MEMORIA FINAL DE PROYECTO

SCRAPING INMOBILIARIO

HOME-FINDER S.L

CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES WEB

AUTOR

JUAN CARLOS MORENO ADÁN

TUTOR INDIVIDUAL

JESÚS VIVES CÉSPEDES

ÍNDICE

1. INTRODUCCIÓN

Este documento recoge el trabajo realizado para el módulo de Proyecto del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web (DAW).
El objetivo principal de este módulo es aplicar de forma integrada los conocimientos adquiridos a lo largo del ciclo, mediante el desarrollo completo de una aplicación web real, desde su fase de análisis hasta su implementación final.

En este caso, el proyecto consiste en el desarrollo de una plataforma web para la comparación y gestión de reservas de inmuebles mediante el scrapeo de diferentes portales inmobiliarios, que permite a los usuarios autenticados seleccionar un inmueble y reservar una visita en una fecha y hora disponibles, ver sus reservas y generar el pdf de la misma. Por su parte, el perfil administrador puede gestionar usuarios y generar informes en formato PDF.

Durante el desarrollo se han aplicado tecnologías y herramientas como PHP, Laravel, JavaScript, MySQL, Bootstrap y Vite, siguiendo lo aprendido en las prácticas y en el curso como organización del código y diseño de interfaz.

2. INTRODUCCIÓN EN INGLÉS

This document presents the work carried out for the Project module of the Higher Education Degree in Web Application Development (DAW).

The main objective of this module is to integrate the knowledge acquired throughout the course by fully developing a real web application, from its analysis phase to its final implementation.

In this case, the project consists of the development of a web platform for comparing and managing property reservations through scraping data from different real estate portals. It allows authenticated users to select a property and book a visit at an available date and time, view their reservations, and generate a PDF of the booking.

The administrator profile, on the other hand, can manage users and generate reports in PDF format.

During development, technologies and tools such as PHP, Laravel, JavaScript, MySQL, Bootstrap, and Vite have been applied, following the practices and lessons learned in the course, such as code organization and interface design.

3. OBJETIVOS

El objetivo del proyecto es el scrapeo de ofertas inmobiliarias, en el que permitiría poder ver las ofertas de diferentes webs inmobiliarias y hacer una comparación de los precios de los inmuebles en las diferentes zonas de la península. Permitiendo a los usuarios registrados poder realizar una reserva donde se guardará en su perfil dicha reserva y poder descargar el pdf de la fecha y hora establecida.

3.1. Objetivos fase actual

El objetivo actual es el scrapeo de los inmuebles de un portal inmobiliario. Que el usuario pueda acceder a ver el inmueble(redirigirse a la web del portal inmobiliario) y la reserva del mismo en caso de querer guardar dicho inmueble.

3.2. Objetivos fases futuras

El Objetivo de fases futuras sería la implementación de hasta 5 portales o más, para conseguir una base de datos o comparación mayor. Para ello haría falta contactar con los portales más conocidos como Idealista, Solvia, Fotocasa, etc.. pidiendoles el código de su API, para poder realizar el scrapeo sin que su sistema de bloqueo llegara a interferir en la web y en el scrapeo de sus inmuebles. A demás, se pasaría de mostrar 9 inmuebles a más de 100, se barajaría la posibilidad de añadir 1 ó 2 imágenes en el scrapeo para ver el inmueble. Dependiendo de la cantidad de carga para el servidor y la base de datos. Ya que con el peso de 1 imagen, se puede almacenar los datos de 50 inmuebles por poner un ejemplo. También la verificación del usuario mediante DNI o código de seguridad para certificar que sea un humano y no un robot que acaparara los inmuebles. Se metería ads o publicidad en los lados para recibir algún tipo de ingreso, incluso el pago de 5€ por la reserva de un

inmueble (como gastos de gestión) que se hablaría con los portales para acordar un reparto equitativo.

4. PLANIFICACIÓN

Para la realización del proyecto se han tardado cerca de 2-3 meses.

Se ha estimado cerca de 1 mes para la creación de la web solo en el apartado visual, filtro y creación de usuario junto a sus pequeñas características(borrar usuario, cambiar contraseña, cambiar nombre y correo. También el rol pero solo podrían los admins) con los elementos antes del scrapeo. Utilizando Laravel, html, css, sass y boostrap y Vue.

Luego casi 1 mes y medio más para implementar el scrapeo y la reserva de las ofertas aprendiendo cual sería el mejor método ya que hubieron dificultades a la hora de buscar maneras. Hay diferentes métodos como Goutte en el que se tuvo que abandonar y rehacer al tratarse de una tecnología anticuada. Se valoró entre Dusk Y Symfony. Dusk se pensó debido a que recrea un navegador interno para engañar a las webs y evitar el bloqueo de bots, ya que al hacer scraping, suelen bloquear la recopilación de información. Al final se decidió por Symfony por ser más sencillo y ser una tecnología más usada por las empresas(Muchas ofertas en Linkedin piden gente que sepa PHP con Symfony).

Los 15 días restantes se usaron en subir el programa al servidor de AWS(Amazon Web Service) y revisar cualquier fallo que hubiera que pulir.

4.1. Tabla de hitos

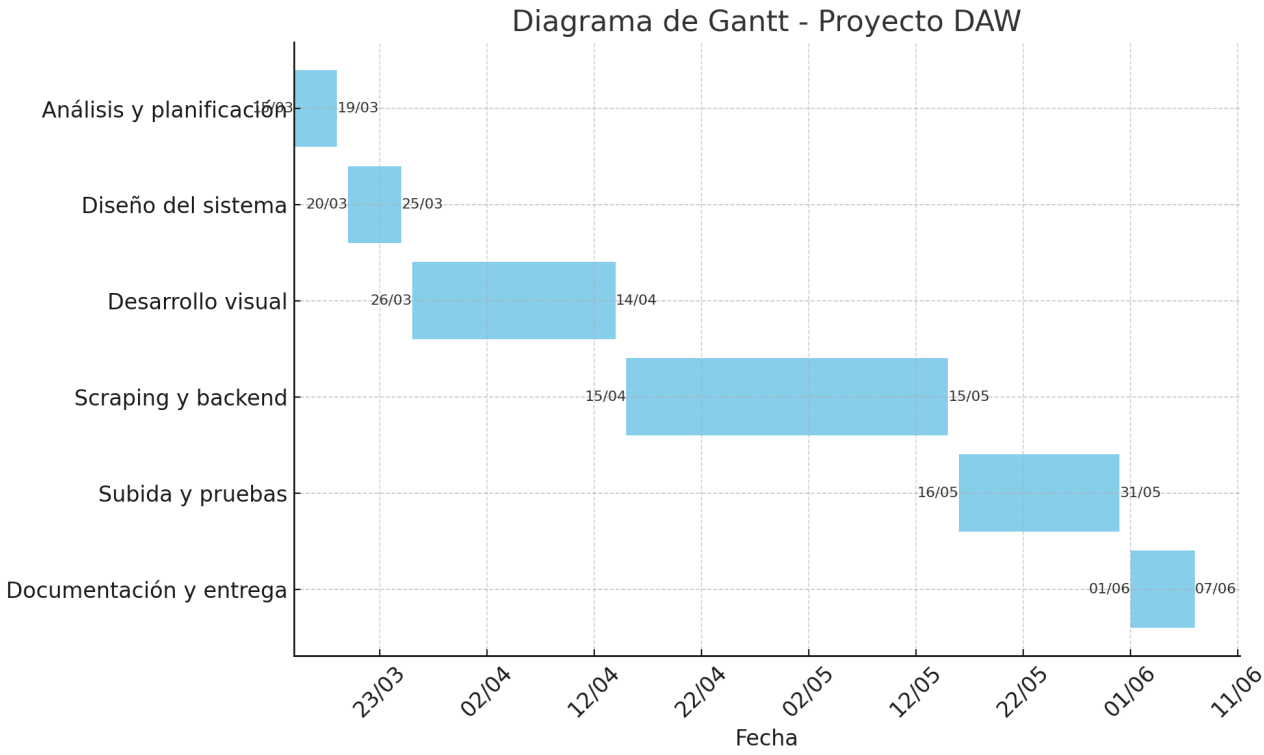
Durante el desarrollo del proyecto se ha trabajado intensamente durante un período aproximado de dos a tres meses. A continuación, se desglosan las fases principales, tareas asociadas, estimación de duración y fechas aproximadas:

Nº	Fase del Proyecto	Tareas Principales	Duración (días)	Fecha Inicio	Fecha Fin
1	Análisis y planificación	-Definición de requisitos -Estudio del contexto y	4 días	15/03/2025	19/03/2025

		herramientas -Valoración de tecnologías (Laravel, Vue, Symfony...)			
2	Diseño del sistema	-Diseño de base de datos -Estructura de vistas y componentes -Mockups iniciales	5 días	20/03/2025	25/03/2025
3	Desarrollo visual	-Implementación de interfaz -Registro y login de usuario -Gestión de roles y perfil	20 días	26/03/205	14/04/2025
4	Scraping y backend	-Investigación de tecnologías de scraping (Goutte, Dusk, Symfony) -Implementación final con Symfony -Lógica de reservas	30 días	15/04/2025	15/05/2025
5	Subida y pruebas	-Migración al servidor AWS -Configuración de entorno -Revisión de errores y últimos ajustes	15 días	16/05/2025	31/05/2025
6	Documentación y entrega	-Redacción de la memoria	4 días	01/06/2025	07/06/2025

	-Capturas, estilos y explicación técnica del proyecto

4.2. Diagrama de Gantt



5. ANÁLISIS

5.1. Estado del arte

Actualmente lo más parecido o existente es Idealista en términos de inmuebles, pero al final han pecado o transformado como digo yo, en una inmobiliaria más, ya que permiten a la gente crear sus anuncios de sus casas y tampoco dejan ver la información de otras webs.

Y en lo referente a que me inspiró, se podría decir que fue Trivago, un portal que nació para comparar las ofertas de viajes y hoteles. Pues viendo la necesidad de vivienda y el grave problema que está teniendo España y muchos otros paises. Me

pareció curioso que no existiera algo parecido cuando los negocios salen a raíz de las necesidades de las personas.

5.2. Funcionalidades

5.2.1.1. Fase 1 – Análisis y planificación

- Definición de requisitos funcionales y técnicos.
- Selección de tecnologías: Laravel, Bootstrap, Vue.js, Symfony para scraping.
- Estudio de roles: usuario normal y administrador.

5.2.1.2. Fase 2 – Diseño del sistema

- Diseño de la base de datos relacional con tablas: usuarios, inmuebles, reservas.
- Creación de estructura de vistas con Blade y componentes reutilizables.
- Elaboración de mockups iniciales para flujos principales.

5.2.1.3. Fase 3 – Desarrollo visual y gestión de usuarios

- Registro de usuarios con validación y confirmación de correo.
- Autenticación y login de usuarios.
- Gestión de perfil (editar nombre/email, cambiar contraseña).
- Gestión de roles: los administradores pueden editar/eliminar otros usuarios.
- Interfaz intuitiva y responsive adaptada con Bootstrap.
- Filtro avanzado de inmuebles por ciudad, tipo, estado, precio, habitaciones.

5.2.1.4. Fase 4 – Scraping y reservas

- Sistema de scraping de portales inmobiliarios usando Symfony DomCrawler.
- Automatización para importar datos de inmuebles.
- Visualización y búsqueda de resultados en tiempo real.
- Sistema de reservas de inmuebles:
    - Validación de horario (09:00–19:00).
    - Restringido el domingo.
    - Evita solapamiento de reservas.
    - Muestra mensaje si la franja ya está ocupada.
- Generación de PDF personalizado tras la reserva.
- Panel de usuario para ver y cancelar reservas.
- Admin puede ver todas las reservas.

5.2.1.5. Fase 5 – Subida y pruebas

- Despliegue en servidor AWS (Linux).
- Configuración de entorno: base de datos, composer, npm, Vite.
- Revisión de errores.
- Validación de funcionalidades completas en entorno real.

5.2.1.6. Fase 6 – Documentación y entrega

- Redacción de memoria técnica del proyecto.

- Capturas, código explicado.
- Tabla de hitos, funcionalidades y Gantt.
- Entrega de versión final funcional.

6. DISEÑO

6.1. Requisitos técnicos

A partir de los requisitos funcionales establecidos previamente, se han definido los siguientes requisitos técnicos que permiten llevar a cabo las funcionalidades descritas. Estos requisitos se han resuelto mediante la elección de tecnologías adecuadas, paquetes del framework Laravel, servicios de terceros y desarrollos específicos cuando ha sido necesario.

-Autenticación y gestión de usuarios

Para implementar el sistema de registro, inicio de sesión, gestión de roles y edición de perfil, se ha utilizado el sistema de autenticación que Laravel ofrece por defecto. Se han aplicado middlewares como auth para proteger rutas privadas y el paquete Spatie/laravel-permission para asignar y controlar los distintos roles de usuario (admin y cliente). Los formularios se validan en el backend para garantizar la integridad de los datos.

-Gestión de reservas

Las reservas se almacenan en una base de datos relacional (MySQL), asegurando que no se pueda reservar el mismo inmueble en la misma fecha y hora mediante validaciones personalizadas en el backend. Se ha utilizado la clase Carbon para trabajar con fechas y horas, evitando reservas en domingos o fuera del horario permitido (09:00 - 19:00). Además, se ha implementado un sistema visual que alerta

al usuario en caso de que una franja ya esté ocupada.

-Scraping de ofertas inmobiliarias

Una parte central del proyecto ha sido la automatización de la recopilación de ofertas inmobiliarias mediante técnicas de scraping. Tras valorar diferentes tecnologías (como Goutte y Laravel Dusk), se optó por utilizar Symfony DomCrawler por su simplicidad, soporte actualizado y facilidad de integración con Laravel. El scraping se gestiona desde controladores específicos y almacena la información extraída directamente en la base de datos.

Generación de PDFs

Para permitir la descarga de información de usuarios y reservas en formato PDF, se ha empleado el paquete barryvdh/laravel-dompdf, ampliamente utilizado en Laravel. Este paquete permite generar PDFs desde vistas Blade personalizadas con Bootstrap, respetando el diseño de la aplicación.

Interfaz y diseño responsivo

La interfaz del proyecto ha sido desarrollada con HTML5, CSS3, Bootstrap 5, y parcialmente con SASS para mejorar la estructura del CSS. El uso de componentes Blade permite mantener una arquitectura ordenada y reutilizable. En la parte frontend, se ha utilizado Vite como empaquetador moderno, lo que mejora el rendimiento en desarrollo y facilita la compilación para producción.

Subida a servidor y despliegue

El proyecto ha sido desplegado en un servidor de Amazon Web Services (AWS), configurando el entorno con PHP, MySQL y Node.js. Se ha realizado la instalación de dependencias mediante Composer y NPM, y se han ejecutado los comandos npm run build y php artisan migrate --seed para dejar la aplicación lista en producción. La seguridad se ha reforzado deshabilitando el acceso a archivos sensibles y protegiendo rutas críticas mediante autenticación.

6.2. Arquitectura web

```
[ Usuario ]
    |
    v
[ Navegador ]
    |
    v
[ Laravel (Controladores) ]
    |
    v
[ Modelos -> Base de datos MySQL ]
    |
    v
[ Scraping (Symfony DomCrawler + Guzzle) ]
    |
    v
[ Vistas (Blade + Bootstrap + Vite) ]
```

La arquitectura de este proyecto se basa en el modelo MVC (Modelo-Vista-Controlador), implementado mediante el framework Laravel, lo que permite una clara separación de responsabilidades entre la lógica de negocio, la presentación y la gestión de datos.

Modelo MVC

- **Modelo (Model)**: Se encarga de representar y gestionar la lógica de acceso a los datos. En este proyecto, modelos como Usuario, Reserva e Inmueble están directamente relacionados con las tablas de la base de datos y encapsulan las reglas de validación, relaciones y consultas frecuentes.

- **Vista (View):** Las vistas están desarrolladas usando el sistema Blade de Laravel y se componen de HTML, CSS, Bootstrap 5 y algo de SASS. Estas

vistas muestran la información al usuario de forma dinámica y responsiva. Además, se emplean layouts comunes para mantener coherencia en la interfaz.

- **Controlador (Controller):** Gestiona la lógica de la aplicación y actúa como intermediario entre los modelos y las vistas. Los controladores validan las solicitudes, ejecutan la lógica de negocio (por ejemplo, evitar reservas duplicadas o filtrar inmuebles) y devuelven respuestas al usuario.

Servicios adicionales y tecnologías utilizadas

- **Scraping:** Implementado con Symfony DomCrawler y Guzzle para recoger información de portales inmobiliarios. Este proceso se gestiona desde Laravel mediante rutas protegidas, y almacena los resultados directamente en la base de datos.

- **PDFs:** El sistema genera documentos PDF con barryvdh/laravel-dompdf, permitiendo descargar reservas y listados de usuarios.

- **Autenticación y Roles:** Gestionado con el sistema nativo de Laravel y el paquete Spatie/laravel-permission, permitiendo diferentes niveles de acceso (admin y usuario).

- **Frontend y recursos:** Se usa Vite como herramienta de construcción para compilar los recursos del frontend de manera rápida y moderna.

- **Base de datos:** Se utiliza MySQL, gestionada mediante migraciones de Laravel que facilitan la creación y mantenimiento del esquema.

Despliegue

- La aplicación ha sido desplegada en un servidor con **Apache y PHP** dentro de un entorno **AWS (Amazon Web Services)**.

- Aunque no se han utilizado contenedores como Docker, el sistema está preparado para ser adaptado a tecnologías basadas en contenedores si se requiriera en el futuro.

- Composer y Node.js fueron instalados en el servidor para gestionar dependencias tanto de backend como de frontend (npm run build, composer install).

6.3. Diseño back-end

Para el desarrollo del back-end se ha utilizado el framework Laravel, uno de los más populares en el ecosistema PHP por su estructura limpia, soporte nativo para MVC (Modelo-Vista-Controlador) y una amplia comunidad que mantiene numerosos paquetes y utilidades adicionales.

Las particularidades serían un sistema de roles, scraping y sistema de reservas.

6.3.1. Modelo de datos

Usuario:

```php
class Usuario extends BaseAuthenticatable
{
    use HasApiTokens, HasFactory, Notifiable, HasRoles;

    protected $guard_name = 'web';

    protected $table = "usuarios";
    protected $fillable = ["nombre", "email", "password", "rol"];
    public $timestamps = false;
    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

```

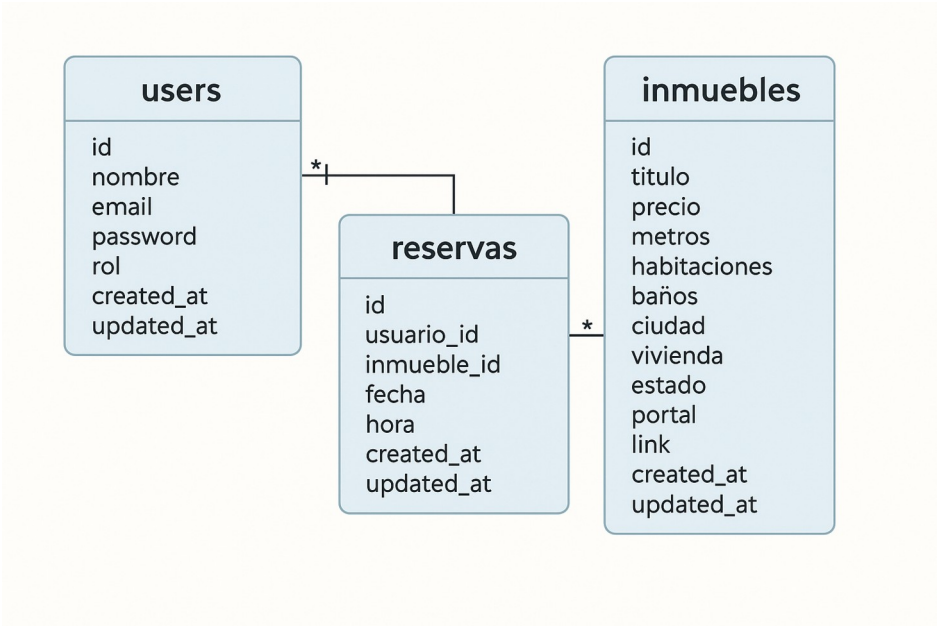
Inmueble:

```php
class Inmueble extends Model
{
    use HasFactory;

    protected $table = 'inmuebles';

    protected $fillable = [
        'titulo',
        'precio',
        'link',
        'metros',
        'habitaciones',
        'banos',
        'operacion',
        'vivienda',
        'estado',
        'ciudad',
        'portal',
    ];

    public function reservas()
{
    return $this->hasMany(\App\Models\Reserva::class, 'inmueble_id');
```

Reserva:

```php
class Reserva extends Model
{
    use HasFactory;

    protected $table = 'reservas';

    protected $fillable = [
        'usuario_id',
        'inmueble_id',
        'fecha',
        'hora',
    ];

    public function usuario()
    {
        return $this->belongsTo(\App\Models\Usuario::class, 'usuario_id');
    }

    public function inmueble()
    {
        return $this->belongsTo(\App\Models\Inmueble::class, 'inmueble_id');
    }
```

Diagrama:



6.3.2. Servicios REST

La aplicación implementa una serie de servicios RESTful para gestionar los recursos de usuarios, reservas, inmuebles y operaciones relacionadas. Estos servicios siguen el estándar HTTP (GET, POST, PUT, DELETE) y están protegidos mediante middleware de autenticación y roles.

Endpoints de la API

Método	Ruta	Funcionalidad
GET	/perfil	Obtener información del perfil del usuario autenticado.
PUT	/perfil/{id}	Actualizar nombre o email del usuario.
GET	/perfil/password	Mostrar formulario para cambiar contraseña.
PUT	/perfil/password	Actualizar contraseña del usuario.
POST	/reservas	Crear una nueva reserva para un inmueble.
GET	/perfil/reservas	Mostrar todas las reservas del usuario.
DELETE	/reservas/{id}	Cancelar una reserva existente.

Método	Ruta	Funcionalidad
GET	/inmuebles	Listado filtrable de inmuebles.
POST	/scraping/todo	(Solo admin) Iniciar scraping y cargar nuevos datos.
GET	/usuarios	(Solo admin) Mostrar lista de usuarios registrados.
DELETE	/usuarios/{id}	(Solo admin) Eliminar un usuario.

Protección de Accesos:

- Los servicios están protegidos por autenticación (auth) y, en algunos casos, verificación (verified).
- Se emplea control de roles (admin/usuario) mediante el paquete spatie/laravel-permission.

Finalidad:

Estos endpoints permiten mantener una arquitectura desacoplada entre el front-end (Blade + Bootstrap + Vite) y el back-end (Laravel), facilitando posibles futuras extensiones a SPA o aplicaciones móviles sin modificar la lógica de negocio.

6.3.3. Paquetes adicionales

Durante el desarrollo del proyecto se han utilizado diversos paquetes adicionales para Laravel, con el objetivo de facilitar ciertas funcionalidades y mejorar la arquitectura de la aplicación. A continuación se detallan los más relevantes:

Paquete	Propósito / Funcionalidad
spatie/laravel-permission	Gestión avanzada de roles y permisos. Permite asignar roles como "admin" o "usuario" y proteger rutas según el rol. Facilita el control de acceso en la aplicación.
symfony/dom-crawler + guzzlehttp/guzzle	Utilizados para el scraping de páginas web. DomCrawler permite recorrer el DOM de una página HTML, y Guzzle se encarga de hacer las peticiones HTTP necesarias.
barryvdh/laravel-dompdf	Generación de archivos PDF desde vistas Blade. Se emplea para descargar fichas de reservas y listados de usuarios en formato PDF.

Paquete	Propósito / Funcionalidad
laravel/ui	Añade soporte para autenticación tradicional con Blade (login, registro, verificación).

6.4. Diseño front-end

El diseño del front-end se ha desarrollado utilizando principalmente el motor de plantillas **Blade** de Laravel, junto con **Bootstrap 5** para la maquetación y estilos visuales. También se ha empleado **SASS** para personalizar estilos y se ha integrado **Vite** como gestor de recursos modernos (CSS/JS) para una carga optimizada y más ágil en desarrollo.

Consumo de servicios web

Los servicios REST creados en el back-end se consumen principalmente mediante:

- **Formularios HTML clásicos** enviados con POST, PUT y DELETE para acciones como crear o eliminar reservas, modificar el perfil o registrar usuarios.
- Las rutas y controladores están protegidos por middleware que filtra por autenticación y rol, asegurando el acceso adecuado según el usuario.
- Los datos generados por el back-end se presentan en vistas dinámicas con Blade, sin necesidad de recarga de página gracias a la estructura modular y reutilizable de componentes visuales.

Funcionalidades de cliente implementadas

- **Filtros interactivos de búsqueda de inmuebles**: A través de formularios con selectores, radios y campos input, el usuario puede filtrar por ciudad, precio, tipo de inmueble, etc. El resultado es renderizado directamente en la vista.
- **Sistema de reservas con validaciones**: Se usan formularios dinámicos en modales para reservar inmuebles, con validación tanto en cliente (HTML5 y JavaScript) como en servidor.
- **Modales personalizadas para reservar**: Cada inmueble incluye un botón de "Reservar" que abre un modal preconfigurado con los datos del inmueble y

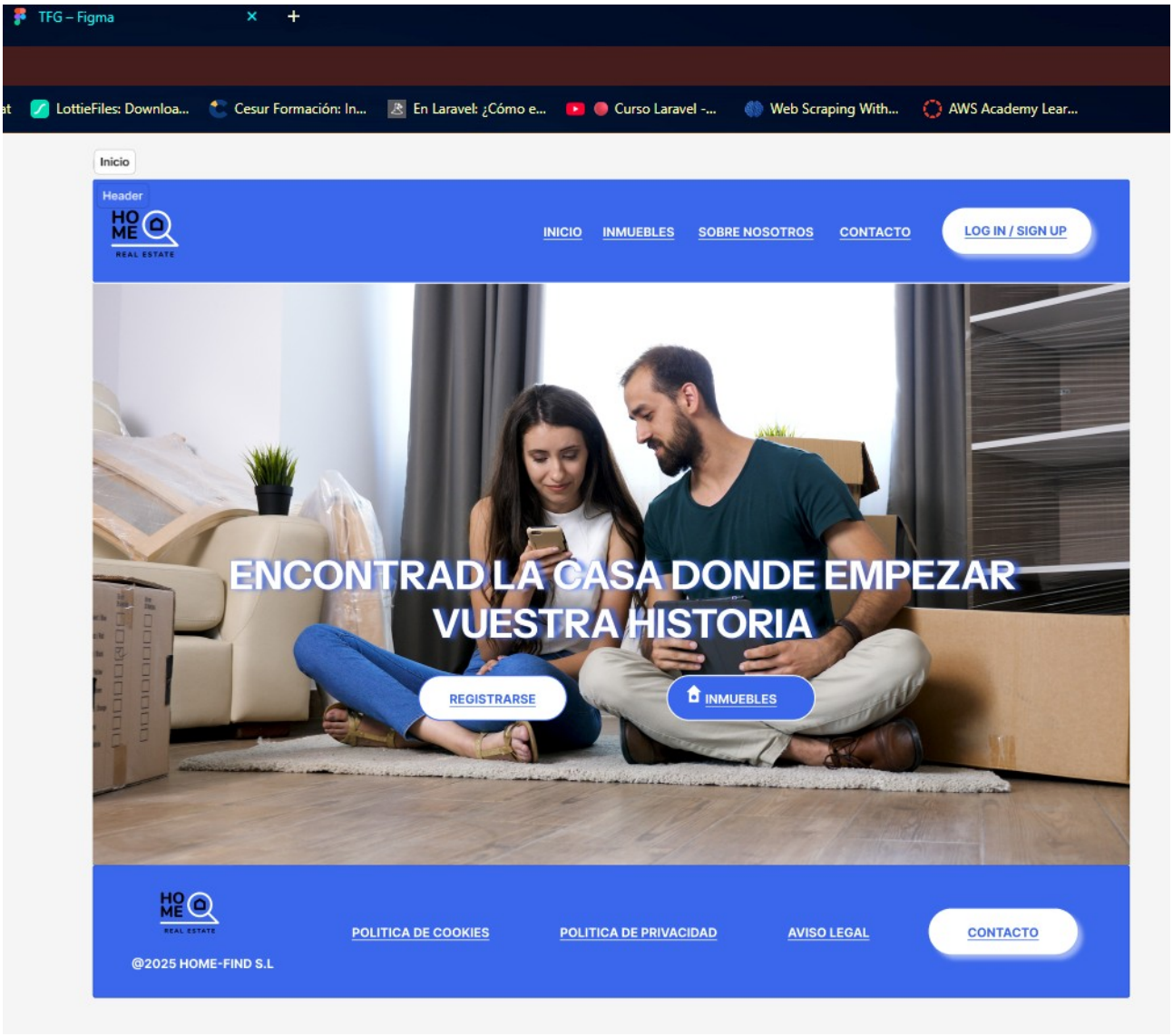
opciones de fecha y hora.

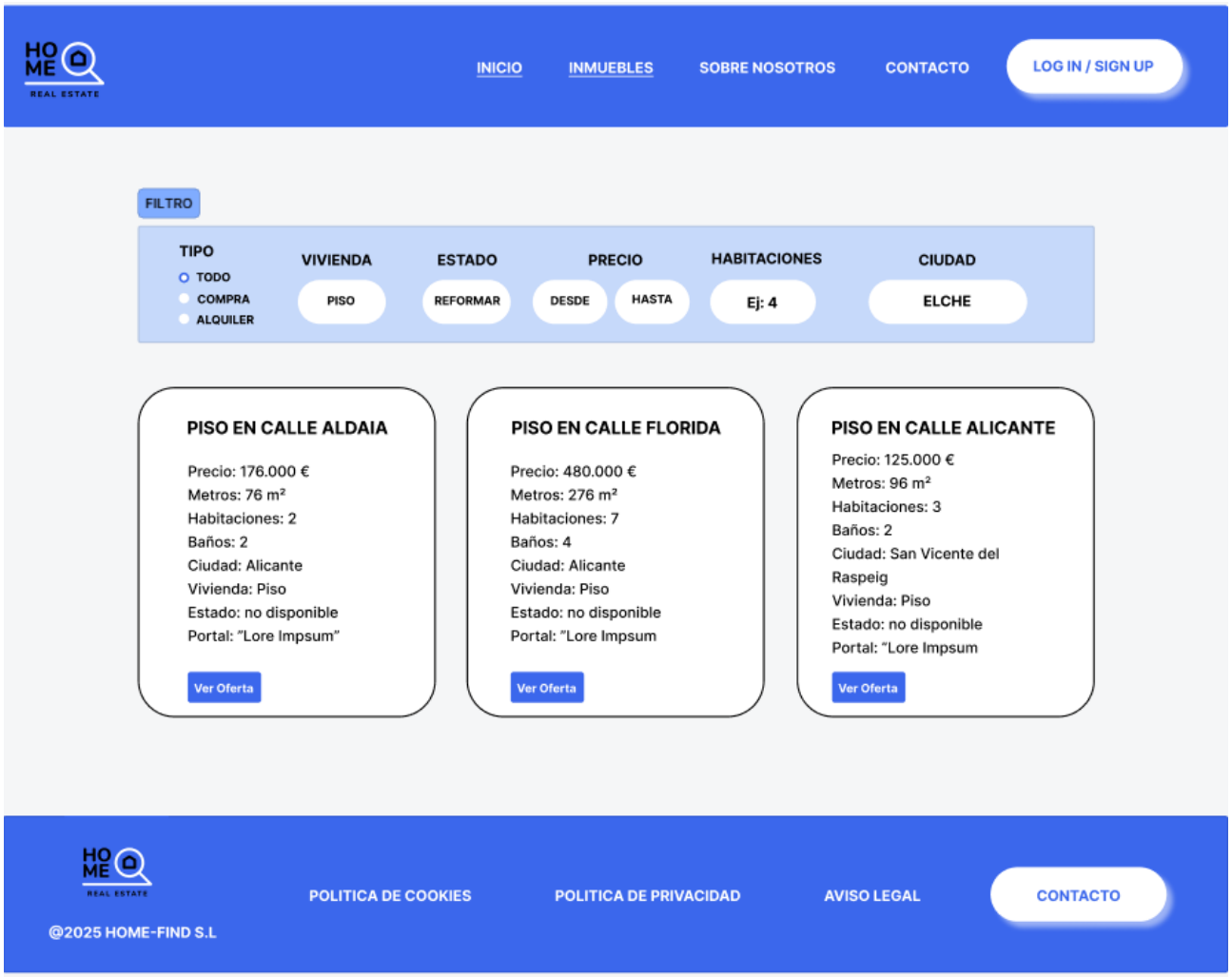
- **Alertas visuales**: Se muestran banners en pantalla para errores (como fechas ya reservadas), usando componentes de Bootstrap (alert) y estilos personalizados.
- **Responsive design**: Se ha garantizado la usabilidad en dispositivos móviles mediante clases de Bootstrap (d-none, d-md-block, offcanvas, etc.), con menús laterales adaptados a pantallas pequeñas.

- **Vue.js (opcional)**: Si bien el uso de Vue ha sido mínimo, se preparó la base del proyecto para una posible implementación futura de componentes dinámicos más complejos.

6.4.1. Mock-ups

El diseño conceptual se ha creado primero en **Figma** para una mejor concepción de que camino seguir. Aquí dos capturas de como se tenía pensado.


TFG – Figma
LottieFiles: Downloa...
Cesur Formación: In...
En Laravel: ¿Cómo e...
Curso Laravel -...
Web Scraping With...
AWS Academy Lear...
Inicio
Header
INICIO
INMUEBLES
SOBRE NOSOTROS
CONTACTO
LOG IN / SIGN UP
ENCONTRAD LA CASA DONDE EMPEZAR VUESTRA HISTORIA
REGISTRARSE
INMUEBLES
POLITICA DE COOKIES
POLITICA DE PRIVACIDAD
AVISO LEGAL
CONTACTO
@2025 HOME-FIND S.L



6.4.2. Guía de estilos

El diseño visual del proyecto busca ofrecer una experiencia clara, moderna y profesional, adaptada tanto a usuarios comunes como administradores. Para ello, se ha definido una guía de estilos coherente, basada en **Bootstrap 5** y personalizada mediante **SASS**.

Colores principales

Uso	Color	Código HEX
Color primario	Azul corporativo	#3B67EC
Fondo general	Gris claro	#f9f9f9
Texto principal	Gris oscuro / Negro	#333333

Tipografía

- **Fuente principal**: Poppins, con fallback a Arial, sans-serif.
- **Cuerpo de texto**: 14px a 16px.
- **Títulos**:
  - h1: 2rem – negrita
  - h2: 1.75rem – seminegrita
  - h3: 1.5rem
- **Botones y links**: 0.9rem – con fw-semibold para resaltar.

Iconografía

- Se utiliza **Font Awesome** para iconos en menús, botones y alertas.
- Ejemplos:
  - fa-user, fa-sign-out-alt, fa-calendar-alt, fa-key, fa-user-edit, fa-user-times, etc.

6.4.3. Paquetes adicionales

Durante el desarrollo del front-end del proyecto se han utilizado varios paquetes y librerías para mejorar la experiencia de usuario, facilitar el diseño visual y acelerar el desarrollo de funcionalidades. A continuación se describen los principales:

Bootstrap 5

- **Uso**: Framework CSS utilizado como base para el diseño visual del proyecto.
- **Funcionalidad**: Facilita la creación de componentes responsive como botones, formularios, tablas, modales y sistemas de columnas adaptables.
- **Integración**: A través de Vite y SASS, con personalizaciones de colores, tamaños y sombras.

Font Awesome

- **Uso**: Biblioteca de iconos vectoriales.
- **Funcionalidad**: Mejora la experiencia visual incluyendo iconos en botones, enlaces de navegación y formularios.
- **Ejemplos**: fa-user, fa-sign-out-alt, fa-calendar, fa-key.

Google Fonts (Poppins)

- **Uso**: Tipografía personalizada.
- **Funcionalidad**: Mejora la estética y la legibilidad del sitio, haciendo que el diseño se vea más profesional.
- **Integración**: Cargada en el layout general del proyecto (layouts/app.blade.php).

Vite

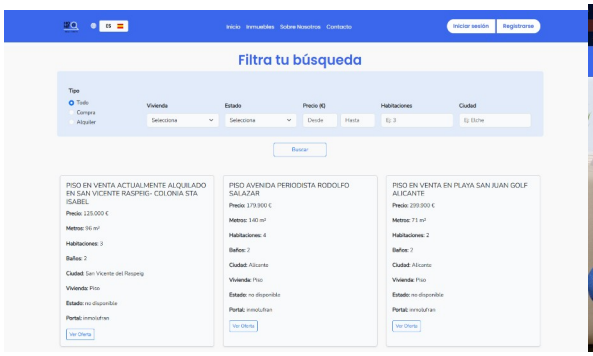
- **Uso**: Bundler moderno para recursos front-end.
- **Funcionalidad**: Procesa los archivos SASS/CSS y JS. Mejora la velocidad de desarrollo gracias a su hot-reload.
- **Ventaja**: Mucho más rápido que Mix, utilizado por versiones anteriores de Laravel.

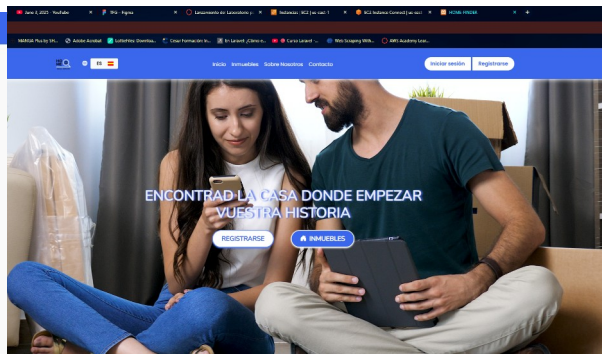
6.4.4. Capturas de la aplicación

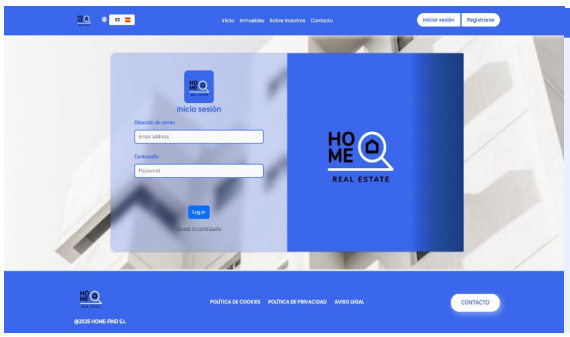
Se ha implementado un sistema de Hamburguesa para la accesibilidad en móvil. A demás, es responsive tanto en Tablet como en formato móvil. Comprobado en un Iphone 13 y Aipad Air.

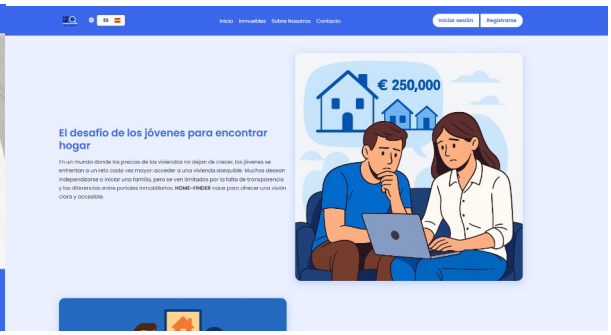
Las imágenes que muestro lo confirman y dan un pequeño ejemplo.

PC

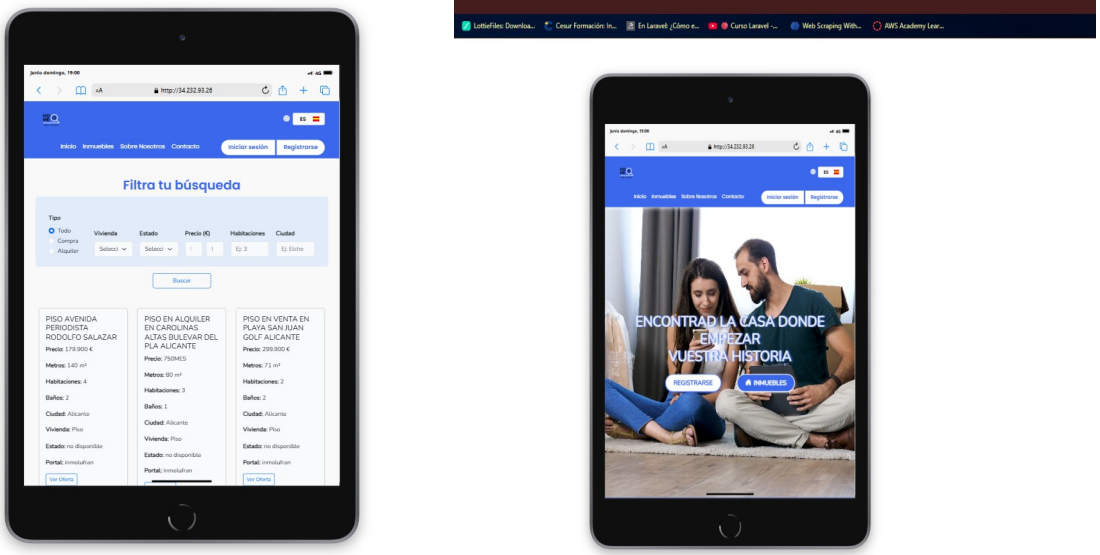

Filtra tu búsqueda


ENCONTRAD LA CASA DONDE EMPEZAR VUESTRA HISTORIA
REGISTRARSE
INMUEBLES

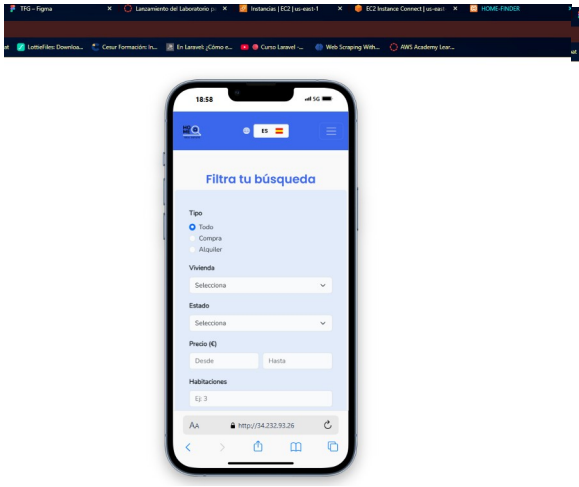

Inicia sesión
HOME
REAL ESTATE

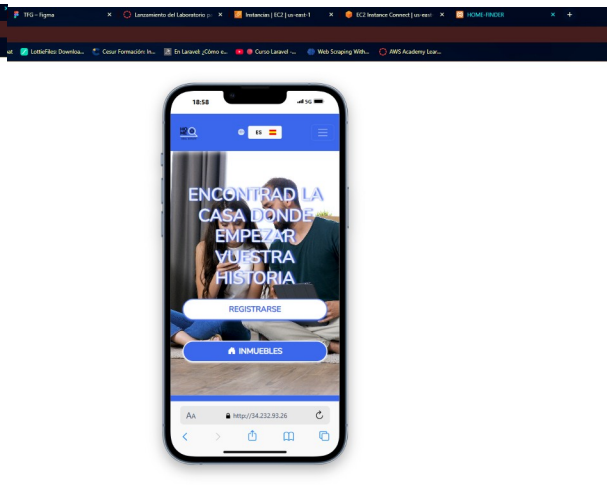

El desafío de los jóvenes para encontrar hogar
€ 250,000

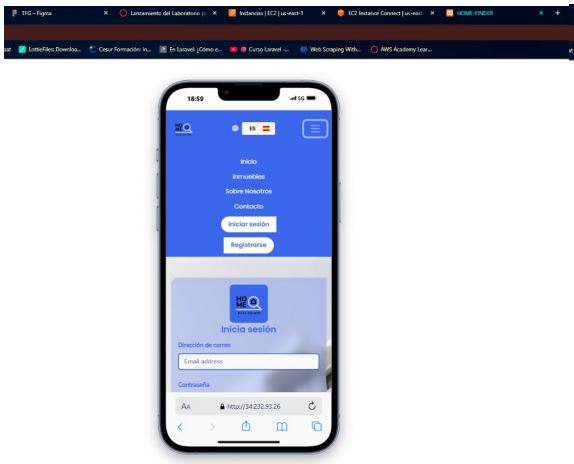
Tablet

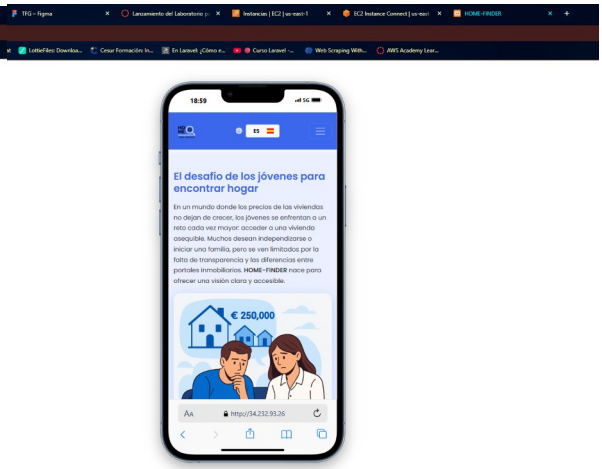

Filtra tu búsqueda
ENCONTRAD LA CASA DONDE EMPEZAR VUESTRA HISTORIA
Inicia sesión
El desafío de los jóvenes para encontrar hogar
Una conversación entre sueños e incertidumbres

Móvil









7. IMPLEMENTACIÓN

7.1. Servidor

Tras configurar el entorno de desarrollo local con Laravel, Composer y Node.js, se llevó a cabo el desarrollo del proyecto de forma estructurada siguiendo el modelo MVC (Modelo-Vista-Controlador). A continuación se detallan los pasos realizados, el orden seguido y los principales elementos desarrollados:

1. **Creación del proyecto y configuración base**
   - Instalación del proyecto Laravel con composer create-project.
   - Configuración del archivo .env (base de datos, URL, credenciales).
   - Instalación de dependencias front-end: npm install, configuración de vite.config.js y compilación inicial con **npm run dev.**

2. **Modelos y Migraciones**
   - **Carpeta:** app/Models/ y database/migrations/.
   - Se desarrollaron los modelos:
     - User (Laravel por defecto)
     - Inmueble
     - Reserva
   - Por cada modelo se generó su respectiva migración con los campos adecuados (string, date, integer, foreign keys, etc.) y se ejecutó php artisan migrate.
   - Relaciones añadidas (hasMany, belongsTo) entre modelos.
3. **Controladores**
   - **Carpeta:** app/Http/Controllers/
   - Se crearon los controladores principales:
     - InmuebleController
     - ReservaController
     - PDFController
   - Se definieron métodos index, create, store, destroy, y en el caso de reservas, se añadió lógica para validar horarios y evitar duplicados.
4. **Rutas**
   - **Archivo:** routes/web.php
   - Se definieron todas las rutas con sus respectivos middleware (auth, role:admin, etc.).
   - Uso de Route::resource() para controladores CRUD y rutas adicionales como /generate-pdf-reserva.
5. **Vistas (Blade)**
   - **Carpeta:** resources/views/
   - Vistas estructuradas por secciones:
     - inmuebles/index.blade.php (lista)
     - reservas/create.blade.php, admin/perfil/reservas.blade.php (panel de

usuario)
    - pdf/reservaPdf.blade.php (para generación de PDFs)
    - layouts/app.blade.php y layouts/admin.blade.php como base.
- Estilo visual implementado con Bootstrap y clases personalizadas.

6. **Semillas y Factories**
    - **Carpetas:** database/seeders/ y factories/
    - Uso de UsuarioSeeder para rellenar la base de datos de prueba.
    - Comando usado: php artisan migrate:fresh --seed.
7. **Scraping (Symfony DomCrawler + Guzzle)**
    - **Carpeta:** app/http/controller/ScrapingController.
    - Lógica de scraping implementada en servicios personalizados.
    - Se descartó Goutte y Dusk por rendimiento, y se usó Symfony por su estabilidad.
8. **Despliegue al servidor (AWS)**
    - Subida del proyecto con PuTTY y PSCP.
    - Instalación de dependencias con composer install, npm install y npm run build.
    - Configuración de permisos, entorno .env, y base de datos remota.
    - Alojamiento en /opt/lampp/htdocs/ con XAMPP y Apache configurado.

7.2. Cliente

En el lado del cliente, se ha utilizado **Vite** como herramienta de construcción y recarga en caliente junto a **Bootstrap 5** para la maquetación y estilo visual. Además, se ha integrado parcialmente **Vue.js** para ofrecer mayor interactividad en componentes específicos del sistema.

Tecnologías empleadas

- **Vite**: Cargador moderno para desarrollo front-end.
- **Bootstrap 5**: Framework CSS utilizado para el diseño responsive.
- **SASS**: Preprocesador para organizar estilos.
- **Vue.js (parcialmente)**: Se utilizó para crear algunos componentes dinámicos.

Elementos desarrollados

Componente / Vista	Archivo	Descripción
Filtros de búsqueda	resources/js/components/ FiltroInmuebles.vue	Componente Vue que gestiona dinámicamente los filtros de búsqueda por tipo, ciudad, precio, etc.
Sidebar responsivo	resources/js/layout/sidebar.js	Script de interactividad para la barra lateral y menús desplegables.
Modal de reserva	resources/views/inmuebles/ index.blade.php	Modal en Blade que permite seleccionar fecha/hora de reserva de forma visual.
Alerta personalizada	resources/js/components/ Alerta.vue (opcional)	Componente (o bloque Blade) que muestra errores o confirmaciones tipo banner modal.
Estilos globales	resources/sass/app.scss	Archivo SCSS principal con variables, colores, tamaños y estilos reutilizables.
Compilación final	public/build/assets/ (auto-generado por Vite)	Resultado del build para producción (npm run build).

Flujo cliente-servidor

- Formularios en Blade (.blade.php) consumen directamente las rutas definidas en Laravel.
- La lógica se realiza en el servidor, pero ciertas validaciones (como deshabilitar domingos o previsualizar errores) se implementaron con JavaScript/Vue para mejorar la UX.
- El uso de modales y alertas mejora la experiencia sin necesidad de recargar la página.

8. DESPLIEGUE

8.1. Modelo de despliegue utilizado

El modelo de despliegue utilizado para esta aplicación es **instalación directa sobre un servidor Linux (AWS EC2)**, sin el uso de contenedores (como Docker). Se optó por una configuración clásica de servidor web con PHP, ideal para entornos de producción sencillos y controlables.

Características del despliegue:

- **Servidor base:** AWS EC2 con sistema operativo Ubuntu Server.
- **Stack de servidor:**
  - Apache (servidor web)
  - MySQL (gestor de base de datos)
  - PHP 8.2 (entorno de ejecución para Laravel)
  - XAMPP para facilitar la gestión de Apache y MySQL.
- **Framework utilizado:** Laravel 10.
- **Despliegue del código:** Subida manual del proyecto vía SSH.
- **Compilación de assets:** Uso de Vite y npm run build para generar recursos productivos (manifest.json, JS/CSS optimizados).
- **Gestión de dependencias:**
  - PHP: mediante Composer (composer install)
  - JS/CSS: mediante NPM (npm install)
- **Base de datos:** MySQL, con migraciones automáticas ejecutadas vía php artisan migrate.
- **Permisos:** Ajuste manual de permisos de escritura para storage/ y bootstrap/cache/.
- **Entorno configurado en .env:** con datos sensibles del entorno de producción (APP_ENV=production, APP_DEBUG=false, etc.).
- **Cachés optimizados:** Configuración y rutas cacheadas con php artisan optimize.

8.2. Datos iniciales y configuración

Para facilitar el despliegue y uso inicial de la aplicación, se han precargado una serie de datos esenciales mediante **seeders** y **migrations** de Laravel. Estos datos permiten arrancar el sistema con una configuración mínima operativa, asegurando que haya acceso administrativo y contenidos de prueba.

Usuario administrador

Al ejecutar php artisan migrate:fresh --seed, se genera automáticamente varios usuarios administradores y 1 normal con las siguientes credenciales:

- **Nombre**: Juanca
- **Email**: ejemplo1@gmail.com
- **Contraseña**: null *(por defecto, se recomienda cambiarla tras el primer acceso)*
- **Rol**: admin (con acceso completo al sistema, incluidos usuarios y reservas)

Datos maestros precargados

- **Roles**: Se crean los roles admin y usuario, mediante el seeder de Spatie\Permission.
- **Usuarios de prueba**: Incluye algunos usuarios estándar para simular uso real del sistema.
- **Inmuebles de prueba**: Se realiza el scrapeo en una de las inmobiliarias que se han testeado para sacar los datos reales.

Configuración adicional

- **Permisos asignados por rol** con Spatie: cada rol tiene sus permisos predefinidos en el seeder correspondiente.
- **Configuración de .env**: se personaliza para el entorno de producción o local, incluyendo las claves de base de datos, URL del frontend, etc.
- **Configuración de Vite y Laravel Mix**: para cargar correctamente los recursos CSS/JS desde npm run dev o npm run build.

Comandos para inicialización rápida en local:

```
php artisan migrate:fresh --seed   # Migrar y poblar con datos iniciales
php artisan serve
npm install && npm run build  && npm run dev   # Compilar frontend con Vite
```

8.3. Pasos para el despliegue

Una vez la aplicación está hecha, se ha de subir al servidor que hemos escogido, en este caso el EC2 de AWS. Lo primero será tener la instancia en EC2, una vez creada, la lanzamos. Podemos usar Putty para pasar el programa, pero primero actualizamos con sudo apt update por si necesitara algún componente actualizarse.
Luego instalamos Xampp con:

```
wget https://www.apachefriends.org/xampp-files/8.2.12/xampp-linux-x64-8.2.12-0-installer.run
sudo chmod +x xampp-linux-x64-8.2.12-0-installer.run
sudo ./xampp-linux-x64-8.2.12-0-installer.run
```

y sudo ln -s /opt/lampp/bin/php /usr/bin/php para que funcione el php en el terminal.
Se contesta a todo que si,
Sudo apt install net-tools para tener todas las herramientas en caso de necesitarlas y ahora sí.
Sudo /opt/lampp/lampp start. Tendremos Xampp instalado.
Podemos dirigirnos ahora a nuestra ip, ahí al phpmyadmin para exportar nuestra base de datos y gestionarla a futuro.
Instalamos Composer(para Laravel)
bash

```
curl -sS https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer
sudo chmod +x /usr/local/bin/composer
```

Verifica la instalación:

composer –version

Instalar Node.js y npm

curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

Verifica la instalación:
node --version
npm --version

Configurar permisos para tu directorio de proyecto

sudo chown -R ubuntu:www-data /opt/lampp/htdocs
sudo chmod -R 777 /opt/lampp/htdocs
sudo chown -R www-data:www-data /opt/lampp/htdocs

Ahora subiremos el proyecto. Si estas en windows como es mi caso, acuerdate de instalar putty o lo que es lo mismo pscp. Lo ejecutas y acuérdate de tener el ppk. Abrimos Putty, en HostName ponemos ubutnu@(tu IP del servidor) y en la sección "Connection > SSH > Auth > Credentials", busca y selecciona tu archivo .ppk
Le damos a Open.

Luego vamos en el cmd de nuestro equipo donde tenemos el pscp.exe y en el cmd escribimos pscp -r FacturacionZataca usuario@ip-servidor:/opt/lampp/htdocs/

Tardará pero lo subirá. También puedes usar Git Clone, pero te tocaría reinstalar ciertos componentes a demás de tener que crear el .env. Una vez subido nos iríamos al directorio. En este caso cd /opt/lampp/htdocs/FacturacionZataca y estando dentro, ejecutamos composer install
npm install

npm run build.

Acuérdate de darle permisos a la carpeta de nuestro repositorio que acabamos de subir.

¡¡¡Importante!!! Se ha explicado de esta manera, ya que es la mas sencilla para subirlo, debido a que tiene instalado Symfony como Spatie, Vite, etc.

8.4. Proveedores y servicios utilizados

Se ha utilizado hosting, o servicio EC2 como el de Amazon, a demás de la web de Figma y webs de inmobiliarias para las pruebas para realizar el scrapeo.

9. HERRAMIENTAS UTILIZADAS

Hemos usado tanto Figma para el desarrollo inicial del arte conceptual del proyecto como idealizar un esquema y su funcionamiento.

Se ha usado VisualStudio Code para programar toda la aplicación, como Google Chrome para los asuntos de internet(Búsqueda de información, acceder al servidor, etc..)

Se ha utilizado también ChatGPT para la creación de ciertos elementos, tales como las imágenes, logotipo,ect.. A demás de ayuda para encontrar la mejor manera de usar el Scraping debido a las dificultades que las webs de portales grandes han llegado a poner. De este modo me pudo recomendar la que aparece dentro del programa, ya que su seguridad para evitar el scrapeo es casi nula.

XAMPP para poder usar MySQL y Apache.

LibreOffice para la creación de la documentación y su conversión a pdf.

PuTTy de Windows para subir los archivos.

Laravel, Boostrap 5, Spatie, HTML5, CSS3, SASS, Scraping (Symfony DomCrawler + Guzzle), Composser, Node.js , Vite, AWS(servidor usado), Blade(plantillas de Laravel), MySQL para el gestor de la base de datos.