de login por defecto de Web Security*** y la experiencia de usuario era mejor sin duda, aunque para simplificarla más tarde habilité el envío de contraseña sin codificar y que es la base de JPA Security. Si esta aplicación fuese un proyecto real o un TFG sin duda utilizaría no solo por seguridad sino por funcionamiento por defecto esta dependencia.

Si se ejecuta el método save del repositorio y el objeto pasado por parámetro no tiene un id válido se generará uno nuevo y el sistema asignará un id nuevo válido. Si el atributo id es válido lo que hace es actualizarlo.

VENTAJAS DEL USO DEL FRAMEWORK SPRINGBOOT Y JUSTIFICACIÓN DEL PROYECTO

UNIFICACIÓN DE ENTORNO DE TRABAJO Y FORMA DE TRABAJAR

En primer lugar, al usar Maven y Docker se estandarizan las dependencias y se trabaja con un marco que es común a cualquier programador, ordenador o equipo usando siempre las herramientas provistas por el inicializador de proyecto.

Además, y cómo pude comprobar en la práctica 3 Spring/Springboot tiene una manera de funcionar de manera invisible al programador que automatiza mecanismos por ejemplo que el controlador retorne una cadena de texto significa que esa ***cadena de texto sea un archivo dentro de la carpeta templates*** del proyecto donde le dirá al modelo que debe generar una página html.

MEJORA DE PRODUCTIVIDAD / AHORRO DE TIEMPO

No solo eso, sino que Spring Boot simplifica el proceso de configuración y creación del código al aportar clases, anotaciones y configuraciones hechas por defecto que el programador puede obviar y que son comunes a todas las aplicaciones web que quiera crear. Por poner un ejemplo implementar un repositorio usando:

import org.springframework.data.jpa.repository.JpaRepository;

Nos permite ahorrarnos el implementar cada método o función de acceso a la base de datos, pudiendo acceder a ella de manera sencilla.

FLEXIBILIDAD Y ESCALABILIDAD

El framework facilita la implementación de microservicios, que son aplicaciones pequeñas y autónomas que pueden desarrollarse, desplegarse y escalarse independientemente. Esto no solo mejora la flexibilidad y la resiliencia del sistema, sino que también facilita el mantenimiento y la actualización del software. Con el soporte de Spring Cloud, los microservicios pueden integrar patrones de diseño avanzados como descubrimiento de servicios, balanceo de carga y tolerancia a fallos, mejorando así la robustez de la aplicación (Spring Boot Tutorial).

CARACTERÍSTICAS DE LOS SISTEMAS DISTRIBUIDOS ENCONTRADAS EN EL PROYECTO VISTAS EN TEORÍA

JPA es una especificación de Java para el mapeo objeto-relacional que permite gestionar datos relacionales en aplicaciones Java. ***Hibernate*** es una implementación de JPA que facilita el manejo de datos, proporcionando una capa de abstracción para la interacción con bases de datos.

CONTROL DE CONCURRENCIA DE SPRINGBOOT gestiona el acceso concurrente a recursos compartidos en una aplicación. Utiliza herramientas como transacciones, bloqueos y mecanismos de sincronización para asegurar que múltiples hilos o procesos puedan operar sin interferencias, manteniendo la integridad y consistencia de los datos como hemos visto en la teoría de la asignatura.

MIDDLEWARE Y MICROSERVICIOS el uso de docker también conlleva un uso de sistema distribuido: para desplegar el proyecto se encapsula en contenedores que se comunican entre ellos y el usuario para facilitar el despliegue de la aplicación, mejorando la escalabilidad. El ***uso de Spring Boot y Docker*** en el proyecto refleja la arquitectura de microservicios, donde cada componente de la aplicación puede ser desarrollado, desplegado y escalado de manera independiente. Spring Boot actúa como un middleware que facilita la creación de servicios web y la comunicación entre ellos, mientras que Docker proporciona la infraestructura para ejecutar estos servicios de manera aislada y consistente.

GESTIÓN DE ESTADOS Y SESIONES En el taller, se menciona la gestión de sesiones de usuario a través de Spring Security y el uso de plantillas Thymeleaf para mantener el estado del usuario en la interfaz web. Esta gestión de estados y sesiones es crucial en sistemas distribuidos para asegurar que las interacciones del usuario sean coherentes y seguras, independientemente de la instancia del servidor que maneje la solicitud.

REPLICACIÓN Y TOLERANCIA A FALLOS En la teoría de la asignatura también se menciona y se usa en el taller este aspecto, el uso de Docker y MySQL con posibles configuraciones de replicación y tolerancia a fallos. La replicación de la base de datos y la ejecución de múltiples contenedores de la aplicación aseguran que el sistema pueda seguir funcionando en caso de fallos en alguna de sus partes, mejorando así la disponibilidad y la fiabilidad del sistema. También la flexibilidad y escalabilidad explicada en el punto anterior como mejora del framework facilita el mantenimiento y la actualización del software, aunque no se haya hecho en el proyecto con Spring se puede balancear carga de trabajo mejorando la robustez de la aplicación.

BIBLIOGRAFÍA

1. Baeldung. *Baeldung*. n.d. Web. Accessed May 16, 2024.
    - "Spring Boot." https://www.baeldung.com/spring-boot.
    - "Spring Boot Start." https://www.baeldung.com/spring-boot-start.
    - "Spring vs Spring Boot." https://www.baeldung.com/spring-vs-spring-boot.
    - "Spring Boot Annotations." https://www.baeldung.com/spring-boot-annotations.
    - "Spring Boot Package Structure." https://www.baeldung.com/spring-boot-package-structure.
    - "JPA and Hibernate Associations." https://www.baeldung.com/jpa-hibernate-associations.
2. Javatpoint. "Spring Boot Architecture." *Javatpoint*, n.d. Web. https://www.javatpoint.com/spring-boot-architecture.
3. Pérez Martínez, Eugenia. *Desarrollo de aplicaciones mediante el Framework de Spring*. Madrid: RA-MA Editorial, 2015. Print.
4. "Thymeleaf: Java XML/XHTML/HTML5 Template Engine." *Thymeleaf*, Thymeleaf, https://www.thymeleaf.org/.
5. Spring.io. "Microservices." Spring.io, n.d. Web. https://spring.io .