CURSO DE PROGRAMACIÓN FULL STACK

SPRING FRAMEWORK GUIA 2

ROLES Y SEGURIDAD

INTRODUCCIÓN

En la guía anterior aprendimos a trabajar con una arquitectura Modelo Vista Controlador (MVC) que separa los datos de una aplicación, la interfaz de usuario, ¡y la lógica de control en tres componentes distintos!
Ahora ya estamos de condiciones de darle más forma y aprender nuevas maneras de desarrollar una aplicación completa tanto a nivel lógico como funcional.

¡Es hora de interactuar con el Usuario! En esta instancia abordaremos las posibilidades que nos brindan las herramientas de seguridad web a la hora de construir una aplicación.
El registro de una cuenta de usuario, el inicio de sesión o las opciones 'administrador' son algunas de las funcionalidades que vamos a desarrollar en esta guía.

¡También aprenderemos a enviar correos electrónicos y a utilizar Lombok! una librería para Java que a través de anotaciones nos reduce el código que codificamos, es decir, nos ahorra tiempo y mejora la legibilidad del mismo.

SPRING SECURITY

Spring Security es el framework (marco de trabajo) encargado de gestionar todo lo relativo a la seguridad dentro de Spring/Spring Boot.

Algunas de las funciones/características principales las que se encarga son Spring Securiy son:

- Protocolos de seguridad.
- Roles para los accesos a los recursos o no.
- Autentificación y autorización con Spring Security.

Mediante a estos mecanismos vamos a intentar proteger la aplicación de una manera sencilla y sin afectar a la lógica de negocio.

AUTORIZACIÓN Y AUTENTICACIÓN BÁSICA

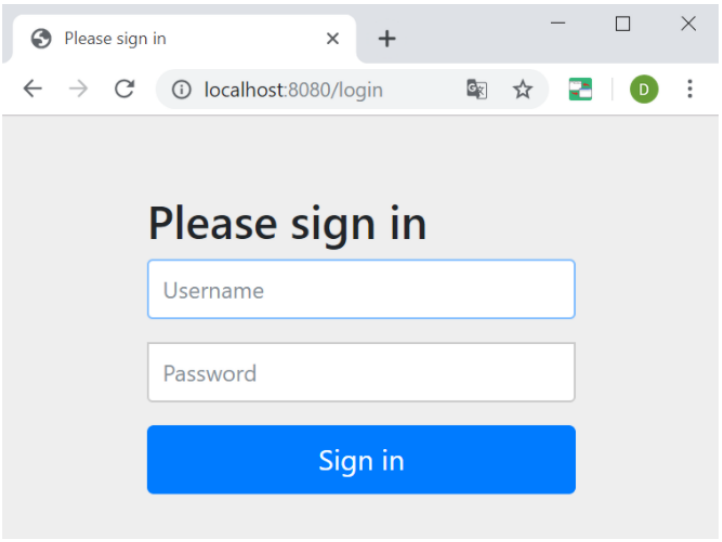
Autenticación: verificamos la identidad del usuario.

Autorización: tipo de permisos que tiene ese usuario.

Lo primero que haremos es incluir la dependencia de Spring Security en el archivo pom.xml del proyecto.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Si creamos un endpoint cualquiera e intentamos consumirlo, se mostrará un formulario de inicio de sesión proporcionado por Spring Security.



Si compruebas la salida por consola, se podrá ver una *contraseña generada automáticamente*. El nombre de usuario por defecto es «user».

```
Using generated security password: 30f20aec-fa73-4dee-a972-22f8ff77a1a5
```

Para crear una clase de seguridad personalizada, necesitamos usar @EnableWebSecurity y extender la clase con @WebSecurityConfigurerAdapter para que podamos redefinir algunos de los métodos proporcionados. Spring Security te fuerza a hashear las contraseñas para que no se guarden en texto plano. Para los siguientes ejemplos, vamos a usar BCryptPasswordEncoder.

Crearemos una clase "WebSecurity" donde ubicaremos la anotación @Configuration.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class WebSecurity extends WebSecurityConfigurerAdapter{


@Autowired
public UsuarioService usuarioService;
```

AUTENTICACIÓN

A continuación del código anterior, vamos a utilizar el método ***configureGlobal*** que recibe como parámetro *AuthenticationManagerBuilder auth* y tiene el método *userDetailsService*, además del encriptador de contraseñas.

```
@Autowired
Public void configureGlobal(AuthenticationManagerBuilder auth) throws
exception{
      auth.userDetailsService(usuarioService).
      passwordEncoder(new BCryptPasswordEncoder());
}
```

ROLES Y AUTORIZACIÓN

Para comenzar, vamos a definir que roles vamos a manejar dentro de nuestra aplicación. Hay diversas maneras de crear los roles, pero en este caso como son roles muy puntuales los manejaremos desde la aplicación en un Enum.

```
public enum Rol {
    ADMIN,
    USER;
}
```

Vamos a crear la clase Usuario con los atributos username, password y rol.

```
@Entity
public class Usuario {

    @Id
    @GeneratedValue(generator = "uuid")
    private String id;

    private String username;
    private String password;
    private String email;

    @Enumerated(EnumType.STRING)
    private Rol rol;
```

Para obtener los usuarios, utilizaremos un Service que se conectará a un repositorio que implementa la interfaz JPA para acceder a nuestra base de datos.

Para que el Service funcione, tiene que estar anotado como @Service y debe implementar la interfaz *UserDetailsService*.

La interfaz *UserDetailsService* se utiliza para recuperar datos relacionados con el usuario. Tiene un método llamado *loadUserByUsername* () que se puede sobreescribir para personalizar el proceso de búsqueda del usuario.

En este caso, vamos a realizar acciones como buscar un Usuario por mail en la base de datos. Al implementar la interfaz UserDetailsService, el IDE nos pedirá que implementemos todos los métodos abstractos. De esta manera, se implementará el @Override del método *loadUserByUsername*.

```java
@Service
public class UsuarioServicio implements UserDetailsService {

    @Autowired
    private UsuarioRepositorio usuarioRepositorio;
```

El repositorio debe ser una interfaz y debe implementar la interfaz JpaRepository.

```java
@Repository
public interface UsuarioRepositorio extends JpaRepository<Usuario, String> {

    public Usuario buscarPorEmail(String email);
```

Veamos el metodo **loadUserByUsername** completo, prestando atención a las referencias del número de línea en el que se encuentra cada parte del código.

```java
42  @Override
    public UserDetails loadUserByUsername(String email)
44          throws UsernameNotFoundException {
45
46  Usuario user = usuarioRepositorio.buscarPorEmail(email);
47
48  if (user != null) {
49
50  List<GrantedAuthority> permissions = new ArrayList<>();
51
52  GrantedAuthority p = new SimpleGrantedAuthority("ROLE_" + user.getRol().toString());
53
54  permissions.add(p);
55
56  ServletRequestAttributes attr = (ServletRequestAttributes) RequestContextHolder.currentRequestAttributes();
57
58  HttpSession session = attr.getRequest().getSession(true);
59
60  session.setAttribute("usuario", user);
61
62  return new User(user.getEmail(), user.getPassword(), permissions);
63  }
64  return null;
65
66  }
```

46- Instanciamos un objeto del tipo Usuario, haciendo uso del método buscarPorEmail(email) de la clase usuarioRepositorio.

48- Verificamos que el objeto Usuario no esté nulo.

50- Otorgaremos PERMISOS según el ROL que tenga cada usuario. Si el usuario existe, es decir, si la búsqueda por mail del repositorio nos retorna un Usuario, vamos a crear una lista de permisos llamada *permissions*

52- Creamos un objeto GrantedAuthority *'p'* y concatenamos la palabra ROLE_ + el rol del usuario.

54- Agregamos el objeto 'p' a la lista de permisos.

56/58- Utilizamos los atributos que nos otorga el pedido al servlet, para poder guardar la información de nuestra HttpSession.

62- por último, retornamos un User con su email, contraseña y permisos!

RESTRINGIR ACCESOS

Por último, la idea sería que apliques tu propia lógica a la hora de restringir el acceso por roles a los usuarios. Para esto, en la clase ***WebSecurity***, extendemos de *WebSecurityConfigurerAdapter, y* se sobreescribe el método *configure* para configurar la entidad http y con ello las url(endpoints) de la página web. En el siguiente ejemplo, sólo los usuarios **con rol ADMIN** podrán ingresar a la ruta **".../admin"**

Del mismo modo, podemos configurar que a la ruta .../index puede ingresar cualquier rol existente.

```
public class WebSecurity extends WebSecurityConfigurerAdapter {

    @Autowired

    public UsuarioServicio usuarioServicio;

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http
                .authorizeRequests()
                .antMatchers("/admin").hasRole("ROLE_ADMIN")
                .antMatchers("/index").hasAnyRole()
                .and()
                .formLogin();

    }
}
```

LOMBOK

A pesar de que NetBeans hace el trabajo por ti cuando creas tus clases entidad, da pereza generar los getters, setters, equals, hashCode, toString... sobre todo cuando tu modelo de datos es grande.

Todo eso nos lo podemos evitar, ya que, **si utilizamos Project Lombok, lo va a hacer por nosotros.**

Lombok es un procesador de anotaciones Java que nos permite agilizar el proceso de desarrollo de una aplicación de software, se ejecuta en tiempo de compilación, durante este proceso crea e inyecta automáticamente el código determinado por la correspondiente anotación Java.

INSTALACION LOMBOK

Para instalar Project Lombok, lo primero que tenemos que hacer es descargar el fichero lombok.jar desde la web de Project Lombok: https://projectlombok.org/

Una vez descargado, ejecutamos el mismo según el entorno en el que trabajemos, y seguimos el asistente que se nos propone.

Lo primero que hará el instalador será escanear los posibles IDEs que tengamos instalados en el sistema. Si no encontrara ninguno, lo que tenemos que hacer es buscarlo de forma manual, seleccionando **Specify location...** y pulsar en Install/Update para instalar Lombok.

AGREGANDO LA DEPENDENCIA

Project Lombok es una dependencia que tenemos que incluir en nuestro proyecto para que podamos añadir una serie de anotaciones que personalizaran nuestro IDE.

```
<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.10</version>
    </dependency>
</dependencies>
```

En el siguiente ejemplo, a la izquierda vemos la clase Person con sus correspondientes campos privados y sus métodos get y set, en el lado derecho vemos la misma clase utilizando lombok, solamente agregando las anotaciones @Getter y @Setter le indicamos que deseamos generar los métodos get y set para todos los campos de la clase Person.

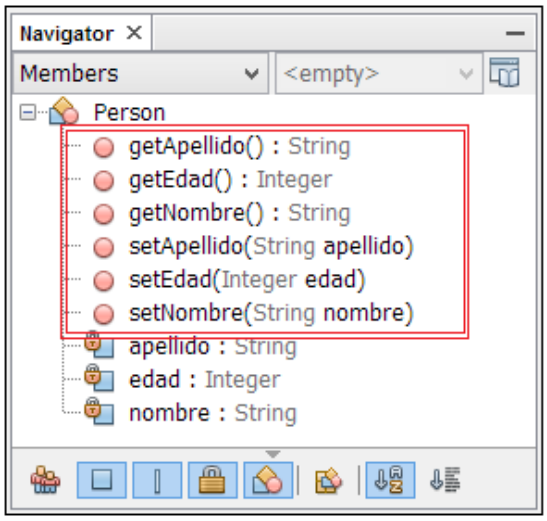
ANOTACIONES LOMBOK

@Getter y @Setter

Estas anotaciones son usadas para generar los métodos getter y setter de manera automática.

```
import lombok.Getter;
import lombok.Setter;
@Getter @Setter
public class Person {
    private String nombre;
    private String apellido;
    private Integer edad;
}
```

Para asegurarnos de que funciona correctamente nos vamos al menú principal de NetBeans y pulsamos la opción: Window | Navigator, esto nos permite mostrar una ventana donde se observa la clase que se está editando con todos los elementos de la misma.



@ToString

Esta anotación la utilizaremos para generar el método toString(), la aplicamos a nivel de clase y cuenta con las siguientes configuraciones:

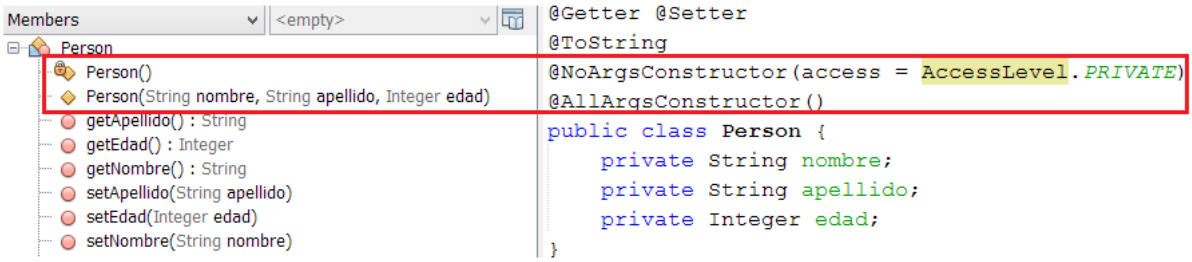
Por defecto la cadena generada corresponde al nombre de la clase seguido de una lista de los campos y su valor actual separados por coma.

```
@Getter @Setter
@ToString
public class Person {
    private String nombre;
    private String apellido;
    private Integer edad;
}
```

```
PERSON: Person(nombre=Leo, apellido=Amadeus, edad=null)
------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------
```

@NoArgsConstructor y @AllArgsConstructor

Usaremos estas anotaciones para generar el constructor de la clase, la primera crea un constructor sin argumentos y el segundo un constructor que permite inicializar cada uno de los campos de la clase.



Controlamos el nivel de acceso con access = AccessLevel.*, en nuestro ejemplo el constructor sin argumentos es privado, si no indicamos el AccessLevel será público como el segundo constructor generado con @AllArgsConstructor.

@Data

Esta anotación es **una combinación de todas las anotaciones que vimos anteriormente** en este tutorial con excepción de @AllArgsConstructor, es muy útil cuando trabajamos con herramientas como el ORM Hibernate.

```
@Data
public class Person {
    private String nombre;
    private String apellido;
    private Integer edad;
}
```

Existen muchas otras anotaciones. En la web https://projectlombok.org/ veras varios ejemplos de cómo utilizarlas.

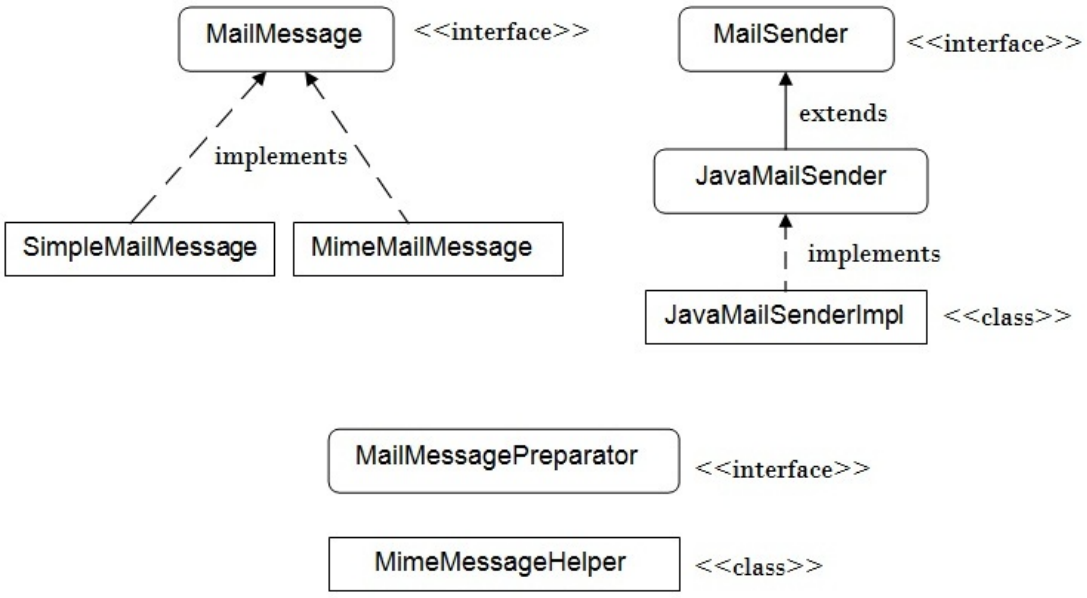
JAVA MAIL SENDER

Spring framework proporciona muchas interfaces y clases útiles para enviar y recibir correos.

El paquete **org.springframework.mail** es el paquete raíz que proporciona soporte de correo electrónico.

API DE CORREO SPRING JAVA

Las interfaces y clases para el soporte de correo de Java en Spring Framework son las siguientes:



- **Interfaz MailSender:** Es la interfaz raíz. Proporciona la funcionalidad básica para enviar correos electrónicos simples.
- **Interfaz JavaMailSender:** Es la subinterfaz de MailSender. Soporta mensajes MIME. Se usa principalmente con la clase MimeMessageHelper para la creación de JavaMail MimeMessage, que nos permite enviar correos con archivos adjuntos, etc. Spring Framework recomienda el mecanismo MimeMessagePreparator para usar esta interfaz.
- **Clase JavaMailSenderImpl:** proporciona la implementación de la interfaz JavaMailSender. Es compatible con JavaMail MimeMessages y Spring SimpleMailMessages.
- **Clase SimpleMailMessage**: se utiliza para crear un mensaje de correo simple que incluye: de, para, cc, asunto y texto.
- **Interfaz MimeMessagePreparator:** es la interfaz de devolución de llamada para la preparación de mensajes JavaMail MIME.
- **Clase MimeMessageHelper:** es la clase auxiliar para crear un mensaje MIME. Ofrece soporte para elementos en línea como imágenes, archivos adjuntos de correo típicos y contenido de texto HTML.

1. DECLARAR DEPENDENCIA PARA SPRING BOOT MAIL

Lo primero que vamos a hacer es seleccionar la dependencia Spring Boot Starter Mail a la hora de crear nuestro proyecto; o agregarla al archivo pom.xml si queremos incluirla en un proyecto existente:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

2. CONFIGURAR PROPIEDADES DE CORREO

Para enviar correos electrónicos desde la aplicación, debemos configurar los ajustes del servidor SMTP en el archivo de configuración de la aplicación Spring Boot (**application.properties**) de la siguiente manera:

```
spring.mail.host=smtp.gmail.com //WEVO VERDE → ESTA CONFIG ES PARA GMAIL//
spring.mail.port=8080
spring.mail.username=direccionDeCorreo
spring.mail.password=contraseñaDelCorreo
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

3. CONFIGURAR UN JAVAMAILSENDER

Spring Mail proporciona JavaMailSender, que es la interfaz clave que define métodos comunes para enviar correos electrónicos. Después de configurar las propiedades del correo en el archivo application.properties, vamos a indicarle a Spring Framework que inyecte la implementación predeterminada de JavaMailSender en una clase Controlador:

```
@Controller
public class AppController {
    @Autowired
    private JavaMailSender mailSender;
    public void sendEmail() {
        // use mailSender here...
    }
}
```

También en la clase de servicio:

```
@Service
Public class BusinessService{

@Autowired
private JavaMailSender mailSender;

public void sendEmail(){
//usar mailSender acá…
}
}
```

4. EJEMPLO DE CÓDIGO PARA ENVIAR UN CORREO ELECTRÓNICO SIMPLE (TEXTO SIN FORMATO)

Necesitamos los siguiente datos:

```
String from = "sender@gmail.com";//dirección de correo que hace el envío.
String to = "recipient@gmail.com";//dirección de correo que recibe el mail.
```

El método sendEmail() se vería así:

```
public void sendEmail(String from, String to) {

SimpleMailMessage message = new SimpleMailMessage();
message.setFrom(from);
message.setTo(to);
message.setSubject("Asunto del correo");
message.setText(“Este es un correo automático!");
mailSender.send(message); //método Send(envio), propio de Java Mail Sender.

}
```

EJERCICIOS DE APRENDIZAJE

Para la realización de este trabajo práctico **se recomienda ver todos los videos de Spring**, de esta manera sabemos todos lo que tenemos que hacer, antes de empezar a hacerlo.

VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

SITIO DE NOTICIAS EGG NEWS PARTE 2

El cliente nos envió nuevos requerimientos para la aplicación web que desarrollamos en la guía anterior. Es por esto que vamos a continuar trabajando sobre el mismo proyecto y usando la misma base de datos para completar nuestro EggNews.

Además de las entidades que ya teníamos, crearemos tres nuevas entidades: Usuario, Periodista y Administrador.

Entidad Usuario

La entidad Usuario modela los usuarios que se registran y loguean en la web. Esta entidad debe poseer los siguientes atributos:

- String nombreUsuario
- String password
- Date fecha de alta.
- Rol rol
- Boolean activo

Un Usuario debe loguearse para ver las noticias, y perderá la posibilidad de ingresar a la vista **panelAdmin**.

Las entidades Periodista y Administrador **deben extender** de Usuario.

Entidad Periodista

La entidad Periodista tendrá como atributos extras:

- ArrayList<Noticia> misNoticias
- Integer sueldoMensual.

Un Periodista debe loguearse en el sistema para poder acceder a **crear y modificar** Noticias. Esta acción deberá realizarse desde la vista **panelAdmin**.

Entidad Administrador

La entidad Administrador podrá:

- Podrá crear, modificar y **eliminar** noticias
- Dar de alta o baja a Periodistas (modificar el atributo activo).
- Indicar cuál va a ser el sueldoMensual de cada Periodista.

Un Administrador debe loguearse en el sistema para poder acceder a sus funcionalidades.

Entidad Noticia

Agregaremos un nuevo atributo Periodista a la entidad Noticia que se va a relacionar con la entidad periodista:

- Periodista **creador;** (relación).

EJERCICIOS EXTRAS

Estos van a ser ejercicios para reforzar los conocimientos previamente vistos. Estos pueden realizarse cuando hayas terminado la guía y tengas una buena base sobre lo que venimos trabajando. Además, si ya terminaste la guía y te queda tiempo libre en las mesas, podes continuar con estos ejercicios extra, recordando siempre que no es necesario que los termines para continuar con el tema siguiente. Por último, recordá que la prioridad es ayudar a los compañeros de la mesa y que cuando tengas que ayudar, lo más valioso es que puedas explicar el ejercicio con la intención de que tu compañero lo comprenda, y no sólo mostrarlo. ¡Muchas gracias!

SISTEMA DE ESTANCIAS EN EL EXTRANJERO WEB

El objetivo de este ejercicio consiste en el desarrollo de un sistema web para una pequeña empresa que se dedica a organizar estancias en el extranjero dentro de una familia. El sistema debe registrar la reserva de casas por parte de los clientes que desean realizar alguna estancia. El objetivo es el desarrollo de un sistema web de reservas de casas para realizar estancias en el exterior, utilizando el lenguaje JAVA, una base de datos MySQL, el framework de persistencia JPA y Spring Boot como framework de desarrollo web.

Creación de la Base de Datos MySQL

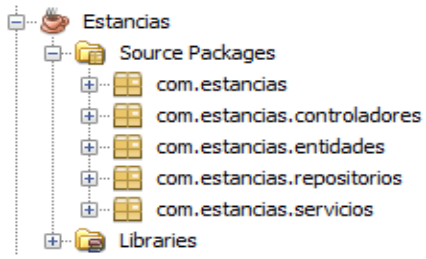
Crear el esquema sobre el cual operará el sistema de reservas de casas. Para esto, en el IDE de base de datos que esté utilizando (por ejemplo, Workbench) se debe ejecutar la sentencia:

```
CREATE DATABASE estancias;
```

De esta manera se creará una base de datos vacía llamada estancias.

Paquetes del Proyecto

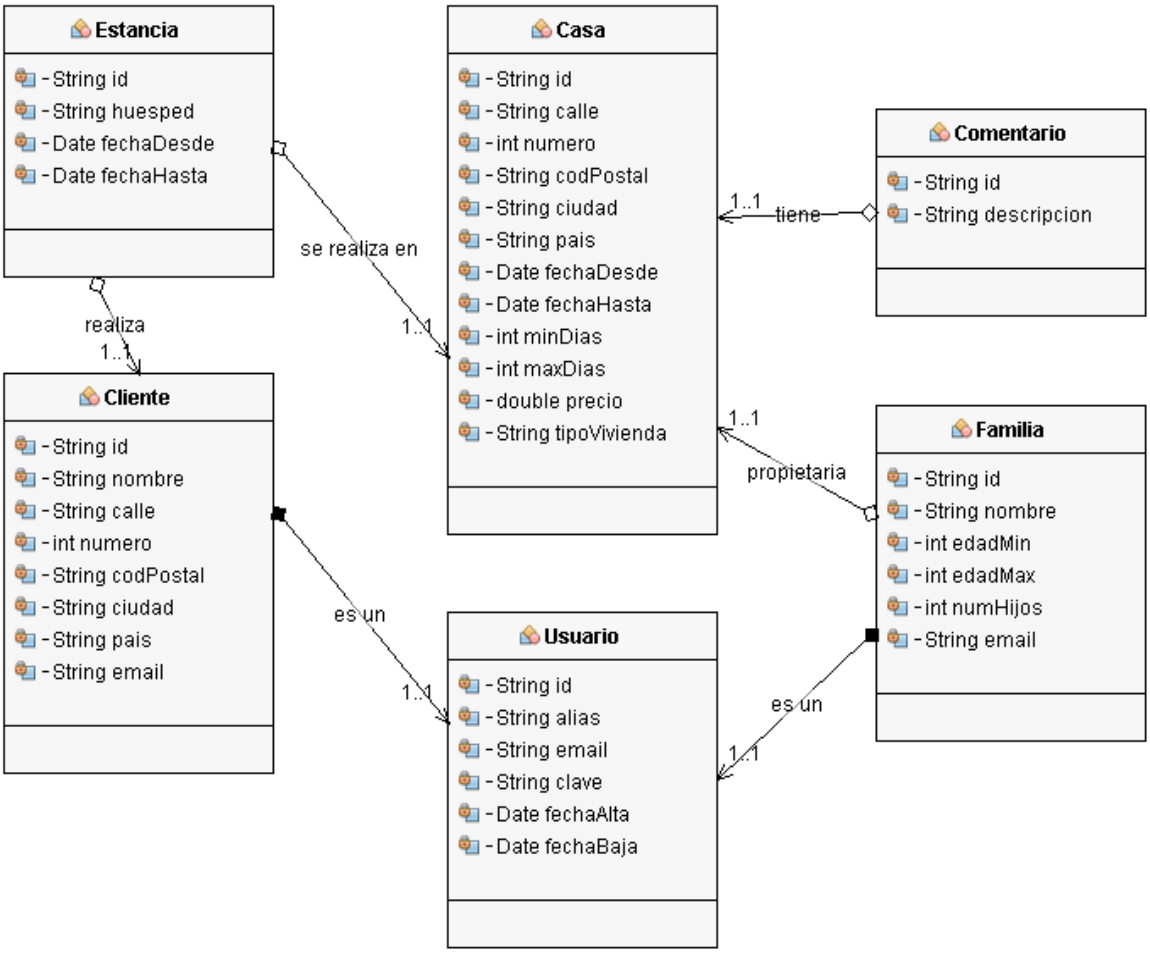
En este proyecto se debe utilizar la misma estructura de capas que en el ejercicio anterior.



Capa de Datos

Entidades y Repositorios

Crear el siguiente modelo de entidades y agregar los repositorios correspondientes. Todas las entidades deben estar marcadas con la anotación @Entity y los repositorios con la anotación @Repository.



Entidad Usuario

La entidad usuario modela los datos de un usuario que accede al sistema para registrarse como familia y ofrecer una habitación de su casa para estancias, o bien, como un cliente que necesita realizar una reserva. De cada usuario se debe registrar el nombre de usuario (alias), el correo electrónico, el password y la fecha de alta. El repositorio que persiste a esta entidad (UsuarioRepositorio) debe contener los métodos necesarios para registrar el usuario en la base de datos, realizar consultas y eliminar.

Entidad Familia

La entidad familia modela las familias que habitan en diferentes países y que ofrecen alguna de las habitaciones de su hogar para acoger a algún chico (por un módico precio). De cada una de estas familias se conoce el nombre, la edad mínima y máxima de sus hijos, número de hijos y correo electrónico. El repositorio que persiste a esta entidad (FamiliaRepositorio) debe contener los métodos necesarios para guardar/actualizar los datos de las familias en la base de datos, realizar consultas y eliminar o dar de baja según corresponda.

Entidad Casa

La entidad casa modela los datos de las casas donde las familias ofrecen alguna habitación. De cada una de las casas se almacena la dirección (calle, numero, código postal, ciudad y país), el periodo de disponibilidad de la casa (fecha_desde, fecha_hasta), la cantidad de días mínimo de estancia y la cantidad máxima de días, el precio de la habitación por día y el tipo de vivienda. El repositorio que persiste a esta entidad (CasaRepositorio) debe contener los métodos necesarios para guardar/actualizar los datos de una vivienda, realizar consultas y eliminar.

Entidad Cliente

La entidad cliente modela información de los clientes que desean mandar a sus hijos a alguna de las casas de las familias. Esta entidad es modelada por el nombre del cliente, dirección (calle, numero, código postal, ciudad y país) y su correo electrónico. El repositorio que persiste a esta entidad (ClienteRepositorio) debe contener los métodos necesarios para guardar/actualizar los datos de un cliente, realizar consultas y eliminar.

Entidad Reserva

La entidad reserva modela los datos de las reservas y estancias realizadas por alguno de los clientes. Cada estancia o reserva la realiza un cliente, y además, el cliente puede reservar varias habitaciones al mismo tiempo (por ejemplo para varios de sus hijos), para un periodo determinado (fecha_llegada, fecha_salida). El repositorio que persiste a esta entidad (ReservaRepositorio) debe contener los métodos necesarios para realizar una reserva, actualizar los datos (por ejemplo, fecha de la reserva), realizar consultas de las reservas realizadas para una determinada vivienda y eliminar reserva.

Entidad Comentario

La entidad comentario permite almacenar información brindada por los clientes sobre las casas en las que ya han estado. El repositorio que persiste a esta entidad (ComentarioRepositorio) debe contener los métodos necesarios para guardar los comentarios que realizan los clientes sobre una determinada una vivienda.

Capa de Servicios

Utiliza la anotación @Service para identificar aquellas clases que serán servicios. Todos los servicios deben estar marcados con esta anotación.

UsuarioServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar usuarios (alta de usuario, consultas, y baja o eliminación).

FamiliaServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar familias (creación, consulta, modificación y eliminación).

CasaServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar las casas (creación, consulta, modificación y eliminación).

ClienteServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar clientes (creación, consulta, modificación y eliminación).

ReservaServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para realizar las reservas de viviendas (reservar, consultar reservas realizadas, modificación y eliminación).

Capa de Comunicación

Spring utiliza una anotación para identificar aquellas clases que serán controladores. Todos los controladores deben estar marcadas con la anotación @Controller. Algunos de los controladores a desarrollar son los siguientes.

UsuarioController

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para operar con la vista del usuario diseñada para la gestión de usuarios (dar de alta un usuario, cambiar clave, listar usuarios registrados, dar de baja un usuario).

FamiliaController

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para operar con la vista del usuario diseñada para la gestión de familias (guardar/modificar datos de la familia, listar familias, eliminar).

CasaController

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para operar con la vista del usuario diseñada para la gestión de viviendas (guardar/modificar datos de la casa, listar viviendas, eliminar).

ClienteController

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para operar con la vista del usuario diseñada para la gestión de clientes (guardar/modificar, listar clientes, eliminar).

ReservaController

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para operar con la vista o portal para gestionar reservas de estancias (guardar/modificar, listar estancias reservadas/realizadas, eliminación).

Capa de Vista

Esta capa tiene la responsabilidad de llevar adelante las funcionalidades necesarias para interactuar con el usuario. Las vistas para este proyecto tienen que estar desarrolladas en HTML5 y se debe utilizar la biblioteca Thymeleaf y CSS para implementar las plantillas.

Se deben diseñar y crear todas las vistas web necesarias para llevar a cabo las siguientes funcionalidades:

- **Administrar usuarios:** registrar nuevos usuarios en el sistema, cambiar clave, listar usuarios, dar de baja o eliminar.

- **Administrar familias:** cargar datos de una familia, modificar datos, consultar familias, eliminar familias que no ofrecen más sus viviendas para estancias. Las familias deben darse de alta una vez que hayan sido registradas como usuario.

- **Administrar casas:** cargar los datos de una vivienda y asociar a la familia correspondiente, modificar los datos (por ejemplo, fechas en las cuales se encuentra disponible), listar casas (país, el periodo de disponibilidad, cantidad de días mínima y máxima de estancia, el precio de la habitación por día, el tipo de vivienda y el nombre del propietario). Se debe eliminar los datos de una casa cada vez que se da de baja la familia propietaria.

- **Administrar clientes:** cargar datos de los clientes que desean reservar una habitación para realizar una estancia, modificar datos de los clientes, realizar consultas y eliminar clientes. Al igual que las familias, un cliente puede darse de alta una vez que se haya creado el usuario correspondiente.

- **Realizar Reservas:** El portal principal debería permitir que una persona que ingresa a la web pueda consultar las viviendas que se encuentran disponibles para realizar estancias (validando fechas disponibles). Una vez que el usuario encuentra una vivienda que se adecua a sus preferencias y quiere realizar una reserva, recién en ese momento se solicita que se registre como usuario y luego se procede a realizar la reserva.

    - Cuando el usuario se registra se le debe dar la opción de elegir si se quiere registrar como familia que ofrece una vivienda o como cliente. Dependiendo de la opción elegida se pide que complete los datos correspondientes (familia o cliente) para continuar con su registro.

    - Si el usuario ya se encuentra restirado entonces debe realizar el login para poder continuar.

    - Se debe tener en cuenta que para realizar una reserva la vivienda no debe estar ya reservada por otro cliente.

    - Se debe permitir que un cliente modifique su reserva, por ejemplo: cambiar las fechas, o la elimine en caso de no poder realizarla.

    - Se deben poder listar las reservas realizadas por parte de los clientes.

BIBLIOGRAFÍA

- https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/protocolo-http/
- https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/http-request/
- https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/una-mirada-a-los-codigos-de-estado-http-mas-comunes/
- https://edytapukocz.com/url-partes-ejemplos-facil/
- https://howtodoinjava.com/maven/maven-dependency-management/
- http://chuwiki.chuidiang.org/index.php?title=Dependencias_con_maven
- https://www.arquitecturajava.com/que-es-spring-framework/
- https://programandoointentandolo.com/2013/05/inyeccion-de-dependencias-en-spring.html
- https://proitcsolution.com.ve/inyeccion-de-dependencias-spring/
- https://www.java67.com/2019/04/top-10-spring-mvc-and-rest-annotations-examples-java.html
- https://www.danvega.dev/blog/2017/03/27/spring-stereotype-annotations/
- https://www.arquitecturajava.com/spring-stereotypes/
- https://www.arquitecturajava.com/que-es-spring-boot/
- https://openwebinars.net/blog/que-es-thymeleaf/