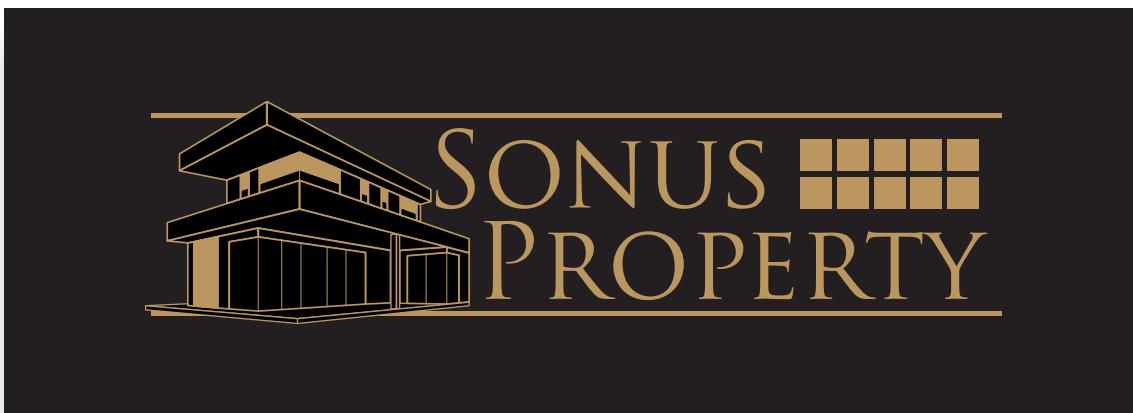
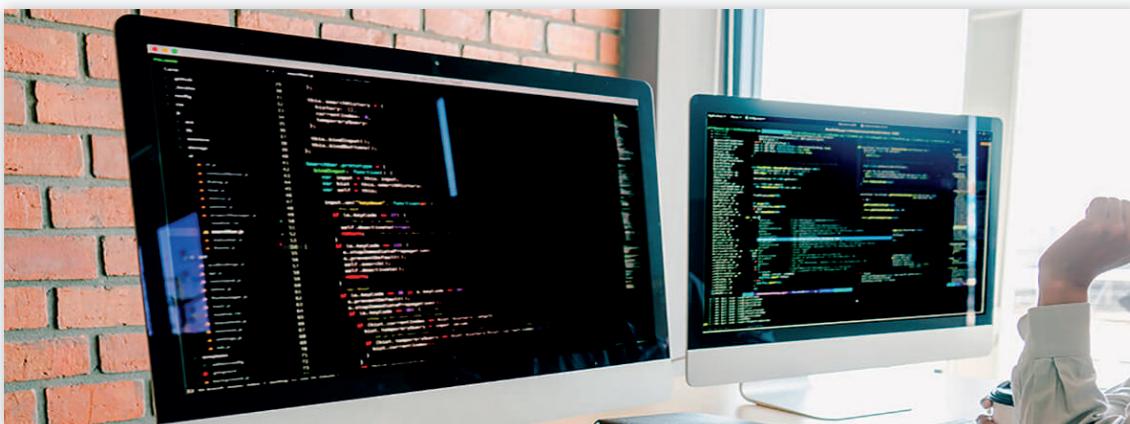




# MP\_Proyecto de Desarrollo de Aplicaciones Web



RAÚL RUIZ SANZ  
MIGUEL ÁNGEL LOZANO BERMEJO



## INTRODUCCIÓN, JUSTIFICACIÓN Y OBJETIVOS

El trabajo realizado en este Proyecto de Fin de Grado, correspondiente al Ciclo Formativo de Desarrollo de Aplicaciones Web, se ha basado en la realización de una aplicación web que permita desarrollar algunas de las funcionalidades necesarias en el día a día de una inmobiliaria.

Dentro de esta aplicación por tanto el cliente podrá encontrar información de los distintos inmuebles gestionados por la inmobiliaria, contactar con esta y tener una lista de inmuebles favoritos en caso de estar registrado.

De igual manera los administradores de la inmobiliaria disponen de un panel de control desde el cual pueden manejar de manera cómoda la gestión de los usuarios registrados, así como de los inmuebles que componen su cartera actual o futura.

Los objetivos que queremos alcanzar con este proyecto es poner en él todos los conocimientos aprendidos durante estos dos años en el instituto Edix.

Hemos intentado usar la mayoría de tecnologías aprendidas y poner en práctica la mayor cantidad de elementos posibles, siempre acorde con la estructura del proyecto y con una justificación real.

## ENLACE A REPOSITORIO GITHUB

[https://github.com/MiguelALozano/TFG-DAW-Sonus\\_Property.git](https://github.com/MiguelALozano/TFG-DAW-Sonus_Property.git)

## ACCESO A LA WEB COMO USUARIOS

En estos momentos existen tres usuarios creados en la base de datos con los cuales podemos acceder a sus correspondientes menús:

Usuario: **RaúlRSanz**

password: **1234**

rol: **Administrador**

Usuario: **MiguelALozano**

password: **1234**

rol: **Administrador**

Usuario: **LauraSAlba**

password: **1234**

rol: **Usaria registrada**

## INDICE

<b>INTRODUCCIÓN, JUSTIFICACIÓN Y OBJETIVOS.....</b>	<b>2</b>
<b>ENLACE A REPOSITORIO GITHUB.....</b>	<b>2</b>
<b>ACCESO A LA WEB COMO USUARIOS.....</b>	<b>2</b>
<b>MODULOS FORMATIVOS APLICADOS.....</b>	<b>4</b>
<b>HERRAMIENTAS/LENGUAJES UTILIZADOS.....</b>	<b>5</b>
<b>EQUIPO DE TRABAJO.....</b>	<b>8</b>
<b>FASES DEL PROYECTO.....</b>	<b>9</b>
<b>CONCLUSIONES Y MEJORAS DEL PROYECTO .....</b>	<b>17</b>
<b>BIBLIOGRAFÍA.....</b>	<b>17</b>

## MODULOS FORMATIVOS APLICADOS

### Desarrollo Web Entorno Cliente

Mediante el uso de JavaScript y más en concreto la API Fetch se ha realizado la petición de ciertos recursos que necesitamos mostrar en distintas partes de la aplicación.

### Desarrollo Web Entorno Servidor

La aplicación realizada en este Trabajo de fin de grado aplica el concepto de MVC y hace uso del framework Spring. En el caso concreto que nos ocupa hacemos uso de Spring Boot que es un subproyecto de Spring

### Diseño de Interfaces Web

Con la aplicación de los conocimientos adquiridos en este módulo se ha intentado conseguir una experiencia de usuario agradable con una interfaz clara y fluida.

### Bases de Datos

Se ha hecho uso de una BBDD relacional para conseguir una persistencia de los datos necesarios para el correcto funcionamiento de la aplicación.

### Entornos de Desarrollo

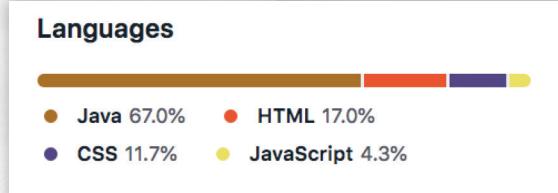
El uso de herramientas CASE, como es el caso del IDE Eclipse, utilizado para la realización de la mayor parte de tareas, así como la depuración de errores mediante el uso de su ‘debugger’ así como el apoyo proporcionado por Git como herramienta para el control de versiones y el trabajo colaborativo han sido un pilar en este proyecto.

### Lenguaje de Marcas y Sistemas de Gestión de Información

El conocimiento de las etiquetas del lenguaje de marcas HTML y la maquetación de las mismas mediante hojas de estilos CSS ha permitido dar forma a la parte de la aplicación que mostramos al usuario.

### Programación

La base sobre la que se asienta la construcción de todo este proyecto, es el lenguaje de programación JAVA, como podemos observar en el enlace a Github, donde se aloja el proyecto, supone alrededor del 67% del mismo.



## HERRAMIENTAS/LENGUAJES UTILIZADOS

### HTML

Lenguaje de Marcas de Hipertexto, del inglés HyperText Markup Language es el componente más básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento (JavaScript).

Fuente: <https://developer.mozilla.org/es/docs/Web/HTML>

### CSS

Hojas de Estilo en Cascada (del inglés Cascading Style Sheets) o CSS es el lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML (en-US) (incluyendo varios lenguajes basados en XML como SVG, MathML o XHTML). CSS describe cómo debe ser renderizado el elemento estructurado en la pantalla, en papel, en voz alta o en otros medios.

Fuente: <https://developer.mozilla.org/es/docs/Web/CSS>

### JavaScript

JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado justo-a tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo, programación funcional).

Fuente: <https://developer.mozilla.org/es/docs/Web/JavaScript>

### Java

Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado, y cada día se crean más. Java es rápido, seguro y fiable. Desde ordenadores portátiles hasta centros de datos, desde consolas para juegos hasta computadoras avanzadas, desde teléfonos móviles hasta Internet, Java está en todas partes. Si es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno

de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web.

Fuente: [https://es.wikipedia.org/wiki/Java\\_\(lenguaje\\_de\\_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n))

### **Eclipse**

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama “Aplicaciones de Cliente Enriquecido”, opuesto a las aplicaciones “Cliente-liviano” basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Fuente: [https://es.wikipedia.org/wiki/Eclipse\\_\(software\)](https://es.wikipedia.org/wiki/Eclipse_(software))

### **Visual Studio Code**

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto.

Fuente: [https://es.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://es.wikipedia.org/wiki/Visual_Studio_Code)

### **MySQL WorkBench**

MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL.

Fuente: [https://es.wikipedia.org/wiki/MySQL\\_Workbench](https://es.wikipedia.org/wiki/MySQL_Workbench)

### **Git/Github**

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena generalmente de forma pública.

Fuente: <https://es.wikipedia.org/wiki/GitHub>

### **Postman**

Se trata de una herramienta dirigida a desarrolladores web que permite realizar peticiones HTTP a cualquier API. Postman es muy útil a la hora de programar y hacer pruebas, puesto que nos ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros desarrollos.

Fuente: <https://lamadriguerabit.com/articulos/que-es-postman/>

### **diagrams.net (Draw.io)**

Es un software de dibujo de gráficos multiplataforma gratuito y de código abierto desarrollado en HTML5 y JavaScript. Su interfaz se puede utilizar para crear diagramas como diagramas de flujo, wireframes, diagramas UML, organigramas y diagramas de red.

Fuente: <https://en.wikipedia.org/wiki/Diagrams.net>

## EQUIPO DE TRABAJO

Los componentes que forman parte de este equipo de trabajo son:

**RAÚL RUIZ SANZ y MIGUEL ÁNGEL LOZANO BERMEJO.**

Para este trabajo, hemos unificado en un único proyecto, las mejores habilidades de cada uno de los miembros del equipo, dividiendo el trabajo en dos grandes campos Frontend y Backend.

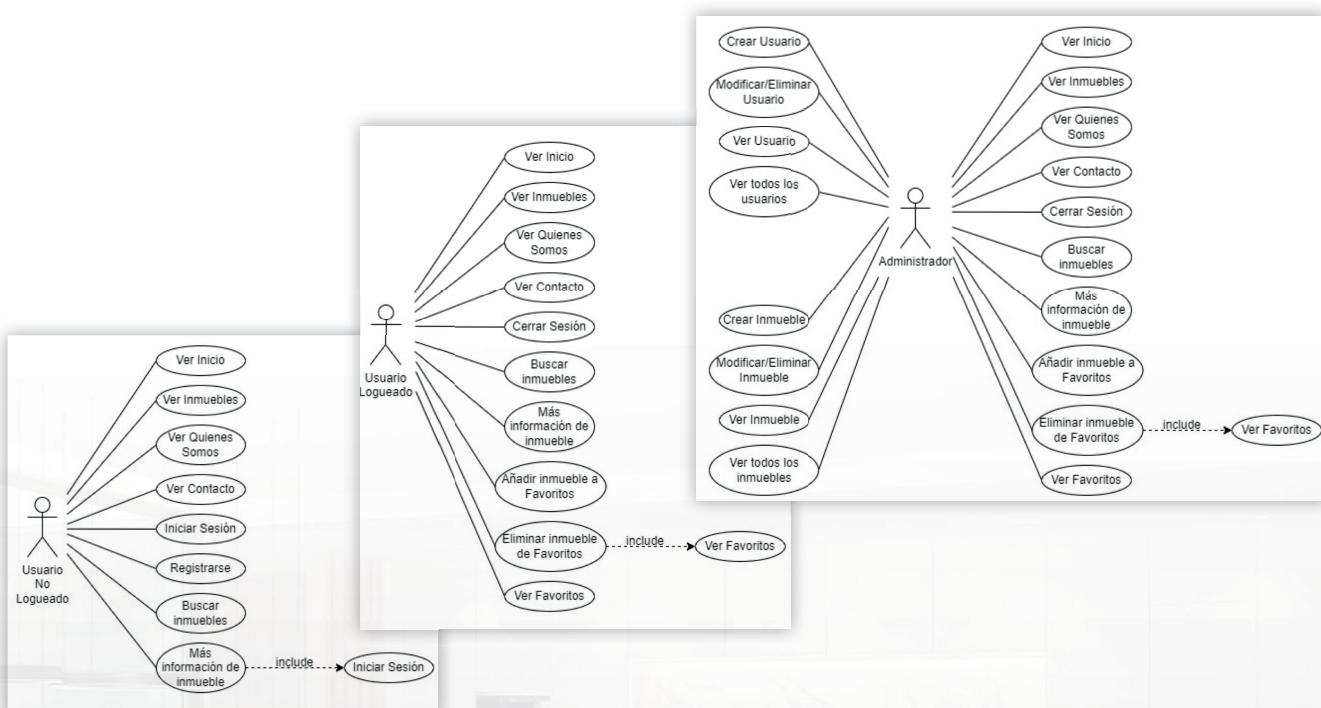
En el Frontend, realizado mayormente por Miguel A., se comenzó con un boceto inicial donde se recogía la estructura del Home y con la creación inicial del logotipo, que se ha diseñado y vectorizado expresamente para este proyecto. La idea del nombre para esta inmobiliaria, **Sonus Property**, surge de combinar dos términos, del Latín Sonus (soñar) y del inglés Property (propiedad), también se decidió la elección de colores y demás elementos del imagotipo para poder tener una referencia a la hora de diseñar el entorno.

En el Backend, realizado íntegramente por Raúl, lo primero fue pensar, diseñar y crear la **BBDD** en **mysql** y el **Proyecto** en **Eclipse**. Estos dos elementos van creciendo y adquiriendo mejoras considerables a medida que trascurren los días. Además de realizar toda la estructura del proyecto, scripts para BBDD, beans, modelos dao, security, etc... desarrolla todo el código necesario para el buen funcionamiento de la web y la base de datos así como su correcto enlace entre ambos.

## FASES DEL PROYECTO

### CASOS DE USO

Gracias a la utilización de Spring Security dentro del proyecto, podemos presentar distintas opciones a los usuarios de la aplicación en función de si han iniciado sesión o no dentro de la aplicación, y de los roles que dispongan. Esto nos da lugar a una serie de casos de uso que diferenciamos en función de estos criterios.



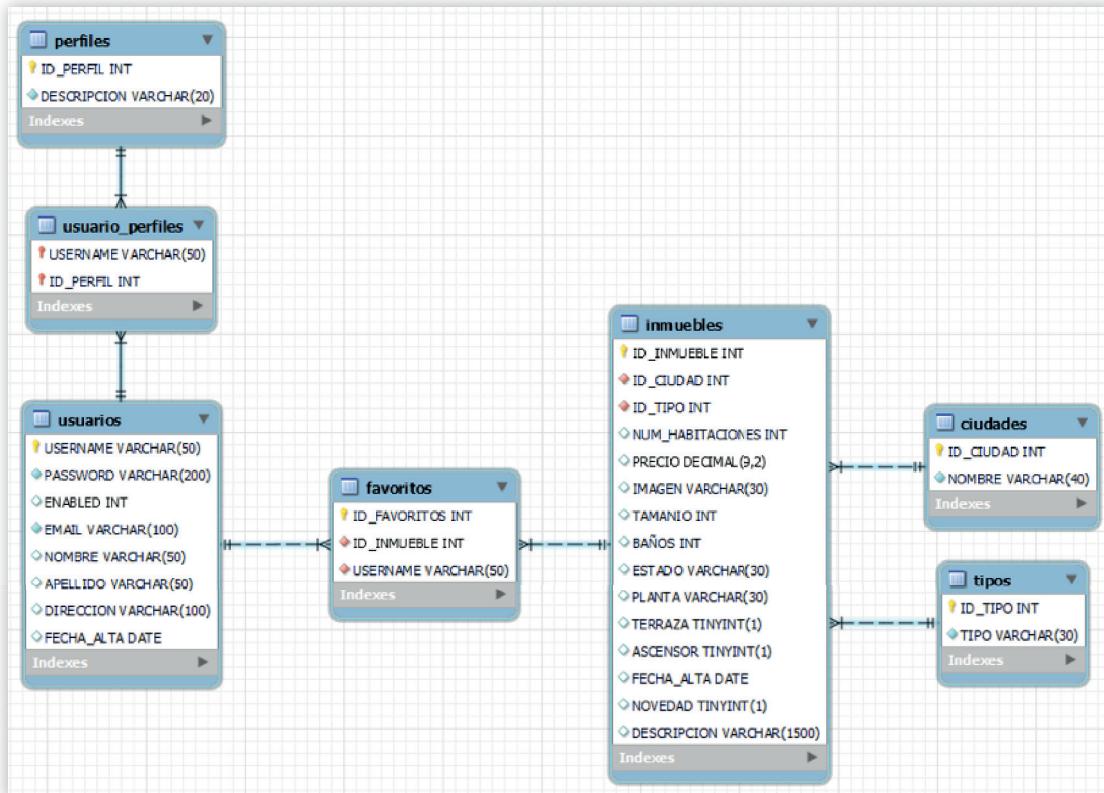
### BASE DE DATOS

Con la vista puesta en los casos de uso anteriormente descritos y teniendo en cuenta el uso de Spring Security como parte de nuestro proyecto se diseñó el siguiente modelo ER para crear nuestra base de datos.

Gracias a este diseño podemos controlar que aspectos de la aplicación se mostrarán a los distintos usuarios en función de los distintos perfiles que estos posean, y podemos persistir todos los datos necesarios en la funcionalidad actual de la aplicación (inmuebles, usuarios, favoritos).

En el repositorio de git cuyo enlace se proporcionó anteriormente se incluyen:

- Script utilizado para la creación de la base de datos.
- Script de carga de datos para las pruebas realizadas en la aplicación.



## SPRING BOOT

Para comenzar el proyecto podríamos haber utilizado la utilidad <https://start.spring.io/>. Esta nos genera un zip que posteriormente podemos descomprimir e importar en Eclipse, que ha sido el IDE utilizado para esta parte del proyecto.

Nosotros optamos por iniciar el trabajo directamente desde Eclipse mediante Spring Starter Project. Básicamente el funcionamiento es igual al de la utilidad anterior, pero nos evitamos el paso de descomprimir e importar. Indicamos las dependencias necesarias para nuestras necesidades, en nuestro caso para simplificar el desarrollo no incluimos en un primer momento la dependencia de Spring Security.

Las dependencias utilizadas en este proyecto fueron:

- **Spring Web.** Necesaria para la construcción tanto de aplicaciones web como de servicios REST.

- **Spring Boot DevTools.** Nos proporciona un Tomcat embutido y nos recarga el proyecto ante los cambios realizados.
- **MySQL Driver.** Necesario ya que nuestra base de datos es de tipo MySQL.
- **Spring Data JPA.** Incluye la Java Persistence API y el motor de persistencia Hibernate.
- **Spring Security.** Nos permite implementar seguridad en nuestro proyecto como, por ejemplo, controles de acceso por URL.

Una vez terminado el proceso anterior configuramos nuestro archivo **application.properties** indicando por ejemplo donde se encontrarán nuestras vistas y los datos necesarios para conectar a nuestra base de datos.

```
server.port=4000
# para las vistas con jsp
spring.mvc.view.prefix=/WEB-INF/vistas/
spring.mvc.view.suffix=.jsp

# para mysql 8 bbdd productos_ite
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/inmobiliaria_bbdd?serverTimezone=Europe/Madrid
spring.datasource.username=uinmueble
spring.datasource.password=uinmueble

spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

Con nuestra base de datos ya creada se hizo uso de la funcionalidad llamada “Project Faces” que permite crear nuestras clases anotadas. Durante este proceso se nos pide crear una conexión con la base de datos que queramos usar y finalmente tendremos disponible la opción “JPA Entities from Tables”. Este proceso nos hace una propuesta de diagrama de clases, su propuesta siempre es bidireccional. En nuestro caso este tipo de relación no es de nuestro interés salvo en el caso de Usuario y Favorito. La clase Usuario si guardara una lista con todos los favoritos de ese usuario y estará anotada con cascadeType. All. Esto permite que cuando borramos un usuario el cambio se difunda por nuestra base de datos y se borren también los favoritos asociados a dicho usuario.

```
@Entity
@Table(name="usuarios")
@NamedQuery(name="Usuario.findAll", query="SELECT u FROM Usuario u")
public class Usuario implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private String username;

    private String apellido;
    private String direccion;
    private String email;
    private int enabled;

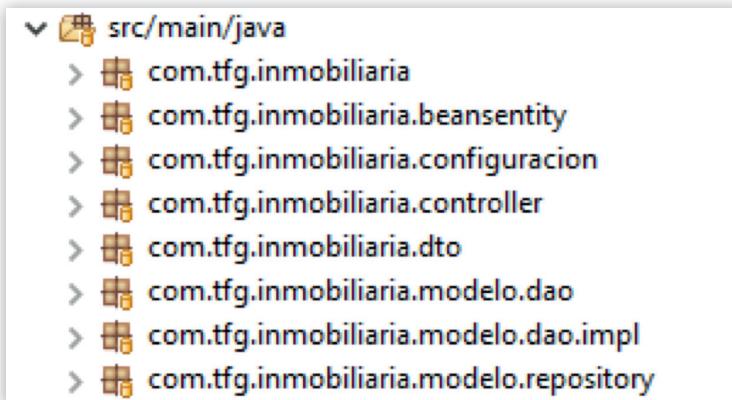
    @Temporal(TemporalType.DATE)
    @Column(name="FECHA_ALTA")
    private Date fechaAlta;

    private String nombre;
    private String password;

    //uni-directional many-to-many association to Perfil
    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
        name="usuario_perfiles"
        , joinColumns={
            @JoinColumn(name="USERNAME")
        }
        , inverseJoinColumns={
            @JoinColumn(name="ID_PERFIL")
        }
    )
    private List<Perfil> perfiles;
```

Como se puede observar en la estructura del proyecto se siguió en patrón modelo vista controlador (MVC) que consiste en la separación de aquello que se va a mostrar al cliente (vista), de lo que es la lógica de negocio y modelo de datos (modelo) y el gestor del flujo de peticiones del cliente (controlador).

## Modelo



Se generaron las clase anotadas mediante el proceso indicado anteriormente que se encuentran en el paquete **com.tfg.inmobiliaria.beansentity**.

También se crearon tres paquetes más para el modelo:

- **com.tfg.inmobiliaria.modelo.repository**. Que contiene interfaces que serán usados en los DAO correspondientes a los distintos beans y nos proporcionan acceso a todos los métodos de JpaRepository al extender de ella.

En el caso de necesitar consultas que no sean las habituales podemos implementarlas aquí de distintas formas como se ve en el siguiente ejemplo.

```
package com.tfg.inmobiliaria.modelo.repository;

import java.util.List;

public interface IntInmuebleRepository extends JpaRepository<Inmueble, Integer>{
    @Query ("select i from Inmueble i where i.novedad = 1")
    public List<Inmueble> findNovedades();
    public List<Inmueble> findByCiudadNombreAndTipoTipo(String nombre, String tipo);
    public List<Inmueble> findByCiudadNombre(String nombre);
    public List<Inmueble> findByTipoTipo(String tipo);
}
```

- **com.tfg.inmobiliaria.modelo.dao**. Es una interface que nos expone los métodos que necesitaremos en el uso del bean inmueble.
- **com.tfg.inmobiliaria.modelo.dao.impl**. Aquí es donde desarrollamos la funcionalidad de la interface anterior haciendo uso de la interface creada en el paquete repository.

```

package com.tfg.inmobiliaria.modelo.dao.impl;

import java.util.List;

@Service
public class InmuebleDaoImpl implements IntInmuebleDao{

    @Autowired
    private IntInmuebleRepository inmuebleRepo;

    @Override
    public List<Inmueble> findAll() {
        return inmuebleRepo.findAll();
    }

    @Override
    public Inmueble findById(int id) {
        return inmuebleRepo.findById(id).orElse(null);
    }

    @Override
    public List<Inmueble> findNovedades(){
        return inmuebleRepo.findNovedades();
    }

    @Override
    public boolean altaInmueble(Inmueble inmueble) {
        if(findById(inmueble.getIdInmueble()) == null) {
            inmuebleRepo.save(inmueble);
            return true;
        }else
            return false;
    }

    @Override
    public boolean modificarInmueble(Inmueble inmueble) {
        if(findById(inmueble.getIdInmueble()) != null) {
            inmuebleRepo.save(inmueble);
            return true;
        }else
            return false;
    }
}

```

- Un último paquete en lo referente al modelo es el que corresponde a **com.tfg.inmobiliaria.dto**, aquí hemos creado una clase que nos permite recoger un objeto fruto de una petición a un servicio REST y poder operar posteriormente con él.

### Petición a servicio REST:

```

fetch('http://localhost:4000/rest/inmobiliaria/añadirInmuebleFavorito', {
    method: 'PUT',
    body: JSON.stringify({
        idInmuebleDTO : idInmueble,
        usernameDTO : username,
        idFavoritoDTO : idFavorito
    }),
    headers: {'Content-type': 'application/json'}
})

```

### Parte de la clase FavoritoDTO:

```

public class FavoritoDTO {

    private String usernameDTO;
    private int idInmuebleDTO;
    private int idFavoritoDTO;

    public FavoritoDTO() {
        super();
    }
}

```

### Controlador

Se han creado distintos controladores en relación a las actividades que se querían controlar. En nuestro caso **Homecontroller** es simplemente un punto de entrada a la

aplicación y en el caso de **AdmonController** se encarga del control de las peticiones realizadas por los administradores de la aplicación a través del panel que se les proporciona en su vista correspondiente.

Así mismo se creó un **ControladorRest** que es el encargado de dar respuesta a las peticiones que se realizan mediante la API fetch en ciertas partes del proyecto.

### Parte del AdmonController:

```
package com.tfg.inmobiliaria.controller;
import java.util.ArrayList;

@Controller
@RequestMapping("/admon")
public class AdmonController {

    @Autowired
    private IntPerfilDao perfilDao;

    @Autowired
    private IntUsuarioDao usuarioDao;

    @Autowired
    private IntCiudadDao ciudadDao;

    @Autowired
    private IntTipoDao tipoDao;

    private IntInmuebleDao inmuebleDao;

    @GetMapping("/panelAdmon")
    public String panelControl() {
        return "panelAdmon";
    }

    @GetMapping("/modificarUsuario")
    public String procesarModificarUsuario() {
        return "modificarUsuario";
    }

    @GetMapping("/altaUsuario")
    public String crearUsuario() {
        return "altausuario";
    }

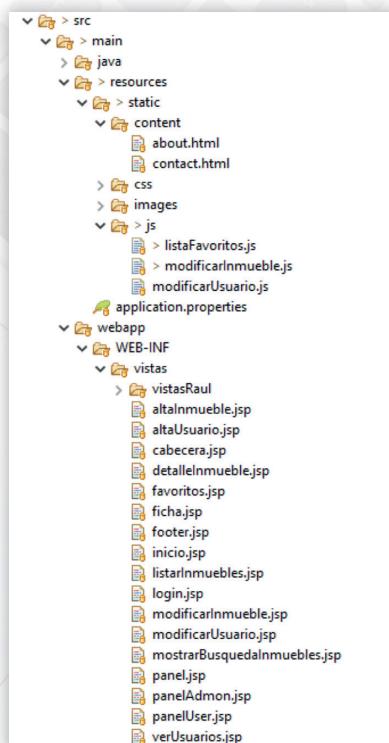
    @PostMapping("/altaUsuario")
    public String procesarAltaUsuario(Usuario usuario, @RequestParam (required = false) boolean perfilAdmon, @RequestParam String passwordRepetido, Model model ) {
        //cuando creo un usuario siempre le doy perfil de usuario
        List<Perfil> perfiles = new ArrayList<Perfil>();
        perfiles.add(perfilDao.findById(1));
        usuario.setEnabled(true);
        usuario.setFechaNacimiento(new Date());
        usuario.setPerfiles(null);
        usuario.setPerfiles(perfiles);
        //compruebo que los password coinciden
        if(!usuario.getPassword().equals(passwordRepetido))) {
            model.addAttribute("mensaje", "Los password introducidos no coinciden");
            return "altausuario";
        }
    }
}
```

### Vistas

El fruto de las distintas peticiones que un usuario hace en la aplicación suele derivar en la presentación de distintos datos, por este motivo es necesaria la creación de una serie de vistas que sirvan estos datos de manera agradable para proporcionar una buena experiencia.

En aras de la reutilización de código se crearon vistas específicas para **cabecera** y **footer**, que serán reutilizadas por el resto de páginas.

También hemos incluido **contenido estático**, como son **about.html** y **contact.html**, y ciertas vistas ven modificado su aspecto mediante el uso de **JavaScript**.



## JavaScript

Una vez realizado todo el trabajo anterior, se dotó de funcionalidad a ciertas partes de la aplicación mediante el uso de JavaScript. Como ejemplo de estos casos tenemos los botones utilizados por los administradores para buscar inmuebles o usuario en la base de datos.

Estos botones están asociados a un eventlistener y cuando hacemos clic sobre ellos se realiza una petición a nuestro servicio REST para recuperar los datos de la base de datos y mostrarlos en el formulario sin necesidad de realizar una carga de la página completa.

Como último paso en el proyecto, encontramos la implementación de **Spring Security**. Para ello añadimos el starter correspondiente y creamos una clase que extiende de **WebSecurityConfigurerAdapter**. Sobrecargando el método **configure** indicamos tanto los campos necesarios para la autenticación como las restricciones correspondientes a las distintas URLs del proyecto.

```
<nav>
    <ul class="menu">
        <li><a href="/" class="activa">INICIO</a></li>
        <li><a href="/inmueble/verTodos">INMUEBLES</a></li>
        <li><a href=".../content/about.html">QUÍENES SOMOS</a></li>
        <li><a href=".../content/contact.html">CONTACTO:</a></li>
        <li><h4 id="usuario">${sessionUserName}</h4></li>
        <sec:authorize access="isAuthenticated()>
            <li><h4><a href="/Login">Inicia Sesión</a></h4></li>
            <li><h4><a href="/admon/altaUsuario">Registrarse</a></h4></li>
        </sec:authorize>
        <sec:authorize access="isAuthenticated()>
            <li><h4><a href="/user/verFavoritos/${sessionUserName}">Favoritos</a></h4></li>
        </sec:authorize>
        <sec:authorize access="hasAuthority('ROL_ADMIN')>
            <li><h4><a href="/admon/panelAdmon">Admin. panel</a></h4></li>
        </sec:authorize>
        <sec:authorize access="isAuthenticated()>
            <li><h4><a href="/logout">Cerrar Sesión</a></h4></li>
        </sec:authorize>
        <%-
            La parte con su usuario correspondiente
        <li><h4>Bienvenido ${session.nombre} </h4></li>
        <li><h4><a href="/logout">Cerrar Sesión</a></h4></li>
        --%
    </ul>
</nav>

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .logout().logoutSuccessUrl("/")
        .authorizeRequests()
        // Los recursos estaticos no requieren autenticacion
        .antMatchers("/bootstrap/**", "/content/**", "/images/**", "/css/**", "js/**").permitAll()
        // Las vistas publicas no requieren autenticacion
        .antMatchers("/", "/login", "/admon/altaUsuario/**", "/rest/inmobiliaria/**", "/inmueble/buscarInmuebles", "/user/verFavoritos/**", "/inmueble/verTodos/**").permitAll()
        // Asignar permisos a URLs por ROLES
        // .antMatchers("admon/**").hasAuthority("ROL_ADMIN")
        .antMatchers("cliente/**").hasAuthority("ROL_USER")
        .antMatchers("user/verFavoritos/**").hasAnyAuthority("ROL_USER", "ROL_ADMIN")
        .antMatchers("admon/panelAdmon/**").hasAnyAuthority("ROL_ADMIN")
        // Todas las demás URLs de la Aplicacion requieren autenticacion
        .anyRequest().authenticated()
        // El formulario de Login no requiere autenticacion
        .and().formLogin().permitAll()
        .defaultSuccessUrl("/", true);
}
```

También se añadieron los tags correspondientes en las vistas para mostrar la información que corresponde en función de si el usuario ha iniciado sesión y los roles que pueda poseer. Un ejemplo lo podemos observar en el menú de navegación de la cabecera.

```
<nav>
    <ul class="menu">
        <li><a href="/" class="activa">INICIO</a></li>
        <li><a href="/inmueble/verTodos">INMUEBLES</a></li>
        <li><a href="../../content/about.html">QUIÉNES SOMOS</a></li>
        <li><a href="../../content/contact.html">CONTACTO</a></li>
        <li><h4 id="usuario">${sessionUserName}</h4></li>
    <sec:authorize access="/isAuthenticated()">
        <li><h4><a href="/Login">Inicia Sesión</a></h4></li>
        <li><h4><a href="/admon/altaUsuario">Regístrate</a></h4></li>
    </sec:authorize>
    <sec:authorize access="isAuthenticated()">
        <li><h4><a href="/user/verFavoritos/${sessionUserName}">Favoritos</a></h4></li>
    </sec:authorize>

    <sec:authorize access="hasAuthority('ROL ADMON')">
        <li><h4><a href="/admon/panelAdmon">Admin. panel</a></h4></li>
    </sec:authorize>

    <sec:authorize access="isAuthenticated()">
        <li><h4><a href="/Logout">Cerrar Sesión</a></h4></li>
    </sec:authorize>
    <%-- La parte con su usuario correspondiente
    <li><h4>Bienvenido ${sesion.nombre }</h4></li>
    <li><h4><a href="/logout">Cerrar Sesión</a></h4></li> --%>
    </ul>
</nav>
```

## **CONCLUSIONES Y MEJORAS DEL PROYECTO**

Algunos puntos de mejora que a los integrantes del equipo les hubiera gustado implementar son:

Tener Modo oscuro/claro

Poder cambiar el idioma entre español e inglés.

Encriptado de las claves de usuario

Realizar test de usabilidad con personas ajenas al proyecto

## **BIBLIOGRAFÍA**

### **EDIX**

Lo aplicado en este proyecto está basado, prácticamente en su totalidad, en lo aprendido con los diferentes materiales ofrecidos por el Instituto Tecnológico de Edix, tanto de sus materiales didácticos, como de lo ofrecido en las clases online impartidas en las diferentes asignaturas y por los diferentes profesores del grado.

### **W3SCHOOLS**

<https://www.w3schools.com>

### **YOUTUBE**

<https://www.youtube.com>