

Centro:

Gaspar Melchor de Jovellanos

Curso: 2021/2022

DOMUS STUDENT



Tutor: Francisco Cortés

Alumnos:

Ricardo Kwapisz Parejo

Juan Horrillo

Jonathan Moreno

Índice

Definición Domus Student

Tejido Empresarial

Requerimientos Funcionales

Diseño UML

Diseño BBDD

Arquitectura Web

Interfaz Web

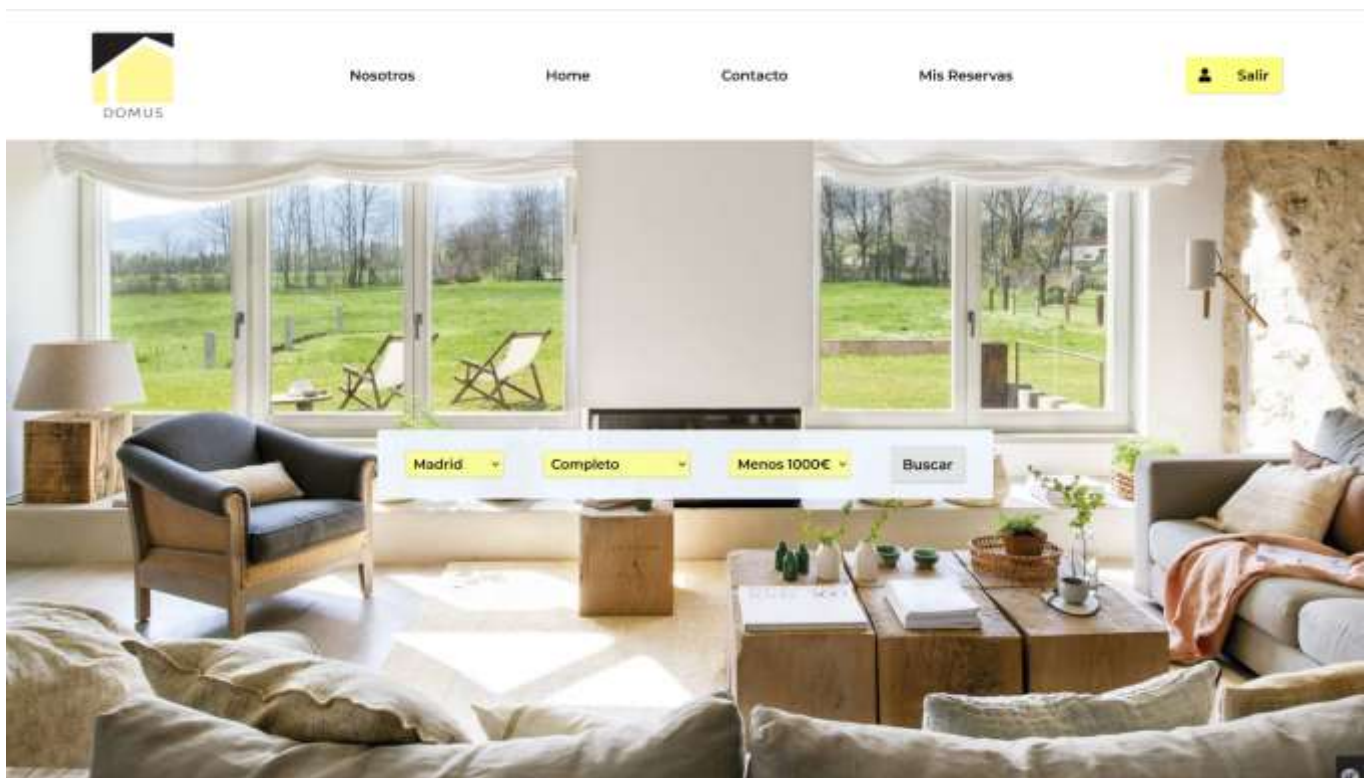
Pruebas

Despliegue

Distribución de horas

1- ¿Qué es Domus Student?

- Domus Student es una página web dedicada al alquiler de pisos para estudiantes de universidad u otros grados de estudio, que hace de intermediaria entre el interesado (cliente) y el dueño del piso.
- Para ello usa una interfaz simple y vistosa, fácil de usar e intuitiva, con diversos pisos en diferentes zonas de España.



(página principal de Domus Student)

2- Tejido Empresarial y requerimientos legales

Domus Student es una inmobiliaria pensada y diseñada para los estudiantes y personas más jóvenes.

Nos introducimos en un sector con gran competitividad y con empresas bastante afianzadas, pero la calidad/precio de nuestros servicios es imbatible, y por eso nos hemos centrado en los estudiantes, debido que a día de hoy hemos observado que los precios son demasiados altos, y suelen ser pisos antiguos, en barrios mal comunicados y sin muchos servicios.

Por ello desde Domus Student apostamos por pisos de la mejor calidad, en las mejores localizaciones de las mejores ciudades de España.

Según el último informe anual de Erasmus+, en el curso 2017/2018 un total de 51321 estudiantes eligieron España como destino Erasmus, sin contar los estudiantes nacionales.

Desde el año 2001, España es el líder indiscutible en recepción de estudiantes de otros países, siendo Alemania, Italia, Francia o Reino Unido, los países más habituales de donde vienen.

Teniendo en cuenta estos datos, Domus ofrece 6 de las mejores ciudades de España, en puntos estratégicos de nuestro país, incluyendo las Islas Baleares

En cuanto a los requerimientos legales que una página web debe tener, Domus Student cuenta con todos los requisitos que se le requieren, teniendo en su disposición y aportando a sus Usuarios:

Aviso Legal

Política de Privacidad

Aviso de cookies

Política de cookies

Datos de contacto

Domicilio social

En cuanto a consideraciones de seguridad, Domus Student cumple con la normativa básica y se puede asegurar qué somos una web fiable y segura para nuestros potenciales clientes.

3- Requerimientos Funcionales

LOGO DOMUS		<div style="display: inline-block; border: 1px solid black; padding: 2px 5px; margin-right: 10px;">Login</div> <div style="display: inline-block; border: 1px solid black; padding: 2px 5px;">Regístrate</div>
<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Sobre Nosotros</div> <div style="border: 1px solid black; padding: 5px;">Quienes somos</div>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Buscador de</div> <div style="border: 1px solid black; padding: 5px;">Alquileres</div>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Contacto</div>

Imagen
Chula

Info

Provincia: [Desplegable]

Nº habitaciones

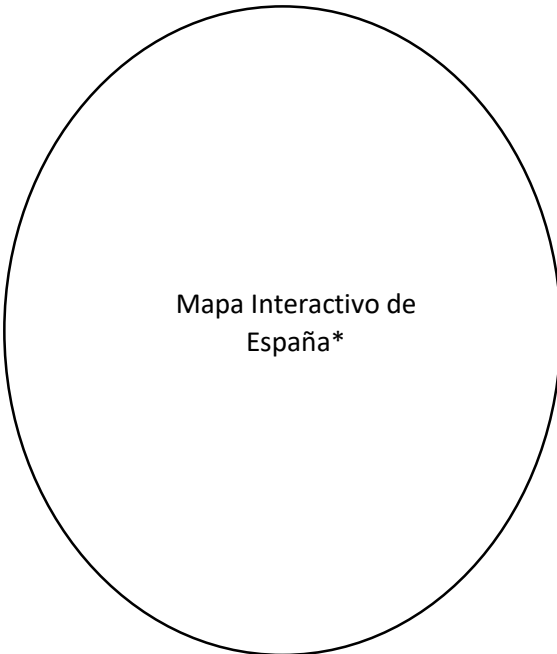
Totales: [Desplegable]

Nº habitaciones disponibles: [Desplegable]

Tamaño: [info]

Metros cuadrados (m2): [info]

[Buscar]



Términos de Uso	Dirección de Empresa	Condiciones de contrato
-----------------	----------------------	-------------------------

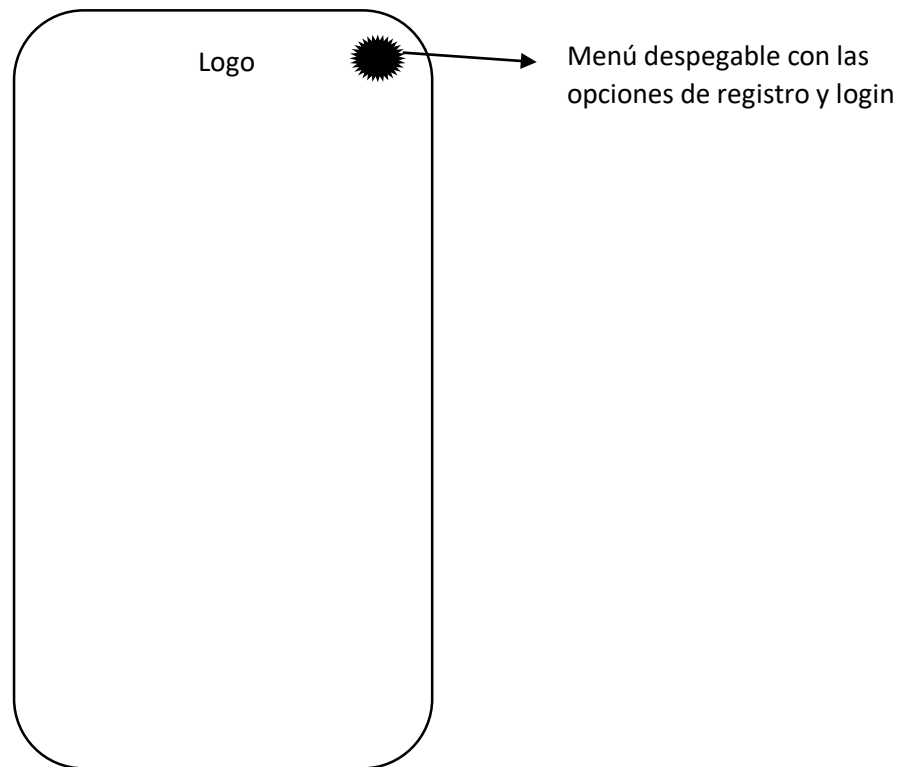
Facebook	Linked In	REDES SOCIALES	Instagram <div style="border: 1px solid black; padding: 2px 5px; display: inline-block;">L</div>
----------	-----------	----------------	---

(primer diseño de la página web de Domus Student)

- Dos barras horizontales en la parte superior de la web.
→ La primera, en la zona de la izquierda, el Logo de la empresa (DOMUS), y en la parte de la derecha, un botón para hacer Login en nuestra página web, y otro para Registrarnos en el caso que no tuviéramos cuenta.
→ La segunda iría justo debajo de la primera, y tendría información sobre la Empresa (quienes somos), un buscador de Alquileres, el cual tendría varios parámetros de búsqueda como Provincia, Número de habitaciones totales y disponibles, etc. Y por último tendría una forma de contacto, email, teléfono, etc.
- En la parte izquierda de la web habría una barra horizontal con imágenes chulas, las cuales podrían tener un efecto de cambio, y justo abajo información.
- En la parte derecha, o central, de la página web habría un mapa interactivo en el cual pincharíamos y saldría información de los diferentes pisos para alquilar.*
- Y por último, en la parte inferior de la página web, un pie de página con diferente información como Términos de Uso, Dirección física de la Empresa, Condiciones de contrato y Redes sociales de la empresa.

*: el mapa interactivo de España sería implementado SOLO si diera tiempo.

➤ Diseño conceptual de la página web para móviles:



- Mismo diseño y concepto que en la página web, pero con un menú desplegable en la parte superior derecha.

4- Diseño UML (Caso de Uso)

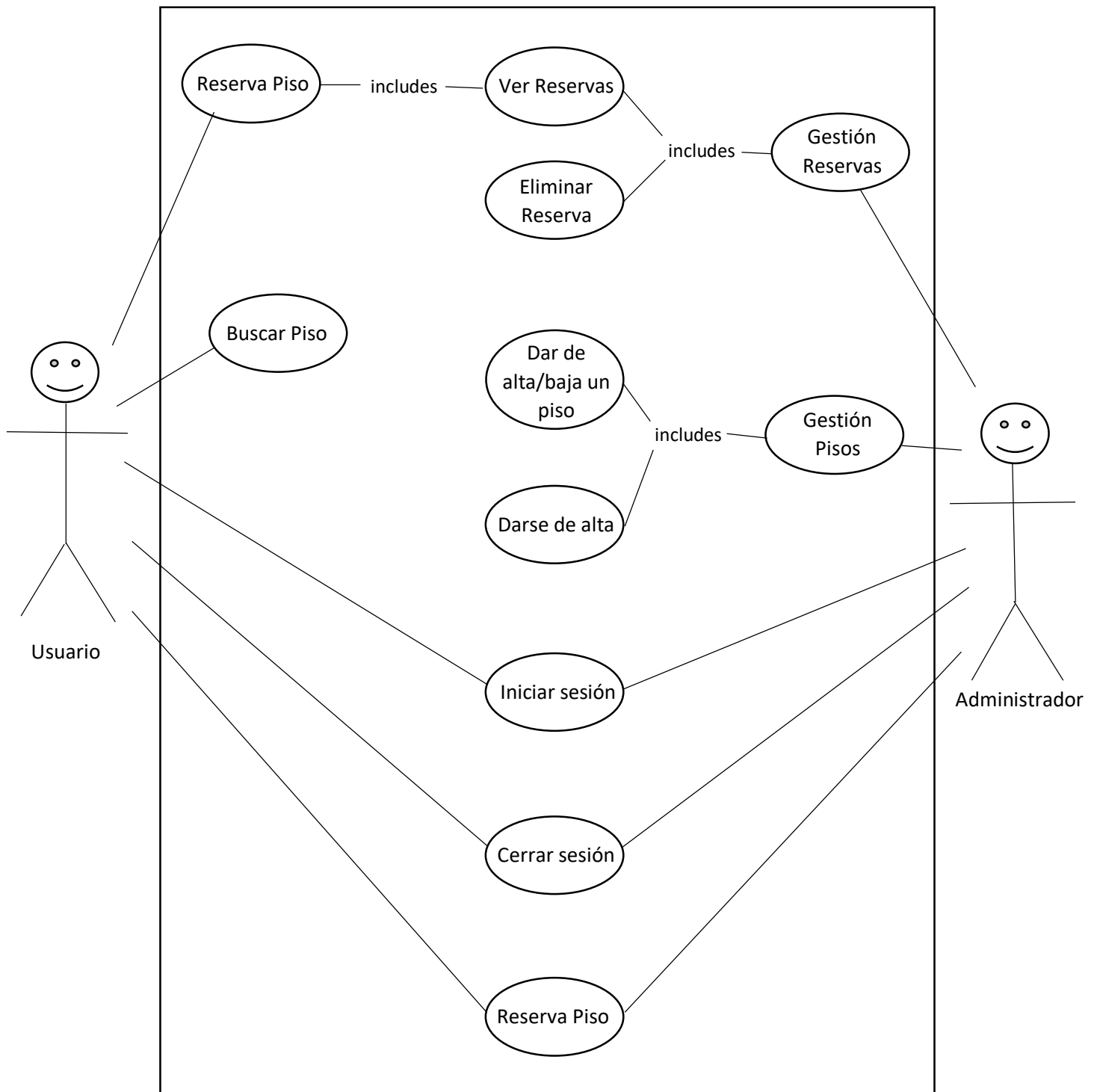
¿Qué es un caso de uso?

Un caso de uso es una representación gráfica de cómo funcionan unos procesos mediante actores y las relaciones con los distintos procesos.

Un actor, es un elemento que inicia el caso de uso, y típicamente representa a usuarios del sistema.

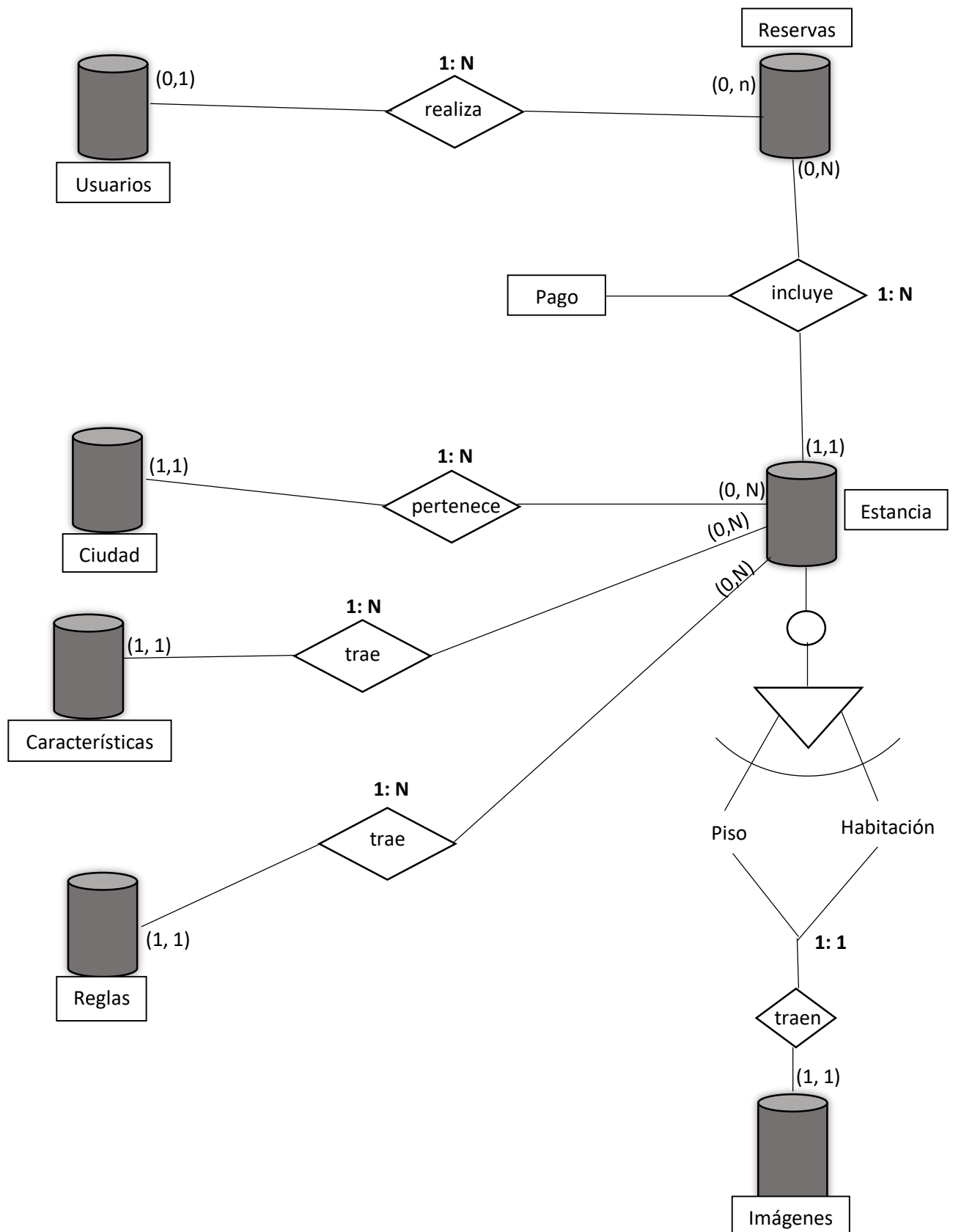
En nuestro caso, tendremos a 2 actores:

- **Usuario:** Este actor puede realizar un total de 5 acciones, 3 compartidas con el otro actor.
 - Puede buscar un piso, y además puede reservar un piso.
 - Entre las acciones compartidas, puede darse de alta como usuario, iniciar sesión o cerrar sesión.
 - **Administrador:** Es el segundo actor, y comparte acciones con el Usuario.
 - En el proceso de Gestión de reservas, incluye ver todas las reservas, realizadas por los usuarios y eliminar una reserva realizada por cualquier usuario.
 - En el proceso de Gestión de Pisos, puede tanto dar de baja como de alta un piso, que son los que luego puede ver un usuario y además puede editar la información de cualquier piso.
 - Tiene otros 3 procesos compartidos con el usuario que son Darse de alta, iniciar sesión y cerrar la sesión.
-



(Diseño UML / Caso de Uso de Domus Student)

5- Diseño BBDD



- Un usuario puede hacer N reservas, y cada reserva es realizada por un Usuario
- Una reserva incluye una estancia, y una estancia es incluida en 0 o N reservas, es decir, un piso puede estar reservado varias veces, para fechas distintas.
- Cada estancia puede ser de tipo Piso completo o Habitación.
- Cada estancia o Piso pertenece a una ciudad, y a cada ciudad pertenecen 0 o N pisos.
- Cada estancia tiene un conjunto de características y ese conjunto puede ser común en varios pisos.
- Cada estancia tiene un conjunto de reglas y ese conjunto de reglas puede ser común en varios pisos.
- Cada estancia tiene un conjunto de imágenes, en cada registro de imágenes, guardamos 4 direcciones a las 4 imágenes de cada piso. También guardamos un nombre UNIQUE como identificador a la hora de insertar nuevo piso, poder asignarle un conjunto de esas imágenes. Las imágenes las guardamos en la carpeta public del proyecto, en la base de datos solo los src asociados.

En nuestra base de datos, los atributos de ConjuntoReglas y CaracterísticasPiso, son, en mayor parte booleanos.

Por ejemplo regla de mascotas si o no (true o false), o si el piso cuenta con garaje o no.

Destacar que en el formulario de dar de alta un nuevo piso, tenemos para elegir 4 configuraciones de piso:

Completo : tiene muchos "true"

Económico: tiene menos atributos "true", no tiene calefacción, no es luminoso, etc.

Y también si tiene playa cercana o no.

6- Arquitectura Web

Cliente-Servidor (3 CAPAS Y 1 NIVEL)

1 Nivel, porque tenemos las 3 capas en el mismo ordenador.

3 capas →

Capa de presentación (Cliente): lo conforma la interfaz de usuario , "lo que se ve". El usuario que utiliza la aplicación, realiza peticiones al backend

(capa de negocio) mediante JavaScript utilizando fetch.

Con fetch realizamos peticiones a rutas de controladores del back, y se manejan las respuestas de estas.

Las respuestas son en formato JSON.

Es decir, la capa de negocio no devuelve nada más que datos en JSON. A partir de esos datos capturados en una promesa de fetch se maneja el DOM de manera dinámica con funciones JavaScript.

```
var zona_misreservas = document.getElementById("zona-misreservas");

function getReservas(){
    let url = new URL("http://localhost:8000/reservas");
    fetch(url)
        .then(function (respuesta) {
            return respuesta.json()
        }).then(function (reservas) {
            console.log(reservas)
            if(reservas.length > 0){
                home();//"Limpia" la pantalla
                document.getElementById("fondo").style.display = "none";
                document.getElementById("swiper-wrap").style.display = "none";
                reservas.forEach(printDivReservas)
                zona_misreservas.style.display = "flex";
            }else{
                Alert.info('No tienes Reservas para ver', 'Sin Reservas', {displayDuration: 6000, pos: 'top'})
            }
        })
}
```

Ejemplo de petición con fetch

Con esto hemos querido implementar un modelo Single Page Application.

(Una Single-Page Application (SPA) es un tipo de aplicación web que ejecuta todo su contenido en una sola página.)

Por tanto, solo utilizamos un archivo de HTML. En nuestro caso el .html es un .twig, que son los templates utilizados por symfony.

(Symfony en principio está pensado para devolver templates .twig con datos , utilizando el modelo MVC , donde la vista la conforman múltiples .twig. En nuestro caso no hemos utilizado MVC y solo tenemos un .html.twig , el cual modificamos dinámicamente con JavaScript.)

Archivos de la capa de presentación :

- -Los css , imágenes y .js están en la carpeta public del proyecto.
- -El marcado se encuentra en templates:
- -Archivo proyecto.html.twig

En los archivos Js utilizamos código vanilla js y algunos elementos visuales con JQuery.

```
/**
 * @Route("/clientesAdmin", methods={"GET"})
 * @param Request $request
 * @return JsonResponse
 * @throws \Exception
 */
public function getAllUser(Request $request, EntityManagerInterface $entityManager, SerializerInterface $serializer)
{
    $users = $entityManager->getRepository(User::class)->findAll();
    $data = $serializer->serialize($users, JsonEncoder::FORMAT);
    return new JsonResponse($data, Response::HTTP_OK, [], true);
}
```

Ejemplo de función con ruta de un controlador de symfony

Capa de negocio: La conforman los distintos controladores de symfony, en los cuales guardamos funciones asociadas a rutas que manejan las peticiones del cliente.

Tenemos un SecurityController , para las peticiones de login y registro.

AdminController para peticiones del administrador.

IndexController para peticiones generales y de usuario cliente.

Capa de datos:

Con el ORM doctrine manejamos la base de datos mediante una capa de abstracción.

Con doctrine se utiliza POO , cada entidad de la base de datos relacional , se representa como un objeto , el cual tiene sus getters y setters.

Estos objetos se encuentran en la carpeta Entity, en la cual hay una clase de cada objeto entidad.

Para realizar consultas a la base de datos, utilizamos el repositorio predeterminado de consultas(findAll(),findOneBy(),findBy(...)),

Y el lenguaje DQL para consultas con doctrine(Similar a SQL).

Dentro de cada objeto se definen los atributos y el tipo de atributo.

También se definen las relaciones @ManyToOne @OneToMany @OneToOne dentro de estos objetos.

```
/**
 * @var \Imágenes
 *
 * @ORM\ManyToOne(targetEntity="Imágenes")
 * @ORM\JoinColumns({
 *   @ORM\JoinColumn(name="id_img", referencedColumnName="id")
 * })
 */
private $idImg;

/**
 * @var \Ciudad
 *
 * @ORM\ManyToOne(targetEntity="Ciudad")
 * @ORM\JoinColumns({
 *   @ORM\JoinColumn(name="id_ciudad", referencedColumnName="id_ciudad")
 * })
 */
private $idCiudad;
```

Ejemplo de relación ManyToOne

Una ventaja que hemos notado utilizando doctrine , es que cuando solicitas por ejemplo una reserva, el IdPiso de esa reserva no es un Int , si no que devuelve directamente el objeto piso con sus atributos:

```
▼ (6) [{...}, {...}, {...}, {...}, {...}, {...}] ⓘ
  ► 0: {idEst: '1', tipoEst: 'piso', precioMes: 1000, fianza: 1200, direccion: 'C/Abanca, Las Rozas', ...}
  ▼ 1:
    direccion: "C/Santander, Las Rozas"
    fianza: 850
    ► idCar: {idCar: '1', numBanos: 2, numHab: 4, television: true, amueblado: true, ...}
    ► idCiudad: {idCiudad: '1', nombre: 'Madrid', __initializer__: null, __cloner__: null, __isInitialize
      idEst: "29"
    ► idImg: {id: '2', nombre: 'piso2', principal: 'images/proyecto/pisos/MADRID/LASROZAS/ROZAS8.png', sk
    ► idReg: {idReg: '2', parejas: false, mascotas: false, fiestas: false, fumar: false, ...}
    latitud: 40.49292
    longitud: -3.87371
    precioMes: 800
    tipoEst: "piso"
    ► [[Prototype]]: Object
  ► 2: {idEst: '30', tipoEst: 'piso', precioMes: 850, fianza: 850, direccion: 'C/Real, Las Rozas', ...}
  ► 3: {idEst: '31', tipoEst: 'habitacion', precioMes: 750, fianza: 700, direccion: 'C/Pinos, Somosaguas', ...}
  ► 4: {idEst: '32', tipoEst: 'piso', precioMes: 600, fianza: 650, direccion: 'C/Pinos, Somosaguas', ...}
  ► 5: {idEst: '33', tipoEst: 'piso', precioMes: 690, fianza: 650, direccion: 'C/Goya, Madrid', ...}
```

console.log() que muestra como las FK son objetos , como idCar

Con ello reducimos las consultas, (no tenemos que capturar el IdPiso y luego buscar el Piso , ya en la propia consulta a reserva se te devuelve un objeto piso con todos sus datos.

El gestor de bases de datos utilizado es MYSQL que se encarga de todo el almacenamiento de datos, el cual recibe solicitudes de almacenamiento o recuperación de información.



The screenshot shows the phpMyAdmin interface with a search bar at the top. Below it, a table list is displayed with columns: Tabla, Acción, Filas, Tipo, Cotejamiento, Tamaño, and Residuo a depurar. The tables listed are: características_piso, ciudad, conjunto_reglas, estancia, imagenes, reserva, and user. The 'user' table is highlighted. At the bottom, there is a summary row for '7 tablas' with a total of 45 rows and a size of 240.0 KB.

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
características_piso	Examinar Estructura Buscar Insertar Vaciar Eliminar	4	InnoDB	utf8mb4_general_ci	16.0 KB	-
ciudad	Examinar Estructura Buscar Insertar Vaciar Eliminar	6	InnoDB	utf8mb4_general_ci	16.0 KB	-
conjunto_reglas	Examinar Estructura Buscar Insertar Vaciar Eliminar	10	InnoDB	utf8mb4_general_ci	16.0 KB	-
estancia	Examinar Estructura Buscar Insertar Vaciar Eliminar	6	InnoDB	utf8mb4_general_ci	16.0 KB	-
imagenes	Examinar Estructura Buscar Insertar Vaciar Eliminar	9	InnoDB	utf8mb4_general_ci	32.0 KB	-
reserva	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	48.0 KB	-
user	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	32.0 KB	-
7 tablas	Número de filas	45	InnoDB	utf8mb4_general_ci	240.0 KB	0 B

Base de datos en phpMyAdmin

Destacar por último , el archivo de configuración .env , en el cual establecemos la conexión con la base de datos y el archivo security.yaml para establecer el logout.

El archivo security.yaml tiene muchas opciones de configuración , pero no hemos usado toda su capacidad debido a nuestra inexperiencia con el framework y que este está diseñado para trabajar con MVC. De hecho se pueden generar las configuraciones y controladores de manera automática utilizando la consola de symfony pero todo ello orientado al modelo MVC.

Resumen tecnologías :

Cliente -> Js(Vanilla y JQuery) , css3 y html5

Servidor -> Symfony PHP con ORM doctrine conectado a MYSQL

Resumen de estructura :

→ Front , un único archivo .html.twig(html), archivos JS con peticiones fetch asociadas a eventos que reciben datos en Json.

A partir de estos datos se modifica el DOM de manera dinámica, evitando que el servidor devuelva HTML.

→Back : Controladores de symfony con rutas , que manejan las peticiones del cliente.

Cada función asociada a una ruta devuelve datos en JsonResponse.

En entity tenemos los objetos entidades del mapeo de la base de datos, necesarios para hacer consultas utilizando doctrine, el ORM predeterminado de symfony.

→BBDD : Para conectar symfony con la base de datos , modificamos el archivo .env de configuración.

Con Doctrine manejamos la base de datos , inserciones , borrados y consultas mediante DQL y el repositorio predeterminado.

Resumen Roles :

Administrador ROLE_ADMIN

Cliente ROLE_USER

El cliente se puede registrar, logear, realizar reservas y ver sus reservas.

El formulario de reservas tiene un control de disponibilidad, entre fechas.

Elegimos la fecha de inicio y fin y comprobamos si está disponible el piso en esas fechas. Estará disponible si no existe ninguna otra reserva de ese piso entre dichas fechas.

El administrador cuenta con un menú de administración , donde puede ver listados de pisos, reservas y clientes.

Puede borrar , editar e insertar nuevos pisos. Editar o borrar clientes .Y borrar y ver todas las reservas.

Por último, destacar que los clientes pueden registrarse siempre que quieran. Sin embargo el administrador lo hemos introducido de manera manual en la base de datos, pues hemos configurado ROLE_USER como predeterminado en el registro.



Primera pantalla (Usuario logeado)

7- Interfaz web

En la primera pantalla, tenemos un header , con el enlace home a esa misma primera pantalla , el enlace “Mis reservas” que nos saldrá solo si está logeado el usuario y nos mostrará las reservas , en un listado de divs.

El botón en el extremo derecho nos permite ir al formulario de registro y login, o salir de la sesión.

También , dentro de esa primera pantalla tenemos un filtro de búsqueda de pisos, por provincia tipo y precios.

Y por último tenemos un Swiper con imágenes de las distintas provincias, en los cuales si pinchamos nos retornara los pisos de dicha provincia.



Swiper con imágenes de las distintas ciudades , si pulsas te devuelve los pisos de esa ciudad

La tranquilidad de acertar, tu nueva vivienda a un click

Login

Correo Electrónico

Contraseña

Entrar

Registro

Correo Electrónico

Nombre

Primer Apellido

Segundo Apellido

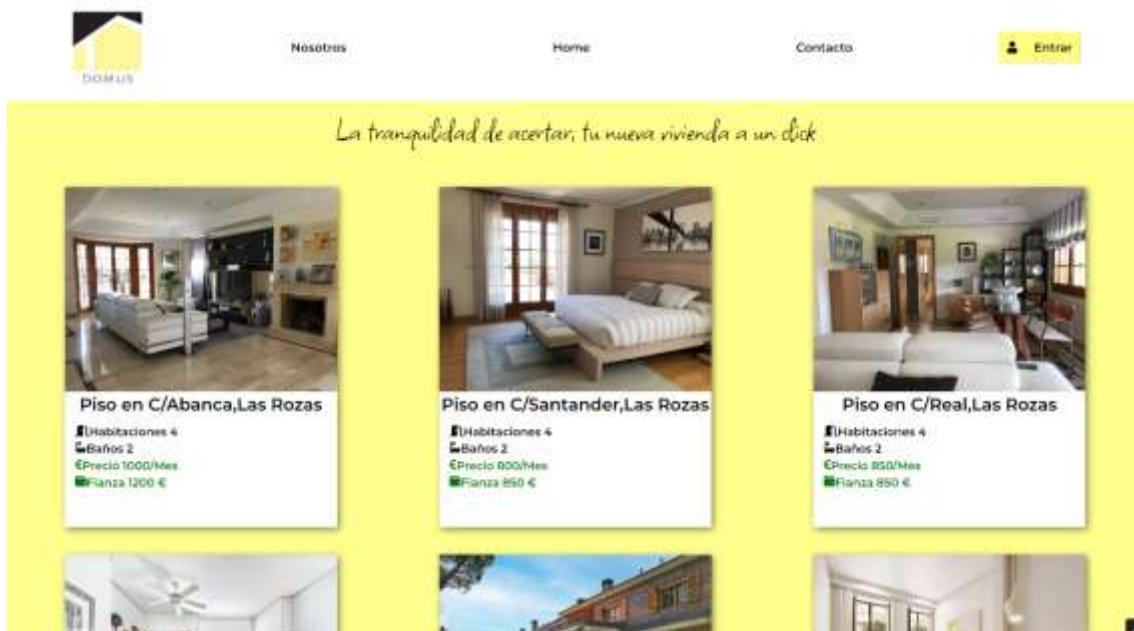
Contraseña

Repetir la contraseña

Regístrate

Formulario de login y registro

El paso natural después de la primera pantalla , seria el formulario de login o si has decidido buscar piso directamente , una pantalla con divs de pisos.



Divs de los pisos de Madrid

Una vez pinches en uno de esos divs te llevará a una pantalla con información de ese piso



En la información nos aparece más fotos, características precio , reglas del piso y un mapa con la ubicación (con coordenadas) del piso.

También hemos implementado un botón con la bandera de Reino Unido, que nos muestra la información en inglés.



El botón de la derecha es fixed, y nos redirige al formulario de login en caso de que no hayamos iniciado sesión, o si lo hemos hecho , nos lleva al formulario de reserva.

Este es un formulario por pasos, donde primero comprobamos la disponibilidad del piso en dos fechas , y luego introducimos numero de tarjeta y teléfono de contacto.

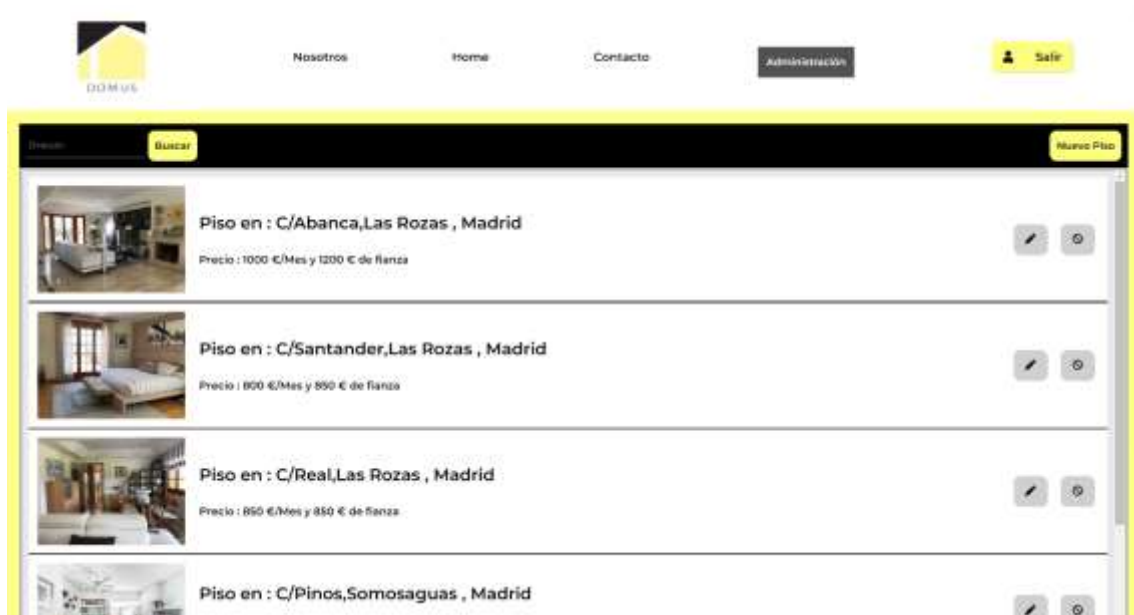




Una vez completemos la reserva, podremos verla en “Mis Reservas”



En cuanto al administrador , este puede ver todos los pisos , clientes y reservas y gestionarlos, como realizar edición , borrado o insertado.



Listado de pisos para editar eliminar e insertar nuevo piso. Tenemos un filtro por dirección.

The screenshot shows a "Nuevo Piso" form for editing apartment data. The form is divided into several sections:

- Datos:**
 - Tipo: ☒ Piso Completo ☐ Habitación
 - Ciudad:
 - Dirección:
 - Precio: Fianza:
- Reglas:**
 - ☐ Parejas ☐ Mascotas ☐ Fiestas ☐ Fumar
- Ubicación:**
 - Latitud: Longitud:
- Características:**
 - ☒ Completo ☐ Económico
 - ☐ Con Playa ☒ Sin Playa
- Imágenes:
-

Formulario de editar Piso. Se rellena los datos automáticamente y se pueden cambiar.

Editar piso de C/Abanca, Las Rozas

Datos:

Tipo: ☒ Piso Completo ☐ Habitación

Ciudad:

Dirección:

Precio: Fianza:

Reglas:

☐ Parejas ☒ Mascotas ☐ Fiestas ☒ Fumar

Ubicación:

Latitud: Longitud:

Características:


☒ Completo ☐ Económico

☐ Con Playa ☒ Sin Playa

Imágenes:

[Guardar](#)

Formulario Nuevo Piso


[Nosotros](#)
[Home](#)
[Contacto](#)
[Administración](#)
[Salir](#)

[Buscar](#)

Usuario: ricl@gmail.com

[Editar](#) [Borrar](#)

Listado de Clientes con editar y borrar





Nosotros Home Contacto Administración  Salir



Editar Usuario: ric1@gmail.com

Usuario:
 Nombre:
 Puesto:
 Nueva Contraseña:

Formulario editar Cliente



Nosotros Home Contacto Administración  Salir

Buscar	
Piso en: C/Santander, Las Rozas, Madrid Usuario: ric1@gmail.com, Ricardo Kwepisz Parejo Reserva desde 8/6/2022 a 26/8/2022	
Piso en: C/Pinos,Somosaguas, Madrid Usuario: ric1@gmail.com, Ricardo Kwepisz Parejo Reserva desde 8/6/2022 a 26/8/2022	

Listado de Reservas, con filtro y eliminar

Diseño Responsive :





Nosotros

Home

Contacto

 Entrar



Madrid ▾

Completo ▾

Menos 1000€ ▾

Buscar

200         

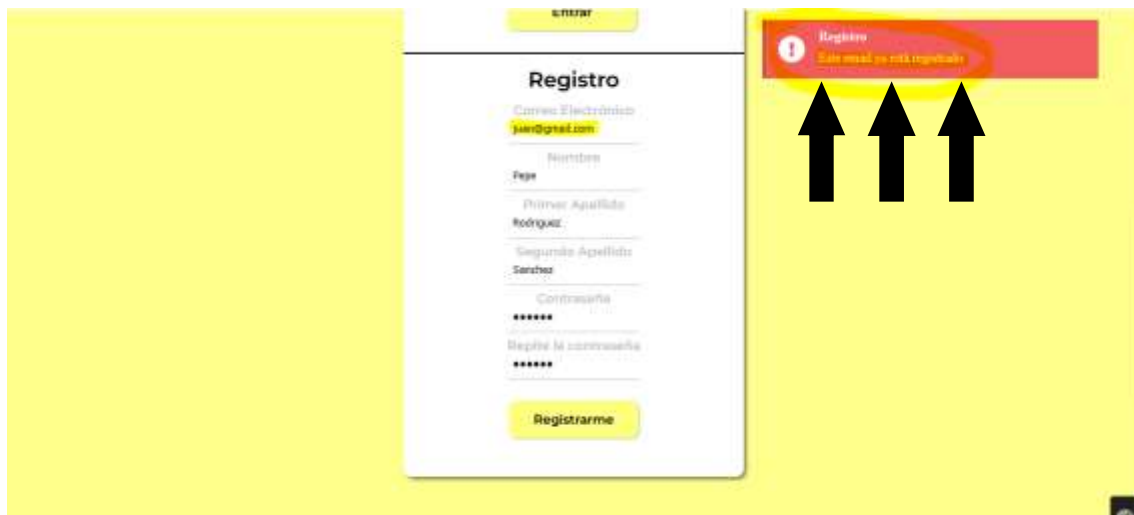
8- Casos de Prueba

Un caso de prueba es un conjunto de condiciones o variables bajo las cuáles se determinará si una aplicación, un sistema de software o una característica resulta o no aceptable.

Se pueden realizar muchos casos de prueba para determinar si es aceptable o no, y es recomendable hacer tanto casos de prueba con resultado positivo, como con resultado negativo, para así comprobar que todas las características de nuestra aplicación o software funcionan correctamente.

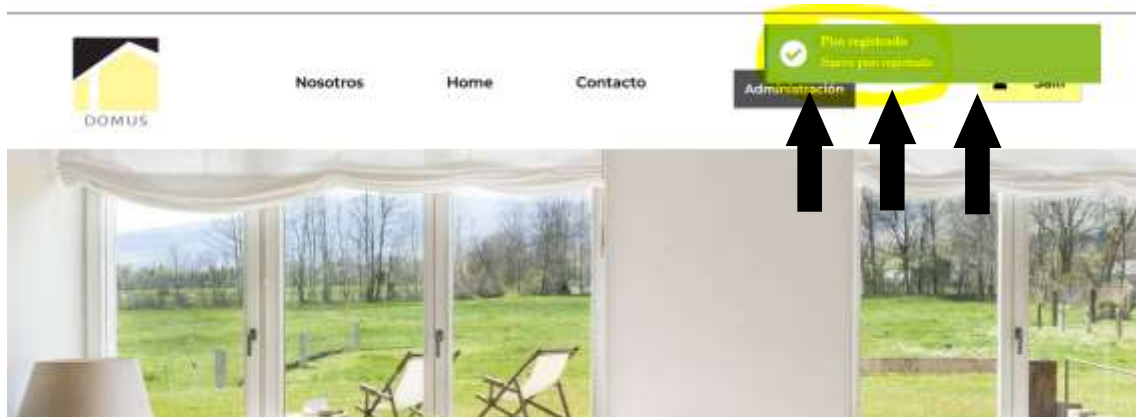
A continuación, hare algunos casos de prueba específicos para comprobar que algunas características de nuestra página web funcionan debidamente.

- Caso de Prueba N°1: Dar de alta 2 usuarios con el mismo email.
 - Datos: creo un usuario con email `juan@gmail.com`
 - Resultado previsto: ERROR
 - Resultado final: ERROR



(Intentando dar de alta otro usuario con el mismo email)

- Caso de Prueba N°2: Alta de piso con administrador.
 - Datos: usuario y contraseña del administrador
 - Resultado previsto: OK
 - Resultado final: OK



(Intentando dar de alta un piso con el administrador)

- Caso de Prueba N°3: Alta de piso con usuario normal.
 - Datos: usuario y contraseña de un usuario normal.
 - Resultado Previsto: ERROR
 - Resultado Final: ERROR



(página principal de un usuario administrador)



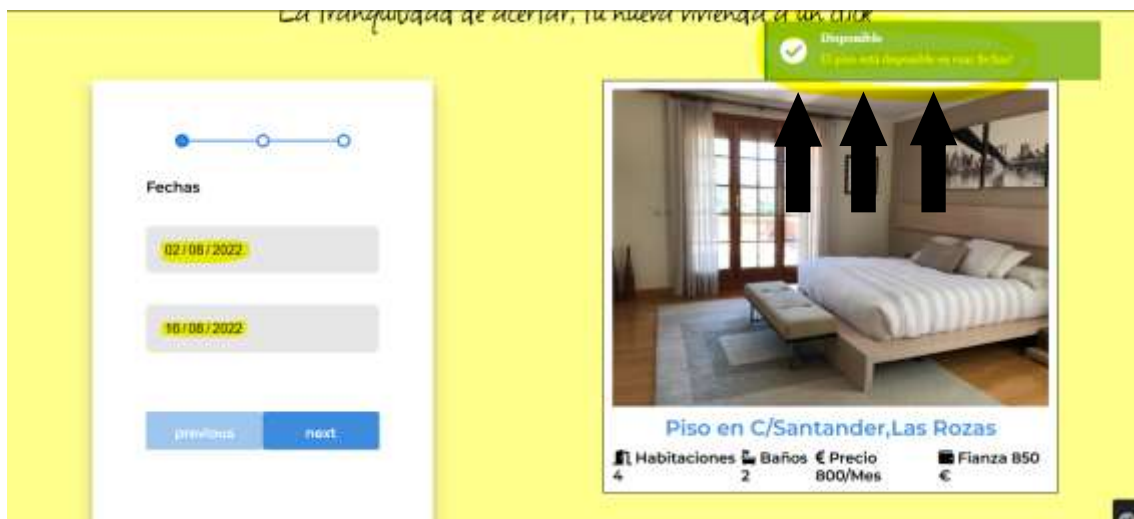
(página principal de un usuario normal)

- Caso de Prueba N°4: Dar de alta otro administrador idéntico al ya existente.
 - Datos: creo un administrador con los mismos datos.
 - Resultado Previsto: ERROR
 - Resultado Final: ERROR



(Intentando dar de alta otro administrador con los mismos datos)

- Caso de Prueba N°5: Comprobar que un piso está disponible en unas fechas a elegir.
 - Datos: login de usuario normal y fechas a elegir.
 - Resultado Previsto: OK
 - Resultado Final: OK



(Comprobando la disponibilidad de un piso en unas fechas a elegir)

9- Despliegue

Requisitos generales

Despliegue en local:

Lo primero que debemos hacer es ir a la página oficial de composer y descargar el .exe (en el caso de Windows), luego lo instalamos.

Composer es un manejador de dependencias el cual permite instalar fácilmente framework, librerías y mucho más de manera automatizada quitándonos el dolor de cabeza de realizar estas instalaciones de manera manual.

Ahora abrimos nuestra consola CMD nos ubicamos dentro de nuestro directorio web (XAMPP o WAMPP etc.) y ejecutamos:

```
composer create-project symfony/framework-standard-edition proyecto-symfony/
"3.*"
```

En dónde "proyecto-symfony" es el nombre del proyecto PHP y "3.*" es la versión del framework en donde especificamos que emplee la 3 en adelante.

Para instalar symfony y doctrine , necesitamos usar Composer.

¿Qué es composer ?

Composer es un gestor de dependencias para PHP (Similar a lo que npm es para JavaScript o pip para Python).

Es una aplicación PHP que ayuda a administrar las librerías desarrolladas por terceros que vas a incorporar a tu proyecto.

Con composer creamos el proyecto symfony con: `composer create-project symfony/skeleton my-project`

Este proyecto lo generamos dentro de htdocs con xampp.

Para utilizar la aplicación hemos utilizado Xampp ,ya que utilizamos mysql de xampp, dentro de htdocs alojamos el proyecto generado con symfony

Para ejecutar el proyecto ,primero mediante cmd nos movemos a la carpeta del proyecto : `cd ... proyecto`

Una vez estamos en el proyecto ejecutamos el comando :

`php -S localhost:8000 -t public/` (Este comando lo hemos utilizado en clase)

* El fichero de la base de datos es `proyectofinal.sql`

En un principio generamos la base de datos con sql, una vez ya estaba generada fuimos implementando con el tiempo mas campos y alguna modificación , la cual la hicimos directamente con PHPMysqlAdmin por comodidad.

Cabe destacar que hay que actualizar el mapeo de la base de datos cada vez que se modifican las entidades o se hace algun añadido.

Con la consola de symfony podemos generar el mapeo de las entidades de forma automática

`php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity`

El comando generará las entidades de las tablas en tu base de datos en el formato de anotación en el directorio `app/src/Entity`

Son entidades simples sin getters ni setters, por lo que no está completo en este momento. Ahora debes ejecutar el siguiente comando para generar los getters y setters de las entidades:

`php bin/console make:entity --regenerate App`

10- Distribución de las Horas

