Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Aplicación Web Casa Damajuana

Alumno: Miguel Esteban Navajo

Tutor/a/es: Fernando Díaz Gómez y José Ignacio Farrán Martín

Aplicación Web Casa Damajuana

Miguel Esteban Navajo

Índice general

Índice de figuras

Índice de cuadros

"La única forma de hacer un gran trabajo es amar lo que haces."

Steve Jobs

Agradecimientos

Al llegar al final de esta etapa universitaria, no quiero dejar pasar la oportunidad de expresar mi más sincero agradecimiento a todas las personas que, de una u otra forma, han contribuido a que este camino haya sido posible y enriquecedor.

En primer lugar, a mi familia, por su apoyo incondicional, por creer en mí incluso en los momentos más difíciles, y por ser siempre un pilar fundamental en mi vida. Gracias por su paciencia, su comprensión y por estar a mi lado durante todo este proceso. Sin ellos, probablemente no habría tenido las mismas oportunidades, ni las fuerzas suficientes ante ciertas adversidades. Este trabajo no es solo el fruto de mi esfuerzo, sino el reflejo de lo que mi familia me ha inculcado: constancia, trabajo duro y amor por lo que hago.

Agradezco también a la Universidad y al profesorado que me ha acompañado a lo largo de estos años. Gracias por compartir vuestros conocimientos, por vuestra vocación y por despertar en mí el interés por seguir aprendiendo y creciendo profesionalmente.

De manera especial, quiero agradecer a mis tutores, Fernando Díaz Gómez y José Ignacio Farrán Martín, por su guía, sus consejos y por ayudarme a convertir una idea en una realidad concreta. Su orientación ha sido clave para llegar a buen puerto con este trabajo.

Finalmente, quiero mencionar a mis compañeros y compañeras de clase, por los buenos momentos compartidos, las horas de estudio, los retos superados juntos y por todo lo que hemos aprendido mutuamente a lo largo del camino.

Este Trabajo de Fin de Grado no solo representa el cierre de una etapa académica, sino también el inicio de una nueva fase personal y profesional, para la cual me siento totalmente preparado.

Resumen

En un contexto donde la tecnología se integra cada vez más en nuestra vida cotidiana, surge la necesidad de elaborar y aplicar soluciones digitales que optimicen la organización y seguridad en entornos rurales.

Este Trabajo de Fin de Grado tiene como objetivo el desarrollo de una aplicación *web* destinada a la gestión de una casa rural familiar, ofreciendo funcionalidades centradas en el control de usuarios y reservas, la realización de pagos seguros, la integración de servicios externos y la protección del acceso a la información. A través de tecnologías modernas como *Angular* y *Spring Boot*, se ha creado una plataforma segura, intuitiva y escalable que facilita la interacción entre huéspedes y propietarios. Durante el desarrollo se han aplicado principios de diseño centrado en el usuario y se han realizado pruebas exhaustivas para garantizar la fiabilidad del sistema.

Esta propuesta demuestra cómo la tecnología puede ser una excelente herramienta a emplear en pequeños negocios familiares, promoviendo una gestión más eficiente y dotándoles de mayor visibilidad.

Palabras claves: tecnología, aplicación *web*, gestión, casa rural familiar, usuarios, reservas, pagos seguros, servicios externos, *Angular*, *Spring Boot*, intuitiva, escalable, fiabilidad, negocios familiares, visibilidad.

Abstract

In a context where technology is increasingly integrated into our daily lives, there arises a need to develop and implement digital solutions that optimize organization and security in rural environments.

This Final Degree Project aims to develop a *web* application designed for the management of a family-owned rural house, offering functionalities focused on user and booking control, secure payments, integration with external services, and protection of information access. Through modern technologies such as *Angular* and *Spring Boot*, a secure, intuitive, and scalable platform has been created to facilitate interaction between guests and property owners. During development, user-centered design principles were applied, and thorough testing was carried out to ensure the reliability of the system.

This proposal demonstrates how technology can be an excellent tool to use in small family businesses, promoting more efficient management and providing greater visibility.

Keywords: technology, *web* application, management, family-owned rural house, users, bookings, secure payments, external services, *Angular*, *Spring Boot*, intuitive, scalable, reliability, family businesses, visibility.

Parte I

Descripción del Proyecto

Capítulo 1

Introducción

En este primer capítulo de la memoria se presenta una visión global de mi Trabajo Fin de Grado, proporcionando el contexto necesario para comprender su motivación, objetivos, condicionantes y alcance. Esta sección sirve como punto de partida para introducir al lector en la naturaleza del proyecto, su justificación y su estructuración.

En primer lugar, se realiza una puesta en contexto donde se reflexiona sobre el papel que juega la tecnología en la vida cotidiana, así como su capacidad para transformar entornos tradicionalmente desconectados del ámbito digital. A partir de esta reflexión general, se deriva la motivación particular del proyecto. De esta manera, se logrará comprender el propósito del trabajo y demostrar su originalidad frente a propuestas similares.

Posteriormente, se detallan los objetivos que guían el desarrollo de la solución propuesta, abordando tanto los fines prácticos como los personales y académicos. Se exponen también los condicionantes del proyecto, incluyendo las reglas de negocio, restricciones técnicas y limitaciones derivadas del entorno de aplicación.

A continuación, se delimita el alcance del proyecto, acotando las funcionalidades desarrolladas y definiendo los límites del sistema, lo cual resulta clave para evaluar la viabilidad y éxito del mismo en secciones posteriores.

Finalmente, se describe la estructura de la memoria, facilitando al lector una guía clara sobre los contenidos y organización del documento.

1.1. Puesta en contexto

Vivimos en una era en la que la tecnología ha adquirido un papel esencial en prácticamente todos los aspectos de la vida cotidiana. Desde la manera en que trabajamos, consumimos contenido o nos comunicamos, hasta cómo planificamos viajes o realizamos compras, la transformación digital ha reconfigurado nuestras rutinas y expectativas. Esta revolución tecnológica ha propiciado una nueva cultura digital que exige a empresas, instituciones y particulares adaptarse a un entorno cada vez más conectado y exigente en términos de accesibilidad, inmediatez y personalización.

En este proceso de modernización, incluso los entornos más tradicionalmente desconectados del ámbito tecnológico, como el medio rural, comienzan a experimentar una

transición hacia la digitalización. El desarrollo de infraestructuras, la mejora de la conectividad y la creciente concienciación sobre la necesidad de presencia en *Internet* han impulsado a numerosos pequeños negocios del entorno rural a explorar soluciones digitales para mejorar su competitividad y proyección.

El sector del turismo rural no es ajeno a esta transformación. La forma en que los viajeros descubren, comparan y reservan alojamientos ha cambiado radicalmente en la última década, con un claro dominio de las plataformas *online* y una fuerte dependencia de la visibilidad digital. En este contexto, la creación de herramientas tecnológicas personalizadas, como páginas *web* específicas para alojamientos rurales, se presenta no solo como una oportunidad, sino como una necesidad para garantizar la sostenibilidad y el crecimiento de este tipo de iniciativas.

Es en este escenario donde se enmarca el presente Trabajo Fin de Grado. La digitalización de un alojamiento turístico familiar, situado en un pequeño municipio rural, sirve como ejemplo concreto de cómo la tecnología puede contribuir activamente a preservar, promover y hacer viable un proyecto ligado al territorio, la tradición y los vínculos emocionales, pero adaptado a las exigencias del mundo actual.

1.2. Motivación

Este Trabajo Fin de Grado surge de la motivación personal y familiar de colaborar en el desarrollo y consolidación de un negocio rural. La iniciativa parte de la necesidad de dotar de presencia digital a una casa rural ubicada en Villoslada de la Trinidad, un pequeño municipio de Segovia (Castilla y León). Dicha vivienda de uso turístico, gestionada por mi familia y promovida inicialmente por mi madre, cuenta con un fuerte componente emocional, al estar construida en el pueblo natal de mi difunto padre. Este entorno tiene un gran valor sentimental, al tratarse del lugar donde crecí y donde residen familiares y amigos que conforman la comunidad del pueblo.

La casa rural, hasta la fecha, carecía de una página *web* propia que ofreciera una visión completa de sus servicios e instalaciones. Su presencia en *Internet* se limitaba a plataformas de terceros como *Airbnb* [23] o *Booking* [26], que imponen comisiones y limitaciones en la personalización del contenido. A su vez, algunos huéspedes manifestaban su interés en consultar una *web* oficial con información directa, fiable y sin intermediarios. En este contexto, se detectó la oportunidad de crear una solución digital personalizada que centralizara la información esencial sobre la casa y su entorno, facilitando tanto la reserva directa como la interacción con los potenciales clientes.

El objetivo general de este TFG es el desarrollo de una aplicación *web* moderna, funcional y escalable, que permita a los usuarios consultar detalles sobre las instalaciones, realizar reservas, acceder a reseñas reales mediante integración con *APIs* de terceros (como *Google* [18]), así como explorar actividades y lugares de interés en la zona. Esta plataforma servirá como punto de contacto directo entre los huéspedes y los responsables de la casa rural, evitando intermediarios y mejorando la experiencia del usuario.

Una de las principales aportaciones de este trabajo radica en el uso de tecnologías no

abordadas en profundidad durante la carrera, que junto al objetivo personal de aportar en este proyecto familiar, dan lugar al caldo de cultivo perfecto para el desarrollo de un proyecto atractivo, innovador y original. En él se emplean tecnologías como el *framework* de *frontend* ***Angular*** [24] y el *backend* desarrollado con ***Spring Boot*** [1], conectados a una base de datos ***MySQL*** (realmente *MariaDB*). Estas herramientas, empleadas habitualmente en entornos profesionales, fueron seleccionadas con el objetivo de ampliar mis competencias técnicas y acercar el proyecto a un entorno de producción real. La decisión se fundamenta en la experiencia adquirida durante mis prácticas en empresa, donde observé su uso frecuente para el desarrollo de aplicaciones *web* robustas.

El entorno de aplicación es, por tanto, una vivienda rural con aproximadamente 16 estancias entre habitaciones, baños y zonas comunes, equipada con servicios como piscina, zona de barbacoa, aparcamiento y conexión *Wi-Fi*. Aunque no se ofrecen servicios de restauración o lavandería, se proporciona a los huéspedes una guía local con recomendaciones de establecimientos. La gestión técnica de la página *web*, si se llevara a la práctica, correría a mi cargo, mientras que mi madre, como administradora del negocio, se encargaría de la atención y actualización de contenidos cuando sea necesario mediante el panel de administrador habilitado.

Este trabajo no se limita al plano académico, sino que se plantea como un prototipo funcional susceptible de ser puesto en producción a corto plazo, adaptado a las necesidades reales identificadas en colaboración con mi familia. A lo largo de esta memoria se describirán los objetivos específicos, la metodología empleada, las herramientas utilizadas, la planificación temporal y económica del desarrollo, así como un análisis crítico de los resultados obtenidos y posibles líneas futuras de mejora.

1.3. Objetivos

Como se menciona anteriormente, el proyecto se centra en elaborar una aplicación *web* que permita gestionar y promocionar la casa rural. Por un lado, brinda a los clientes una plataforma en la que pueda informarse acerca de los servicios ofrecidos, realizar sus reservas de la vivienda e incluso conocer ciertas actividades de los alrededores. Por otro, permite al administrador del negocio una gestión sencilla y centralizada de todo lo que incumbe a la casa.

En la tabla 1.1 se presentan los diversos objetivos que permiten construir el camino para alcanzar la meta del proyecto.

1.4. Condicionantes del proyecto

Antes de proceder al diseño y desarrollo técnico de la solución, es fundamental establecer el conjunto de elementos que condicionan y delimitan su alcance y funcionamiento. Estos condicionantes permiten comprender el entorno en el que se desarrolla el sistema, las reglas que debe respetar, y las suposiciones o dependencias que se dan por sentadas durante su implementación.

Objetivo 1	Desarrollar una página *web* funcional y escalable que permita promocionar y gestionar de forma integral una casa rural.
Obj 1.1	Permitir la gestión simple y eficiente de las reservas de la casa rural.
Obj 1.2	Mejorar la visibilidad del alojamiento mediante una plataforma propia que evite las comisiones de terceros.
Obj 1.3	Ofrecer información útil y detallada sobre la casa, sus estancias, servicios disponibles y entorno turístico.
Obj 1.4	Integrar *APIs* externas para facilitar la gestión de los pagos (*PayPal*) y la obtención de reseñas y mapas (*Google*).
Objetivo 2	Diseñar una interfaz intuitiva orientada a usuarios sin conocimientos técnicos.
Obj 2.1	Facilitar al administrador el control de la casa mediante un panel propio.
Obj 2.2	Garantizar una experiencia accesible y fluida para cualquier perfil de huésped.
Objetivo 3	Emplear tecnologías modernas para elaborar la solución y consolidar competencias profesionales.
Obj 3.1	Utilizar *Angular* como *framework* de *frontend*, *Spring Boot* como *framework* de *backend* y *MySQL* (*MariaDB*) como sistema de gestión de bases de datos relacional, para ir más allá de lo aprendido en el grado.
Objetivo 4	Construir un prototipo funcional realista que pueda ser usado y gestionado por la familia.

Cuadro 1.1: Objetivos del proyecto

En este apartado se recogen tanto las reglas y requisitos de negocio como las restricciones técnicas, suposiciones realizadas y dependencias externas, que afectan directa o indirectamente al desarrollo del proyecto. Su correcta identificación y documentación facilita la toma de decisiones, reduce riesgos y permite que otras personas comprendan mejor las características del sistema.

1.4.1. Requisitos de negocio

Paso a presentar los requisitos de negocio identificados, que representan aquello que la casa rural necesita lograr para alcanzar los objetivos definidos, sin especificar cómo se logrará.

Por tanto, los requisitos de negocio definidos irán alineados con la necesidad de existencia del proyecto, desgranándose a nivel técnico en la sección de requisitos funcionales. Se establecen en la tabla 1.2.

1.4.2. Reglas de negocio

A continuación, se exponen las reglas de negocio como condiciones, normas o políticas a seguir por parte del sistema, en la tabla 1.3.

RN 1	Permitir a los clientes reservar la casa rural sin intervención humana directa.
RN 2	Reducir la dependencia de plataformas de terceros (como *Airbnb*, *Booking*, etc.) para mejorar la rentabilidad.
RN 3	Facilitar una gestión ágil y autónoma por parte de la administradora del negocio (sin conocimientos técnicos avanzados).
RN 4	Mostrar información completa y fiable del alojamiento y su entorno sin necesidad de contacto previo.
RN 5	Habilitar la visualización y recolección de reseñas y mapas desde fuentes externas (*Google*).
RN 6	Ofrecer una plataforma segura para el pago *online* mediante sistemas ampliamente utilizados como *PayPal*.
RN 7	Permitir a los huéspedes registrados gestionar sus reservas de forma autónoma, consultar el historial y cancelarlas.

Cuadro 1.2: Requisitos de Negocio del proyecto

RgN 1	Las reservas deben realizarse con al menos 1 día de antelación.
RgN 2	No se permiten reservas en fechas pasadas.
RgN 3	Las reservas deben tener fechas de inicio y fin distintas y coherentes.
RgN 4	La casa solo se puede reservar en su totalidad (no por habitaciones).
RgN 5	La capacidad máxima de la casa es de 18 personas (flexible si hay niños).
RgN 6	No se permite fumar en la vivienda ni traer mascotas.
RgN 7	Deben respetarse normas internas como los horarios de descanso.
RgN 8	La piscina solo está disponible del 20 de junio al 20 de septiembre.
RgN 9	Existen tarifas diferenciadas según temporada (alta / baja).
RgN 10	Se aplican descuentos por número de noches reservadas.
RgN 11	Cancelaciones a menos de 7 días vista implican una penalización de 100€.
RgN 12	Cancelaciones con más de 7 días de antelación se reembolsan al 100 %.
RgN 13	Solo los usuarios registrados pueden cancelar reservas desde la aplicación, mientras que los demás deben de contactar con la dueña.

Cuadro 1.3: Reglas de Negocio del proyecto

1.4.3. Restricciones

Paso a enumerar las diversas restricciones que, actuando como límites de diseño, técnicos o económicos, condicionan de alguna manera el desarrollo de la solución propuesta.

Estos factores vendrán impuestos por el entorno, los clientes, los recursos disponibles

de todo tipo e incluso por el propio alcance del Trabajo Fin de Grado. Se presentan en la tabla 1.4.

Res 1	Uso de tecnologías modernas (*Angular*, *Spring Boot*, *MySQL* (*MariaDB*)) y usadas en el mercado.
Res 2	Buscar un presupuesto mínimo, evitando el uso de *software* o servicios con licencias de coste elevado, sin poner en riesgo la calidad.
Res 3	La mayoría de funcionalidades deben estar disponibles sin necesidad de registro.
Res 4	El sistema está diseñado para gestionar exclusivamente una única casa rural.
Res 5	Recursos limitados: el proyecto ha sido desarrollado por una sola persona sin equipo de soporte.
Res 6	Tiempo limitado para el desarrollo completo, condicionado por el calendario académico del TFG y el resto de asignaturas.

Cuadro 1.4: Restricciones del proyecto

1.4.4. Suposiciones

También, procedo a citar diversas condiciones que se dan por ciertas durante el desarrollo del sistema. Cabe destacar que puede que no se logre garantizar su cumplimiento en todo momento, al encontrarse fuera del control total de la persona responsable. Se citan las suposiciones del proyecto en la tabla 1.5.

Sup 1	Las *APIs* externas (*Google JavaScript Maps*, *Google Places*, *PayPal*, *reCAPTCHA Enterprise*, etc.) funcionan correctamente y están disponibles.
Sup 2	Los usuarios y el servidor disponen de conexión a *Internet* en todo momento.
Sup 3	El entorno de despliegue de la aplicación se mantiene estable y accesible.
Sup 4	Los usuarios acceden desde navegadores actualizados en dispositivos compatibles.
Sup 5	La aplicación solo se empleará para la gestión de una única casa rural.
Sup 6	Los clientes leen y respetan las normas y condiciones publicadas en la *web*.

Cuadro 1.5: Suposiciones del proyecto

1.4.5. Dependencias

El proyecto cuenta con una serie de dependencias externas, a nivel técnico y funcional, que son necesarias para su correcto funcionamiento.

Encontramos entonces ciertos recursos *software*, bibliotecas y servicios que son requeridos para la elaboración de la solución, y en los que se hará énfasis en secciones posteriores. Las dependencias del proyecto se exponen en la tabla 1.6.

Dep 1	*Google JavaScript Maps API* y *Google Places API* para mostrar mapas e integrar reseñas.
Dep 2	*API* de *PayPal* para el procesamiento de pagos y devoluciones *online*.
Dep 3	*Gmail* y *JavaMail* para el envío de correos electrónicos desde formularios de la *web*.
Dep 4	Diversas dependencias de *Angular*: *Bootstrap*, *NG Bootstrap*, *Angular Material*, *Forms*, *HttpClient*, etc.
Dep 5	Varias dependencias de *Spring Boot*: *Spring Web*, *Spring Data JPA*, *JavaMailSender*, dependencias de seguridad y *PayPal*, etc.
Dep 6	*MariaDB* (derivado de *MySQL*) como sistema de gestión de base de datos.
Dep 7	Plataforma de despliegue *online* (en un futuro) para publicar la aplicación.

Cuadro 1.6: Dependencias del proyecto

1.5. Alcance

Con la intención de delimitar las funcionalidades incluidas en el proyecto, procedo a resumir el alcance de la aplicación.

La aplicación *web* Casa Damajuana WebApp tiene la misión de mostrar cierta información acerca de las instalaciones y de facilitar la gestión y realización de reservas. La intención del proyecto es obtener una aplicación *web* completa, funcional y escalable, atendiendo a las limitaciones temporales impuestas por la propia naturaleza del Trabajo Fin de Grado.

Por tanto, podríamos definir el alcance exponiendo de forma general las funcionalidades contempladas, que paso a enumerar:

- Visualización de la información de estancias, servicios y actividades de la zona, permitiendo a los huéspedes contar con un punto de referencia oficial para ver la casa rural y sus características.

- Funcionalidades de uso no restringido, como la realización de reservas o la consulta de información acerca de la casa. Esto permite que usuarios no registrados, por el

motivo que sea, se animen a consultar información o planear su estancia en nuestro alojamiento.

- Sistemas de registro e inicio de sesión, que ofrezcan ciertos privilegios y facilidades a aquellos usuarios que decidan crear una cuenta.

- Sistema de reservas intuitivo, seguro y automatizado, que permita registrar reservas a todo tipo de usuarios. De esta manera, se podrán gestionar las compras sin necesidad de plataformas de terceros, o al menos reduciendo su uso.

- Integración con *APIs* de uso global, como *PayPal* y *Google*, que permitan aumentar la calidad de los servicios ofrecidos, empleando herramientas conocidas y confiables para los usuarios. Se logrará ofrecer reseñas, mapas y pasarelas de pago sin necesidad de gestionar localmente tanta información.

- Inclusión de componentes amigables, que aumenten la satisfacción de los usuarios al interactuar con la plataforma. Se cuidará la estética y usabilidad de los mismos.

- Puesta a disposición de los usuarios registrados (usuarios comunes y administradores) de paneles especiales, que permitan un control simple de las reservas y/o usuarios. Se velará por lograr que los paneles sean sencillos de usar para cualquier perfil de usuario, ofreciendo filtros interactivos, funcionalidades de extracción de datos, etc.

- Mantenimiento de un diseño adaptable o *responsive*, que permita el uso de la aplicación en dispositivos de distintos tamaños. Se cuidará la reorganización de contenidos, la adaptación de tablas y el redimensionamiento de imágenes, entre otros. De esta manera, se tratará de fomentar la accesibilidad de la aplicación, de forma que sea apta para usuarios de todo tipo.

- Garantía de seguridad de procedimientos, en especial de los pagos de las reservas. Se validarán los pagos realizados con *PayPal*, evitando reservas no autorizadas, cobros accidentales e intentos de accesos no autorizados mediante aplicaciones que permitan la realización de peticiones.

- Medidas de seguridad variadas para garantizar el cumplimiento de los estándares mínimos. Se cifrarán datos sensibles, se trabajará con procedimientos seguros y se protegerán las claves secretas de desarrollo de la aplicación.

Por otro lado, algunas funcionalidades u opciones que no se contemplan o se incluyen parcialmente, quedando pendientes para mejoras futuras, son las citadas a continuación:

- Internacionalización de la aplicación. Se han realizado las configuraciones base, pero no una traducción global de la página. Este tema se ataja en profundidad en la sección 6.1.3.

- No se gestiona más de una casa o negocio, pues la aplicación *web* está dedicada única y exclusivamente al negocio familiar mencionado en las secciones introductorias. Por tanto, no es un producto pensado para escalar y derivar en la gestión de otros negocios hosteleros.

- No se ofrece un sistema de registro personalizado para administradores. Al tratarse de un pequeño negocio familiar, el número de usuarios con rol de `administrador` sería muy reducido. Por tanto, he considerado más oportuno utilizar el registro general, y delegar la función de asignar el rol de usuario administrador en el administrador del sistema.

- No se incluye en el proyecto el desarrollo de una aplicación móvil nativa, ofreciendo en su defecto un diseño adaptable, que permita un uso correcto de la aplicación en dispositivos menos voluminosos. No se descarta la posibilidad de desplegar una aplicación móvil en una fase futura del proyecto.

- No se realiza un análisis *SEO* (*Search Engine Optimization*) como tal, entendiendo esto último como un proceso de evaluación y optimización del sitio *web* para mejorar su posicionamiento en los resultados de búsqueda de motores de búsqueda. Se trata en todo momento de evitar la repetición de código, elaborando y usando componentes reutilizables.

- Tampoco se lleva a cabo un estudio de *marketing* profesional, usando herramientas y estrategias como los famosos *CRMs* (*Customer Relationship Management*) u otros mecanismos de segmentación de clientes y gestión de campañas.

- No se incluyen opciones que resulten demasiado redundantes y repetitivas a nivel funcional, como la creación o modificación de espacios y actividades. Esto se debe a que la inclusión de estas funcionalidades no agrega apenas valor al proyecto, al tratarse de contenido que difícilmente varíe.

Cabe destacar que estas y más cuestiones que quedan fuera del alcance, pero podría resultar añadir en versiones futuras del proyecto, se analizan en profundidad en la sección 10.2.

Para representar de forma visual las diversas partes del proyecto a atajar para satisfacer los objetivos principales, se presenta un diagrama de causa-efecto, también conocido como árbol de características o diagrama de *Ishikawa*. Este diagrama, presentado en la figuras 1.1 y B.1 (versión ampliada), permitirá entender de un vistazo qué características estarán dentro del alcance del proyecto, siguiendo una organización por categorías.

Por último, me gustaría comentar ciertos aspectos no ligados a funcionalidades de la aplicación, pero que se tendrán en cuenta durante la elaboración del proyecto.

Para facilitar la elaboración y realizar un seguimiento de los avances, se utilizará de forma exhaustiva un control de versiones, de forma que no se pierdan las nuevas implementaciones en caso de error. Esto permitirá retornar a versiones antiguas en caso de conflictos e incluso facilitar la organización y el estudio de los avances del proyecto.

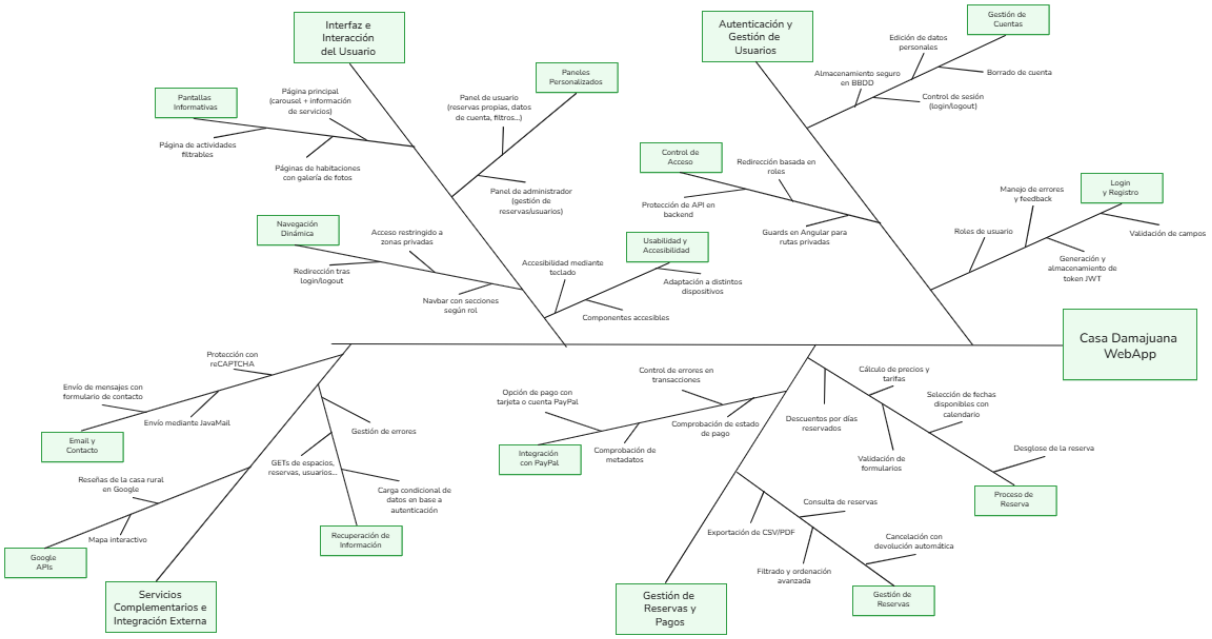

Figura 1.1: Diagrama de Características del proyecto

También es necesario mencionar la aplicación de ciertas metodologías que faciliten la organización a la hora de trabajar en el proyecto, maximizando los avances. Dichas estrategias se desglosarán en la sección 2.1.

En definitiva, contando con un alcance claro y bien definido, podremos acotar el proyecto inicial, evitando retrasos o malos usos de los recursos.

1.6. Estructura de la documentación

En esta última sección del capítulo introductorio, expondré un breve resumen de los capítulos que componen el documento. Se explicará la misión de cada uno de ellos, facilitando la comprensión y conocimiento de los contenidos que contienen. Cabe destacar que algunas secciones se dividirán a su vez en subsecciones.

Dicho esto, la memoria cuenta con los siguientes capítulos:

- **Capítulo 1 - Introducción:** permite poner en contexto al lector, exponiendo la motivación que ha llevado a tomar la decisión de realizar el proyecto. Además, reúne el análisis inicial de los objetivos a cumplir, diversos condicionantes del proyecto, la definición del alcance y la estructura que seguirá la documentación presentada.

- **Capítulo 2 - Planificación:** reúne la explicación de las técnicas y metodologías empleadas para gestionar los recursos usados para el desarrollo del proyecto. Se realizarán las estimaciones temporales, económicas y de esfuerzo correspondientes,

desembocando en un balance final que represente si se han cumplido o no, y en que medida, los cálculos estimados.

- **Capítulo 3 - Antecedentes:** trata de ilustrar al lector acerca de la magnitud y campos abarcados por la aplicación. Se comentará brevemente la solución tecnológica por la que se ha optado, así como las herramientas empleadas para la construcción del proyecto. Además, se realizará un breve estudio de antecedentes y competidores o soluciones similares, identificando similitudes y diferencias.

- **Capítulo 4 - Análisis:** incluye un estudio exhaustivo de los requisitos identificados, las necesidades y roles de los usuarios, las posibles interacciones y flujos al interactuar con la aplicación y las funcionalidades que ofrece esta.

- **Capítulo 5 - Diseño:** detalla las arquitecturas del sistema, el modelado de los datos y el diseño de la interfaz de la aplicación.

- **Capítulo 6 - Implementación:** aborda cuestiones ligadas a la implementación en las diversas capas que componen la aplicación. Se explicarán ciertos elementos propios de los *frameworks* utilizados, así como *ítems* relevantes. Además, se expondrán los servicios externos integrados en la aplicación. También se tratarán temas importantes, considerados pilares en la implementación realizada, como son la autenticación, la seguridad, la transaccionalidad, etc.

- **Capítulo 7 - Pruebas y aceptación:** recoge la batería de pruebas realizadas de diversos tipos que demuestre el funcionamiento esperado de la aplicación, verificando que se cumplen los requisitos de manera objetiva.

- **Capítulo 8 - Manual de Instalación:** enumera los pasos a seguir para lograr la puesta a punto del entorno de trabajo, incluyendo configuraciones adicionales, instalaciones necesarias y acciones requeridas para la obtención de claves para ciertas *APIs*.

- **Capítulo 9 - Manuales de Uso:** facilita ciertos manuales que explican el uso de la aplicación desarrollada desde el punto de vista de los diversos tipos de usuarios reconocidos, evitando bloqueos y dudas tanto a usuarios corrientes como a administradores.

- **Capítulo 10 - Conclusiones y mejoras:** enumera ciertas consideraciones futuras, como mejoras y ampliaciones que se pueden realizar, cerrando la documentación de la aplicación con una serie de conclusiones al respecto.

- **Capítulo A - Contenido del repositorio:** define la estructura de directorios de la entrega del código fuente del proyecto.

- **Capítulo B - Anexos:** contiene versiones ampliadas de ciertos diagramas, con la finalidad de mejorar su visualización. De esta forma, al consultar la documentación

en papel, se dispondrá de diagramas en formato horizontal, que permitan al lector una consulta de la información más sencilla.

- **Bibliografía y *Webgrafía*:** contiene una serie de citas relevantes para complementar la documentación expuesta. Se citarán numerosas fuentes de información y dominios de interés.

Capítulo 2

Planificación

A la hora de realizar un proyecto de esta envergadura, es esencial contar con un plan robusto y fiable, que permita establecer una rutina de progreso efectiva. Por tanto, en este capítulo se abordarán los temas relativos al plan mencionado, tanto en el aspecto económico-temporal, como a nivel de esfuerzo.

2.1. Metodología de trabajo

En primer lugar, se expondrá la metodología acatada para desarrollar el trabajo. Todo proyecto *software* adopta una metodología, dependiendo de ciertas necesidades del cliente, del equipo de trabajo o de la propia naturaleza de la organización que lo desempeña.

En mi caso, en lugar de decantarme por una metodología tradicional, que se atiene a los requisitos establecidos inicialmente, mostrando menos flexibilidad y un progreso diferenciado por etapas, decidí elegir una metodología ágil.

Las metodologías ágiles son de las más usadas actualmente en empresas dedicadas a crear *software*, debido a la gran capacidad de coordinación que ofrecen entre compañeros de equipo, así como a la comunicación e implicación del cliente.

Sin embargo, marcos ágiles como *Scrum*, que ya hemos aplicado en asignaturas o prácticas de la carrera, son propios de equipos con varios integrantes, por lo que no tendría sentido aplicarlos en un trabajo individual.

Por ello, tras una larga investigación al respecto, encontré la metodología *ASAP* (*Agile Student Academic Projects*) [51]. Esta metodología busca rescatar ciertas prácticas empleadas en las famosas metodologías ágiles, con el fin de adaptarlas a la elaboración del Trabajo Fin de Grado.

Procedo a citar varios elementos que describen la dinámica de trabajo con esta metodología, conectándolos con la forma en la que la he empleado para elaborar mi proyecto:

- **1 - Roles:** como se comenta en el artículo citado, a pesar de que el Trabajo Fin de Grado es un proyecto individual, hay varias figuras que afectan al proyecto. Los roles que distingue *ASAP* son el de estudiante, tutor/es, comunidad y tribunal.

Efectivamente esta distinción de individuos es acorde a la que ha existido en mi proyecto a lo largo de su desarrollo, pues como estudiante, he establecido contacto y recibido *feedback* de mis tutores, he consultado ciertas opiniones a diversos profesores de la universidad y tendré que explicar y defender mi trabajo ante el tribunal.

- **2 - Eventos:** al igual que en las metodologías *agile*, *ASAP* fomenta la división del Trabajo Fin de Grado en *sprints*. En este caso, aplica una duración inferior a un mes. Durante esos *sprints*, se realizan las reuniones de inicio (al comienzo de cada *sprint*), las reuniones de sincronización, las comunicaciones de progresos y las retrospectivas.

  En mi caso particular, las reuniones de inicio de *sprint* se llevaban a cabo de forma presencial con ambos tutores (salvo si alguno necesitaba incorporarse a la reunión vía *Teams*). Respecto a las reuniones de sincronización, normalmente tenían lugar por correo electrónico si no era posible asistir de forma presencial, exponiendo los avances y dudas, y recibiendo *feedback*. A pesar de no ser la filosofía original, se trató de adaptar a las necesidades del proyecto y a los tiempos y horarios de los asistentes. Por su parte, durante las comunicaciones de progresos expuse los avances realizados a los tutores, a modo de *demo* explicativa, con la finalidad de derivar en las retrospectivas finales.

- **3 - Artefactos:** según *ASAP*, encontramos los incrementos y las retroalimentaciones. Los incrementos son aquellos avances que fui realizando que iban aportando valor, mientras que las retroalimentaciones me permitían reflexionar a nivel individual y colectivo junto a los tutores. En definitiva, estuvieron presentes de una o de otra manera en mi proyecto.

- **4 - Entorno de trabajo:** este espacio debía facilitar la coordinación entre tutores y estudiante, pero al mismo tiempo adaptarse a las necesidades de cada uno. Por ello, tras debatirlo con mis tutores, decidimos que el canal de comunicación a usar, a excepción de las reuniones presenciales, sería el correo electrónico. De esta manera, ciertas dudas o bloqueos eran comunicados por este medio de preferencia de mis tutores.

  En el caso de la compartición de ciertos recursos, como la memoria técnica, me adapté a las preferencias personales de cada tutor: compartí la memoria en un entorno *online* (*Overleaf*) para uno de los tutores, mientras que generaba el archivo *PDF* para el segundo, debido a que le facilitaba la corrección.

  Respecto a ciertos elementos que *ASAP* considera importantes para la organización, utilicé un tablero *Kanban* para la organización personal de tareas, facilitando así su categorización. Apliqué diversos colores en base al tipo: rojo para las tareas de *frontend*, verde para las tareas de *backend*, azul para las relacionadas con la base de datos, morado para las actividades relacionadas con la redacción de la memoria y amarillo para las tareas de investigación. Se muestra un ejemplo en la figura 2.1.

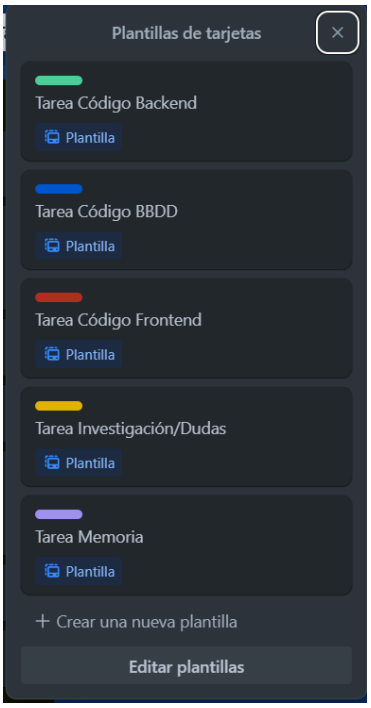

Figura 2.1: Tipos de tareas en el tablero *Kanban* del proyecto

> Cabe destacar que no se empleó un cuaderno de trabajo elaborado exhaustivamente para recopilar los avances realizados, puesto que consideramos que no aportaba tanto valor. En su defecto, fui recopilando ciertas dudas, avances y cuestiones de manera menos formal.

En definitiva, escogí *ASAP* como metodología base para elaborar mi proyecto, realizando pequeñas modificaciones para ajustarme a mis necesidades y a las de los tutores, sin romper con la filosofía de esta forma de trabajo.

De esta manera, al finalizar cada *sprint* podía presentar a mis tutores un subconjunto funcional de la aplicación, obteniendo *feedback* constante y siendo flexibles de cara a adaptarnos a las circunstancias que iban surgiendo.

2.2. Estimación de esfuerzo

Es bien sabido que una estimación *software* trata de una predicción de cuánto tiempo durará y costará desarrollar y mantener cierto proyecto, pudiéndose medir en unidades de esfuerzo y de coste, entre otras.

Existen numerosas técnicas de estimación del *software*, que se suelen distinguir en los grupos de técnicas basadas en la experiencia (como el juicio de expertos, la analogía, la descomposición, etc.) y técnicas basadas en el modelado algorítmico (destacando *COCOMO II*).

Para la estimación del esfuerzo de mi proyecto he optado por emplear la técnica de Puntos de Función (*Function Points*) [48] desarrollada por Albrecht, en lugar del modelo *COCOMO*, debido a varias razones:

- En primer lugar, el modelo *COCOMO* requiere como entrada una estimación de las líneas de código fuente (*SLOC*), un parámetro difícil de determinar con precisión en entornos donde se emplean *frameworks* modernos como *Angular* o *Spring Boot*, que encapsulan gran parte del comportamiento en bibliotecas predefinidas.

  En cambio, el método de Puntos de Función permite realizar una estimación basada en las funcionalidades que percibe el usuario, lo que lo hace más adecuado para proyectos como el mío, centrados en el desarrollo de una aplicación *web* con componentes bien definidos (entradas, salidas, consultas, etc.).

- En segundo lugar, este método se adapta mejor a proyectos de desarrollo individual o con equipos reducidos, como en este Trabajo Fin de Grado, en el que el grueso del proyecto lo desarrolla el estudiante. *COCOMO*, en cambio, está pensado para entornos empresariales con múltiples factores de coste asociados al personal, gestión del proyecto y herramientas de desarrollo.

- Por último, la técnica de Puntos de Función permite obtener una estimación razonable incluso en fases intermedias o finales del proyecto, funcionando como una herramienta útil tanto para una planificación como esta, como para fases de validación. En este sentido, se ajusta más a las necesidades de este trabajo, pudiendo adaptarse al momento en el que se encuentra el proyecto.

En definitiva, aplicaré el método de Puntos de Función, comenzando con la identificación de los elementos funcionales del sistema, continuando con el cálculo de los Puntos de Función No Ajustados (PFNA) y culminando con el ajuste de puntos de función.

Para la primera fase, debemos identificar una serie de elementos funcionales del sistema, que pueden ser de los siguientes tipos:

- **Entradas Externas (EE):** son entradas que provienen del exterior del sistema, como datos ingresados mediante formularios.

- **Salidas Externas (SE):** son salidas que genera el sistema y envía al exterior, como informes o resultados para visualizar.

- **Consultas Externas (CE):** son interacciones que recuperan datos de la base de datos sin realizar cálculos complejos.

- **Archivos Lógicos Internos (ALI):** son bases de datos o archivos que el sistema mantiene y gestiona.

- **Archivos de Interfaz Externa (AIE):** son archivos o bases de datos mantenidos y gestionados por otro sistema, pero que nuestro sistema emplea para obtener información.

Los elementos de los tipos citados deben catalogarse en base a dos categorías, según su complejidad:

- **Tipos de Datos Elementales (TED):** es un dato único e identificable que representa un elemento de información. Se usa para medir la complejidad de EE, SE, CE, ALI Y AIE.

- **Tipos de Registros (TR):** es un subgrupo lógico de datos dentro de un ALI o un AIE.

Para el cálculo de la complejidad de cada elemento, emplearé las tablas 2.1, 2.2 y 2.3.

ALI o AIE	1-19 TED	20-50 TED	>50 TED
1 TR	Baja	Baja	Media
2 a 5 TR	Baja	Media	Alta
6 o más TR	Media	Alta	Alta

Cuadro 2.1: Determinación de la complejidad para ALI/AIE

EE y CE	1-4 TED	5-15 TED	>15 TED
0-1 TR accedidos	Baja	Baja	Media
2 TR accedidos	Baja	Media	Alta
>2 TR accedidos	Media	Alta	Alta

Cuadro 2.2: Determinación de la complejidad para EE y CE

SE	1-5 TED	6-19 TED	>19 TED
0-1 FR accedidos	Baja	Baja	Media
2-3 FR accedidos	Baja	Media	Alta
>3 FR accedidos	Media	Alta	Alta

Cuadro 2.3: Determinación de la complejidad para SE

Conociendo estas tablas, pasamos a evaluar la complejidad de cada elemento, como se muestra en las tablas 2.4, 2.5, 2.6, 2.7 y 2.8.

Descripción	**Interacción del usuario**	**Complejidad**	**Justificación**
Formulario de *login*	Usuario introduce *email* y contraseña	Baja	Solo 2 campos, validación simple, lógica directa.
Formulario de registro	Introduce nombre, apellidos, *email*, contraseña y confirmación	Media	Validación + comprobación existencia + escritura BBDD.
Formulario de reserva	Selección fechas, datos personales, mensaje, forma de pago...	Alta	Calcula precios, valida fechas, descuentos, múltiples validaciones.
Formulario de contacto	Nombre, *email*, asunto y mensaje	Media	Se envía *email* vía *backend*. Validaciones básicas.
Cancelar reserva	Botón de acción con modal de confirmación	Alta	Modifica BBDD y lanza reembolso vía *PayPal API*.
Eliminar usuario	Botón de acción con modal de confirmación	Media	Llama `DELETE` y elimina la entidad.
Filtros en tablas	Filtrado por nombre, fecha, estado...	Media	Campos de búsqueda afectan *query* dinámica en BBDD.

Cuadro 2.4: Entradas Externas (EE)

Descripción	**Contenido mostrado**	**Complejidad**	**Justificación**
Mensaje de *login*/registro	Éxito o error	Baja	Mensaje simple.
Resumen de reserva	Fechas, desglose del precio, nombre y *email*	Media	Datos calculados dinámicamente.
Toast de cancelación/errores	Mensajes simples en pantalla	Baja	Muestra *feedback* de acción del usuario.
Tabla de reservas	Datos de reserva, cancelación activa	Alta	Contenido filtrable, exportable, interactivo.
Tabla de usuarios	Datos básicos de usuario	Media	Visualización simple con filtrado.
Mapa de contacto (*Google Maps*)	Mapa interactivo, *street view*, *pin* personalizado	Media	*API* funcional e incrustada, con elementos interactivos.

Cuadro 2.5: Salidas Externas (SE)

Descripción	**Tipo de consulta**	**Complejidad**	**Justificación**
Verificar *login*	Consulta si existe usuario con credenciales dadas	Baja	Solo lectura, rápida y directa.
Comprobar duplicado en registro	Comprobar si el *email* ya existe	Baja	Consulta con resultado *booleano*.
Consultar habitaciones	Obtener todas las habitaciones disponibles	Media	Carga en tabla con imagen y botones.
Consultar detalles habitación	`GET` por `id` habitación	Media	Incluye múltiples fotos, texto, estructura.
Cargar reseñas *Google*	Llamada a *Google Places API*	Alta	*API* externa con procesamiento de datos.
Cargar reservas del usuario	Consulta con filtros múltiples	Alta	Consulta compleja con múltiples parámetros.
Consultar usuarios (*admin*)	Consulta todos los usuarios con filtros	Media	Solo lectura, pero con filtros múltiples.

Cuadro 2.6: Consultas Externas (CE)

Entidad	**Uso**	**Complejidad**	**Justificación**
Usuarios	Alta (*login*, registro, *admin*)	Media	*CRUD* completo con roles, seguridad.
Reservas	Alta (reserva, cancelación, cálculo)	Alta	Múltiples atributos y reglas.
Habitaciones	Consulta general y detalle	Media	Datos estáticos + visualización.

Cuadro 2.7: Ficheros Internos (ALI)

Servicio	**Uso**	**Complejidad**	**Justificación**
Google Maps JavaScript API	Mostrar mapa interactivo	Media	Embebido, configurable, datos externos.
Google Places API	Carga reseñas públicas	Alta	Consulta, *parseo*, *formateo*.
PayPal API	Crear pago, verificar, reembolsar	Alta	*API* crítica, varios *endpoints*.
JavaMail (*SMTP*)	Envío de correos desde *backend*	Media	Validación, formato, *backend* dependiente.

Cuadro 2.8: Ficheros Externos (AIE)

Pasamos ahora a utilizar la tabla de conversión, asignando ciertos pesos en base a la complejidad del componente. La tabla a emplear será la referenciada como 2.9.

Tipo de función	**Simple**	**Media**	**Alta**
Entradas de usuario (EE)	3	4	6
Salidas de usuario (SE)	4	5	7
Consultas de usuario (CE)	3	4	6
Ficheros internos (ALI)	7	10	15
Ficheros externos (AIE)	5	7	10

Cuadro 2.9: Ponderación de funciones por complejidad

Y empleando la tabla anterior, pasamos a realizar la ponderación aplicada a mi sistema, mostrando antes la tabla del resumen de complejidades en la figura 2.10.

Tipo	**Simple**	**Media**	**Alta**
Entradas Externas (EE)	1	4	2
Salidas Externas (SE)	2	3	1
Consultas Externas (CE)	2	3	2
Ficheros Lógicos Internos (ALI)	0	2	1
Ficheros de Interfaz Externa (AIE)	0	2	2

Cuadro 2.10: Resumen de transacciones por tipo y complejidad

Ahora sí, pasamos a calcular las ponderaciones:

$$\text{EE} = (1 \times 3) + (4 \times 4) + (2 \times 6) = 3 + 16 + 12 = 31$$

$$\text{SE} = (2 \times 4) + (3 \times 5) + (1 \times 7) = 8 + 15 + 7 = 30$$

$$\text{CE} = (2 \times 3) + (3 \times 4) + (2 \times 6) = 6 + 12 + 12 = 30$$

$$\text{ALI} = (0 \times 7) + (2 \times 10) + (1 \times 15) = 0 + 20 + 15 = 35$$

$$\text{AIE} = (0 \times 5) + (2 \times 7) + (2 \times 10) = 0 + 14 + 20 = 34$$

Sumando todo, obtenemos los puntos de función no ajustados (PFNA).

$$\text{PFNA} = \text{EE} + \text{SE} + \text{CE} + \text{ALI} + \text{AIE}$$

$$\text{PFNA} = 34 + 30 + 30 + 35 + 34 = \boxed{160}$$

Mediante el cálculo del Factor de Ajuste del Valor (FAV), lograremos realizar el ajuste necesario. Para ello, emplearemos la tabla 2.11.

Nº	**Descripción del factor de ajuste**
1	Comunicación de datos
2	Funciones distribuidas
3	Prestaciones
4	Gran uso de la configuración
5	Velocidad de las transacciones
6	Entrada *online* de datos
7	Diseño para la eficiencia del usuario final
8	Actualización de datos *online*
9	Complejidad del proceso lógico interno de la aplicación
10	Reusabilidad del código
11	Facilidad de instalación
12	Facilidad de operación
13	Localizaciones múltiples
14	Facilidad de cambios

Cuadro 2.11: Factores de ajuste de valor (FAV)

El cálculo del FAV está basado en 14 características generales de los sistemas que miden la funcionalidad general y complejidad o influencia de la aplicación. A cada característica se le atribuye un peso de 0 a 5, como realizo en la tabla 2.12.

Por tanto, calculamos el FAV:

$$\text{Suma } GFI = 4 + 4 + 2 + 4 + 1 + 2 + 3 + 4 + 3 + 3 + 1 + 0 + 2 + 2 = \boxed{35}$$

$$\text{FAV} = 0{,}65 + (0{,}01 \times \text{Suma } GFI) = 0{,}65 + (0{,}01 \times 35) = \boxed{1{,}00}$$

Y en base a ello, usando el FAV y los PFNA podemos calcular los PFA:

$$\text{PFA} = \text{PFNA} \times \text{FAV} = 160 \times 1{,}00 = \boxed{160}$$

Una vez obtenidos los PFA, paso a calcular aproximadamente el tiempo requerido para desarrollar el proyecto. En mi caso, he tomado una relación de equivalencia (RE) de 3 horas para cada punto de función ajustado.

Nº	Descripción del Factor	Valor
1	¿El sistema requiere comunicación intensiva con otros sistemas?	4
2	¿Hay procesamiento complejo interno?	4
3	¿El sistema debe ser eficiente en rendimiento?	2
4	¿El sistema requiere un diseño eficiente de usuario final (UX)?	4
5	¿El sistema requiere instalación/configuración especial o personalización?	1
6	¿El sistema requiere procesamiento simultáneo/-multiplataforma?	2
7	¿Es importante la reutilización del *software*?	3
8	¿El sistema incluye entrada/salida de datos compleja?	4
9	¿El sistema incluye actualizaciones de datos complejas?	3
10	¿Hay procesamiento complejo de transacciones internas?	3
11	¿La instalación requiere capacitación especial para los usuarios?	1
12	¿El sistema está diseñado para múltiples sitios u organizaciones?	0
13	¿El sistema requiere mantenimiento o adaptación frecuente?	2
14	¿El sistema debe manejar múltiples idiomas o localizaciones?	2

Cuadro 2.12: Factores Generales de Influencia

$$\text{RE} = 3 \text{ horas/PFA}$$

Esta correspondencia puede variar, pero me he decantado por ella debido a que la curva de aprendizaje de *frameworks* como *Angular* es muy elevada al principio, pero poco a poco comienza a facilitar dichas tareas, mediante reutilización de código. Por ello, considero que puede ser una aproximación acorde a la realidad.

$$\text{Tiempo estimado} = \text{PFA} \times \text{RE} = 160\ PFA \times 3\ horas/PFA = \boxed{480\ h}$$

Según el cálculo realizado, completar mi proyecto llevaría unas 480 horas de trabajo.

2.3. Planificación temporal

Todo proyecto requiere establecer ciertos plazos o estimaciones temporales que nos permitan hacernos a la idea de cómo irá progresando. Por ello, es esencial tener en cuenta una planificación temporal que permita llevarlo a cabo correctamente.

Como se menciona en el apartado anterior, estudiando a fondo la complejidad de las funcionalidades de mi proyecto, su desarrollo completo tomaría unas 480 horas aproximadamente. Este cálculo puede variar, dependiendo del grado de dificultad de ciertos problemas, de la aparición de inconvenientes e incluso de las circunstancias personales del alumno.

Sin embargo, nos ayuda a estimar el tiempo a dedicar para completar los objetivos del proyecto, estableciendo un abanico más o menos ajustado de meses de trabajo.

El apartado anterior estaba centrado en calcular la complejidad y el esfuerzo necesario para elaborar este trabajo, mientras que en esta sección nos centraremos en realizar una predicción justificada de cuál será la distribución de dicho esfuerzo a lo largo del tiempo, calculando el trabajo a ejecutar a lo largo de cada semana, así como estableciendo un cronograma base a seguir.

Obviamente pueden existir variaciones a lo largo del proyecto, pues la realidad plantea diversas situaciones que llevan a reconducir el proyecto y adaptarnos a los cambios surgidos, más aún al emplear una metodología ágil, cuya base es la adaptación continua a los cambios.

En definitiva, este apartado nos permitirá realizar una planificación temporal inicial, de forma que podamos mirar atrás según vaya avanzando el proyecto y verificar si hemos seguido las pautas establecidas.

Partiendo del esfuerzo estimado de 480 horas para completar el proyecto, habiendo establecido previamente una equivalencia de 3 horas de trabajo por cada Punto de Función Ajustado, podemos estimar la disposición de dichas horas a lo largo de los meses de trabajo.

Para ello, estableceré dos jornadas: por un lado, la de una persona o profesional totalmente dedicado al trabajo, y por el otro una jornada realista adaptada a mis necesidades, teniendo en cuenta otras ocupaciones semanales.

Un profesional en el sector contaría con una jornada laboral de 8 horas diarias, de lunes a viernes, computando un total de 40 horas semanales. Además, a lo largo del mes contaría con 21 días laborables aproximadamente.

$$\text{Horas mensuales} = 8 \text{ horas/día} \times 21 \text{ días/mes} = 168 \text{ horas/mes}$$

Por tanto, dedicaría unas 168 horas mensuales a realizar el proyecto.

$$\text{Tiempo estimado en meses} = 480 \text{ horas} \div 168 \text{ horas/mes} = 2{,}5871 \text{ meses}$$

Entonces, una persona totalmente dedicada al proyecto, con una jornada laboral estándar de 40 horas semanales, tardaría unos 2 meses y medio en completarlo.

Sin embargo, como comentaba anteriormente, hay que tener en cuenta que al tratarse de un Trabajo Fin de Grado, realizado normalmente durante un cuatrimestre universitario, hay que incluir en la ecuación ciertas ocupaciones adicionales del alumno, como otras asignaturas, prácticas curriculares, etc.

En mi caso, he establecido entonces un aproximado de 6 horas diarias de trabajo, contando con unos 21 días de trabajo inamovibles cada mes.

$$\text{Horas mensuales} = 6 \text{ horas/día} \times 21 \text{ días/mes} = 126 \text{ horas/mes}$$

Ahora, pasaría a calcular el total de meses aproximado para ejecutar el proyecto.

$$\text{Tiempo estimado en meses} = 480 \text{ horas} \div 126 \text{ horas/mes} = 3{,}8095 \text{ meses}$$

Es decir, planteando una distribución temporal que tenga en cuenta otras ocupaciones del alumno, mi proyecto podría completarse en casi 4 meses de trabajo.

Muestro un cronograma base, que representa una distribución ideal del trabajo en *sprints*.

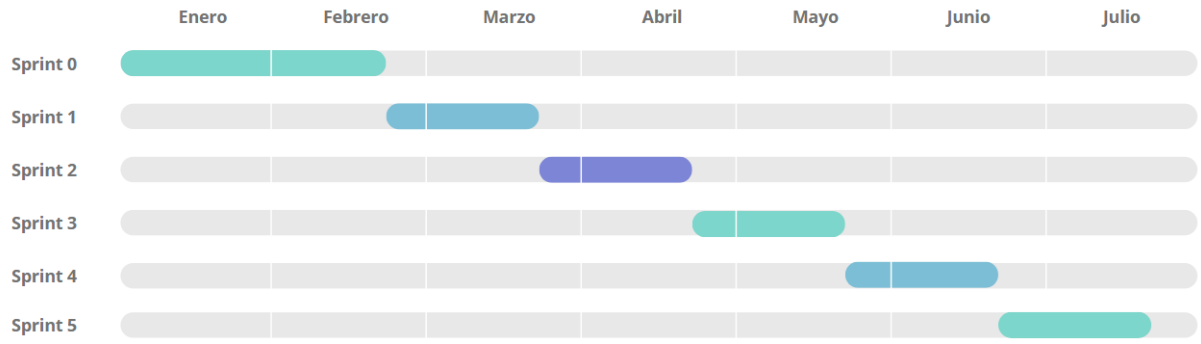

Figura 2.2: Diagrama de *Gantt* (cronograma) ideal del proyecto

Nótese que el *Sprint 0* se corresponde con una etapa inicial de investigación, reuniones previas y decisiones estructurales. A partir de ahí, y tras la presentación de la Comunicación Oficial de Inicio, se establecen una serie de *sprints* de un mes.

Este planteamiento inicial indica que, idealmente, el trabajo se ejecutaría en periodos de un mes, delimitados por las reuniones correspondientes explicadas en la sección 2.1.

La distribución real del tiempo se podrá ver, según avance el proyecto, en la sección 2.5, en la que se contrastará la planificación inicial con las fases reales llevadas a cabo con el transcurso del proyecto.

En definitiva, se da por estudiada la organización temporal a seguir, adecuándose a los cálculos relativos al esfuerzo estimado, así como a la metodología a aplicar.

2.4. Presupuesto económico

En todo proyecto debemos plantear una estimación de los costos que puede acarrear la realización del trabajo. Para ello, a lo largo de esta sección citaré los diversos gastos *hardware*, *software*, de personal y de infraestructura.

Cabe destacar que enumeraré tanto costos directos como indirectos, siendo los primeros aquellos que dependen del proyecto, mientras que los segundos no se pueden determinar con exactitud.

En primer lugar, presento una tabla en la que recojo los diversos costes *hardware* provocados por la realización del proyecto. Como el *hardware* no se ha comprado única y exclusivamente para este proyecto, sería un enfoque erróneo incluir el precio completo de equipos y componentes, por lo que se aplicará la técnica de amortización de equipos.

Esta técnica trata de estimar el uso de los componentes empleados. teniendo en cuenta la vida útil esperada para cada tipo de herramienta. De esta forma, se puede realizar un cálculo más realista, sin engordar innecesariamente el presupuesto final.

Componentes	**Coste (€)**	**Vida útil (años)**	**Uso (%)**	**Total (€)**
Ordenador portátil	700,00€	4	8,33 %	58,31€
Ratón	50,00€	3	11,11 %	5,56€
Monitor	129,00€	5	6,67 %	8,60€
Teclado	40,00€	3	11,11 %	4,44€
PC Sobremesa	800,00€	5	6,67 %	53,36€
Total	—	—	—	**130,27€**

Cuadro 2.13: Coste proporcional del *hardware* utilizado durante la duración del proyecto

Para realizar el cálculo anterior, se ha estimado una vida útil de 3 años para los periféricos menores, 4 años para el portátil y 5 años para el monitor y el ordenador de sobremesa. Para ello, se han tenido en cuenta las características de cada tipo de componente, la facilidad con la que pueden estropearse, etc.

Además, por mi experiencia con estos artilugios, la vida útil es relativamente acertada, pues he trabajado con numerosos componentes de cada tipo, y su duración total ha sido similar a las que se exponen en la tabla previa.

Pasamos a enumerar los costes *software* del proyecto.

Servicio	**Coste (€)**	**Duración**	**Uso (%)**	**Total (€)**
Windows 11 Pro (Portátil)	199,00€	4 años	8,33 %	16,58€
Windows 10 Home (*PC*)	139,00€	5 años	6,67 %	9,26€
XAMPP	Gratuito	-	-	0,00€
VSCode	Gratuito	-	-	0,00€
Spring Tool Suite 4	Gratuito	-	-	0,00€
Postman	Gratuito	-	-	0,00€
GitHub/SourceTree	Gratuito	-	-	0,00€
MySQL	Gratuito	-	-	0,00€
Navegadores varios	Gratuito	-	-	0,00€
Overleaf	Gratuito	-	-	0,00€
PayPal	Gratuito *	-	-	0,00€
API Places Google	Gratuito *	-	-	0,00€
API Maps JavaScript Google	Gratuito *	-	-	0,00€
API reCAPTCHA Enterprise	Gratuito *	-	-	0,00€
JavaMail	Gratuito	-	-	0,00€
Total	—	—	—	**25,84€**

Cuadro 2.14: Coste estimado de *software* y servicios durante la duración del proyecto

* *Nota:* Actualmente, los servicios de ***PayPal***, ***Google Maps JavaScript***, ***Google reCAPTCHA Enterprise*** y ***Google Places*** se encuentran bajo planes gratuitos o con créditos promocionales. Sin embargo, en un entorno de producción o con mayor volumen de uso, estos servicios podrían tener un coste asociado. Por ejemplo:

- ***Google Maps Platform*** incluye 200$ mensuales gratuitos, que una vez superados hace que se tarifiquen las peticiones (p.e. 7$ por 1000 cargas estáticas de mapa).
- ***Google Places API*** también se tarifica tras cierto umbral gratuito (p.e. 17$ por 1000 solicitudes de *'Place Details'*).
- ***PayPal*** no cobra por el uso de su *API*, pero sí aplica comisiones sobre las transacciones económicas (habitualmente entre el 2,9 % + 0,35€ por operación).

Por lo tanto, en lo relativo al proyecto acotado como Trabajo Fin de Grado, que no contempla el despliegue real y utiliza licencias gratuitas, el coste de *software* y servicios sería de unos 25,84€. Nótese que las licencias de *Windows 11 Pro* y *Windows 10 Home* se agregan a la tabla aplicando la misma técnica de amortización empleada para los equipos que las contienen.

Pasamos a estimar los costes humanos, teniendo en cuenta los diversos roles desempeñados por el alumno durante la elaboración del proyecto. Para ello, se diferencian diversos tipos de tareas ejecutadas, estableciendo unas aproximaciones temporales en base al tiempo dedicado por el alumno.

Obviamente, este cálculo no es totalmente preciso, pero nos permite hacernos a la idea de ciertos costes ligados a salarios del personal de desarrollo, dando toque formal y realista a las estimaciones económicas.

Rol	**% del tiempo**	**Horas**	**€/h**	**Coste (€)**
Analista funcional	10 %	48	11€	528,00€
Diseñador *UX/UI*	10 %	48	10€	480,00€
Desarrollador *frontend*	25 %	120	12€	1.440,00€
Desarrollador *backend*	25 %	120	12€	1.440,00€
Administrador de BBDD	10 %	48	11€	528,00€
Tester / QA	10 %	48	10€	480,00€
Documentación/redacción	10 %	48	9€	432,00€
Total estimado	**100 %**	**480**	—	**5.328,00€**

Cuadro 2.15: Costes humanos estimados según rol desempeñado (perfil *junior*)

Nótese que se han empleado salarios aproximados para un perfil *junior* en España, simplemente por hacerse a la idea de los costes humanos que podría acarrear la realización del proyecto. En la realidad, al ser un Trabajo Fin de Grado, el alumno no recibe ningún salario, pero no está de más tener estas cifras presentes.

Además, los salarios mostrados varían con mucha facilidad en base al país, sector, empresa e incluso individuo, siendo imposible establecer una cifra exacta.

Por último, se agregan una serie de costes añadidos, denominados indirectos, que no se pueden establecer con tanta precisión. Entre ellos encontramos la luz o la conexión a *Internet*. Ambos recursos son claves para el correcto funcionamiento de las herramientas empleadas para construir el proyecto.

Concepto	**Coste mensual (€)**	**Duración (meses)**	**Total (€)**
Electricidad	35,00€	4	140,00€
Internet (fibra)	55,00€	4	220,00€
Total	—	—	**360,00€**

Cuadro 2.16: Otros costes estimados durante el desarrollo del proyecto

Finalmente, podemos computar el total de costes que acarrea la realización del proyecto.

Categoría	**Coste (€)**
Coste *Hardware*	130,27€
Coste *Software*	25,84€
Coste Humano	5.328,00€
Otros Costes (luz, *Internet*)	360,00€
Total General	**5.844,11€**

Cuadro 2.17: Resumen general de costes del proyecto

Por lo que estimo que la realización del proyecto propuesto costaría unos 5.844,11€ teniendo en cuenta la amortización de equipos para el *hardware*, aprovechando las licencias de *software* gratuitas y teniendo en cuenta costes humanos y de recursos esenciales como si de un proyecto real se tratara.

2.5. Balance final

Tras realizar las estimaciones oportunas en términos de esfuerzo, costes y tiempo, es relevante presentar una comparación de estas predicciones con los recursos empleados a lo largo del proyecto, una vez se encuentra este en su fase final.

Atendiendo al desarrollo del proyecto a lo largo de estos meses, y teniendo en cuenta los diversos *sprints* en los que se ha fragmentado la ejecución de este, puedo presentar un diagrama de *Gantt* que ilustre la división del trabajo a lo largo del ciclo de vida de este Trabajo Fin de Grado.

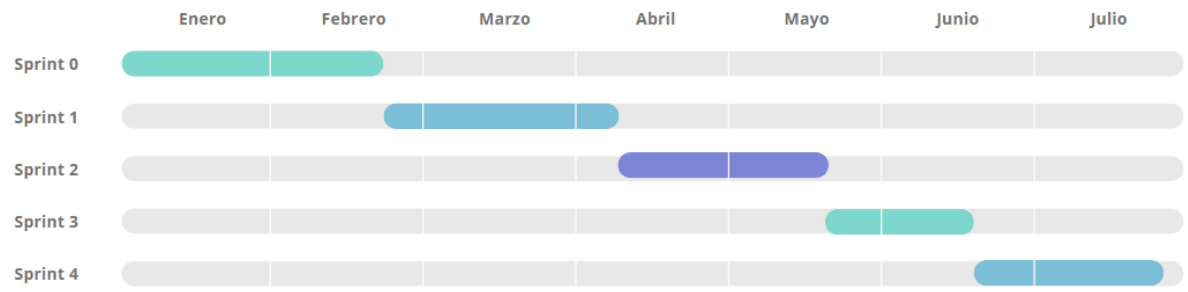

Figura 2.3: Diagrama de *Gantt* (cronograma) real del proyecto

Podemos observar que el trabajo se ve repartido en una serie de *sprints*, de aproximadamente un mes de duración cada uno. Durante estos periodos temporales se trataba de

avanzar en las funcionalidades y tareas previamente consensuadas con los tutores, en las llamadas reuniones de inicio.

Una vez determinadas las tareas a realizar durante el *sprint*, muchas veces definiéndolas como historias de usuario (expuestas en la sección 4.1.3), comenzaba con las labores de trabajo. Muchos de los avances requerían de investigación propia, lecturas de documentación de servicios externos, etc. Además, en ocasiones surgían ciertos bloqueos o dudas, que se trataban de resolver mediante reuniones de sincronización.

Al finalizar cada *sprint* se realizaban retrospectivas, obteniendo *feedback* de los tutores e incluso de ciertos profesores a los que se les realizaba alguna consulta. En definitiva, la distribución temporal se vio fragmentada en los siguientes *sprints*:

- **Sprint 0:** este *sprint* especial incluye las reuniones iniciales con los tutores, estableciendo ciertas pautas, los periodos iniciales de investigación y de formación en las tecnologías, etc. Básicamente, durante este periodo de tiempo se dedicaron muchos más recursos a la planificación y al aprendizaje de ciertas herramientas, que a la realización de trabajo efectivo como tal.

- **Sprint 1:** durante este periodo, realicé numerosas tareas ligadas al desarrollo de código en las tres capas de la aplicación. En *Angular* realicé el esqueleto inicial de la aplicación, agregué el *routing* base, establecí *mockups* de registro de usuarios y *login*, entre otros avances. En cuanto al *backend* logré elaborar parte de la lógica de la *API REST* de usuarios, vinculando el *frontend* y *backend*, a la par que realicé los primeros ajustes de *Spring Security*. Respecto a la base de datos, comencé a trabajar con los usuarios, vinculando el *backend* con la base a emplear.

- **Sprint 2:** a lo largo de este *periodo* se comenzó a trabajar en funcionalidades ligadas a los espacios de la casa rural. Además, comenzó la investigación e implementación de servicios externos con la aplicación *web*, logrando renderizar las primeras versiones del mapa y las reseñas. También se logró establecer la primera versión de pagos con *PayPal*. Respecto al *backend*, se mejoró la configuración de *Spring Security*, se implementó la *API REST* de espacios y se establecieron las primeras versiones de la lógica de reservas y pagos. En base de datos, se hizo hincapié en la realización de diversos *triggers*, claves para garantizar la integridad.

- **Sprint 3:** el tercer *sprint* fue el más breve de todos, pero uno de los más importantes. Se implementaron numerosas mejoras respecto a la autenticación, agregando los *tokens JWT*, se realizaron mejoras de seguridad, transaccionalidad e integridad de pagos, tarifas para las reservas, cancelaciones de reservas, etc. También se trabajó en mejoras sustanciales ligadas a la *API REST* y a la base de datos.

- **Sprint 4:** durante el último *sprint*, se realizaron importantes mejoras respecto a las cancelaciones de reservas, ciertas funcionalidades especiales para los usuarios registrados y más novedades. Se refinaron algunos componentes, estilos de la interfaz, etc. Además, se introdujeron mejoras como el uso de `HTTPS` en ambas capas.

Cabe destacar que este breve resumen por *sprints* se centra en mencionar ciertos avances funcionales, pero durante estos periodos hubo mucha tarea ligada a la investigación, aprendizaje y documentación del proyecto.

Echando la vista atrás, observamos que en las estimaciones se estableció una aproximación inicial de cuatro meses de duración del proyecto, basada en el cálculo de los puntos de función ajustados. Si observamos el cronograma, podemos comprobar que en realidad se ha empleado algo más de tiempo, aproximadamente un mes adicional. Obviamente, no estoy tomando en cuenta el tiempo transcurrido durante el *Sprint 0*, ya que como comentaba previamente, al ser un momento en el que realizaba las prácticas curriculares y el proyecto estaba en fases iniciales, el trabajo no era tan efectivo.

Por tanto, partiendo de una estimación inicial de unos cuatro meses de trabajo y teniendo en cuenta que en muchos casos la necesidad de emplear horas y horas para la formación en nuevas tecnologías y el estudio de documentación de servicios externos, considero que no se ha alejado tanto la aproximación previa de la realidad. A nivel temporal y de esfuerzo, por tanto, el estudio previo mostró una cantidad de horas de trabajo (ligado a funcionalidades) bastante coherente.

En cuanto a los costes, se han seguido a rajatabla las cifras establecidas durante las estimaciones. A nivel de uso de *hardware*, los componentes mencionados en las tablas de estimación son los que se han usado en la realidad. Los servicios externos empleados, gracias a las pruebas gratuitas obtenidas, no han supuesto cargos importantes.

Por todo esto, a pesar de no haber cumplido totalmente los plazos y estimaciones iniciales, considero que las variaciones no han sido demasiado grandes, tal y como suele ocurrir en un proyecto real.

Capítulo 3

Antecedentes

Una vez puesto en contexto el proyecto y definida una planificación fiable, quiero mostrar el entorno tecnológico escogido para implementar la aplicación. Además será esencial analizar otras aplicaciones y páginas *web* cuya misión es similar, con la finalidad de establecer una comparativa que recalque la necesidad de elaborar la aplicación *web* deseada.

Por último se expondrán las herramientas, tecnologías y aplicaciones empleadas para contribuir en cualquier fase o parte del proyecto, resumiendo su función y grado de importancia y uso durante el desarrollo.

3.1. Entorno de la aplicación

Como se comenta en secciones anteriores, la motivación general del proyecto es el desarrollo de una aplicación *web* destinada a la gestión de reservas en un alojamiento rural familiar, permitiendo a los usuarios consultar disponibilidad, realizar reservas y contactar con los administradores del servicio.

Este tipo de entornos suelen estar caracterizados por una gestión tradicional, muchas veces manual (correo, teléfono o incluso papel), lo que conlleva riesgos como errores humanos, solapamiento de fechas, falta de trazabilidad o tiempos de respuesta prolongados.

Por lo tanto, para el desarrollo se propone un enfoque *full-stack*, permitiendo desplegar una solución completa accesible desde cualquier navegador y adaptable a distintos dispositivos.

Este entorno de tipo ***full-stack web***, utiliza tecnologías modernas tanto para el cliente como para el servidor. El *stack* tecnológico principal podría denominarse de forma informal como ***ASMX***, por estar compuesto por:

- ***Angular*** [24] para el desarrollo del *frontend*, proporcionando una *SPA* (*Single Page Application*) responsiva y modular.
- ***Spring Boot*** [1] en el *backend*, ofreciendo una *API REST* robusta, segura y escalable.

- ***MySQL* [35]/ *MariaDB* [33]** como sistema de gestión de bases de datos relacional, donde se almacenan las entidades clave (usuarios, reservas y habitaciones).
- ***XAMPP*** [46] como entorno de servidor local, facilitando el despliegue de módulos como *Apache* y *MySQL* (en realidad *MariaDB*) durante la fase de desarrollo.

La elección de este entorno tecnológico se debe a motivos técnicos, a la relevancia de estas herramientas en el mercado actual y al propio reto que supone aplicar estas tecnologías no estudiadas en el Grado. Resumo a continuación ciertos motivos por los que me he decantado por este *stack*:

- ***Angular***: es uno de los *frameworks frontend* más utilizados a nivel empresarial en la actualidad. Desarrollado y mantenido por *Google*, *Angular* permite construir aplicaciones *SPA* (*Single Page Application*) robustas, mantenibles y escalables. Su arquitectura basada en módulos y componentes (aunque se está migrando hacia componentes *standalone*, sin módulos), la integración con *TypeScript*, y su sistema de inyección de dependencias facilitan el desarrollo profesional. Según encuestas como la de *Stack Overflow Developer Survey* [49] o el índice de *GitHub*, *Angular* sigue siendo una de las opciones preferidas por los desarrolladores con enfoque empresarial, sobre todo en sectores donde se requiere una estructura sólida y herramientas avanzadas para pruebas y desarrollo.
- ***Spring Boot***: es uno de los *frameworks* más populares para el desarrollo de *APIs* y aplicaciones *backend* en *Java*. *Spring Boot* es ampliamente adoptado en la industria gracias a su escalabilidad, seguridad, integración con bases de datos y soporte para arquitecturas modernas (*RESTful APIs*, microservicios, etc.). Organizaciones grandes y medianas lo emplean como estándar en el *backend*. Además, al ser un *framework* basado en *Java*, lenguaje que manejamos habitualmente en la carrera, he podido partir de cierta base de conocimiento, logrando profundizar más aún en las ventajas que ofrece.
- ***MySQL/MariaDB***: este sistema de gestión de bases de datos relacional es uno de los más utilizados en todo el mundo, tanto en proyectos pequeños como en soluciones empresariales. Su código abierto, fiabilidad, velocidad y compatibilidad con numerosos lenguajes y plataformas lo convierten en una elección natural para aplicaciones *web*. *MySQL* es compatible con *Spring Boot* de forma nativa, lo que facilita su integración. También destaca su facilidad de uso durante el desarrollo local con herramientas como *XAMPP*. Todo ello sumado a que, de nuevo, cuento con cierta experiencia en el manejo de *MySQL*, me permite trabajar cómodamente con los datos de mi aplicación. Nótese que aunque *XAMPP* mediante su panel de control permite la activación de un módulo llamado *MySQL*, este es en realidad *MariaDB*, por lo que a lo largo de la documentación se hablará indistintamente de uno o de otro, al ser *MariaDB* un *fork* de *MySQL*.

  De hecho, como mencionaba anteriormente, esta elección tecnológica está respaldada por la gran adopción de estas herramientas en el mercado actual:

- ***Angular***: según la encuesta de *Stack Overflow* 2023 [49] que se muestra en la figura 3.1, el 17,46 % de los desarrolladores indicaron haber trabajado con *Angular* en el último año. Aunque *React* supera a *Angular* en este caso (casi 41 %), este último mantiene una adopción sólida, especialmente en proyectos empresariales y entornos corporativos.

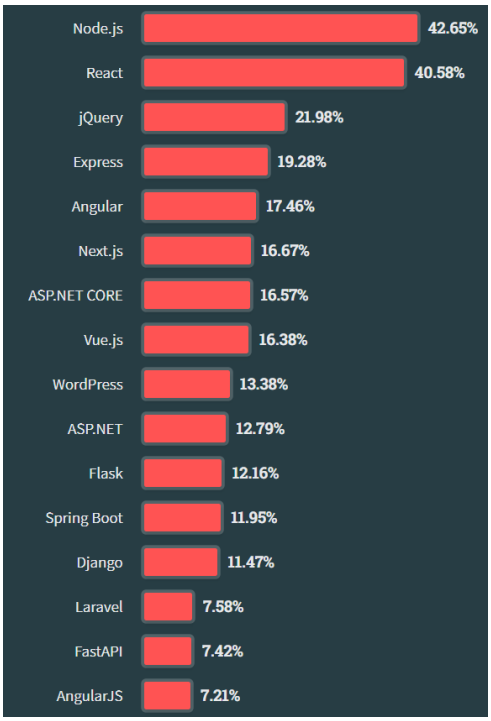

Figura 3.1: Encuesta de *Stack Overflow* 2023 - *Web frameworks and technologies*

- ***Spring Boot***: en la misma encuesta, *Spring Boot* aparece en la 11ª posición entre *frameworks* de *backend*, con una adopción del 11,95 %.
- ***MySQL***: respecto a sistemas de gestión de bases de datos, *MySQL* es la segunda opción más empleada, con un 41,09 % de adopción, superada solo por *PostgreSQL* (45,55 %). Por su parte, *MariaDB* cuenta con un 17,61 %, lo cual no está nada mal. Todo esto se expone en la encuesta de la figura 3.2.

La combinación de estas tecnologías ofrece un ecosistema sólido, probado y adecuado para un proyecto académico con visión profesional como este, permitiendo al mismo tiempo mantener buenas prácticas de desarrollo, separación de capas y escalabilidad futura.

En definitiva, es el caldo de cultivo perfecto para lograr desarrollar una aplicación *web* distintiva para un negocio familiar.

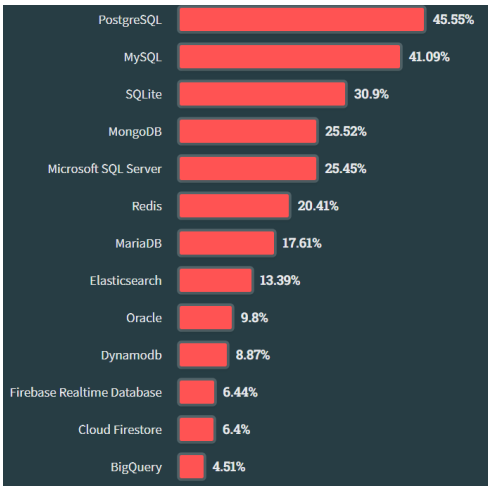

Figura 3.2: Encuesta de *Stack Overflow* 2023 - *Databases*

3.2. Estado del arte

Ya estudiado y seleccionado el entorno tecnológico que actuará como base del proyecto, toca analizar el mercado, pero en lugar de a nivel tecnológico como en la sección anterior, centrándonos en soluciones similares a la que proponemos.

Al hablar de alojamientos rurales, es inevitable que nos vengan a la cabeza numerosas plataformas para la gestión de reservas como *Booking.com* [26], *Airbnb* [23] o *Ruralidays* [42]. Estas aplicaciones suelen estar diseñadas para un público amplio y generalista, orientadas principalmente a grandes operadores o a particulares que gestionan múltiples propiedades.

Estas plataformas, a pesar de ser una gran opción al comenzar un negocio turístico, pueden resultar excesivamente complejas o costosas para pequeños alojamientos que buscan mantener un control directo de sus reservas sin depender de intermediarios o comisiones por cada transacción. Además, limitan la interacción entre emprendedor y cliente, con la intención de evitar puenteos o contactos directos con el alojamiento.

Existe otro tipo de soluciones *software* más ligeras, como sistemas *CMS* (*Content Management Systems*, p.e. *WordPress*), que permiten a usuarios sin excesivos conocimientos técnicos diseñar y lanzar su *web* sin mayores complicaciones, obteniendo resultados muy competitivos.

Sin embargo, el desarrollo de una aplicación *web* personalizada desde cero, centrada en las necesidades concretas del alojamiento en cuestión, permite una mayor flexibilidad, autonomía y control. El proyecto planteado busca cubrir precisamente ese hueco, aportando una herramienta moderna, escalable, segura y optimizada para un escenario real. Al mismo tiempo, sirve tanto como ejercicio práctico de aplicación de conocimientos adquiridos durante el Grado, como reto personal en cuanto a aprendizaje de tecnologías nuevas se refiere.

Planteo a continuación una comparativa entre mi aplicación y algunas de las platafor-

mas y referentes del sector, con la intención de establecer diferencias y similitudes entre ellas:

Características	***Booking.com***	***Airbnb***	***Ruralidays***	**Casa Damajuana WebApp**
Comisiones por reserva	Sí	Sí	Sí	No
Control completo por el propietario	Limitado	Parcial	Limitado	Total
Personalización del sistema	No	No	No	Sí
Código fuente accesible	No	No	No	Sí
Dependencia de terceros	Alta	Alta	Alta	Baja
Coste para el propietario	Medio/Alto	Alto	Medio	Bajo/Nulo
Internacionalización	Alta	Alta	Media	Parcial (Ejemplo implementado)
Escalabilidad y mantenimiento propio	No	No	No	Sí
Uso de *APIs* modernas (*REST*, *Maps*, *PayPal*...)	Sí	Sí	Sí	Sí

Cuadro 3.1: Comparativa entre plataformas de reserva y Casa Damajuana WebApp

Como se puede observar, las plataformas líderes del mercado permiten que cualquier emprendedor publicite su negocio, logrando gestionar una o varias viviendas. Estas soluciones tienen un carácter general, que además ata al propietario a terceros, comisiones abusivas y menos personalización.

En cambio, la aplicación propuesta está totalmente centrada en los requerimientos de un pequeño negocio familiar, cubriendo las necesidades específicas, permitiendo así un enfoque autogestionado, flexible y escalable.

A la vista de la comparación realizada, a pesar de que las plataformas más conocidas son productos maduros y con cierto rodaje en el mercado, la propuesta planteada podría tener buena acogida entre los clientes de la casa rural.

Por lo tanto, se considera interesante desarrollar una aplicación propia, enfocada en el negocio particular comentado, tratando así de disminuir el uso de aplicaciones de terceros.

3.3. Tecnologías y herramientas utilizadas

Para culminar el capítulo actual, procedo a enumerar las diversas tecnologías, herramientas y aplicaciones que me han permitido emprender el camino de desarrollar la aplicación *web*.

Expondré dichas herramientas, describiéndolas brevemente y comentando cuál ha sido su uso en el proyecto.

***Angular* [24]:** es un *framework* para aplicaciones *web* desarrollado en *TypeScript*, de código abierto, mantenido por *Google*. Permite la creación de aplicaciones *web* dinámicas y complejas.

He decido optar por *Angular* debido a que permite construir aplicaciones *web* robustas y escalables. Además, cuenta con una arquitectura robusta, basada en el patrón `Modelo-Vista-Controlador` (MVC), fomenta la reutilización de código, cuenta con documentación detallada y una amplia comunidad de desarrolladores.

Figura 3.3: Logo *Angular*

***TypeScript* [45]:** es un superconjunto de *JavaScript* desarrollado por *Microsoft* que añade tipado estático opcional y otras funciones avanzadas. El código *TypeScript* se compila a *JavaScript* estándar, lo que significa que puede ser ejecutado en cualquier navegador y no afecta al rendimiento de los programas.

Ha sido la base para poder trabajar con el *framework Angular*, desarrollado en *TypeScript*.

Figura 3.4: Logo *TypeScript*

***Node.js* [6]:** es un entorno de ejecución multiplataforma de *JavaScript* de código abierto que permite a los desarrolladores escribir código *JavaScript* en el lado del servidor.

En este caso, su instalación es fundamental para el proyecto para el desarrollo de código *JavaScript*. *Node.js* y su gestor de paquetes, *npm*, son necesarios para instalar y gestionar las dependencias y herramientas de *Angular*. Además, *Angular* se ejecutará sobre *Node.js*.

Figura 3.5: Logo *Node.js*

***npm* (*Node Package Manager*) [37]:** es una herramienta fundamental para el desarrollo con *JavaScript*. Es un gestor de paquetes que facilita la gestión de dependencias, permitiendo instalar, actualizar y eliminar librerías de *JavaScript* de manera sencilla y automatizada.

He empleado *npm* para la instalación, actualización y manejo de paquetes en mi proyecto, usando su gran abanico de comandos.

Figura 3.6: Logo *NPM*

***Visual Studio Code* [11]:** es un editor de código fuente gratuito, desarrollado por *Microsoft*, disponible para cualquier plataforma.

Me he decantado por este *IDE* debido a que *VSCode* facilita la personalización del entorno, el uso de extensiones y brinda un ecosistema perfecto para trabajar con infinidad de lenguajes de programación. Además, al haber trabajado con este entorno en otras ocasiones, me permite partir de una base, haciendo que esté cómodo y pueda centrar mis esfuerzos en entender los nuevos *frameworks* a emplear.

Figura 3.7: Logo *VSCode*

***Spring Boot* [1]:** es un *framework* de *Java* de código abierto que simplifica el desarrollo de aplicaciones y microservicios. Ofrece herramientas y bibliotecas para crear aplicaciones *Spring* de forma más rápida y sencilla, automatizando tareas de configuración y optimizando el proceso de despliegue. Permite mayor efectividad y despliegues más rápidos.

Elegí *Spring Boot* como *framework* para el *backend* con la intención de ampliar y aplicar mis conocimientos sobre *Java*, logrando aproximarme a soluciones usadas en el mercado actual. De esta forma, puedo trabajar con una herramienta que me permita elaborar un *backend* moderno y adaptable.

Figura 3.8: Logo *Spring Boot*

***Spring Tool Suite 4* [10]:** es una herramienta de desarrollo para *Spring*, especialmente diseñada para facilitar el desarrollo de aplicaciones empresariales basadas en *Spring Boot* y *Spring Framework*.

Elegí *STS* debido a que proporciona un entorno robusto y completo para la creación de aplicaciones *Spring* de forma eficiente y productiva.

Figura 3.9: Logo *Spring Tool Suite 4*

***MySQL* [35]:** es uno de los sistemas de gestión de bases de datos relaciones por excelencia. Al ser de código abierto, es ampliamente utilizado para almacenar y gestionar información.

A lo largo de la carrera, he trabajado de forma intensiva con *MySQL*, por lo que me pareció clave plasmar mis conocimientos en este proyecto y aprovecharlos para realizar un manejo correcto de los datos.

Figura 3.10: Logo *MySQL*

***MariaDB* [33]:** es un sistema de gestión de bases de datos relacionales de código abierto, una alternativa a *MySQL* que fue creada por los desarrolladores originales de *MySQL*. Surgió como respuesta a la preocupación de que *MySQL* se comercializara tras su adquisición por *Oracle*.

MariaDB es un *fork* de *MySQL*, y es en realidad el sistema de gestión de bases de datos que incluye *XAMPP*. Por tanto, se mencionará indistintamente a lo largo de la memoria, ya que a pesar de que *XAMPP* permita la activación de un módulo llamado *MySQL*, este es en realidad *MariaDB*.

Figura 3.11: Logo *MariaDB*

***PhpMyAdmin* [40]:** es una aplicación *web* de código abierto que proporciona una interfaz gráfica para gestionar bases de datos *MySQL* y *MariaDB*. Proporciona un entorno gráfico que facilita la creación de consultas, la realización de operaciones y en definitiva la gestión de las bases de datos de tus proyectos.

De nuevo, he decidido mantener esta herramienta tan empleada a lo largo de mi formación para aprovechar mi familiarización con ella, maximizando mi productividad. Se logra acceder a esta herramienta gracias al despliegue del servidor *Apache* desde el panel de control de *XAMPP*.

Figura 3.12: Logo *PhpMyAdmin*

***XAMPP* [46]:** es un paquete de *software* de código abierto que proporciona un entorno de desarrollo de *web* local, fácil de usar y multiplataforma. Al ser sencillo de instalar y multiplataforma, es perfecto para desarrollar aplicaciones *web*.

En mi caso, he empleado sus componentes o módulos *Apache* (servidor *web*) y *MySQL* (sistema de gestión de bases de datos).

Figura 3.13: Logo *XAMPP*

***Postman* [41]:** es una plataforma integral para la gestión de *APIs*, utilizada principalmente para diseñar, probar y documentar *API*.

En mi caso, ha resultado esencial para la realización de pruebas de los *endpoints* de mi proyecto, permitiéndome modificar y refinar cada uno de ellos. Su interfaz sencilla e intuitiva, permite analizar peticiones y respuestas de cara a mejorar el *backend* de la aplicación. Además, me ayuda a tener un resumen organizado por paquetes funcionales de los diversos *endpoint* de mi aplicación.

Figura 3.14: Logo *Postman*

***Git* [5]:** es un sistema de control de versiones distribuido de código abierto que permite a los desarrolladores rastrear los cambios en el código y colaborar en el desarrollo de *software*.

En este caso, *Git* me ha permitido crear ramas en mi proyecto, experimentando mejoras sin poner en riesgo el producto total. Además, es esencial de cara a mantener un historial de versiones y funcionalidades, que me permite avanzar y asegurar poco a poco las mejores, mediante integraciones continuas.

Figura 3.15: Logo *Git*

***SourceTree* [9]:** es un cliente de *Git* (un sistema de control de versiones) con interfaz gráfica de usuario *GUI* (*Graphical User Interface*). Es una herramienta que permite visualizar y gestionar repositorios de *Git* de manera más sencilla, especialmente para aquellos que no están familiarizados con la línea de comandos.

A pesar de comenzar trabajando con control de versiones de forma manual, mediante la línea de comandos, al final decidí utilizar esta interfaz gráfica para ser más productivo. *SourceTree* me ha permitido visualizar mis proyectos, sus ramas y *commits* rápidamente, efectuando operaciones clásicas del control de versiones en segundos.

Figura 3.16: Logo *SourceTree*

***GitHub* [30]:** es una plataforma *web* de desarrollo de *software* que sirve para alojar proyectos utilizando el sistema de control de versiones *Git*.

En mi caso, he utilizado *GitHub* para controlar los subproyectos (*frontend* y *backend*) que conforman mi aplicación *web*. Esta plataforma me permitía la supervisión de los cambios realizados mediante las herramientas de control de versiones mencionadas previamente.

Figura 3.17: Logo *GitHub*

***Trello* [44]:** es una herramienta visual de gestión de proyectos basada en la metodología *Kanban*, que permite organizar tareas y flujos de trabajo de forma colaborativa y sencilla.

Trello me ha permitido crear un tablero para lograr una organización eficiente del proyecto, asignando diversos tipos de tareas, organizándolas en base a su estado y evitando olvidos y despistes, típicos en cualquier proyecto *software*.

Figura 3.18: Logo *Trello*

***LaTeX* [32]:** es un sistema de composición de textos de alta calidad, especialmente utilizado para la creación de documentos científicos y técnicos. Es un lenguaje de marcado, no un procesador de textos como *Word*, que requiere ser compilado para obtener el documento final.

En mi caso, me aventuré a utilizar *LaTeX* para la elaboración de la memoria tras descubrir en el curso impartido en la universidad el gran poder que tenía. Me ha facilitado enormemente la gestión de índices, tablas, apéndices y demás elementos difíciles de mantener actualizados en documentación tan extensa como es la de un Trabajo Fin de Grado.

Figura 3.19: Logo *LaTeX*

***Overleaf* [39]:** es un editor *LaTeX* en línea que permite crear, editar y compartir documentos de forma colaborativa. Es una herramienta *web* que no requiere instalación ni actualizaciones y funciona en cualquier dispositivo con conexión a *Internet*.

En este caso, *Overleaf* fue mi elección debido a que me ahorraba realizar pesadas instalaciones de entornos *LaTeX*, permitiendo además una sincronización mayor con los tutores, al poder compartir con ellos el documento en tiempo real. Sin duda, las facilidades que ofrece para ser eficiente en la creación de textos *LaTeX* ha hecho que se haya convertido en una herramienta esencial para la elaboración de la documentación de mi proyecto.

Figura 3.20: Logo *Overleaf*

***Outlook* [38]:** es un programa de *Microsoft* que se usa para gestionar el correo electrónico, el calendario, los contactos y las tareas. Es una aplicación versátil disponible para varios dispositivos y plataformas, como *PC*, *Mac*, dispositivos móviles y *web*.

Esta aplicación, utilizada por todo miembro de la universidad, me ha permitido comunicarme con los diversos roles que mencionaba al hablar de la metodología: tutores, comunidad, etc.

Figura 3.21: Logo *Outlook*

Bloc de notas: es un editor de texto sencillo que viene incluido con todas las versiones de *Windows*.

Esta sencilla aplicación me ha servido en numerosas ocasiones para plantear dudas, anotaciones rápidas o ideas.

Figura 3.22: Logo Bloc de Notas

***Mozilla Firefox* [34]:** es un navegador *web* de código abierto, gratuito y multiplataforma, desarrollado por *Mozilla Corporation*.

En mi caso, lo he empleado principalmente para probar el desarrollo y los avances del proyecto, debido a sus características. Nótese que he utilizado otros navegadores como *Chrome* o *Edge* para probar funcionalidades, realizar transacciones paralelas y comprobar el correcto funcionamiento de la aplicación.

Figura 3.23: Logo *Mozilla Firefox*

***PayPal* [21]:** es un sistema de pago en línea que permite realizar transacciones electrónicas de dinero de forma segura y eficiente. En concreto empleé *PayPal Developer* (Portal de Desarrollador de *PayPal*), una plataforma donde los desarrolladores pueden encontrar guías, documentación y recursos para integrar las funcionalidades de pago de *PayPal* en sus aplicaciones, sitios *web* y plataformas.

Gracias a los entornos de pruebas que facilitan (*sandbox*), así como el amplio catálogo de *APIs* con el que cuentan, he podido integrar este sistema de pago para la realización de reservas en mi proyecto.

Figura 3.24: Logo *PayPal Developer*

***Google Cloud Console* [18]:** permite gestionar y supervisar los recursos de *Google Cloud Platform* (*GCP*). Es una herramienta esencial para la administración de proyectos, servicios y aplicaciones en la nube.

En mi caso, me permite acceder a diversas *APIs* de *Google*, facilitando documentación que permita la integración de estas con tu proyecto.

Figura 3.25: Logo *Google Cloud Console*

***StarUML* [43]:** es una herramienta de *software* para el modelado de sistemas que utiliza el Lenguaje Unificado de Modelado (*UML*) y otras notaciones de modelado.

Con esta aplicación puedo desarrollar la gran mayoría de diagramas y esquemas necesarios para documentar el proyecto.

Figura 3.26: Logo *StarUML*

***Canva* [28]:** es una plataforma *online* de diseño gráfico que permite crear, compartir e imprimir diseños de manera fácil y rápida. Sirve para crear una amplia variedad de elementos visuales, desde diseños para redes sociales y presentaciones hasta infografías y documentos, todo con un editor intuitivo y fácil de usar.

En mi proyecto he usado esta herramienta para el diseño de ciertos elementos gráficos que aporten un toque profesional y cuidado al diseño de la aplicación. También se ha utilizado para elaborar ciertos diagramas que ilustran aspectos explicados en la documentación.

Figura 3.27: Logo *Canva*

***Excalidraw* [29]:** es una herramienta de dibujo digital de código abierto que se caracteriza por su interfaz sencilla y su estilo "hecho a mano", ideal para crear diagramas, bocetos y visualizaciones de manera colaborativa. Es gratuita, accesible en línea y permite a varios usuarios trabajar juntos en tiempo real en el mismo proyecto.

En mi proyecto he usado esta herramienta para la elaboración de diversos diagramas y esquemas.

Figura 3.28: Logo *Excalidraw*

***Draw.io* [47]:** es una herramienta gratuita de diagramación en línea que permite crear diagramas de flujo, mapas mentales, diagramas *UML*, esquemas de red, maquetas y muchos otros tipos de diagramas.

En mi proyecto he usado esta herramienta para la elaboración de diversos *wireframes* o bocetos para el diseño temprano de la interfaz.

Figura 3.29: Logo *Draw.io*

Todas las herramientas, aplicaciones y soluciones tecnológicas mencionadas han constituido parte de mi abanico de recursos empleados para construir el proyecto que presento en esta memoria, facilitando las labores de escritura de código, diseño de elementos, conexión con otros sistemas y redacción de la documentación.

Parte II

Desarrollo de la Aplicación

Capítulo 4

Análisis

El análisis de un sistema *software* es un proceso esencial que permite la recolección y entendimiento de ciertos requisitos y funcionalidades, permitiendo definir cómo interactuarán los usuarios con el sistema y qué debe hacer el sistema para satisfacer sus necesidades.

En este capítulo se expondrán los requisitos ligados a los diversos usuarios contemplados en la aplicación, así como los requisitos del sistema, que explican qué funcionalidades llevará a cabo este último y qué características debe tener.

4.1. Requisitos de usuario

Los requisitos de usuario recogen las funcionalidades y expectativas desde la perspectiva de los distintos actores que interactúan con la aplicación. Estas necesidades se traducen en interacciones clave con el sistema, recogidas mediante casos de uso.

4.1.1. Actores del sistema

Como bien sabemos, los casos de uso describen las interacciones que acontecen entre ciertos actores y el sistema. El rol de actor lo puede representar una persona o, también, otro sistema *software* que interactúa con el actual en el ámbito del caso de uso.

Es importante remarcar que el mismo usuario puede interactuar con el sistema desempeñando diferentes roles de actor, como en el caso del administrador.

Paso a mostrar los diversos actores identificados en mi sistema:

- **Usuario anónimo:** usuario no autenticado que accede a funcionalidades básicas del sistema, como visualizar los espacios disponibles, realizar una reserva o enviar un mensaje de contacto. Cabe destacar que, como se comenta a lo largo de la memoria, se espera que los usuarios anónimos puedan acceder a la gran mayoría de las funcionalidades de la aplicación, con la finalidad de lograr captar a personas que se muestren reticentes en cuanto al registro se refiere.

- **Usuario registrado:** individuo que ha creado una cuenta en la aplicación y puede realizar operaciones adicionales, como consultar el estado de sus reservas desde el Panel de Usuario.

- **Administrador:** usuario con privilegios elevados que puede gestionar usuarios, reservas y espacios del sistema. Además, tiene acceso a funciones de mantenimiento, actualización y eliminación de registros.

- ***PayPal*:** plataforma externa utilizada para gestionar el proceso de pago. Se considera un actor ya que interactúa con el sistema enviando respuestas que determinan el estado de las reservas. En este caso, *PayPal* constituye el único actor externo no humano de la aplicación.

Nota: APIs como Google Maps JavaScript o Google Places, utilizadas únicamente para mostrar información en el frontend, no se consideran actores del sistema ya que no generan ni modifican flujos de control en la lógica de negocio.

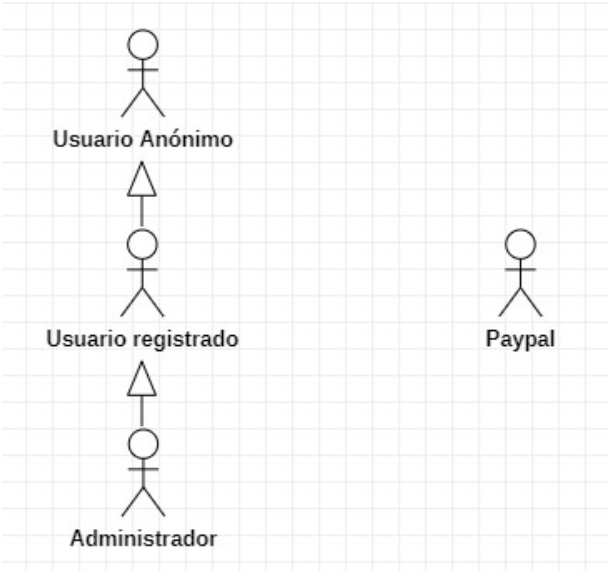

Figura 4.1: Actores del sistema

Como se puede comprobar en el diagrama presentado, un usuario registrado podrá realizar todas las funciones que desempeña un usuario anónimo, agregando una serie de acciones extra. Por su parte, el administrador será una especie de superusuario, con acceso total a todas las funcionalidades de usuarios anónimos y registrados.

Una vez identificados correctamente los diversos actores que interactuarán con el sistema, podemos pasar a realizar un análisis de los casos de uso.

4.1.2. Casos de uso

Un caso de uso es la descripción de cómo un usuario interactúa con un sistema o producto para lograr un objetivo específico. Es una forma de visualizar y documentar las funcionalidades del sistema desde el punto de vista del usuario.

Al haber especificado previamente los actores que interactúan con el sistema, pasamos a enumerar y describir los casos de uso principales de cada actor:

- **CU01 – Registro de usuario**
  - **Actor:** Usuario anónimo
  - **Descripción:** Permite a un nuevo usuario registrarse mediante un formulario, proporcionando datos como nombre, correo electrónico y contraseña. Si los datos son válidos, se crea el usuario y se redirige al *login*.
- **CU02 – Inicio de sesión**
  - **Actor:** Usuario anónimo
  - **Descripción:** El usuario proporciona sus credenciales (correo y contraseña), las cuales se validan. Si son correctas, se inicia sesión y se genera una *cookie* *JWT* segura.
- **CU03 – Explorar espacios**
  - **Actor:** Todos los usuarios
  - **Descripción:** Permite ver una lista filtrable de los espacios disponibles en la casa, mostrando información como nombre y tipo, así como acceder a su detalle con información ampliada.
- **CU04 – Realizar reserva**
  - **Actor:** Todos los usuarios
  - **Descripción:** Permite seleccionar fechas de reserva, completar los datos de contacto y efectuar el pago a través de *PayPal*. Si es exitoso, se registra la reserva.
- **CU05 – Cancelar reserva**
  - **Actor:** Usuario registrado / Administrador
  - **Descripción:** Permite cancelar una reserva previamente creada, devolviendo el importe abonado a través de *PayPal*.
- **CU06 – Enviar mensaje de contacto**
  - **Actor:** Todos los usuarios

  - **Descripción:** Permite enviar un mensaje al equipo gestor mediante un formulario con *CAPTCHA*. El mensaje se envía por correo electrónico a la bandeja de correo de la casa.

- **CU07 – Ver reseñas de *Google***
  - **Actor:** Todos los usuarios
  - **Descripción:** Muestra las reseñas públicas extraídas desde la *API* de *Google Places*, sin interacción directa con el sistema interno.

- **CU08 – Interacción con el mapa**
  - **Actor:** Todos los usuarios
  - **Descripción:** Permite al usuario consultar el mapa interactivo mediante *Google Maps*, hacer *zoom*, moverse, cambiar de vista y utilizar la vista de calle.

- **CU09 – Ver información general de la casa**
  - **Actor:** Todos los usuarios
  - **Descripción:** Permite visualizar información institucional o promocional de la casa desde las secciones "Página Principal" y "Más", como historia, reglas, fotos, etc.

- **CU10 – Consultar actividades en la zona**
  - **Actor:** Todos los usuarios
  - **Descripción:** Permite consultar actividades y experiencias cercanas, y filtrarlas por categoría desde la sección "Más".

- **CU11 – Ver resumen de la reserva**
  - **Actor:** Todos los usuarios
  - **Descripción:** Permite consultar el resumen de la reserva tras completar su proceso de creación de forma satisfactoria, permitiendo imprimir o descargar el *PDF* de los detalles.

- **CU12 – Consultar historial de reservas**
  - **Actor:** Usuario registrado / Administrador
  - **Descripción:** Permite al usuario ver sus reservas desde el panel de usuario (el administrador puede ver solo las propias desde aquí), así como exportar en diversos formatos las reservas filtradas según diversos criterios.

- **CU13 – Consultar/modificar información de la cuenta**
  - **Actor:** Usuario registrado / Administrador

  - **Descripción:** Permite al usuario ver los datos de la cuenta (nombre, apellidos y correo), dándole la opción de modificarlos y de cambiar su contraseña.

- **CU14 – Borrar la cuenta**
  - **Actor:** Usuario registrado / Administrador
  - **Descripción:** Permite al usuario eliminar su propia cuenta, avisándole de que es una acción irreversible.

- **CU15 – Gestión de usuarios**
  - **Actor:** Administrador
  - **Descripción:** Lista, filtra, exporta y elimina usuarios registrados desde el panel de administrador.

- **CU16 – Gestión de reservas**
  - **Actor:** Administrador
  - **Descripción:** Permite al administrador consultar, filtrar, exportar y cancelar todas las reservas del sistema desde el panel de administrador.

- **CU17 – Cierre de sesión**
  - **Actor:** Usuario registrado / Administrador
  - **Descripción:** Elimina la *cookie* de sesión, cerrando la sesión activa y redirigiendo al inicio.

Con la intención de visualizar de forma sencilla los casos de uso identificados y los actores que los llevan a cabo, elaboramos una tabla que ilustre todos ellos, expuesta en la figura 4.1.

Contando con la definición de los casos de uso y mostrando su organización en base a los actores que los realizan, nos aventuramos a realizar un diagrama ilustrativo de todas estas interacciones.

Para la representación del diagrama, emplearé *UML* (*Unified Modeling Language*) un lenguaje de modelado gráfico que se utiliza para representar visualmente sistemas complejos, especialmente en el desarrollo de *software*.

Un diagrama de casos de uso *UML* es una representación gráfica que muestra cómo un sistema interactúa con sus usuarios o actores. Se utiliza para modelar el comportamiento del sistema y las relaciones entre los usuarios y las funciones del sistema.

En mi caso, distinguiré cuatro tipos de elementos en el diagrama:

- **Actores:** entidades externas al sistema que interactúan con él, como usuarios, sistemas externos o dispositivos.

- **Casos de uso:** representan las funcionalidades o acciones que el sistema puede realizar en respuesta a las interacciones de los actores.

Caso de Uso	**Usuario Anónimo**	**Usuario Registrado**	**Administrador**
CU01 – Registro de usuario	X		
CU02 – Inicio de sesión	X		
CU03 – Explorar espacios	X	X	X
CU04 – Realizar reserva	X	X	X
CU05 – Cancelar reserva		X	X
CU06 – Enviar mensaje de contacto	X	X	X
CU07 – Ver reseñas de *Google*	X	X	X
CU08 – Interacción con el mapa	X	X	X
CU09 – Ver información general de la casa	X	X	X
CU10 – Consultar actividades en la zona	X	X	X
CU11 – Ver resumen de la reserva	X	X	X
CU12 – Consultar historial de reservas		X	X
CU13 – Consultar/modificar información de la cuenta		X	X
CU14 – Borrar la cuenta		X	X
CU15 – Gestión de usuarios			X
CU16 – Gestión de reservas			X
CU17 – Cierre de sesión		X	X

Cuadro 4.1: Resumen de casos de uso por tipo de actor

- **Sistema:** se representa como un rectángulo que contiene los casos de uso y delimita el alcance del sistema que se está modelando.

- **Relaciones:** se utilizan para mostrar las conexiones entre los actores y los casos de uso, como la comunicación o la dependencia entre ellos. En mi diagrama, habrá relaciones simples (entre actores y casos de uso), relaciones de tipo *extend* (que representan flujos alternativos) y relaciones de tipo *include* (que representan pasos o flujo obligatorio).

Por simplicidad, habrá relaciones que no se modelarán debido a que se dan por hecho, como por ejemplo que tras cerrar sesión se puede iniciar sesión.

Muestro el diagrama de casos de uso en la figura 4.2, así como en la figura B.2 con la intención de mejorar su visualización.

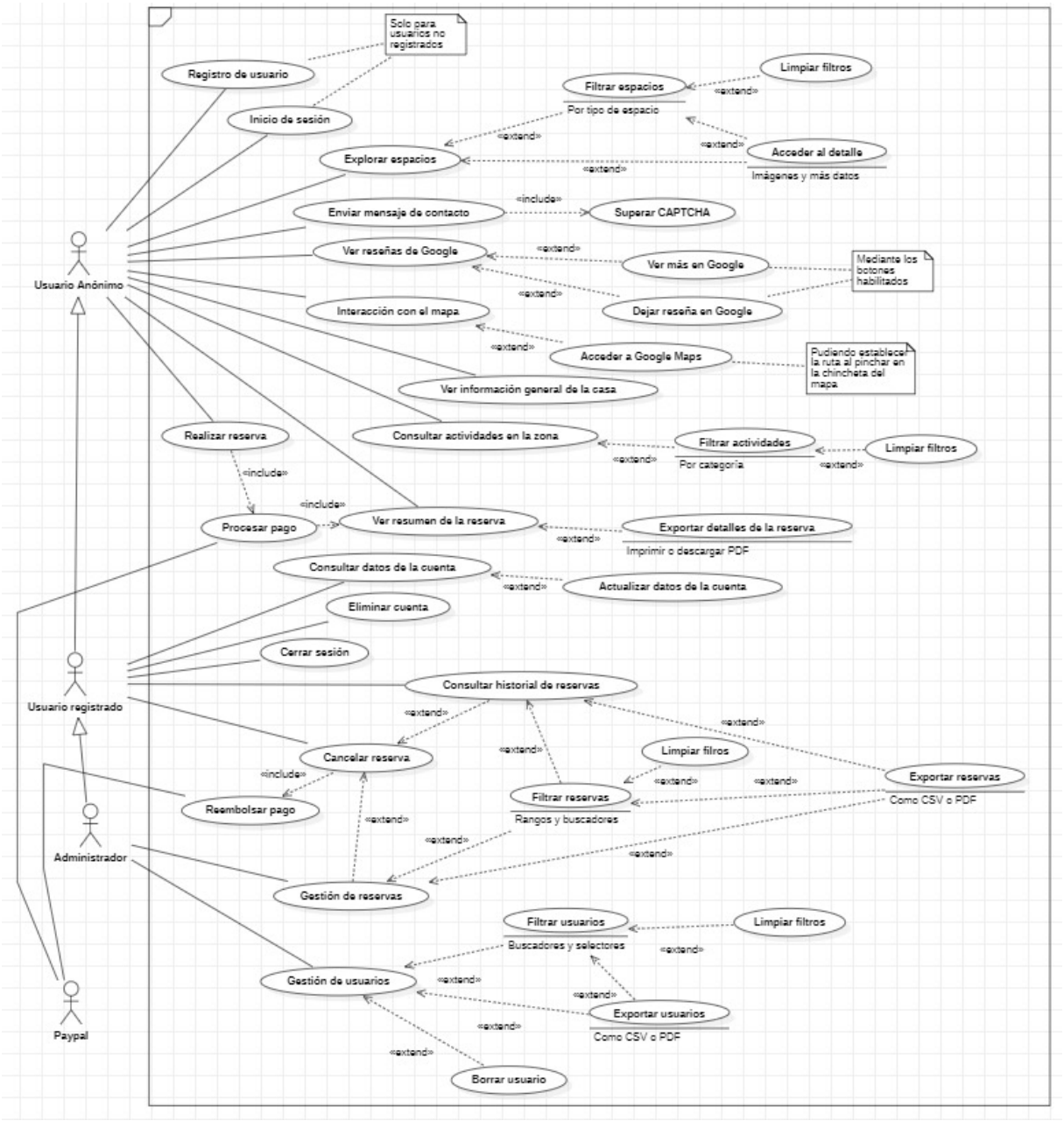

Figura 4.2: Diagrama de Casos de Uso del sistema

4.1.3. Historias de usuario

A pesar de haber realizado una especificación de casos de uso, ya que a mi parecer permiten identificar las funcionalidades de la aplicación desde el punto de vista del usuario de manera sencilla, no podemos olvidar que al aplicar una metodología ágil es esencial contar con las famosas historias de usuario.

Las historias de usuario son una descripción breve, informal y centrada en el usuario de una funcionalidad de *software*. Se escribe desde la perspectiva del usuario final y se enfoca en el valor que esa funcionalidad aporta. Gracias a la inclusión de las historias de usuario, podremos no solo compararlas con los casos de uso y complementar el análisis, sino entender de forma directa las necesidades de los usuarios.

- **HU01:** Como usuario anónimo, quiero registrarme en la plataforma para poder gestionar mis datos y reservas.
- **HU02:** Como usuario anónimo, quiero iniciar sesión con mis credenciales para acceder a funcionalidades exclusivas de usuarios registrados.
- **HU03:** Como usuario, quiero explorar los distintos espacios de la casa para decidir cuál se adapta mejor a mis necesidades y conocer el encanto de esta.
- **HU04:** Como usuario, quiero poder reservar mediante un formulario de forma sencilla y pagar con *PayPal* rápidamente.
- **HU05:** Como usuario registrado, quiero poder cancelar una reserva activa para recuperar el dinero si ya no puedo asistir de forma automatizada y rápida.
- **HU06:** Como usuario, quiero enviar un mensaje de contacto mediante un formulario para comunicarme con el equipo gestor de la casa.
- **HU07:** Como usuario, quiero ver reseñas de otros visitantes desde *Google* para conocer su experiencia previa.
- **HU08:** Como usuario, quiero interactuar con un mapa de la ubicación de la casa para saber como llegar y explorar los alrededores.
- **HU09:** Como usuario, quiero ver información general sobre la casa para conocer sus instalaciones, historia y normativa.
- **HU10:** Como usuario, quiero consultar actividades disponibles en la zona para planificar mi estancia con opciones de ocio.
- **HU11:** Como usuario, quiero acceder al resumen de mi reserva y descargarlo o imprimirlo en *PDF* para tener un comprobante de la misma.
- **HU12:** Como usuario registrado, quiero consultar mi historial de reservas para revisar mis visitas anteriores o próximas y optar a su cancelación.

- **HU13:** Como usuario registrado, quiero consultar y/o modificar mis datos personales desde mi perfil para mantener mi información actualizada.
- **HU14:** Como usuario registrado, quiero optar a eliminar mi cuenta para pasar a ser un usuario anónimo.
- **HU15:** Como administrador, quiero gestionar los usuarios registrados para mantener la base de datos actualizada y segura.
- **HU16:** Como administrador, quiero gestionar todas las reservas para tener control sobre el calendario y detectar incidencias.
- **HU17:** Como usuario registrado, quiero cerrar sesión cuando finalice mi uso para proteger mis datos personales.

Como se puede comprobar, las historias de usuario y los casos de uso se alinean, marcando las necesidades de los diversos actores del sistema. Mientras que los casos de uso detallan más la funcionalidad esperada, las historias de usuario permiten comprender para qué necesita el usuario esas herramientas.

4.2. Requisitos del sistema

Los requisitos del sistema según el modelado del *software*, describen las funcionalidades y restricciones que un sistema debe cumplir. En esencia, son la especificación de lo que el sistema debe hacer y cómo debe comportarse. Esto incluye tanto los requisitos funcionales (qué debe hacer el sistema) como los no funcionales (cómo debe hacerlo a nivel de, por ejemplo, rendimiento, seguridad, etc.).

4.2.1. Requisitos funcionales

Los requisitos funcionales son especificaciones que detallan cómo debe comportarse un sistema, *software*, producto o servicio para cumplir con los objetivos establecidos y satisfacer las necesidades de los usuarios o del negocio. En otras palabras, definen lo que el sistema debe hacer para funcionar como se espera.

Se diferencian de los requisitos no funcionales, que se abordarán más adelante en esta sección, en que los requisitos funcionales atienden al comportamiento del sistema, mientras que los no funcionales hacen alusión a características de este.

Se presenta a continuación una tabla que recoge los diversos requisitos funcionales identificados:

ID	Requisitos funcionales
RF01	El sistema debe permitir al usuario iniciar sesión con correo y contraseña.
RF02	El sistema debe validar las credenciales y controlar el acceso si son correctas.

ID	Requisitos funcionales
RF03	El sistema debe registrar nuevos usuarios mediante un formulario con validación.
RF04	El sistema debe redirigir a la pantalla principal o a la de registro tras un registro exitoso o un inicio de sesión, respectivamente.
RF05	El sistema debe adaptar el menú de navegación dinámicamente según el rol del usuario.
RF06	El sistema debe mostrar un carrusel de fotos e información general en la pantalla principal.
RF07	El sistema debe mostrar una tabla de habitaciones con filtros por tipo y un botón de detalles.
RF08	El sistema debe permitir acceder al detalle de una habitación al pulsar el botón correspondiente, mostrando información ampliada como imágenes y descripciones.
RF09	El sistema debe mostrar una pantalla de error si no se encuentra la habitación.
RF10	El sistema debe permitir seleccionar fechas para una reserva mediante un calendario.
RF11	El sistema debe mostrar en distintos colores los días pasados, el día actual, las fechas reservadas, las fechas seleccionadas y las fechas disponibles.
RF12	El sistema debe mostrar un formulario de datos personales tras validar las fechas seleccionadas.
RF13	El sistema debe rellenar los datos del usuario en el formulario si este estuviera registrado.
RF14	El sistema debe calcular el precio total de la reserva aplicando reglas de tarifa y descuentos.
RF15	El sistema debe permitir realizar pagos con *PayPal* mediante redirección (botón *PayPal*) o formulario (botón tarjeta).
RF16	El sistema debe validar los datos del pago con la *API* de *PayPal* antes de registrar la reserva.
RF17	El sistema debe registrar una reserva en base de datos tras un pago válido.
RF18	El sistema debe mostrar un resumen de la reserva con opción a imprimir o descargar en *PDF*.
RF19	El sistema debe mostrar reseñas públicas desde la *API* de *Google Places*.
RF20	El sistema debe permitir acceder a los enlaces de *Google* para ver y dejar reseñas.
RF21	El sistema debe mostrar la ubicación de la casa en un mapa interactivo mediante *Google Maps*.
RF22	El sistema debe permitir enviar mensajes al equipo mediante formulario con *CAPTCHA*.
RF23	El sistema debe permitir consultar actividades turísticas filtrables por categoría.

ID	Requisitos funcionales
RF24	El sistema debe permitir a un usuario *logueado* ver sus reservas con filtros avanzados.
RF25	El sistema debe permitir a un usuario *logueado* cancelar una reserva y devolver el importe mediante *PayPal*.
RF26	El sistema debe permitir exportar reservas a *CSV* o *PDF* desde el panel del usuario.
RF27	El sistema debe permitir a los administradores ver todas las reservas con los mismos filtros.
RF28	El sistema debe permitir a los administradores ver, filtrar y eliminar usuarios registrados.
RF29	El sistema debe permitir exportar reservas o usuarios a *CSV* y *PDF* desde el panel de administrador.
RF30	El sistema debe permitir cerrar sesión, eliminando la *cookie JWT* y redirigiendo al inicio.
RF31	El sistema debe evitar reservas simultáneas para las mismas fechas, mostrando un mensaje de error al usuario afectado.
RF32	El sistema debe validar la disponibilidad en tiempo real justo antes de confirmar la reserva y procesar el pago.
RF33	El sistema debe mostrar mensajes de error claros ante fallos de red, caída del sistema o respuestas inválidas de la *API* de *PayPal*.
RF34	El sistema debe notificar si no se encuentran resultados al aplicar filtros de búsqueda o reservas.
RF35	El sistema debe garantizar que las operaciones críticas, como pago y creación de reserva, sean transaccionales: si una parte falla, no se debe persistir nada.

Cuadro 4.2: Requisitos funcionales del sistema

Gracias a estos requisitos funcionales, podemos resumir la funcionalidad del sistema, el comportamiento que este va a tener frente a los eventos que puedan desencadenarse. Para corroborar que se basan en los casos de uso planteados previamente, se expone una matriz de trazabilidad que relaciona los casos de uso con los requisitos funcionales.

RF\CU	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
1		X															
2		X															
3	X																
4	X	X															
5																	
6									X								
7			X														
8			X														
9			X														

RF\CU	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
10				X													
11				X													
12				X													
13				X									X				
14				X													
15				X													
16				X													
17				X													
18				X							X						
19							X										
20							X										
21								X									
22						X											
23										X							
24												X					
25					X												
26												X					
27																X	
28															X		
29															X	X	
30																	X
31				X													
32				X													
33	X	X	X	X	X	X								X			
34			X							X		X			X	X	
35				X													

Cuadro 4.3: Matriz de trazabilidad entre Requisitos Funcionales y Casos de Uso

Esta matriz permite verificar que existe cierta coherencia entre los diversos tipos de requisitos que modelan la interacción entre usuarios y el sistema.

4.2.2. Requisitos de información

Los requisitos de información definen los datos que el sistema debe almacenar, procesar y gestionar para cumplir con sus funcionalidades. Estos requisitos permiten identificar las entidades principales del sistema, las relaciones entre ellas y las necesidades de almacenamiento y consulta.

Los requisitos de información son clave para el diseño del modelo de datos y para garantizar que el sistema pueda responder adecuadamente a las acciones del usuario, soportar reportes, y cumplir con requisitos legales, de trazabilidad o de análisis de negocio.

A continuación, se enumeran los requisitos de información identificados para esta aplicación:

ID	Requisitos de información
RI01	El sistema debe almacenar los datos de los usuarios, incluyendo nombre, apellidos, correo electrónico, contraseña (*hasheada*) y rol.
RI02	El sistema debe almacenar información de las habitaciones: identificador, nombre, descripción, tipo y capacidad.
RI03	El sistema debe almacenar las reservas realizadas, incluyendo *ID* de la reserva (autogenerado), fechas de inicio y fin, correo electrónico y nombre del comprador, mensaje adicional (de forma opcional), *ID* de usuario asociado (si se hubiera efectuado la reserva desde una cuenta), estado de la reserva y precio final.
RI04	El sistema debe guardar un historial de pagos realizados, agregando a cada reserva el *ID* de captura del pago de *PayPal*, que permitirá la devolución del importe al cancelar la reserva.
RI05	El sistema debe generar un *ID* único para cada usuario.
RI06	El sistema debe generar un *ID* único para cada espacio.
RI07	El sistema debe generar un *ID* único para cada reserva.

Cuadro 4.4: Requisitos de información del sistema

Cabe destacar que los requisitos de información no se limitan únicamente a los datos almacenados de forma persistente en el sistema (por ejemplo, en una base de datos relacional), sino que también incluyen aquellos datos que el sistema debe procesar o manejar temporalmente para cumplir con su funcionalidad. Esto incluye, por ejemplo, los datos introducidos en formularios que se envían por correo, o la información estática que se utiliza para construir las actividades recomendadas. No se incluirán en la tabla de requisitos de información, pero explicaré a continuación cómo y por qué se han definido de manera especial algunas entidades de la aplicación.

Para no engordar el proyecto agregando inserciones en base de datos de elementos como actividades en la zona, imágenes, reseñas y mensajes propios, se ha optado por su gestión a nivel estático o mediante servicios externos. De esta forma, se delega la gestión de reseñas y mensajes en servicios externos especializados, y se gestionan los elementos gráficos de forma estática, evitando agregar operaciones *CRUD a priori* innecesarias.

Por tanto, las actividades recomendadas se definen en base a una interfaz existente en el módulo `more` del proyecto *Angular*, y se crean una serie de ejemplos siguiendo esta plantilla en un archivo de *TypeScript* del mismo módulo. Por su parte, los mensajes de contacto vuelven a crearse siguiendo una interfaz definida, delegando su almacenamiento a *Gmail*, servicio que el *backend* utiliza para enviar correos. Las imágenes de los espacios y actividades, lejos de almacenarse en la base de datos, están disponibles en el directorio habilitado en *Angular* para la carga de contenido estático. Por último, las reseñas y mapas dependen de los servicios de *Google*, en lugar de gestionarse de forma local.

En definitiva, los requisitos de información expuestos permiten definir la estructura de la base de datos, así como los mecanismos de acceso y consulta necesarios para asegurar el correcto funcionamiento del sistema.

4.2.3. Requisitos no funcionales

Por su parte, los requisitos no funcionales son características y restricciones que describen cómo debe funcionar un sistema, más allá de sus funciones específicas. En lugar de definir qué hace el sistema, especifican cómo debe hacerlo.

Hay requisitos no funcionales de muchos tipos. Se explican los más relevantes en la tabla mostrada a continuación:

Tipo	**Descripción**
Disponibilidad	Se refiere al tiempo durante el cual el sistema está operativo y accesible para los usuarios.
Rendimiento	Evalúa la velocidad de respuesta del sistema y el uso eficiente de recursos.
Usabilidad	Evalúa lo fácil e intuitivo que resulta el sistema para los usuarios.
Seguridad	Abarca protección de datos, cifrado, control de accesos y resistencia ante ataques.
Mantenibilidad	Se refiere a la facilidad con la que se puede actualizar, corregir y mejorar el sistema.
Escalabilidad	Evalúa la capacidad del sistema para manejar un crecimiento en la carga o funcionalidad.
Compatibilidad	Indica si el sistema funciona correctamente en distintos navegadores, dispositivos o plataformas.
Recuperación ante fallos	Evalúa la capacidad del sistema para recuperarse rápidamente después de errores o interrupciones.
Legalidad	Se asegura de que el sistema cumpla con normativas legales y reglamentarias, como *GDPR* (*General Data Protection Regulation*).
Calidad / *Testing*	Establece requisitos para pruebas automatizadas, revisiones y aseguramiento de calidad del *software*.

Cuadro 4.5: Tipos principales de requisitos no funcionales

Existen muchos más tipos de requisitos no funcionales, como los de robustez o los de reusabilidad. Además, hay tipos que son muy similares, e incluso se denominan de formas diferentes, aunque en esencia son iguales. Por tanto, la tabla anterior no recoge todo el abanico de posibilidades en lo relativo a los tipos de requisitos no funcionales, sino que explica algunos de los tipos más relevantes.

Una vez comprendidos algunos de los diversos tipos de requisitos no funcionales que existen, paso a especificar los RNF que se ajustan a las necesidades de la aplicación, definiendo así características que esta poseerá.

ID	Requisito no funcional	Tipo
RNF01	El sistema debe estar disponible 24/7 con una tasa de disponibilidad mayor al 99 %.	Disponibilidad
RNF02	Las páginas deben cargarse en menos de 2 segundos en condiciones normales.	Rendimiento
RNF03	El sistema debe ser accesible desde dispositivos móviles.	Usabilidad
RNF04	Las comunicaciones deben estar cifradas mediante *HTTPS*.	Seguridad
RNF05	Las contraseñas deben almacenarse de forma segura mediante *hashing* (*BCrypt*).	Seguridad
RNF06	El sistema debe ser intuitivo para usuarios sin conocimientos técnicos.	Usabilidad
RNF07	Se deben seguir buenas prácticas de desarrollo (estructura modular, uso de *APIs REST*, etc.).	Mantenibilidad
RNF08	Se debe realizar validación tanto en *frontend* como en *backend*.	Seguridad
RNF09	El sistema debe permitir la escalabilidad para futuras ampliaciones.	Escalabilidad
RNF10	Las respuestas de errores deben ser claras y no exponer información sensible.	Seguridad
RNF11	El sistema debe ser compatible con los navegadores modernos más usados (*Chrome*, *Firefox*, *Safari*, *Edge*).	Compatibilidad
RNF12	El sistema debe estar documentado para facilitar su mantenimiento y despliegue.	Mantenibilidad
RNF13	El tiempo de recuperación ante fallos críticos no debe superar los 15 minutos.	Recuperación ante fallos
RNF14	El sistema debe cumplir con la normativa vigente de protección de datos.	Legalidad
RNF15	El sistema debe permitir realizar pruebas automatizadas para asegurar la calidad del *software*.	Calidad/ *Testing*
RNF16	En la documentación del sistema se deben incluir manuales de instalación y de usuario que faciliten el uso y mantenimiento de la aplicación.	Usabilidad
RNF17	El sistema protegerá contra la inserción, modificación o borrado no autorizado de datos.	Integridad
RNF18	El sistema registrará y cobrará solo una reserva si llegan dos para los mismos días de forma paralela, notificando a ambos usuarios del éxito/fracaso de la transacción.	Robustez

ID	Requisito no funcional	Tipo
RNF19	El sistema empleará numerosos componentes reutilizables a lo largo de la aplicación, disminuyendo la complejidad y extensión del código fuente.	Reusabilidad
RNF20	El sistema garantizará la consistencia de los datos en caso de fallos del sistema.	Mantenibilidad
RNF21	La aplicación se desarrollará mediante las herramientas y soluciones tecnológicas especificadas en la sección correspondiente de la documentación.	Implementación

Cuadro 4.6: Requisitos no funcionales del sistema

Estos son los requisitos no funcionales que considero más representativos del sistema, y que ayudan a mantener unos estándares altos en todos los campos que abarcan.

Capítulo 5

Diseño

En este capítulo se detallarán las decisiones técnicas adoptadas durante la fase de diseño del sistema. En él se presentan tanto la arquitectura lógica como la física de la aplicación, reflejando cómo se organiza internamente el sistema, cómo se comunican sus componentes y cómo se distribuyen en el entorno de ejecución. También se representa el modelo conceptual y lógico, mostrando las entidades del dominio y las relaciones, así como el diccionario de datos, exponiendo al detalle características de los datos. Por último, se aborda el diseño de la interfaz, mostrando ciertas pantallas y componentes, aspectos relativos al diseño, etc.

El diseño se ha orientado hacia la modularidad, la separación de responsabilidades y la escalabilidad, buscando facilitar tanto el mantenimiento como la futura ampliación del proyecto. A lo largo de este capítulo se podrá visualizar mediante diversos diagramas la estructura de cada capa que compone la aplicación.

5.1. Arquitectura lógica

La arquitectura lógica de la aplicación define la estructura y organización conceptual de sus componentes y la forma en que interactúan entre sí, sin entrar en detalles específicos de implementación física. Es el plano que define cómo se divide el *software* en módulos, sus responsabilidades y cómo se comunican para lograr los objetivos del sistema.

Para comprender los módulos que integran el sistema, se presentará un diagrama de arquitectura lógica, en el que se podrá apreciar la distribución basada en capas que se ha seguido. Esta organización aplica el patrón clásico de separación entre presentación, lógica de negocio y acceso a datos, que permite desacoplar el *frontend* del *backend*.

Además, se han empleado principios como el control de acceso por roles, la gestión de sesiones mediante *tokens JWT* y la integración con servicios externos como *PayPal* o *Google Maps*, los cuales se detallan en la sección 6.4.

Sin más dilación, se muestra la arquitectura lógica sin detallar del sistema, que explicaré brevemente para mejorar su comprensión.

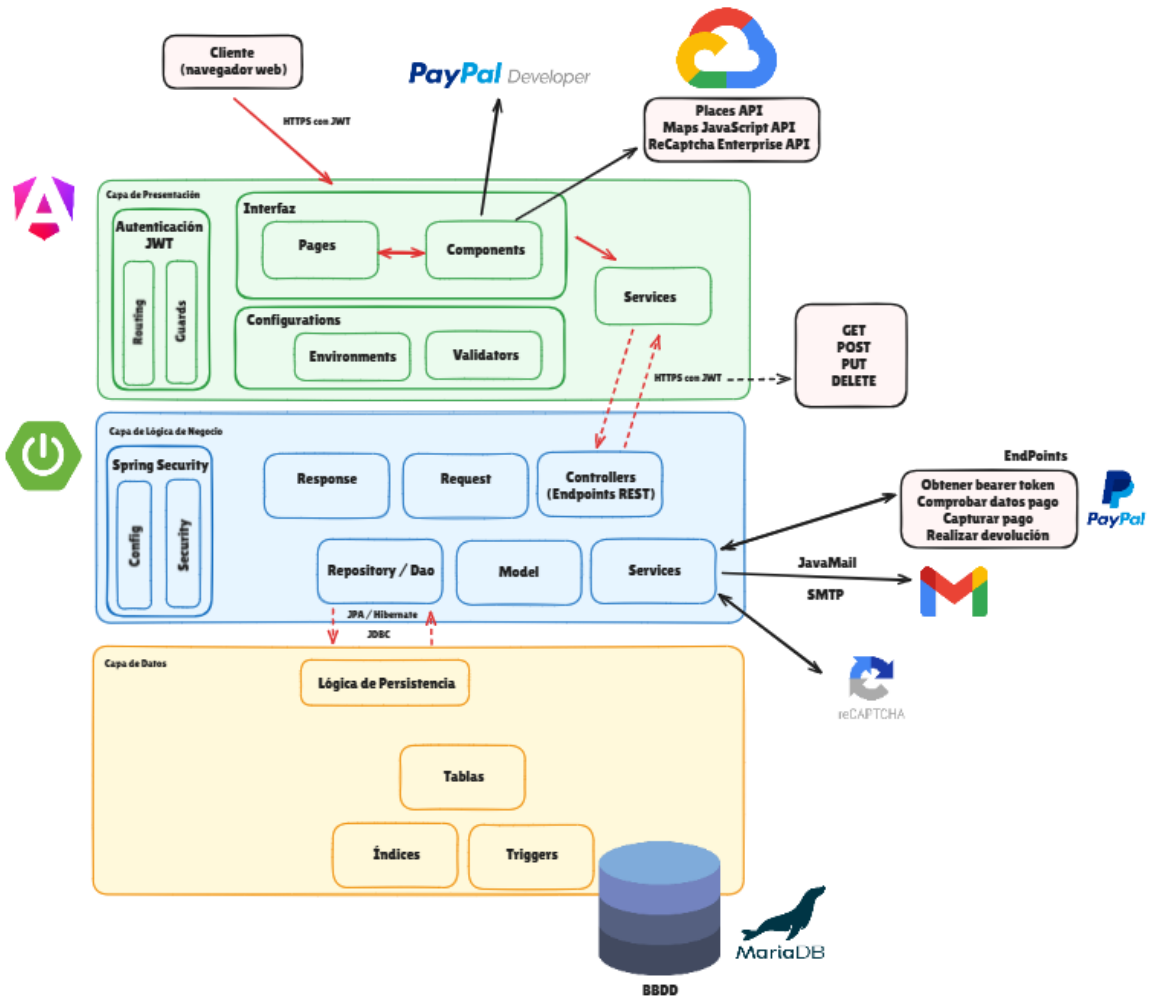

Figura 5.1: Arquitectura Lógica del sistema

Como se puede observar en el diagrama expuesto, se distinguen tres capas bien diferenciadas mediante diversos colores, además de una serie de servicios externos:

- **Capa de Presentación:** contiene los elementos que componen la interfaz con la que el usuario interactuará, una serie de configuraciones propias de *Angular* y los servicios que permiten la comunicación con la *API REST* de *Spring Boot*. Además, de forma transversal, encontramos los elementos que permiten la autenticación y seguridad en la capa *frontend*, como el *routing* y los *guards*, que evitan accesos indebidos.

- **Capa de Lógica de Negocio:** engloba una serie de paquetes propios del proyecto *backend*. Entre ellos, destacan los controladores, que mapean los diversos *endpoints* con los servicios a ejecutar. Encontramos además cierto modelado de respuestas y peticiones, así como las representaciones mediante clases de las entidades de la base de datos, y las clases orientadas a la persistencia de estos modelos. De nuevo, encontramos elementos cuya misión es garantizar la seguridad de la capa, en este caso estableciendo configuraciones propias de *Spring Security* y protegiendo *endpoints*.

- **Capa de Datos:** reúne la lógica de persistencia necesaria para guardar los datos recibidos o enviar los datos necesarios de la base de datos relacional. Alberga ciertas medidas de seguridad e integridad, como los *triggers*, así como de rendimiento, mediante índices.

- **Servicios externos:** representan a las *APIs* externas que se consultan, tanto desde el *frontend* para el renderizado de componentes especiales, como desde el *backend* para realizar ciertas comprobaciones o funcionalidades. Destacan *Google Cloud Console*, que facilita *APIs* para mostrar reseñas, mapas interactivos y protecciones *reCAPTCHA*, *PayPal Developer* para gestionar los pagos y *Gmail* para el envío de correos. A pesar de no ser una capa como tal, los servicios externos dotan al sistema de grandes funcionalidades necesarias para el correcto funcionamiento de este.

Esta lógica separada en capas permite diferenciar los subproyectos que componen la aplicación final.

Atendiendo a la secciones 6.1.1 y 6.2.1, podemos agregar mayor detalle a esta arquitectura, observando cada componente presente en los módulos del proyecto *Angular*, así como las clases que integran los paquetes del *backend*. De esta manera, lograremos desgranar los diversos elementos presentes en el diagrama anterior, comprendiendo la importancia de cada uno de ellos.

5.2. Arquitectura física

La arquitectura física de un proyecto *software* representa la disposición y configuración de los componentes *hardware* y *software* que integran el sistema. En esencia, es la representación de cómo se implementa el diseño abstracto del *software* en el mundo real, especificando qué componentes de *hardware* ejecutan qué tareas y cómo interactúan entre sí.

Al tratarse de un proyecto que, a día de hoy, se encuentra en una fase de desarrollo en local, se mostrará el diagrama correspondiente a la disposición actual. Aún así, al final de la sección se darán ciertas pinceladas acerca de posibles despliegues reales que se realizarían de la aplicación, de cara a futuro.

Sin más dilación, la arquitectura física de la aplicación sería la siguiente:

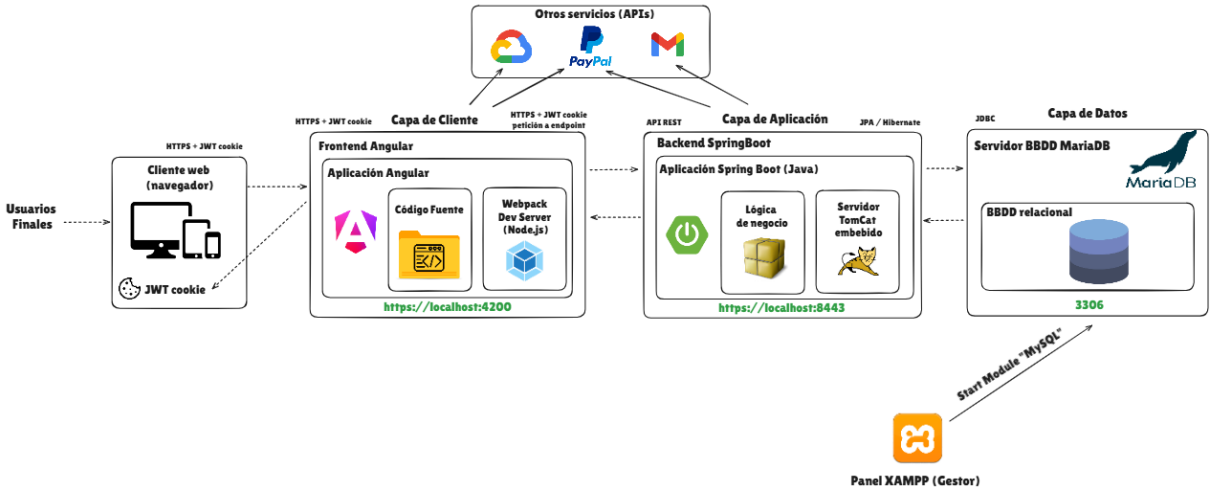

Figura 5.2: Arquitectura Física del sistema

Nótese que se puede consultar la versión ampliada en la figura B.3. Como se puede comprobar, existen varias capas diferenciadas en la arquitectura física expuesta, que procedo a explicar:

- El **cliente o navegador *web*** representa el punto de interacción de los usuarios reales con la aplicación desarrollada. Mediante navegadores como *Mozilla Firefox*, los usuarios podrán acceder a la página, realizando las diversas operaciones e interactuando con la interfaz.

- La segunda capa representa el *frontend* de la aplicación, es decir, la **aplicación de *Angular***. Estará compuesta por el código fuente como tal, dividido y organizado mediante carpetas y módulos. Esta estructura se desglosa en la sección 6.1.1. Para su ejecución en local, se empleará *Webpack Dev Server*, un servidor de desarrollo incorporado. Funciona manteniendo los archivos del proyecto en memoria y sirviéndolos a través de un servidor *HTTP*, permitiendo actualizaciones en tiempo real y recarga del navegador sin necesidad de guardar los archivos en el disco.

- En la capa de aplicación, encontramos el *backend* del proyecto, es decir, la **aplicación *Spring Boot***. Esta incluye tanto la lógica de negocio, organizada en diversos paquetes (como se explica en la sección 6.2.1), y un servidor *TomCat* embebido, permitiendo que las aplicaciones sean auto-contenidas y se ejecuten sin necesidad de un servidor externo.

- En la capa de datos, un servidor de BBDD *MariaDB* facilita la gestión de las tablas de la base de datos relacional. El gestor de bases de datos lo proporciona el módulo *MySQL* del panel de control de *XAMPP*.

- Como capa adicional, encontramos los servicios externos consumidos en la aplicación. Estas *APIs* de *Google Cloud Console*, *PayPal Developer* y *Gmail*, aportan funcionalidad a la página. Los servicios externos son explicados en profundidad en la sección 6.4.

Como se puede comprobar en el diagrama, las comunicaciones entre el cliente *web*, el *frontend* y el *backend*, utilizan *HTTPS*, a pesar de tratarse de un desarrollo en local. Esto se consigue gracias a ciertos certificados autofirmados y a configuraciones relacionadas.

Además, destaca la presencia de *cookies JWT*, inyectadas en el navegador, para el control de sesiones de usuario. Esto permitirá determinar qué usuarios podrán acceder a cada *endpoint* protegido de la *API REST*.

Por último, gracias al panel de control proporcionado por *XAMPP*, se logra desplegar el servidor de *MariaDB* para la gestión de la base de datos de la aplicación. Gracias a *JPA*, *Hibernate* y *JDBC*, ciertas tecnologías relacionadas con el acceso a bases de datos en aplicaciones *Java*, se logra establecer las conexiones oportunas entre la aplicación *Spring Boot* y la base de datos relacional.

Una vez comprendida la distribución física actual del proyecto, debemos tener en cuenta que tras el despliegue futuro, mencionado en las ampliaciones a realizar, la disposición cambiará. Considero interesante plantear esta situación, pues de esta forma, podremos hacernos a la idea de cuál será la organización de los elementos físicos al llevarse a cabo esta mejora a futuro.

Por ejemplo, es bastante probable que al realizarse un despliegue real de la aplicación, se empleen:

- En el caso de la aplicación *Angular*, servidores *web* como *Apache* o *Nginx*, servicios de alojamiento en la nube como *AWS S3*, *Azure App Service* o *Google Cloud Platform* e incluso plataformas de *PaaS* (*Platform as a Service*) como *Vercel* para servir archivos estáticos. Ya no se emplearía el comando `ng serve`, sino `ng build` para generar los archivos estáticos de la aplicación en la carpeta `dist`.

- En el caso de la aplicación *Spring Boot*, servidores *web* que ejecuten la aplicación gracias al archivo `JAR`, o bien servidores de aplicaciones externos que desplieguen la aplicación como un archivo `WAR`. Se podría incluso utilizar servicios en la nube empaquetar la aplicación *Spring Boot* en contenedores *Docker* y desplegarla en plataformas como *Kubernetes*.

- Algún tipo de servicio en la nube o servidor para instalar la base de datos, de forma que sea accesible desde el *backend* y se proteja.

De esta manera, se obtendría una arquitectura física real, realizando un despliegue que permita pasar a trabajar con la aplicación desarrollada en entornos reales. Además, podrían emplearse elementos que aumenten la seguridad, como *firewalls* para el control del tráfico o *proxys* para balanceos de carga, redirecciones o *caching*.

En resumen, el despliegue de un proyecto que combine las tecnologías *Angular*, *Spring Boot* y *MySQL*, conlleva compilar y desplegar cada componente por separado, configurando la conexión entre ellos a través de *APIs REST*. Para facilitar el proceso de despliegue y gestión, se pueden usar servidores *web*, contenedores y sistemas *CI/CD* (*Continuous Integration/Continuous Delivery/Deployment*).

5.3. Diagrama Entidad-Relación

El modelo entidad-relación (E-R) representa de forma conceptual los datos que maneja el sistema, así como las relaciones existentes entre ellos.

Gracias al modelo entidad-relación, considerado una de las técnicas *top-down* más empleadas para el diseño conceptual de la base de datos, podemos identificar las entidades relevantes del dominio del problema, sus atributos y restricciones característicos y las relaciones que se establecen entre dichas entidades.

Los elementos del diagrama, como se mencionaba previamente, son los siguientes:

- **Tipo de entidad:** una serie de elementos independientes desde el punto de vista de la organización, propios del mundo real, con una serie de propiedades en común, cuya existencia puede ser física o conceptual. Cada tipo de entidad tendrá una serie de instancias, llamadas ejemplares de la entidad.
- **Atributo:** es una propiedad característica de un tipo de entidad o relación.
- **Tipo de relación:** es una asociación significativa entre dos tipos de entidades.
- **Restricción:** establece ciertas condiciones y limitaciones a las relaciones entre instancias de entidades.

Analizando los diversos elementos, así como el problema a enfrentar, identificamos una serie de entidades, relaciones y atributos, que conforman el diagrama entidad-relación del proyecto, representado en la figura 5.3.

Como se puede comprobar, habrá usuarios corrientes, usuarios registrados y usuarios administradores, debido a la jerarquía de entidades que se presenta. La entidad `USUARIO` da lugar a una especialización parcial (no obligatoria) y disjunta, de forma que un usuario de la aplicación podrá considerarse `USUARIO`, `USUARIO REGISTRADO` o `ADMINISTRADOR`, pero nunca más de un tipo al mismo tiempo. Nótese que a pesar de no existir atributos distintos entre los diversos tipos de usuarios, para facilitar la comprensión de los roles que pueden tomar, se modela esta relación *IS_A*. Posiblemente, en versiones futuras se agreguen atributos que distingan entre las subclases y la superclase, teniendo más sentido entonces la especialización comentada. En ese momento, probablemente sería necesaria la inclusión de las tablas `USUARIO_REGISTRADO` y `USUARIO_ADMINISTRADOR`.

Las entidades presentes serán entidades fuertes, pues su existencia no dependerá de otro tipo de entidad. Los atributos serán simples y monovaluados. Las claves primarias

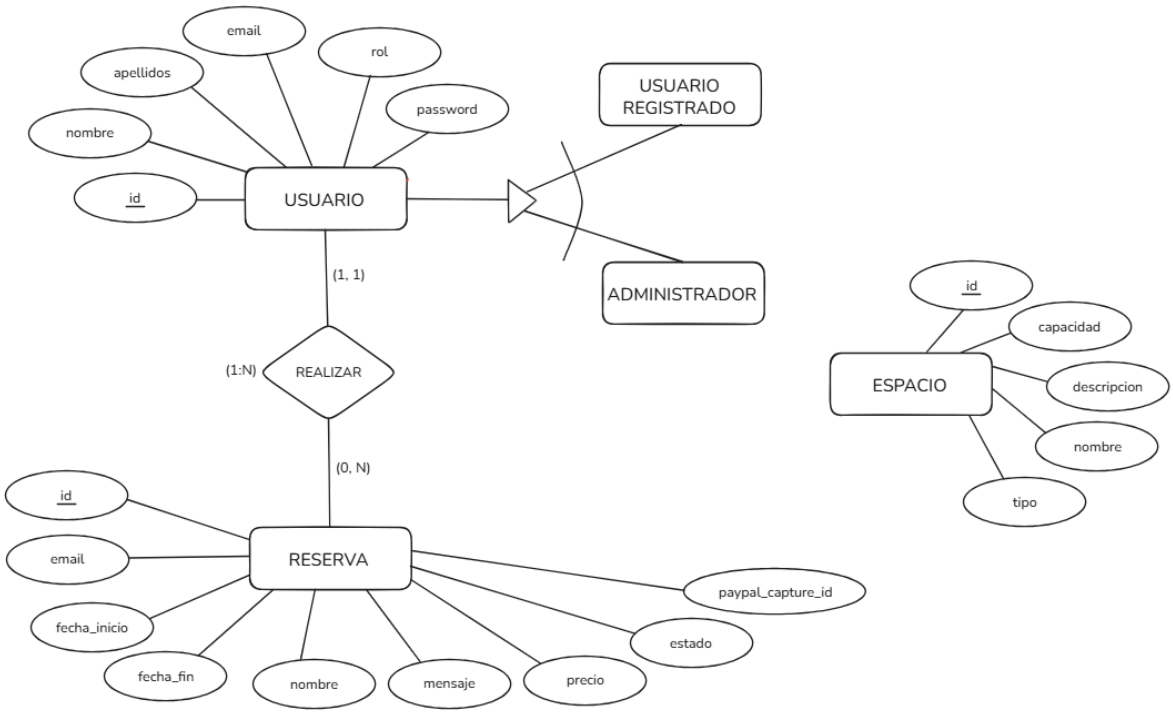

Figura 5.3: Diagrama Entidad-Relación

seleccionadas para identificar a cada instancia de los tipos de entidad, serán `IDs` auto-incrementales. Por su parte, existirán otras claves candidatas, como `email` en la entidad `USUARIO`. A simple vista, podría parecer que las dos fechas (inicio y fin) que determinan el rango temporal de la reserva, pueden formar una clave compuesta, ya que según la filosofía de la casa no podrían existir dos reservas en las mismas fechas. En cambio, al poder cancelar una reserva, existirían tuplas con las mismas fechas, pero distinto estado. En cuanto a las relaciones, encontraremos una relación común entre `USUARIO` y `RESERVA`, que modela la efectuación de reservas de la casa rural. Esta relación uno a muchos, tiene sentido ya que un `USUARIO` puede no realizar ninguna reserva, o efectuar muchas, así como una `RESERVA` debe ser realizada por uno y solo un usuario.

Para consultar ciertos detalles acerca de los metadatos, como ciertos rangos de los atributos, se recomienda acceder a la sección 5.4.

Nótese que el modelo conceptual no reúne una gran cantidad de entidades y relaciones, debido principalmente a que muchos elementos reconocidos en la aplicación no se persistirán en la base de datos, sino que serán gestionados por servicios externos. Además, la inconexión presente en el diagrama puede llamar la atención, pero tiene una explicación. La base de datos de la aplicación *web* desarrollada, modela dos realidades distintas: por un lado, la gestión de la casa rural, con sus diversos espacios. Por otro lado, modela la gestión de las reservas de la casa rural, lo que incluye tanto usuarios como reservas asociadas.

5.4. Diccionario de Datos

El diccionario de datos proporciona una descripción detallada de los atributos de cada una de las tablas que conforman el modelo de datos. Incluye el nombre del campo, tipo de dato, si es clave primaria, si puede ser nulo y una breve descripción, entre otras cosas.

Proporciona contenidos relativos al significado, dominio, formato, validación, origen, cálculo y valores por defecto de los datos. Recogerá los metadatos que describen las tres entidades principales del modelo conceptual, así como la relación existente.

A continuación, se muestra el diccionario de datos correspondiente:

Datos de la Entidad	
ID	E01
Nombre	`USUARIO`
Definición	Representa a los distintos tipos de usuarios del sistema.
Reglas	Cada usuario debe tener un identificador y correo únicos.
Atributos	
ID	A01.01
Nombre	`id`
Definición	Identificador autoincremental único del usuario.
Tipo de dato	`BIGINT(20)`
Reglas	Autoincremental, no modificable.
`UNIQUE`	SÍ
`DEFAULT`	-
`NULL`	NO
ID	A01.02
Nombre	`apellidos`
Definición	Apellidos (típicamente dos) del usuario registrado.
Tipo de dato	`VARCHAR(255)`
Reglas	Se permiten apellidos compuestos.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A01.03
Nombre	`email`
Definición	Correo electrónico del usuario registrado.
Tipo de dato	`VARCHAR(255)`
Reglas	Con formato de correo electrónico.
`UNIQUE`	SI
`DEFAULT`	-
`NULL`	NO
ID	A01.04
Nombre	`nombre`
Definición	Nombre de pila del usuario.

Atributos (continuación)	
Tipo de dato	`VARCHAR(255)`
Reglas	-
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A01.05
Nombre	`password`
Definición	Contraseña de la cuenta del usuario.
Tipo de dato	`VARCHAR(255)`
Reglas	Se mantendrá hasheada con *BCrypt*.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A01.06
Nombre	`rol`
Definición	Rol de la cuenta del usuario.
Tipo de dato	`ENUM`
Reglas	Existen dos tipos de rol, `Administrador` y `Usuario`.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
Identificador de la entidad	
ID	A01.01
Nombre del atributo	`id`

Cuadro 5.1: Diccionario de datos de la entidad `USUARIO`

Datos de la Entidad	
ID	E02
Nombre	`RESERVA`
Definición	Representa las reservas de la casa rural que se efectuarán por usuarios del sistema.
Reglas	Las reservas podrán ser realizadas por usuarios de todo tipo.
Atributos	
ID	A02.01
Nombre	`id`
Definición	Identificador autoincremental único de la reserva.
Tipo de dato	`BIGINT(20)`
Reglas	Autoincremental, no modificable.
`UNIQUE`	SÍ
`DEFAULT`	-

Atributos (continuación)	
`NULL`	NO
ID	A02.02
Nombre	`email`
Definición	Correo electrónico escogido para efectuar la reserva.
Tipo de dato	`VARCHAR(255)`
Reglas	No es necesario que se corresponda con el correo del usuario vinculado (si lo hay).
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A02.03
Nombre	`fecha_fin`
Definición	Fecha de finalización de la reserva.
Tipo de dato	`DATE`
Reglas	Debe ser posterior a la fecha de comienzo de la reserva. Se validará que, junto con la fecha de inicio, no se solapen con otras reservas en ese mismo rango temporal.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A02.04
Nombre	`fecha_inicio`
Definición	Fecha de comienzo de la reserva.
Tipo de dato	`DATE`
Reglas	Debe ser anterior a la fecha de fin de la reserva. Se validará que, junto con la fecha de fin, no se solapen con otras reservas en ese mismo rango temporal.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A02.05
Nombre	`mensaje`
Definición	Mensaje acerca de la reserva, totalmente opcional.
Tipo de dato	`VARCHAR(255)`
Reglas	-
`UNIQUE`	NO
`DEFAULT`	`NULL`
`NULL`	SI
ID	A02.06
Nombre	`nombre`
Definición	Nombre de la persona para quien es la reserva.
Tipo de dato	`VARCHAR(255)`

Atributos (continuación)	
Reglas	-
UNIQUE	NO
DEFAULT	-
NULL	NO
ID	A02.07
Nombre	precio
Definición	Precio de la reserva calculado en base a los días seleccionados, tarifas de temporada y descuentos aplicados.
Tipo de dato	INT(11)
Reglas	No podrá ser negativo. Será un entero, ya que se redondeará el precio calculado.
UNIQUE	NO
DEFAULT	-
NULL	NO
ID	A02.08
Nombre	estado
Definición	Estado en el que se encuentra la reserva.
Tipo de dato	ENUM
Reglas	La reserva podrá encontrarse en estado ACTIVA o CANCELADA.
UNIQUE	NO
DEFAULT	-
NULL	NO
ID	A02.09
Nombre	paypal_capture_id
Definición	Identificador del pago de *PayPal*, necesario para efectuar devoluciones al cancelar una reserva. Se agrega a la reserva tras capturar el pago.
Tipo de dato	VARCHAR(255)
Reglas	-
UNIQUE	NO
DEFAULT	NULL
NULL	SI
Identificador de la entidad	
ID	A02.01
Nombre del atributo	id

Cuadro 5.2: Diccionario de datos de la entidad RESERVA

Datos de la Entidad	
ID	E03
Nombre	`ESPACIO`
Definición	Representa los espacios que componen la casa rural.
Reglas	-
Atributos	
ID	A03.01
Nombre	`id`
Definición	Identificador autoincremental único del espacio.
Tipo de dato	`BIGINT(20)`
Reglas	Autoincremental, no modificable.
`UNIQUE`	SÍ
`DEFAULT`	-
`NULL`	NO
ID	A03.02
Nombre	`capacidad`
Definición	Cantidad de personas que caben en el espacio.
Tipo de dato	`INT(11)`
Reglas	En espacios comunes, con capacidad indeterminada, se asigna el valor O. No podrá tomar un valor negativo.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A03.03
Nombre	`descripcion`
Definición	Información acerca del espacio, de carácter opcional.
Tipo de dato	`VARCHAR(255)`
Reglas	-
`UNIQUE`	NO
`DEFAULT`	`NULL`
`NULL`	SI
ID	A03.04
Nombre	`nombre`
Definición	Nombre del espacio.
Tipo de dato	`VARCHAR(255)`
Reglas	No se permitirá que esté formado por espacios en blanco o que esté vacío.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
ID	A03.05
Nombre	`tipo`
Definición	Tipo del espacio.

Atributos (continuación)	
Tipo de dato	`ENUM`
Reglas	Los tipos posibles son `Aseo`, `Cocina`, `Dormitorio`, `Especial`, `Salon`. La capacidad para el tipo `Especial` será 0.
`UNIQUE`	NO
`DEFAULT`	-
`NULL`	NO
Identificador de la entidad	
ID	A03.01
Nombre del atributo	`id`

Cuadro 5.3: Diccionario de datos de la entidad `ESPACIO`

Datos de la Relación	
ID	R01
Nombre	`REALIZAR`
Definición	Modela el hecho de que un usuario pueda realizar reservas de la casa rural. Modela una asociación fuerte entre `USUARIO` y `RESERVA`.
Reglas	-
Entidades	
ID	E01
Nombre Entidad	`USUARIO`
Participación	1
Cardinalidad	N
ID	E02
Nombre Entidad	`RESERVA`
Participación	1
Cardinalidad	1

Cuadro 5.4: Diccionario de datos de la relación `REALIZAR`

Gracias al diccionario de datos planteado, se puede contar con una descripción detallada de la base de datos, comprendiendo múltiples características y metadatos asociados.

5.5. Modelo Relacional

A partir del diagrama E-R, se ha realizado su transformación al modelo relacional. Este modelo representa las estructuras lógicas que serán implementadas en el sistema gestor de bases de datos relacional (SGBDR), especificando las tablas, claves primarias y foráneas, y los atributos correspondientes.

Antes de nada, es relevante recordar una serie de definiciones clave para comprender ciertas transformaciones y equivalencias entre el modelo conceptual y lógico:

- Una **relación** es un conjunto de filas y columnas que componen una tabla.
- Un **atributo** es una columna con cierto nombre dentro de una relación.
- Una **tupla** es una fila de una relación.
- Una **clave primaria** es la clave candidata seleccionada para identificar a las tuplas de una relación.
- Una **clave foránea** es un conjunto de uno o más atributos de una relación que, a su vez, son la clave primaria de otra relación.

Una vez entendido esto, podemos comenzar a plantear el modelo lógico de nuestro sistema.

Para realizar la transformación del modelo conceptual al modelo lógico, se aplican una serie de reglas en base a los tipos de entidades, relaciones y atributos presentes. A continuación, se presenta el modelo relacional obtenido:

- `USUARIO(id, nombre, apellidos, email, rol, password)`
- `RESERVA(id, email, fecha_inicio, fecha_fin, nombre, mensaje, precio, estado, paypal_capture_id, usuario_id)`
  `FK: usuario_id -> USUARIO(id)`
- `ESPACIO(id, capacidad, descripcion, nombre, tipo)`

Donde las claves primarias están subrayadas y las claves foráneas están indicadas como *FK* (*Foreign Key*). Gracias a este esquema, podemos hacernos a la idea de la apariencia que tendrán las tablas en nuestra base de datos.

Se puede representar de forma visual el modelo relacional comentado, con la intención de ilustrar las relaciones entre entidades, los diversos atributos, etc. Dicha representación se expone en la figura 5.4.

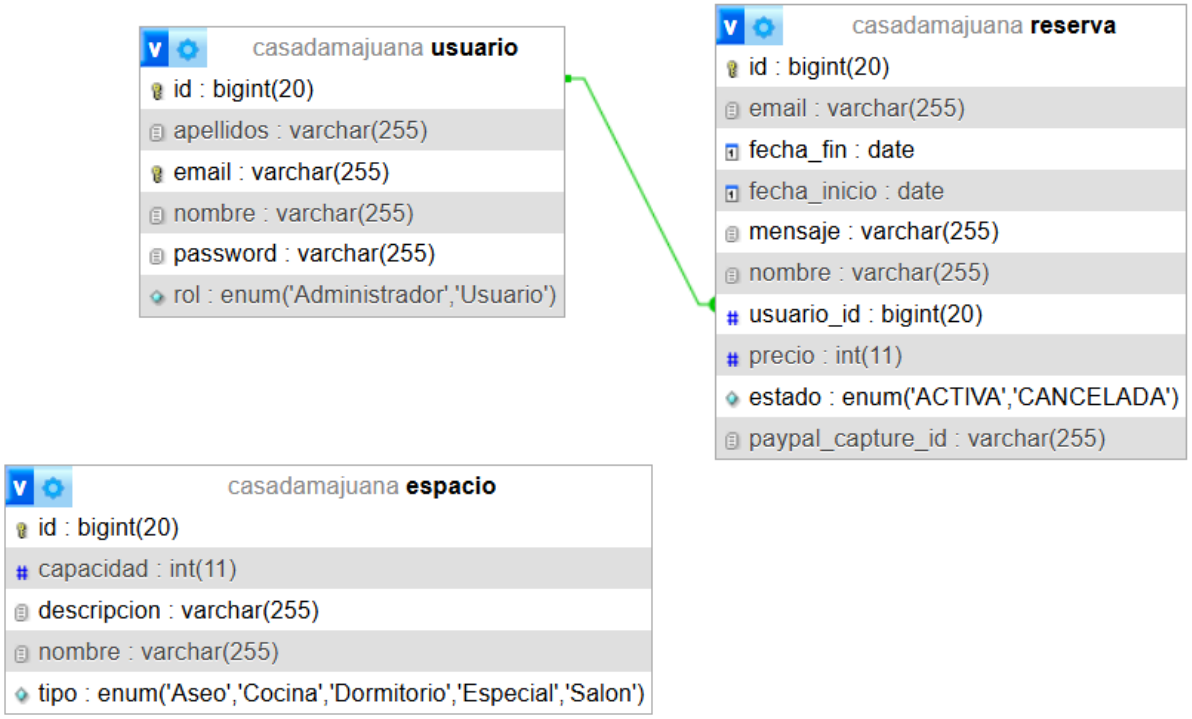

Figura 5.4: Modelo Relacional

En cuanto a las restricciones de integridad se refiere, podemos llegar a distinguir los siguientes tipos:

- **Integridad de entidad:** se satisface al verificar que ninguno de los atributos que forman las claves primarias puedan tener valor `NULL`. En este caso, los identificadores de las entidades son atributos autoincrementales, que siempre serán diferentes para cada nueva instancia, a pesar de la presencia de modificaciones, borrados o inserciones de tuplas.
- **Integridad referencial:** establece que el valor de una clave foránea debe coincidir con el valor de la clave candidata de alguna tupla en la relación referenciada o, por el contrario, ser `NULL`. De nuevo, se satisface este requisito, pues debido a la participación opcional presente en la relación `REALIZAR`, entre `USUARIO` y `RESERVA`, siempre encontraremos valores válidos para el atributo `usuario_id` de esta última entidad. Como los identificadores de usuario no podrán variar, únicamente el borrado de un usuario podría comprometer la integridad referencial de las tuplas de la tabla `RESERVA` que apunten a dicho identificador. Para ello, se establece la estrategia `SET NULL`, ya que según la lógica de negocio, al borrar la cuenta de usuario, las reservas seguirán manteniéndose activas, y el usuario pasará a ser un usuario anónimo.
- **Restricciones generales:** que marcan una serie de reglas, cuya misión es definir o restringir algún aspecto a controlar sobre los datos. Destacan la posibilidad de

establecer valores `NULL` o las restricciones de dominio, que podemos ver al detalle en la sección 5.4. Además, habrá otras restricciones, como las controladas con los famosos *triggers*, de los que se hablará en la sección 6.3.2.

5.6. Diseño de la interfaz

El diseño de la interfaz de usuario fue planificado mediante la elaboración de bocetos (*wireframes*), los cuales permitieron definir de forma temprana la disposición, jerarquía y navegación de las distintas pantallas del sistema. Esto favoreció la validación temprana del flujo de interacción con los usuarios antes de proceder con el desarrollo en *Angular*.

Los *wireframes* son esquemas visuales en blanco y negro, elaborados en este caso con *Draw.io* [47], que permiten representar la estructura y disposición de los elementos que componen la aplicación *web*.

La implementación de la interfaz se realizó utilizando *Angular*, apoyado en diversos recursos para facilitar la construcción de interfaces responsivas y accesibles:

- ***Angular Material*** [25]: para componentes *UI* (*User Interface*) consistentes y accesibles (botones, formularios, diálogos, etc.).
- ***NG Bootstrap*** [36]: para integraciones rápidas de *Bootstrap* en *Angular*.
- ***Bootstrap 5*** [27]: incluido vía *CDN* (*Content Delivery Network*), utilizado principalmente para *layout* y estilos.
- ***Google Fonts*** [31] **y** ***Material Icons***: para tipografía moderna y uso de iconografía estandarizada.

Tras explicar algunas de las herramientas empleadas para la construcción de la interfaz, pasamos a mostrar algunos *mockups*. Los siguientes bocetos representan las principales pantallas del sistema. Fueron elaborados en etapas tempranas del desarrollo para definir la estructura visual y la interacción del usuario con el sistema. Por simplicidad, se mostrarán los elementos más relevantes, por lo que si se requiere consultar en detalle la apariencia de la interfaz de usuario, recomiendo acceder al capítulo 9, que muestra las pantallas disponibles para cada tipo de usuario.

Comenzamos mostrando la apariencia general de las pantallas de la aplicación mediante la figura 5.5. Se dispondrá de un encabezado, que contendrá el logo, así como cierta información (títulos, botones de inicio de sesión o en su defecto botón de cerrar sesión, etc.). El *navbar* mostrará diversos botones que permitirán acceder a las secciones disponibles de la aplicación según el rol. El contenido de cada ventana se mostrará en el centro, y debajo de él estará presente el *footer*, con cierta información relevante.

Las pantallas de registro e inicio de sesión serán similares, pues presentarán un formulario que permita rellenar los datos requeridos, seguido de un botón que envíe ese formulario. Al accionar el botón, siempre y cuando los datos sean correctos, se redirigirá a otra ventana: a la página principal en el caso del *login* y a la pantalla de inicio de sesión en

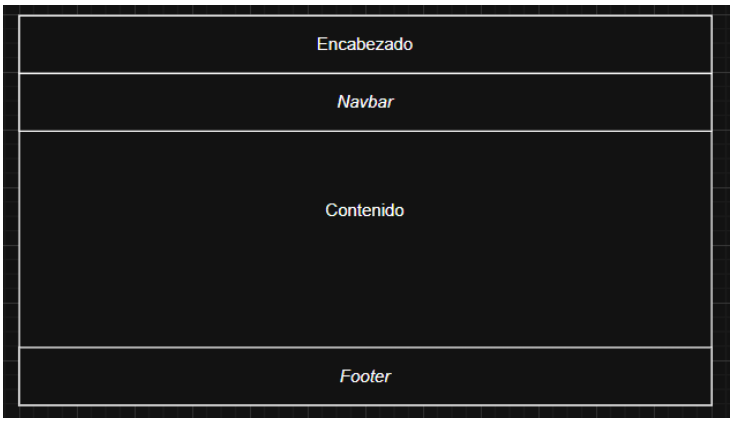

Figura 5.5: Boceto de la apariencia general de las pantallas

el caso del registro. Se muestra el *wireframe* de las pantallas de inicio de sesión y registro en la figura 5.6.

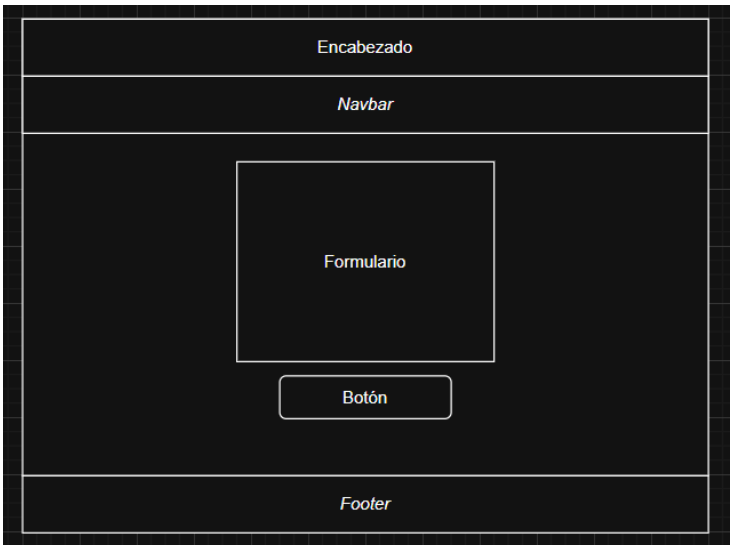

Figura 5.6: Boceto de las pantallas de inicio de sesión y registro

El menú principal contará con elementos que permitan a los usuarios, de un vistazo, hacerse a la idea de la apariencia y servicios de la casa. Para ello, se mostrará un carrusel de fotografías interactivo, información general, paneles con servicios, etc. El boceto de esta pantalla se muestra en la figura 5.7.

En cuanto a la página de habitaciones, la idea es mostrar una tabla que las agrupe, con una serie de filtros interactivos. Además, cada habitación contará con un botón que permita acceder a una página de información específica. Se expone dicho boceto en la figura 5.8.

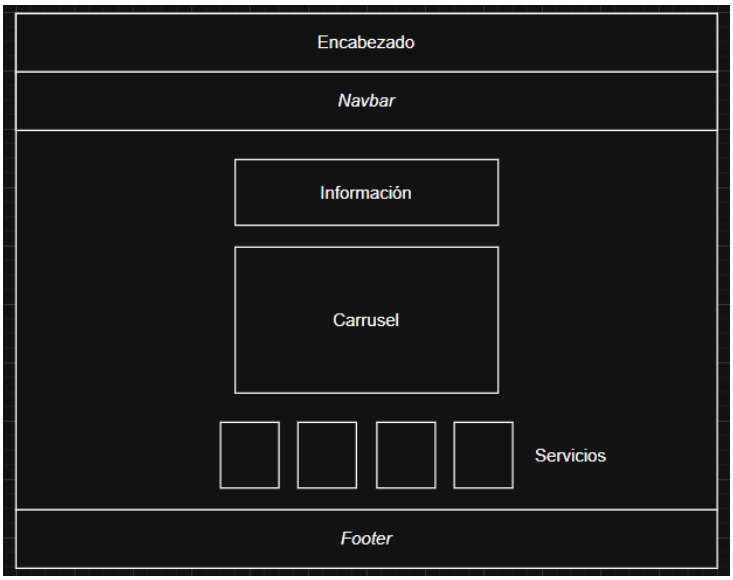

Figura 5.7: Boceto de la pantalla del menú principal

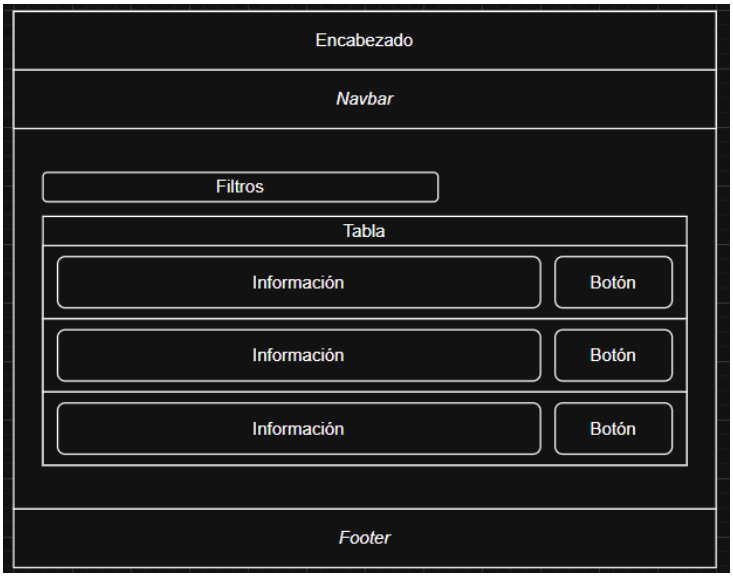

Figura 5.8: Boceto de la pantalla de habitaciones

Si accedemos al detalle de una habitación, encontraremos una composición de fotografías de la misma, con un diseño *Masonry*, así como cierta información adicional del espacio. Además, existirá un botón que permita retornar a la página de habitaciones general. Este *wireframe* se expone en la figura 5.9.

Por su parte, en la pantalla de reservas habrá numerosos elementos que faciliten al usuario la tramitación de su reserva. Contará con un calendario interactivo para la selec-

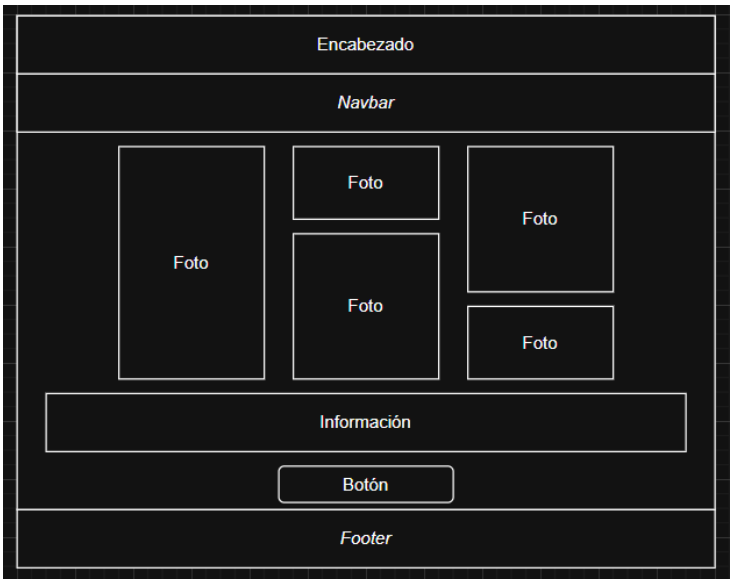

Figura 5.9: Boceto de la pantalla de detalles de la habitación

ción de fechas, un formulario para establecer los datos del comprador, botones de pago con *PayPal* y diversos botones para realizar acciones (limpiar selección y completar formulario). Dicho boceto se dispone en la figura 5.10.

En la pantalla de reseñas, se mostrarán una serie de valoraciones destacadas, así como botones que permitan a los usuarios acceder a más reseñas y a formularios de valoración de su experiencia. El boceto de la pantalla correspondiente se muestra en la figura 5.11.

Pasamos a la página de contacto, en la que se espera contar con un mapa interactivo, cierta información de contacto y un formulario que permita a los usuario consultar dudas a la propietaria. Este boceto se expone en la figura 5.12.

En la pestaña de más información, encontraremos datos de la casa rural, recomendaciones de actividades en la zona (que podrán filtrarse), las normas de la casa, etc. Represento este boceto en la figura 5.13.

Para los usuarios registrados, se habilitará un panel de usuario que contenga los datos de su cuenta (permitiendo su modificación), así como una tabla de sus reservas. Se habilitarán botones para el borrado de la cuenta, la cancelación de reservas, etc. Podemos observar este boceto en la figura 5.14.

Por último, se habilita un panel de administración para los administradores registrados. Este contendrá dos tablas, una con el total de usuarios de la aplicación y otra con las reservas globales. Podrá filtrar la información, exportar datos y realizar acciones como la eliminación de usuarios o la cancelación de reservas. Dicho boceto se encuentra representado mediante la figura 5.15.

Se obvian ciertas pantallas de éxito o error, mensajes informativos, *modals* y *toasts*, que no se representan en los bocetos. Con la intención de mejorar la experiencia de usuario, se utilizan ciertos componentes y elementos reutilizables y totalmente personalizados, que

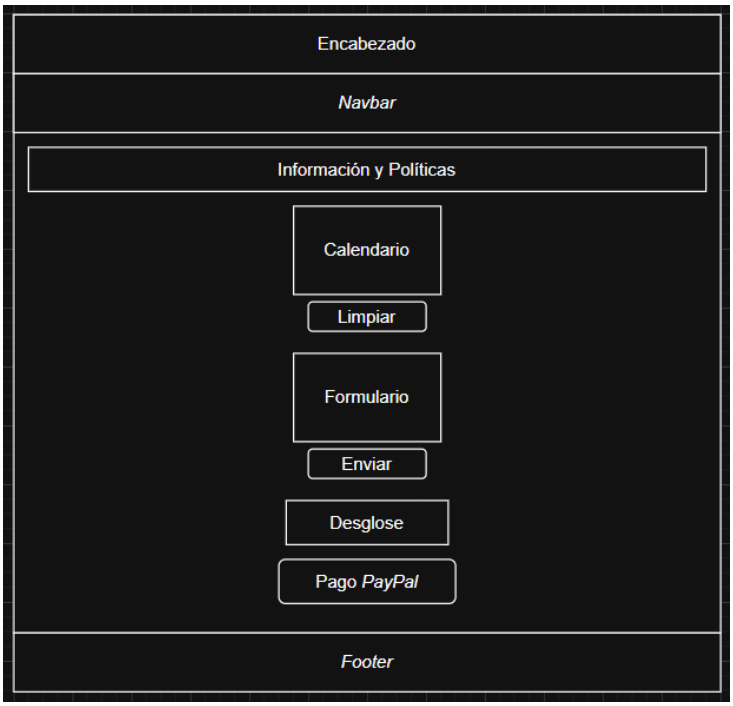

Figura 5.10: Boceto de la pantalla de reserva

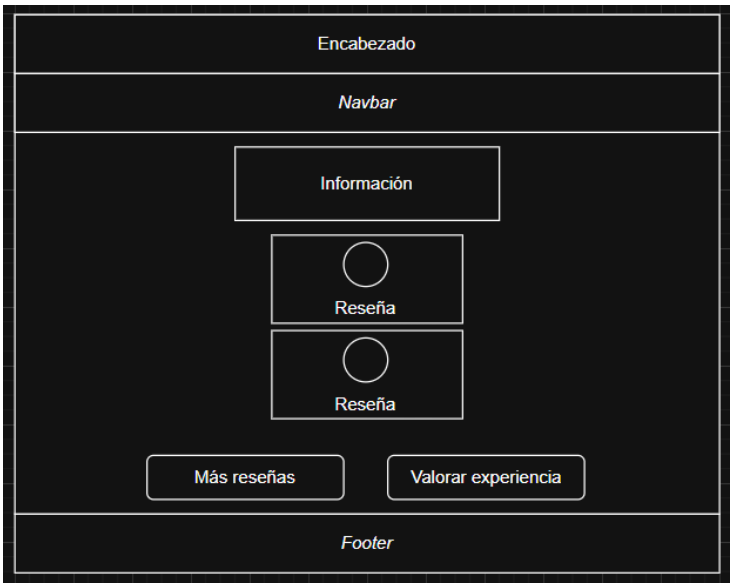

Figura 5.11: Boceto de la pantalla de reseñas

resumo a continuación:

- **Modales (*modals*):** es un componente de la interfaz de usuario que aparece sobre

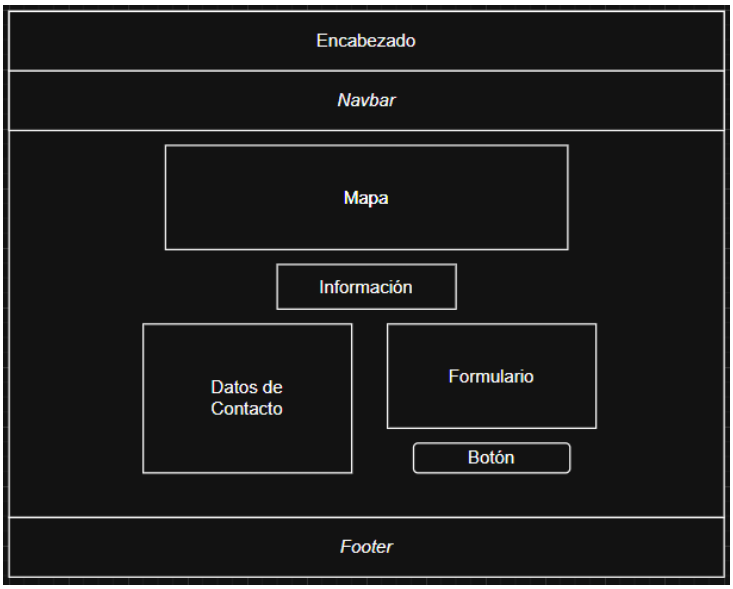

Figura 5.12: Boceto de la pantalla de contacto

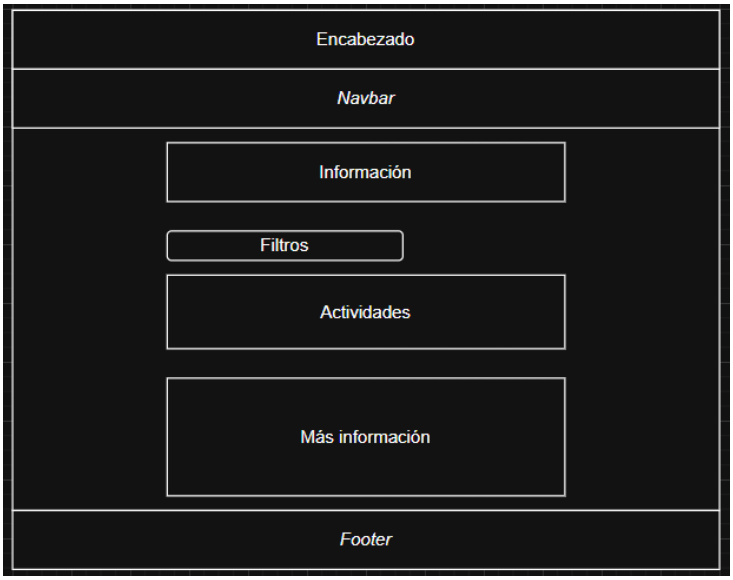

Figura 5.13: Boceto de la pantalla de más información

el contenido principal de la página, con la intención de mostrar información adicional previa a cierta acción, permitiendo confirmar su realización o abortar el proceso. Se han empleado para la confirmación de acciones importantes, como cancelaciones de reservas o eliminación de cuentas de usuario.

- ***Toasts*:** es un componente de interfaz de usuario que muestra notificaciones tem-

Figura 5.14: Boceto del panel de usuario

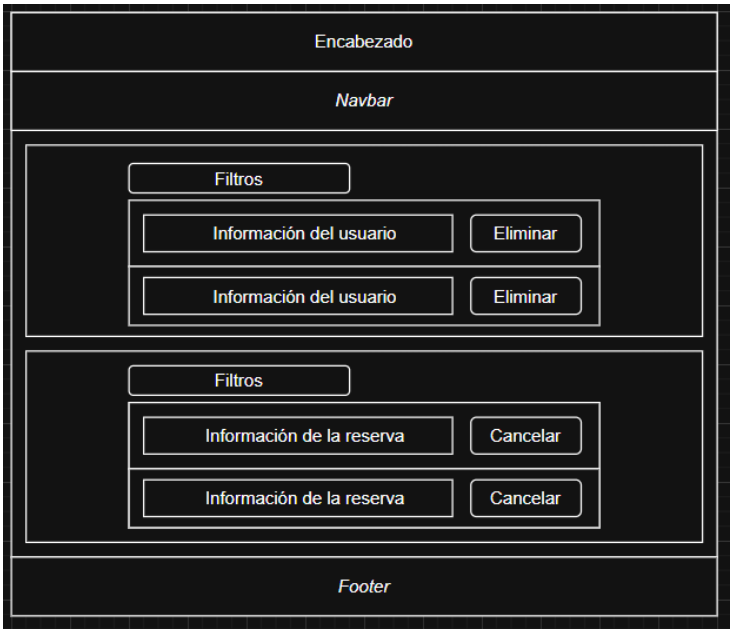

Figura 5.15: Boceto del panel de administrador

porales y breves al usuario. Se ha empleado para mostrar mensajes de éxito o error, posteriores a las acciones importantes, en la parte superior de la pantalla, desapareciendo tras unos segundos.

- **Desplegables (*accordions*):** es un componente de interfaz de usuario que permite mostrar contenido verticalmente expandible, haciendo que el usuario pueda ocultar o mostrar ciertas tablas (como las de reservas o usuarios), organizando el espacio de trabajo.
- **Botones (*buttons*):** los clásicos componentes que permiten a un usuario desencadenar una acción tras su pulsado. Se han combinado con mensajes de éxito y error, que aporten *feedback* a los usuarios tras las interacciones.
- **Paginación:** se añade un componente de interfaz de usuario que permite la organización de la información presentada en tablas, distribuyéndose en diversas páginas, simplificando su visualización y obteniendo una interfaz más limpia.
- **Otros componentes:** elementos como el *carousel* de imágenes, las tablas (que se convierten en tarjetas en pantallas pequeñas) y otros componentes han permitido mejorar la experiencia de usuario al interactuar con la aplicación.

Todos estos elementos permiten dar a luz una interfaz sencilla a la par que funcional, que fomente la reutilización de código y permita mejorar la usabilidad y accesibilidad, como se comenta en la sección 6.8.

Es interesante comentar que para la gestión del estilo, se ha desarrollado una paleta de colores basada en el logotipo de la casa rural, combinando tonos verdosos característicos con colores básicos, que permitan tematizar la aplicación *web* y sus componentes, a la vez que se mantiene un diseño limpio y profesional.

Figura 5.16: Paleta de colores de la aplicación

Cabe destacar que se han empleado colores adicionales, como tonos rojizos para los mensajes de error, tonos oscuros para mejorar el contraste, entre otras cosas. Sin embargo, al haber hecho un pequeño análisis del estilo a aplicar, la página cuenta con un aspecto propio y elaborado, que permite su distinción de los competidores, potenciando su imagen de marca.

Capítulo 6

Implementación

El capítulo actual estará dedicado a explicar la organización a nivel de código seguida en los subproyectos que componen este Trabajo Fin de Grado.

Se tratarán temas referentes al código desarrollado en el *frontend* (*Angular*), *backend* (*Spring Boot*) y base de datos (*MySQL/MariaDB*). Explicaré desde la estructura de cada subproyecto, hasta ciertos matices ligados a la seguridad e integridad en las diversas capas, entre otros.

6.1. *Angular*

Comenzaremos tratando el tema de la implementación en la parte *frontend* de la aplicación. Como se menciona en otras secciones, se ha trabajado con *Angular* y otras tecnologías complementarias, con la intención de dar a luz una capa de interacción con el cliente acorde a los requisitos y necesidades de la aplicación.

6.1.1. Módulos y Componentes

Los proyectos realizados en *Angular* siguen una arquitectura modularizada, que permite mantener una separación de responsabilidades y funcionalidad de cada elemento, ayudando a la organización, mantenimiento y escalabilidad del código.

Para comprender la función que desempeña cada carpeta y archivo, expondré un breve resumen explicativo acerca de los elementos incluidos en la raíz del proyecto:

- La carpeta `.angular` es generada por el *Angular CLI* para almacenar la caché de compilaciones previas, permitiendo reutilizar información de compilaciones anteriores.

- La carpeta `.vscode` contiene configuraciones específicas para el entorno de desarrollo en *Visual Studio Code*. Ayuda a configurar el editor que he decidido emplear en el proyecto.

- El directorio `certs` alberga un certificado y una clave privada autogenerados que permiten establecer *https* en el *frontend* a pesar de estar trabajando en un entorno local. Se hará énfasis en este tema en secciones posteriores.
- Los `node_modules` son todas las dependencias y módulos de *Node.js* necesarios para que el proyecto funcione correctamente. Se crea esta carpeta automáticamente al instalar paquetes con `npm`.
- La carpeta `public` aloja archivos estáticos que se sirven directamente al usuario, como imágenes o el *favicon*. En mi caso, los contenidos estáticos están organizados en subcarpetas referentes a las diversas páginas de la aplicación, agrupándolos por su pertenencia a dichas secciones.
- El directorio `src` contiene el código fuente principal de la aplicación, que se expone más adelante en esta sección.
- El archivo `.editorconfig` define la configuración del editor de código, asegurando estilos de codificación consistentes.
- El archivo `.gitignore` es crucial para decirle a *Git* qué archivos y carpetas debe ignorar y no *trackear* en el repositorio. En el proyecto, es esencial para evitar que se suban al repositorio diversos archivos de configuración o que contengan información sensible.
- El fichero `angular.json` es el archivo de configuración principal para proyectos *Angular* generados por la *CLI* de *Angular*. Contiene diversas configuraciones (desarrollo, espacio de trabajo, proyecto, compilación, etc.) que permiten establecer el comportamiento a seguir de la *CLI* (*Command Line Interface*).
- El archivo `package-lock.json` es un archivo generado automáticamente que registra las versiones exactas de todas las dependencias instaladas, incluyendo sus dependencias anidadas. Al ejecutar `npm install` para agregar dependencias, garantiza que se instalen las mismas versiones de los paquetes, manteniendo la consistencia entre diferentes entornos de desarrollo, pruebas y producción.
- El fichero `package.json` facilita la gestión de dependencias y la configuración del proyecto. Es un archivo *JSON* que contiene información sobre el proyecto, incluyendo su nombre, versión, descripción, y, lo más importante, la lista de dependencias y herramientas necesarias para su correcto funcionamiento.
- El fichero `README.md` incluye cierta documentación, instrucciones relevantes, etc. En mi caso, no es un archivo sumamente importante, debido a que esta memoria constituye el grueso de la documentación.
- El archivo `tsconfig.app.json` es un archivo de configuración de *TypeScript* específico para la aplicación, que extiende la configuración realizada en el fichero expuesto a continuación.

- El fichero `tsconfig.json` es un archivo de configuración para el compilador de *TypeScript*. Define cómo el compilador debe compilar el código *TypeScript* a *JavaScript*.
- El archivo `tsconfig.spec.json` se utiliza para configurar la compilación de *TypeScript* para las pruebas unitarias, es decir, para los archivos `.spec` ligados a los componentes.

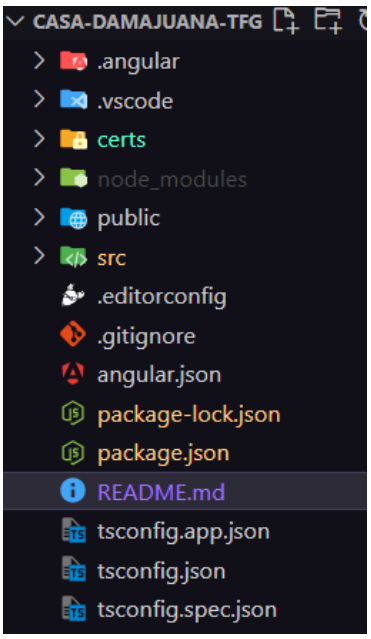

Figura 6.1: Estructura raíz del proyecto *Angular*

Si nos adentramos en la carpeta `src`, encontraremos otros elementos relevantes:

- La carpeta `app` es el directorio principal donde se encuentra la lógica y los componentes de la aplicación. La veremos en profundidad en breves.
- El directorio `environments` contiene archivos de configuración que permiten definir variables de entorno para diferentes entornos de la aplicación, como desarrollo y producción. En este proyecto, al haberse demostrado también una prueba de internacionalización, existen archivos de configuración de entorno para diversos idiomas.
- La carpeta `locale` se utiliza para organizar los archivos de traducción, permitiendo la internacionalización (`i18n`) de la aplicación. En este proyecto, se expone un ejemplo de internacionalización para demostrar que se ha investigado y aplicado esta configuración, mediante los archivos `.xlf` presentes en esta carpeta.
- El archivo `custom-theme.scss` se utiliza para definir estilos personalizados para *Angular Material*.

- El fichero `index.html` es el punto de entrada principal para una aplicación *Angular*, y contiene la estructura básica del documento *HTML*. En este caso, contiene un elemento `<app-root>` que actúa como contenedor para la aplicación *Angular*.
- El fichero `main.ts` es el punto de entrada principal para una aplicación *Angular*. Es donde la aplicación comienza a ejecutarse y donde se inicia el módulo raíz.
- El archivo `styles.css` se utiliza para estilos globales de la aplicación.

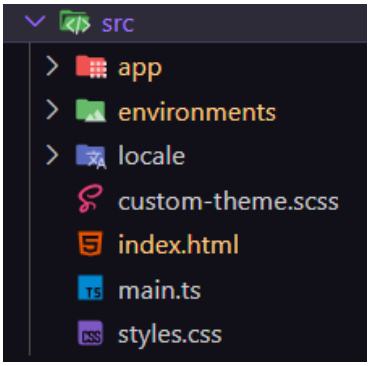

Figura 6.2: Estructura de la carpeta `src` del proyecto *Angular*

Como comentaba previamente, la carpeta `app` alberga los distintos paquetes funcionales de la aplicación. Encontramos el módulo principal del proyecto, conocido como `app.module.ts`. Este módulo raíz constituye el punto de entrada a la aplicación, declarando otros submódulos y dando lugar a la configuración y organización inicial.

También podemos observar los archivos de tipo `app.component.*`, en los que se definen la lógica, estructura y aspecto de la clase principal de la aplicación.

Además, contamos con `app.routing.module.ts`, un archivo que define la configuración del enrutamiento, especificando qué componente se debe mostrar cuando el usuario visita una *URL* específica. En mi proyecto, empleo este fichero para definir las rutas base de la aplicación, realizar cargas perezosas para ahorrar recursos, realizar redirecciones y proteger ciertas rutas mediante *guards*.

Por último, dentro de la mencionada carpeta `app` encontramos subcarpetas llamadas módulos. Los módulos en *Angular* son contenedores que agrupan componentes, directivas, *pipes* y servicios relacionados, creando una unidad funcional cohesiva. De esta forma, resulta más sencillo reutilizar y mantener el código.

En este proyecto se han empleado tanto módulos tradicionales como **componentes *standalone***, una nueva característica de *Angular* que permite definir componentes que no requieren estar declarados en un módulo específico. Esta práctica reduce el acoplamiento y simplifica la reutilización de componentes en diferentes contextos. Básicamente, es una nueva tendencia fomentada por los propios encargados del mantenimiento de *Angular*. En este caso, se han empleado componentes *standalone* para ciertos elementos reutilizables

y autosuficientes, y por organización del propio proyecto se han incluido en carpetas que actúan como módulos, pero sin ser declarados propiamente en estos últimos, conservando su independencia.

Cada módulo, salvo que se indique que es una carpeta con otro propósito, contiene la declaración de los componentes que alberga, así como configuraciones de su enrutado, que extiende de la configuración global de las rutas de la aplicación. Encontramos el siguiente abanico de carpetas y módulos:

- **Módulo `admin`:** presenta páginas propias de los usuarios administradores, como el panel de administrador, así como componentes como la tabla de usuarios de la aplicación.
- **Módulo `auth`:** contiene las páginas de registro e inicio de sesión, los servicios necesarios para realizar estas funciones, los *guards* que protegen ciertas rutas en base al rol, etc.
- **Módulo `contact`:** almacena la página de contacto, el servicio que permite la comunicación vía formulario de contacto con la administradora de la casa y ciertas interfaces.
- **Módulo `home`:** contiene las páginas de inicio de la aplicación *web*, así como las plantillas de estas secciones.
- **Módulo `material`:** importa ciertos componentes de *Angular Material*, que pueden ser utilizados si se necesitan a la hora de elaborar el código.
- **Módulo `more`:** tiene las interfaces y páginas necesarias para mostrar el catálogo de actividades en la zona, las reglas de la casa y cierta información adicional.
- **Módulo `reservations`:** contiene todo lo relativo a la gestión de reservas, desde los componentes para el pago (botones de *PayPal*), las interfaces y servicios necesarios para la comunicación con las *APIs* y el *backend*, y las páginas de reservas, errores y resumen de reserva.
- **Módulo `reviews`:** recoge las páginas relativas a mostrar las reseñas mediante la *API* de *Google*, facilitando botones que llevan a los usuarios de la aplicación a consultar más reseñas e incitan a estos a comentar su experiencia.
- **Módulo `rooms`:** contiene la lógica ligada a la gestión de habitaciones. Recoge las páginas, tablas, servicios e interfaces necesarios para mostrar las habitaciones de la casa rural, así como el detalle de estas.
- **Módulo `shared`:** este módulo especial tiene la misión de recoger diversos componentes que pueden ser reutilizados a lo largo de la aplicación, ya que no están atados a una sección en particular. Entre ellos, encontramos el `navbar`, el `footer`

o el `loader` de la página. También cuenta con el servicio que gestiona las comunicaciones con la *API* de *Google*, usado a lo largo de la aplicación para facilitar las reseñas y los mapas.

- **Módulo `user`:** presenta páginas propias de los usuarios registrados, como el panel de usuario, que contiene información como el histórico de reservas.
- **Carpeta `validators`:** este directorio especial no es un módulo, pues recoge en un archivo de *TypeScript* los diversos validadores a emplear en los formularios reactivos que hay en la página. Gracias a este fichero, se lanzan ciertos errores, comprueba patrones, entre otras cosas.

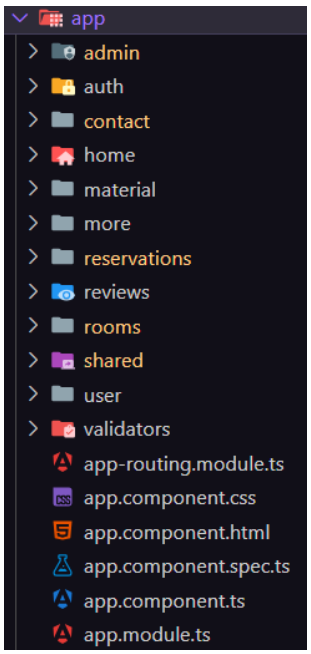

Figura 6.3: Estructura de la carpeta `app` del proyecto *Angular*

6.1.2. *Routing* y *Guards*

Una vez desgranada la arquitectura modularizada propia de los proyectos *Angular*, concretamente la del proyecto presentado, pasamos a comentar algunos conceptos interesantes, como el del enrutamiento y protección de rutas.

Como se comentaba en la sección anterior, en el proyecto se establecen las rutas globales en el archivo `app-routing.module.ts`. Define una lista de rutas, y para cada una de ellas se establece lo siguiente:

- La ruta propiamente dicha, definida con la propiedad `path`.

- El módulo que carga al acceder a dicha ruta, en este caso con *lazy load* o carga perezosa. Dicho módulo tendrá otro archivo de configuración de rutas para su enrutado local.

- Otros elementos, como `canActivate, canMatch` y `data` para establecer *guards*, así como propiedades especiales como `redirectTo y pathMatch`.

Se incluye un fragmento de la configuración global de rutas, en el que podemos observar que al acceder a la ruta `user`, se cargará el módulo de usuarios, siempre y cuando se respete lo establecido en el `AuthGuard`. Se brindan parámetros (en este caso los roles válidos) para el *guard*.

Además, se establece algo más abajo la configuración para las rutas vacías o que no coincidan con las rutas establecidas en el fichero. En este caso se redirige a la ruta `home`.

```
{
  path: 'user',
  loadChildren: () => import('./user/user.module').then(m => m.UserModule),
  canActivate: [AuthGuard],
  canMatch: [AuthGuard],
  data: { roles: ['Administrador', 'Usuario'] }
},
{
  path: '',
  redirectTo: 'home',
  pathMatch: 'full'
},
{
  path: '**',
  redirectTo: 'home'
}
```

Figura 6.4: Ejemplo de definición de rutas del proyecto *Angular*

Por último, expongo los dos *guards* empleados para proteger ciertas rutas en la aplicación: `AuthGuard y PublicGuard`. Antes de explicar la función que desempeña cada uno de ellos, es importante entender qué es un *guard*, así como ciertos matices ligados a los roles de la aplicación.

Un *guard* en un proyecto de *Angular* es un servicio que permite controlar el acceso a las rutas de una aplicación, actuando como un *middleware* antes de que se cargue una ruta. Básicamente, ayuda a controlar ciertos accesos en base a la seguridad y autenticación que se quiere establecer en la aplicación.

Como se menciona varias veces en esta memoria, la idea de la aplicación es que cualquier usuario, registrado o sin registrar, pueda acceder a la gran mayoría de funcionalidades de la página, a excepción de los paneles especiales.

Debido a esto último, se crea el `AuthGuard`, que comprueba que el usuario esté autenticado como usuario registrado para permitirle acceder a la ruta `/user`, o como administrador, dejándole acceder tanto a `/user` como a `/admin`. En caso de que un usuario sin iniciar sesión trate de acceder mediante la *URL* a estas rutas, el *guard* se encarga de bloquear el acceso y redirigirle al `home`.

Por su parte, ante la necesidad de que un usuario ya registrado no pueda acceder de nuevo a las pestañas de registro e inicio de sesión, a menos que cierre sesión mediante el botón habilitado en el `navbar`, surge `PublicGuard`. Este servicio especial, si un usuario registrado o administrador trata de acceder vía *URL* a las rutas `/auth/login` o `/auth/register`, le redirigirá a su `home`.

6.1.3. Entornos e Internacionalización

Para cerrar las explicaciones relativas a la implementación de *Angular*, quería comentar brevemente algunos añadidos, como son la internacionalización de la aplicación y la configuración de los entornos.

En primer lugar, a pesar de no contemplar la internacionalización total de la aplicación *web*, como se establece en las decisiones iniciales, se ha optado por investigar y dejar preparada la configuración requerida para ello.

En *Angular* la internacionalización recibe el nombre de `i18n`, al ser el proceso que trata de adaptar una aplicación para que pueda ser utilizada en diferentes idiomas y regiones.

Para ello, se realiza una configuración inicial que determina los idiomas con los que se va a trabajar. En mi caso, me decanté por el español como base y el inglés como idioma alternativo. Como se comentaba en secciones anteriores, se generaron los ficheros `.xlf` encargados de realizar este mapeo.

Posteriormente, hay que realizar un marcado de texto que permita la traducción, incorporando a las etiquetas *HTML* ciertos elementos que identifiquen cada *tag*, para su inclusión en los ficheros previamente mencionados.

Luego, mediante el *Angular CLI*, se actualizan los ficheros de traducción, y se debe de realizar la traducción de cada fragmento de información que se establece en ellos. Esta tarea la suele llevar a cabo un empleado especializado en las empresas. Al ser un trabajo tedioso, no he contemplado la traducción de toda la página, pero he realizado la configuración inicial, mostrando un ejemplo. Para ello, he utilizado el título de la página de habitaciones, cuya traducción se configura en el archivo `messages.en.xlf`, como se expone en la figura 6.5.

Se puede observar como mediante las etiquetas `source` y `target` se van realizando las traducciones oportunas. Cada fragmento a traducir está identificado con un `id` y ubicado en cierto archivo `.html` del código fuente.

Por último, se debe de lanzar la aplicación seleccionando la configuración deseada del idioma, usando una versión más detallada del clásico `ng serve`. En condiciones normales, se usaría el comando `ng serve -configuration=en`, de forma que se emplearía el fichero `.xlf` para la traducción de las etiquetas. En mi caso, al emplear el comando `ng serve -ssl true -ssl-key ./certs/key.pem -ssl-cert ./certs/cert.pem` para el lanzamiento estándar de la aplicación, con intención de utilizar los certificados, el comando necesario para un despliegue en inglés sería `ng serve -configuration=en -ssl true -ssl-key ./certs/key.pem -ssl-cert ./certs/cert.pem`. De esta manera, pasaríamos de ver el título de ejemplo como muestra la figura 6.6, a ver el resultado expuesto en la figura 6.7.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="es" datatype="plaintext" original="ng2.template">
    <body>
      <trans-unit id="425889281933385551" datatype="html">
        <source>Página de Habitaciones</source>
        <target>Rooms Page</target>
        <context-group purpose="location">
          <context context-type="sourcefile">src/app/rooms/pages/rooms-page/rooms-page.component.html</context>
          <context context-type="linenumber">1,3</context>
        </context-group>
        <note priority="1" from="description">Título de la página</note>
      </trans-unit>
    </body>
  </file>
</xliff>
```

Figura 6.5: Ejemplo de traducción mediante archivos `.xlf`

Figura 6.6: Título de la Página de habitaciones en español

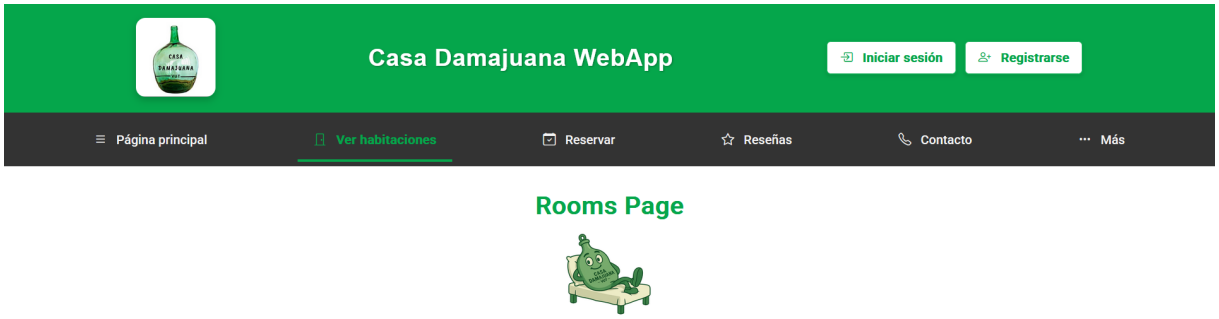

Figura 6.7: Título de la Página de habitaciones en inglés

Siguiendo el hilo referido a los lanzamientos alternativos de la aplicación, es importante mencionar los diversos entornos contemplados. Los entornos o *environments* se gestionan con archivos de configuración que permiten adaptar el comportamiento de una aplicación según el entorno en el que se está ejecutando. Los más conocidos son desarrollo (*development*) y producción (*production*). Estos entornos permiten configurar diferentes valores

para variables dentro de la aplicación, como *URLs* de *API* o claves de acceso, adaptándose a las necesidades de cada fase del ciclo de vida de la aplicación (desarrollo, pruebas, producción).

En el proyecto, he configurado como entorno predeterminado el de desarrollo, mediante la edición del archivo `angular.json`. Como entornos adicionales contemplados están los relativos al idioma, establecidos en `environment.es.ts` y `environment.en.ts`. Su lanzamiento concreto se realiza agregando ciertos argumentos a `ng serve`, como se exponía anteriormente.

Estos archivos de entorno contienen ciertas claves de *API*, rutas base como la del *backend*, etc. De esta forma, esta información sensible y variante no se verá reflejada en el código fuente principal, haciendo de la aplicación algo más escalable, mantenible y seguro.

6.2. *Spring Boot*

En la capa del *backend*, se ha empleado *Spring Boot* como tecnología principal que permita la comunicación del *frontend* con ciertos *endpoints* que permitan la interacción con la base de datos, mediante operaciones variopintas. Por tanto, ha permitido desarrollar una *API REST* robusta, modular y mantenible.

En esta sección, se explicará brevemente la distribución y organización seguida en el subproyecto realizado en *Spring Boot*, haciendo especial hincapié en *Spring Security*.

Antes de comenzar con la explicación general de los diversos paquetes y clases, es esencial destacar otros elementos presentes en el proyecto de *Spring Boot*. Por ejemplo, un archivo de extrema importancia es el `pom.xml`, un componente central en los proyectos *Spring Boot* construidos con *Maven*. Define la configuración del proyecto, las dependencias, los *plugins* y las configuraciones de compilación. En mi caso, almacena dependencias como las necesarias para el servicio de correos, los *JWT* o *Spring Security*.

Por otro lado, tenemos `application.properties`, un archivo de configuración clave que se utiliza para personalizar diversos aspectos de la aplicación. En el proyecto, se definen en este fichero ciertas *url* base, claves de *API*, puertos a emplear, etc. De esta manera, esta información secreta y sensible queda almacenada de forma segura, sin usarse en plano en el código fuente.

Por último, un elemento a destacar es `keystore.p12`, ubicado en la misma ruta que `application.properties`. Este elemento especial se emplea para el lanzamiento en *https* del *backend*, como explicaré en la sección 6.6, dedicada a la seguridad.

6.2.1. Paquetes y Clases

En *Spring Boot*, un paquete (*package*) es una forma de organizar clases relacionadas, similar a una carpeta en un sistema de archivos, mientras que una clase (*class*) es una plantilla o plano para crear objetos. *Spring Boot* utiliza estos conceptos para estructurar las aplicaciones en capas lógicas, como las capas de controlador, servicio, repositorio y modelo.

En mi caso, se han dividido las responsabilidades en una serie de paquetes lógicos (que comienzan con `com.company.casadamajuana`):

- El paquete `backend` contiene la clase principal que inicia la aplicación con `@SpringBootApplication`.
- El paquete `backend.config` incluye clases de configuración como `ConfigSecurity`, donde se define el `SecurityFilterChain`, el `PasswordEncoder`, y se gestiona la configuración *CORS* (*Cross-Origin Resource Sharing*) y *JWT* (*Json Web Tokens*).
- El paquete `backend.controllers` alberga los controladores *REST* (*Representational State Transfer*) que exponen los diferentes *endpoints* públicos. Por ejemplo, `ReservaRestController` maneja operaciones relacionadas con reservas como la creación, consulta, actualización o eliminación de las mismas.
- El paquete `backend.model` contiene las entidades *JPA* (*Java Persistence API*) que representan las tablas de la base de datos. Por ejemplo, la clase `Reserva`, que incluye atributos como fechas, usuario asociado, información de pago y estado de la reserva, entre otros.
- El paquete `backend.model.dao` almacena las interfaces de persistencia que extienden de `CrudRepository`, permitiendo ejecutar consultas *SQL* a través de *JPQL* (*Java Persistence Query Language*). Por ejemplo, `IReservaDao` define métodos para buscar reservas por fechas o por usuario/*email*.
- El paquete `backend.repository` define interfaces de persistencia que extienden de `JpaRepository`, permitiendo por ejemplo la búsqueda de usuarios por *email*.
- El paquete `backend.request` modela la forma que tendrán ciertas solicitudes, como las de *login* o contacto, definiendo atributos a solicitar como el *email*.
- El paquete `backend.response` define clases que encapsulan las respuestas *REST* personalizadas, siguiendo una estructura común para *status*, mensaje y datos.
- El paquete `backend.service` contiene interfaces y clases de servicio donde se implementa la lógica de negocio. Por ejemplo, `IReservaService` y su implementación manejan validaciones, operaciones con *PayPal* y llamadas al repositorio.
- El paquete `backend.security` agrupa clases auxiliares relacionadas con la autenticación y autorización, como el filtro *JWT* o la utilidad para generar y validar *tokens*.

De esta manera, queda ilustrada la organización en paquetes seguida para el proyecto en la capa *backend*.

Figura 6.8: Estructura de paquetes del proyecto *Spring Boot*

6.2.2. *Spring Security*

Una vez comentada la disposición de las diversas clases y su agrupación en paquetes, me gustaría explicar brevemente la configuración de seguridad implementada en *Spring Boot*.

Para proteger los *endpoints* de la *API*, se ha empleado *Spring Security* junto con autenticación basada en *tokens JWT*. La configuración se encuentra centralizada en la clase `ConfigSecurity`, donde se especifican:

- Las rutas públicas (como el inicio de sesión o creación de usuarios) y las rutas protegidas, con distintos niveles de acceso según el rol (usuario o administrador).
- La política de sesiones sin estado (*stateless*) adecuada para *APIs REST*.
- La integración del filtro `JwtAuthenticationFilter` antes de `UsernamePasswordAuthenticationFilter`, lo que permite validar el *token JWT* en cada petición.
- La codificación de contraseñas mediante `BCryptPasswordEncoder`.
- La configuración *CORS* para permitir peticiones desde el *frontend* (en este caso corriendo en `https://localhost:4200`), así como establecer los métodos y cabeceras *HTTP* permitidas.

Los *endpoints* se han protegido analizando meticulosamente si es necesaria su consulta por parte de cada tipo de usuario, limitando el acceso a ciertas funcionalidades a los usuarios con menos privilegios. Los detalles de estas restricciones se pueden consultar en la sección 6.5.

Como se indicaba previamente, para poder emplear las herramientas del módulo *Spring Security*, es necesario configurar correctamente el `pom.xml`.

Este enfoque permite un sistema seguro, manteniendo un equilibrio entre usabilidad y protección frente a ataques comunes como el *Cross-Site Request Forgery* (*CSRF*) y accesos no autorizados.

6.3. Base de datos relacional

La aplicación *web* desarrollada se apoya en una base de datos relacional *MySQL* (*MariaDB*) para la persistencia de la información crítica del sistema. La elección de un modelo relacional se fundamenta en la naturaleza estructurada y bien definida de los datos manejados (usuarios, reservas y espacios), así como en la necesidad de garantizar integridad referencial y soporte robusto para transacciones.

El diseño de la base de datos sigue una estructura normalizada y está orientado a representar las entidades fundamentales del sistema y sus relaciones. Se han definido claves primarias, foráneas e índices para optimizar el acceso a los datos y garantizar la coherencia de la información, como bien se explica en capítulos anteriores.

A lo largo de esta sección se analizan aspectos específicos como la indexación utilizada para mejorar el rendimiento de las consultas más frecuentes, y se discute el uso de disparadores (*triggers*) como mecanismos automáticos para desencadenar acciones internas en el sistema de gestión de bases de datos (SGBD).

La existencia de esta sección resulta relevante de cara a ampliar la información brindada en capítulos anteriores, focalizándose en temas de mejoras de implementación mediante los elementos de optimización comentados.

Antes de abordar los temas de indexación y disparadores, considero relevante estudiar el motor de almacenamiento a emplear para cada una de las tablas, así como el formato más oportuno. De esta forma, a lo largo de la sección se expondrán numerosas decisiones ligadas al diseño físico, con la intención de tratar de optimizar y tener presente la escalabilidad de la base de datos.

Por defecto, las tablas emplean `InnoDB` como motor de almacenamiento. Como bien sabemos `InnoDB` soporta transaccionalidad (muy importante para el proyecto, como se explica en la sección 6.7), integridad referencial, bloqueo a nivel de fila, entre otras ventajas. En cambio, otras opciones como `MyISAM` o `Memory`, no ofrecen una gestión de claves foráneas o transacciones *ACID* (*Atomicity-Consistency-Isolation-Durability*). A pesar de que `MyISAM` y `Memory` pueden resultar más ligeros, y por ende apropiados para otro tipo de situaciones, merece la pena mantener `InnoDB` en un proyecto que requiera de concurrencia, integridad referencial y persistencia garantizada.

En cuanto a los formatos que ofrece `InnoDB`, `REDUNDANT` no suele recomendarse, ya que está bastante obsoleto. Por su parte, `COMPRESSED` puede servir para ahorrar espacio en tablas grandes, mientras que `COMPACT` puede ser útil para reducir el tamaño total siempre y cuando los campos no sean extensos. Por lo tanto, el formato `COMPRESSED` comprime páginas de datos e índices en el caso de tener demasiados datos y necesitar ahorrar espacio de disco, que no es el caso. `COMPACT` ahorra espacio en datos pequeños, pero en mi caso cuento con numerosos atributos `VARCHAR` largos. En cambio, `DYNAMIC` almacena datos de tipo `VARCHAR` fuera de la página principal (con punteros), resultando más eficiente y reduciendo la fragmentación. Básicamente permite optimizar la persistencia de información variable.

Atendiendo a cada una de las tablas de la base de datos, podríamos justificar el uso de `InnoDB DYNAMIC`.

- En la tabla `reserva`, esta decisión es ideal debido a la presencia de claves foráneas, índices (tanto autogenerados como agregados, como veremos en esta misma sección) y campos `VARCHAR`.

- En la tabla `usuario`, es conveniente su uso ya que existe, además de la clave primaria, otro atributo `UNIQUE`.

- En la tabla `espacio` podrían haberse empleado otros formatos, pero al tener un tamaño reducido, no merece la pena alterar la configuración por defecto.

En definitiva, el uso de `InnoDB DYNAMIC` es óptimo en la versión inicial en la que nos encontramos, así como de cara al futuro, debido al buen rendimiento, escalabilidad, compatibilidad con indíces y claves foráneas y correcta gestión del espacio de forma eficiente. Cabe destacar que no se ha considerado la posibilidad de realizar un análisis *benchmark*, con herramientas como `mysqlslap`, debido a la poca madurez del proyecto, que cuenta con una base de datos aún muy reducida y poco poblada. Además, en base a los conocimientos adquiridos en asignaturas de la carrera, se puede afirmar que normalmente las configuraciones por defecto suelen resultar más adecuadas.

6.3.1. Indexación

El sistema utiliza técnicas de indexación para optimizar el rendimiento de consultas frecuentes, en especial aquellas relacionadas con la autenticación de usuarios y la gestión de reservas, al utilizar tablas más voluminosas.

Las claves primarias (`PRIMARY KEY`) de las tablas (`id` en las tablas `usuario`, `reserva` y `espacio`) generan automáticamente índices únicos, permitiendo accesos rápidos por identificador. Por tanto, no será necesario agregarlos manualmente.

Los atributos marcados como `UNIQUE` también presentan un índice generado de forma automática, facilitando las consultas que los emplean en modificadores como `WHERE` u `ORDER BY`. En este caso, encontramos `email` como atributo `UNIQUE` adicional en la tabla `usuario`. Al realizar búsquedas que incluyan en el `WHERE` cualquiera de las `PRIMARY KEY` o atributos `UNIQUE` mencionados, se utiliza el índice generado por defecto, como se puede comprobar al agregar `EXPLAIN` al comienzo de la consulta, mostrando que se realiza una búsqueda directa. De hecho, muestra que `type=const`, que es la mejor categoría de acceso posible. El optimizador sabe que como `email` es único, solo puede haber una fila como mucho, así que lo resuelve instantáneamente.

De forma similar, la columna `usuario_id` en la tabla `reserva`, al ser una clave foránea, genera automáticamente un índice que facilita las operaciones de `JOIN` entre `usuario` y `reserva`, reduciendo el tiempo de respuesta de consultas relacionadas con el historial de reservas. De hecho, este índice se emplea al realizar consultas del tipo `SELECT * FROM reserva WHERE usuario_id = 1;` ya que al agregar `EXPLAIN` al comienzo de la consulta, se muestra con los valores `possible_keys` y `key` que se emplea el índice, indicando con `type=ref` y `rows=8` (valor bajo) que el optimizador está usando este índice de forma

eficiente. En definitiva, permite buscar filas específicas, en lugar de tener que realizar un *full scan*.

Tocaría entonces valorar la inclusión de índices de forma manual en otras columnas que se utilicen para ajustar los criterios de ordenación o selección de ciertas consultas. Cabe destacar que en un proyecto a esta escala, en el que la afluencia de clientes no será tan masiva al comenzar, hay que valorar si merece la pena agregar ciertos índices, ya que una sobreindexación puede resultar contraproducente, lastrando los tiempos de consulta.

Un índice a considerar sería aquel que incluya, en la tabla `reserva`, tanto la fecha de inicio (`fecha_inicio`) como la fecha de finalización (`fecha_fin`). Para ello, podríamos ejecutar una sentencia del estilo de `CREATE INDEX idx_reserva_fechas ON reserva (fecha_inicio, fecha_fin);` que genere un índice compuesto para ambas columnas. En este caso, al utilizar el comando `EXPLAIN` al comienzo de una consulta como `EXPLAIN SELECT * FROM reserva WHERE fecha_inicio >= '2025-06-01' AND fecha_fin <= '2025-06-30';`, parece que el optimizador considera mejor recorrer la tabla al completo, ya que muestra con `possible_keys` la existencia del índice, pero finalmente descarta su uso, como muestra con `key=NULL` y `type=ALL`. Probablemente, esto se debe a que la tabla `reserva` no tiene aún una magnitud suficiente como para incluir este índice y que se aproveche. Como la casa rural no tendrá una cantidad inmensa de reservas a pesar del paso del tiempo, podríamos ahorrarnos la inclusión de este índice de momento, dejando la posibilidad de incluirlo conforme crezca el negocio y la afluencia de clientes. Si establecemos un índice individual para alguna de las fechas, ocurre algo similar, pues no se emplea en consultas por rangos. En cambio, en consultas para una fecha concreta, se emplea el índice. En definitiva, debido al bajo volumen de datos actual, no merece la pena indexar estas columnas.

En la propia tabla `reserva`, hay otras columnas que sugieren la inclusión de un índice, al usarse en el filtrado de consultas con los operadores `WHERE` y `ORDER BY`. Por ejemplo, podría resultar interesante acelerar consultas que utilicen un filtrado por estado de la reserva. Sin embargo, la columna `estado` tiene una cardinalidad baja, al contar únicamente con los valores `ACTIVA` y `CANCELADA`. Por lo tanto, un índice simple no ayudaría en absoluto. En cambio, utilizar la columna `estado` junto con `usuario_id` para formar un índice compuesto que ayude en la búsqueda de reservas activas o canceladas por usuario, podría ser más fructífero. Por ejemplo, al realizar pruebas con `EXPLAIN`, el optimizador usa este índice, reduciendo el número de tuplas consultadas. De hecho, el optimizador decide, en consultas del estilo `EXPLAIN SELECT * FROM reserva WHERE estado = 'ACTIVA' AND usuario_id = 1;` utilizar el índice compuesto. Si el volumen de datos crece bastante, dando lugar a diversas reservas con estados distintos por parte de cada usuario, el índice puede ayudar en este tipo de búsquedas.

También se plantea el uso de un índice en la columna `email` de la tabla `reserva`, debido a que una reserva se puede realizar a nombre de alguien con un correo electrónico distinto al de la cuenta de usuario desde la que se efectúa. También sería útil si en un futuro se permitiera a los usuarios anónimos, de alguna manera, consultar reservas por *email*, agregando verificaciones por correo. En definitiva, creando el índice con `CREATE INDEX idx_reserva_email ON reserva (email);` y probando una consulta que filtre

con `WHERE` empleando el campo indexado, se obtiene un gran aprovechamiento del índice, reduciendo las filas consultadas a pesar de no contar con demasiadas reservas.

Algo similar ocurre con la indexación, esta vez con un índice `FULLTEXT`, de la columna `mensaje` en la misma tabla. Utilizando consultas que se beneficien del índice, como `SELECT * FROM reserva WHERE MATCH(mensaje) AGAINST ('+cambio horario' IN BOOLEAN MODE);`, se logra un aprovechamiento a tener en cuenta. El operador `EXPLAIN` muestra con `type=fulltext` el uso de este índice, y corrobora que se mejora el rendimiento mediante valores de estimación con `rows` muy bajos.

En cuanto a la tabla `usuario`, al contar con los campos `id` y `email` ya indexados, las posibilidades de mejora son reducidas. Se podría plantear la indexación de `rol`, al tratarse de un `ENUM`. Sin embargo, sucede algo parecido a lo estudiado con `estado` en la tabla `reserva`, pues el campo tiene una cardinalidad baja. Además, se sabe que en la aplicación la gran mayoría de usuarios serán corrientes, por lo que apenas se filtrará.

Por su parte, en la tabla `espacio`, que cuenta por defecto con el índice en su clave primaria, no tendría mucho sentido indexar algo más. Al tratarse de una tabla con menos de veinte tuplas, que rara vez crecerá, no tendría mucho sentido agregar índices. Además, las consultas ligadas a esta tabla suelen ir encaminadas al uso del `id`, columna que ya está indexada por defecto. La columna `tipo`, que cuenta con cinco posibles valores, sería interesante, pero de nuevo se descarta su indexación por el reducido tamaño de los datos.

Por resumir, además de los índices existentes en las columnas `PRIMARY KEY`, `UNIQUE` y `FOREIGN KEY`, se ha considerado agregar algún índice únicamente en la tabla `reserva`. Las tablas `espacio` (al ser demasiado pequeña y estática, ya que no crecerá) y `usuario` (al tener indexados los campos más relevantes de cara a consultas) no son interesantes respecto a agregar índices. En cuanto a `reserva`, consideramos:

- Índice compuesto en `usuario_id` y `estado`, para consultas que filtren empleando ambos campos, resultando más eficiente que el índice autogenerado en la primera columna mencionada.
- Índice `FULLTEXT` en `mensaje`, muy útil en consultas que empleen `MATCH`.
- Índice simple en `email`, al tratarse de un campo no indexado por defecto y utilizarse en consultas para el filtrado de reservas.

Para decidir si su inclusión merece o no la pena, además de las comprobaciones con `EXPLAIN`, se valorará el tamaño de los índices. Se descarta la realización de un estudio intensivo con herramientas como `mysqlslap`, debido a que no se espera un crecimiento gigante en poco tiempo, sino escalar poco a poco la base de datos. Además, habría que generar datos sintéticos para simular la presencia de tablas pobladas, lo cual no interesa en absoluto en este Trabajo Fin de Grado. Antes de agregar índices adicionales, se observa que en la tabla `reserva` hay un `Data_length` de `16,384 bytes`, así como un `Index_length` de `16,384 bytes`.

Comenzamos con el índice compuesto para `usuario_id` y `estado`. Al agregarlo, el `Data_length` obviamente se mantiene constante, mientras que el `Index_length` pasa a

`32,768 bytes`, aumentando en unos `16,384 bytes`. Eso supone un aumento de unos `16 KB` aproximadamente, lo que resulta despreciable a escala pequeña y perfectamente asumible en sistemas más grandes. Básicamente, como en `InnoDB` cada página suele ocupar `16 KB`, se agrega una página más de índice. En cuanto a las modificaciones del índice debido a operaciones sobre `reserva`, es cierto que el volumen de `INSERT` y `UPDATE` (para la cancelación) pueden resultar pesados, pero casi nunca habrá `DELETE`.

La agregación del índice simple en `email`, tras eliminar el estudiado anteriormente, supone un `Index_length` de `32,768 bytes` de nuevo, ocupando de nuevo una página adicional para índices.

Lo mismo sucede al eliminar los índices previos y probar con el `FULLTEXT`, ya que se reservan otros `16 KB` para el índice.

Agregando los tres índices en la tabla `reserva`, el almacenamiento empleado en índices pasa de los `16,384 bytes` iniciales a `65,536 bytes`. Esto supone un gasto de `64 KB`, contando los generados automáticamente y los agregados. Teniendo en cuenta que cada índice tiene una finalidad distinta, son mutuamente exclusivos y el gasto de `64 KB` no es algo escandaloso, considero aplicar los tres.

Además, han sido los tres índices que he observado que el optimizador decide utilizar, descartando otros índices que, a pesar de sobre el papel resultar interesantes, a la hora de la verdad no resultaban tan eficientes.

En definitiva, aunque la mejora de rendimiento no se podrá apreciar más allá de que el optimizador utilice estos índices ya que los considera eficientes, según vayan realizándose más reservas, se notará la diferencia. A mayores, cuando la aplicación vaya creciendo, se podría considerar la indexación de otros campos estudiados durante este breve análisis, que no han sido agregados ya que ni siquiera se empleaban, como se comprobaba con el uso de `EXPLAIN`.

La indexación ha sido diseñada con criterio, evitando una sobreindexación innecesaria, que podría penalizar el rendimiento de inserciones y actualizaciones. Por ello, según fuera creciendo la base de datos, podríamos plantear nuevos índices para mejorar el rendimiento, pero encontrándonos en una etapa inicial del proyecto, no merece la pena sobrecargar con índices la base de datos.

6.3.2. Disparadores (*triggers*)

Como bien sabemos, un *trigger* en *SQL* es un procedimiento almacenado que se ejecuta automáticamente en respuesta a ciertos eventos en una base de datos, como la inserción, actualización o eliminación de datos en una tabla.

A pesar de que la lógica de negocio se encuentra centralizada en el *backend*, desarrollado con *Spring Boot*, resulta interesante agregar una capa más que garantice la integridad de la base de datos. Como se menciona a lo largo del documento, se realizan validaciones exhaustivas en en lado cliente y lado servidor, que arrojarán errores en caso de vulnerar las restricciones impuestas, evitando consultas indebidas en la base de datos. De todas formas, toda adición de seguridad es beneficiosa, por lo que considerar la agregación de disparadores que aseguren la ocurrencia (o no ocurrencia) de ciertas consultas resulta en

una mejora significativa de la aplicación, complementando las medidas implementadas en las demás capas.

En la tabla `espacio`, a pesar de considerarse poco variable, debido a que rara vez se requerirá realizar cambios sobre elementos que representan entidades físicas estáticas, se han incluido una serie de disparadores. Por ejemplo, se utilizan los *triggers* `validar_tipo_espacio_insert` y `validar_tipo_espacio_update` para garantizar que al insertar o modificar un espacio, el `tipo` tome uno de los valores permitidos (`Aseo, Cocina, Dormitorio, Salon` y `Especial`). Además, gracias a `validar_capacidad_espacio_insert` y `validar_capacidad_espacio_update` se evita la inserción de capacidades negativas. Por último, se aplican los disparadores `validar_nombre_espacio_insert` y `validar_nombre_espacio_update` para evitar incluir nombres formados por una cadena vacía o por espacios en blanco, usando el operador `TRIM`, que permite suprimir dichos espacios.

En cuanto a la tabla `reserva`, se emplean *triggers* como `before_insert_reserva` y `before_update_reserva` que comprueban que no existan reservas cuyas fechas se solapen con las nuevas a insertar, así como que las fechas sean coherentes, es decir, que `fecha_inicio` no sea posterior a `fecha_fin`. Por otro lado, uso los disparadores `validar_estado_reserva_insert` y `validar_estado_reserva_update`, que comprueban que las reservas únicamente tengan estados `CANCELADA` o `ACTIVA`. Para culminar, empleo `validar_precio_reserva_insert` y `validar_precio_reserva_update` con la intención de garantizar que las cantidades sean positivas.

En la tabla `usuario`, se incluyen `validar_rol_usuario_insert` y `validar_rol_usuario_update`, para garantizar que los únicos roles insertados sean `Administrador` o `Usuario`.

Con los *triggers* mencionados, se obtiene una protección adicional sobre los atributos que podrían tomar valores indeseados con una configuración base.

En definitiva, los disparadores considerados aumentan la fiabilidad de la base de datos, logrando asegurar un grado de integridad mayor en todo momento, al complementarse con las validaciones y la lógica de negocio expuesta en el *frontend* y *backend*.

6.4. Servicios externos

Una vez explicados ciertos conceptos ligados a la implementación en las diversas capas que componen la aplicación, resulta relevante exponer algunos de los servicios externos que se emplean en esta. Los servicios externos seleccionados no se emplean sin motivo alguno, sino para dar una mejor experiencia a los usuarios de la aplicación en algunos ámbitos.

Por ejemplo, no tendría sentido alguno elaborar un sistema de pago autónomo, ya que tendría más vulnerabilidades, complicaciones al tener que manejar dinero real, etc. Para ello, utilizamos *PayPal* [21] como servicio para mejorar los pagos. Los servicios de esta plataforma de pagos se emplean tanto en la capa *frontend* como en el *backend*. De esta manera, se asegura una gestión de los pagos segura y eficiente, delegando parte de la

responsabilidad a un gigante del sector.

En el *frontend* se consulta la *API* de *PayPal* para lograr hacer funcionar ciertos componentes. Logramos renderizar un botón de pago con *PayPal*, así como un botón que despliega un formulario para realizar transacciones con tarjeta. Gracias a estas consultas en el proyecto de *Angular*, se consigue iniciar sesión en una cuenta *PayPal* o verificar la validez de la tarjeta para realizar los pagos. Cabe destacar que el primero de los botones de pago, abre una ventana emergente para el inicio de sesión en *PayPal*, mientras que el segundo renderiza un formulario dentro de la propia aplicación. Ambos son totalmente seguros, ya que el formulario lo genera *PayPal* dinámicamente, por lo que ningún dato sensible es gestionado o almacenado por la aplicación *web*. De hecho, lo que se renderiza es un `iframe`, es decir, un elemento `HTML` que permite insertar un documento `HTML` dentro de otro. Básicamente, crea un espacio en una página donde se carga y muestra contenido de otro sitio *web*, en este caso un formulario de *PayPal*. En definitiva, no se almacena ni gestiona información sensible, ya que el formulario está alojado y gestionado por parte de *PayPal*.

Por su parte, en el *backend*, se realizan numerosas consultas a los diversos *endpoints* que ofrece la plataforma. Al confirmar un pago desde el *frontend*, se sigue cierto flujo de comprobaciones hasta efectuar el pago. Este proceso se explica de forma detallada en la sección 6.7. Sin embargo, quiero mencionar algunas de estas consultas relevantes a la *API* de *PayPal*. Se exponen a continuación estas llamadas, comentando brevemente su misión:

- Se trata de obtener un *token* de acceso mediante una consulta a un *endpoint* terminado en `/v1/oauth2/token`. Se agregan a esta petición tanto el *id* de cliente como el secreto proporcionado a la hora de crear la cuenta de desarrollador de la aplicación.

- Para verificar el estado del pago de la reserva, una vez obtenido el *token* para autenticarse (adjuntándolo como *Bearer Token*), se realiza una llamada a un *endpoint* acabado en `/v2/checkout/orders/`. Esta consulta permite obtener información acerca del pago realizado, para contrastarla mediante ciertas comprobaciones en el *backend*. De esta forma se logra comprobar si las cantidades a desembolsar, el destinatario o la divisa coinciden con las esperadas.

- Tras realizar las comprobaciones gracias a la información obtenida mediante la *API*, se realizan operaciones internas en el sistema (disponibilidad de fechas, transaccionalidad, persistencia de la reserva, etc.). Si todo sale como se espera, se realiza otra petición a un nuevo *endpoint* de *PayPal*. Al consultar el *endpoint* terminado en `/v2/checkout/orders/`, concatenando al final del *path* el *id* del pago y `/capture`, se logra capturar el pago, es decir, hacerlo efectivo.

- Adicionalmente, se realiza una cuarta petición a la *API* de *PayPal*, ya que en el caso de solicitar una devolución, el sistema obtiene el *id* del pago capturado, almacenado en la base de datos en la tupla de la reserva asociada. Realiza una petición a un *endpoint* terminado en `/v2/payments/captures/`, agregando a la *url* el *id* asociado y terminando con `/refund`. Para la cancelación de reservas se aplica la política

expuesta en la página, permitiendo la cancelación de reservas con un día de margen. Además, si se cancela en un plazo de siete días o menos, se aplica una penalización de 100 euros sobre el total.

Todas estas llamadas a la *API* de *PayPal* permiten integrar los servicios de esta compañía en la aplicación, obteniendo así una pasarela de pago moderna, efectiva y segura. Se complementa con las medidas de seguridad y transaccionalidad aplicadas en las capas de la aplicación.

Por otro lado, es necesario mencionar los servicios de *Google* que se integran en la página, brindando una experiencia inmersiva a los usuarios, que pueden consultar información de calidad sin necesidad de acceder a las páginas oficiales, pero sin renunciar a interactuar con componentes idénticos a los originales.

En etapas tempranas del proyecto, se pensó en agregar un sistema de reseñas y mapas autogestionado. Sin embargo, se pensó que estos servicios no serían utilizados de forma activa, ya que los usuarios tienden a emplear el sistema de reseñas y mapas de compañías conocidas como *Google*. La mayoría de clientes interesados en reservar u opinar, no tendrían en cuenta un sistema menor de una pequeña casa rural. Por tanto, se decidió tratar de comunicar la página *web* con las tecnologías mas usadas, en este caso las ofrecidas mediante *APIs* de *Google*.

Las *APIs* utilizadas son las siguientes:

- ***API* de *Google Places***, un servicio que permite obtener los datos del negocio, entre ellos las reseñas de los usuarios que han visitado la casa rural. Para ello, mediante las claves de *API* facilitadas por *Google Cloud Console* [18] y el *id* asociado al negocio, se obtiene cierta información de la casa rural. Dicha información se renderiza de forma atractiva empleando componentes propios, incluyendo enlaces al sitio oficial de *Google* para consultar el total de reseñas e incluso ofreciendo la posibilidad de con un *click* acceder a un formulario de valoración.

- ***Maps JavaScript API* de *Google***, otro servicio de la compañía estadounidense que permite integrar un mapa totalmente funcional en la aplicación. Para su correcta renderización, hay que realizar muchos ajustes, emplear las claves ya mencionadas y adaptar el mapa a las necesidades y estilos de la página. Tras este trabajo, se obtuvo un mapa adaptable y funcional, con ciertas personalizaciones, que permite visualizar los alrededores de la casa, e incluso acceder a *Google Maps* para guiar a los viajeros con un solo *click*.

- ***reCAPTCHA Enterprise API***, otro servicio que ofrece *Google*, en este caso para la detección de *bots*. A diferencia de los dos servicios mencionados previamente, este último se aplica tanto en el *frontend* como en el *backend*. Gracias a unas claves personalizadas para esta función, se renderiza el típico panel de *reCAPTCHA* [19] antes de poder enviar el formulario de contacto. Dependiendo de tus interacciones con la página, ciertas conductas e incluso la presencia de *cookies*, el servicio sospecha o no del usuario, planteándole un desafío previo al envío del formulario. La clave

pública del servicio se usa en la capa de *Angular*, renderizando el componente, mientras que en el *backend* está presente una clave privada. Al tratar de resolver el desafío, se envía un *token* junto con los datos de la petición de contacto hacia el *endpoint* de la *API REST* de *Spring Boot*, y desde el *backend* se consultará un *endpoint* de la *API* de *reCAPTCHA* que determinará la validez (o fracaso) del desafío, permitiendo (o no) el contacto.

Por tanto, los servicios de *Google* logran ofrecer una experiencia de usuario más centralizada, ofreciendo funcionalidades que se nutren de datos de esta compañía, en lugar de partir de cero con una implementación sin sentido, que apenas sería usada.

Por último, merece la pena destacar que a nivel de *backend*, si la petición de contacto de la que hablaba arriba fuera correcta, superando el desafío *reCAPTCHA* para evitar *spam*, se enviaría un correo a la bandeja de *Gmail* de la casa. Para ello, se emplea un servicio de correo basado en *JavaMail*, en el que se tematiza el mensaje, se incrustan los datos y mediante *SMTP* se envía el correo. Para lograr que se envíe a la bandeja de la casa, fue necesario generar una clave de aplicación desde la cuenta de *Gmail* [20], así como activar la verificación en dos pasos. De esta forma, se logra un envío de información a una bandeja de correo que de verdad use la administradora, ya que todos empleamos *Gmail* en nuestros dispositivos. De lo contrario, si se hubiera diseñado un servicio de correo autónomo, la persistencia en base de datos de los mensajes sería muy costosa, y la administradora estaría obligada a iniciar sesión en la página para consultar las dudas de los clientes.

En definitiva, la integración de diversos servicios externos con la aplicación desarrollada no solo supone un salto de calidad en los servicios ofrecidos, sino que demuestra que es posible hacer compatible la página desarrollada de forma autónoma con servicios ya maduros y experimentados.

Además, sigue una filosofía de delegación de trabajo en servicios externos que puedan gestionar ciertas situaciones mejor que tú, aligerando la aplicación, evitando problemas legales y burocráticos.

6.5. Autenticación y Autorización

Respecto a la autenticación y la autorización, el sistema implementa un esquema de seguridad basado en autenticación con *JWT* (*JSON Web Tokens*) y control de acceso por roles. Esta arquitectura se aplica tanto en el *backend* como en el *frontend*, permitiendo un flujo seguro y eficiente para gestionar las sesiones de los usuarios. Se hace uso de *cookies HTTP* seguras para manejar el *token* y protegerlo contra ataques *XSS*.

La seguridad en el *backend* se gestiona mediante *Spring Security*. La configuración define una política de sesiones sin estado (`SessionCreationPolicy.STATELESS`) y aplica un filtro personalizado `JwtAuthenticationFilter` que intercepta cada solicitud para validar el *token JWT* incluido en una *cookie* `HttpOnly`. EL flujo es el siguiente:

- Al realizar el inicio de sesión mediante `/auth/login`, se validan las credenciales usando un repositorio *JPA*.

- Si las credenciales son correctas, se genera un *token JWT* que incluye el *ID* de usuario, el correo electrónico y el rol del usuario.
- Este *token* se retorna en una *cookie* `HttpOnly` con una duración de 2 horas. Esta decisión trata de establecer un balance entre seguridad (evitando mantener sesiones demasiado extensas que puedan facilitar la entrada de atacantes) y usabilidad (al evitar que el usuario deba iniciar sesión cada poco tiempo). Básicamente, constituye un término medio entre las sesiones de aplicaciones bancarias (con vencimientos de apenas unos minutos) y aplicaciones que no manejan información sensible (con sesiones de duraciones extensas).
- En cada petición posterior, el filtro `JwtAuthenticationFilter` extrae y valida el *JWT* desde la *cookie*, configurando el contexto de seguridad para permitir o denegar el acceso según el rol.

El proceso completo se puede observar en el diagrama de secuencia 6.9 o en su versión ampliada en la figura B.4.

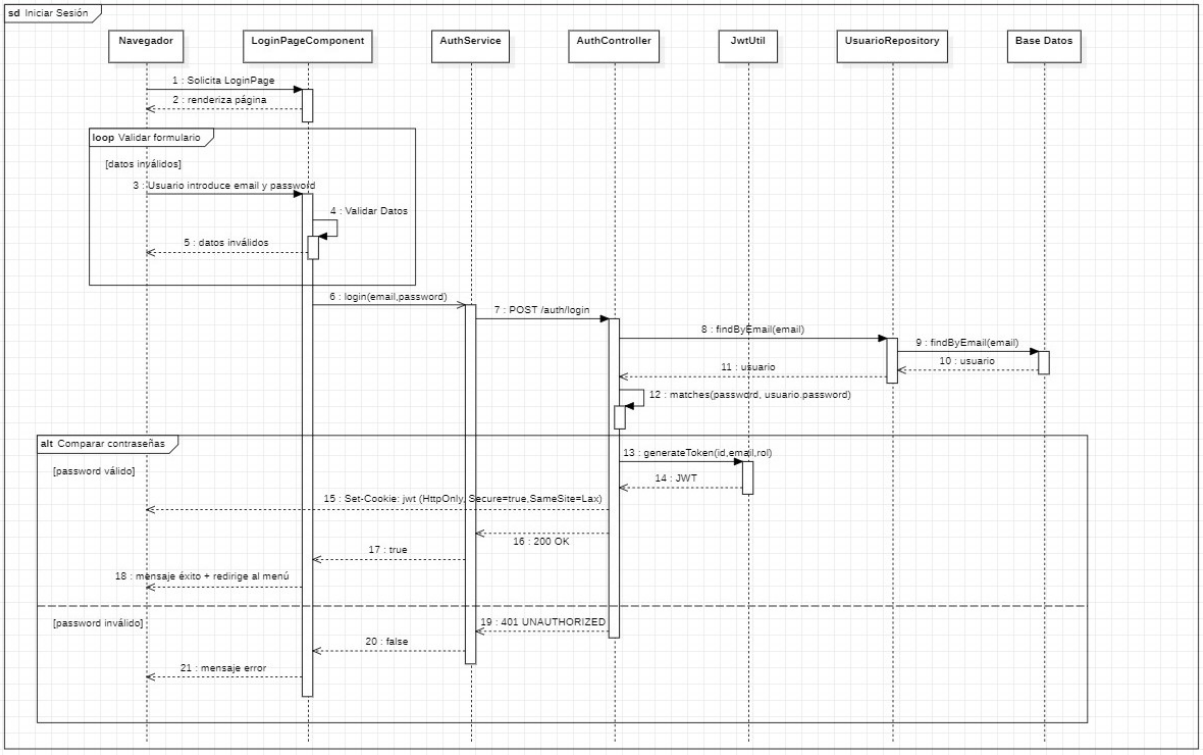

Figura 6.9: Diagrama de Secuencia de inicio de sesión

Podemos observar que se obvian algunas cosas, como que el usuario utiliza el navegador para interactuar con la aplicación, que en el caso de no encontrar un usuario en base de datos con ese correo saltará cierta excepción que abortará el proceso, etc.

La configuración de *endpoints* públicos y privados (según el rol) es la siguiente:

- ***Endpoints* públicos:** `/auth/login` (`POST`), `/v1/usuarios` (`GET-POST`), `/v1/espacios` (`GET`), `/v1/espacios/**` (`GET`), `/v1/reservas` (`GET-POST`), `/v1/contact` (`POST`).
- ***Endpoints* restringidos por rol:**
  - **Usuario** o **Administrador**: `/v1/usuarios/**` (`GET-PUT-DELETE`), `/v1/reservas/*` (`GET`), `/v1/reservas/filtro` (`GET`) `/v1/reservas/**` (`PUT`).
  - **Administrador**: `/v1/espacios` (`POST`), `/v1/espacios/**` (`PUT-DELETE`), `/v1/reservas/**` (`DELETE`).

Además, el *backend* expone un *endpoint* `/auth/me` que permite validar la sesión actual decodificando el *JWT* en la *cookie*, y un *endpoint* `/auth/logout` que invalida la *cookie*, cerrando sesión. Ambos son accesibles para usuarios autenticados.

En el cliente, desarrollado con *Angular*, no se guarda el *token JWT* directamente, sino que se maneja mediante *cookies* de tipo `HttpOnly`, lo que aumenta la seguridad del sistema frente a ataques de tipo *XSS*. El navegador se encarga automáticamente de enviar la *cookie* en cada solicitud al *backend*, al incluir en las peticiones esta información usando `withCredentials: true`. Se siguen los pasos citados a continuación:

- Al iniciar sesión, el *backend* responde con una *cookie* que contiene el *token JWT*.
- Las solicitudes posteriores al *backend* ya contienen automáticamente la *cookie*.
- Para comprobar si el usuario sigue autenticado, el *frontend* consulta el *endpoint* `/auth/me` y recibe información del usuario (`ID`, `correo`, `rol`).
- Las rutas en *Angular* se protegen mediante *guards*, que verifican si el usuario tiene un rol válido para acceder.

La configuración de rutas públicas y privadas (*routing*) es la siguiente:

- **Rutas públicas:** `/auth/login`, `/auth/register`, `/home/main`, `/rooms` (así como las rutas de cada habitación), `/reservations`, `/reviews`, `/contact`, `/more`. Se obvian las rutas relativas a páginas de éxito o error, ciertas redirecciones, etc. Nótese que las dos rutas que dan acceso a los servicios de autenticación sólo serán accesibles para usuarios sin autenticar. Si un usuario o administrador quiere acceder a ellas, primero debe cerrar sesión.
- **Rutas restringidas por rol:**
  - **Usuario** o **Administrador**: `/user`.
  - **Administrador**: `/admin`.

Este enfoque mejora la seguridad al evitar exponer el *JWT* en el almacenamiento local del navegador y asegura la escalabilidad del sistema al mantenerlo sin estado en el servidor.

6.6. Seguridad

En este apartado me enfocaré en comentar diversos mecanismos y configuraciones agregados al proyecto que permiten mantener un entorno más seguro, previniendo accesos no deseados, robos de información y demás vulnerabilidades. Iré tratando diversos temas ligados a la seguridad, comentando cómo se ha aplicado cada mecanismo en el proyecto.

En primer lugar, se realizan numerosas **validaciones de datos**, tanto en el lado cliente como en el lado servidor, reduciendo la carga de consultas erróneas con la primera capa y filtrando con la segunda por si se saltara la primera barrera.

En *Angular*, he utilizado formularios reactivos, que permiten más control y escalabilidad. Realizo tanto validaciones síncronas con validadores estándar proporcionados por *Angular* (datos requeridos, longitud mínima, etc.) como validaciones personalizadas. Entre las validaciones asíncronas y personalizadas, destacan aquellas que comprueban que coincidan la contraseña y la confirmación de esta, que exista el *email* o contraseña en la base de datos, que los correos sigan cierto patrón o que el correo a la hora de registrarse esté disponible.

Los formularios reactivos me permiten controlar los cambios que van ocurriendo en los campos que contienen, lanzando mensajes de error, desactivando botones si son inválidos y ahorrando constantes peticiones al *backend*. También se realizan comprobaciones en el cliente en lo relativo a las reservas, permitiendo seleccionar únicamente las fechas no pasadas y no reservadas, entre otros procedimientos, a pesar de que esto se valide también en el *backend* si alguien lograra sortear la primera verificación. En definitiva, estas validaciones mejoran la experiencia de usuario, a la vez que reducen la carga de trabajo del *backend*, sin llegar a suplirla.

Por su parte, en *Spring Boot* se realizan numerosas comprobaciones para evitar la introducción de datos inválidos en la base de datos, amenazando así su consistencia. Se verifica que el correo no esté ya almacenado en base de datos antes de registrar una cuenta, comprueba que las fechas sean consistentes en cuanto a formato y orden, así como que no se solapen con otras al hacer reservas, entre otras operaciones.

Además, en la base de datos existen una serie de *triggers* que, en caso de que se trate de hacer persistir algún dato que no cuadre con la lógica o formato esperado, se evite su almacenamiento.

Otro tema a tratar es el **control de acceso**, que se gestiona en la aplicación mediante herramientas como la distinción de roles en el *frontend* atendiendo al *JWT* (*JSON Web Token*) generado al iniciar sesión y los *guards*, así como con los filtros establecidos con *Spring Security* en el *backend*.

La autenticación y autorización, tratadas en el apartado inmediatamente anterior, son esenciales para evitar accesos indebidos a ciertas secciones mediante el cliente o *endpoints* con aplicaciones como *Postman*.

Se **previenen ataques comunes** como los expuestos a continuación:

- Ataques ***CSRF* (*Cross-Site Request Forgery*)**, o falsificación de solicitud entre sitios, un tipo de ataque en el que un atacante engaña a un usuario autenticado

en un sitio *web* para que realice una acción no deseada sin su conocimiento o consentimiento. En esencia, el atacante fuerza al navegador del usuario a enviar una solicitud a un sitio *web* que el usuario ya ha autenticado, aprovechándose de la confianza que el sitio *web* tiene en el usuario autenticado. Para mitigar el riesgo de sufrir estos ataques, se utilizan *JWT* mediante *cookies* marcadas con `SameSite=Lax` así como `HttpOnly=true`. Es importante destacar que `SameSite=Lax` funciona ya que al estar trabajando en local, `localhost:4200` y `localhost:8443` comparten *host*, aunque cambien los puertos. En cambio, si se realizara un despliegue real en entornos diferentes, habría que usar `SameSite=None`.

- Ataques ***XSS* (*Cross-Site Scripting*)**, puesto que *Angular* ofrece protección incorporada, escapando contenido interpolado, bloqueando inserciones de *scripts* o código en el *DOM* y usando un servicio llamado `DomSanitizer`.

- Ataques de ***CORS* (*Cross-Origin Resource Sharing*)**, configurando los orígenes permitidos (como el del *frontend*), estableciendo cabeceras como `Authorization` y `Content-Type` y métodos como `GET`, `POST` y demás que estén permitidos. De esta forma se evita que otros sitios *web* no autorizados puedan hacer peticiones ilegítimas al *backend*.

- Ataques ***SQL Injection***, basados en aprovecharse de la ausencia de consultas preparadas para incrustar código *SQL* mediante formularios, por ejemplo. El uso de consultas parametrizadas y métodos de repositorio personalizados, que realizan validaciones y no ejecutan consultas manuales predefinidas evita que los parámetros del usuario puedan alterar la lógica de las consultas.

Gracias a estas prevenciones, se evitan los típicos ataques que aprovechan vulnerabilidades para obtener información o realizar acciones ilícitamente.

También se lleva a cabo una **gestión de errores correcta**, que no exponga mensajes técnicos del *backend* al cliente en crudo. Los errores se transforman en mensajes de error personalizados y códigos de error representativos (`404`, `500`, etc). Se trata de no revelar datos comprometidos, como *stack traces* de capas internas o información derivada (como un mensaje que indique que la contraseña no es correcta pero el correo sí, revelando que dicho correo está registrado en la aplicación). Básicamente, los mensajes son lo suficientemente claros para que el usuario entienda el error pero sin revelar detalles internos del sistema ni estructuras de base de datos.

Además, se **integran los servicios externos de forma segura**. Para ello, en la configuración de las *APIs* usadas se limitan por dominio, restringiendo su uso al del *frontend* oficial. Las claves no se exponen en el código fuente principal, sino que se establecen en las configuraciones del entorno, protegidas y excluídas de los controles de versiones.

En el caso de las *APIs* de *Google*, se limita el tipo de *APIs* usadas, para evitar un uso erróneo y descontrolado de servicios, que pueda desembocar en costes adicionales.

Respecto a la *API* de *PayPal*, se trabaja en el *backend* de forma segura, validando en todo momento datos como el estado del pago, el receptor del dinero, la cantidad ligada

al pago, la moneda con la que se está trabajando y la coherencia y disponibilidad de las fechas. De esta forma, se evita que se persista una reserva sin realizar un pago y viceversa, que se falsifique un pago, que se emplee un `ID` de pago antiguo para guardar una reserva, entre otras situaciones. Esto permite que el sistema de reservas sea seguro, evitando que los intermediarios puedan manipular los pagos. Además, se tiene muy en cuenta el concepto de transaccionalidad, estudiando la posibilidad de que surjan reservas paralelas para rangos de fechas iguales o solapados. En este caso, se evita la persistencia y pago de varias reservas para los mismos días, como se explicará posteriormente en la sección 6.7, dedicada a la transaccionalidad.

Se asegura un **almacenamiento de datos sensibles seguro y privado**. Para ello, las contraseñas no se almacenan en texto plano, sino que se usa *BCrypt*, un algoritmo de *hash* lento que resiste a ataques por fuerza bruta. Este algoritmo incluye en el *hash* un *salt* único (de 128 *bits* de forma automática), de forma que si dos usuarios tienen la misma contraseña, su apariencia *hasheada* difiere, rompiendo así con la uniformidad de contraseñas iguales. Esto es muy beneficioso, puesto que evita ataques usando *rainbow tables*, una serie de tablas precomputadas con contraseñas comunes y sus *hashes*, de forma que si roba las contraseñas *hasheadas* de la base de datos, las compara con la tabla *rainbow* y coincide en algún caso, conocerá las contraseñas originales. Pero al existir el *salt*, el atacante tendría que generar una *rainbow table* nueva por cada posible *salt*, lo cual es computacionalmente inviable. En cuanto a los *JWT*, se firman con una clave secreta almacenada de forma segura, cuentan con cierto periodo de expiración y pierden validez tras cerrar sesión.

Destacar que en las primeras versiones del proyecto, las contraseñas se enviaban en plano desde el *frontend*, y ya en el *backend* se *hasheaban*. Esto hacía que las comunicaciones entre las capas tuvieran un tráfico sin cifrar, estando expuestos a ataques *Man-in-the-Middle (MitM)*. Para evitar esto, se ha aplicado seguridad *end-to-end*, obteniendo un certificado autofirmado y generando ciertas claves para que tanto *frontend* como *backend* trabajen con `https`. Para ello, generé con *openssl* una clave privada y un certificado autofirmado para *Angular*, sirviendo la aplicación con `ng serve` y el parámetro `-ssl`. Por su parte, en *Spring Boot* generé un archivo `keystore.p12`, habilité las configuraciones necesarias en `application.properties` y modifiqué el dominio permitido (puesto que *Angular* ya no usaba `http`, sino `https`). Del mismo modo, en los ficheros de entorno de *Angular*, modifiqué la ruta base del *backend*, que pasó de servirse en `http://localhost:8080` a hacerlo en `https://localhost:8443`. En definitiva, logré que el canal de comunicación estuviera cifrado, evitando el espionaje o alteración de los datos. Todo ello sumado a las *cookies* de *JWT* que pasaron a activar el *flag* `Secure`, permiten que se utilice un entorno más confiable para el usuario, a pesar de ser un prototipo aún no lanzado. Esto en un futuro supondrá una gran ventaja, tanto a nivel de seguridad, como respecto a rendimiento y prácticas *SEO* (*Search Engine Optimization*), puesto que el posicionamiento de *webs* seguras es mejor.

Además, se trata de guardar y manipular el menor número posible de datos sensibles, delegando en servicios externos como *PayPal* la gestión de números de tarjeta, direcciones y claves de acceso a las cuentas bancarias, como se mencionaba en la sección 6.4.

En cuanto a ciertas claves y rutas sensibles, se **emplean variables de entorno protegidas** que no se suben al repositorio encargado del control de versiones, y se almacenan en los archivos de entorno en el *frontend* y en ficheros de configuración en el *backend*. Entre estos datos sensibles, encontramos rutas base, la clave de *JWT*, *API keys* o claves de correo. En adición, los *logs* del *backend* se clasifican con niveles como `INFO` o `WARN` para evitar su exposición en producción y evitan incluir información sensible, simplemente comentarios necesarios para entender el flujo de ciertos métodos.

En definitiva, la seguridad constituye uno de los pilares de este proyecto, puesto que el servicio brindado debe ser confiable, robusto y seguro para los usuarios. Con todas las medidas implementadas, se reduce la posibilidad de sufrir ataques, inserciones indebidas y sustracciones ilícitas de información.

6.7. Transaccionalidad

Una transacción es un colección de operaciones que forman parte de una unidad lógica de trabajo. Dichas operaciones pueden acceder y afectar a diversos datos, pero se debe garantizar en todo momento que la totalidad de las operaciones se lleva a cabo. De no ser así, las operaciones deberían de revertirse, para evitar estados de inconsistencia en la base de datos.

Para que la base de datos permanezca íntegra, las transacciones deben cumplir las propiedades *ACID*, es decir, ser atómicas, consistentes, aisladas de otras transacciones y garantizando la durabilidad.

En el proyecto, se detectó la necesidad de gestionar la transaccionalidad a la hora de realizar reservas y pagos. Por ejemplo, si dos reservas para el mismo rango temporal (o rangos que se solapen) se realizaran al mismo tiempo, cada una desde el navegador se un usuario, se podrían dar las siguientes situaciones:

- Ambas reservas persistirían en la base de datos, cuando no pueden existir varias reservas para los mismos días. Además, se cobrarían ambas reservas, resultando en una situación complicada a nivel organizativo para el administrador.
- Solo persiste una de las reservas, pero ambas son cobradas a sus respectivos usuarios.
- Solo una de las reservas se llega a cobrar, pero ambas se guardan en la base de datos, resultando en un solapamiento de días y en un fallo a nivel de cobro.

Todas estas situaciones resultan desfavorables, tanto para la experiencia de usuario como para el administrador del sistema. Pues bien, todas ellas sucedían al realizar pruebas en la aplicación antes de haber aplicado una correcta gestión de la transaccionalidad.

Al tratarse de operaciones que manejan dinero, resulta muy importante establecer un nivel de aislamiento acorde a la relevancia y trascendencia de estas transacciones. Por tanto, se descartó la posibilidad de simplemente adoptar niveles de aislamiento como `READ UNCOMMITTED`, `READ COMMITTED` o `REPETEABLE READ`. Dichos grados de aislamiento

permiten una mayor concurrencia y rendimiento, pero aceptan riesgos que no estamos dispuestos a asumir en este caso. Por ello, me decanté por el uso de `SERIALIZABLE` como nivel de aislamiento para la transaccionalidad relativa a las reservas y los pagos. Al ser el nivel más estricto, protege contra todo tipo de situaciones desfavorables que puedan ocurrir, asegurando que, aunque varias transacciones se ejecuten simultáneamente, el resultado final será el mismo que si se hubieran ejecutado una a la vez, en secuencia.

En este caso, la transaccionalidad se aplicó mediante el uso de decoradores facilitados por *Spring Boot*, situados en la cabecera de la función encargada de crear la reserva, ubicada en el servicio correspondiente del *backend*. En este caso, se aplicó la cabecera `@Transactional(isolation = Isolation.SERIALIZABLE)` que significa que la transacción se ejecutará como si fuera la única operación en la base de datos, previniendo cualquier problema de concurrencia, como lecturas sucias, lecturas no repetibles o lecturas fantasma.

De esta manera, aplicando el nivel de aislamiento más alto debido a la importancia de no exponernos a errores relativos al saldo o persistencia de reservas, se logra evitar las situaciones desfavorables explicadas anteriormente. Con la intención de explicar los pasos que se siguen a la hora de realizar una reserva, garantizando tanto la transaccionalidad como la seguridad a la hora de realizar el pago, se exponen a continuación las fases de este proceso:

1. Un usuario accede a la aplicación *web* desde su navegador, entra en la sección de reservas y selecciona un rango válido para su estancia. Rellena el formulario y se le muestran el desglose y las opciones de pago.

2. Al seleccionar una de las opciones de pago, presionando por ejemplo el botón de pago con *PayPal*, se genera un `ID` de orden de pago, cuyo estado interno será `CREATED`. En este punto, el pago no se encuentra en un estado válido para el *backend*, por lo que resulta inútil sortear la pasarela de pago mediante aplicaciones como *Postman*.

3. Tras acceder a la cuenta de *PayPal* con las credenciales necesarias y pulsar en `Completar pago`, se realiza una llamada al *backend* para comenzar con el proceso de creación de una reserva, a la par que el estado del pago pasa a `APPROVED`. A partir de este punto, se representan los pasos seguidos en el diagrama de secuencia 6.10.

4. El método encargado de la creación de la reserva sigue una serie de fases de comprobación de datos y verificación de fechas. Comienza validando la orden de pago capturada mediante un método secundario:

    a) En primer lugar, comprueba que el estado del pago sea `APPROVED`, es decir, que el pago esté creado y confirmado por el usuario, pero que no esté ni en el estado previo (`CREATED`) ni en el estado posterior (`COMPLETED`). Esto evita que se realicen reservas con *IDs* de pagos creados, que se puedan revertir fácilmente, o ya completados.

   b) Se realizan comprobaciones adicionales, para evitar que se usen aplicaciones como *Postman* para realizar peticiones al *endpoint* con *IDs* de pagos en estado `APPROVED`, pero de cantidades, monedas o destinatarios que no se correspondan. Por ende, se comprueba que coincida el destinatario del pago (la casa rural), la moneda aceptada (euros) y la cantidad a depositar (el precio de la reserva).

5. Si la comprobación inicial mediante las correspondientes peticiones a la *API* de *PayPal* es satisfactoria, se procede a consultar internamente si las fechas seleccionadas son válidas desde el punto de vista de la lógica del negocio. Básicamente, se comprueba que la fecha de inicio no sea previa o igual al día actual, que la fecha de inicio sea previa a la de fin y que las fechas no se encuentren ocupadas por otra reserva.

6. Si todo sigue adelante, se trata de guardar la reserva en la base de datos. Si falla, se abortará el proceso, cancelando el pago para no realizar ningún cargo.

7. Si la reserva se logra persistir correctamente, se trata de capturar el pago, es decir, de hacer efectivo el desembolso de dinero por parte del comprador, resultando en su ingreso en la cuenta de la casa. Si todo sale bien, se modifica la reserva en base de datos, agregando un atributo que permitirá cancelar la reserva y devolver el importe en un futuro, acabando así el proceso de reserva. En cambio, si fallara la captura del pago, se abortaría el proceso.

Cualquier fallo en este proceso, desembocaría en una excepción, que tras ser capturada realizaría un `ROLLBACK`. Esto último es una operación que deshace los cambios realizados en una transacción, devolviendo la base de datos a su estado anterior. Por tanto, la base de datos jamás quedaría en un estado de inconsistencia, a la par que no se desembolsaría el dinero si no se ha logrado persistir la reserva.

Por tanto, la combinación de la transaccionalidad con las comprobaciones propias de un sistema seguro, hacen que el sistema de pagos y reservas no permita la aparición de ninguna de las situaciones desfavorables que listábamos al inicio de la sección.

Desde el punto de vista del navegador del cliente, en caso de realizarse dos o más reservas para el mismo periodo a la vez, ocurriría lo siguiente:

- Ambos usuarios presionarían el botón de *Completar pago* a la vez, por lo que dos transacciones con un `ID` de pago en estado válido (`APPROVED`) entrarían al sistema.

- Al tener un aislamiento de tipo `SERIALIZABLE`, a pesar de que ambas transacciones lleguen en paralelo, se gestionarán de forma secuencial, iniciando el proceso de reserva para una de ellas.

- La transacción que inicie el método será sometida a las comprobaciones mencionadas, y si llegara a terminar de forma satisfactoria, provocaría la persistencia de la reserva y el correspondiente pago. Se mostraría la pantalla de resumen de reserva al cliente que inició dicha transacción.

- Por su parte, la transacción que estaba en espera, al desbloquearse las tablas debido a la finalización de la primera transacción, iniciará el método de creación de reserva.
- La comprobación inicial relativa al pago (estado, entidad destinataria, etc.) será satisfactoria, pero al verificarse el rango de fechas, se abortará el proceso al comprobar que ya existe una reserva para ese rango (al haberse persistido la reserva de la primera transacción). Por tanto, se realizará un `ROLLBACK`, finalizando la segunda transacción, evitando así llegar al punto de persistir la reserva o confirmar el pago.
- En el navegador del segundo usuario, se mostrará un mensaje que indicará que alguien ha realizado una reserva instantes antes que él.

Además, en caso de caída del servidor, el orden de los métodos evitan que se realice un pago si no se llega a persistir la reserva que lo origina, mostrando en el cliente una pantalla de error.

Para ilustrar y lograr comprender mejor este proceso, se adjunta un diagrama de secuencia que expone ciertos pasos realizados, obviando detalles menores para no engordar la imagen innecesariamente. Se ofrece una versión ampliada del diagrama en la figura B.5.

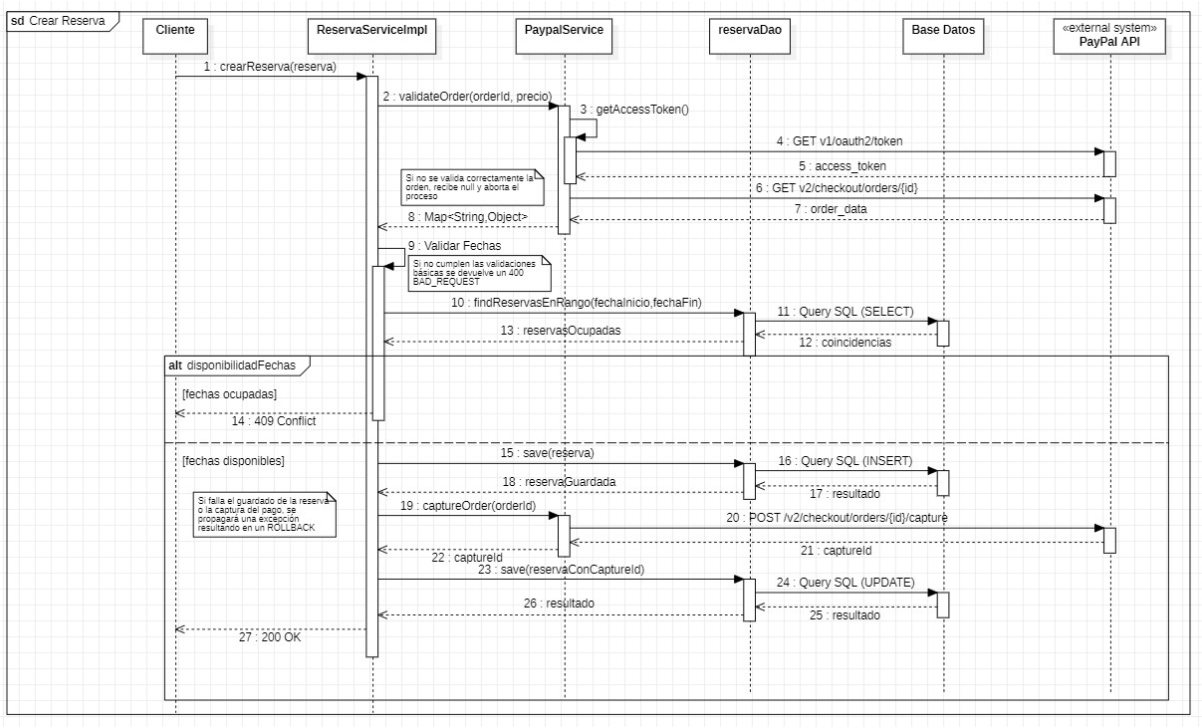

Figura 6.10: Diagrama de Secuencia de creación de la reserva

Se obvian las acciones iniciales ligadas a completar y validar el formulario, comenzando a analizar la secuencia en el momento de envío de los datos de la reserva para iniciar el proceso correspondiente en el *backend*. Se evita también incluir diversos caminos

alternativos, explicados anteriormente al detalle, como los posibles errores que abortan el proceso, las numerosas validaciones de la orden (cantidad, destinatario, moneda, etc.) y la posibilidad de controlar excepciones que desemboquen en un `ROLLBACK` que mantenga la integridad.

De hecho, en alguna ocasión, pero muy rara vez, pueden llegar a ocurrir `DEADLOCKS`. Un `DEADLOCK` o interbloqueo consiste en una situación donde dos o más transacciones se encuentran bloqueadas mutuamente, esperando que la otra libere un recurso que necesita, impidiendo así que cualquiera de las transacciones pueda avanzar. Al utilizar `SERIALIZABLE` como modo de aislamiento, no estamos exentos de la ocurrencia de estos bloqueos. Sin embargo, en las situaciones en las que han ocurrido, se ha mantenido la consistencia de los datos y pagos, evitando cualquier tipo de situación desfavorable. Esto se debe, en parte, a la correcta configuración realizada.

Por último, mencionar que como se comentaba previamente, al finalizar el proceso de reserva se almacena un `paypal_capture_id`, que permite la cancelación de la reserva, como se explicaba en la sección 6.4.

Todo esto permite mantener un entorno seguro, tanto para usuarios como para administradores, evitando que se incumplan las políticas de la casa rural y garantizando que jamás se perderá dinero de forma injustificada en ninguna de las partes participantes.

6.8. Usabilidad y Accesibilidad

La usabilidad y la accesibilidad han sido pilares fundamentales en el diseño e implementación de la aplicación *web*. Ambos aspectos se han tenido en cuenta para mejorar la experiencia del usuario, reducir la curva de aprendizaje y garantizar el acceso a la plataforma a un público lo más amplio posible.

Desde el punto de vista de la usabilidad, se ha priorizado una interfaz limpia, coherente y fácil de navegar. Las secciones están claramente organizadas mediante un menú principal (*navbar*) accesible desde todas las páginas. Se han utilizado iconos y colores intuitivos, y se han evitado elementos innecesarios que pudieran distraer o generar confusión.

La experiencia del usuario se ha optimizado con componentes como botones de acción bien diferenciados, validación de formularios en tiempo real, mensajes de error claros y retroalimentación visual inmediata tras cada acción importante (por ejemplo, mediante *toasts* al realizar una reserva o enviar un mensaje de contacto). Básicamente, la aplicación arrojará un *feedback* constante al usuario, con la intención de que comprenda lo que está ocurriendo en cada instante.

Además, se ha tratado de guiar y advertir en todo momento a los usuarios acerca de las acciones que van a realizar y sus posibles consecuencias. Por ejemplo, a la hora de realizar acciones comprometidas como el borrado de un usuario o la cancelación de una reserva, se usan modales (*modals*) que explican al detalle las consecuencias de ejecutar dicha acción, permitiendo al usuario que se retracte de su decisión.

La utilización de componentes como desplegables (*accordions*) y paginación en las tablas permite que los clientes puedan disponer de un espacio ordenado, mostrando el

contenido que deseen. De esta manera, se pueden incluir explicaciones e instrucciones que los usuarios más experimentados podrán ocultar, mientras que los usuarios nuevos podrán informarse y aprender a usar correctamente la aplicación.

La inclusión de filtros de todo tipo, permite al cliente (o administrador) disponer cuanto antes de la información necesaria, sin pérdidas de tiempo. El sistema de filtrado se complementa con acciones como la exportación e impresión de la información mostrada en tablas. También se incluyen botones cuya misión es revertir el filtrado o modificación de información en los paneles de usuario y administrador, evitando que el usuario final pierda la paciencia tratando de revertir acciones de forma totalmente manual.

La aplicación *web* es totalmente adaptable o *responsive*, es decir, se ajusta a las distintas dimensiones de pantalla que puede tener un dispositivo. De esta manera, a pesar de no contar con una aplicación móvil nativa, se logra una gran experiencia de usuario al interactuar con la aplicación. No sólo se han realizado ajustes y redimensionamientos de contenido como imágenes y textos, sino que los componentes que conforman la aplicación *web* están preparados para variar en base a la resolución a la que se enfrentan. Por ejemplo, el `carousel` del menú principal o el mapa del menú de contacto reducen sus tamaños manteniendo las funcionalidades que desempeñan, los filtros se aglomeran de manera limpia y simétrica, el calendario pasa de mostrar dos páginas a renderizar un sólo mes para resultar más estrecho y las imágenes y componentes se apilan para formar un *scroll* amigable. Destaca la adaptación de las tablas, que para evitar una disposición horizontal en pantallas estrechas, se convierten en tarjetas unitarias por cada fila de la tabla, resultando visiblemente más presentables.

En lo relativo a la accesibilidad, se han seguido buenas prácticas recomendadas por la *WAI* (*Web Accessibility Initiative*) [53], como el uso correcto de etiquetas semánticas en *HTML*, la posibilidad de navegación mediante teclado, y contrastes de colores adecuados.

Además, se han añadido atributos `alt` a todas las imágenes y se han empleado componentes accesibles (por ejemplo, botones y formularios que respetan el *focus* del teclado).

Para comprobar que se han seguido la mayoría de estas buenas prácticas, he empleado herramientas como *Lighthouse* [15], que permite analizar el rendimiento y la calidad de la página, así como la accesibilidad que esta permite.

Realizando un rápido estudio con esta herramienta automatizada, se obtienen unos valores relativamente altos en las secciones de accesibilidad y *SEO*, al realizar buenas prácticas como las siguientes en el ámbito de la accesibilidad:

- Botones con nombre accesibles.
- Atributos `alt` en todas las imágenes mostradas.
- Buen uso de los atributos `ARIA`.
- *Tags* `HTML` accesibles, ordenados y coherentes.

Figura 6.11: Ejemplo de análisis de accesibilidad con *Lighthouse*

También se aplican decisiones bien valoradas desde el punto de vista de un análisis *SEO*, como las siguientes:

- Correcto manejo de códigos `HTTP`.
- Atributos `alt` en las imágenes.
- Enlaces rastreables y coherentes.

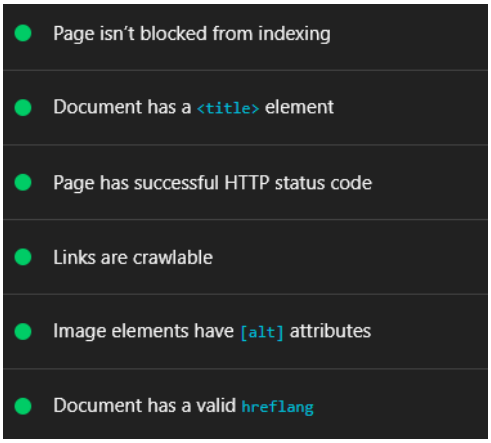

Figura 6.12: Ejemplo de análisis *SEO* con *Lighthouse*

En otros análisis secundarios, como el de rendimiento o el de buenas prácticas, la aplicación *web* no despunta, pero obtiene resultados decentes. A nivel de rendimiento, perjudica el hecho de cargar de forma estática las imágenes, que se encuentran almacenadas en un directorio del proyecto. Esto no significa que los tiempos de carga sean lentos, al contrario, ya que en cuanto a tiempos de respuesta y tiempos de ejecución de código *JavaScript* se superan las pruebas realizadas.

Además, la herramienta destaca que se llevan a cabo buenas prácticas, como el uso de `HTTPS` a pesar de tratarse de un entorno local, el establecimiento de resoluciones correctas y *aspect ratios* balanceados, así como la ausencia de errores de navegador por consola.

También se han realizado pruebas manuales de navegación mediante teclado, simulaciones con diferentes resoluciones de pantalla, y se ha verificado que los componentes interactivos son comprensibles incluso sin el uso del ratón. Para realizar estas comprobaciones, he utilizado tanto herramientas especiales como el Narrador de *Windows* [17], como pruebas manuales mediante teclas usadas para la navegación sin ratón.

El Narrador de *Windows* permite usar el equipo sin un ratón para completar tareas comunes si eres invidente o tienes deficiencias visuales. En el caso de la aplicación *web*, lee e interactúa con elementos en la pantalla, como texto y botones sin problema alguno, demostrando la accesibilidad de la aplicación.

Por otro lado, mediante el uso de comandos como `Tab` (avanzar al siguiente elemento seleccionable), `Shift+Tab` (retroceder al elemento seleccionable anterior) y `Enter` (confirmar), se puede comprobar que los elementos de la aplicación *web* son perfectamente reconocidos. Podemos entonces desplazarnos e interactuar con los componentes usando el teclado.

Por último, gracias a elementos *HTML* como `section`, usados en ciertas páginas de la *web*, algunos navegadores permiten habilitar la vista de lectura, logrando recopilar el texto e imágenes que conforman la página, y ofreciendo herramientas como la lectura en voz alta.

En resumen, la aplicación busca ser inclusiva, comprensible y fácil de usar, reduciendo las barreras tecnológicas para los usuarios, independientemente de sus habilidades técnicas o condiciones físicas. La correcta organización de elementos, la adopción de técnicas que facilitan la accesibilidad y el seguimiento de estándares recomendados, permiten a diversos usuarios interactuar sin problemas con la aplicación.

Capítulo 7

Pruebas y aceptación

Este capítulo presenta las distintas pruebas realizadas para verificar el correcto funcionamiento del sistema, tanto a nivel funcional como técnico. Se han empleado diferentes tipos de pruebas: pruebas funcionales (caja negra), pruebas estructurales (caja blanca), pruebas unitarias automatizadas y una fase final de aceptación. El objetivo es garantizar la calidad del *software* y la conformidad con los requisitos definidos.

Cabe destacar que no se han centrado grandes cantidades de recursos en realizar las pruebas unitarias para los distintos *frameworks* empleados, pero se ha investigado y comprendido cómo podrían realizarse en caso de necesitarlo.

7.1. Pruebas de caja negra

Las pruebas de caja negra son una técnica de pruebas de *software* donde se evalúa la funcionalidad de un sistema sin conocer su estructura interna, código o diseño. Se centra en las entradas y salidas del sistema, tratando al *software* como una caja negra donde solo se observan sus respuestas a diferentes entradas, simulando el comportamiento de un usuario final.

Estas pruebas verifican si las entradas producen las salidas esperadas, evaluando principalmente aspectos funcionales, flujos de uso y validación de formularios.

Se han planteado varios escenarios típicos de interacción con el sistema (registro, *login*, reserva, etc.), comprobando que el sistema reacciona correctamente ante entradas válidas e inválidas.

Tras visualizar la tabla 7.1, debo matizar que el formulario será inválido y por ende no podrá enviarse cuando: dejas sin rellenar los campos requeridos, si no se introducen dos apellidos, si el correo no tiene el formato adecuado, si el correo ya está registrado en la aplicación, si la contraseña tiene menos de 6 caracteres, si las contraseñas no coinciden.

En base a los expuesto en la tabla 7.2, hay que tener presente que el formulario será inválido si no se rellenan los campos requeridos, si el correo no tiene formato adecuado, si el correo no existe en la base de datos, si el correo y la contraseña no son válidos.

En el caso de la tabla 7.3, hay que agregar que al seleccionar otro tipo de espacios, el filtro funciona correctamente. El botón para limpiar los filtros, lo restablece al valor por

PCN01	**Registrarse como usuario**
Objetivo	Verificar que el sistema permite el registro de un nuevo usuario.
Precondiciones	No debe existir un usuario registrado con el mismo correo.
Datos de entrada	Nombre: Miguel, Apellidos: Esteban Navajo, Correo: miguel@gmail.com, Contraseña: miguel123, Confirmar contraseña: miguel123
Acción esperada	Al rellenar el formulario, debe mostrarse un mensaje de éxito de registro y redirigir al *login*.
Resultado	Apto

Cuadro 7.1: Prueba de caja negra – Registro de usuario

PCN02	**Iniciar sesión en la aplicación**
Objetivo	Verificar que el sistema permite el inicio de sesión en la aplicación.
Precondiciones	Debe existir una cuenta creada con dicho correo.
Datos de entrada	Correo: miguel@gmail.com, Contraseña: miguel123
Acción esperada	Al rellenar el formulario, debe mostrarse un mensaje de éxito de *login* y redirigir al *home*.
Resultado	Apto

Cuadro 7.2: Prueba de caja negra – Inicio de sesión

PCN03	**Filtrar habitaciones por Tipo**
Objetivo	Verificar que el sistema permite el filtrado de habitaciones por tipo.
Precondiciones	Deben existir espacios registrados en el sistema.
Datos de entrada	Tipo: Salon
Acción esperada	Al seleccionar el tipo de espacio, deben de mostrarse únicamente los salones.
Resultado	Apto

Cuadro 7.3: Prueba de caja negra – Filtrado de espacios

defecto.

Continuando con lo dispuesto en la tabla 7.4, si se selecciona un rango que contiene

PCN04	**Seleccionar fechas para la reserva**
Objetivo	Verificar que el sistema permite seleccionar un rango temporal para realizar una reserva.
Precondiciones	Deben existir fechas disponibles.
Datos de entrada	Fecha Inicio: 06/09/2025, Fecha Fin: 08/09/2025
Acción esperada	Al seleccionar las fechas mediante el calendario, debe mostrarse un mensaje de éxito y habilitarse el formulario de compra.
Resultado	Apto

Cuadro 7.4: Prueba de caja negra – Selección de fechas para la reserva

PCN05	**Rellenar formulario de reserva**
Objetivo	Verificar que el sistema permite cumplimentar el formulario de reserva.
Precondiciones	El rango temporal seleccionado debe de haber sido válido.
Datos de entrada	Nombre: Miguel, Correo: miguel@gmail.com, Mensaje: Reserva para Miguel
Acción esperada	Al rellenar el formulario, debe mostrarse un desglose del precio en base a las tarifas expuestas en la página y habilitarse los botones de pago.
Resultado	Apto

Cuadro 7.5: Prueba de caja negra – Relleno del formulario de reserva

fechas reservadas, se mostrará un mensaje de error. No se mantienen habilitadas las fechas pasadas o del día actual.

Respecto a la tabla 7.5, el nombre y el correo son requeridos, de lo contrario el formulario no será válido. Del mismo modo, el correo debe tener un formato válido. El mensaje es opcional.

En cuanto a la tabla 7.6, hay que matizar algunas cosas. Se debe escoger el método de pago preferido. Al pagar con *PayPal*, se muestra una ventana emergente para iniciar sesión y confirmar el pago. Si seleccionas el pago con tarjeta, debes completar un extenso formulario totalmente validado. Normalmente la reserva se efectuará correctamente, mostrando una pantalla de éxito con un resumen. En caso de error, ya sea por fallo interno del servidor o por temas de transaccionalidad 6.7, se mostrará un mensaje explicativo y no se efectuará ningún cargo ni se registrará la reserva.

Continuando con la tabla 7.7, es relevante remarcar que en esta misma página, se cargan correctamente ciertas reseñas de la casa rural en *Google*.

PCN06	**Realizar pago de la reserva**
Objetivo	Verificar que el sistema permite el pago para completar la reserva.
Precondiciones	El formulario previo al pago debe ser válido.
Datos de entrada	Datos de pago dependiendo del método de pago escogido.
Acción esperada	Al rellenar los datos de pago, debe mostrarse una pantalla de resumen de la reserva, registrarse en base de datos y efectuar el cargo.
Resultado	Apto

Cuadro 7.6: Prueba de caja negra – Realizar pago de la reserva

PCN07	**Consultar más reseñas**
Objetivo	Verificar que el sistema permite acceder al total de reseñas mostrado en *Google*.
Precondiciones	Debe contar con conexión estable a la red.
Datos de entrada	Pulsado del botón.
Acción esperada	Al pulsar el botón, el sistema redirigirá al usuario a una nueva pestaña que muestre el total de las reseñas de la casa.
Resultado	Apto

Cuadro 7.7: Prueba de caja negra – Mostrar más reseñas

PCN08	**Redirigir al usuario para valorar la casa**
Objetivo	Verificar que el sistema permite la redirección desde la pestaña de reseñas a un formulario de *Google* para valorar su estancia.
Precondiciones	El usuario debe tener su sesión de *Google* iniciada en el navegador.
Datos de entrada	Pulsado del botón correspondiente.
Acción esperada	Al pulsar el botón, el sistema abre una nueva pestaña mostrando un formulario de *Google* para dejar un reseña.
Resultado	Apto

Cuadro 7.8: Prueba de caja negra – Redirigir al usuario para dejar una reseña

PCN09	**Interactuar con el mapa**
Objetivo	Verificar que el sistema permite interactuar con el mapa.
Precondiciones	Disponer de conexión a la red.
Datos de entrada	Acciones listadas para interacción con el mapa.
Acción esperada	El mapa permite la interacción: cambios de modo, vista de calle, etc.
Resultado	Apto

Cuadro 7.9: Prueba de caja negra – Interacción con el mapa

PCN10	**Contactar con la propietaria**
Objetivo	Verificar que el sistema permite el envío de correos a la bandeja de la casa mediante el formulario de contacto.
Precondiciones	Debe superar el desafío de *reCAPTCHA*.
Datos de entrada	Nombre: Miguel, Asunto: Duda casa, Correo: miguel@gmail.com, Mensaje: Tengo una duda
Acción esperada	Al rellenar los datos del formulario y habiendo superado el desafío para comprobar que no eres un robot, se enviará un correo a la dueña.
Resultado	Apto

Cuadro 7.10: Prueba de caja negra – Contactar mediante el formulario por correo

PCN11	**Filtrar actividades por categoría**
Objetivo	Verificar que el sistema permite el filtrado de actividades por categoría.
Precondiciones	Deben existir actividades mostradas por pantalla.
Datos de entrada	Categoría: Naturaleza
Acción esperada	Al seleccionar la categoría mediante el filtro, se muestran las actividades de dicho tipo.
Resultado	Apto

Cuadro 7.11: Prueba de caja negra – Filtrado de actividades

En cuanto a la prueba de caja negra expuesta en la tabla 7.10, el formulario debe ser válido, por lo que todos los campos deben tener contenido, el correo debe tener un formato adecuado y debe haberse superado el desafío.

Hay que matizar algunos aspectos relativos a la tabla 7.12. Se cargan por defecto el

PCN12	**Modificar datos de la cuenta**
Objetivo	Verificar que el sistema permite la modificación de diversos datos de la cuenta de usuario.
Precondiciones	Debe iniciar sesión para poder acceder al panel de usuario.
Datos de entrada	Nombre: Miguel, Apellidos: Esteban Navajo, Email: miguel1@gmail.com, Contraseña y Confirmar contraseña: vacío
Acción esperada	Al haber modificado únicamente el *email*, ya que el nombre y apellidos se mantienen por defecto y la contraseña se deja sin rellenar, el sistema debe modificar únicamente el dato modificado, cerrar sesión y redirigir al *login*.
Resultado	Apto

Cuadro 7.12: Prueba de caja negra – Modificación de datos de la cuenta

PCN13	**Eliminar la cuenta**
Objetivo	Verificar que el sistema permite eliminación de la cuenta de usuario.
Precondiciones	Debe iniciar sesión para poder acceder al panel de usuario.
Datos de entrada	Presionar el botón de eliminación y confirmar la acción.
Acción esperada	Se eliminará el perfil del usuario, cerrando sesión y redirigiendo al *login*.
Resultado	Apto

Cuadro 7.13: Prueba de caja negra – Eliminación de la cuenta

nombre, apellidos y correo actuales. En caso de modificar alguno de ellos, se permite pulsar el botón de modificar datos. La contraseña no se carga por defecto por seguridad, por lo que si se mantiene intacto el campo, no se tendrá en cuenta para la modificación. Los campos mantienen las validaciones ya mencionadas en otros apartados, como la comprobación de formato del correo. Además, se solicita la confirmación del usuario previa a la modificación.

Es importante remarcar que respecto a la tabla 7.13, se mostrará un cuadro de confirmación advirtiendo al usuario.

Continuando con la tabla 7.14, decir que se pueden realizar muchísimas combinaciones, que han sido probadas y han pasado satisfactoriamente las pruebas. Si se usa el botón de limpiado de filtros, se muestra el total de las reservas del usuario (o de la aplicación si se consulta desde el panel de administrador).

PCN14	**Filtrado de reservas**
Objetivo	Verificar que el sistema permite la correcta aplicación de filtros sobre las reservas del usuario (o las totales mediante una cuenta de administrador).
Precondiciones	Debe iniciar sesión para poder acceder al panel de usuario (o de administrador).
Datos de entrada	Buscador: Miguel, Calendarios: del 22/06/2025 al 25/06/2025, Estado: ACTIVA, Precio: Ascendente
Acción esperada	Al haber aplicado varios filtros, deben mostrarse las reservas que cumplan todos ellos.
Resultado	Apto

Cuadro 7.14: Prueba de caja negra – Filtrado de datos de reservas

PCN15	**Cancelar una reserva**
Objetivo	Verificar que el sistema permite la cancelación de una reserva de usuario (tanto una propia desde el panel de usuario como una de un cliente desde el panel de administrador).
Precondiciones	Debe iniciar sesión para poder acceder al panel de usuario (o de administrador). La reserva debe ser cancelable, es decir, no ser ni pasada ni comenzada.
Datos de entrada	Pulsado del botón de cancelación y confirmación posterior.
Acción esperada	Al presionar el botón y confirmar la cancelación, se debe modificar el estado de la reserva, pasando a liberar esas fechas. Además se devolverá el importe (parcial o total en base a políticas) de la reserva en la cuenta de *PayPal* del usuario.
Resultado	Apto

Cuadro 7.15: Prueba de caja negra – Cancelación de una reserva

Cabe destacar que, en base a lo expuesto en la tabla 7.15, si la reserva tiene una fecha de inicio que comienza en 7 días o menos, se devolverá el importe a falta de 100 euros. En cambio, si está prevista a una semana vista, se abonará la completitud del dinero. La cancelación de la reserva requiere una confirmación explícita.

Siguiendo con lo explicado en la tabla 7.16, de nuevo, hay muchas combinaciones posibles, todas ellas probadas y superadas con éxito. Si accionas el botón de limpiado de filtros, restablece la búsqueda.

PCN16	**Modificar datos de la cuenta**
Objetivo	Verificar que el sistema permite el filtrado de usuarios desde el panel de administrador.
Precondiciones	Debe iniciar sesión como administrador para poder acceder al panel. Deben existir usuarios registrados.
Datos de entrada	Buscador: Miguel, Rol: Usuario
Acción esperada	Se mostrará el usuario cuyo nombre/apellidos/correo contenga el término Miguel.
Resultado	Apto

Cuadro 7.16: Prueba de caja negra – Modificación de datos de la cuenta

PCN17	**Exportar información**
Objetivo	Verificar que el sistema permite la exportación de información en distintos formatos de las tablas de usuarios o reservas.
Precondiciones	Debe iniciar sesión para poder acceder al panel de usuario (o de administrador).
Datos de entrada	Filtrado previo (opcional) y pulsado del botón correspondiente.
Acción esperada	Al presionar el botón de exportación, se descargará un fichero con la información filtrada.
Resultado	Apto

Cuadro 7.17: Prueba de caja negra – Exportación de información

PCN18	**Navegar por la aplicación**
Objetivo	Verificar que el sistema permite la correcta navegación entre las pestañas ofrecidas.
Precondiciones	-
Datos de entrada	-
Acción esperada	Al pulsar los botones el sistema redirige a la página deseada.
Resultado	Apto

Cuadro 7.18: Prueba de caja negra – Navegación mediante *navbar*

Respecto a la tabla 7.17, se logra exportar información tanto en formato *PDF* como

PCN19	**Interacción con componentes**
Objetivo	Verificar que el sistema permite la interacción con otros botones, elementos y componentes de la página.
Precondiciones	-
Datos de entrada	Interacciones varias.
Acción esperada	Al presionar ciertos botones, se redirige o realiza la acción esperada, al interactuar con componentes, realizan las acciones que prometen.
Resultado	Apto

Cuadro 7.19: Prueba de caja negra – Interacciones con elementos varios

CSV. También se permite en algunos casos la impresión de ciertos resúmenes (reserva exitosa).

7.2. Pruebas de caja blanca

Por su parte, las pruebas de caja blanca son un método de prueba de software que implica el conocimiento de la estructura interna y el código de la aplicación para evaluar su funcionalidad y detectar errores. A diferencia de las pruebas de caja negra, donde se evalúa el comportamiento externo, las pruebas de caja blanca se centran en el código fuente.

Las pruebas de caja blanca se han centrado en validar la lógica interna del sistema, asegurando que todas las ramas, condiciones y estructuras de control se ejecutan correctamente al menos una vez. A diferencia de las pruebas de caja negra, que se enfocan en las entradas y salidas del sistema, estas pruebas inspeccionan el comportamiento interno del código.

A continuación, se enumeran algunas de las pruebas más relevantes realizadas durante el desarrollo:

- **PCB01 – Validación del formulario de *login* y registro:** se comprueba que las condiciones de validez del correo electrónico y la contraseña se activan correctamente antes de enviar la petición.
- **PCB02 – Comprobación del control de errores en el *backend*:** se testea que los errores de autenticación generen códigos de respuesta adecuados (`401` para credenciales incorrectas, `403` para accesos no permitidos, `404` para contenidos no encontrados, etc.).
- **PCB03 – Lógica de cancelación de reservas:** se cubren los posibles escenarios en los que se puede o no cancelar una reserva (por fechas, estado de la reserva, usuario autorizado, etc.) tanto desde el *frontend* como desde el *backend*.

- **PCB04 – Gestión de *cookies* y cierre de sesión:** se analiza que la *cookie JWT* sea eliminada correctamente al cerrar sesión y que se redirija al usuario fuera del área privada. De igual manera, se comprueba que al iniciar sesión y/o al realizar acciones que requieran cierto rol, se deposite o compruebe la validez del *token*.
- **PCB05 – Acceso a rutas protegidas:** se comprueba que los interceptores (*guards*) en *Angular* bloquean correctamente las rutas privadas si no hay sesión activa, o si no se tienen los permisos necesarios. De igual manera, si hay una sesión activa no se permite el acceso a la zona de autenticación.
- **PCB06 – Validación de filtros en el historial de reservas o en el registro de usuarios:** se recorre todo el flujo lógico de la aplicación de filtros y se verifica que no haya errores en combinaciones vacías o inválidas.
- **PCB07 – Carga condicional de datos en el panel del usuario:** se evalúa que solo se muestren los datos correspondientes al usuario autenticado y que las peticiones al *backend* incluyan el *token* correctamente.
- **PCB08 – Lógica de borrado de cuentas de usuario:** se evalúa que solo se permita borrar la cuenta del propio usuario desde su panel, o cualquier cuenta desde el de administrador, gestionando los diversos flujos y comprobando la validez de los *token*.
- **PCB09 – Comprobación del flujo de reservas:** se analizan los escenarios que pueden ocurrir al realizar reservas (varias reservas concurrentes, caída del servidor, etc.) y se garantiza que nunca se carguen importes indebidos o se registren reservas no deseadas.
- **PCB10 – Adaptabilidad de la interfaz:** se evalúa la adaptabilidad de cada componente y página en diversas resoluciones y navegadores para comprobar que es accesible para cualquier persona. Se realizan pruebas de manejo por teclado, *tests* de accesibilidad de componentes y pruebas con aplicaciones de asistencia.
- **PCB11 – Comprobación de conexiones con servicios externos y con la base de datos:** se evalúa que las conexiones con servicios externos, *APIs* y las propias capas de la aplicación sean consistentes. También se prueba que en caso de error se muestren los debidos mensajes informativos.

Cabe destacar que para las comprobaciones del buen funcionamiento de las *APIs*, tanto la *API REST* del *backend* con todos sus *endpoints*, como las *APIs* de servicios externos, se ha empleado *Postman* [8]. Esta aplicación facilita el estudio intensivo de las peticiones y respuestas a las *APIs*.

Estas pruebas han sido ejecutadas tanto manualmente como mediante pruebas unitarias y de integración, permitiendo detectar errores lógicos internos que no podrían haberse identificado mediante pruebas de caja negra únicamente.

7.3. Pruebas unitarias en *Angular* y *Spring Boot*

Además de las diversas pruebas realizadas de forma manual, probando la aplicación desde el punto de vista de un usuario y analizando flujos principales y alternativos, se pueden realizar una serie de pruebas automatizadas en ambas capas.

Estas pruebas específicas reciben el nombre de pruebas unitarias, un tipo de prueba de *software* que se centra en verificar el correcto funcionamiento de las unidades más pequeñas de código, como funciones, métodos o clases, de forma aislada. El objetivo principal de las pruebas unitarias es garantizar que cada componente individual funcione según lo esperado, antes de integrarlo con el resto del sistema.

Se ha seguido un proceso de investigación y formación, aprendiendo a utilizar estas pruebas. Sin embargo, debido a las limitaciones temporales del proyecto, se ha decidido no realizar pruebas unitarias para las funciones que componen la aplicación, ya que el desembolso de tiempo y recursos sería demasiado elevado. Este tipo de pruebas están orientadas a proyecto de mayor envergadura, llevados a cabo por equipos con varios integrantes. Normalmente algunos de estos integrantes se dedican a la realización de pruebas unitarias para verificar que cada componente se construye correctamente, que las funciones devuelven valores esperados, que los servicios se cargan de forma satisfactoria, etc.

De todas formas, considero interesante la explicación de las pruebas unitarias que se pueden realizar tanto en *Angular* como en *Spring Boot*, ya que a pesar de no haber considerado su inclusión en el proyecto, he investigado al respecto, a nivel de uso y de generación de las mismas.

Comenzamos hablando de las pruebas unitarias realizadas en *Angular*, que se ejecutan mediante el entorno *Karma* [14] y utilizan el *framework Jasmine* [13] para definir expectativas y aserciones.

En Angular, cada componente o servicio puede incluir un archivo `.spec.ts` que contiene las pruebas unitarias asociadas. De hecho, por configuración del propio proyecto, al generar un componente o servicio mediante el comando `ng g component` o `ng g service`, se generan automáticamente los archivos `.ts, .html, .css` y `.spec` que lo componen.

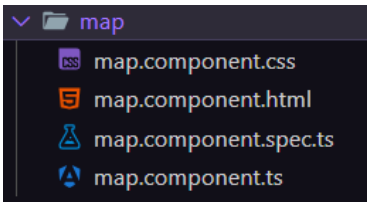

Figura 7.1: Ejemplo de existencia de archivos `.spec` en el proyecto *Angular*

Estos archivos contienen código escrito en *Jasmine* que ejecuta pruebas automatizadas para asegurar que el código funciona correctamente. Por ejemplo, se comprueba que el componente se instancia correctamente, o que los métodos presentes arrojan los valores esperados. Para ejecutar estas pruebas, se emplea el comando `ng test`, que abre una

pestaña nueva del navegador en las que el código de pruebas escrito en *Jasmine*, se ejecuta y muestra gracias a *Karma* un resultado de las pruebas realizadas.

Por ello, a pesar de no haberse empleado este tipo de pruebas de forma exhaustiva en el proyecto, resulta interesante saber por qué cada componente presenta los archivos `.spec`, así como conocer cómo ejecutar esta batería de pruebas.

Por otro lado, existen ciertas pruebas unitarias que podemos ejecutar en el proyecto *backend*, creado con *Spring Boot*. Estas pruebas se pueden implementar con *JUnit* [50] y *Mockito* [16]. *JUnit* permite definir y ejecutar tests individuales, mientras que *Mockito* permite simular dependencias como servicios, repositorios o controladores.

Básicamente, *JUnit* es un *framework* de pruebas unitarias de código abierto para *Java*. En este caso, se puede utilizar en proyectos *Spring Boot*. Se utiliza para escribir y ejecutar pruebas automatizadas de forma repetible, lo que ayuda a garantizar la calidad y fiabilidad del código implementado. Se podrían generar pruebas para las diversas clases de mi proyecto, comprobando la correcta instanciación de las clases, el resultado de los métodos, etc.

Por su parte, *Mockito* es es un *framework* de código abierto en *Java* que se utiliza para crear objetos simulados o *mocks*, que permiten aislar y probar el comportamiento de las clases en pruebas unitarias. Básicamente, en lugar de perder el tiempo en las pruebas unitarias en establecer conexiones con servicios externos, podemos emplear esta herramienta para sustituir y falsear estas conexiones, centrándonos en las pruebas unitarias.

De nuevo, no se han realizado estas pruebas para el proyecto *backend*, pues duplican el trabajo necesario para sacar adelante el proyecto. De todas formas, se ha considerado estudiar y entender estas herramientas, pues resulta una forma interesante de probar las clases y métodos que componen el *backend*.

En definitiva, a pesar de no haber empleado pruebas unitarias para corregir y supervisar ciertos aspectos de los proyectos *Angular* y *Spring Boot*, he investigado al respecto, pues resultan herramientas interesantes. Podrían aplicarse en un futuro próximo, cuando la aplicación *web* se lance, con la intención de depurar más aún el código y pulir ciertos aspectos de los componentes y clases que integran ambos proyectos.

7.4. Aceptación de las pruebas

Tras la implementación y ejecución de un conjunto extenso de pruebas de validación funcional y estructural del sistema, puedo concluir que la aplicación cumple con los requisitos definidos inicialmente.

A través de las pruebas de caja negra se ha comprobado que el sistema responde de manera adecuada ante entradas válidas e inválidas, cubriendo escenarios positivos y negativos. Estas pruebas, aplicadas sobre la interfaz y las funcionalidades clave del sistema, han permitido garantizar el correcto funcionamiento desde la perspectiva del usuario final.

En paralelo, las pruebas de caja blanca han ofrecido una visión más profunda del comportamiento interno del sistema. A través de ellas, he logrado evaluar condiciones, estructuras de control y rutas de ejecución internas tanto en el *frontend* como en el

backend. Se han tratado aspectos como la validez de los *tokens*, la gestión de sesiones, los permisos por rol, las restricciones de acceso, la lógica de cancelación o borrado de usuarios, la interacción con servicios externos y la accesibilidad.

Aunque no se han implementado pruebas unitarias automatizadas, se ha descrito detalladamente su funcionamiento tanto en *Angular*, mediante los archivos `.spec.ts` ejecutados con *Jasmine* y *Karma*, como en *Spring Boot*, a través de pruebas con *JUnit* y *Mockito*. Esto demuestra el conocimiento de su relevancia en el proceso de desarrollo ágil, y sugiere un posible plan de mejora para incorporar pruebas automatizadas en fases futuras del proyecto.

En conjunto, la cobertura de pruebas ha sido suficiente para considerar que la aplicación está en condiciones de ser utilizada de forma estable. De hecho, se abarcan los principales casos de uso del sistema, incluyendo operaciones críticas como inicio de sesión, autorización por roles, gestión de usuarios y tratamiento de errores. Se ha validado tanto la robustez como la seguridad del sistema, y se ha comprobado que responde adecuadamente ante errores, fallos de conexión o uso indebido. La aceptación de estas pruebas implica que los objetivos del proyecto, en términos de fiabilidad, accesibilidad y experiencia de usuario, han sido alcanzados de forma satisfactoria.

Parte III

Manuales de la Aplicación

Capítulo 8

Manual de Instalación

Se explicará a continuación el proceso a seguir para instalar las aplicaciones de terceros necesarias, las configuraciones y puesta a disposición del entorno para lograr poner en funcionamiento la aplicación *web* desarrollada. Nótese que el proyecto está previsto para su lanzamiento local, con la intención de mantener un entorno controlado en el que realizar las pruebas y demostraciones necesarias. El despliegue real, entre otras mejoras, constituye parte del trabajo futuro, abordado en la sección 10.2.

Paso a detallar las instalaciones previas y las configuraciones a realizar:

- Para el proyecto *frontend* con *Angular*:
  - Instalar *Node.js* [6] en la versión recomendada. En mi caso, dispongo de la versión `v22.14.0` de *Node.js*. Puedes revisar la versión instalada ejecutando `node -version` (doble guion) desde el *cmd*. Puede que se requiera reiniciar la terminal tras instalar el *software* para que se reconozca la versión.
  - Descarga el *IDE* de *VSCode* (*Visual Studio Code*) [11] para trabajar en el código *frontend*. Las extensiones que ofrece simplifican el desarrollo de las aplicaciones *Angular*.
  - Tener disponible cualquier navegador de tu preferencia. En mi caso, he empleado *Mozilla Firefox* [34] para la gran mayoría de las fases de desarrollo. En ocasiones, empleé *Google Chrome* [4] para complementar el desarrollo. También he usado *Microsoft Edge* para realizar ciertas pruebas.
  - Instalar *Git* [5] para aplicar el control de versiones, optar a clonar repositorios, etc. Tras completar la instalación, se recomienda ejecutar en el *cmd* los comandos `git config -global user.name "Tu nombre"` y `git config -global user.email "Tu correo"`. Para conocer la versión instalada, se debe ejecutar el comando `git -version` (doble guion), en mi caso la `git version 2.48.1.windows.1`.
  - Instalar *Angular*, abriendo el *cmd* en modo Administrador y ejecutando el comando `npm install -g @angular/cli`. De esta manera, se logrará obtener el *CLI* (*Command Line Interface*) propio de *Angular*.

- Para resolver dudas acerca de *Angular*, se recomienda tener a mano la documentación oficial de instalación [2] de este *framework*. Para comprobar la versión instalada del *Angular CLI*, ejecutaremos el comando `ng -version` (doble guion) en el *cmd*. En mi caso, dispongo de la versión `19.1.8`.
- El proyecto *frontend* estará alojado en el repositorio facilitado por los tutores, en el directorio que se mencione en el capítulo A. Una vez descargado el material, debe acceder desde *VSCode* a la carpeta del proyecto *Angular* e instalar las dependencias, empleando `npm install`. En caso de presentar algún error o duda, estaré dispuesto a facilitar una *url* de *GitHub* para poder clonar el repositorio original. En ese caso, el proyecto alojado en *GitHub* será clonado, utilizando `git clone <url>`. Tras clonar el repositorio, al igual que antes, se deben instalar las dependencias desde la carpeta del proyecto, empleando `npm install`.
- En el proyecto se emplean los archivos de entorno, para almacenar las rutas base y ciertas claves de *APIs*. Por tanto, se debe ejecutar `ng generate environments` para su creación, puesto que son archivos que no se guardan al realizar un `push` en *GitHub*. En mi proyecto, como se explica en la sección 6.1.1, existe un directorio llamado `environments`, que alberga los archivos `environment.ts`, `environment.development.ts`, `environment.en.ts` y `environment.es.ts`. Por ello, habrá que crearlos y en ellos incluir las rutas base y claves necesarias que se explican más adelante en este mismo capítulo. Nótese que en el repositorio de entrega se agregará un archivo a modo de plantilla para entender cómo rellenarlo con las claves y rutas base.
- Para habilitar *HTTPS* en el entorno de desarrollo, para el *frontend*, debemos acceder a la consola de *XAMPP* (pues cuenta con la instalación de *OpenSSL*). Una vez allí, usaremos el comando `cd` para desplazarnos al directorio donde se aloje el proyecto de *Angular*, crearemos con `mkdir certs` una carpeta para almacenar las claves y nos desplazaremos de nuevo con `cd` a ese directorio. Ejecutaremos el comando `openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout key.pem -out cert.pem -subj "sujeto-apropiado"` para generar la clave privada `key.pem` y el certificado autofirmado `cert.pem`. Pasaremos de servir la aplicación Angular con `ng serve` a emplear `ng serve -ssl true -ssl-key ./certs/key.pem -ssl-cert ./certs/cert.pem` en su lugar. Ahora accederemos al *frontend* desde `https://localhost:4200`. Nótese que debemos decirle al navegador que confíe en este certificado autofirmado.

- Para el proyecto *backend* con *Spring Boot*:
  - Instalar la versión deseada de *Java* [7], en mi caso la versión es `openjdk version "23.0.1"2024-10-15`. Para consultar la versión instalada ejecutaremos el comando `java -version`. Para la instalación se recomienda seguir un tutorial avanzando si no se cuenta con los conocimientos básicos para realizar este tipo de instalaciones. Los pasos a seguir, resumidamente, son

los siguientes: descargar el archivo comprimido en base al sistema operativo, descomprimir, copiamos el archivo a la raíz del sistema operativo, configuramos las variables necesarias de entorno (en variables del sistema, `path`, agregamos la ruta del `/bin` de la instalación como nueva variable).

- Instalar *STS* (*Spring Tool Suite*) [10] como *IDE* para trabajar en el proyecto backend. Seleccionamos la versión necesaria en base al sistema operativo.
- De nuevo, se debe consultar en el repositorio facilitado por los tutores, el directorio referente al subproyecto *backend* que se mencione en el capítulo A. Una vez descargado el material, debe acceder desde *Spring Tool Suite* a la carpeta del proyecto *Spring Boot* y ejecutarlo con *Run As - Spring Boot App*. En caso de presentar algún error o duda, estaré dispuesto a facilitar una *url* de *GitHub* para poder clonar el repositorio original. En ese caso, clonaremos el proyecto usando `git clone <url>` y ejecutaremos desde *STS* como se menciona previamente.
- En este caso, se incluye el archivo `application.properties` en la entrega, pero se deben incluir manualmente algunas claves. Básicamente, las variables de este archivo que estén vacías, deben completarse manualmente al adquirirlas. Estas explicaciones relativas a los servicios adicionales se mencionan un poco más adelante.
- Para lograr servir el *backend* en HTTPS en el entorno local, debemos generar un *keystore*. Para ello, empleamos `cd` para desplazarnos a la ruta en la que se encuentre el `pom.xml` de nuestro proyecto. A continuación, ejecutamos el comando `keytool -genkeypair -alias <alias>-keyalg RSA -keysize 2048 -storetype PKCS12 -keystore keystore.p12 -validity 365 -storepass password -keypass password -dname "valor-tipo"`. Esto generará el archivo `keystore.p12`, que debemos mover a `/resources`. De esta forma, al tener ya la configuración de `application.properties` lista al haber clonado el proyecto, el *backend* se levantará en `https://localhost:8443`.

- Para la base de datos relacional:
  - Descargar *XAMPP* [12] con la intención de disponer del servidor de *Apache* y el módulo *MySQL* (*MariaDB*) que contiene. La instalación es sencilla, pero se debe seleccionar como módulos a instalar tanto *Apache* como *MySQL* (*MariaDB*). Una vez instalado, abriremos el panel y pulsaremos `Start` en los módulos necesarios.
  - El propio panel de *XAMPP* incluye *PhpMyAdmin*, al que accederemos desde `http://localhost/phpmyadmin/`. Nos facilitará la gestión de nuestra base de datos relacional, proporcionando una interfaz sencilla. Podremos acceder a *PhpMyAdmin* gracias al despliegue del servidor *Apache*.
  - Si deseamos clonar la base de datos del proyecto, accederemos a la sección *Importar* para cargar el archivo `.sql` que contenga las tuplas, configuración,

tablas y *triggers*. Dicho archivo se podrá encontrar en el directorio relativo a la base de datos, mencionado en el capítulo A. La base de datos de la entrega contiene una serie de tuplas de ejemplo, por lo que se facilitan las claves de inicio de sesión de las cuentas de administrador, por si se quisiera interactuar con la aplicación. Ambos usuarios administradores, con correos `miguelestebannavajo@gmail.com` y `casadamajuana@gmail.com`, tendrán la contraseña `123456`.

- Para los servicios adicionales:

  - Para obtener unas credenciales válidas de *APIs* para *Google Cloud Console* [18], se debe registrar como desarrollador en dicha página. Una vez obtenida la cuenta, debe crear un proyecto nuevo, habilitar las *APIs* requeridas para el proyecto (*Maps JavaScript API, Places API*) y limitarlas a la *URL* requerida. La clave de estas *APIs* se debe incluir en el archivo `environment.ts` del proyecto *frontend*, en concreto en la variable de entorno `googleKey` de la plantilla facilitada en la entrega. Además, la *API* de *Google JavaScript Maps* requerirá la creación de un `ID de mapa`, que de nuevo habrá que introducir en el archivo de configuración de entorno, en concreto en la variable de entorno `mapId` de la plantilla.
  - Para habilitar la *reCAPTCHA Enterprise API*, podrá realizarlo desde la propia *Google Cloud Console* o acceder a la dirección propia de *reCAPTCHA* [19] para obtener las claves. Obtendrá una *site key* para introducir en el archivo de entorno `environment.ts` (en la variable `siteKeyReCaptcha` de la plantilla) y una *secret key* para usar en el `application.properties` del *backend* (en la variable `google.recaptcha.secret`).
  - Para lograr que funcione el envío de correos a la bandeja de *Gmail*, es importante acceder a la cuenta deseada y activar la verificación en dos pasos. Tras esto, se debe acceder a la sección de contraseñas de aplicación [20], para generar una clave de aplicación válida que habrá que depositar en `application.properties` (en la variable `spring.mail.password`). Además, se debe incluir el correo electrónico a emplear en la variable `spring.mail.username`.
  - Para configurar la pasarela de pago con *PayPal* [21], es necesario crear una cuenta de desarrollador. Posteriormente, se debe acceder al *Dashboard*, ir a *My Apps & Credentials* y obtener el `Client ID`. Este dato se debe incluir en el archivo `index.html` (ver etiqueta `script` referente a *PayPal* y sustituir el `Client ID` antiguo por el obtenido) del *frontend* y en el `application.properties` del *backend* (en la variable `paypal.client-id`). Además, se debe de agregar a este último también el `secret key` de *PayPal*, para realizar las verificaciones oportunas (en la variable `paypal.secret`). Es recomendable modificar la variable `paypal.expected.merchant-id` del archivo `application.properties` por el identificador del perfil de comercio o receptor de pagos que se cree en el *sandbox* de *PayPal*.

- Se recomienda la instalación de *Postman* [8] para facilitar la realización de pruebas de los *endpoints* servidos desde el *backend*, resultando interesante para mejorar la *API REST*.
- Para una gestión más fácil del control de versiones del proyecto, recomiendo descargar *SourceTree* [9], una interfaz gráfica que acelera el uso de comandos propios de *Git*.
- Recomiendo el uso de ciertas extensiones de *VSCode* que pueden facilitar el uso de *Angular*, mejorando la experiencia del desarrollador. En este caso, pongo a disposición una hoja de instalaciones [3] facilitada por un experto en la materia como es Fernando Herrera.

Capítulo 9

Manuales de Uso

Este capítulo está orientado a explicar las posibles interacciones que tendrán los diversos tipos de usuarios con la aplicación. La idea principal es constituir un manual de uso que pueda emplear el cliente como guía.

En las diversas secciones, se mostrarán las pantallas disponibles para cada usuario, siguiendo los posibles flujos de ejecución y mostrando los *outputs* que se arrojarán en cada caso. Cabe destacar que, debido a la jerarquía de usuarios planteada, un usuario registrado podrá realizar cualquiera de las funcionalidades disponibles para un usuario anónimo. A su vez, los usuarios administradores tendrán acceso a los privilegios de usuarios anónimos y registrados, y gozarán de una serie de funcionalidades exclusivas.

9.1. Manual de Usuario Anónimo

Los usuarios anónimos accederán a la aplicación, llegando a la página principal [9.1]. Desde ahí, podrán desplazarse a otras páginas y pantallas mediante el *navbar* o los botones del encabezado.

En la página principal encontrarán un mensaje de bienvenida, seguido de un atractivo *carousel* de fotos interactivo [9.2].

Si continúan bajando, dispondrán de un botón que redirigirá a la pestaña de reservas [9.3], así como información relativa a los servicios adicionales de la casa [9.4].

Por último, encontrarán un listado de ventajas que podrán obtener si se registran [9.4], seguidas de un botón que redirigirá al menú de autenticación si así lo desean.

Al presionar el botón ubicado en el *navbar* llamado `Ver habitaciones`, se accederá a la página de habitaciones [9.5]. En ella, encontraremos una tabla contenida en un desplegable [9.6], que expone los espacios que componen la casa rural.

El usuario podrá filtrar los espacios por tipo, mediante los botones ubicados en la parte superior de la tabla. También podrá revertir el filtrado con el botón `Limpiar filtros`. Se muestra información como el nombre, el tipo, la capacidad de la estancia, una pequeña foto interactiva y un botón para acceder a los detalles.

Nótese que en caso de ocurrir un error al conectar con el servidor [9.7], se mostrará un aviso al usuario, para evitar confusiones en caso de no cargar la información.

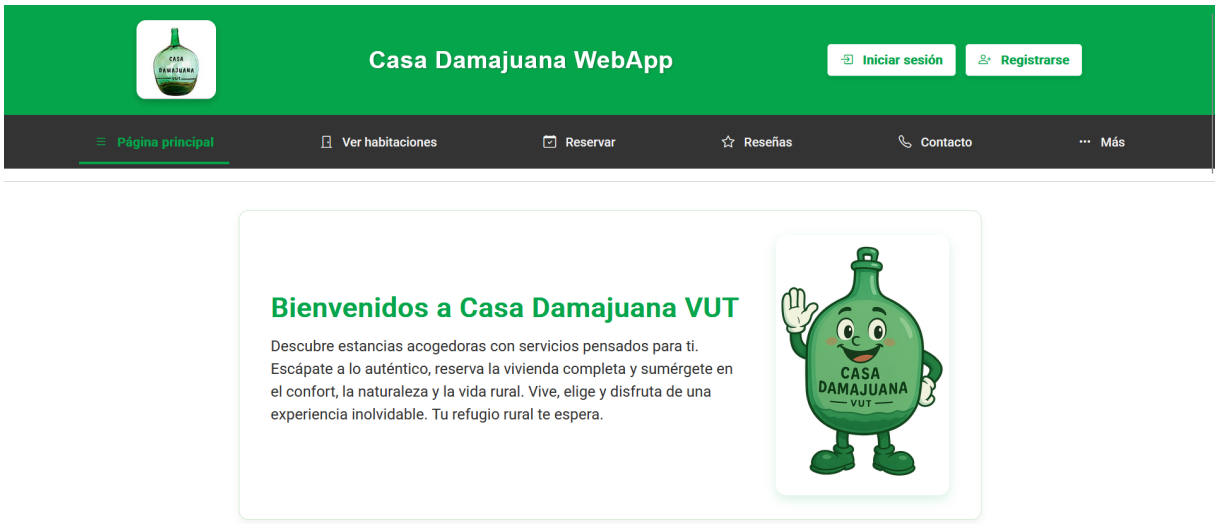

Figura 9.1: Página principal de la aplicación

Figura 9.2: *Carousel* de la Página principal

Pulsando el botón `Ver Detalles`, se accederá a una página que muestra una composición de fotografías de la estancia [9.8], así como los detalles de la habitación. El usuario podrá interactuar con las imágenes, consultar la capacidad de la estancia y leer ciertas observaciones personalizadas [9.9].

En caso de que un usuario trate de manipular la `URL` para acceder a una página de habitación usando un `id` inválido, se redirigirá a una página de error `404` [9.10]. De esta forma, se garantizará un experiencia de usuario mejor.

En la propia página de detalles de la habitación, dispondrá de un botón llamado `Volver a Habitaciones`, que como su propio nombre indica, permitirá retornar a la página de

Figura 9.3: Botón de reserva de la Página principal

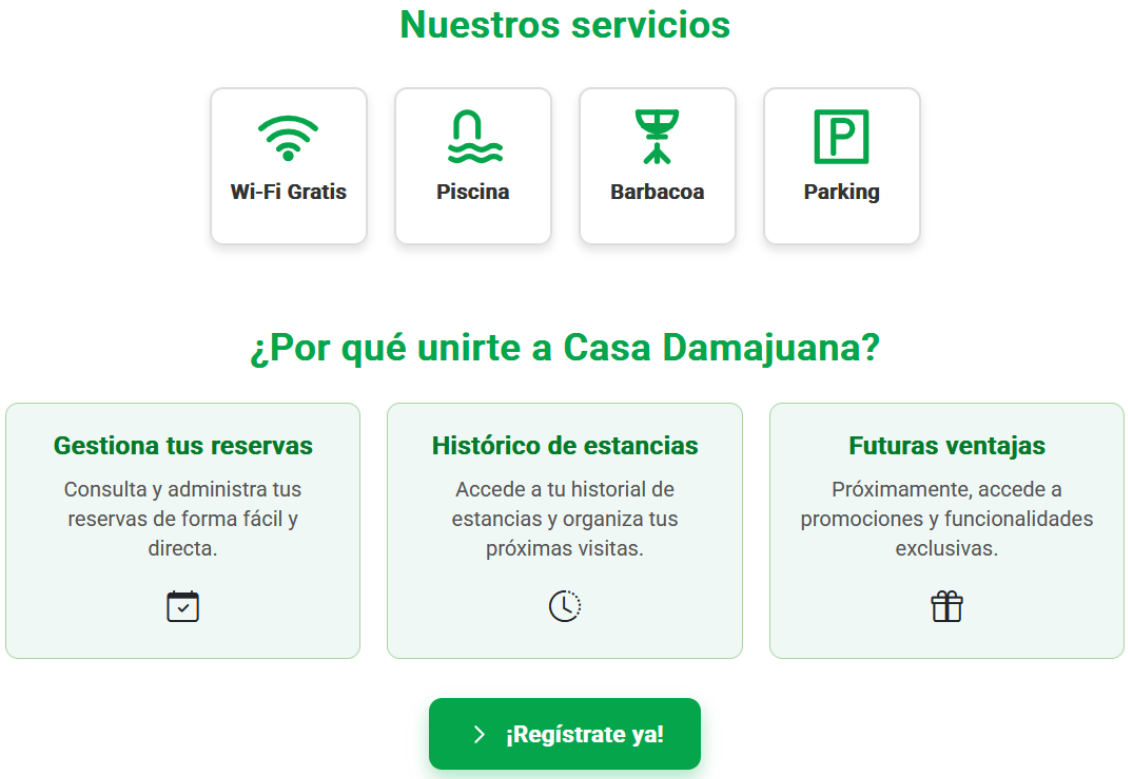

Figura 9.4: Servicios y ventajas de la Página principal

habitaciones original [9.5].

Al accionar el botón del *navbar* o el ubicado en el mensaje que anima a la realización de reservas en la página principal, se accederá a la página de reservas. En ella, se muestra cierta información relativa a la forma de reservas, las tarifas y la política de cancelación [9.11]. Nótese que si ocurre un fallo de servidor durante el acceso a la página, se ocultará el contenido de esta, mostrando el error oportuno [9.7].

Si el usuario baja en la propia página, encontrará un calendario de reservas [9.12]. Como explica la leyenda del propio calendario, los días pasados se mostrarán en color

Figura 9.5: Página de habitaciones de la aplicación

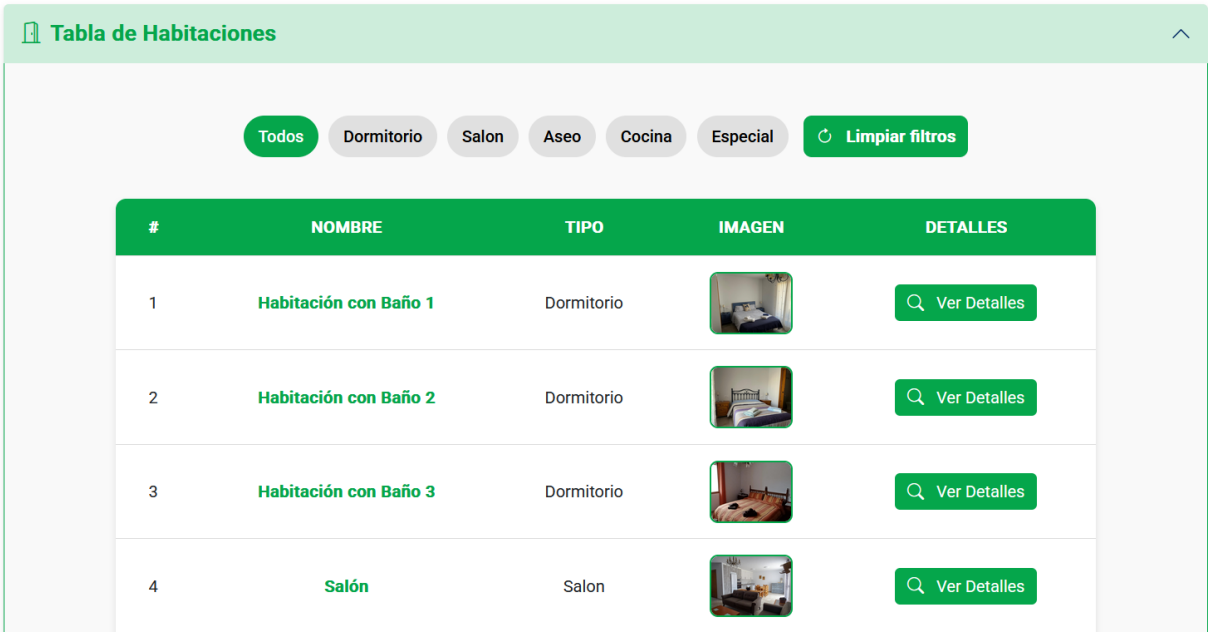

Figura 9.6: Tabla de la Página de habitaciones

gris, los días ocupados en rojo, el día actual en amarillo y las fechas disponibles en negro. Para seleccionar el rango temporal deseado, se debe hacer *click* en la fecha de inicio, y seguidamente hacer lo mismo para la fecha de fin. Se marcará el rango temporal en color verde, y se mostrará un mensaje de éxito a la vez que se despliega el formulario de reserva. En caso de querer revertir la selección, se puede usar el botón `Limpiar selección`.

Figura 9.7: Error del servidor

Figura 9.8: Página de habitación de la aplicación

Si el usuario selecciona un rango temporal inválido, es decir, que contenga fechas ya reservadas, se mostrará un mensaje de error y se limpiará la selección de fechas [9.13].

Una vez desplegado el formulario de reserva [9.14], el usuario debe rellenar adecuadamente los campos de nombre y correo electrónico. De esta manera, la reserva se pondrá a nombre del sujeto mencionado. Adicionalmente, podrá incluir un mensaje o comentario a la reserva. Se validará el formato de los campos. Cuando los datos sean aceptados por el sistema, debe presionar el botón `Seleccionar Reserva`, pasando al proceso de pago.

Se mostrará el resumen de la reserva: fechas, precio desglosado, descuentos, etc. Se habilitarán dos botones de pago, para poder efectuar la reserva [9.15].

El primero de ellos, desplegará una ventana de *PayPal*, para iniciar sesión en la cuenta

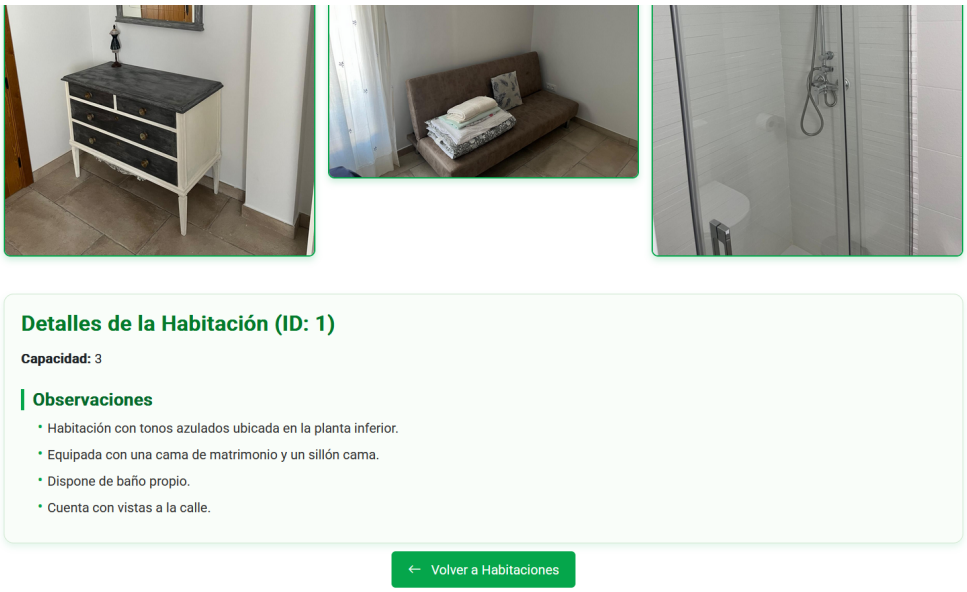

Figura 9.9: Observaciones de la Página de habitación

Figura 9.10: Error 404 al consultar habitaciones

de esta plataforma y aceptar el pago [9.16]. El segundo botón mostrará un formulario de pago con tarjeta [9.17], que deberá ser rellenado cuidadosamente por el usuario. En caso de abortar el proceso de pago, se mostrará un mensaje adecuado a la situación [9.18]. Si decide confirmar el pago, se comenzará a tramitar la reserva, de forma que se puedan dar tres situaciones:

- La reserva se podrá efectuar correctamente, redirigiendo al usuario a una página de resumen de la reserva [9.19]. Esta es la situación normal e ideal. Se mostrará una

Figura 9.11: Página de reservas de la aplicación

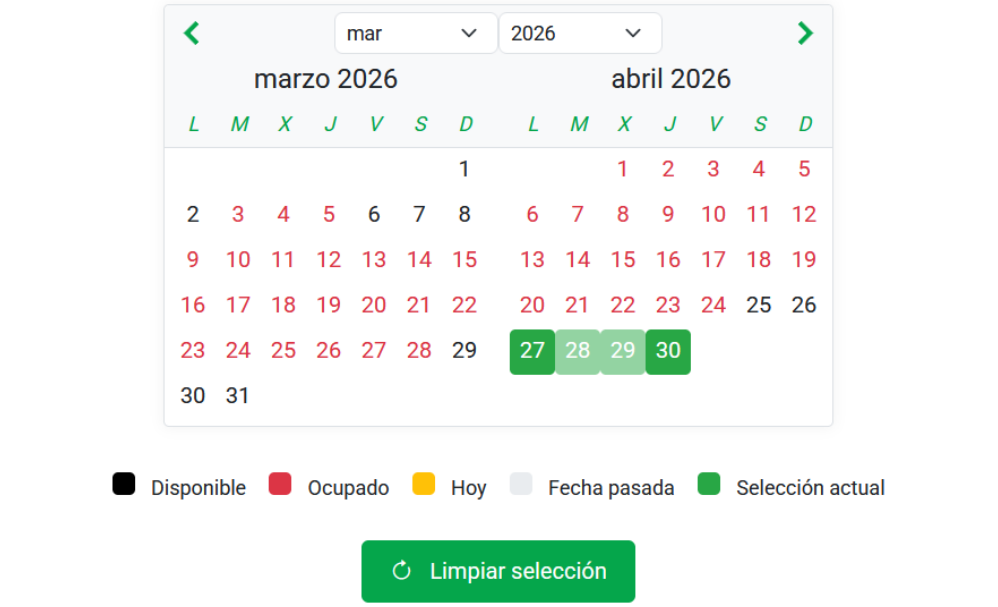

Figura 9.12: Calendario de la Página de reservas

tabla resumen, dando la opción al usuario mediante ciertos botones de imprimir [9.20], descargar el *PDF* de la reserva [9.21] o volver a la página principal.

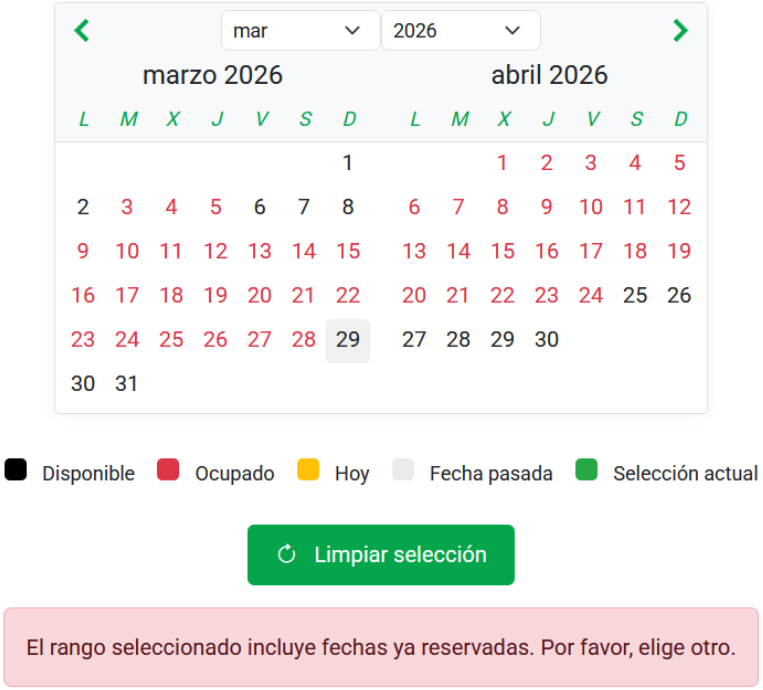

Figura 9.13: Error en el calendario de la Página de reservas

- Puede que la reserva se vea interrumpida por un fallo del servidor, llevando al usuario a una página de error [9.22]. No se realizará ningún cargo en la cuenta.
- En caso de que varios usuarios, al mismo tiempo, traten de reservar rangos temporales solapados o iguales, sólo uno de ellos obtendrá la reserva. El usuario que complete la reserva, seguirá el proceso mencionado en el primer punto, mientras que el segundo usuario será avisado mediante un mensaje de que alguien efectuó antes la reserva [9.23]. Obviamente, el usuario cuya reserva no se pudo completar no sufrirá ningún cargo.

En cuanto a la página de reseñas [9.24], los usuarios podrán acceder mediante el menú superior. Nótese que en ocasiones el proceso de carga se puede demorar unos segundos, por lo que se agrega un *loader* tematizado. Se presentará un listado de reseñas de usuarios satisfechos [9.25], provenientes de *Google*. El usuario podrá acceder al perfil del autor de la reseña, haciendo *click* en la foto de perfil. Del mismo modo, podrá comprobar la existencia de estas y más reseñas accionando el botón `Ver más reseñas en Google`, abriendo así una nueva pestaña de forma automática. En caso de presionar el botón `Deja tu reseña`, se redirigirá a una nueva pestaña de *Google*, que permitirá valorar la estancia en la casa.

Al acceder a la página de contacto, mediante la barra de navegación, se mostrará un mapa interactivo [9.26]. El usuario podrá visualizar los alrededores de la casa, presionar en la chincheta de ubicación para acceder a *Google Maps* para habilitar el *GPS*, e incluso visitar virtualmente el municipio. Las instrucciones de uso del mapa se establecen justo

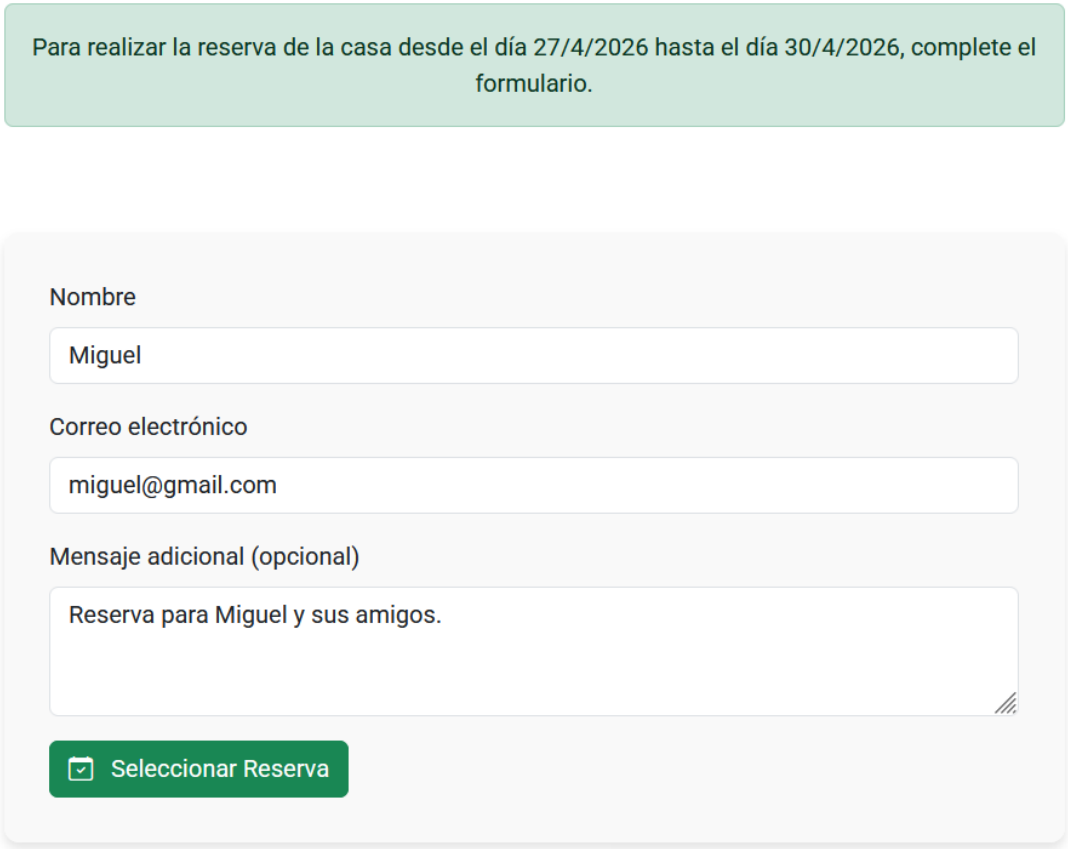

Figura 9.14: Formulario de reserva de la Página de reservas

debajo de este [9.27]. Por último, al final de la página se encuentran ciertos datos de contacto, así como un formulario automatizado [9.28].

Presionando en el teléfono o en el correo electrónico de la sección `Datos de contacto`, se podrá abrir una aplicación de mensajería para contactar de forma autónoma con los dueños. Mediante el formulario de contacto, se podrá depositar un mensaje en la bandeja de *Gmail* de la casa. Para ello, debe rellenar los campos solicitados, con un formato válido, superar el desafío *reCAPTCHA* [9.29] y accionar el botón `Enviar mensaje` [9.30]. Si el mensaje se envía correctamente, se mostrará una notificación de éxito [9.31], mientras que si algo falla, la notificación reflejará un error. En caso de caída del servidor, se ocultará el formulario de contacto, mostrando el error correspondiente [9.7].

Accediendo a la página `Más`, mediante el menú de navegación, el usuario podrá consultar cierta información adicional de la casa y sus alrededores [9.32]. En primer lugar, encontrará información relativa a las instalaciones de la casa. Más abajo, podrá consultar ciertas actividades y negocios de interés en los alrededores [9.33], mediante tarjetas visuales. Si hace *click* en las imágenes de las actividades, se redirigirá al usuario a la página *web* del sitio en cuestión. A continuación, encontrará un panel de normas del local, ideal para conocer la filosofía del negocio [9.34]. Por último, podrá consultar cierta información

Figura 9.15: Botones de pago de la Página de reservas

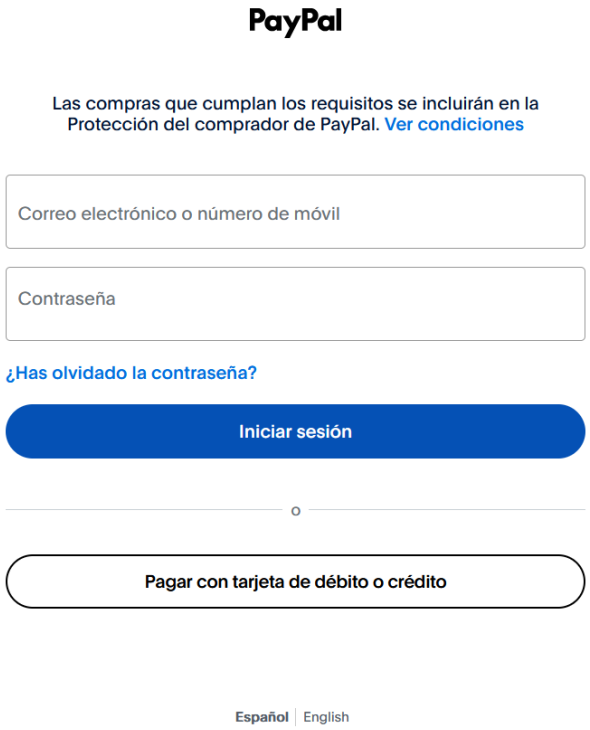

Figura 9.16: Ventana emergente para el pago con *PayPal*

cultural y gastronómica de la provincia de Segovia.

Los usuarios anónimos, tanto mediante los botones `Iniciar sesión` y `Registrarse`

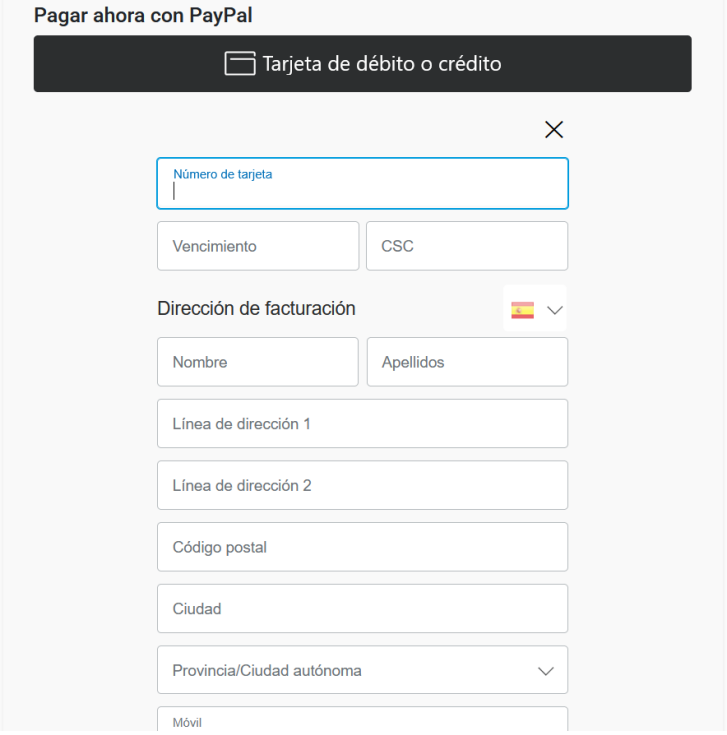

Figura 9.17: Formulario para el pago con tarjeta

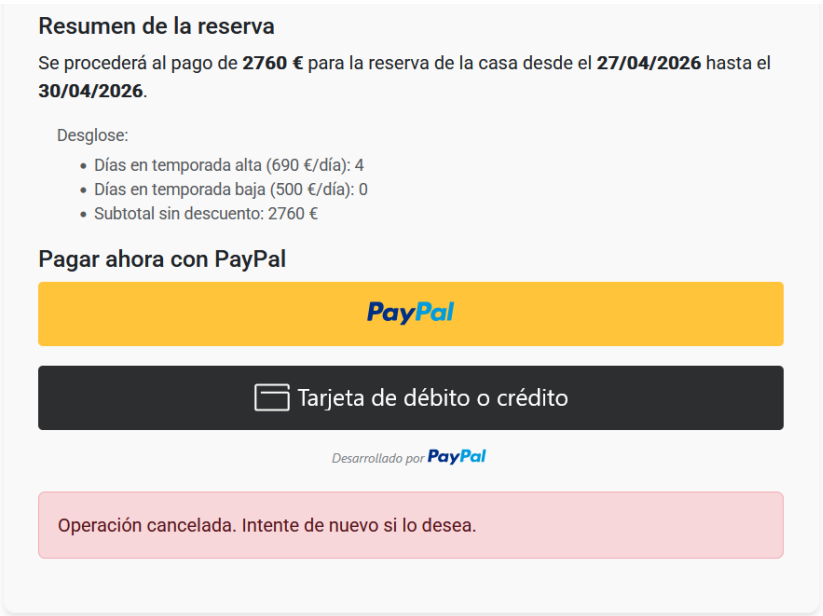

Figura 9.18: Mensaje tras cancelar la operación de pago

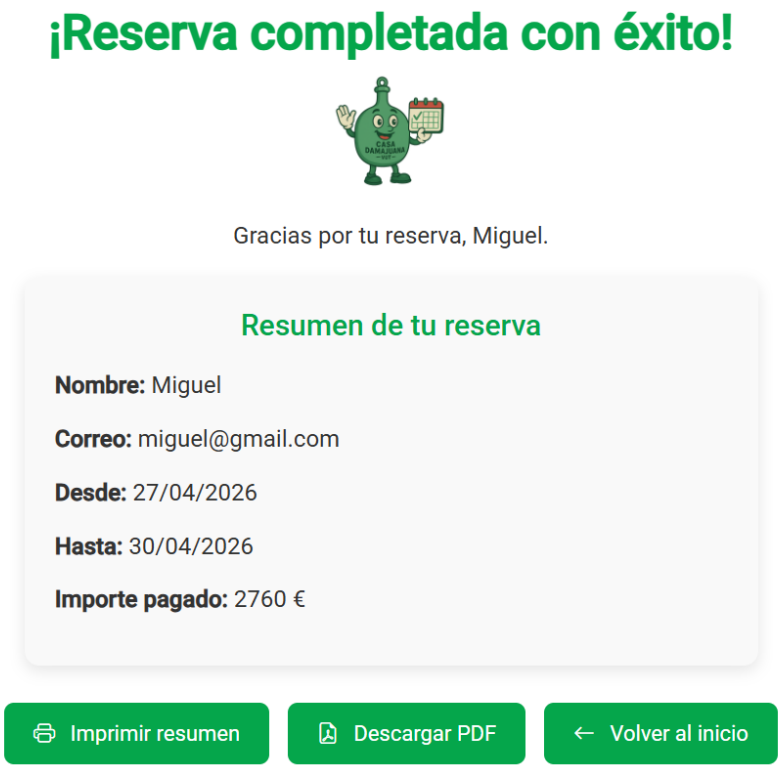

Figura 9.19: Pantalla de éxito al realizar la reserva

ubicados en la parte superior [9.35], como mediante el botón informativo ubicado al final de la página principal [9.4], accederán a la página de autenticación.

Desde allí, podrán darse de alta en la *web*, mediante la creación de una cuenta de usuario. Para ello, deben rellenar el formulario de la página de registro [9.36]. Este formulario presenta una serie de restricciones, que se mostrarán en forma de mensajes de validación, para ayudar al usuario a disponer de valores válidos. Tras rellenar los datos, presionará el botón `Crear cuenta`, haciendo efectivo el registro y causando una redirección al *login* [9.37].

Accediendo a la página de inicio de sesión, ya sea mediante el menú de navegación, una redirección o el accionamiento de un botón del encabezado, se mostrará un formulario [9.38]. Para iniciar sesión, el usuario debe proporcionar el correo electrónico y la contraseña vinculados a la cuenta. Se mostrarán mensajes de éxito [9.39] o error [9.40] en base a la exactitud de los valores. Tras iniciar sesión adecuadamente, se redirigirá al menú principal, mostrando en la parte superior el correo electrónico de la cuenta. De esta forma, el usuario pasará a ser un usuario registrado (o administrador).

Para descubrir las funcionalidades propias de los usuarios registrados (o administradores), se recomienda seguir los manuales expuestos a continuación.

Resumen de tu reserva

Nombre: Miguel
Correo: miguel@gmail.com
Desde: 27/04/2026
Hasta: 30/04/2026
Importe pagado: 2760 €

¡Gracias por confiar en Casa Damajuana! Esperamos que disfrutes de tu estancia.

Información adicional

¿Cómo cancelo mi reserva?
Si ha reservado desde una **cuenta de usuario** de Casa Damajuana WebApp, acceda al **Panel de Usuario**. Una vez ahí, ubique la reserva en su historial de reservas y presione el botón "Cancelar". La reserva se cancelará y se le reembolsará el dinero **automáticamente**.
Si ha reservado **sin registrarse** en la aplicación, **póngase en contacto con nosotros** usando alguno de los medios del apartado "Contacto". Trataremos de atenderle cuanto antes para tramitar la cancelación.

¿Hasta cuándo puedo cancelar mi reserva?
Podrá cancelar la reserva como máximo **un día antes de la fecha de inicio**.

¿Recibiré alguna penalización?
Si cancela la reserva con más de **7 días de antelación**, se le devolverá el **importe íntegro** pagado a la hora de reservar.
Si cancela la reserva **7 días o menos** antes del comienzo de esta, Casa Damajuana **descontará 100 €** del total a devolver en concepto de penalización.

vivienda de uso turístico
CASA DAMAJUANA

Figura 9.20: Pantalla de impresión de la reserva

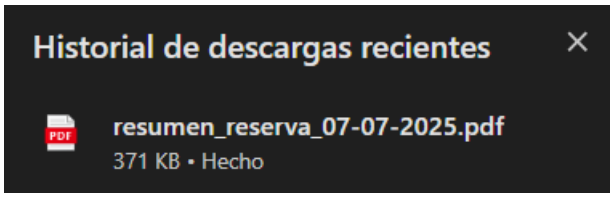

Figura 9.21: Documento *PDF* exportado de la reserva

9.2. Manual de Usuario Registrado

Una vez que un usuario logra iniciar sesión en la aplicación, se convierte en un usuario registrado (salvo que se le asigne el rol de administrador). Estos usuarios dispondrán de todas las páginas y funcionalidades citadas en la sección 9.1.

Mediante el menú de navegación, el usuario registrado podrá acceder a la sección llamada `Usuario`. En ella, tendrá acceso a dos desplegables, para la consulta y edición de sus datos de la cuenta y reservas efectuadas [9.41].

En el primer desplegable, se mostrarán el nombre, apellidos y correo electrónico de la cuenta del usuario [9.42]. Podrá cambiar cualquiera de estos datos, así como de la contraseña de su cuenta, mediante la edición del formulario. Cabe destacar que cada dato se podrá cambiar de forma opcional e individual. Por tanto, si por ejemplo desea cambiar el nombre de la cuenta, no tiene por qué modificar el correo electrónico. Mediante el botón `Reiniciar cambios` podrá establecer los valores por defecto, sin modificación alguna.

Figura 9.22: Pantalla de error al realizar la reserva

Figura 9.23: Mensaje de *overbooking* al realizar la reserva

Al presionar `Actualizar datos`, se solicitará una confirmación expresa del usuario para modificar los valores que hayan sido sustituidos [9.43]. Si se selecciona la opción `No, volver`, no se efectuarán los cambios. Si se confirma la acción, tras mostrar un mensaje de éxito se cerrará sesión por seguridad y se solicitará al usuario que vuelva a iniciar sesión [9.44]. En caso de error, se mostrará el mensaje correspondiente, de la misma forma que

Figura 9.24: Página de reseñas de la aplicación

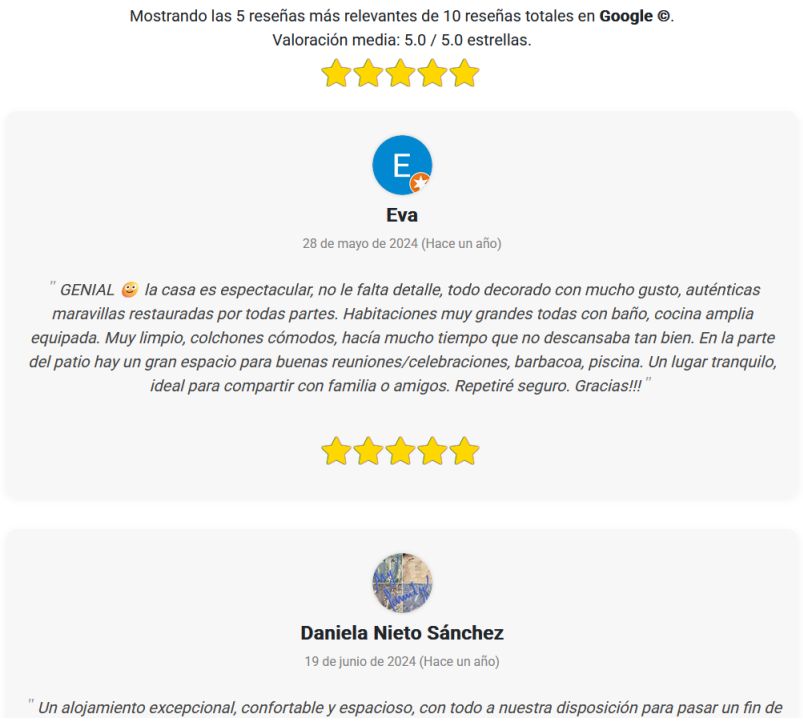

Figura 9.25: Opiniones de la Página de reseñas

se muestra el mensaje de éxito.

Adicionalmente, existe un botón llamado `Eliminar cuenta`, que al ser presionado solicitará una confirmación para eliminar la cuenta del usuario [9.45]. Si selecciona la opción

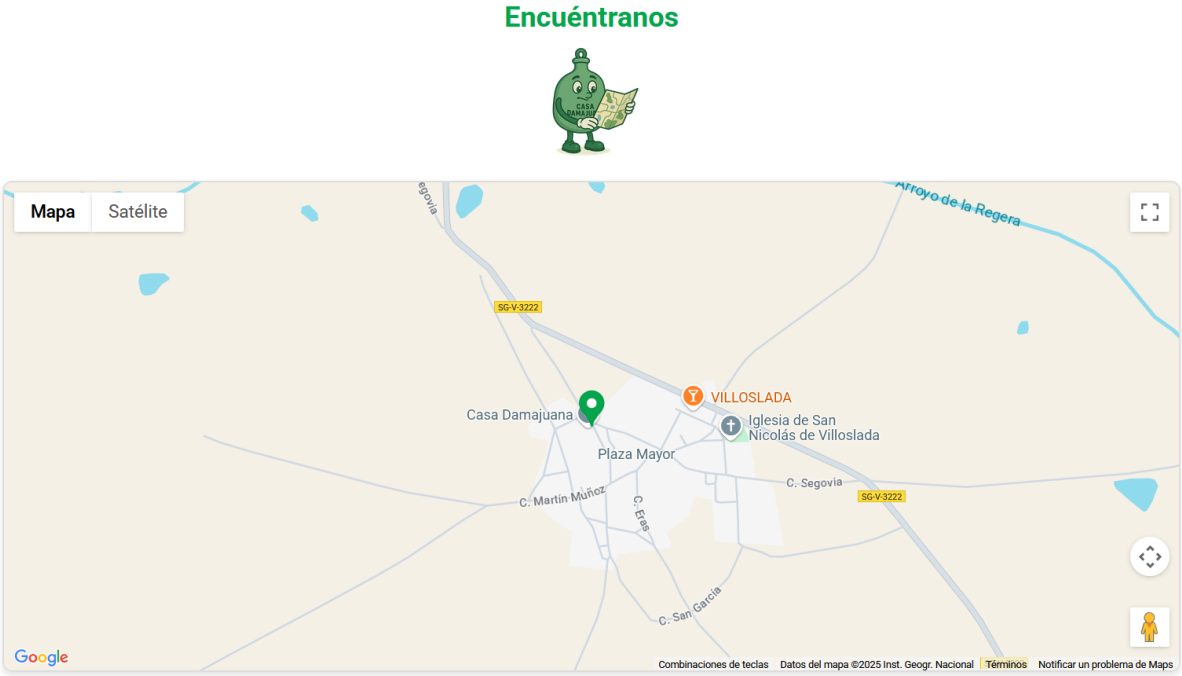

Figura 9.26: Página de contacto de la aplicación

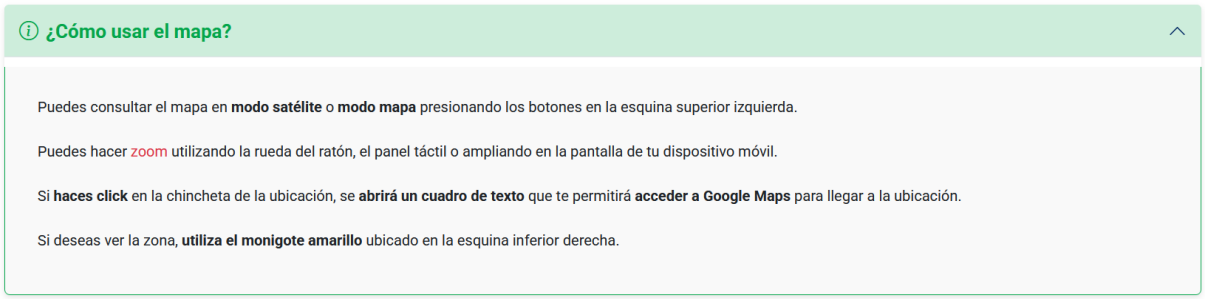

Figura 9.27: Instrucciones del mapa de la Página de Contacto

`No, volver`, se mantendrá en la página de usuario, mientras que si decide confirmar la acción, se mostrará un mensaje de éxito, seguido de una redirección al menú principal, habiendo cerrado sesión y borrado la cuenta [9.46]. En caso de error, se mostrará el mensaje informativo correspondiente.

En el segundo desplegable, encontramos la tabla de reservas del usuario [9.47]. Nótese que reunirá tanto las reservas efectuadas con su cuenta de usuario, como las reservas cuyo correo asociado a la reserva coincida con el correo asociado a la cuenta del usuario. Para obtener una disposición ordenada de las reservas, se paginan en grupos de diez reservas. La tabla de reservas muestra información como el nombre y *email* asociados a la reserva, las fechas, el estado, etc. Para una búsqueda cómoda de las reservas deseadas, se

Figura 9.28: Datos y formulario de la Página de contacto

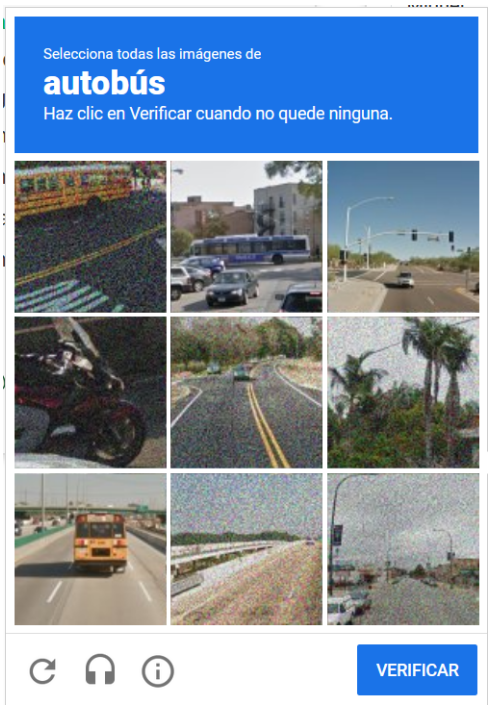

Figura 9.29: Desafío *Captcha* de la Página de Contacto

ofrecen una serie de filtros al usuario. En primer lugar, un buscador por nombre, correo

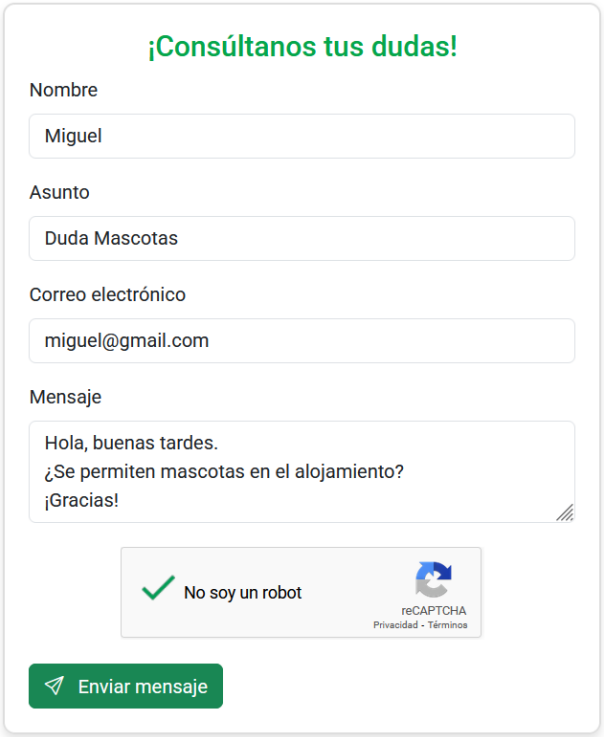

Figura 9.30: Envío del formulario de la Página de Contacto

o mensaje. Si el usuario escribe ciertos caracteres, se mostrarán las coincidencias en los campos mencionados. Por otro lado, se habilitan selectores de rangos, con intención de buscar las reservas de un determinado periodo de tiempo. Mediante un desplegable, se permite filtrar por estado de la reserva. A su vez, se permite ordenar los resultados en base a ciertos criterios, ascendente o descendentemente. Finalmente, se habilita un botón de limpiado de filtros, reestableciendo los valores.

Además, la tabla incluye funcionalidades ligadas a la obtención de información, como los botones de `Exportar PDF` y `Exportar CSV`, que descargan en el equipo del usuario la tabla de reservas filtradas.

Por último, cada reserva muestra un botón de cancelación. Obviamente, lo hace únicamente si esta es cancelable, es decir, debe de quedar mínimo un día para su comienzo. Al presionar el botón `Cancelar`, se requerirá la confirmación del usuario, a la par que se muestra un mensaje informativo que recuerda la importancia de leer la política de cancelación [9.49]. Si el usuario cancela la acción, no sucederá nada, mientras que si confirma la acción, se mostrará un mensaje de éxito o error, al estilo de los de eliminación o actualización de la cuenta. Si se cancela la reserva con éxito, se devolverá el dinero [9.50] en base a la política de la casa, actualizando el estado de la reserva en la tabla y liberando las fechas a otros usuarios.

En caso de que un usuario registrado desee cerrar sesión, deberá oprimir el botón ubicado en la parte superior [9.51], convirtiéndose así en un usuario anónimo. De hecho,

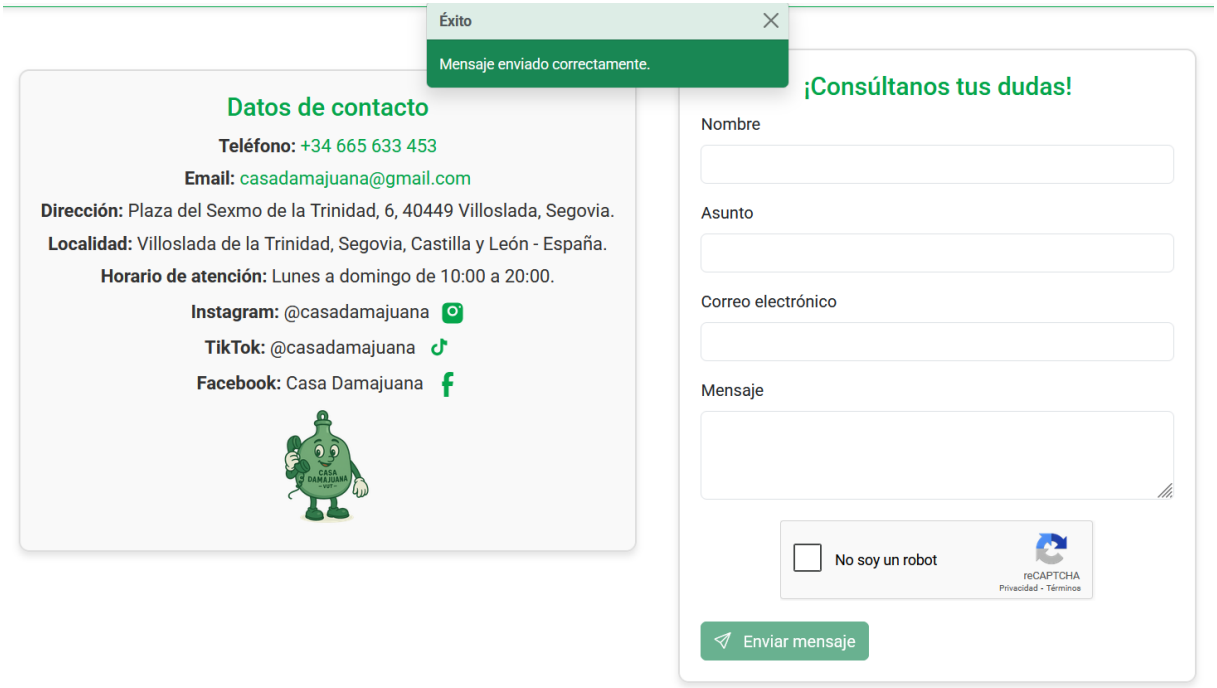

Figura 9.31: Éxito de envío del formulario de la Página de Contacto

Figura 9.32: Página de más de la aplicación

cada dos horas la sesión de usuario caducará, requiriendo un nuevo inicio de sesión para acceder a la sección exclusiva.

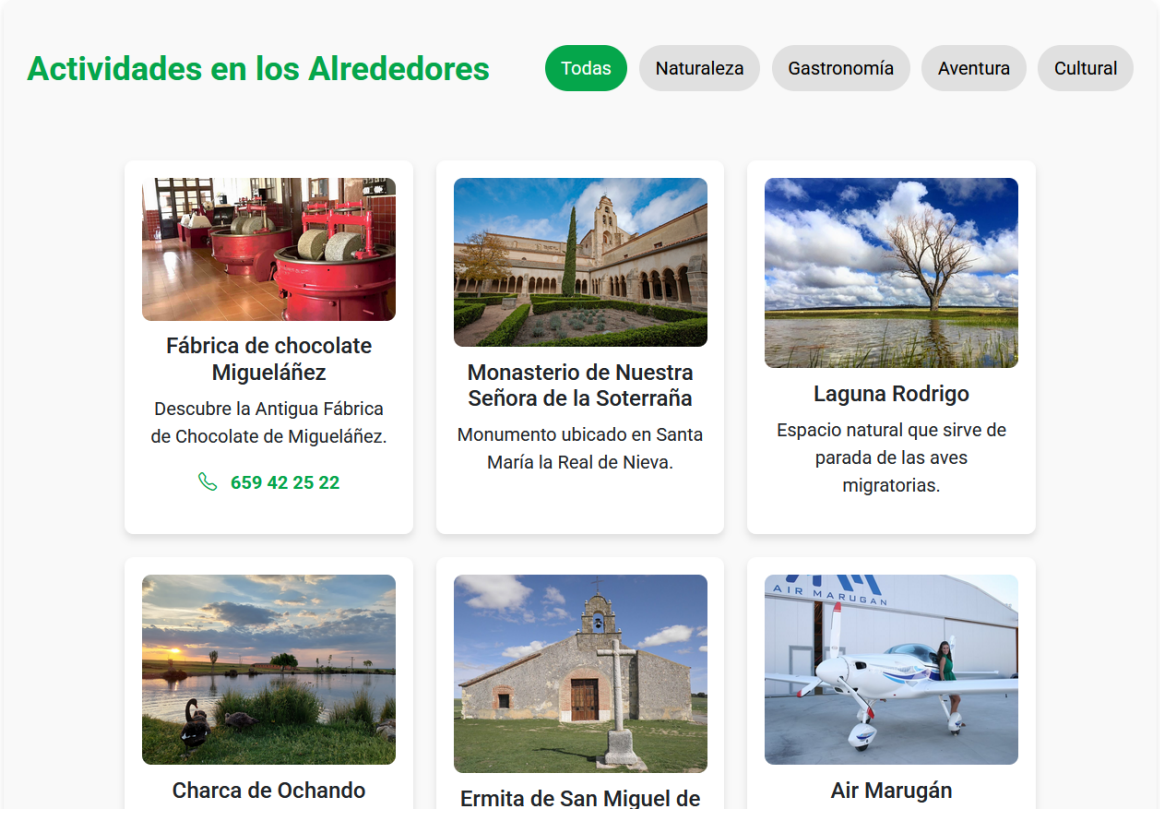

Figura 9.33: Actividades de la Página de más

9.3. Manual de Administrador

Para que un usuario registrado pase a ser administrador, el administrador del sistema deberá concederle manualmente este rol. Los administradores contarán con las funcionalidades y pantallas de usuarios anónimos y registrados, explicadas en las secciones 9.1 y 9.2. Además, dispondrán de un panel de administración exclusivo.

Para acceder al panel en cuestión, deben oprimir el botón `Admin`, ubicado en el menú de navegación. Accederán a una página con dos desplegables, uno para el registro de usuarios y otro para el registro de reservas completo [9.52].

En el primer desplegable, encontramos una tabla de usuarios, que muestra información como el nombre, apellidos, *email* y rol del usuario [9.53]. Además, facilita un botón de eliminación de la cuenta del usuario, que trataremos más adelante.

Permite el filtrado de usuarios, mediante diversos buscadores, botones y desplegables. El administrador puede escribir en el buscador para filtrar los usuarios en base al nombre, apellido o correo electrónico. Mediante los botones dispuestos, se permite el filtrado de usuarios por rol [9.54]. Por último, el administrador podrá revertir el filtrado oprimiendo el botón `Limpiar filtros`.

Si desea descargar dicha información filtrada, podrá usar los botones `Exportar PDF`

Figura 9.34: Normas e información de la Página de más

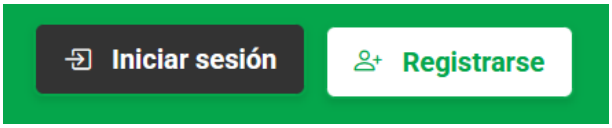

Figura 9.35: Botones de autenticación del menú de navegación

[9.55] o `Exportar CSV`. Esto realizará una descarga de todos los usuarios filtrados [9.56].

El botón `Eliminar` permite acabar con la cuenta de cierto usuario. Para ello, se requiere la confirmación del administrador, de modo que si rechaza se revertirá el proceso, y si confirma se eliminará el usuario de la aplicación [9.57]. Se mostrarán los mensajes de éxito o error correspondientes tras esta acción.

Por su parte, en el segundo desplegable encontraremos la tabla de reservas [9.58]. Reunirá todas las reservas de la aplicación. La tabla de reservas muestra información como el nombre y *email* asociados a la reserva, las fechas, el estado, etc. Para una búsqueda cómoda de las reservas deseadas, se ofrecen una serie de filtros al administrador. En primer lugar, un buscador por nombre, correo o mensaje. Si el administrador escribe ciertos caracteres, se mostrarán las coincidencias en los campos mencionados. Por otro lado, se habilitan selectores de rangos, con intención de buscar las reservas de un determinado pe-

Figura 9.36: Página de registro de la aplicación

riodo de tiempo. Mediante un desplegable, se permite filtrar por estado de la reserva. A su vez, se permite ordenar los resultados en base a ciertos criterios, ascendente o descendentemente [9.59]. Finalmente, se habilita un botón de limpiado de filtros, reestableciendo los valores. Para mejorar el orden, debido a la existencia de numerosas reservas, se establece una paginación de diez en diez.

Además, la tabla incluye funcionalidades ligadas a la obtención de información, como los botones de `Exportar PDF` [9.60] y `Exportar CSV`, que descargan en el equipo del administrador la tabla de reservas filtradas [9.48].

Por último, cada reserva muestra un botón de cancelación. Obviamente, lo hace únicamente si esta es cancelable, es decir, debe de quedar mínimo un día para su comienzo. Al presionar el botón `Cancelar`, se requerirá la confirmación del administrador [9.61]. Si este cancela la acción, no sucederá nada, mientras que si confirma la acción, se mostrará un mensaje de éxito o error, tal y como podían hacer los usuarios registrados. Si se cancela la reserva con éxito, se devolverá el dinero en base a la política de la casa al usuario que efectuó dicha reserva, actualizando su estado en la tabla y liberando las fechas a otros usuarios.

Si el administrador desea cerrar sesión, podrá hacerlo mediante el botón `Cerrar sesión`, ubicado en el encabezado de la aplicación [9.62]. Su sesión caducará de forma automática a las dos horas.

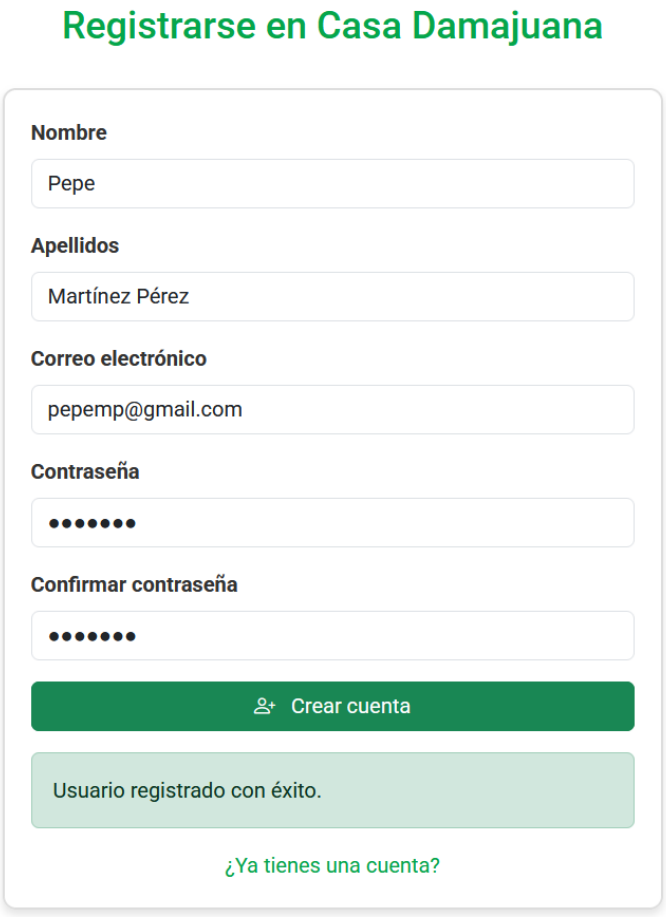

Figura 9.37: Éxito en el formulario de la Página de registro

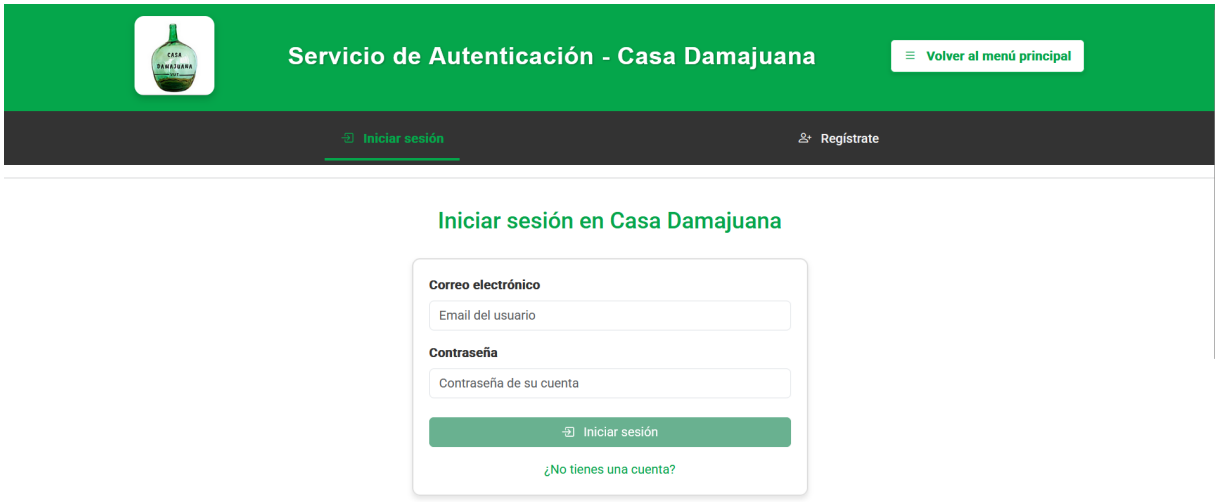

Figura 9.38: Página de inicio de sesión de la aplicación

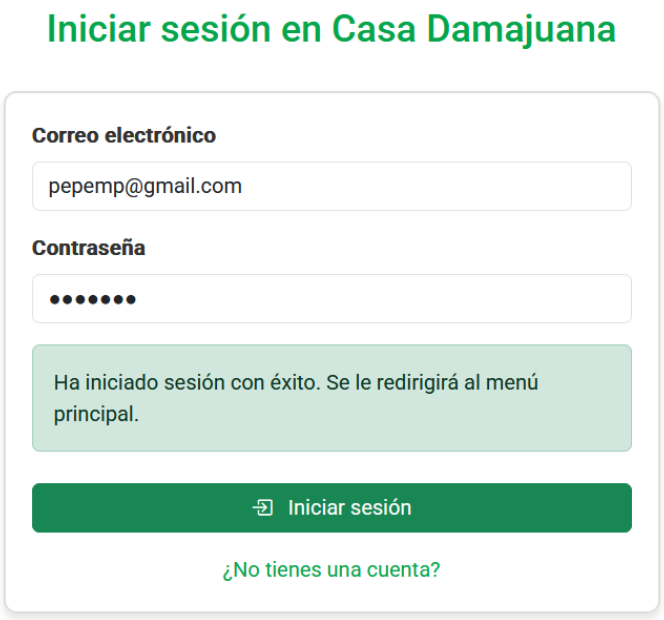

Figura 9.39: Éxito en el formulario de la Página de inicio de sesión

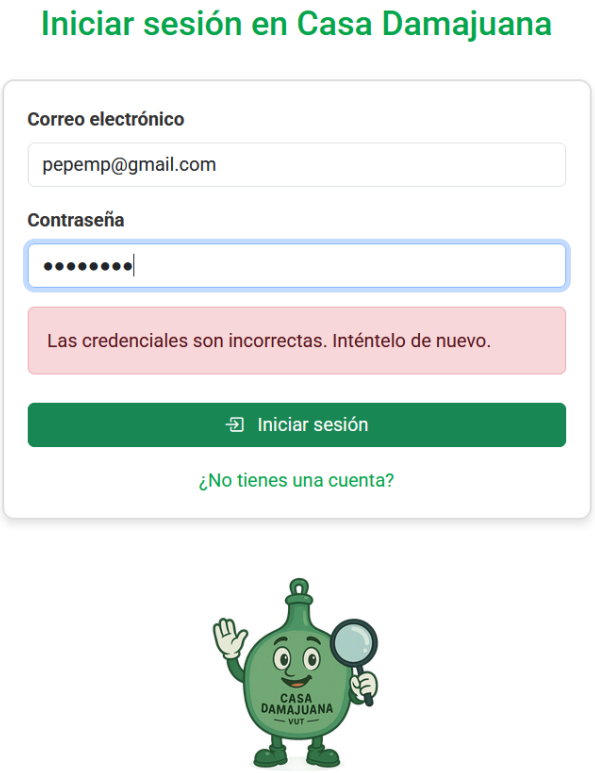

Figura 9.40: Error en el formulario de la Página de inicio de sesión

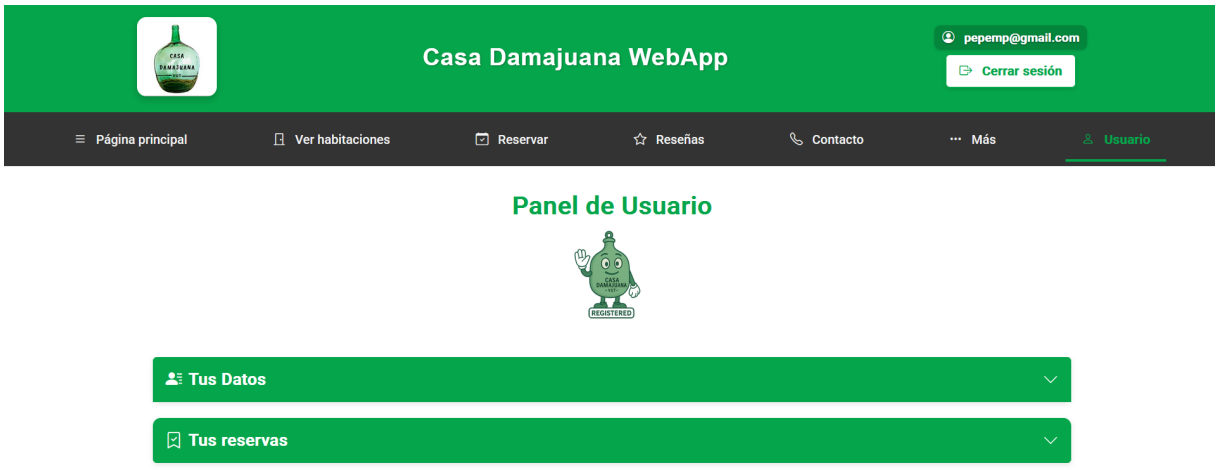

Figura 9.41: Página de usuario de la aplicación

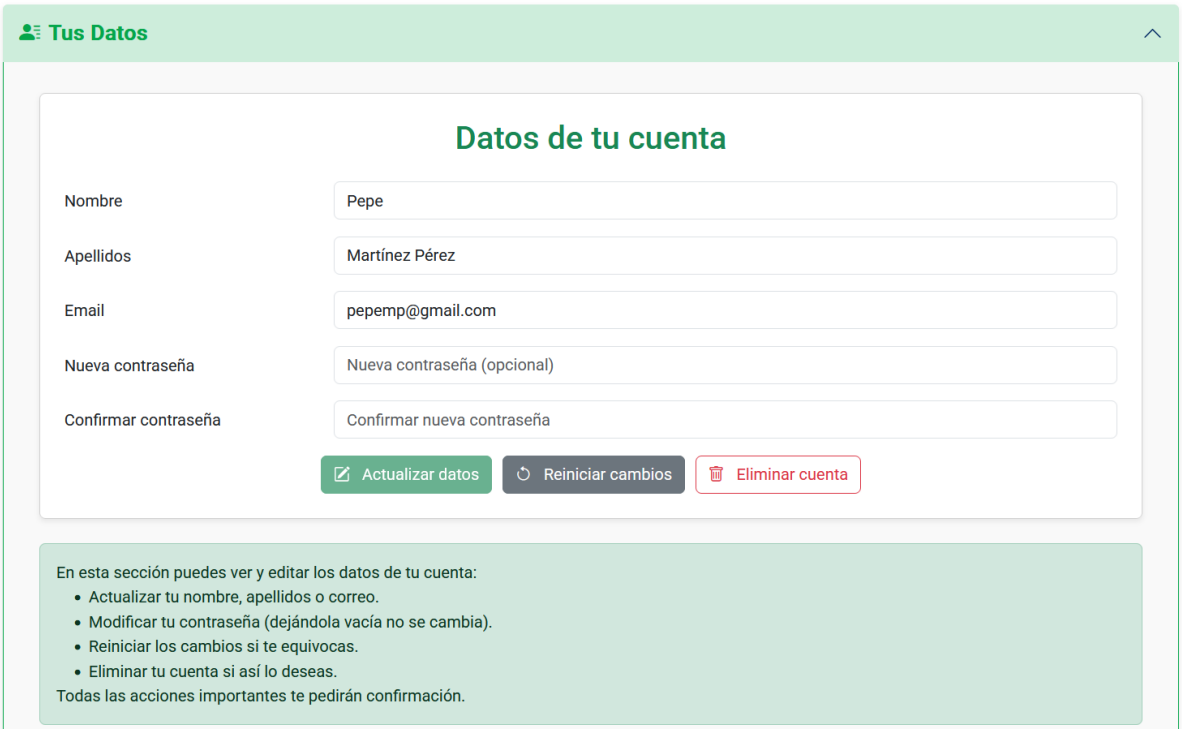

Figura 9.42: Tabla con datos de la cuenta de la Página de usuario

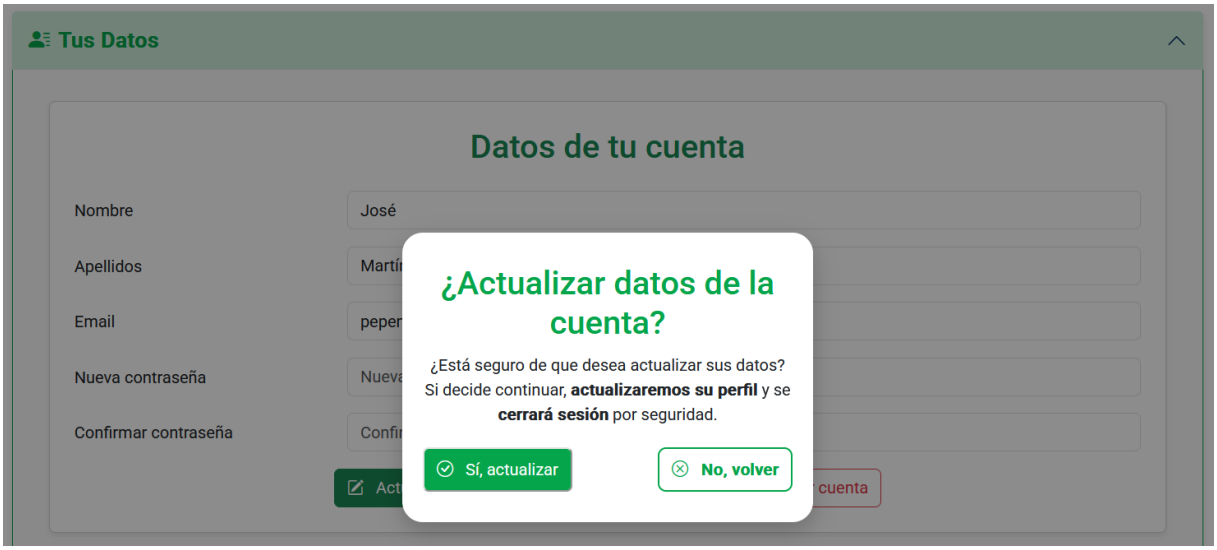

Figura 9.43: Actualización de datos de la cuenta

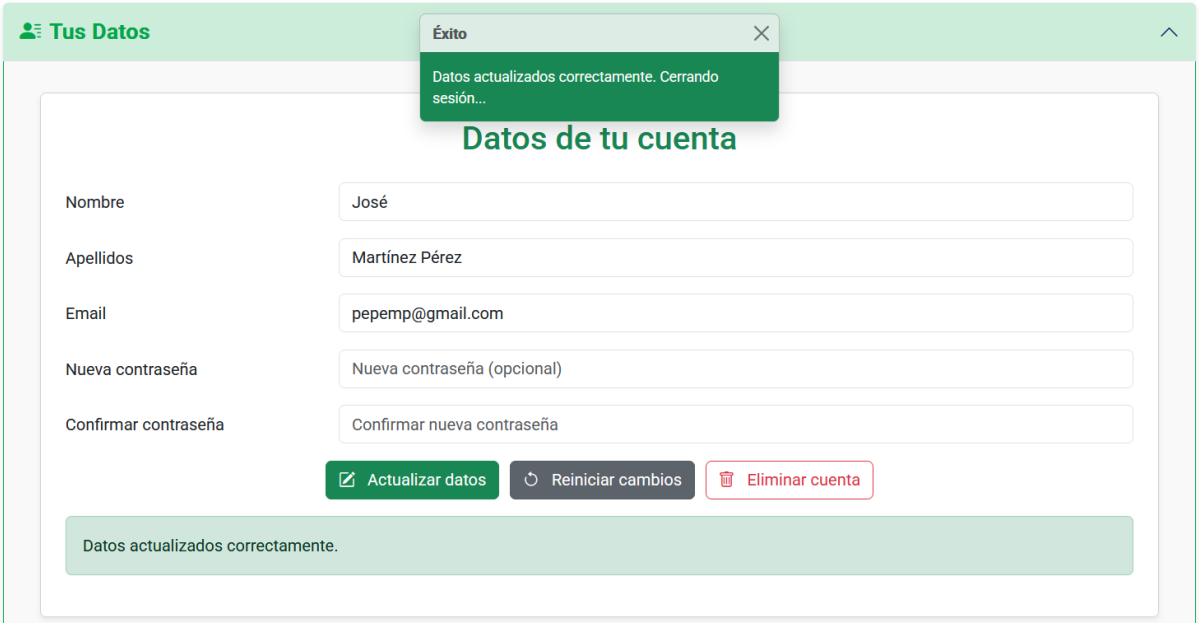

Figura 9.44: Éxito en la actualización de datos de la cuenta

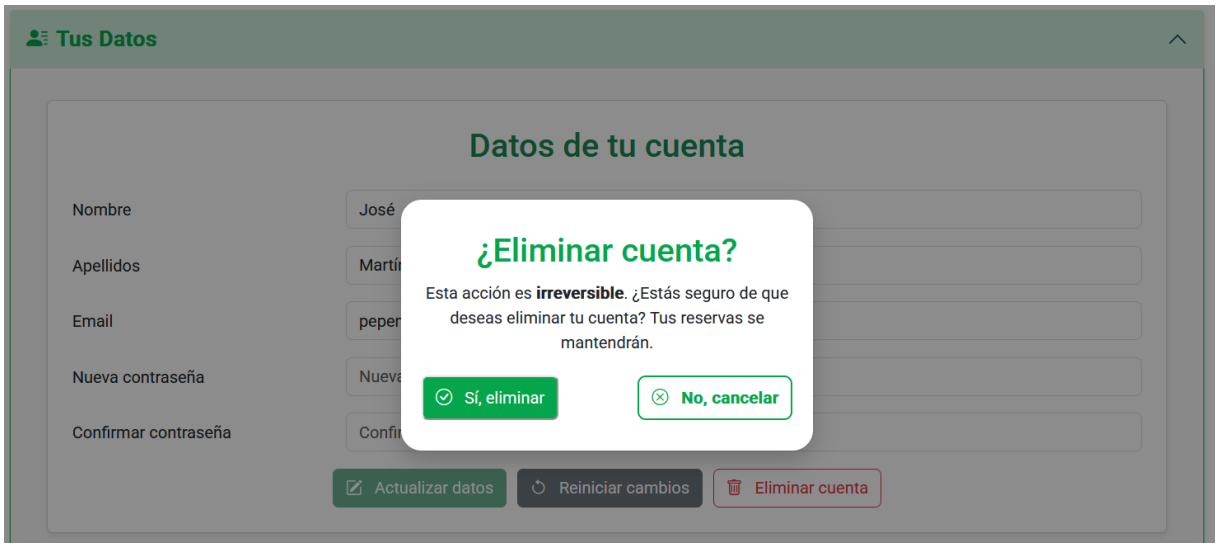

Figura 9.45: Eliminación de la cuenta

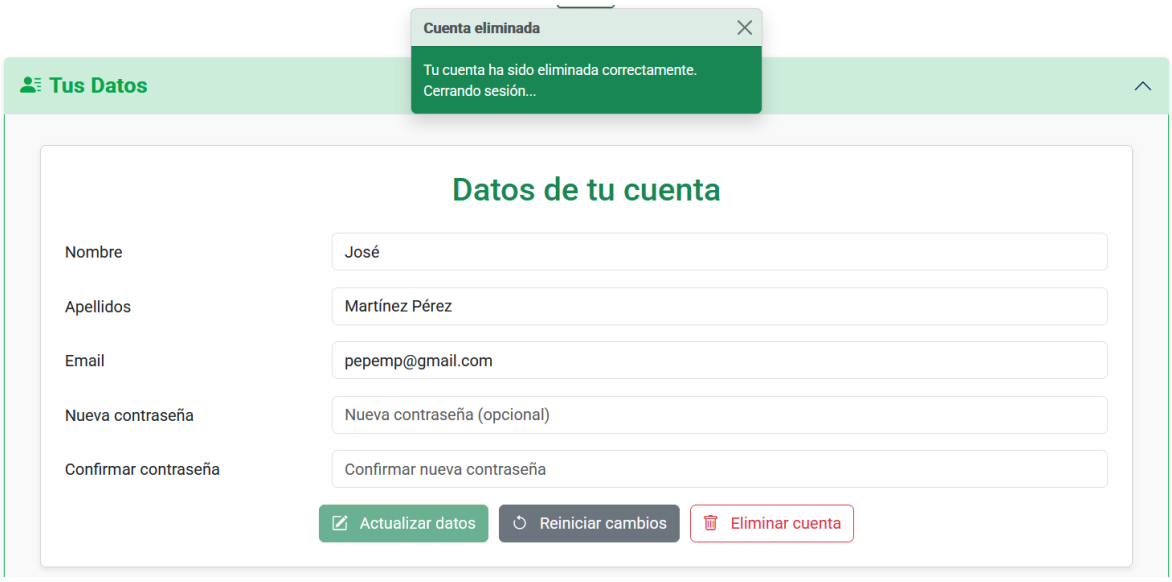

Figura 9.46: Éxito en la eliminación de la cuenta

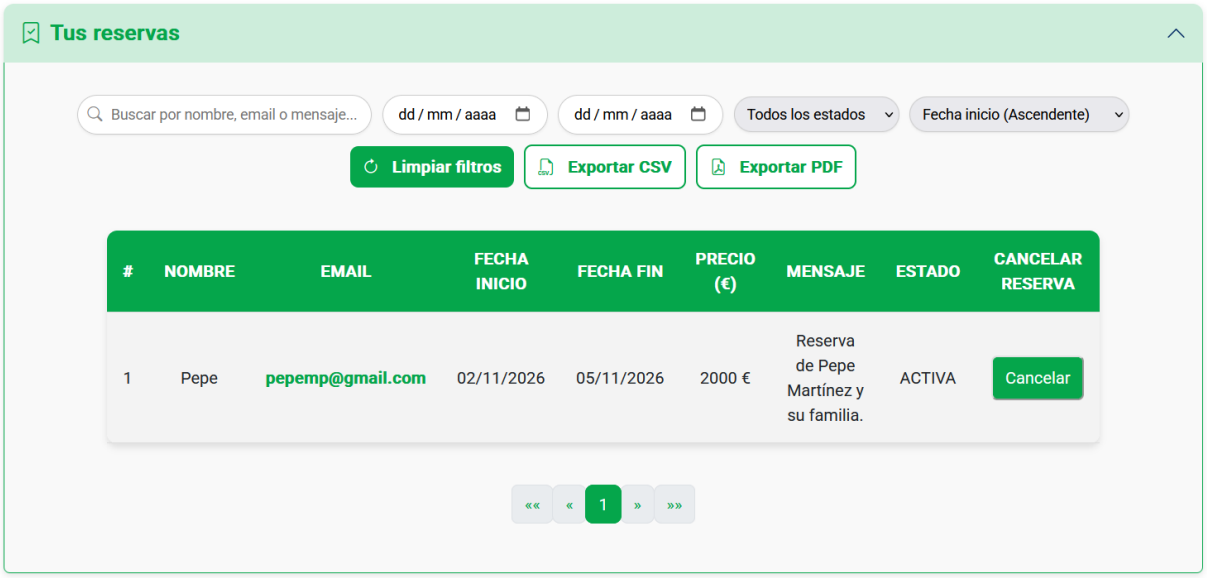

Figura 9.47: Tabla de reservas de la Página de usuario

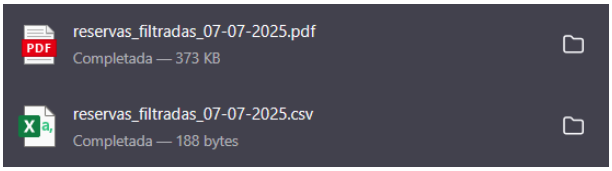

Figura 9.48: Exportar reservas desde la Página de usuario

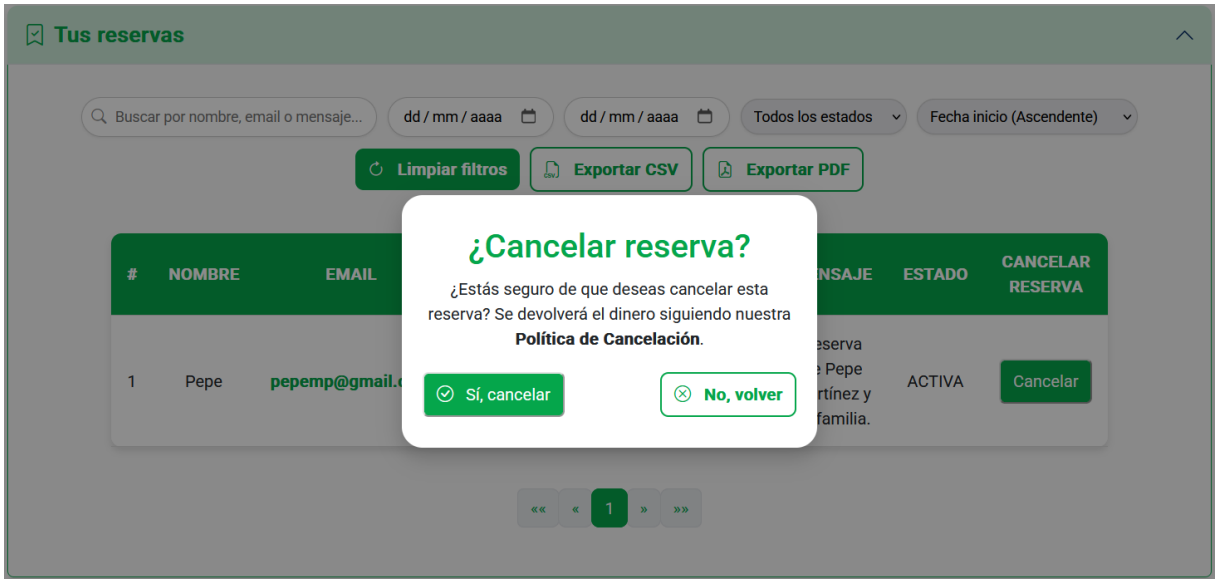

Figura 9.49: Cancelar reservas desde la Página de usuario

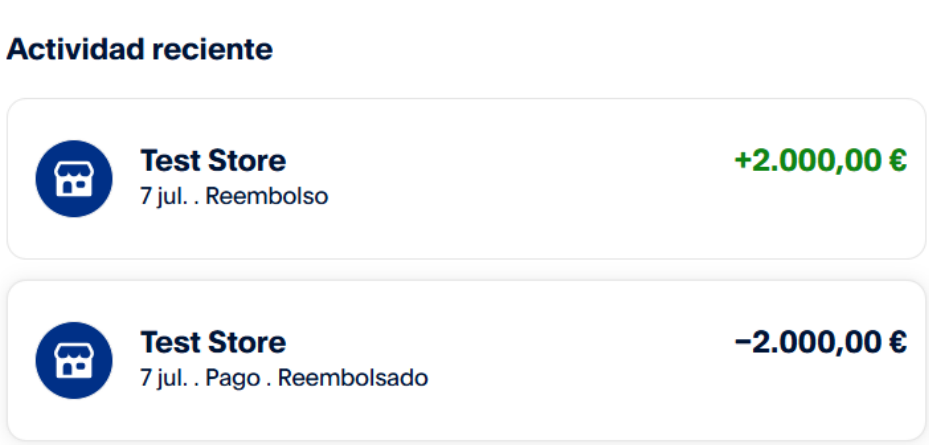

Figura 9.50: Reembolso del dinero al cancelar una reserva

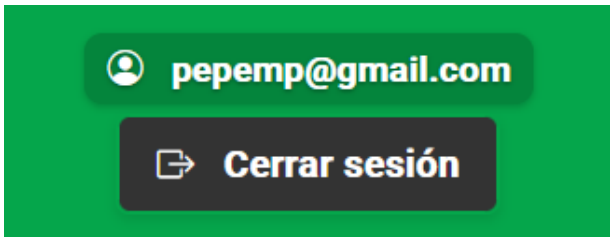

Figura 9.51: Cerrar sesión de forma manual

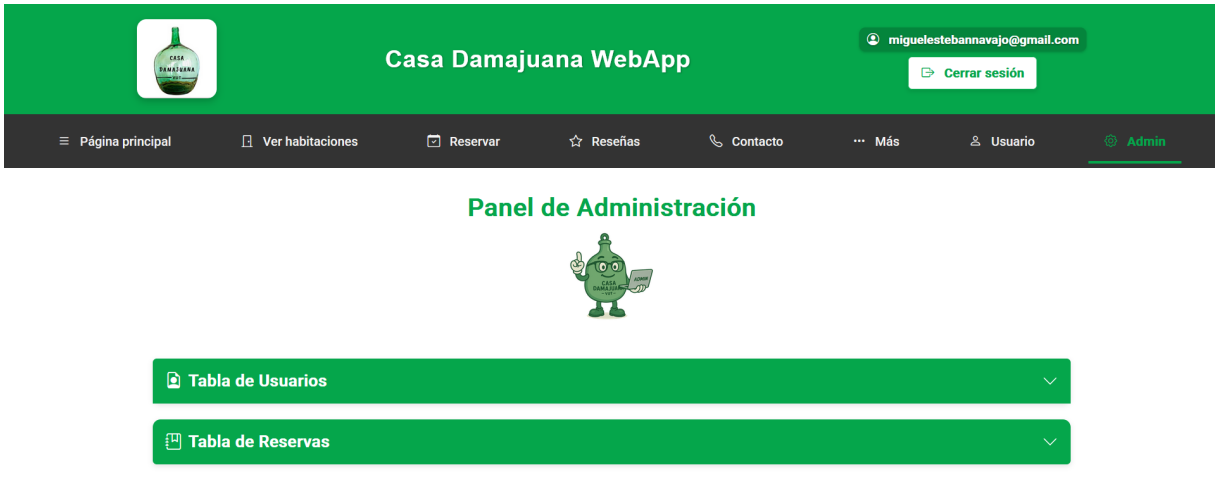

Figura 9.52: Página de administrador de la aplicación

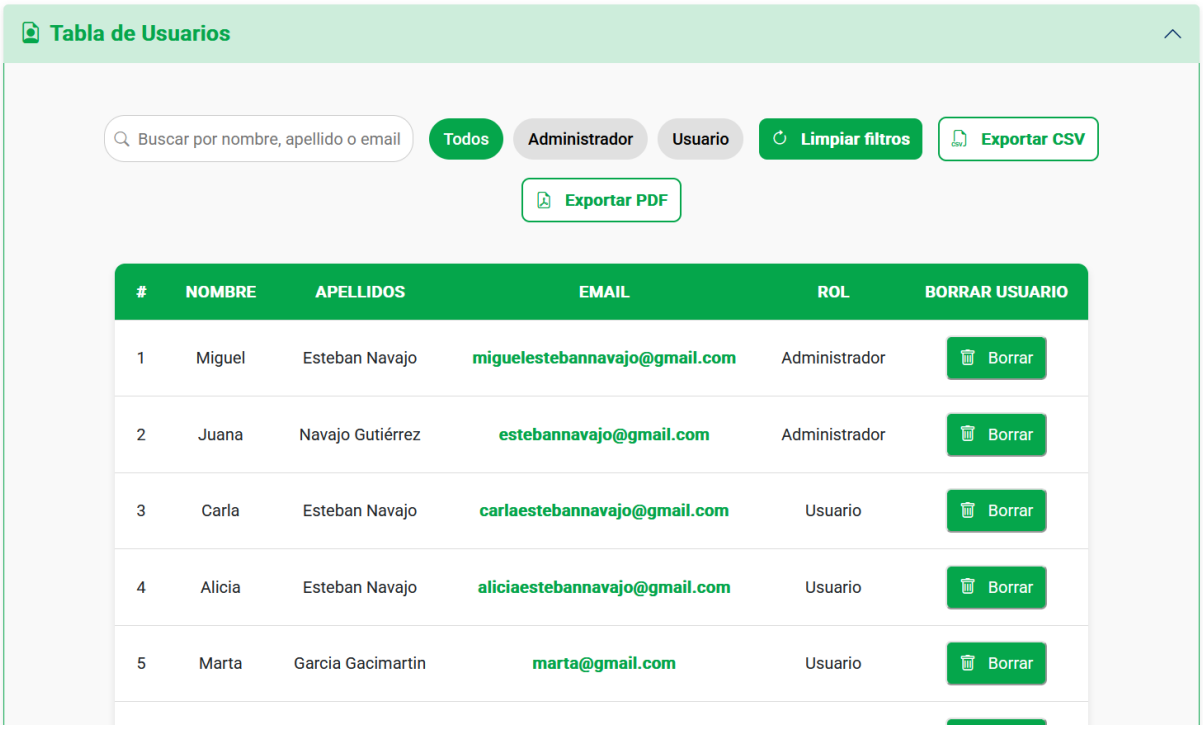

Figura 9.53: Tabla de usuarios de la Página de administrador

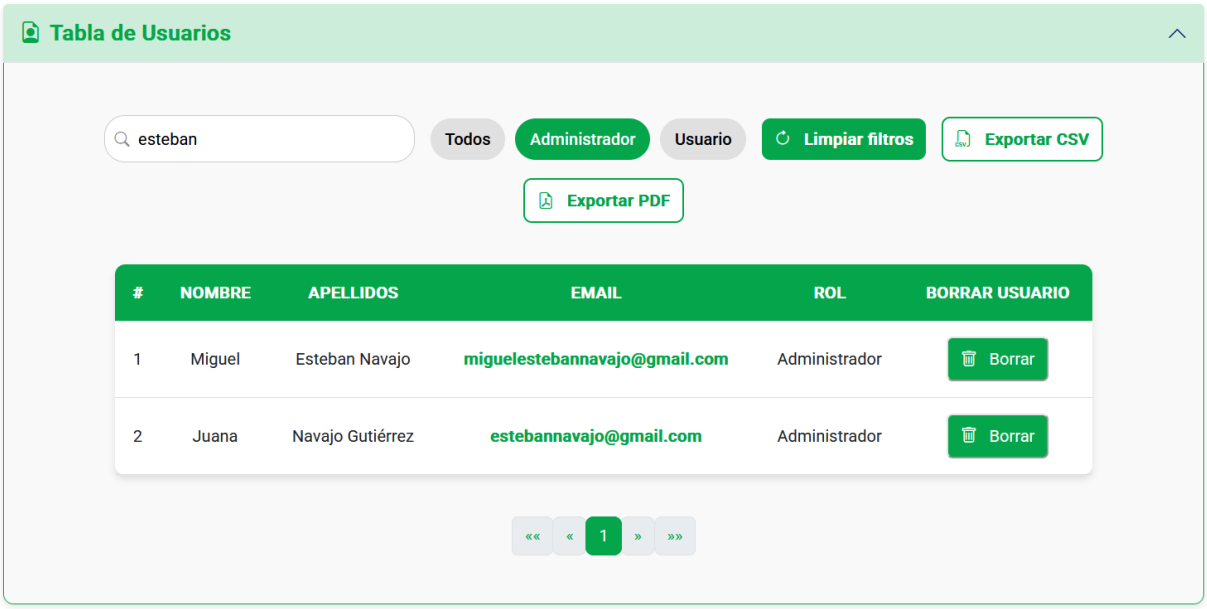

Figura 9.54: Filtrado de usuarios en la Página de administrador

Usuarios Filtrados

Listado de usuarios de Casa Damajuana WebApp

Datos del 07/07/2025 - Obtenidos según los filtros aplicados en el Panel de Administrador

#	ID interno	Nombre	Apellidos	Email	Rol
1	1	Miguel	Esteban Navajo	miguelestebannavajo@gmail.com	Administrador
2	2	Juana	Navajo Gutiérrez	estebannavajo@gmail.com	Administrador

Figura 9.55: *PDF* exportado de usuarios en la Página de administrador

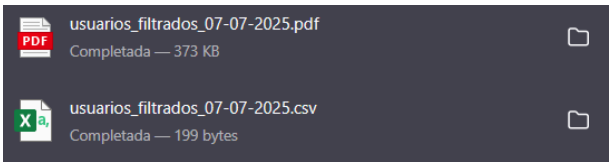

Figura 9.56: Exportación de usuarios en la Página de administrador

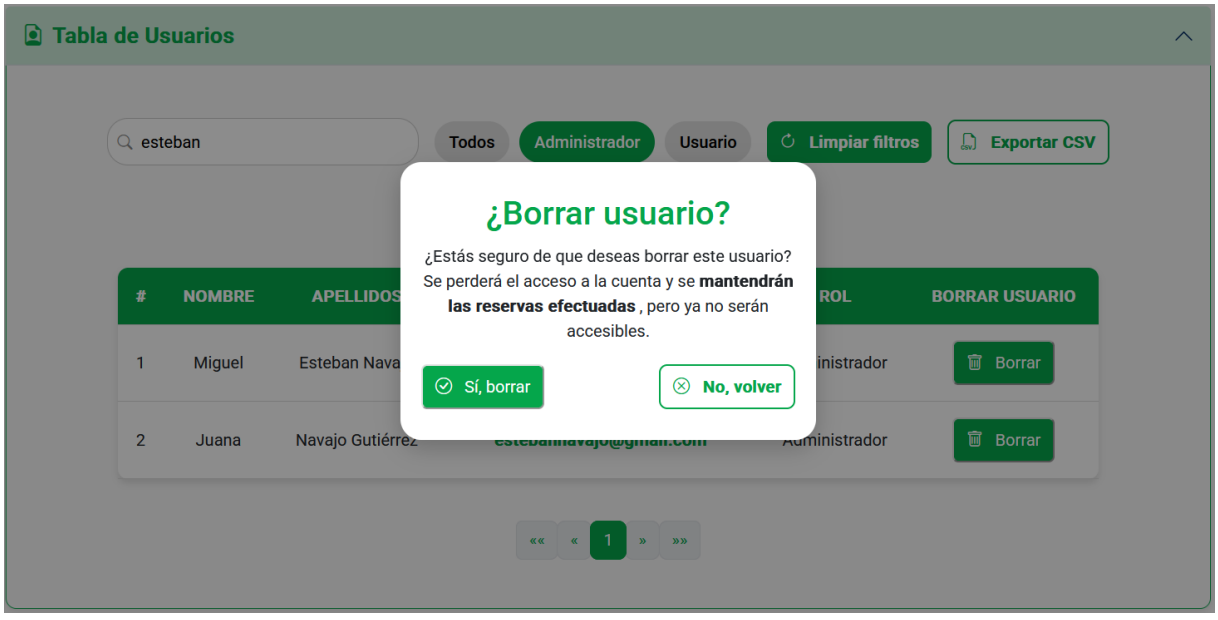

Figura 9.57: Borrado de usuarios en la Página de administrador

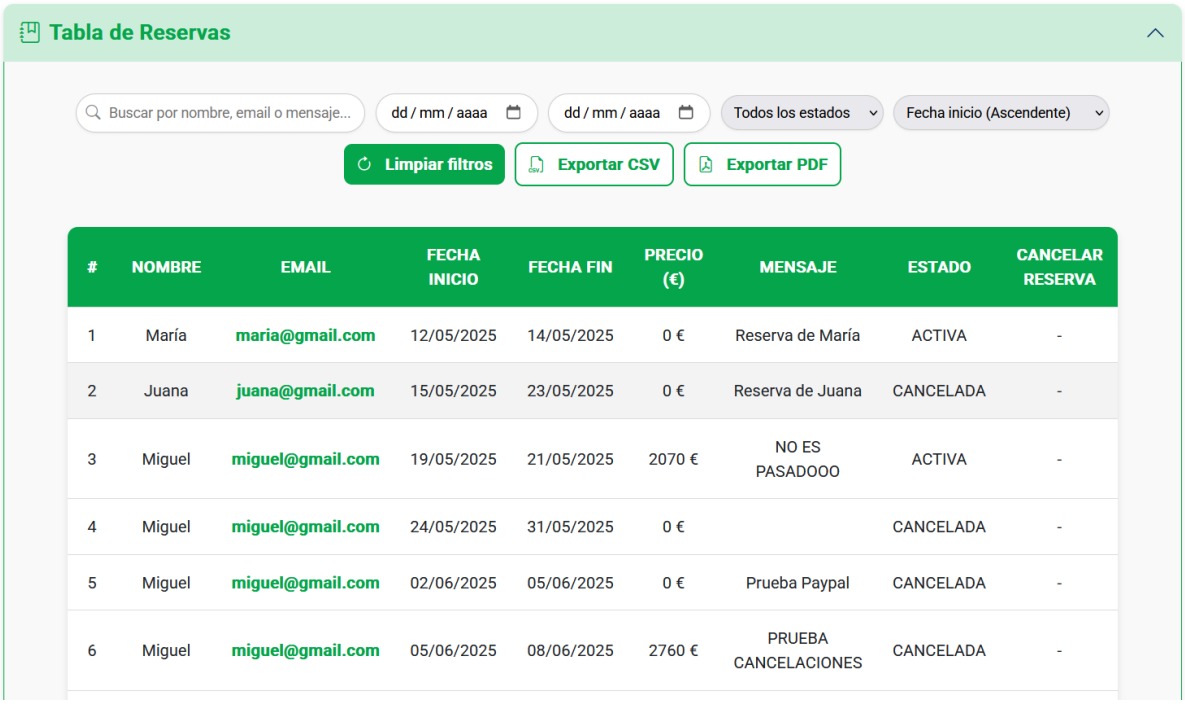

Figura 9.58: Tabla de reservas en la Página de administrador

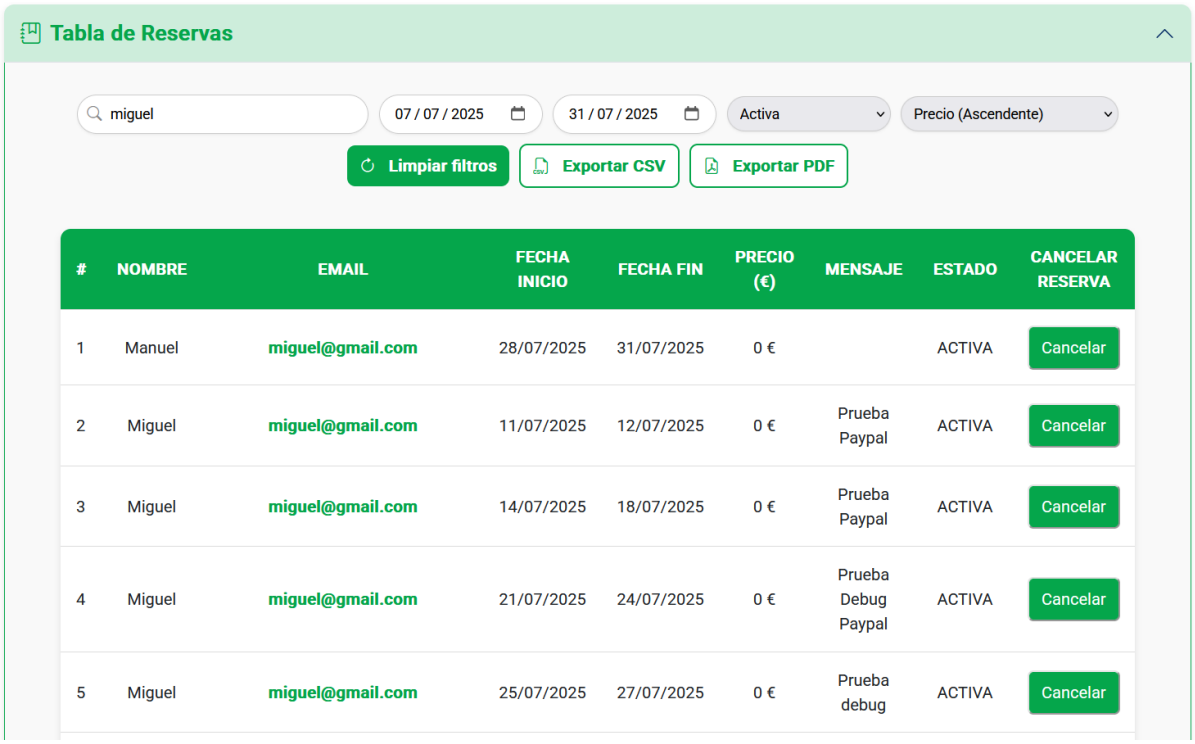

Figura 9.59: Filtrado de reservas en la Página de administrador

Reservas Filtradas

Listado de reservas de Casa Damajuana WebApp

Datos del 07/07/2025 - Obtenidas según los filtros aplicados en el Panel de Administrador

#	ID interno	Nombre	Email	Fecha Inicio	Fecha Fin	Precio (€)	Mensaje	Estado
1	8	Manuel	miguel@gmail.com	28/07/2025	31/07/2025	0		ACTIVA
2	16	Miguel	miguel@gmail.com	11/07/2025	12/07/2025	0	Prueba Paypal	ACTIVA
3	17	Miguel	miguel@gmail.com	14/07/2025	18/07/2025	0	Prueba Paypal	ACTIVA
4	18	Miguel	miguel@gmail.com	21/07/2025	24/07/2025	0	Prueba Debug Paypal	ACTIVA
5	19	Miguel	miguel@gmail.com	25/07/2025	27/07/2025	0	Prueba debug	ACTIVA
6	113	Miguel	aliciaestebannavajo@gmail.com	19/07/2025	20/07/2025	1380	Reserva para mi hermana	ACTIVA

Figura 9.60: *PDF* exportado de reservas en la Página de administrador

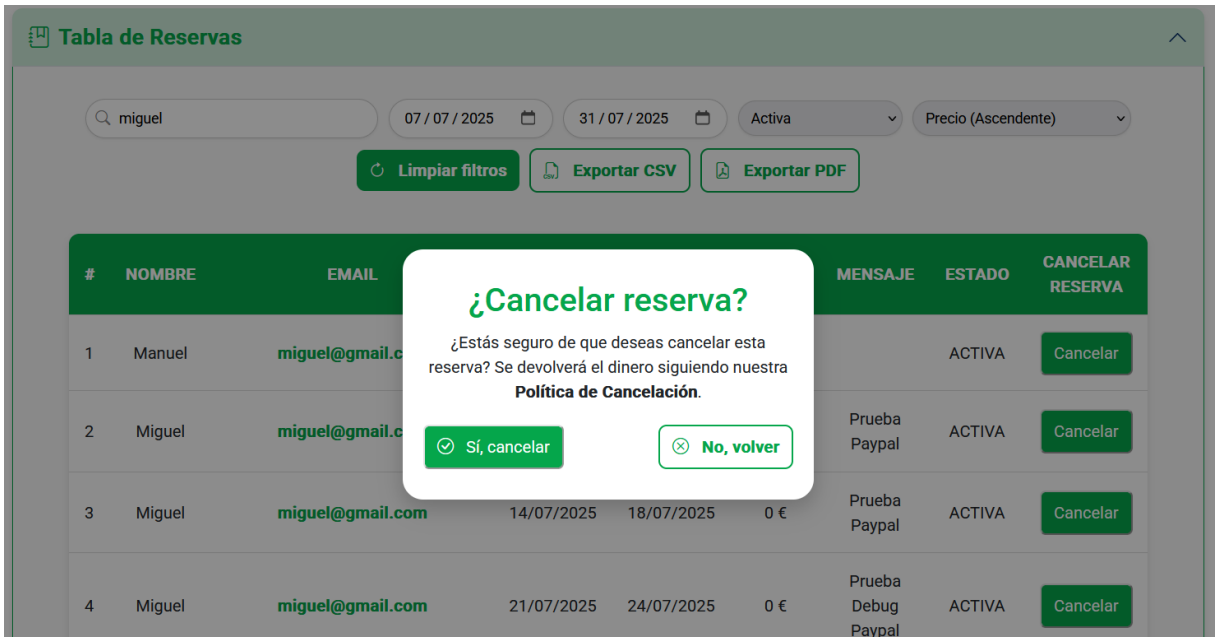

Figura 9.61: Cancelación de reservas en la Página de administrador

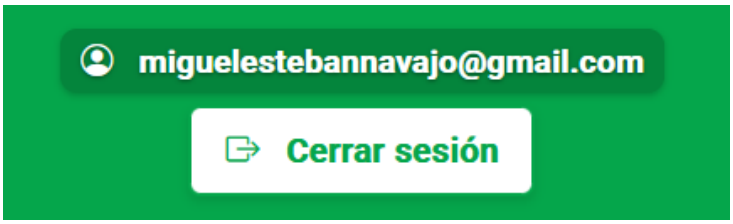

Figura 9.62: Cerrar sesión como administrador

Capítulo 10

Conclusiones y mejoras

La elaboración de este proyecto me ha hecho reflexionar acerca de diversas cuestiones ligadas al trabajo realizado, derivando en nuevas ideas y mejoras que podrían ser agregadas en un futuro.

Por lo tanto, en este último capítulo quiero exponer las conclusiones a las que he llegado durante y tras la realización de la aplicación *web*, así como las propuestas de mejora que planteo, demostrando que no se trata de un proyecto que tenga fecha de caducidad, sino de una oportunidad de mantener y mejorar este trabajo aplicando las nuevas ideas surgidas.

10.1. Conclusiones finales

En el momento en el que me decanté por aventurarme en la realización de una aplicación *web* utilizando tecnologías desconocidas y novedosas, supe que sería un gran desafío. Sin embargo, estaba convencido de que con esfuerzo, lograría llevar a buen puerto este proyecto, obteniendo un resultado funcionalmente correcto y satisfactorio en todos los sentidos.

La curva de aprendizaje de los *frameworks* empleados es elevada al inicio, pero poco a poco pude ir adquiriendo más destreza, potenciando mis habilidades y adquiriendo otras nuevas. Para ello, he tenido que seguir un proceso de formación continua para conocer estas herramientas y exprimirlas para obtener buenos resultados.

Desde el punto de vista técnico, he logrado implementar un sistema funcional de reservas para el alojamiento, adaptado a las necesidades del negocio y enfocado en la facilidad de uso tanto para el usuario final como para los administradores. A lo largo del proyecto se ha trabajado en una solución robusta, preocupándose de tener en cuenta todas las capas (*frontend, backend* y base de datos), aprovechando tecnologías actuales como *Angular* o *Spring Boot*, y la integración con servicios externos como *PayPal* para pagos o *Google* para *APIs* que agreguen valor.

Resumidamente, se ha logrado:

- Diseñar e implementar una arquitectura y un entorno tecnológico moderno e interesante.

- Desarrollar un sistema de autenticación seguro con credenciales y autorización mediante roles.
- Implementar una gestión de reservas con almacenamiento seguro de datos, gestionando los pagos de forma robusta.
- Integrar funcionalidades externas relevantes (correo, pagos, *reCAPTCHA*) que aportan valor añadido y simulan escenarios reales.
- Aplicar principios de usabilidad y accesibilidad en el diseño de interfaz, orientados a una navegación intuitiva.

El hecho de reflejar los conocimientos que he ido adquiriendo durante estos años en la universidad, complementándolo con el aprendizaje de nuevas tecnologías, refleja a la perfección la realidad del mundo de la informática, ya que se trata de un entorno cambiante, en el que no puedes permanecer estático. Esa ha sido la clave del éxito para mí, pues he contado con esa chispa de ilusión para investigar, entender e ir encontrando soluciones óptimas y adaptadas a mis necesidades.

También creo firmemente que al ligar este proyecto con un tema presente en mi día a día, como es el de un negocio familiar, me ha permitido mantener una motivación y disciplina mayor. Puede que hubiera perdido la ilusión en otros tópicos, pero el vínculo con lo que es tuyo me ha permitido seguir luchando en los momentos menos agradables.

Considero que el resultado final se alinea con los objetivos planteados al principio, demostrando que una planificación adecuada, junto con el deseo de mejora constante y la adaptación frente a los cambios pueden resultar en proyectos exitosos. Otra de las claves ha sido, precisamente, esa flexibilidad y adaptación continua a las situaciones, potenciada en parte por haber seguido una filosofía ágil, como he hecho en tantas asignaturas de la carrera.

En definitiva, tras meses de planificación, investigación, formación y trabajo, se ha logrado obtener una versión bastante robusta de una aplicación *web* que, no solo cumple con los objetivos planteados inicialmente, sino que demuestra el crecimiento personal y académico que he vivido en los últimos meses.

Por tanto, creo firmemente que este proyecto me ha servido para crecer, mejorar y desarrollar nuevas habilidades. Además, considero que aplicando las mejoras y llevando a cabo un mantenimiento oportuno, es una aplicación que puede aportar gran valor y servir de referencia para otros establecimientos y emprendedores que quieran seguir manteniendo vivo el entorno rural.

10.2. Ampliaciones y mejoras

Si bien el sistema cumple con los objetivos funcionales definidos al inicio del proyecto, se han identificado diversas posibles mejoras y ampliaciones que podrían implementarse en futuras versiones. Este trabajo futuro permitiría no solo el correcto mantenimiento de

la aplicación, sino el desarrollo de nuevas funcionalidades que mejoren su experiencia de uso. Por tanto, se muestran algunas de estas propuestas:

- **Despliegue en entorno real:** actualmente, la aplicación se ejecuta en local tanto en el cliente como en el servidor. Una mejora natural sería el despliegue en un entorno productivo en la nube, mediante contenedores (*Docker*) o plataformas como *Vercel*, *Heroku* o servidores propios. El despliegue de la aplicación supondría un lanzamiento real, permitiendo que la casa rural ponga a prueba la efectividad de la aplicación.

- **Análisis *SEO*:** permitiría optimizar el sitio *web*, mejorando su posicionamiento en ciertos motores de búsqueda, permitiendo así una captación de clientes mayor.

- **Gestión de redes sociales:** a pesar de contar con redes sociales del establecimiento y referenciarlas en numerosos apartados de la aplicación, es importante mantener un uso estratégico de las mismas, con la intención de motivar a los clientes a compartir sus experiencias etiquetando a la cuenta de la casa, mostrar contenido promocional e incluso realizar campañas que permitan obtener una mayor cantidad de público interesado. Podría resultar interesante mostrar publicaciones de las cuentas de *RRSS* en tiempo real en la *web*, o conectar de alguna forma la plataforma con estas aplicaciones tan usadas en el día a día.

- **Desarrollo de una aplicación móvil:** a pesar de haber desarrollado una aplicación *web* adaptable, logrando una experiencia excelente en dispositivos de tamaño reducido, podría resultar clave el obtener una aplicación móvil nativa. Este ambicioso proyecto, permitiría la integración con otras aplicaciones (como redes sociales, aplicaciones de mapas, etc.) para ofrecer una experiencia inmersiva a los usuarios.

- **Persistencia de mensajes de contacto:** aunque el sistema ya permite enviar los mensajes al correo de la casa, sería útil almacenar un historial de mensajes en base de datos, permitiendo su consulta y seguimiento desde un panel de administración. De esta manera, podría mantenerse un seguimiento más sencillo y centralizado de las comunicaciones con los clientes.

- **Sistema de *logs* y auditoría:** se podría incorporar un mecanismo de registro de eventos relevantes (inicio de sesión, errores, modificaciones críticas, etc.), lo cual contribuiría al mantenimiento y la seguridad del sistema. Esta idea tendría sentido de cara a facilitar al administrador del sistema, no al de la casa rural, ciertos datos para mejorar sus labores de control y seguridad.

- **Mejoras en accesibilidad:** aunque se han seguido buenas prácticas de diseño, se podrían utilizar herramientas como *WAVE* [52] o validadores *WCAG* [22] para asegurar un mayor cumplimiento de estándares de accesibilidad. *WAVE* es una herramienta gratuita de evaluación de la accesibilidad web que ayuda a identificar problemas de accesibilidad en sitios web, permitiendo que sean más fáciles de usar para personas con discapacidades. Por otro lado, la aplicación de *Web Content*

Accessibility Guidelines (*WCAG*), permitiría lograr un contenido más accesible en la página, por ejemplo para usuarios daltónicos. Todo ello mejoraría con creces la accesibilidad de la aplicación *web*, potenciando y complementando los propios análisis mencionados en la sección 6.8.

- **Panel de administración más completo:** mejorar la interfaz para la administración, con la intención de permitir la modificación de políticas de cancelación, tarifas u otros parámetros del sistema. Además de las interesantes herramientas que se le facilitan a los administradores, siempre es conveniente seguir estudiando sus necesidades, con la intención de facilitar sus labores de gestión del establecimiento.

- **Internacionalización (`i18n`):** partiendo de la base expuesta en el proyecto, realizar una internacionalización completa de la aplicación, para hacer la plataforma más accesible a usuarios internacionales. Se podrían introducir idiomas como el inglés, que permitan a los turistas contar con un entorno adaptado a sus necesidades.

- **Análisis estadístico de uso:** implementar una capa de analítica que permita extraer información sobre el comportamiento de los usuarios (fechas más reservadas, filtros más usados, tipo de habitaciones preferidas, etc.), siempre respetando la privacidad y las normas al respecto. De esta manera, las tareas de mantenimiento serían más sencillas, y podríamos identificar nuevas necesidades para materializarlas en forma de mejoras.

- **Notificaciones automáticas:** podría resultar interesante integrar sistemas de notificación, como correos automáticos tras reservas o cancelaciones, o incluso avisos por *WhatsApp* mediante servicios externos. De esta manera, se podría recordar a los clientes la existencia de su reserva, mostrarles información relevante, etc.

- **Inclusión de pruebas unitarias:** a pesar de haber realizado una extensa batería de pruebas, podría resultar interesante la inclusión de pruebas unitarias, tanto en el proyecto *Angular*, exprimiendo los archivos `.spec` gracias a tecnologías como *Jasmine* y *Karma*, como en el *backend* de *Spring Boot*, elaborando pruebas con `JUnit` y simplificando conexiones con `Mockito`, tal y como se explica en la sección 7.3.

Estas ampliaciones no solo enriquecerían la experiencia del usuario final, sino que también dotarían al sistema de una mayor escalabilidad, robustez y adaptabilidad a entornos reales de producción.

Parte IV

Apéndices

Apéndice A

Contenido del repositorio

En este capítulo se explicará el árbol de directorios presente en el repositorio habilitado para la entrega del código. Es importante atender a las instrucciones facilitadas en el capítulo 8, para realizar una correcta instalación de aplicaciones y herramientas. Además, en dicho capítulo se exponen ciertos comandos y acciones necesarias para desplegar correctamente cada subproyecto.

En la carpeta de entregada ubicada en el repositorio facilitado, llamada en este caso `Entrega-Proyecto-MiguelEstebanNavajo`, encontraremos los siguientes subdirectorios y archivos:

- Una carpeta llamada `proyecto-frontend`, que contendrá el proyecto de *Angular* elaborado, a falta de los certificados y su configuración, así como de los archivos de entorno. Ambos se obtendrán siguiendo las configuraciones e instrucciones descritas en el capítulo 8.

- Un archivo llamado `plantilla-environment.ts`, cuya misión es ilustrar cuál debe ser la disposición de las variables de entorno en los archivos de *environment* generados manualmente. De hecho, en el capítulo 8 se hace referencia a las variables de esta plantilla para especificar cómo denominarlas y rellenarlas una vez generados los archivos de entorno.

- Una carpeta llamada `proyecto-backend`, que contendrá el proyecto *Spring Boot*. En este caso, el archivo `application.properties` vendrá incluido, pero algunas de las variables estarán vacías. Dichas variables deben rellenarse obteniendo ciertas claves y datos, explicadas en el capítulo 8.

- El archivo `casadamajuana.sql`, que contiene las sentencias necesarias para desplegar la base de datos. Incluye la generación de las tablas, *triggers*, índices y algunas tuplas de prueba. En el capítulo 8 se incluyen las claves de acceso a las cuentas de administrador.

- El archivo `TFG_Informática___Aplicación_Web_Casa_Damajuana.pdf`, que constituye el documento elaborado con *LaTeX* exportado en formato *PDF*.

- El archivo `README.txt`, que resume el contenido de este capítulo, por si algún usuario no dispusiera de la documentación.

En definitiva, con la estructura del repositorio desglosada en este capítulo, complementada con las explicaciones del capítulo 8, se espera que cualquier individuo sea capaz de desplegar el proyecto elaborado, haciendo uso de los servicios externos integrados.

De todos modos, en caso de producirse cualquier error o inconveniente, quedo a disposición de los interesados de cara a facilitar el proceso de instalación.

Apéndice B

Anexos

En este breve capítulo adicional, se expondrán de forma apaisada ciertos diagramas presentes a lo largo de la documentación. De esta forma, se facilitará su consulta y visualización en papel.

Nótese que dichos diagramas se encuentran ya presentes en sus respectivas secciones, pero de esta manera, se podría hacer uso de su versión ampliada si fuera necesario.

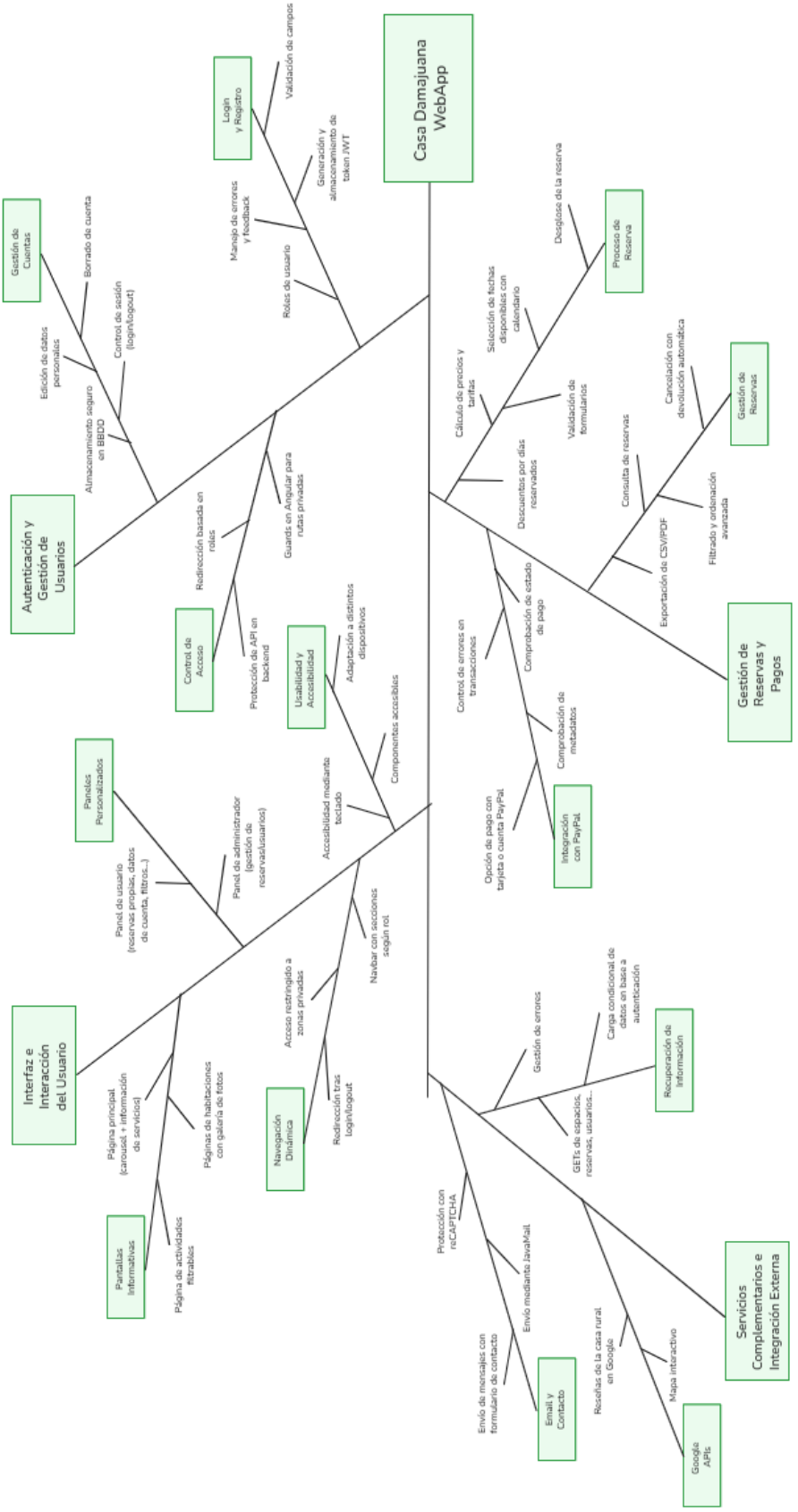

Figura B.1: Diagrama de Características del proyecto (versión ampliada)

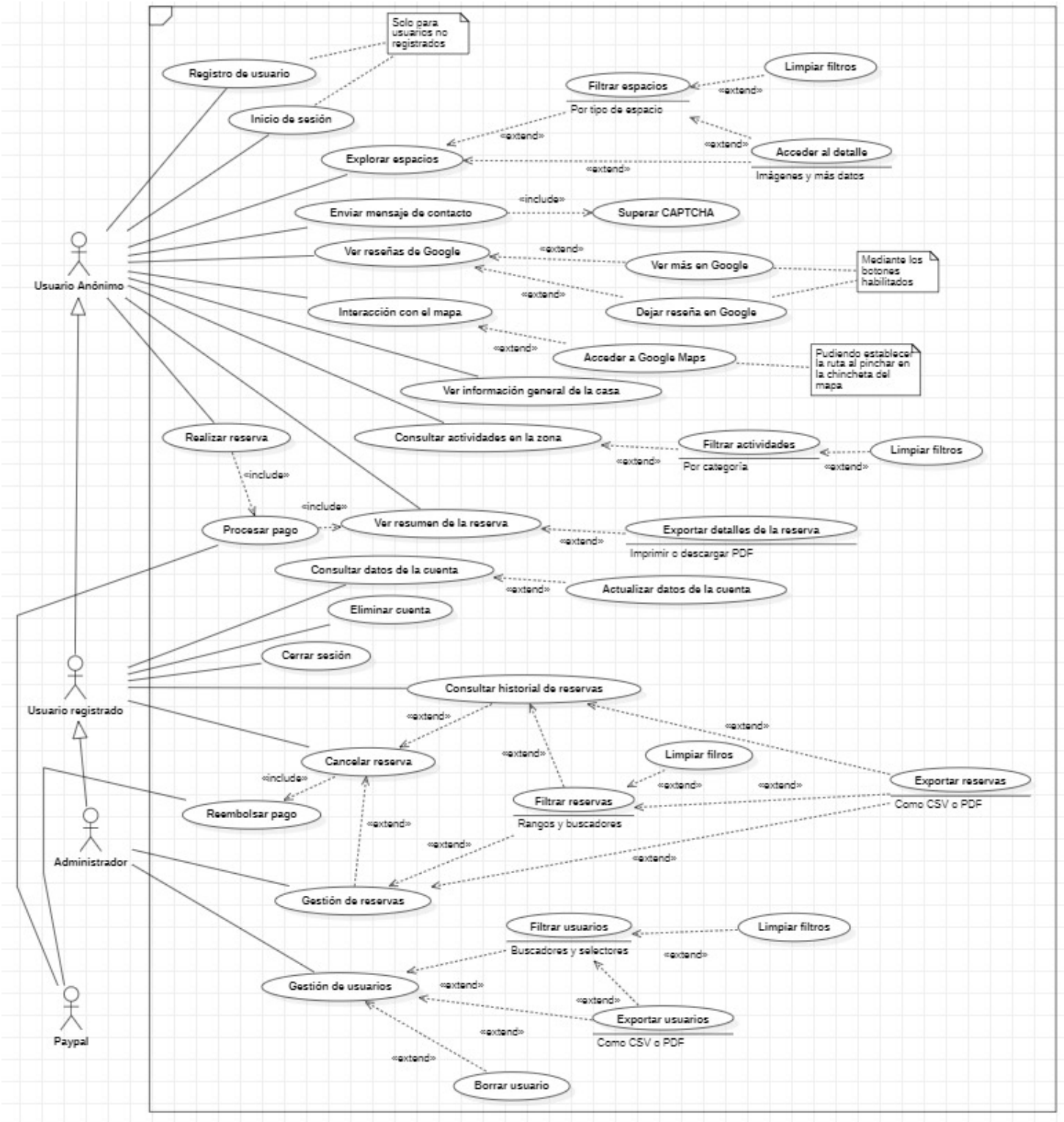

Figura B.2: Diagrama de Casos de Uso del sistema (versión ampliada)

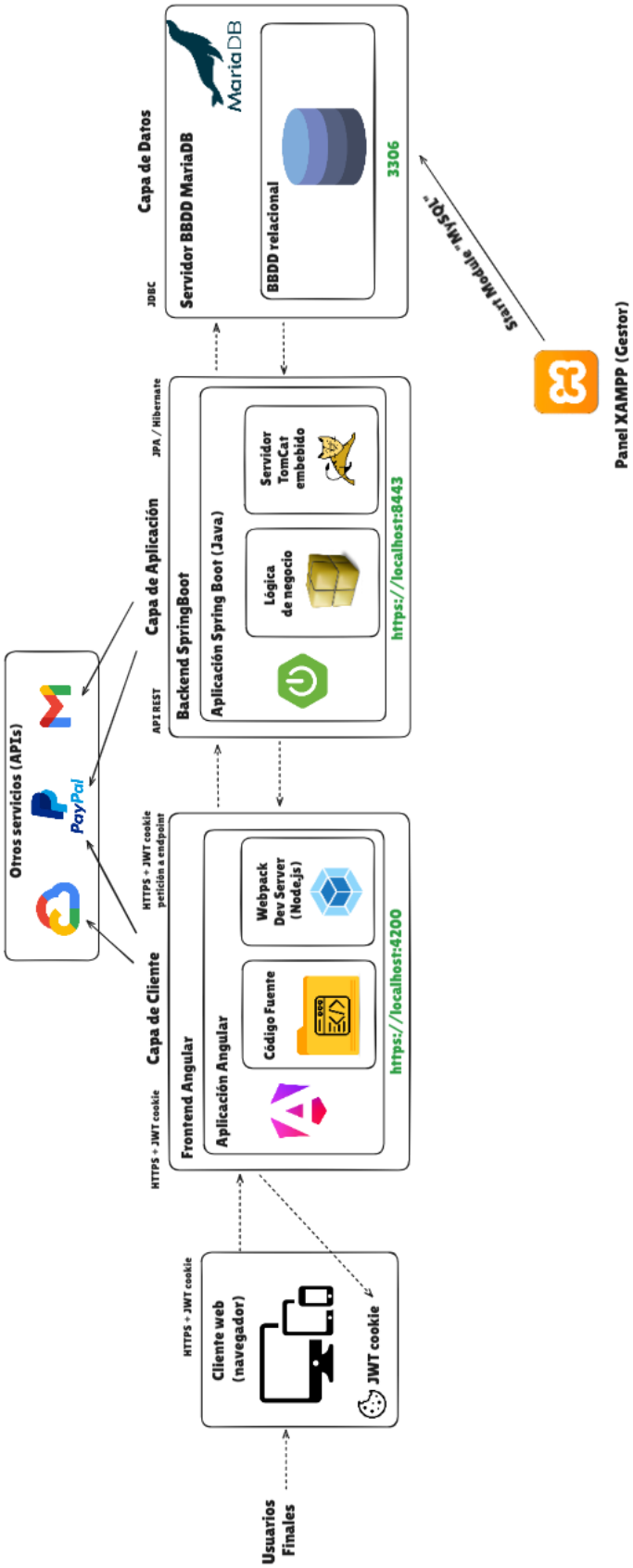

Figura B.3: Arquitectura Física del sistema (versión ampliada)

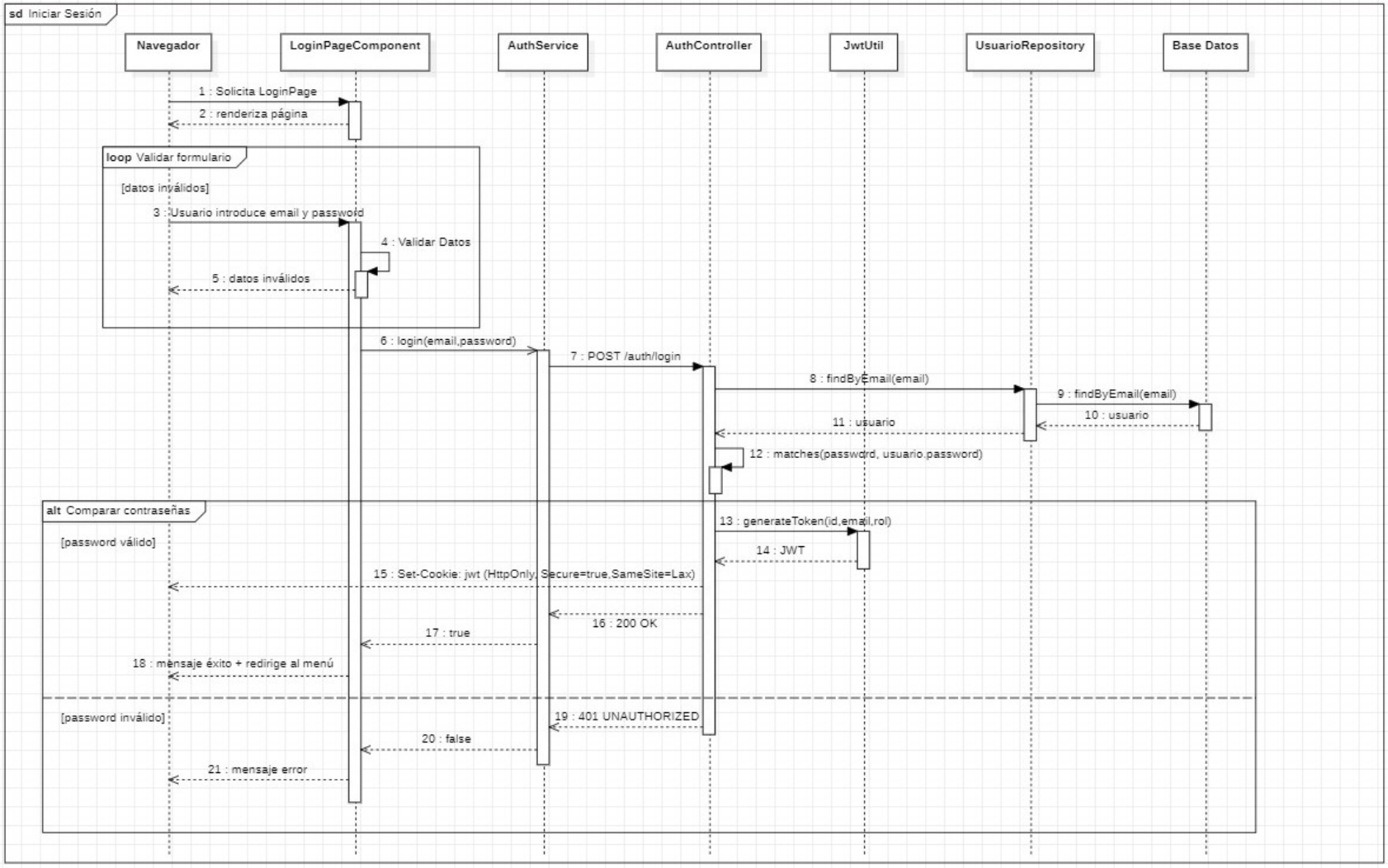

Figura B.4: Diagrama de Secuencia de inicio de sesión (versión ampliada)

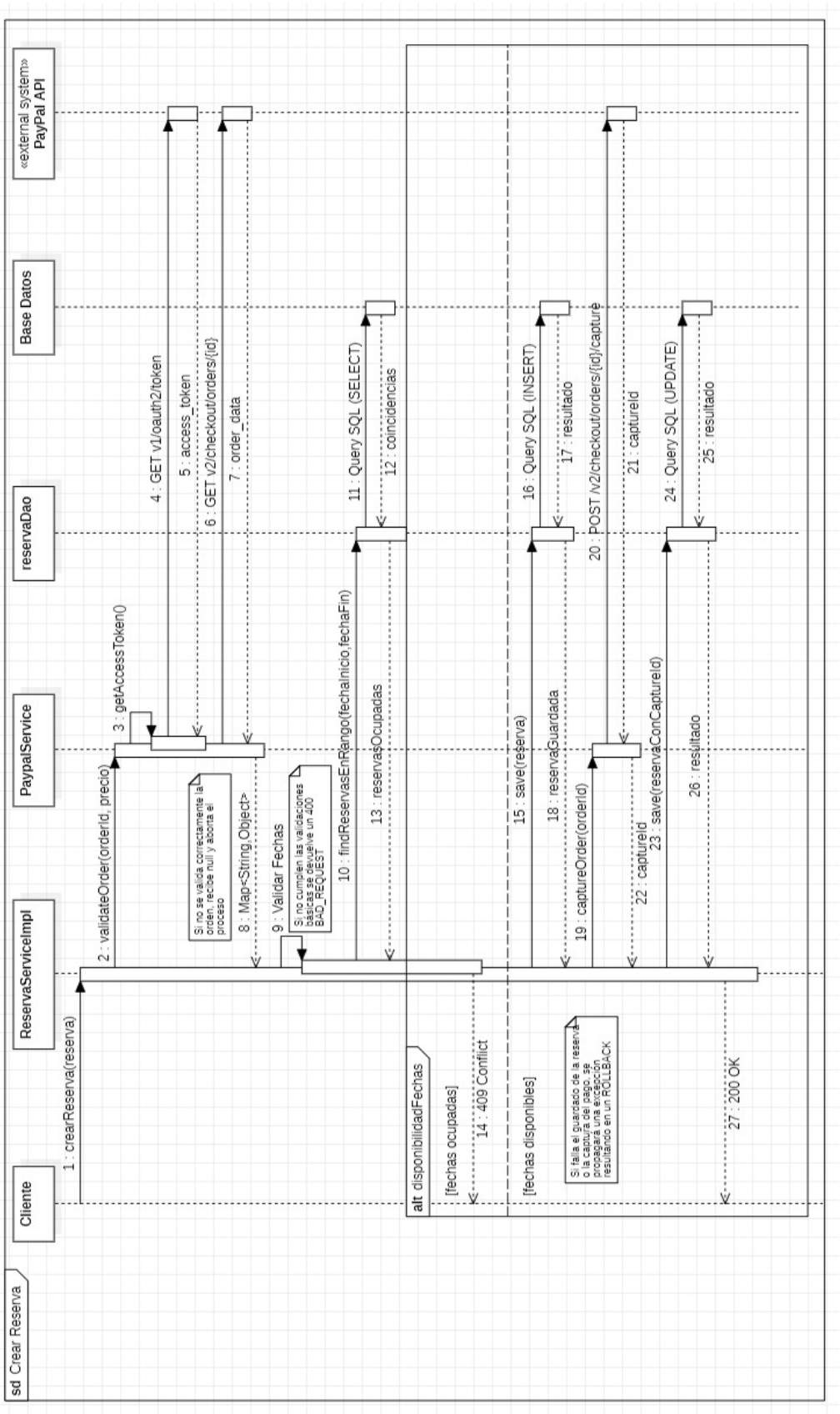

Figura B.5: Diagrama de Secuencia de creación de la reserva (versión ampliada)

Bibliografía

[1] *Documentación de Spring Boot*. VMWare Tanzu. 2025. URL: https://spring.io/projects/spring-boot.

[2] *Documentación oficial de la Instalación de Angular CLI*. Google. 2025. URL: https://angular.dev/installation.

[3] *Hoja de instalaciones recomendadas para trabajar con Angular*. Fernando Herrera. 2025. URL: https://gist.github.com/Klerith/4816679624c1cb528f8e05d902fd7cff.

[4] *Instalación de Chrome*. Google. 2025. URL: https://www.google.com/intl/es_es/chrome/.

[5] *Instalación de Git*. Git. 2025. URL: https://git-scm.com/.

[6] *Instalación de Node.js*. OpenJS Foundation. 2025. URL: https://nodejs.org/es/.

[7] *Instalación de OpenJDK*. Oracle. 2025. URL: https://jdk.java.net/archive/.

[8] *Instalación de Postman*. Postman. 2025. URL: https://www.postman.com/downloads/.

[9] *Instalación de SourceTree*. Atlassian. 2025. URL: https://www.sourcetreeapp.com/.

[10] *Instalación de STS*. VMware. 2025. URL: https://spring.io/tools.

[11] *Instalación de VSCode*. Microsoft. 2025. URL: https://code.visualstudio.com/.

[12] *Instalación del panel de XAMPP*. Apache Friends. 2024. URL: https://www.apachefriends.org/es/download.html.

[13] *Jasmine Documentation*. Jasmine. 2025. URL: https://jasmine.github.io/.

[14] *Karma Site*. Karma. 2025. URL: https://karma-runner.github.io/latest/index.html.

[15] *Lighthouse*. Google Ireland. 2025. URL: https://chromewebstore.google.com/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=es.

[16] *Mockito Framework Site*. Mockito. 2025. URL: https://site.mockito.org/.

[17] *Narrador de Windows*. Microsoft. 2025. URL: https://support.microsoft.com/es-es/windows/cap%C3%ADtulo-1-introducci%C3%B3n-al-narrador-7fe8fd72-541f-4536-7658-bfc37ddaf9c6.

[18] *Obtención de API keys de Google Cloud Console*. Google. 2025. URL: https://console.cloud.google.com/.

[19] *Obtención de API keys de reCAPTCHA*. Google. 2025. URL: https://www.google.com/recaptcha/admin/create.

[20] *Obtención de clave de aplicación para Gmail*. Gmail, Google. 2025. URL: https://myaccount.google.com/apppasswords.

[21] *Obtención de claves PayPal Developer*. PayPal. 2025. URL: https://developer.paypal.com/.

[22] *Pautas de Accesibilidad para el Contenido Web*. W3C. 2025. URL: https://www.w3.org/WAI/standards-guidelines/wcag/es.

[23] *Página oficial de Airbnb*. Airbnb. 2025. URL: https://www.airbnb.es/.

[24] *Página oficial de Angular*. Google. 2025. URL: https://angular.dev/.

[25] *Página oficial de Angular Material*. Google. 2025. URL: https://material.angular.dev/.

[26] *Página oficial de Booking*. Booking. 2025. URL: https://www.booking.com/index.es.html.

[27] *Página oficial de Bootstrap 5*. Bootstrap. 2025. URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

[28] *Página oficial de Canva*. Canva. 2025. URL: https://www.canva.com/es_es/.

[29] *Página oficial de Excalidraw*. Excalidraw. 2025. URL: https://excalidraw.com/.

[30] *Página oficial de GitHub*. GitHub. 2025. URL: https://github.com/.

[31] *Página oficial de Google Fonts*. Google. 2025. URL: https://fonts.google.com/.

[32] *Página oficial de LaTeX (software libre)*. The LaTeX Project. 2025. URL: https://www.latex-project.org/.

[33] *Página oficial de MariaDB*. MariaDB Foundation. 2025. URL: https://mariadb.org/.

[34] *Página oficial de Mozilla*. Mozilla Foundation. 2025. URL: https://www.mozilla.org/.

[35] *Página oficial de MySQL*. Oracle. 2025. URL: https://www.mysql.com/.

[36] *Página oficial de NG Bootstrap*. NG Bootstrap. 2025. URL: https://ng-bootstrap.github.io/.

[37] *Página oficial de npm*. npm. 2025. URL: https://www.npmjs.com/.

[38] *Página oficial de Outlook*. Microsoft. 2025. URL: https://outlook.live.com/.

[39] *Página oficial de Overleaf*. Overleaf. 2025. URL: https://es.overleaf.com/.

[40] *Página oficial de PhpMyAdmin*. PhpMyAdmin Contributors. 2025. URL: https://www.phpmyadmin.net/.

[41] *Página oficial de Postman*. Postman. 2025. URL: https://www.postman.com/.

[42] *Página oficial de Ruralidays*. Ruralidays. 2025. URL: https://www.ruralidays.com/.

[43] *Página oficial de StarUML*. MKLabs Co. 2025. URL: https://staruml.io/.

[44] *Página oficial de Trello*. Atlassian. 2025. URL: https://trello.com/es.

[45] *Página oficial de TypeScript*. Microsoft. 2025. URL: https://www.typescriptlang.org/.

[46] *Página oficial de XAMPP*. Apache Friends. 2024. URL: https://www.apachefriends.org/es/index.html.

[47] *Sitio oficial de Draw.io*. JGraph Ltd y draw.io AG. 2025. URL: https://www.drawio.com/.

[48] *Sitio oficial de IFPUG, mantenedora del Function Point Counting Practices Manual*. International Function Point Users Group (IFPUG). 2025. URL: http://www.ifpug.org.

[49] *Stack Overflow Developer Survey*. Stack Overflow. 2023. URL: https://survey.stackoverflow.co/2023/#technology.

[50] *The programmer-friendly testing framework for Java*. The JUnit Team. 2025. URL: https://junit.org/.

[51] *Una metodología basada en prácticas ágiles para la realización de Trabajos Fin de Grado*. Depto. de Informática, Universidad de Valladolid. 2023. URL: https://uvadoc.uva.es/bitstream/handle/10324/60235/JENUI_2023_paper_44.pdf?sequence=2%20METODOLOGIA%20ASAP.

[52] *WAVE Web Accessibility Evaluation Tools*. WAVE. 2025. URL: https://wave.webaim.org/.

[53] *Web Accessibility Initiative*. World Wide Web Consortium (W3C). 2025. URL: https://www.w3.org/WAI/.