UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria **Informàtica**

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño de un portal web para el intercambio de viviendas

Trabajo de Fin de Grado
Grado en Ingeniería Informática

Autor: Víctor Díaz Marco
Tutor: Sergio Sáez Barona
Curso: 2015-2016

Resumen

El presente proyecto consiste en el desarrollo y despliegue de un sitio web que permite el intercambio y alquiler temporal de viviendas, orientado tanto a particulares como autónomos y empresas. La aplicación posibilita la publicación de alojamientos, a los que es posible añadir ofertas para fechas específicas. Además, se contempla la posibilidad de enviar y gestionar solicitudes para determinadas ofertas, así como la valoración por parte de los huéspedes de su experiencia con un alojamiento. También se ofrecen sistemas de búsqueda acotada, de mensajería privada, de notificaciones y la posibilidad de gestionar los datos de las cuentas. La interfaz está disponible en valenciano, castellano e inglés.

Partiendo de las fases de análisis y diseño, y siguiendo una estrategia *API-first*, se ha desarrollado una *RESTful* API utilizando el lenguaje de programación PHP, el *micro-framework* Lumen y el sistema de gestión de bases de datos MariaDB. Por otra parte, se ha hecho uso del lenguaje de marcado HTML, el lenguaje de hojas de estilo CSS y el lenguaje de programación JavaScript, junto con el *framework* AngularJS, para el desarrollo del cliente web.

Palabras clave: web, vivienda, intercambio, alquiler, PHP, Lumen, JavaScript, AngularJS.

Resum

El present projecte consistix en el desenvolupament i desplegament d'un lloc web que permet l'intercanvi i lloguer temporal d'habitatges, orientat tant a particulars com a autònoms i empreses. L'aplicació resultant possibilita la publicació d'allotjaments, als quals és possible afegir ofertes per a dates específiques. A més, es contempla la possibilitat d'enviar i gestionar sol·licituds per a determinades ofertes, així com la valoració per part dels hostes de la seua experiència amb un allotjament. També s'oferixen sistemes de cerca fitada, de missatgeria privada, de notificacions i la possibilitat de gestionar les dades dels comptes. La interfície està disponible en valencià, castellà i anglés.

Partint de les fases d'anàlisi i disseny, i seguint una estratègia *API-first*, s'ha desenvolupat una *RESTful* API utilitzant el llenguatge de programació PHP, el *micro-framework* Lumen i el sistema de gestió de bases de dades MariaDB. D'altra banda, s'ha fet ús del llenguatge de marcat HTML, el llenguatge de fulles d'estil CSS i el llenguatge de programació JavaScript, juntament amb el *framework* AngularJS, per al desenvolupament del client web.

Paraules clau: web, habitatge, intercanvi, lloguer, PHP, Lumen, JavaScript, AngularJS.

Abstract

This project involves the development and deployment of a web site that allows temporary exchange and rental of accommodations; aimed at individuals, freelancers and companies. The resulting application enables the publication of accommodations, to which may be added offers for specific dates. In addition, it is possible to send and manage requests for certain offers and guests may also rate their experience with accommodations. Furthermore, there is a filterable search system, private messaging, notifications and the ability to manage accounts data. The interface is available in Valencian, Spanish and English.

Starting from the analysis and design phases, and following an API-first strategy, it has been developed a RESTful API using the PHP programming language, the micro-framework Lumen and the database management system MariaDB. Alternatively, the markup language HTML, the stylesheet language CSS and the programming language JavaScript, along with the AngularJS framework, have been used to develop the web client.

Keywords: web, house, exchange, rental, PHP, Lumen, JavaScript, AngularJS.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

El contexto actual de larga crisis económica ha impulsado el crecimiento de la economía colaborativa, la cual permite un mejor aprovechamiento de los recursos y, por consiguiente, un ahorro económico. Internet ha posibilitado este crecimiento gracias a la popularización de los dispositivos móviles con sistemas operativos avanzados, las redes sociales y la World Wide Web en general.[24] Es precisamente debido a la crisis económica que la cantidad de ciudadanos españoles que no pueden permitirse sus vacaciones de verano asciende al 26 por ciento en 2016.[37]

La economía colaborativa, también denominada consumo colaborativo, consiste en compartir recursos ya existentes (normalmente en desuso) para ahorrar u obtener ingresos extra. En esta línea existen servicios como BlaBlaCar, que permite compartir vehículo para realizar viajes largos, o Wallapop, que faculta la compraventa entre particulares. Mientras que el primero posibilita aprovechar un recurso ya existente para disminuir gastos, el segundo se basa en crear un modelo de negocio partiendo de un recurso que no se utiliza.

La World Wide Web es un sistema que permite distribuir tanto documentos de hipertexto como completas aplicaciones web multiplataforma. En este sentido complementa con el consumo colaborativo al tener una baja exigencia de recursos, tanto económicos como tecnológicos y humanos, y abre la puerta a que este tipo de proyectos empiecen por ella.

El presente proyecto pretende aprovechar este contexto y desarrollar un portal web que, en base a la economía colaborativa, permita a cualquier persona irse de vacaciones sin requerir un gran desembolso económico; lo que además implica un aprovechamiento de recursos que repercute positivamente en el conjunto de la sociedad.

1.1. Objetivos

El objetivo principal del proyecto es la puesta en marcha de una aplicación web que cumpla con los objetivos secundarios que se enumeran a continuación:

- Demostrar la capacidad del autor para desarrollar un proyecto completo de forma parcialmente autónoma, partiendo de los conocimientos adquiridos en la titulación y de la capacidad de aprendizaje por cuenta propia.

- Desarrollar una aplicación web que permita el intercambio y alquiler de viviendas entre usuarios.

- Ofrecer la posibilidad de gestionar cuentas, alojamientos, imágenes, ofertas, solicitudes, mensajes privados y notificaciones. Entiéndase como gestionar la capacidad para obtener, crear, modificar, eliminar y/o asociar elementos, según corresponda. Véase el ámbito y los casos de uso.

- Aplicar una arquitectura que tenga en cuenta la internacionalización (*i18n*) y localización (*l10n*), soportando por defecto las dos lenguas oficiales de la Comunidad Valenciana (valenciano y castellano), además del inglés.

- Seguir unas normas de diseño coherentes, en este caso Material Design, que den como resultado una interfaz usable e intuitiva.

- Desplegar la aplicación web en un servidor público accesible desde Internet.
- Seguir unas buenas prácticas que garanticen la seguridad de los usuarios y su información personal, tanto por parte de la aplicación como del servidor en el que se despliegue. Cumplir, por tanto, con la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal y el Real Decreto que la complementa.

Establecer los objetivos anteriores permite acotar el alcance del proyecto y definir los hitos, lo que forma parte indispensable de la planificación.

1.2. Estructura de la memoria

La memoria comienza con el análisis del contexto del proyecto y las tecnologías relacionadas con su desarrollo. Al tratarse de una aplicación distribuida que sigue el modelo cliente-servidor, implica el uso de dos tipos de tecnologías diferentes, por lo que ambas son tratadas individualmente. Por otra parte, se realiza un breve análisis de las aplicaciones similares ya existentes en el mercado.

A continuación se realiza la especificación formal de requisitos de acuerdo con el estándar IEEE 29148-2011, recogiendo de forma detallada los requisitos y funcionalidades de la aplicación web. Esta especificación se realiza en base a los objetivos y permite acotar el alcance del proyecto, aspecto fundamental en la gestión de proyectos.

Los capítulos que hay a continuación coinciden con las diferentes fases del desarrollo siguiendo los principios de la ingeniería de software (excepto la fase de despliegue). En primer lugar se aborda la fase de análisis, definiendo los requisitos del producto y obteniendo como resultado los diagramas de clases y de casos de uso. En segundo lugar se emprende la fase de diseño, en la cual se plantea la arquitectura completa de la aplicación desglosándola por las capas del patrón modelo-vista-controlador.

A continuación se emprende la fase de implementación, en la que se convierten todas las especificaciones anteriores en un producto tangible, pero no final. Para ello se analizan las tecnologías utilizadas, el entorno de trabajo y la estructura de directorios de ambos proyectos (la interfaz de usuario y la *RESTful* API).

Seguidamente se plantea la fase de preparación de la infraestructura y despliegue, relacionada con la administración de sistemas. En ella se analizan los aspectos vinculados con la preparación del servidor, el software escogido, la configuración y la puesta a punto.

Para terminar con el proceso de ingeniería de software se acomete la fase de pruebas, que consiste en verificar empíricamente que el software es capaz de efectuar todas las tareas contempladas en la especificación tal y como se precisan y sin ningún tipo de error.

Por último se analizan las conclusiones del autor derivadas de la ejecución del proyecto.

La estructuración de la memoria sigue un orden tanto cronológico como causal, es decir, cada capítulo depende y parte del anterior. Por lo tanto, permite realizar un seguimiento cómodo de la evolución del proyecto.

2. Contexto

Los sitios web dinámicos tradicionales que respetan el patrón modelo-vista-controlador están formados por una serie de vistas definidas mediante el lenguaje de marcado HTML (HyperText Markup Language), estilos asociados a las mismas definidos por medio de hojas de estilo CSS (Cascading Style Sheets) y diversos controladores, clases, *scripts* u otro tipo de código escrito en un lenguaje de programación del lado del servidor, que normalmente suele ser interpretado. Ejemplos de este último son PHP, Python, Ruby y Java. Además, es común ver el uso de JavaScript para realizar tareas secundarias que den sensación de dinamismo al uso de la web aunque esta no esté interactuando con el servidor.

Este tipo de sitios web se construyen alrededor del paradigma petición/respuesta de HTTP: el usuario carga una página y no ocurre nada hasta que este accede a la siguiente. Sin embargo, a partir del año 2005, AJAX empezó a modificar este planteamiento añadiendo la posibilidad de, una vez cargada la página, realizar peticiones para obtener información adicional del servidor, ya sea de forma periódica o debido a la interacción del usuario. Estas peticiones se llaman peticiones XHR debido al nombre de la API del navegador que permite realizarlas: *XMLHttpRequest*.

Esta posibilidad abrió la puerta a la introducción de un nuevo tipo de sitio web: las aplicaciones web interactivas. En ellas existe una comunicación constante entre cliente y servidor: el usuario interactúa con la interfaz de la aplicación y su contenido va cambiando sin necesidad de cargar de nuevo la página completa. Para hacer posible este nuevo modelo es necesaria la existencia de una interfaz de programación de aplicaciones (API) que sirva de capa entre la aplicación web en sí (*front-end*), que ejecuta el usuario en su navegador, y la capa de persistencia, que se encuentra en el servidor en cuestión. Esta interfaz, también denominada *back-end* en el ámbito del desarrollo web, se encuentra igualmente en el servidor.

Las *RESTful* API son un tipo de API que siguen los principios REST, los cuales se basan en los siguientes planteamientos:

- **Comunicaciones sin estado** mediante HTTP. Todos los mensajes son autocontenidos y no tienen contexto. Esto facilita la implementación y la escalabilidad.

- Un **conjunto de operaciones**, verbos o métodos que se aplican a recursos y que permiten realizar operaciones CRUD (crear, leer, actualizar y borrar) sobre los mismos. Los métodos HTTP equivalentes a las operaciones CRUD son *POST*, *GET*, *PATCH* y *DELETE*, respectivamente. Sin embargo, el estándar HTTP define muchos más.

- Un **conjunto de recursos**, cada uno de ellos identificado por una URI. Puede haber recursos dependientes de otros, es decir, puede existir jerarquía.

- Uno o diversos **formatos de texto** para el intercambio de la información que viaja en el cuerpo de las peticiones y respuestas. JSON y XML son los más utilizados, siendo preferible el primero por cuestiones de practicidad y simplicidad.[23]

En [Fielding 2000] puede encontrarse una explicación detallada de la arquitectura REST, mientras que en [Sahni 2015] es posible hallar una serie de buenas prácticas que ayudan a comprenderla en profundidad.

Recapitulando, el enfoque de las aplicaciones web interactivas permite añadir una capa de abstracción más al planteamiento de la World Wide Web, separando la aplicación en dos partes: el *front-end* y el *back-end*. Cabe destacar que también existen API que ofrecen servicios como la compresión de imágenes o realización de cálculos complejos, en cuyo caso no sería apropiado indicar que el *back-end* trabaja con la capa de persistencia.

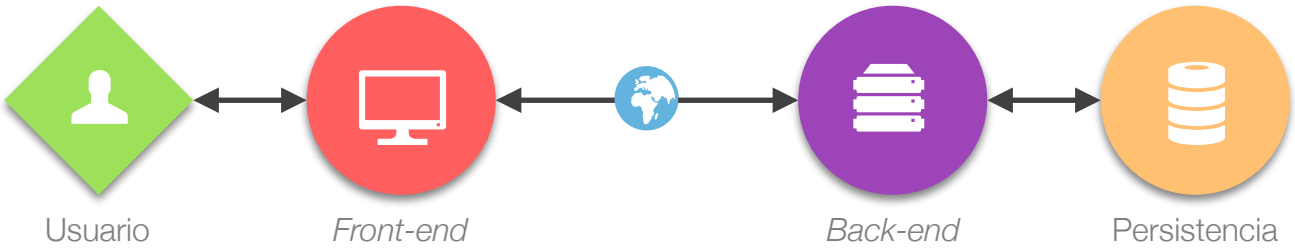

Figura 1. Flujo de interacción en una aplicación web.

Desde esta perspectiva es posible servir la interfaz de la aplicación[n1] como un conjunto de recursos estáticos (donde entran en juego las redes de distribución de contenidos) y ofrecer acceso a la lógica a través de una API. Esta aproximación permite liberar al servidor de una parte notable de su carga a la vez que se simplifica el desarrollo. Además da cabida a nuevos planteamientos como el denominado *API-first*, que consiste en iniciar las fases de diseño e implementación centrándose en la API. Podría verse como la división de un problema complejo en dos subproblemas más sencillos.

Por otra parte, JavaScript cobra relevancia y pasa de ser un lenguaje de uso anecdótico a ser el lenguaje de programación principal de la parte del cliente. Al tratarse de un lenguaje interpretado de alto nivel proporciona muchas facilidades para el desarrollo. Sin embargo, su naturaleza asíncrona puede plantear problemas de aprendizaje en las primeras etapas.

[n1] Téngase en cuenta que es muy conveniente replicar la lógica de validación en la aplicación cliente, lo que permite tener una interfaz que ofrezca una mayor realimentación además de evitar que el servidor realice comprobaciones innecesarias.

2.1. Programación del lado del servidor

Actualmente la programación del lado del servidor suele realizarse mediante lenguajes interpretados de alto nivel. En la siguiente tabla puede apreciarse la evolución de los lenguajes más utilizados en el contexto del desarrollo web entre los años 2010 y 2016.

	2010	2011	2012	2013	2014	2015	2016
PHP	72,5%	75,3%	77,3%	78,7%	81,6%	82,0%	81,7%
ASP.NET	24,4%	23,4%	21,7%	20,2%	18,2%	17,1%	16,0%
Java	4,0%	3,8%	4,0%	4,1%	2,7%	2,8%	3,0%
ColdFusion		1,3%	1,2%	1,1%	0,8%	0,7%	0,7%
Ruby	0,5%	0,5%	0,6%	0,5%	0,4%	0,6%	0,6%
Perl		1,1%	1,0%	0,8%	0,6%	0,5%	0,5%
Python	0,3%	0,3%	0,2%	0,2%	0,2%	0,2%	0,2%
JavaScript			0,1%	0,1%	0,1%	0,1%	0,2%

Tabla 1. Uso de lenguajes del lado del servidor en sitios web.

Fuente: W3Techs. Tendencias anuales en el uso de lenguajes de programación del lado del servidor en sitios web.[36]

PHP, utilizado por más del 80% de sitios web, fue diseñado específicamente para el desarrollo web y originalmente fue concebido como un lenguaje de *scripting*. Sin embargo, debido a su evolución y la incorporación de características como el paradigma de la programación orientada a objetos o el tipado fuerte, ha pasado a ser un lenguaje de propósito general. Su intérprete, basado en el motor Zend, es multiplataforma y gratuito. Esto, junto con la facilidad de instalación, el soporte nativo para diferentes sistemas de gestión de bases de datos y la existencia de *frameworks* como Symphony y Laravel, ha permitido que su cuota de uso permanezca alta. Es compatible con los servidores web más comunes, ya que incluye una interfaz soportada por Apache HTTP Server y Microsoft IIS, entre otros. Sin embargo, existe una implementación alternativa de FastCGI, un protocolo que define una interfaz entre programas interactivos y servidores web, llamada PHP-FPM que permite soportar mayores cargas de trabajo además de ser compatible con cualquier servidor web que soporte dicho estándar, como el servidor web Nginx.

Por otra parte se encuentra ASP.NET, utilizado por un 16% de los sitios web, que no es un lenguaje de programación, sino un *framework* pensado para desarrollar sitios, aplicaciones y servicios web dinámicos. Es el sucesor de ASP y permite el uso de cualquier lenguaje soportado por el *framework .NET* de Microsoft. Al estar completamente integrado en el ecosistema de Microsoft, permite desarrollos rápidos y con poco código. Además, al no depender de un lenguaje específico ofrece versatilidad a los desarrolladores.

Por último, entre los lenguajes utilizados por más de un 1% de sitios web, se encuentra Java con un 3%. Mediante el uso de entornos de desarrollo, como Eclipse, es posible crear *servlets* y desplegarlos. Existen multitud de servidores web que permiten la ejecución de *servlets*, como Apache Tomcat o Jetty.

2.2. Programación del lado del cliente

En la actualidad más del 90% de sitios web utilizan JavaScript debido a la estandarización de facto de su uso entre los navegadores más utilizados. Tecnologías en desuso como Flash y Silverlight están siendo reemplazadas progresivamente por los estándares aconsejados por el World Wide Web Consortium. Téngase en cuenta que existen lenguajes de programación que compilan a JavaScript, como CoffeScript, TypeScript o Dart. Es complicado conocer qué porcentaje corresponde realmente a estos lenguajes.

	2015 (1 de agosto)	2016 (1 de agosto)
JavaScript	89,8%	93,6%
Flash	10,4%	8,1%
Silverlight	0,1%	0,1%
Ninguno	10,0%	6,3%

Tabla 2. Uso de lenguajes del lado del cliente en sitios web.

Fuente: W3Techs. Tendencias anuales en el uso de lenguajes de programación del lado del cliente en sitios web.[35]

2.3. Aplicaciones relacionadas

Antes de abordar la especificación de requisitos se pretende realizar un pequeño análisis del contexto existente en el ámbito de las aplicaciones web para intercambios o alquileres de viviendas. A continuación se muestran unos ejemplos representativos.

2.3.1. Airbnb

Airbnb es un catálogo donde se pueden publicar alojamientos para ser alquilados, a cambio de una tasa de gestión. Permite la búsqueda por ciudad, fecha y número de huéspedes. Ofrece información detallada sobre las características de los alojamientos y los servicios de los que disponen.

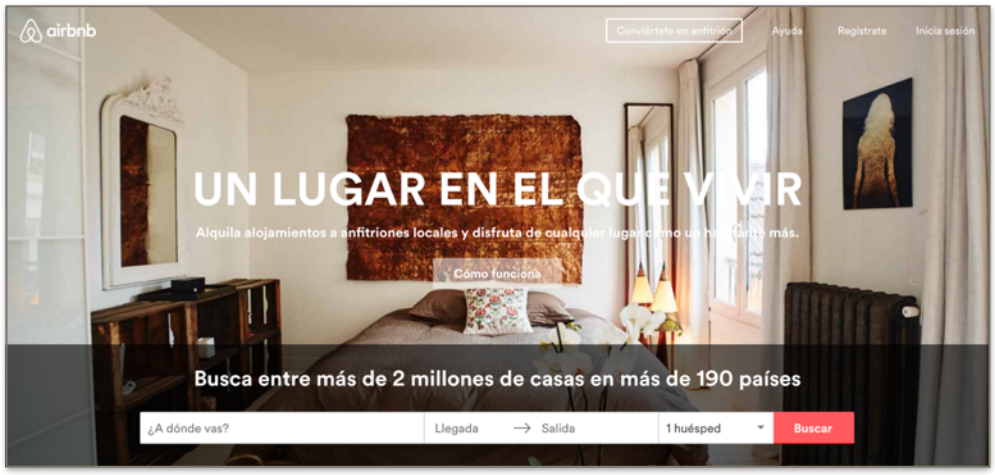

Figura 2. Captura de pantalla de la página principal de Airbnb.

2.3.2. HomeAway

Por otra parte se encuentra HomeAway, que ofrece prácticamente las mismas opciones que Airbnb. Sin embargo tiene una característica diferenciadora: muestra un calendario con las fechas en las que un alojamiento está ocupado o libre.

Figura 3. Captura de pantalla de la página principal de HomeAway.

2.3.3. Intercambio Casas

Por último, cabe destacar Intercambio Casas, que es de las pocas páginas web que no se centran en el alquiler, sino únicamente en el intercambio. Durante la búsqueda no ha sido posible encontrar ninguna página web que ofrezca tanto intercambios como alquileres.

En cuanto a las características destacables, ofrece un buscador que permite filtrar solamente por ubicación, sin poder concretar fechas ni número de huéspedes. Respecto a la vista de alojamientos, muestra prácticamente la misma información que Airbnb.

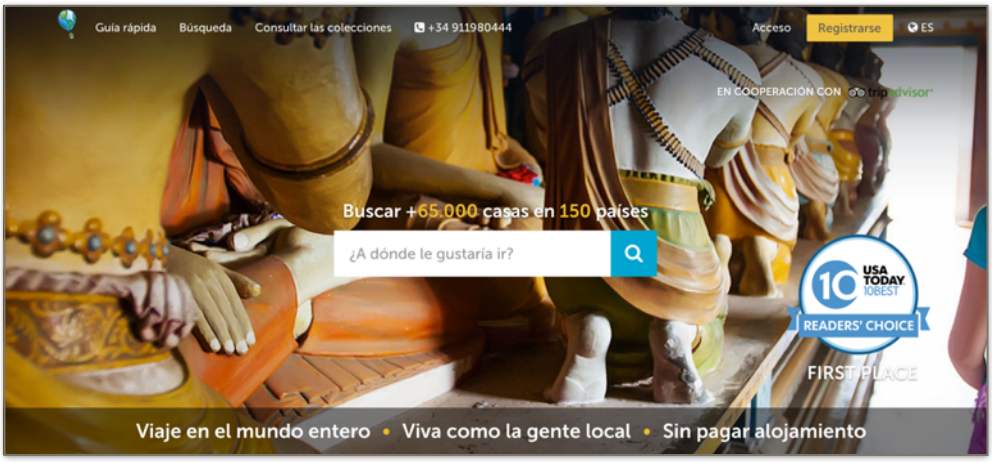

Figura 4. Captura de pantalla de la página principal de Intercambio casas.

3. Especificación de requisitos

La especificación de requisitos, basada en lo que establece el estándar IEEE 29148-2011, puede ser considerada como una submemoria dentro de este trabajo. Su objetivo es ofrecer una visión detallada del proyecto, describiéndolo de forma general, enumerando sus funcionalidades y definiendo conceptos fundamentales para su planteamiento.

3.1. Introducción

3.1.1. Propósito

El propósito del proyecto es proporcionar una plataforma que permita intercambiar y alquilar viviendas, para lo cual será imprescindible establecer ofertas con condiciones particulares que podrán ser solicitadas por otros usuarios. A partir de la presente especificación de requisitos será posible proceder al diseño de la arquitectura de la aplicación.

3.1.2. Ámbito

El producto deberá, además de satisfacer las funcionalidades que establece esta especificación, contemplar los aspectos que se enumeran a continuación:

- **Internacionalización y localización:** el producto posibilitará su internacionalización y estará disponible en valenciano, castellano e inglés. Además, las fechas se mostrarán en el formato correspondiente al idioma que esté en uso.
- **Despliegue:** la aplicación se encontrará disponible bajo el dominio swapper.es.
- **Seguridad:** toda información entre servidor y usuario estará cifrada de punto a punto mediante el protocolo criptográfico TLS. Además se tomarán las medidas necesarias para almacenar y tratar la información personal de los usuarios de acuerdo con la LOPD.
- **Escalabilidad:** la aplicación se adaptará a cualquier cantidad de elementos. En caso de existir limitaciones de hardware, la arquitectura permitirá ser escalada sin necesidad de modificar el código fuente.

3.1.3. Definiciones, acrónimos y abreviaturas

- **Alojamiento.** Hospedaje para el que es posible crear ofertas de alquiler e intercambio.
- **Alquiler.** Tipo de oferta en la que un usuario paga por disponer de un alojamiento de forma temporal.
- **API:** Application Programming Interface (en español, interfaz de programación de aplicaciones). Es un conjunto de subrutinas, funciones y procedimientos ofrecidos por una biblioteca para ser utilizado por otro software como capa de abstracción.

- **BD:** base de datos. Capa de software encargada de almacenar de forma persistente un conjunto de datos pertenecientes a un mismo contexto.

- **Cuenta.** Conjunto de información que identifica a un único usuario del sistema.

- **ECMAScript.** Especificación de lenguaje de *scripting* en la que se basa JavaScript.

- **Intercambio.** Tipo de oferta en la que un usuario ofrece un alojamiento a cambio de disponer de otro alojamiento de forma temporal, normalmente durante el mismo periodo de tiempo.

- **JSON:** JavaScript Object Notation (en español, notación de objetos de JavaScript). Es un formato ligero de texto para el intercambio de datos. Está inspirado en el lenguaje de programación JavaScript, pero puede ser usado perfectamente en otros contextos.

- **MVC:** modelo-vista-controlador, un patrón de arquitectura de software que desglosa la misma en datos, lógica de negocio e interfaz por medio de los componentes que dan nombre al patrón.

- **LOPD:** Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal.

- **Oferta.** Propuesta de intercambio o alquiler de un determinado alojamiento.

- ***Responsive*.** Es la cualidad de una interfaz para adaptarse a diferentes resoluciones de forma nativa.

- **TLS:** Transport Layer Security. Es un protocolo criptográfico que permite establecer conexiones seguras de punto a punto mediante criptografía asimétrica.

- **URL:** Uniform Resource Locator. Es una sucesión de caracteres que indican la localización de un recurso en Internet y que permiten acceder a él.

- **WAI-ARIA:** Accessible Rich Internet Applications. Es una especificación que establece la forma de definir semántica en un documento HTML enfocada a las tecnologías de ayuda a personas con diversidad funcional.

- **W3C:** World Wide Web Consortium. Es un consorcio internacional que se encarga de desarrollar y promover los estándares para la World Wide Web.

3.2. Descripción general

Este apartado describe el producto de forma abstracta y general, sin entrar en concreciones técnicas.

3.2.1. Perspectiva del producto

Se pretende desarrollar un producto orientado a diferentes plataformas simultáneamente, ya sea en cuanto a sistema operativo como en lo referente al tamaño de la pantalla. Por tanto, el producto funcionará en todo dispositivo electrónico de consumo que cuente con un navegador web que respete los estándares actuales de la industria. Esto se conseguirá mediante la aplicación de planteamientos *responsive* (adaptativos) para que la aplicación no dependa de un tamaño de pantalla concreto; en ningún caso se desarrollarán diferentes *front-end*.

El producto no dependerá de servicios ni recursos externos, es decir, el resultado final será una aplicación autocontenida.

3.2.2. Funciones del producto

El producto deberá satisfacer las funcionalidades que se enumeran y detallan a continuación:

- **Gestión de cuentas:** los usuarios podrán registrar cuentas e iniciar sesión en la aplicación con ellas para realizar todas las funciones permitidas. Además, también podrán modificar y eliminar su información personal.
- **Gestión de alojamientos:** se permitirá ver, buscar, crear, modificar y eliminar alojamientos, incluyendo toda su información relacionada.
- **Gestión de imágenes:** se permitirá ver, crear y eliminar imágenes, además de poder asignarlas como avatar de una cuenta, cabecera de una cuenta o foto de un alojamiento.
- **Gestión de ofertas:** el sistema permitirá la visión, búsqueda, creación, modificación y borrado de ofertas asociadas a un determinado alojamiento.
- **Gestión de solicitudes:** se posibilitará la visión, creación, aceptación, rechazo y valoración de solicitudes de intercambio o alquiler para una determinada oferta.
- **Comunicaciones internas:** los usuarios podrán ponerse en contacto entre ellos mediante un sistema de mensajería privada.
- **Notificaciones:** se notificará al usuario correspondiente cuando reciba una solicitud y cuando una solicitud enviada previamente se confirme o rechace.

Cabe destacar que el producto no ofrecerá ningún método de pago. Su propósito es la puesta en contacto entre interesados, por lo que la gestión económica queda a cargo de los mismos.

3.2.3. Características del usuario

El producto está orientado a los usuarios con propiedades en desuso que decidan intercambiarlas o alquilarlas temporalmente. Pueden tener cualquier edad, aunque al tener propiedades en desuso lo más probable es que no sean muy jóvenes.

Existirán dos roles de usuario, que se detallan en las siguientes tablas.

Tipo	Usuario anónimo
Acciones permitidas	- Crear y acceder a cuentas. - Ver perfiles. - Ver y buscar alojamientos. - Ver imágenes. - Ver ofertas.

Tabla 3. Características de los usuarios anónimos.

Tipo	Usuario autenticado
Acciones permitidas	- Modificar su cuenta. - Ver y buscar alojamientos. Crear, modificar y borrar los propios. - Ver imágenes. Asignarlas y desasignaslas como avatar, cabecera o hacerlo a un alojamiento. - Ver ofertas. Crear, modificar y borrar las propias. - Ver y crear las solicitudes propias. Aceptar y rechazar las dirigidas a él. Valorar las enviadas y aceptadas. - Enviar y recibir mensajes privados a los demás usuarios. - Recibir notificaciones.

Tabla 4. Características de los usuarios autenticados.

3.2.4. Restricciones generales

- El producto será una aplicación web interactiva.
- El servidor deberá poder atender una cantidad considerable de peticiones concurrentes.
- El servidor dispondrá de un espacio de almacenamiento que pueda contener tanto la base de datos como las imágenes que los usuarios suban.

3.2.5. Suposiciones y dependencias

Se espera la utilización, por parte del usuario, de navegadores que cumplan con los estándares que recomienda actualmente el World Wide Web Consortium y de los que la aplicación hace uso. La mayoría de navegadores actuales se mantienen actualizados automáticamente, por lo que es viable suponer que no habrá problemas de compatibilidad en este sentido.

3.3. Requisitos específicos

Este apartado describe el producto de forma detallada, abordando aspectos técnicos. De esta forma es posible conocer qué aspectos debe cumplir la aplicación para ser considerada como plenamente funcional.

3.3.1. Requisitos de interfaz externos

3.3.1.1. Interfaz de usuario

Estos requisitos son aquellos que deberá cumplir la interfaz de la aplicación.

- **IR-1.** Debe ser compatible con la última especificación de los estándares recomendados por el World Wide Web Consortium para el desarrollo de aplicaciones web.

- **IR-2.** Ha de ser adaptativa, utilizando para ello las cualidades *responsive* que contemplan los estándares correspondientes.

- **IR-3.** Debe estar disponible en valenciano, castellano e inglés. Por defecto utilizará el idioma del sistema operativo del usuario. Además, permitirá modificarlo a los usuarios registrados.

- **IR-4.** Ha de respetar la especificación WAI-ARIA del W3C, cumpliendo así con los estándares de accesibilidad de la World Wide Web.

- **IR-5.** No debe depender de eventos del ratón u otro tipo de evento que no sea multiplataforma.

3.3.2. Requisitos funcionales

Los requisitos funcionales establecen el comportamiento de la aplicación.

- **FR-1. Ver perfil.** Ha de ser posible acceder a los perfiles de los usuarios conociendo su alias. En estos perfiles se ha de mostrar toda la información pública que no viole la privacidad del usuario: nombre, alias, idioma, fecha de registro, avatar y cabecera.

- **FR-2. Crear cuenta.** Se podrán crear cuentas de usuario indicando una dirección de correo electrónico válida, un alias y una contraseña.

- **FR-3. Iniciar sesión.** Deberá ser posible iniciar sesión mediante la dirección de correo y contraseña de una cuenta previamente registrada.

- **FR-4. Cerrar sesión.** Será posible cerrar una sesión previamente iniciada.

- **FR-5. Modificar cuenta.** Debe ser posible editar y eliminar cualquier dato de una cuenta, exceptuando el alias, la dirección de correo y la contraseña, que únicamente podrán ser editados.

- **FR-6. Ver alojamiento.** Se ofrecerá la posibilidad de acceder a los detalles de un alojamiento, donde se mostrarán su datos completos (incluyendo imágenes y valoración media) y sus ofertas relacionadas.

- **FR-7. Buscar alojamiento.** Será posible buscar alojamientos por ubicación. No se mostrarán los alojamientos del usuario que realiza la búsqueda, si procede.

- **FR-8. Crear alojamiento.** Debe ofrecerse la posibilidad de crear alojamientos nuevos, indicando su nombre, una descripción, su dirección (incluyendo código postal, localidad y país) y el número máximo de huéspedes que puede albergar.

- **FR-9. Modificar alojamiento.** Podrá modificarse toda la información de un alojamiento. Además, también se ofrecerá la posibilidad de asociar y desasociar imágenes.

- **FR-10. Eliminar alojamiento.** Será posible eliminar un alojamiento, siempre y cuando no tenga ofertas relacionadas.

- **FR-11. Ver oferta.** Se posibilitará la visualización detallada de las ofertas relacionadas con un alojamiento.

- **FR-12. Buscar oferta.** Será posible buscar ofertas por fecha. No se mostrarán las ofertas del usuario que realiza la búsqueda, si procede.

- **FR-13. Crear oferta.** Debe ofrecerse la posibilidad de crear ofertas asociadas a un alojamiento, indicando la fecha mínima de llegada, la fecha máxima de salida, si permite intercambios o no, el número de ofertas iguales que se quieren ofrecer, el precio por noche y el precio si se opta por el periodo completo que se oferta.

- **FR-14. Modificar oferta.** Podrá modificarse toda la información de una oferta excepto las fechas, por motivos de integridad.

- **FR-15. Eliminar oferta.** Se podrá eliminar toda oferta que no tenga solicitudes relacionadas.

- **FR-16. Ver solicitud.** Debe ser posible ver todas las solicitudes, clasificándose en enviadas y recibidas.

- **FR-17. Crear solicitud.** Se podrá crear una solicitud para una oferta existente y con plazas disponibles, indicando la fecha de llegada y de salida.

- **FR-18. Aceptar solicitud.** Será posible aceptar una solicitud recibida, lo que confirmará la petición del usuario.

- **FR-19. Rechazar solicitud.** Debe poderse rechazar una solicitud recibida, lo que denegará la petición del usuario.

- **FR-20. Valorar solicitud.** Los usuarios que han recibido confirmación para una solicitud podrán valorar su experiencia con la misma. Dicha puntuación se mostrará en el alojamiento correspondiente.

- **FR-21. Ver mensaje.** Se ofrecerá la posibilidad de ver los mensajes recibidos y enviados, clasificándolos por conversaciones con una cuenta determinada.

- **FR-22. Crear mensaje.** Debe ser posible enviar un mensaje a otra cuenta. En caso de no existir una conversación previa, se posibilitará el acceso a dicha conversación vacía.

- **FR-23. Ver notificación.** Se podrán ver las notificaciones recibidas.

3.3.3. Requisitos lógicos de base de datos

- **LR-1.** La base de datos deberá ser de tipo relacional, ya que es el modelo de datos más adecuado para este propósito.

- **LR-2.** Se establecerán las restricciones de integridad necesarias tanto en el modelo de la base de datos como en la lógica de la aplicación para garantizar la integridad de la información en todo momento.

3.3.4. Requisitos de rendimiento

- **PR-1.** El sistema deberá tener un tiempo de respuesta no superior a un segundo, de acuerdo con [Nielsen 1993].

3.3.5. Restricciones de diseño

Al utilizarse un sistema de gestión de bases de datos concreto será necesario hacer uso de un lenguaje de programación y/o *framework* que disponga de una API que permita conexiones a dicho sistema.

3.3.6. Atributos

3.3.6.1. Disponibilidad

Se establecerá un acuerdo de nivel de servicio (SLA) superior al 99%. Cuando el servicio no esté disponible se responderá con un estado HTTP 503 (*service unavailable*).

3.3.6.2. Fiabilidad

La aplicación deberá ser fiable, es decir, la probabilidad de su buen funcionamiento deberá tender a 1. En caso de fallo se tratará en la medida de lo posible y, en caso de no poder ser tratado, deberá ser notificado al usuario de forma clara y sencilla.

3.3.6.3. Seguridad

Las conexiones entre el usuario y la aplicación se cifrarán de punto a punto mediante el uso del protocolo TLS. Se dará soporte únicamente a las versiones 1, 1.1 y 1.2 del mismo. Por tanto, será incompatible con el protocolo SSL, que es considerado inseguro en la actualidad. Se utilizarán certificados firmados por una autoridad de certificación válida. Las claves de cifrado serán de curva elíptica (ECDSA) de tipo *secp256r1* (o equivalente) y la firma utilizará el algoritmo de resumen SHA-256.

El *back-end* de la aplicación requerirá un *token* de sesión para cada solicitud que requiera autenticación. Además, realizará las comprobaciones de autorización necesarias para garantizar que dicha acción puede ser realizada por el solicitante. Dicho *token* será un número pseudoaleatorio con una longitud no inferior a los 256 bits, que se almacenará en formato binario. Fuera de la base de datos se trabajará con él en base 62. El sistema hará una recolección automática de los *tokens* de sesión que no se hayan utilizado en un tiempo determinado.

En cuanto a las contraseñas de los usuarios, estas se almacenarán cifradas mediante el algoritmo Blowfish. Este algoritmo permite comprobar si un valor dado es igual al valor cifrado sin necesidad de conocer la semilla que se utilizó para cifrar, por lo que se utilizará una semilla aleatoria que será desechada. De esta forma, si un atacante consiguiese acceso a la base de datos, sería computacionalmente impracticable que obtuviese alguna contraseña.

Por otra parte, el inicio de sesión se realizará mediante la dirección de correo electrónico del usuario y su contraseña. La dirección de correo será totalmente privada, de tal forma que un posible atacante no podría intentar iniciar sesión sin conocerla. Cuando se produzca un intento fallido de inicio de sesión, tanto el *back-end* como el *front-end* emitirán la misma respuesta independientemente de cuál sea la razón de que dicho inicio de sesión no se haya producido correctamente. Además, se implementará una protección contra ataques de fuerza bruta dirigidos al inicio de sesión. Se limitarán estos intentos fallidos por hora y por IP.

En cuanto al servidor, se inhabilitarán los inicios de sesión mediante el protocolo SSH que no hagan uso de certificados digitales.

Por último, el *back-end* establecerá un máximo de peticiones por minuto y por IP. Esta limitación será efectiva para todo el *back-end*, estableciéndose así una protección doble para el inicio de sesión.

4. Análisis

El objetivo de la fase de análisis es definir los requisitos de la aplicación, independientemente de cómo se vayan a satisfacer. Para ello se utiliza el lenguaje de modelado UML.

4.1. Diagrama de clases

Para especificar la estructura del sistema se ha diseñado un diagrama de clases, el cual muestra las clases del sistema explicadas a continuación, sus atributos y sus relaciones.

- **Account:** cuenta. Representa a un usuario en el sistema y contiene toda su información personal.
- **Role:** rol. Cada rol es un conjunto de acciones permitidas (permisos) relacionadas entre sí de algún modo. Por ejemplo, un administrador tendría todos los permisos posibles.
- **Session:** sesión. Representa un conjunto de información que permite la autenticación de un usuario frente al sistema.
- **Sign in attempt:** intento de inicio de sesión. Representa un intento fallido de inicio de sesión. Sirve para implementar una protección contra ataques de fuerza bruta y obtener estadísticas sobre ataques en curso, pudiendo así mitigarlos.
- **Country:** país. Cada país incluye su código *alpha-2*, definido por el estándar ISO 3166-1, y su código de llamada, de acuerdo con lo que establece el estándar ITU-T.
- **Accommodation:** alojamiento. Representa un alojamiento y toda su información. Es propiedad de una cuenta.
- **Offer:** oferta. Representa una oferta, la cual está asociada a un alojamiento.
- **Request:** solicitud. Representa una solicitud, ya sea de intercambio o alquiler, asociada a una oferta, que a su vez está asociada a un alojamiento.
- **Image:** imagen. Se trata de la representación de un archivo de tipo GIF, JPEG, PNG o SVG subido por un usuario. Puede ser el avatar de una cuenta, la cabecera de una cuenta o estar asociada a un alojamiento a través de la relación “image of accommodation”.
- **Image of accommodation:** imagen de alojamiento. Es la relación entre las clases *accommodation* e *image*. Es necesaria debido a que contiene información adicional derivada de la relación, que en este caso se trata de la posición de la imagen respecto a las demás.
- **Message:** mensaje. Representa un mensaje privado enviado de una cuenta a otra.
- **Notification:** notificación. Representa la notificación de un suceso a una cuenta.

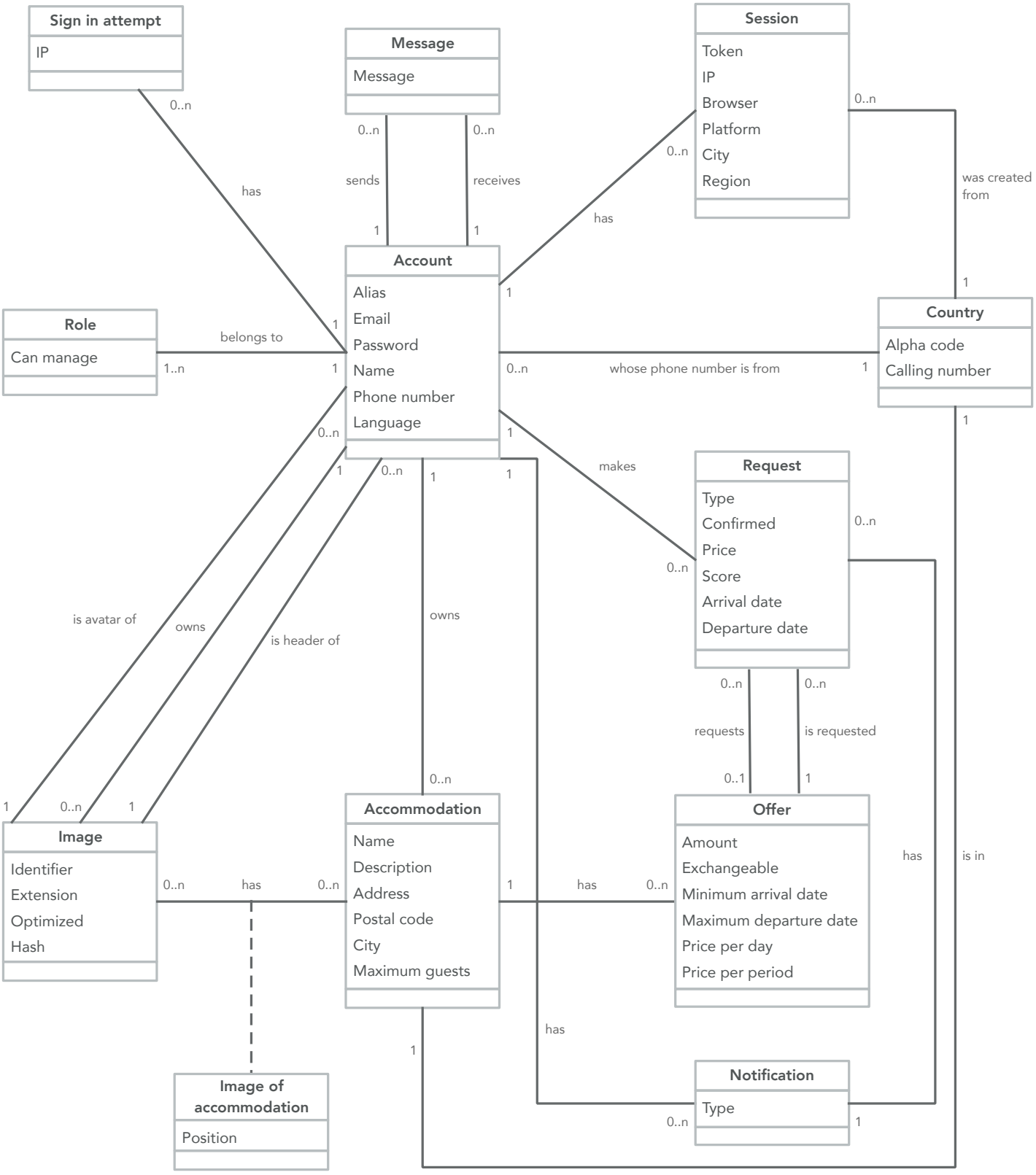

Figura 5. Diagrama de clases de la aplicación.

4.2. Diagrama de casos de uso

El diagrama de casos de uso representa gráficamente la interacción del usuario con el sistema, mostrando la relación del mismo con los diferentes casos de uso establecidos.

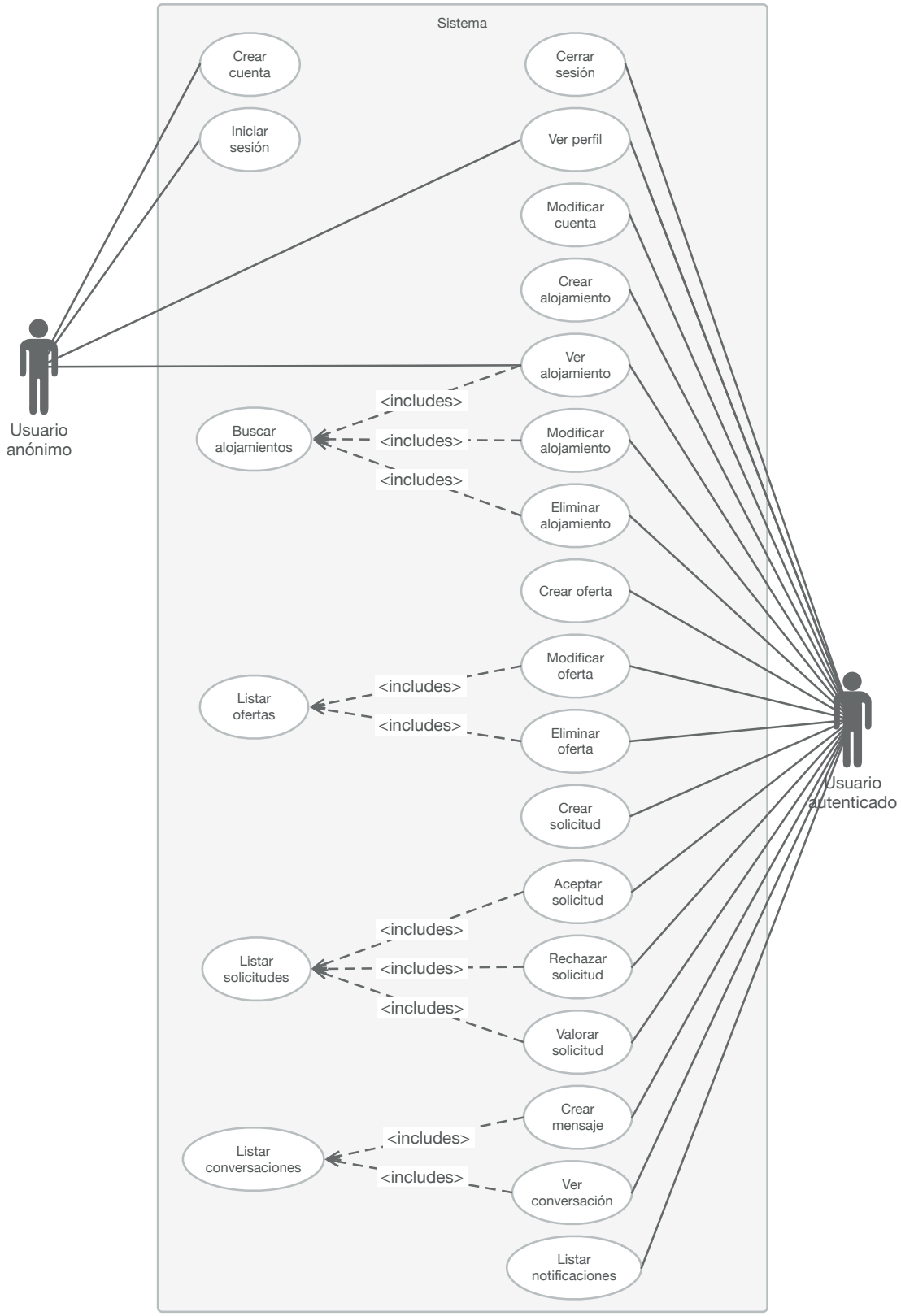

Figura 6. Diagrama de casos de uso de la aplicación.

4.3. Casos de uso

Caso de uso	Crear cuenta
Actores	Usuario anónimo
Descripción	Permite a un usuario crear una cuenta nueva con la que poder acceder a todas las funcionalidades de la aplicación.
Secuencia principal	- El usuario solicita registrarse. - El sistema le facilita la posibilidad de indicar sus datos. - El usuario introduce la información que se le solicita. - El sistema comprueba que los datos no producen ningún conflicto. - El sistema crea la cuenta e inicia la sesión automáticamente, devolviendo al usuario donde se encontraba previamente.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces. - Si ya existe una cuenta con la misma dirección de correo electrónico o alias, se le indicará al usuario que debe modificar estos datos.
Requisitos	Ninguno.

Tabla 5. Caso de uso #1: “crear cuenta”.

Caso de uso	Iniciar sesión
Actores	Usuario anónimo
Descripción	Permite a un usuario acceder a una cuenta previamente creada para utilizar la aplicación autenticándose con ella.
Secuencia principal	- El usuario solicita iniciar sesión. - El sistema le facilita la posibilidad de indicar sus datos. - El usuario introduce la información que se le solicita. - El sistema comprueba que los datos son correctos. - El sistema crea una sesión y se devuelve al usuario a la ruta donde se encontraba inicialmente.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces. - Si los datos introducidos no se corresponden con los de una cuenta (ya sea la dirección de correo electrónico o la contraseña), se mostrará un aviso al usuario para que introduzca los datos correctos. - Si el usuario ya ha realizado más intentos de inicio de sesión de los permitidos, se le notificará igualmente.
Requisitos	Ninguno.

Tabla 6. Caso de uso #2: “iniciar sesión”.

Caso de uso	Cerrar sesión
Actores	Usuario autenticado
Descripción	Permite a un usuario eliminar una sesión tanto del *front-end* como del *back-end*, pasando de esa forma a ser un usuario anónimo.
Secuencia principal	- El usuario solicita el cierre de sesión. - El sistema invalida la sesión y, en caso de que se encuentre en una ruta que requiera autenticación, lo envía de vuelta a la vista principal.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 7. Caso de uso #3: “cerrar sesión”.

Caso de uso	Ver perfil
Actores	Usuario anónimo y usuario autenticado
Descripción	Permite ver la información pública de una cuenta: imagen de avatar, imagen de cabecera, nombre, alias, idioma y fecha de registro.
Secuencia principal	- El usuario pulsa sobre un elemento que enlaza con un el perfil de una cuenta. - El sistema le proporciona la información pública del usuario.
Flujos alternativos	Ninguno.
Requisitos	Ninguno.

Tabla 8. Caso de uso #4: “ver perfil”.

Caso de uso	Modificar cuenta
Actores	Usuario autenticado
Descripción	Permite modificar la información de una cuenta: alias, dirección de correo electrónico, nombre, número de teléfono, país del número de teléfono, idioma, contraseña, avatar y cabecera. Además, permite eliminar cualquier información excepto el alias, dirección de correo electrónico y contraseña.
Secuencia principal	- El usuario solicita modificar sus datos. - El sistema le facilita la posibilidad de indicar los nuevos datos. - El usuario introduce la información que desea. - El sistema comprueba que los datos no producen ningún conflicto. - Se modifican los datos y se reflejan los cambios.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces. - Si ya existe una cuenta con la misma dirección de correo electrónico o alias, se le indicará al usuario que debe modificar estos datos.
Requisitos	El usuario debe estar autenticado.

Tabla 9. Caso de uso #5: “modificar cuenta”.

Caso de uso	Crear alojamiento
Actores	Usuario autenticado
Descripción	Permite crear un alojamiento especificando la siguiente información: nombre, descripción, dirección, código postal, localidad, país y número máximo de huéspedes.
Secuencia principal	- El usuario solicita la creación de un alojamiento. - El sistema le facilita la posibilidad de indicar los datos. - El usuario introduce la información que se le solicita. - El sistema comprueba que los datos no producen ningún conflicto. - Se crea un nuevo alojamiento y así se refleja.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	El usuario debe estar autenticado.

Tabla 10. Caso de uso #6: “crear alojamiento”.

Caso de uso	Buscar alojamientos y ofertas
Actores	Usuario anónimo y usuario autenticado
Descripción	Permite buscar alojamientos y ofertas de entre los que se encuentran publicados. Requiere especificar una ciudad, una fecha de llegada y una fecha de salida. A partir de estas condiciones se mostrarán los alojamientos con ofertas que cumplan con las mismas. En caso de que el usuario esté autenticado, nunca mostrará sus alojamientos.
Secuencia principal	- El usuario solicita buscar alojamientos. - El sistema le facilita la posibilidad de indicar los datos de búsqueda. - El usuario introduce la información que se le solicita. - El sistema muestra los alojamientos y ofertas que concuerdan con los criterios de búsqueda. - El usuario puede pulsar sobre cualquiera de ellos para obtener más información en detalle.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	Ninguno.

Tabla 11. Caso de uso #7: “buscar alojamientos y ofertas”.

Caso de uso	Ver alojamiento
Actores	Usuario anónimo y usuario autenticado
Descripción	Muestra un alojamiento, ofreciendo la siguiente información: nombre, descripción, puntuación media, dirección, código postal, localidad, país y número máximo de huéspedes. También permite abrir una galería con las fotos asociadas al alojamiento. Además, muestra las ofertas asociadas al alojamiento, indicando su fecha mínima de llegada, fecha máxima de salida, precio por noche (si procede), precio por periodo completo (si procede) y la oferta de intercambio (si procede).
Secuencia principal	- El usuario solicita ver un alojamiento en detalle. - El sistema le muestra toda la información relacionada con el mismo.
Flujos alternativos	Ninguno.
Requisitos	Ninguno.

Tabla 12. Caso de uso #8: “ver alojamiento”.

Caso de uso	Modificar alojamiento
Actores	Usuario autenticado
Descripción	Permite modificar la siguiente información de un alojamiento: nombre, descripción, dirección, código postal, localidad, país y número máximo de huéspedes. También permite adjuntar fotos.
Secuencia principal	- El usuario solicita modificar los datos de un alojamiento. - El sistema le facilita la posibilidad de indicar los nuevos datos. - El usuario introduce la información que desea. - El sistema comprueba que los datos no producen ningún conflicto. - Se modifican los datos y se reflejan los cambios.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	El usuario debe estar autenticado.

Tabla 13. Caso de uso #9: “modificar alojamiento”.

Caso de uso	Eliminar alojamiento
Actores	Usuario autenticado
Descripción	Permite eliminar un alojamiento.
Secuencia principal	- El usuario solicita eliminar un alojamiento. - El sistema comprueba que el borrado no crea conflictos y lo elimina.
Flujos alternativos	- Si el alojamiento tiene ofertas, no permitirá eliminarlo y así lo indicará.
Requisitos	El usuario debe estar autenticado.

Tabla 14. Caso de uso #10: “eliminar alojamiento”.

Caso de uso	Crear oferta
Actores	Usuario autenticado
Descripción	Permite crear una oferta asociada a un alojamiento, indicando su fecha mínima de llegada, fecha máxima de salida, si es intercambiable, la cantidad de ofertas como esa que se publican, el precio por noche y el precio por periodo.
Secuencia principal	- El usuario solicita la creación de una oferta. - El sistema le facilita la posibilidad de indicar los datos. - El usuario introduce la información que se le solicita. - El sistema comprueba que los datos no producen ningún conflicto. - Se crea una nueva oferta y así se refleja.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	El usuario debe estar autenticado.

Tabla 15. Caso de uso #11: “crear oferta”.

Caso de uso	Listar ofertas
Actores	Usuario autenticado
Descripción	Muestra una lista de las ofertas asociadas a un alojamiento. En ella indica sus fechas y si es de tipo alquiler, intercambio o ambos.
Secuencia principal	- El usuario solicita ver las ofertas asociadas a un alojamiento. - El sistema le facilita la información.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 16. Caso de uso #12: “listar ofertas”.

Caso de uso	Modificar oferta
Actores	Usuario autenticado
Descripción	Permite modificar los siguientes datos de una oferta: si es intercambiable, la cantidad de ofertas como esa que se publican, el precio por noche y el precio total del periodo. No es posible modificar ni la fecha mínima de llegada ni la fecha máxima de salida por cuestiones de integridad.
Secuencia principal	- El usuario solicita modificar los datos de una oferta. - El sistema le facilita la posibilidad de indicar los nuevos datos. - El usuario introduce la información que desea. - El sistema comprueba que los datos no producen ningún conflicto. - Se modifican los datos y se reflejan los cambios.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	El usuario debe estar autenticado.

Tabla 17. Caso de uso #13: “modificar oferta”.

Caso de uso	Eliminar oferta
Actores	Usuario autenticado
Descripción	Permite eliminar una oferta.
Secuencia principal	- El usuario solicita eliminar una oferta. - El sistema comprueba que el borrado no crea conflictos y la elimina.
Flujos alternativos	- Si la oferta tiene solicitudes, no permitirá eliminarla y así lo indicará.
Requisitos	El usuario debe estar autenticado.

Tabla 18. Caso de uso #14: “eliminar oferta”.

Caso de uso	Crear solicitud
Actores	Usuario autenticado
Descripción	Permite crear una solicitud para una oferta, indicando las fechas de llegada y de salida. Si la solicitud es de tipo intercambio, permitirá seleccionar una oferta propia para ofrecer a cambio. Si la solicitud es de tipo alquiler, mostrará el precio del alquiler en función del tiempo de estancia que se solicite.
Secuencia principal	- El usuario solicita la creación de una solicitud. - El sistema le facilita la posibilidad de indicar los datos. - El usuario introduce la información que se le solicita. - El sistema comprueba que los datos no producen ningún conflicto. - Se crea una nueva solicitud y así se refleja.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	El usuario debe estar autenticado.

Tabla 19. Caso de uso #15: “crear solicitud”.

Caso de uso	Listar solicitudes
Actores	Usuario autenticado
Descripción	Muestra todas las solicitudes con las que el usuario está involucrado, diferenciando entre enviadas y recibidas. La ordenación se realiza en función de la última modificación realizada. Para cada solicitud se indica la otra cuenta y alojamiento involucrados, así como su información. Se ha de indicar también de forma clara el tipo de solicitud: intercambio o alquiler.
Secuencia principal	- El usuario solicita ver las solicitudes, ya sea enviadas o recibidas. - El sistema le facilita la información.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 20. Caso de uso #16: “listar solicitudes”.

Caso de uso	Aceptar solicitud
Actores	Usuario autenticado
Descripción	Permite aceptar una solicitud recibida, iniciándose así el proceso de intercambio o alquiler, según corresponda.
Secuencia principal	- El usuario solicita aceptar una solicitud recibida. - El sistema comprueba que los datos no producen ningún conflicto. - Se acepta la solicitud y así se refleja.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 21. Caso de uso #17: “aceptar solicitud”.

Caso de uso	Rechazar solicitud
Actores	Usuario autenticado
Descripción	Permite rechazar una solicitud recibida, finalizándose así el proceso de intercambio o alquiler, según corresponda.
Secuencia principal	- El usuario solicita rechazar una solicitud recibida. - Se rechaza la solicitud y así se refleja.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 22. Caso de uso #18: “rechazar solicitud”.

Caso de uso	Valorar solicitud
Actores	Usuario autenticado
Descripción	Permite valorar, en una escala del 1 al 5, la experiencia con una solicitud aceptada. Esto permite puntuar tanto la experiencia con la solicitud como con el alojamiento.
Secuencia principal	- El usuario solicita valorar una solicitud enviada y aceptada. - El sistema almacena la valoración y así se refleja.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 23. Caso de uso #19: “valorar solicitud”.

Caso de uso	Crear mensaje
Actores	Usuario autenticado
Descripción	Permite enviar un mensaje privado a otra cuenta.
Secuencia principal	- El usuario solicita la creación de un mensaje. - El sistema le facilita la posibilidad de indicar los datos. - El usuario introduce la información que se le solicita. - El sistema crea un nuevo mensaje y así se refleja.
Flujos alternativos	- Si los datos introducidos no pasan las comprobaciones de validación, se muestran indicaciones para que se corrijan los errores, no permitiendo enviar el formulario hasta entonces.
Requisitos	El usuario debe estar autenticado.

Tabla 24. Caso de uso #20: “crear mensaje”.

Caso de uso	Listar conversaciones
Actores	Usuario autenticado
Descripción	Permite listar todas las conversaciones que tiene una cuenta con el resto de cuentas.
Secuencia principal	- El usuario solicita ver las conversaciones. - El sistema le facilita la información.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 25. Caso de uso #21: “listar conversaciones”.

Caso de uso	Ver conversación
Actores	Usuario autenticado
Descripción	Permite ver los mensajes de una conversación y la información básica de la otra cuenta.
Secuencia principal	- El usuario solicita ver una conversación en detalle. - El sistema le muestra toda la información relacionada con la misma, incluidos los mensajes.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 26. Caso de uso #22: “ver conversación”.

Caso de uso	Listar notificaciones
Actores	Usuario autenticado
Descripción	Permite ver las notificaciones recibidas, mostrando una descripción y permitiendo al usuario dirigirse directamente al apartado de la aplicación con el que poder atender el aviso que realice.
Secuencia principal	- El usuario solicita ver las notificaciones recibidas. - El sistema le facilita la información.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 27. Caso de uso #23: “listar notificaciones”.

5. Diseño

El objetivo de la fase de diseño es definir la arquitectura de la aplicación en base a los requisitos de la misma. La arquitectura define los componentes, sus interfaces y los comportamientos.

Al tratarse de una aplicación web interactiva es conveniente plantear la arquitectura en las dos partes en las que se dividirá esta aplicación distribuida: el *back-end* y el *front-end*. En ambos casos se plantea una arquitectura de tres capas (datos, lógica y presentación) que además sigue el patrón de diseño modelo-vista-controlador (MVC).

La arquitectura de tres capas es un patrón de arquitectura cliente-servidor que separa el software, como el propio nombre indica, en tres capas diferentes: **datos** (o persistencia), **lógica** (o lógica de negocio) y **presentación**. La primera capa se encarga de almacenar los datos de forma persistente, ya sea mediante el uso de sistemas de gestión de bases de datos, sistemas de archivos u otros. Debe proporcionar una interfaz a la capa de lógica para que esta pueda acceder a los datos sin importar la forma en la que estén guardados, es decir, debe encargarse de abstraer el almacenamiento de los datos. La segunda capa, la de lógica, es la encargada de controlar las diferentes funcionalidades de la aplicación mediante el procesamiento de los datos. Por último, la capa de presentación es la que interactúa con el usuario. Muestra la información que obtiene de la capa de lógica y convierte las acciones del usuario en llamadas a dicha capa.

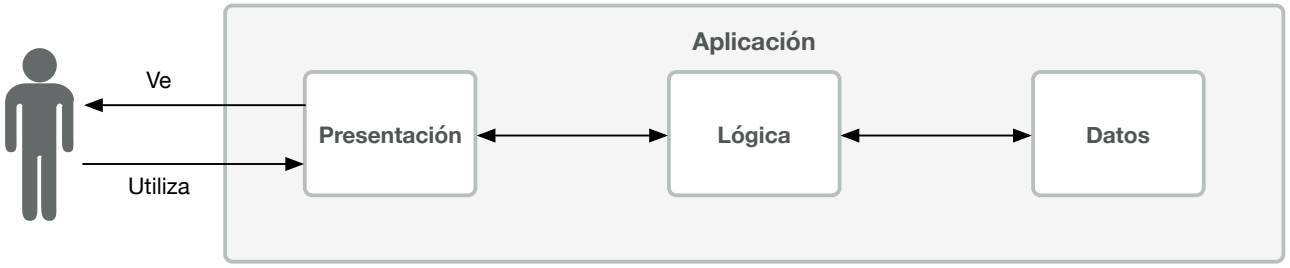

Figura 7. Flujo de interacción de la arquitectura de tres capas.

Por otra parte, el patrón de diseño modelo-vista-controlador, es un patrón de arquitectura que divide una aplicación en tres tipos de componente:

- **Modelos.** Son representaciones internas de la información almacenada. Manejan los datos y la lógica de la aplicación. Si se trabaja con bases de datos relacionales y con el paradigma de programación orientada a objetos, son clases que representan tablas de la base de datos.

- **Vistas.** Son formas de representar y mostrar la información al usuario. En una aplicación web serían archivos HTML que definen interfaces.

- **Controladores.** Se encargan de procesar la interacción del usuario y transformarla en interacciones con los modelos. En una aplicación web se definirían mediante código JavaScript.

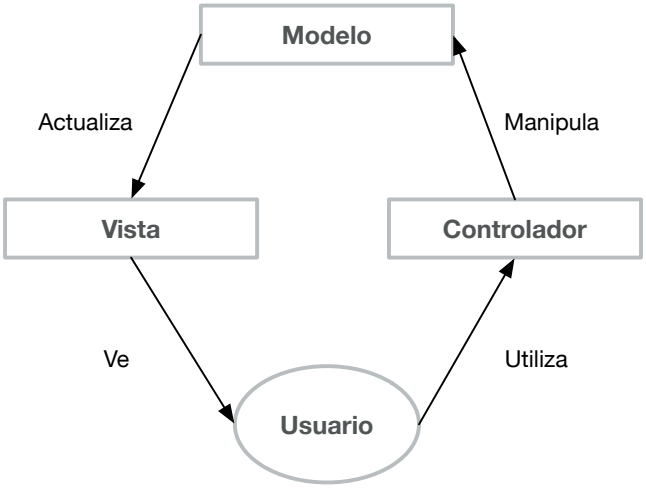

Figura 8. Flujo de interacción del patrón modelo-vista-controlador.

Combinando ambos modelos se obtiene la arquitectura que se plantea para el desarrollo de esta aplicación.

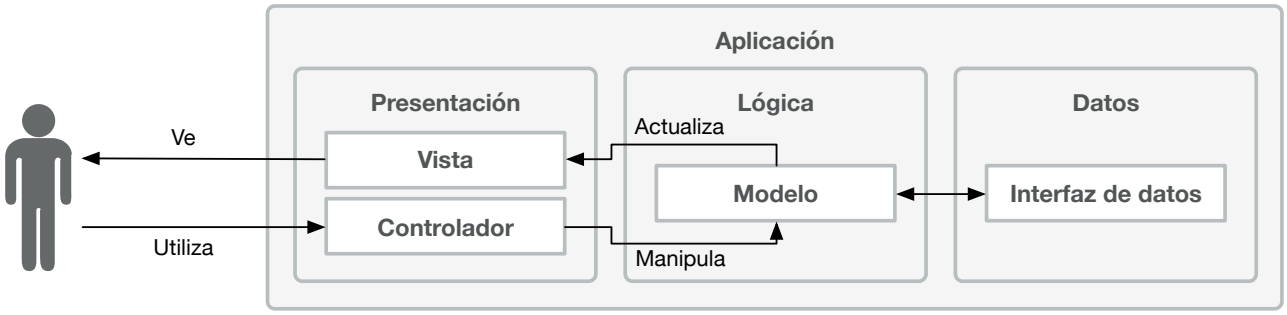

Figura 9. Flujo de interacción de la arquitectura de tres capas y el patrón MVC.

Este planteamiento permite una gran facilidad y modularidad a la hora de la implementación. De esta forma es posible la reutilización del código y se abre la puerta a poder incorporar fácilmente cualquier mejora en el proyecto una vez finalizado su desarrollo.

5.1. *Back-end*

El *back-end* de la aplicación consiste en una *RESTful* API escrita en PHP utilizando el *micro-framework* Lumen. Es la parte que se ejecuta en el servidor.

5.1.1. Capa de datos

Como se ha indicado anteriormente, la capa de datos o persistencia es la encargada de almacenar la información de forma permanente y hacerla accesible a la capa de lógica. En este caso se opta por utilizar un sistema de gestión de bases de datos relacionales (SGBDR), concretamente MariaDB.

MariaDB es una bifurcación de MySQL liderada por los desarrolladores originales de este último. Dicha bifurcación se realizó debido a las preocupaciones por la compra de MySQL por parte de Oracle. En la actualidad se trata de un SGBD con mejor rendimiento que MySQL y con un ritmo de evolución más ágil. Además, sigue una política de transparencia en sus actualizaciones de seguridad, especificando los identificadores CVE (Common Vulnerabilities and Exposures) de los problemas, tanto existentes como solucionados.

A continuación, antes de pasar a mostrar el diagrama entidad-relación resultante del modelado de la base de datos, se detallan aquellos aspectos del modelo que se consideran destacables.

- Todas las tablas disponen de una columna `id`, que es su clave primaria, a excepción de las tablas derivadas de relaciones de tipo muchos a muchos.
- Todas las columnas `id` son de tipo entero de 4 bytes sin signo. Por tanto, todos los campos que las referencian mediante claves ajenas también lo son.
- Los campos `ip` son de tipo `VARBINARY(16)` para poder almacenar tanto direcciones IPv4 como IPv6.
- Se definen dos vistas: `conversations_ungrouped` y `conversations`, cuyas declaraciones se muestran a continuación, respectivamente:

```
1  CREATE VIEW conversations_ungrouped AS
2  (
3        SELECT id, receiver_id AS account_id, sender_id AS viewer_id, 0 AS
   is_incoming, message, created_at, updated_at, seen_at, deleted_at
4        FROM messages
5        WHERE deleted_at IS NULL
6  ) UNION ALL (
7        SELECT id, sender_id AS account_id, receiver_id AS viewer_id, 1 AS
   is_incoming, message, created_at, updated_at, seen_at, deleted_at
8        FROM messages
9        WHERE deleted_at IS NULL
10 )
11 ORDER BY id DESC
```

```
1  CREATE VIEW conversations AS
2  SELECT * FROM conversations_ungrouped GROUP BY account_id, viewer_id
```

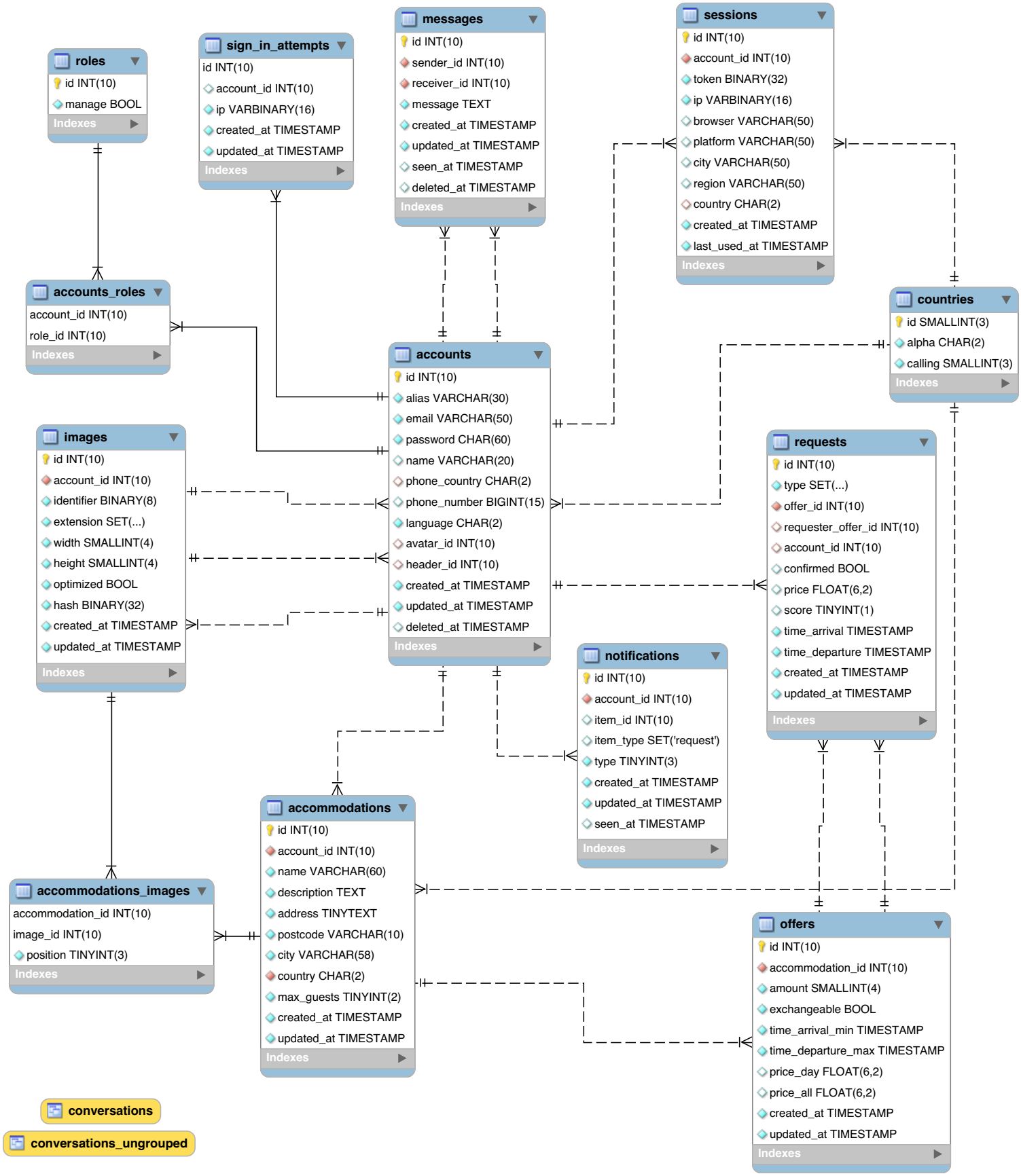

Figura 10. Diagrama entidad-relación de la aplicación.

Por último se explica en detalle lo que representa cada columna:

- ***accommodations.***
  - ***id.*** Identificador numérico interno.
  - ***account_id.*** Clave ajena a la cuenta que posee el alojamiento.
  - ***name.*** Nombre del alojamiento.
  - ***description.*** Descripción detallada del alojamiento.
  - ***address.*** Dirección donde se encuentra el alojamiento.
  - ***postcode.*** Código postal donde se encuentra el alojamiento.
  - ***city.*** Localidad donde se encuentra el alojamiento.
  - ***country.*** Clave ajena al país donde se encuentra el alojamiento.
  - ***max_guests.*** Número máximo de huéspedes que pueden ocupar el alojamiento.
  - ***created_at.*** Fecha y hora en la que se creó el alojamiento.
  - ***updated_at.*** Fecha y hora de la última vez que se modificó el alojamiento.
- ***accommodations_images.***
  - ***accommodation_id.*** Clave ajena al alojamiento que está relacionado con la imagen.
  - ***image_id.*** Clave ajena a la imagen que está relacionada con el alojamiento.
  - ***position.*** Posición de la imagen respecto a las demás imágenes del alojamiento.
- ***accounts.***
  - ***id.*** Identificador numérico interno.
  - ***alias.*** Identificador público formado por letras (mayúsculas y minúsculas), números, guiones y guiones bajos.
  - ***email.*** Dirección de correo electrónico de la cuenta.
  - ***password.*** Contraseña cifrada mediante el algoritmo Blowfish.
  - ***name.*** Nombre real del usuario.
  - ***phone_country.*** Clave ajena al país del número de teléfono.
  - ***phone_number.*** Número de teléfono del usuario.
  - ***language.*** Código ISO 639-1 del idioma del usuario.
  - ***avatar_id.*** Clave ajena a la imagen de avatar del usuario.
  - ***header_id.*** Clave ajena a la imagen de cabecera del usuario.
  - ***created_at.*** Fecha y hora en la que se creó la cuenta.
  - ***updated_at.*** Fecha y hora de la última vez que se modificó la cuenta.
  - ***deleted_at.*** Fecha y hora de la cuando se eliminó la cuenta, para implementar borrado blando.

- ***accounts_roles.***
  - ***account_id.*** Clave ajena al rol que está relacionado con la cuenta.
  - ***role_id.*** Clave ajena a la cuenta que está relacionada con el rol.
- ***countries.***
  - ***id.*** Identificador numérico interno.
  - ***alpha.*** Código ISO 3166-1 alpha-2 del país.
  - ***calling.*** Código de llamada ITU-T E.164 del país.
- ***images.***
  - ***id.*** Identificador numérico interno.
  - ***account_id.*** Clave ajena a la cuenta que creó la imagen.
  - ***identifier.*** Número aleatorio de 64 bits utilizado como identificador público, convertido a base 62 cuando se utiliza fuera de la base de datos.
  - ***extension.*** Extensión del archivo de la imagen: GIF, JPG, PNG o SVG.
  - ***width.*** Ancho en píxeles de la imagen.
  - ***height.*** Alto en píxeles de la imagen.
  - ***optimized.*** Si la imagen ha sido optimizada o no.
  - ***hash.*** Salida de la función resumen SHA-256, utilizada para evitar imágenes repetidas.
  - ***created_at.*** Fecha y hora en la que se creó la imagen.
  - ***updated_at.*** Fecha y hora de la última vez que se modificó la imagen.
- ***messages.***
  - ***id.*** Identificador numérico interno.
  - ***sender_id.*** Clave ajena a la cuenta que envió el mensaje.
  - ***receiver_id.*** Clave ajena a la cuenta que recibió el mensaje.
  - ***message.*** El texto del propio mensaje.
  - ***created_at.*** Fecha y hora en la que se creó el mensaje.
  - ***updated_at.*** Fecha y hora de la última vez que se modificó el mensaje.
  - ***seen_at.*** Fecha y hora en la que el receptor del mensaje lo leyó.
  - ***deleted_at.*** Fecha y hora de cuando se eliminó el mensaje, para implementar borrado blando.
- ***notifications.***
  - ***id.*** Identificador numérico interno.
  - ***account_id.*** Clave ajena a la cuenta que recibe la notificación.

    - ***item_id.*** Clave primaria del elemento relacionado con la notificación.
    - ***item_type.*** Tipo de elemento relacionado con la notificación.
    - ***type.*** Tipo de notificación.
    - ***created_at.*** Fecha y hora en la que se creó la notificación.
    - ***updated_at.*** Fecha y hora de la última vez que se modificó la notificación.
    - ***seen_at.*** Fecha y hora en la que el receptor de la notificación la leyó.
- ***offers.***
    - ***id.*** Identificador numérico interno.
    - ***accommodation_id.*** Clave ajena al alojamiento al que pertenece la oferta.
    - ***amount.*** Cantidad de ofertas de este tipo que están disponibles.
    - ***exchangeable.*** Si la oferta permite intercambios o no.
    - ***time_arrival_min.*** Fecha en la que la oferta empieza a aceptar huéspedes.
    - ***time_departure_max.*** Fecha en la que la oferta termina de aceptar huéspedes.
    - ***price_day.*** Precio por noche en euros.
    - ***price_all.*** Precio para el periodo completo en el que la oferta está disponible.
    - ***created_at.*** Fecha y hora en la que se creó la oferta.
    - ***updated_at.*** Fecha y hora de la última vez que se modificó la oferta.
- ***requests.***
    - ***id.*** Identificador numérico interno.
    - ***type.*** Si la oferta es un intercambio o un alquiler.
    - ***offer_id.*** Clave ajena a la oferta a la que se refiere la solicitud.
    - ***requester_offer_id.*** Clave ajena a la oferta que solicita el intercambio, si procede.
    - ***account_id.*** Clave ajena a la cuenta que solicita el alquiler, si procede.
    - ***confirmed.*** Si la solicitud ha sido confirmada por su propietario o no.
    - ***price.*** Precio total del alquiler, si procede. Si la solicitud cubre el periodo entero de la oferta, es igual al campo `price_all`. En caso contrario, es igual al número de noches multiplicado por el campo `price_day`.
    - ***score.*** Puntuación que da el usuario a su experiencia.
    - ***time_arrival.*** Fecha y hora de comienzo del intercambio o alquiler.
    - ***time_departure.*** Fecha y hora de finalización del intercambio o alquiler.
    - ***created_at.*** Fecha y hora en la que se creó la solicitud.
    - ***updated_at.*** Fecha y hora de la última vez que se modificó la solicitud.

- ***roles.***
  - ***id.*** Identificador numérico interno.
  - ***manage.*** Permiso que posibilita al usuario la realización de tareas de administración.
- ***sessions.***
  - ***id.*** Identificador numérico interno.
  - ***account_id.*** Clave ajena a la cuenta a la que pertenece la sesión.
  - ***token.*** Número aleatorio de 256 bits que sirve como forma de autenticación para demostrar que se es poseedor de una sesión. Fuera de la base de datos se representa en base 62.
  - ***ip.*** Dirección IP desde la que se inició la sesión.
  - ***browser.*** Navegador desde el que se inició la sesión.
  - ***platform.*** Sistema operativo desde el que se inició la sesión.
  - ***city.*** Ciudad desde la que se inició la sesión.
  - ***region.*** Región desde la que se inició la sesión.
  - ***country.*** Clave ajena al país desde el que se inició la sesión.
  - ***created_at.*** Fecha y hora en la que se creó la sesión.
  - ***last_used_at.*** Fecha y hora de la última vez que se utilizó la sesión.
- ***sign_in_attempts.***
  - ***id.*** Identificador numérico interno.
  - ***account_id.*** Clave ajena a la cuenta para la que se ha producido el intento fallido de inicio de sesión, si procede.
  - ***ip.*** Dirección IP que produjo el intento.
  - ***created_at.*** Fecha y hora en la que se produjo el intento.
  - ***updated_at.*** Fecha y hora de la última vez que se modificó el intento.

5.1.2. Capa de lógica

Tal y como se ha señalado con anterioridad, la capa de lógica se encarga de procesar los datos.

Para trabajar con la capa de datos se utilizará Eloquent, una biblioteca que implementa la técnica de mapeo objeto-relacional (ORM), la cual consiste en una interfaz que abstrae la capa de datos de tal forma en la que cada tabla de la base de datos corresponde a un modelo de la capa de lógica. De esta forma los controladores pueden interactuar con los modelos y, sin necesidad de ninguna implementación adicional, estos cambios se ven reflejados de forma automática en la capa de datos.

Aun así sigue siendo necesario especificar los modelos, estableciendo sus particularidades y definiendo sus relaciones, cosas que también facilita Eloquent.

Los modelos que se implementarán serán los siguientes: `Accommodation`, `Account`, `Conversation`, `Country`, `Image`, `Message`, `Notification`, `Offer`, `Request`, `Role`, `Session` y `SignInAttempt`. Además, se implementará un modelo abstracto llamado `Model`, del que heredarán el resto de modelos para establecer en él aspectos comunes.

5.1.3. Capa de presentación

Es común pensar que una *RESTful* API no dispone de capa de presentación. Si bien es cierto que parece no disponer de vistas como tales, sí tiene controladores. Además, las vistas están implícitas en el propio *framework* o lenguaje de programación. En este caso consisten en una interfaz que transforma la estructura de datos interna del lenguaje en una respuesta HTTP cuyo cuerpo está en formato JSON. Por otra parte, le añade las cabeceras HTTP correspondientes.

Los controladores a implementar serán los siguientes, cada uno con sus clases pertinentes, que tienen una relación directa con los casos de uso que se indican.

Controladores	Métodos	Casos de uso
AccommodationController	create() 6 getCollection() 7 get() 8 edit() 9 delete() 10 attachImage() 9 detachImage() 9	
AccountController	create() 1 get() 4 edit() 5	
ConversationController	getCollection() 21	
ImageController	create() 5 y 9	
MessageController	create() 20 getCollection() 22	
NotificationController	getCollection() 23	

Controladores	Métodos	Casos de uso
OfferController	create()	11
	getCollection()	12
	edit()	13
	delete()	14
RequestController	create()	15
	getCollectionRelatedToAccount()	16
	getCollectionRelatedToOffer()	16
	edit()	17, 18 y 19
SessionController	create()	2
	delete()	3

Tabla 28. Controladores, métodos y casos de uso relacionados.

5.2. *Front-end*

El *front-end* de la aplicación consiste en una aplicación web desarrollada haciendo uso del *framework* AngularJS.

5.2.1. Capa de datos

La capa de datos del *front-end* es gestionada por el navegador web. En este caso se pretende utilizar la API Web Storage que define el estándar HTML 5, que permite almacenar pares clave-valor de forma sencilla. Al no utilizar *cookies* se obtiene la ventaja adicional de no tener que mostrar el molesto e inapropiado aviso obligatorio sobre el uso de las mismas que requiere la Ley 34/2002 de 11 de julio, de Servicios de la Sociedad de la Información.

La única información que se pretende almacenar son los datos de sesión, además de aquella información que las dependencias requieran guardar.

5.2.2. Capa de lógica

La única lógica de la que dispone el *front-end* es la necesaria para la inicialización, estructuración y funcionamiento del mismo. Al tratarse de una interfaz ejecutada de forma remota por el usuario depende del *back-end*, que es el que se encarga de este tipo de tareas.

Se definirán diversas factorías, que son componentes de la aplicación que ofrecen servicios que pueden ser utilizados por el resto de componentes. Dichas factorías ofrecerán los siguientes servicios de la lógica:

- Manejo de las llamadas al *back-end*.
- Gestión de la autenticación.

Puede haber otras factorías para ofrecer otro tipo de servicios que no estén relacionados con la lógica de la aplicación.

5.2.3. Capa de presentación

La capa de presentación del *front-end* es la más significativa, ya que es el punto de entrada del usuario a la aplicación. Está formada por vistas, cada una de las cuales tiene asignado su controlador y su plantilla. El controlador se encarga de llamar al *back-end* para obtener la información necesaria, mientras que el *framework* se encarga de incrustar dicha información en las plantillas y gestionar los flujos de navegación e interacción.

Para el desarrollo de la interfaz se siguen las directrices del lenguaje visual Material Design. Esto permite conseguir una interfaz que respete la filosofía del diseño centrado en el usuario.

5.2.3.1. Vistas

En este apartado se exponen los bocetos de todas las vistas relevantes de la aplicación, que son todas excepto las de los apartados de “acerca” y “agradecimientos”, que se tratan de apartados de cortesía cuyo diseño es trivial y no tiene relevancia en el ámbito de este trabajo.

Para el desarrollo de los bocetos se ha hecho uso de la aplicación de prototipado Sketch, la cual permite bocetos muy realistas, con lo que estos serán muy similares a la interfaz que resulte de la implementación.

Figura 11. Boceto de la vista de navegación.

La vista de navegación consiste en un menú lateral desplegable que contiene todos los elementos que permiten la navegación por la aplicación. Dependiendo de la resolución de pantalla se mostrará desplegada automáticamente o, en su defecto, se mostrará un botón que permita desplegarla.

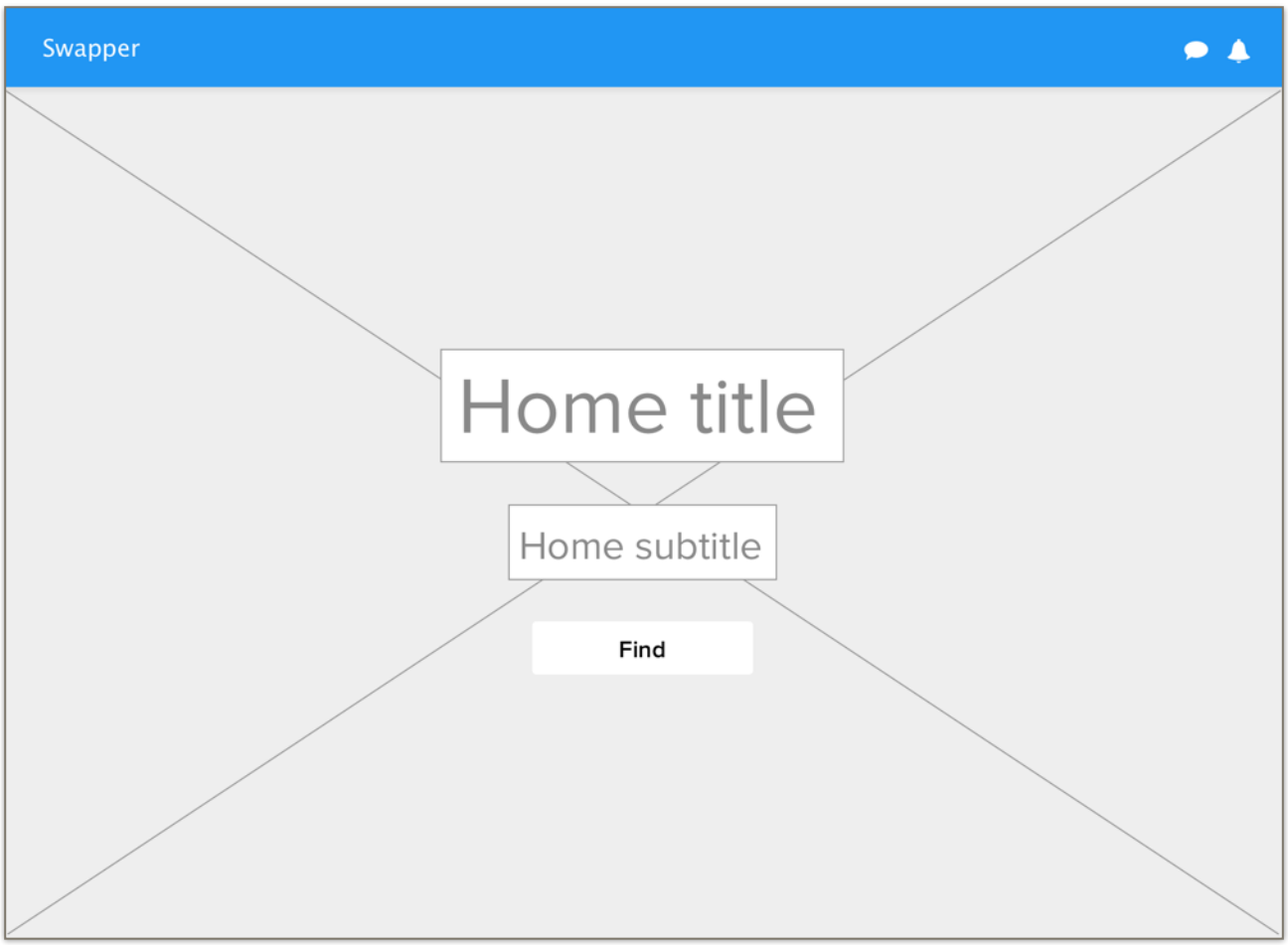

Figura 12. Boceto de la vista de inicio.

La vista de inicio sirve como punto de entrada al usuario. Muestra un título, un subtítulo y un botón que lleva directamente a la vista de búsqueda. El objetivo de esta vista es dar la bienvenida al usuario a la aplicación de una forma que le anime a empezar a hacer uso de la misma. El fondo consiste en una imagen alternante.

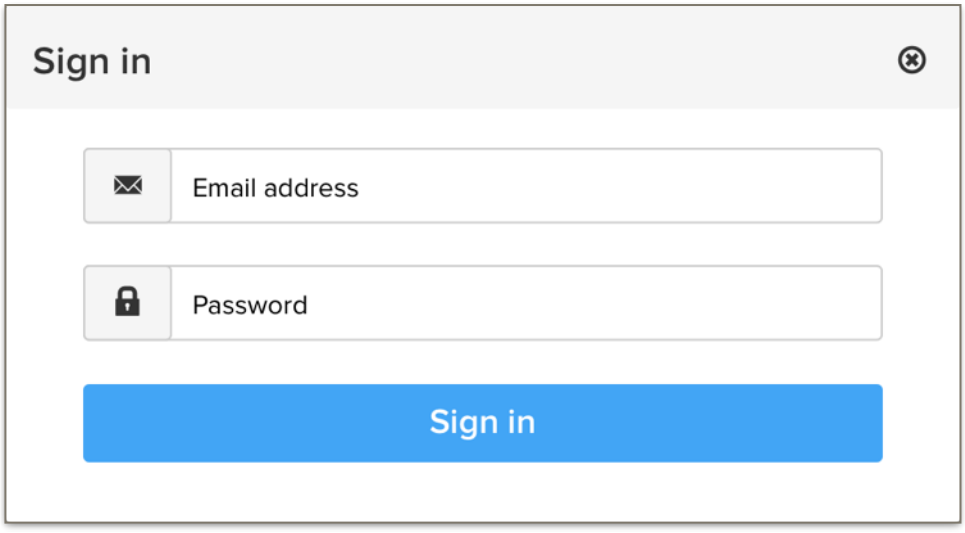

Figura 13. Boceto de la vista de inicio de sesión.

Por otra parte, las vistas de inicio de sesión y registro consisten en una sencilla ventana de diálogo.

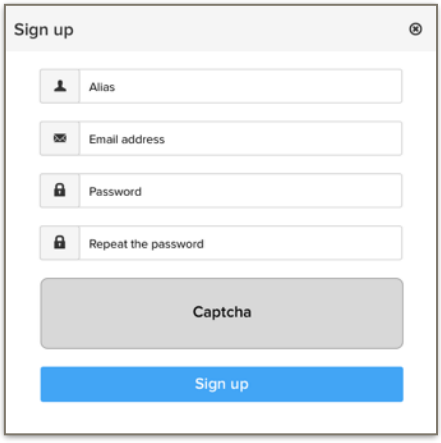

Figura 14. Boceto de la vista de registro.

Las vistas de conversaciones y mensajes muestran, respectivamente, todas las conversaciones que un usuario tiene iniciadas y todos los mensajes pertenecientes a una conversación. Además, la primera permite acceder a una conversación en concreto e iniciar una nueva.

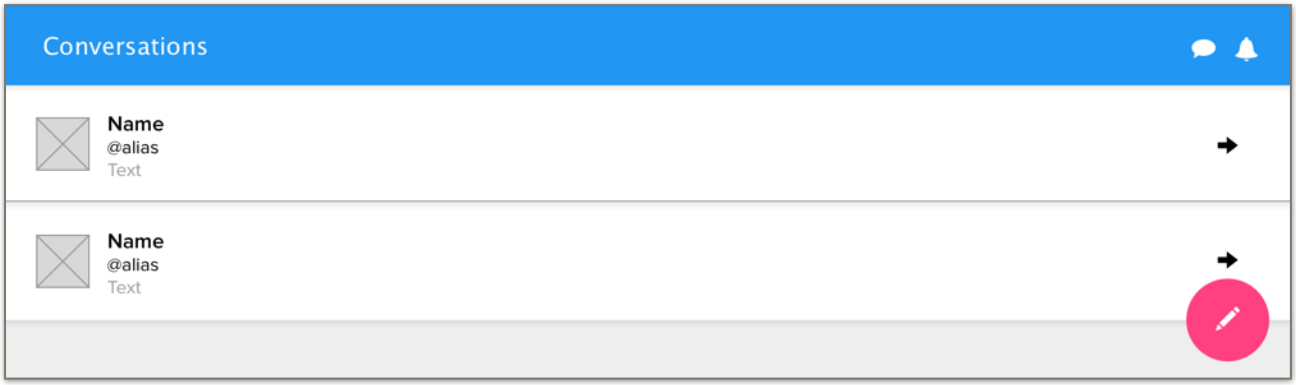

Figura 15. Boceto de la vista de conversaciones.

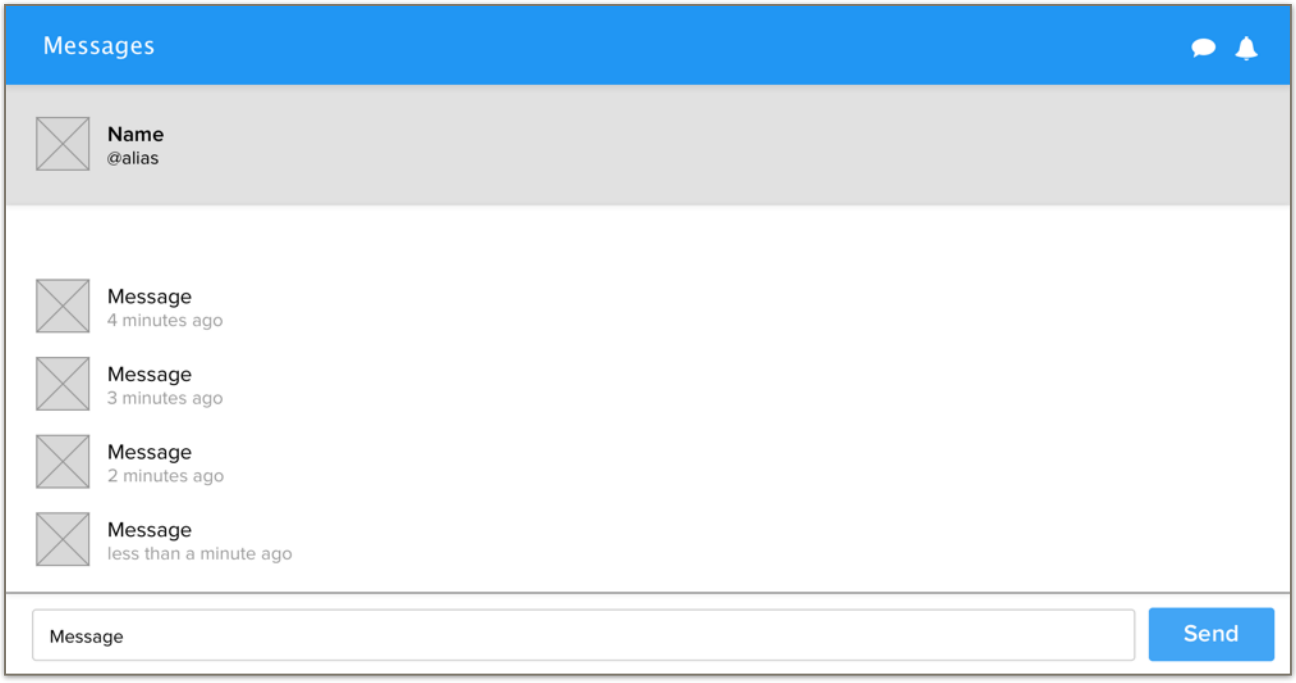

Figura 16. Boceto de la vista de mensajes.

La vista de notificaciones muestra las notificaciones que ha recibido un usuario y permite navegar desde ella hasta la vista que requiera la atención del usuario, si procede.

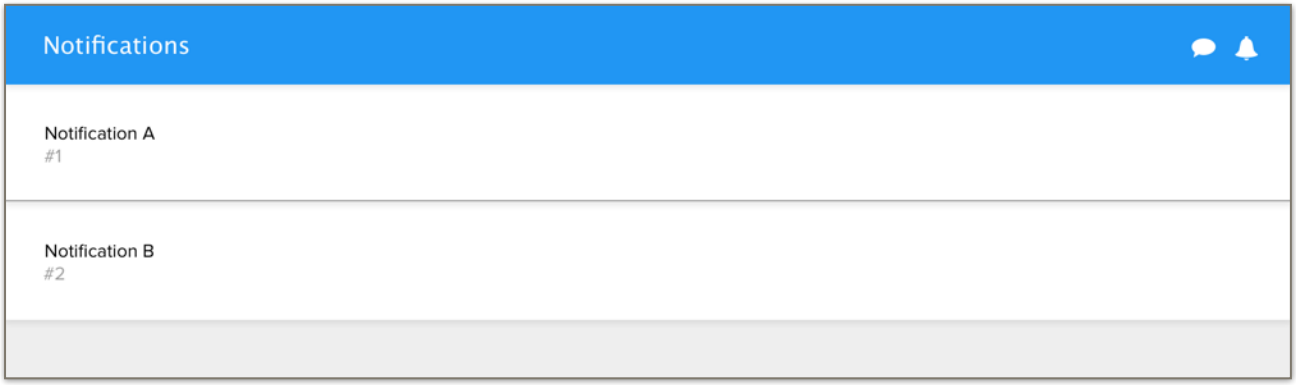

Figura 17. Boceto de la vista de notificaciones.

La vista de búsqueda es una de las más importantes de la aplicación, ya que permite buscar alojamientos y ofertas.

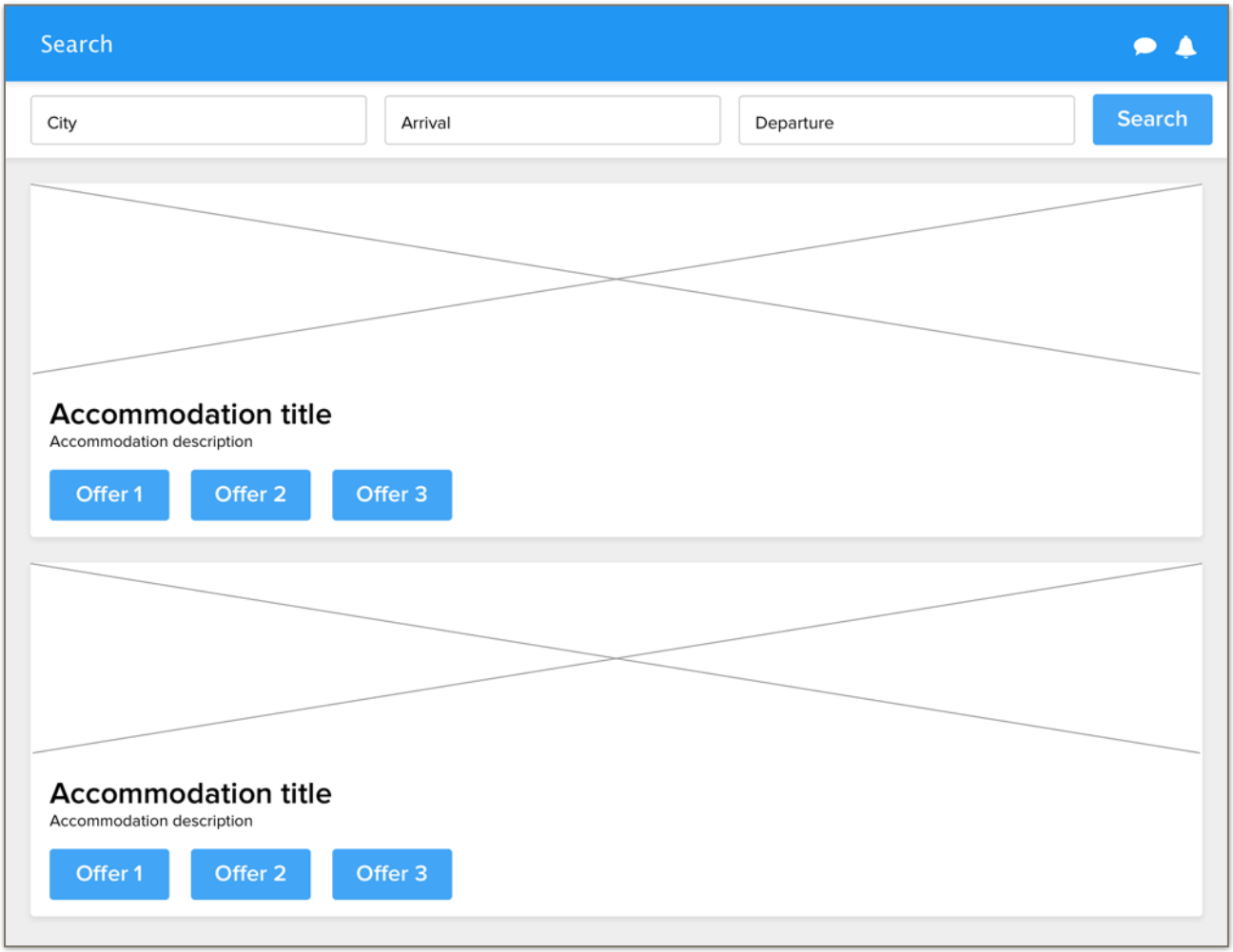

Figura 18. Boceto de la vista de búsqueda.

Desde la vista de búsqueda se podrá acceder a un alojamiento en concreto, cuyo boceto se muestra a continuación.

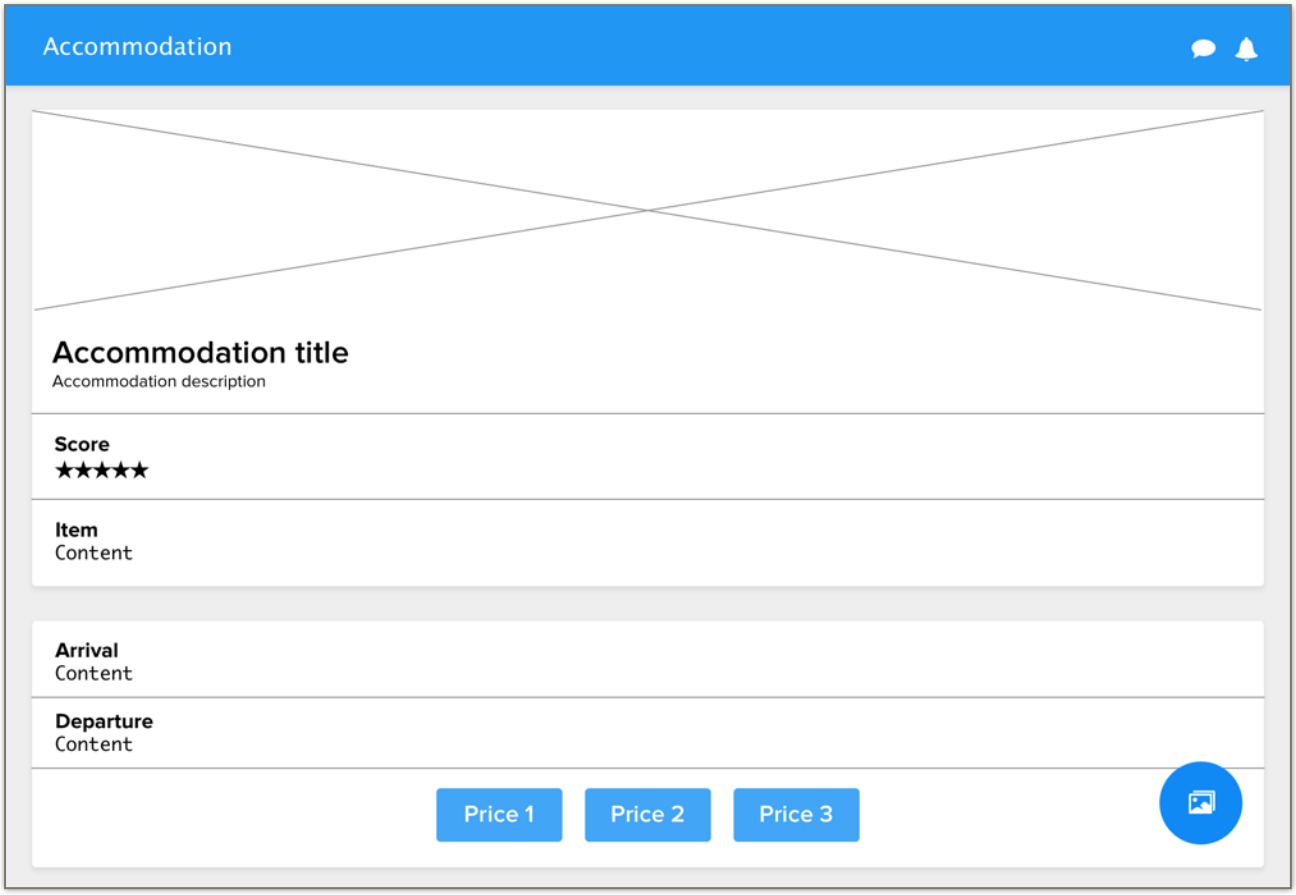

Figura 19. Boceto de la vista de alojamiento.

Por otra parte, la vista de perfil muestra la información pública de un usuario.

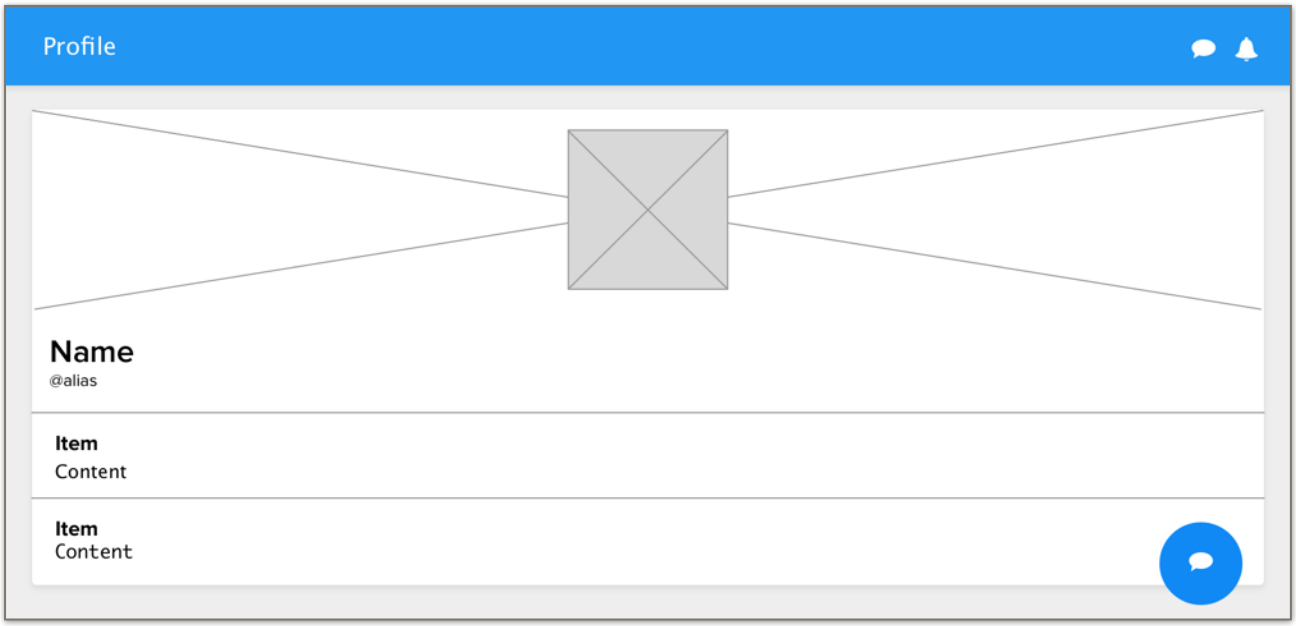

Figura 20. Boceto de la vista de perfil.

La vista de alojamientos de una cuenta muestra los alojamientos propiedad de la misma. Además, permite abrir la ventana de diálogo de creación de nuevos alojamientos.

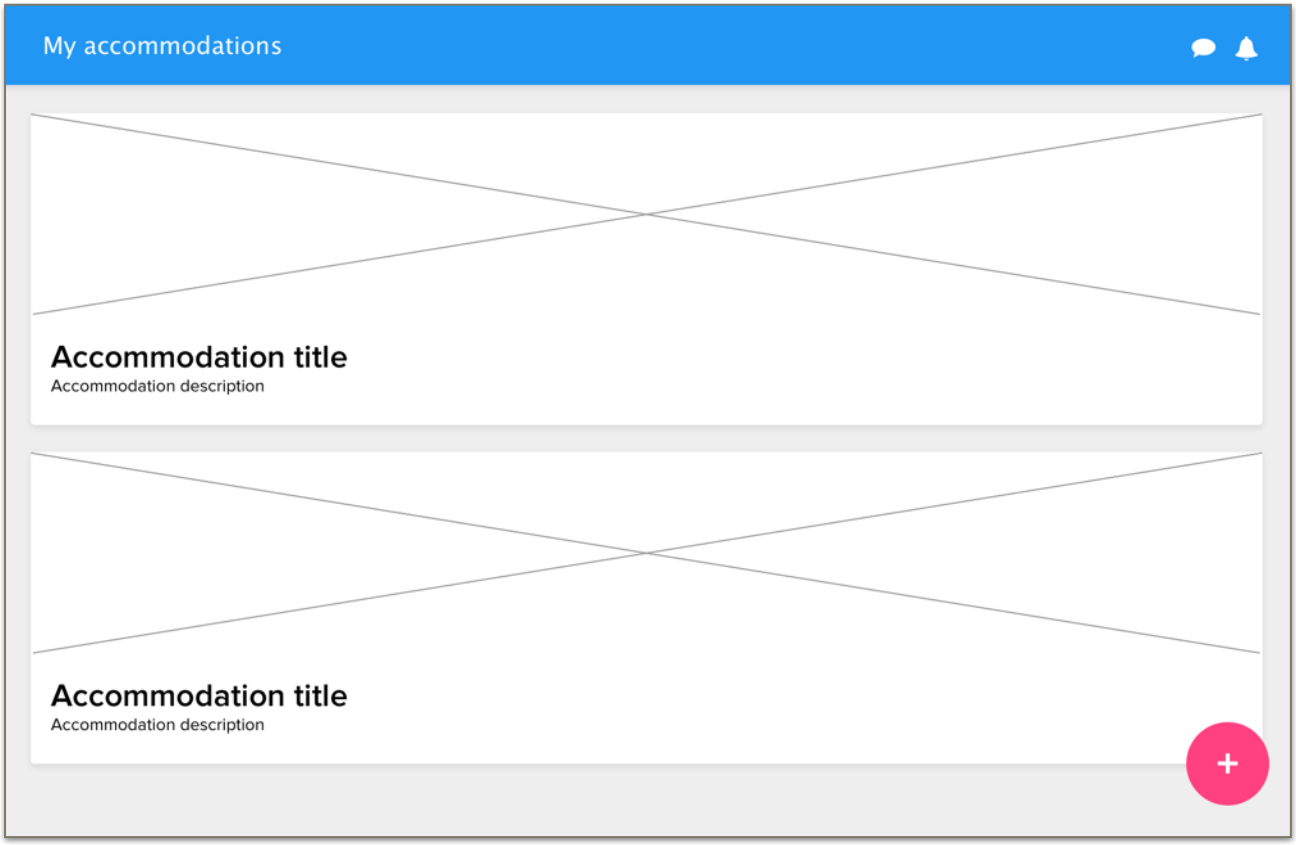

Figura 21. Boceto de la vista de alojamientos de una cuenta.

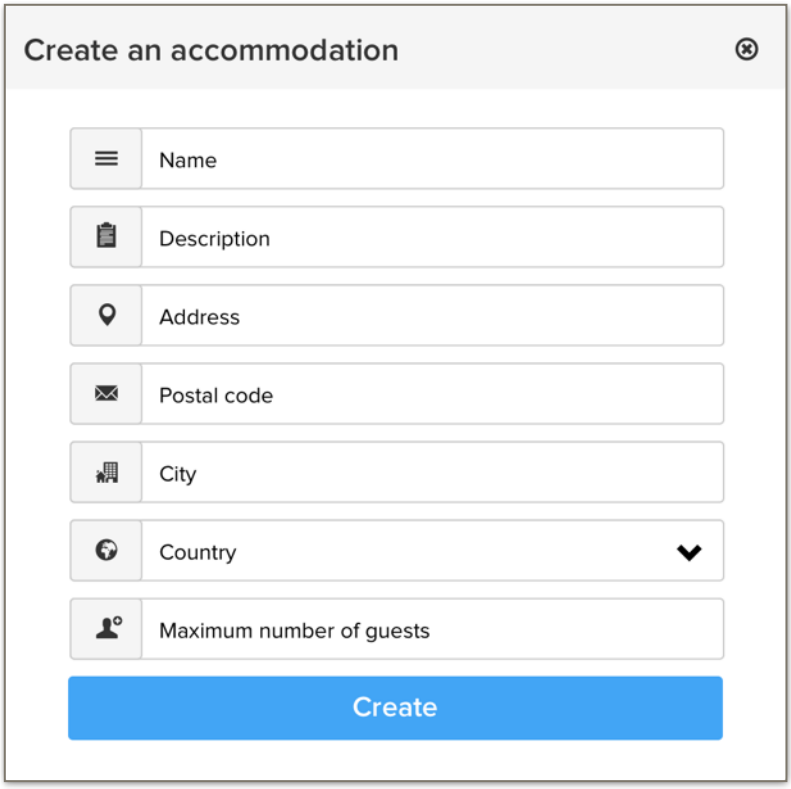

Figura 22. Boceto de la vista de creación de alojamiento.

La vista de edición de alojamiento se divide en tres pestañas: información personal, fotos y ofertas.

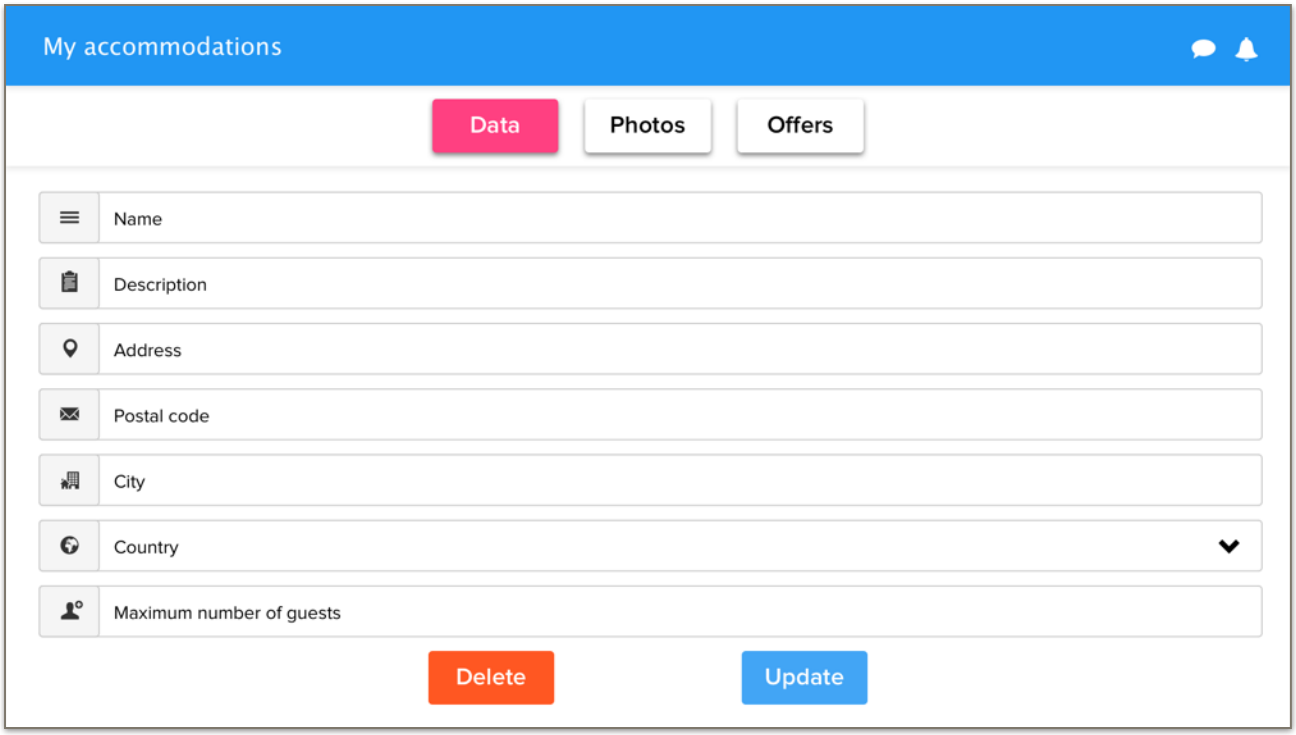

Figura 23. Boceto de la vista de edición de alojamiento (pestaña de información).

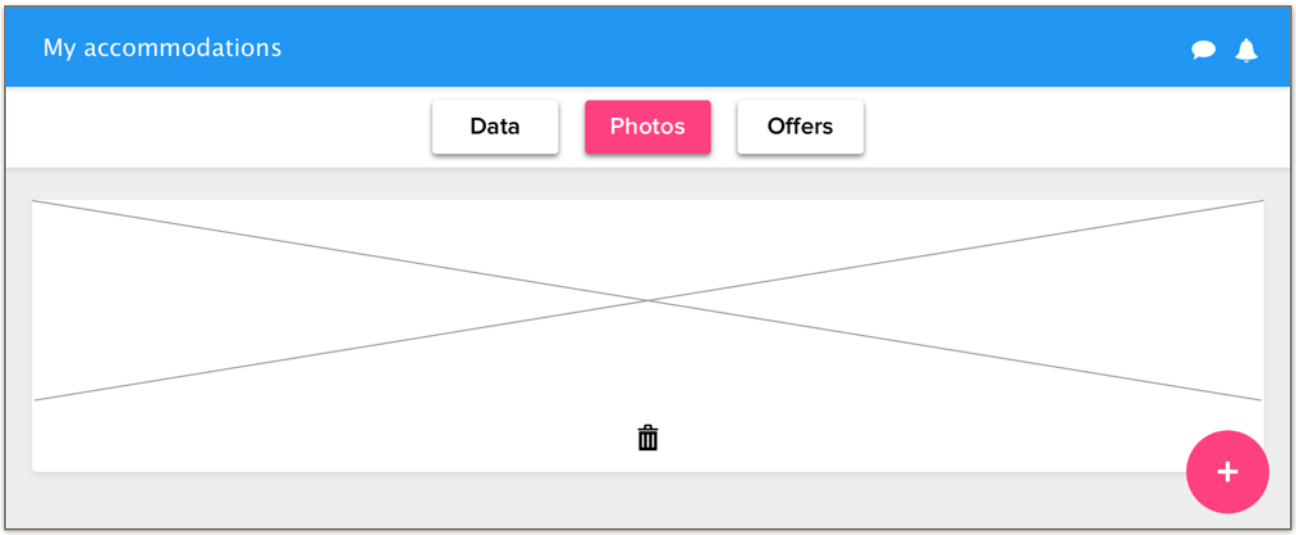

Figura 24. Boceto de la vista de edición de alojamiento (pestaña de fotos).

Figura 25. Boceto de la vista de adición de foto.

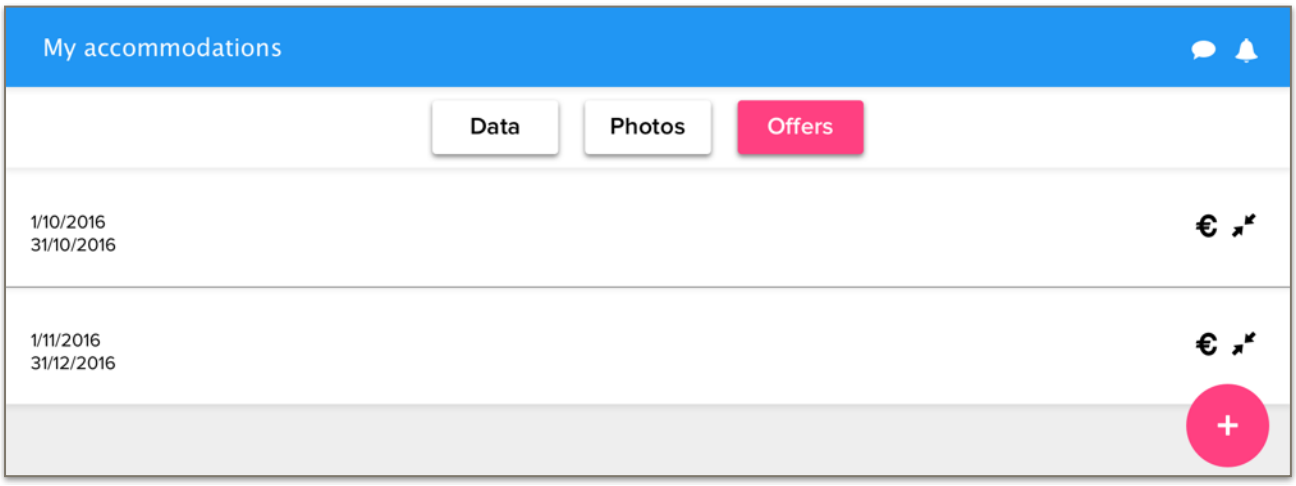

Figura 26. Boceto de la vista de edición de alojamiento (pestaña de ofertas).

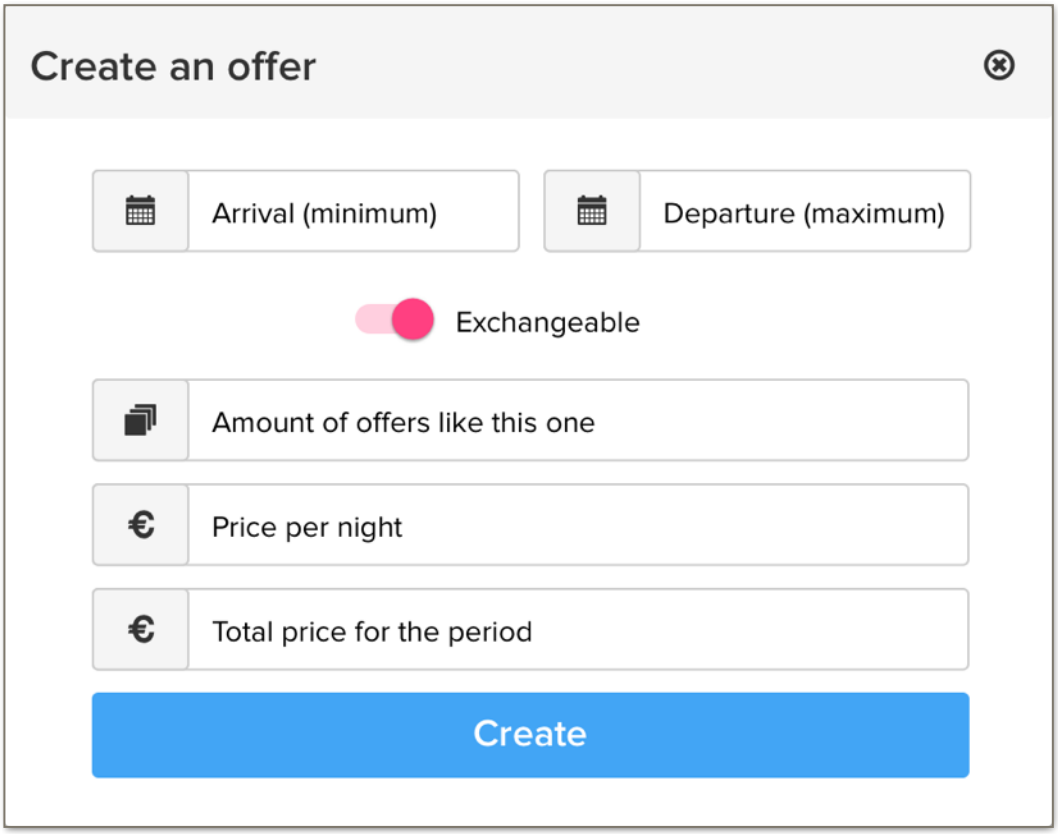

Figura 27. Boceto de la vista de creación de oferta.

En cuanto a las solicitudes, la vista se divide en dos pestañas, lo que permite clasificarlas en enviadas y recibidas. La pestaña de solicitudes enviadas permite ver el estado de todas las solicitudes que un usuario ha enviado. Además, permite emitir una valoración en caso de haber sido aceptadas. Por otra parte, la pestaña de solicitudes recibidas permite aceptar y rechazas las solicitudes que otros usuarios han enviado al usuario que la visualiza.

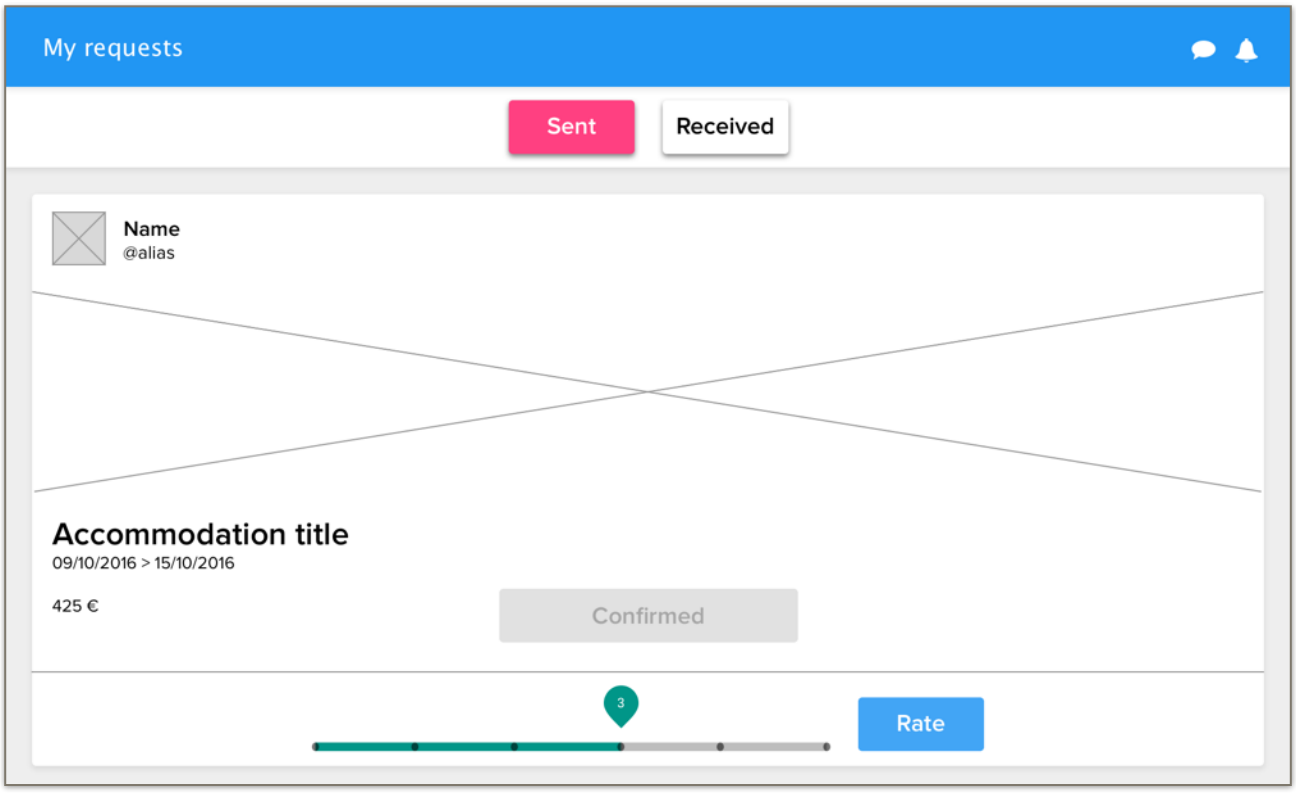

Figura 28. Boceto de la vista de solicitudes (pestaña de enviadas).

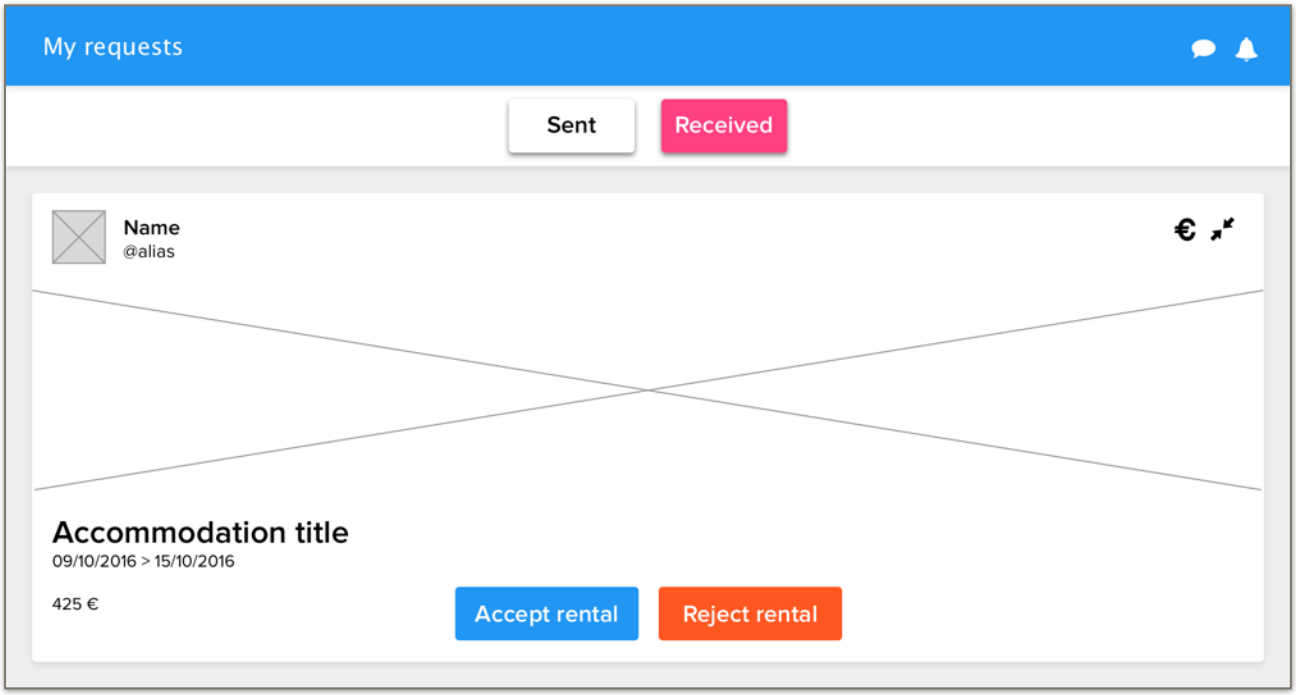

Figura 29. Boceto de la vista de solicitudes (pestaña de recibidas).

Por último, la vista de ajustes permite modificar toda la información personal de una cuenta. Se divide en cuatro pestañas: datos, contraseña, avatar y cabecera.

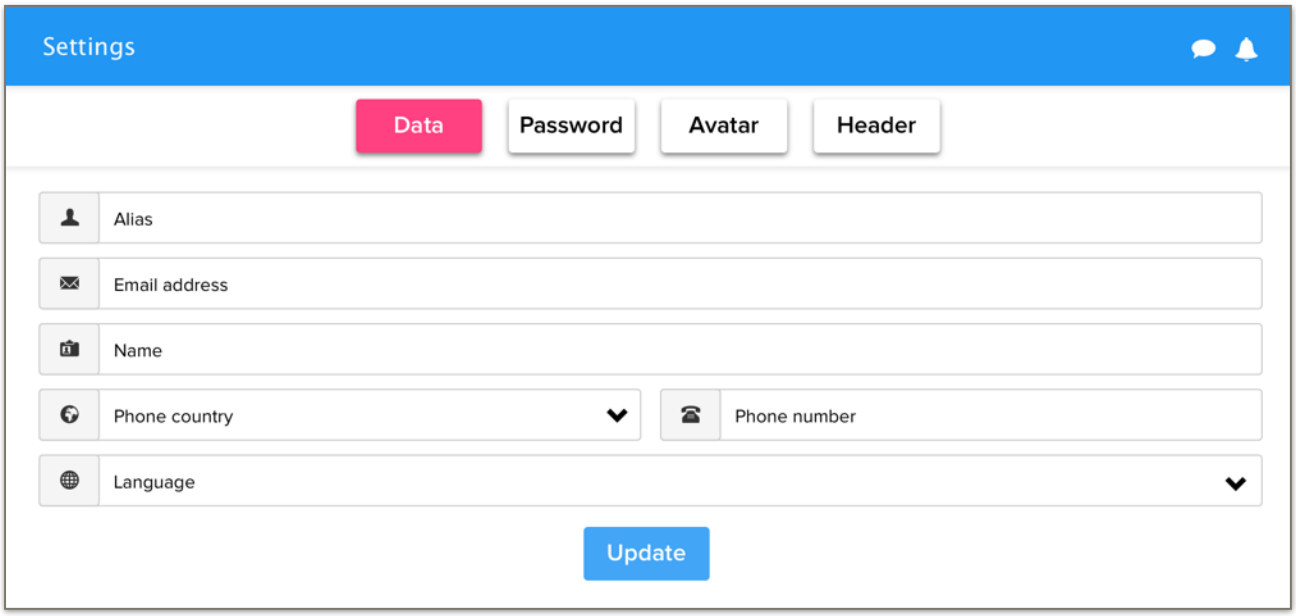

Figura 30. Boceto de la vista de ajustes (pestaña de datos).

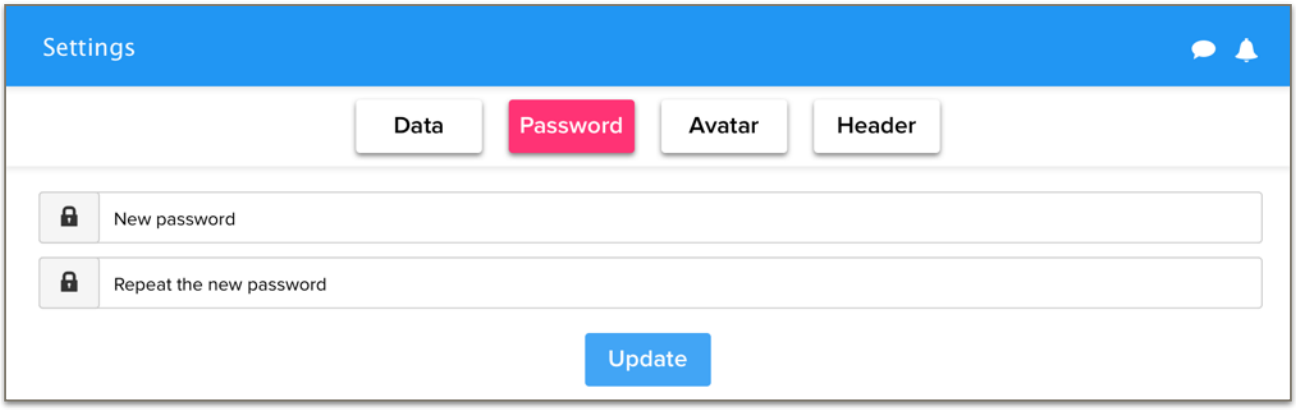

Figura 31. Boceto de la vista de ajustes (pestaña de contraseña).

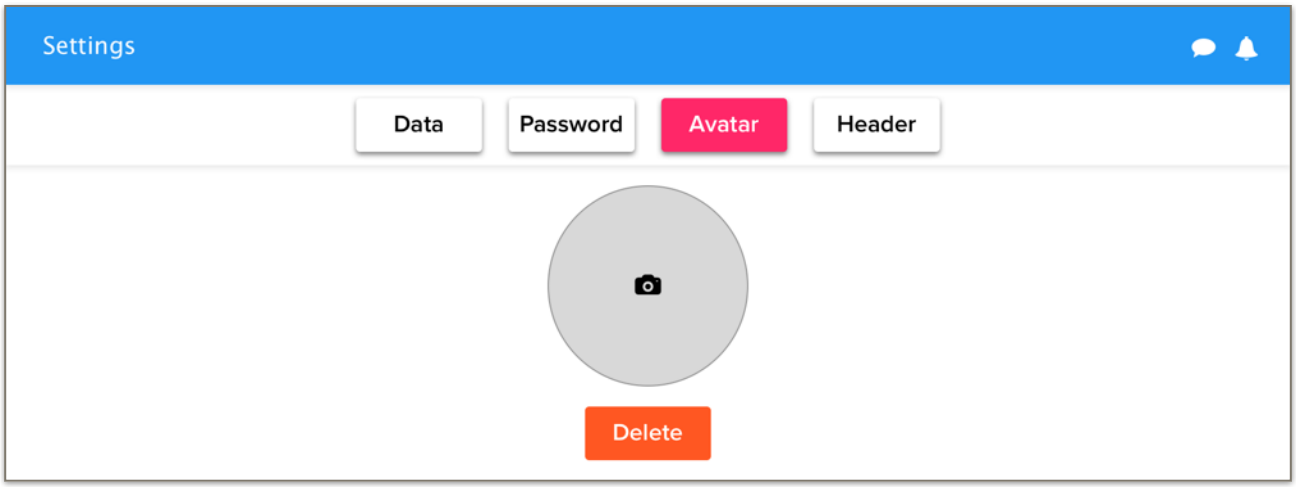

Figura 32. Boceto de la vista de ajustes (pestaña de avatar).

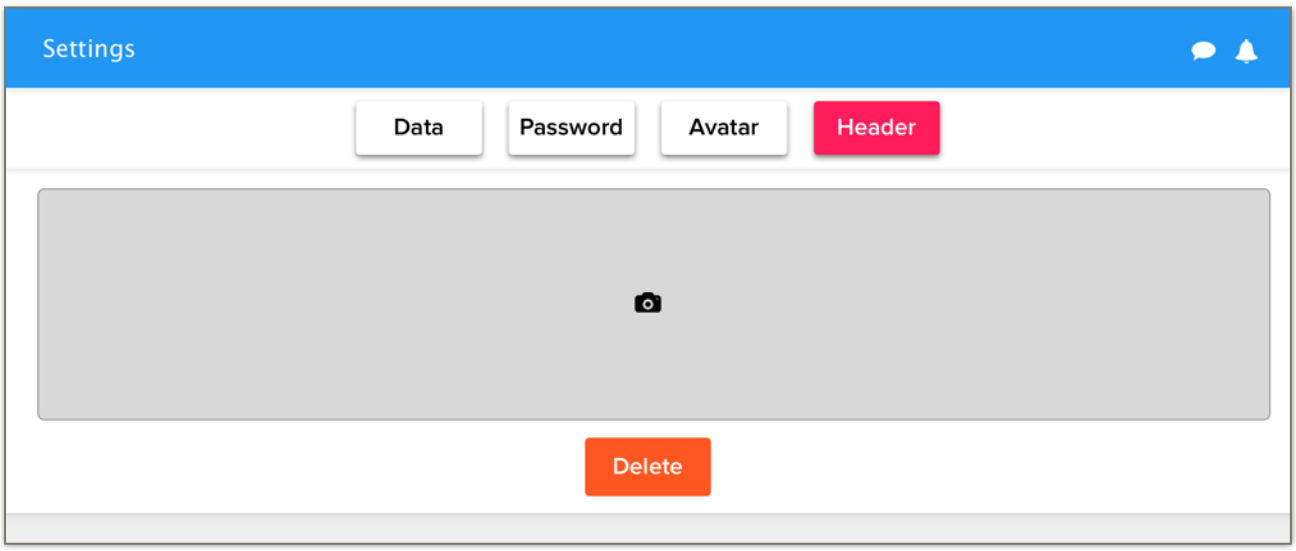

Figura 33. Boceto de la vista de ajustes (pestaña de cabecera).

6. Implementación

El objetivo de la fase de implementación es crear la aplicación, ya sea completamente desde cero o reutilizando otros componentes (bibliotecas). Esta fase se lleva a cabo en base a los documentos de Requisitos (fase de análisis) y de arquitectura (fase de diseño).

6.1. Tecnologías empleadas

Las tecnologías implicadas en la fase de implementación del proyecto son los lenguajes de programación que servirán para crear tanto el *back-end* como el *front-end*, además de los *frameworks* y las bibliotecas que vayan a utilizarse junto con el código propio de la aplicación.

6.1.1. PHP

Como ya se ha indicado, para la implementación del *back-end* se ha escogido el lenguaje de programación PHP. Se trata de un lenguaje interpretado multiparadigma: imperativo, funcional, orientado a objetos, procedural y reflexivo. En este caso se aprovechará mayoritariamente su paradigma orientado a objetos. Sin embargo, también se empleará el paradigma imperativo, ya que el punto de entrada de ejecución de la aplicación es una sucesión de *scripts*.

6.1.2. *Frameworks* para PHP

Un *framework* es un entorno de trabajo que ofrece las herramientas necesarias para desarrollar una aplicación. Es posible desarrollar software sin este tipo de dependencias. Sin embargo, estos entornos ofrecen una capa de abstracción adicional sobre el lenguaje de programación que permite centrarse en implementar funcionalidad sin tener que lidiar con las dificultades a bajo nivel del lenguaje. Esto permite reducir el tiempo de desarrollo a la vez que se obtiene un producto final con una estructura robusta que permite su mantenimiento y mejora.

Existen diversos *frameworks* muy utilizados, como Laravel, Symfony, CodeIgniter y Yii. En este caso se ha escogido el *micro-framework* Lumen, que es una versión ligera y rápida de Laravel específicamente diseñada para el desarrollo de *RESTful* APIs.

Lumen permite definir rutas que son gestionadas automáticamente. También posibilita definir *middleware*, que es un tipo de componente intermedio que se ejecuta antes y/o después de una petición HTTP para modificarla. En este caso se utiliza para autenticación, para limitar la cantidad de peticiones por unidad de tiempo y para gestionar el mecanismo de seguridad Cross-Origin Resource Sharing (CORS), el cual es obligatorio en las peticiones de tipo *XMLHttpRequest*.

La siguiente figura muestra un fragmento de código real como ejemplo. Se trata del método `create()` del controlador `AccommodationController`, el cual gestiona la creación de alojamientos nuevos.

```php
/**
 * Create a new accommodation in the database.
 *
 * @access public
 * @param \Illuminate\Http\Request $request Instance of the user request.
 * @return \Illuminate\Http\JsonResponse
 */
public function create(HttpRequest $request) {
	// Validate
	$attributes = $this->validateCreationRules($request);

	// Create and fill the item
	$item = Accommodation::firstOrNew([
		'account_id' => $request->user()->id
	] + $request->only($attributes));

	// Check if exists a duplicated entry
	if($item->exists)
		return $this->get($request, $item->id);

	// Save the item otherwise
	$item->save();

	// Reply
	return response()->json($item->toArray(), 201)->header('Location',
'/v1/accommodations/'.$item->id);
}
```

Figura 34. Fragmento de un controlador del *back-end*.

6.1.3. Bibliotecas para PHP

Mediante el gestor de paquetes Composer, se han utilizado un total de seis bibliotecas para satisfacer diferentes necesidades que han ido surgiendo durante la implementación.

PHP dotenv se ocupa de cargar como variables de entorno las variables definidas en el fichero de configuración, cuyo nombre es `.env`. De esta forma es posible definir parámetros específicos como la información de conexión a la base de datos en un fichero independiente del proyecto que se carga de forma rápida en la ejecución.

Como ya se ha indicado anteriormente, era necesario que el *back-end* tratase correctamente las peticiones CORS. Para ello se ha utilizado el *middleware* *cors-illuminate*, preparado específicamente para ser usado con Lumen. Permite la personalización de todos los parámetros que define la especificación CORS.

Por otra parte, existe información que se almacena en binario (como los *tokens* de sesión) que es necesario transportar fuera de la base de datos. Para ello se ha utilizado la biblioteca Base62, que codifica y decodifica datos en crudo y en base 62.

Para disponer de protección contra ataques se hace uso de la prueba de Turing *reCAPTCHA* y su biblioteca para PHP.

Por último, en los inicios de sesión se obtiene el sistema operativo y navegador del usuario además de su ubicación geográfica aproximada. Para ello se utilizan las bibliotecas Browscap y GeoIP.

6.1.4. HTML

HTML (HyperText Markup Language) es el lenguaje de marcado que se utiliza en la World Wide Web. A partir de su versión 5 se utiliza exclusivamente para definir la semántica de los documentos web, quedando cualquier aspecto visual a cargo de las hojas de estilo.

En el ámbito del proyecto este lenguaje se ha utilizado para definir las vistas del *front-end* de la aplicación. Téngase en cuenta que al utilizar la biblioteca Angular Material (de la que se hablará más adelante) existen etiquetas personalizadas que pueden no respetar el estándar HTML 5. Esto ocurre porque las vistas son compiladas por dicha biblioteca en tiempo de ejecución y son transformadas a código HTML 5 válido antes de ser insertadas en el Document Object Model (DOM).

6.1.5. CSS

Para definir el aspecto visual se ha hecho uso de las hojas de estilo en cascada (CSS). Al emplear la biblioteca Angular Material únicamente ha sido necesario definir algunos estilos básicos y adaptaciones, por lo que no se hace un uso amplio de esta tecnología.

6.1.6. JavaScript

En la parte del *front-end* se utiliza el lenguaje interpretado JavaScript, estandarizado por la especificación ECMAScript. Se trata, al igual que PHP, de un lenguaje multiparadigma: funcional, imperativo y basado en prototipos (en lugar de clases).

Prácticamente la totalidad de navegadores modernos son capaces de interpretar código JavaScript.

6.1.7. *Frameworks* para JavaScript

Existen diversos *frameworks* y bibliotecas JavaScript para el desarrollo de aplicaciones web. Destacan *jQuery*, AngularJS, React y Polymer. En este caso se ha optado por AngularJS, ya que plantea un enfoque que, una vez superada la curva de aprendizaje, agiliza y facilita el desarrollo de aplicaciones web completas. Al igual que Lumen, se basa en una arquitectura MVC.

Angular permite asociar un controlador y una plantilla a cada ruta, lo que define una vista. También ofrece herramientas para organizar la aplicación en diferentes componentes, como pueden ser servicios, factorías, directivas o filtros.

La siguiente figura muestra un fragmento de código real del controlador `HomeController`, el cual gestiona la vista de inicio. Se encarga de asignar un valor aleatorio a una variable que es utilizado para escoger la imagen de fondo de la vista.

```
1  angular.module('app').controller('HomeController', ['$scope',
   function($scope) {
2        $scope.background = Math.floor(Math.random() * 4) + 1;
3  }]);
```

Figura 35. Fragmento de un controlador del *front-end*.

6.1.8. Bibliotecas para JavaScript

Se han utilizado los gestores de paquetes Bower, para gestionar las dependencias de la aplicación web, y NPM, para las herramientas de desarrollo.

Además del propio AngularJS se han incluido sus siguientes módulos:

- **Animate.** Anima ciertos comportamientos de la interfaz de forma automática.
- **ARIA.** Proporciona soporte para los atributos de la especificación WAI-ARIA más comunes.
- **Cookies.** Da soporte a cookies. A pesar de que no se vaya a hacer uso de ellas, algunas dependencias de la aplicación requieren que se incluya para evitar problemas de compatibilidad con navegadores antiguos.
- **Messages.** Automatiza en tiempo real la realimentación de errores de validación en formularios.
- **Route.** Gestiona las rutas de la aplicación y las hace compatibles con la API History de HTML 5.

En cuanto a las bibliotecas de terceros, se han utilizado las que siguen:

- **Angular Material.** Es un *framework* de componentes de interfaz que sigue el lenguaje visual Material Design. Se ha utilizado para desarrollar la interfaz de toda la aplicación.
- **Moment.** Ofrece funcionalidades adicionales para el tratamiento de fechas. Además soporta internacionalización y localización para las mismas. Se utiliza para mostrar fechas, concretamente fechas relativas que facilitan la lectura al usuario.
- **PhotoSwipe.** Es una biblioteca que genera una galería de imágenes de forma automática. Se utiliza para mostrar las imágenes asociadas a un alojamiento en la vista de alojamientos.
- ***angular-load*.** Posibilita la carga dinámica de *scripts* y hojas de estilo. Se utiliza para cargar las dependencias de la librería *reCAPTCHA* únicamente cuando se va a utilizar.
- ***angular-local-storage.*** Sirve de interfaz para utilizar la API Web Storage del navegador, haciendo su uso más sencillo y facilitando el control de errores.
- ***angular-recaptcha*.** Permite implementar fácilmente la parte del cliente para poder hacer uso de la API de *reCAPTCHA*.
- ***angular-scroll*.** Ofrece una API para gestionar el *scroll* del navegador con animaciones automáticas. Se utiliza para poder volver a la parte superior de un *scroll* infinito.
- ***angular-translate*.** Posibilita la internacionalización de una aplicación Angular.
- ***angular-loading-bar*.** Muestra automáticamente una barra de progreso cuando hay contenido cargándose.
- ***ng-file-upload*.** Permite enviar archivos a una *RESTful API*. Automatiza las llamadas.

En cuanto a las dependencias de desarrollo se ha hecho uso de Gulp, una herramienta que permite automatizar flujos de trabajo. Se han utilizado diversos *plugins*, para juntar, minimizar y postprocesar los archivos del proyecto, quedando así preparados para ser utilizados en un entorno de producción.

6.2. Plataformas de desarrollo

Para la implementación de código fuente, tanto del *front-end* como del *back-end*, se ha hecho uso de la aplicación PhpStorm. Proporciona asistencia en tiempo real para autocompletado, prevención de errores e incluso buenas prácticas. Está integrado con Git y trabaja con multitud de lenguajes de desarrollo web, incluyendo PHP, HTML, CSS y JavaScript.

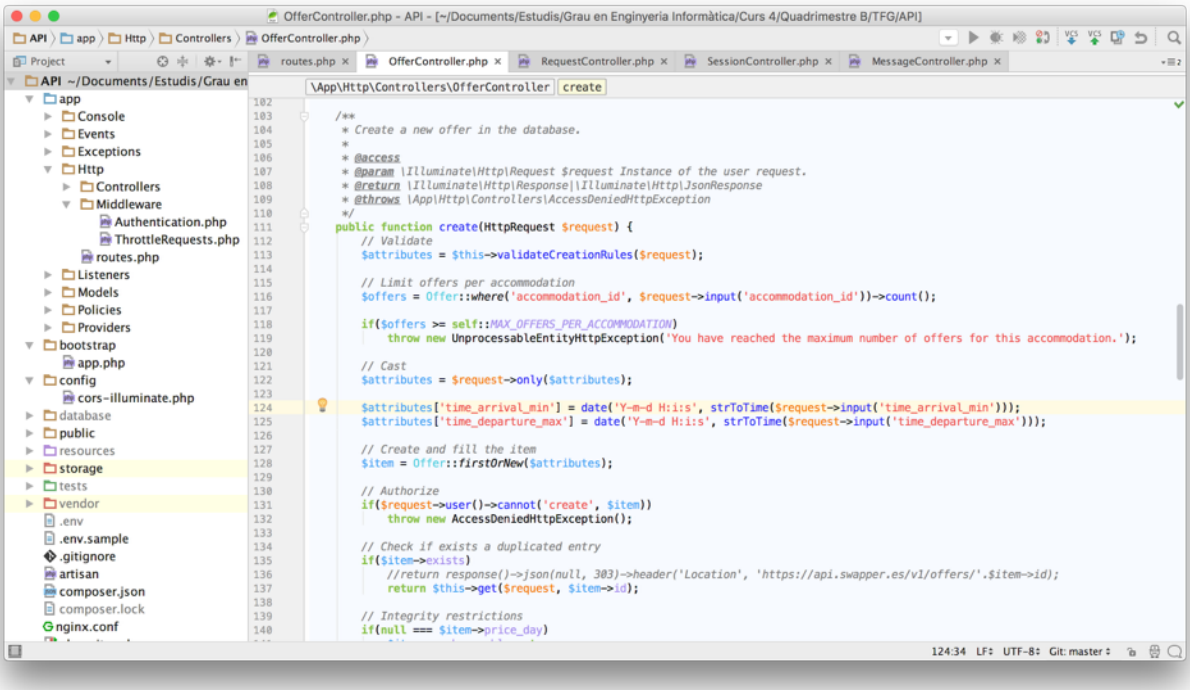

Figura 36. Captura de pantalla de PhpStorm.

En cuanto al modelado de la base de datos se ha utilizado la aplicación MySQL Workbench. Permite crear modelos completos que se pueden exportar directamente a un sistema de gestión de bases de datos compatible con MySQL, como es el caso de MariaDB.

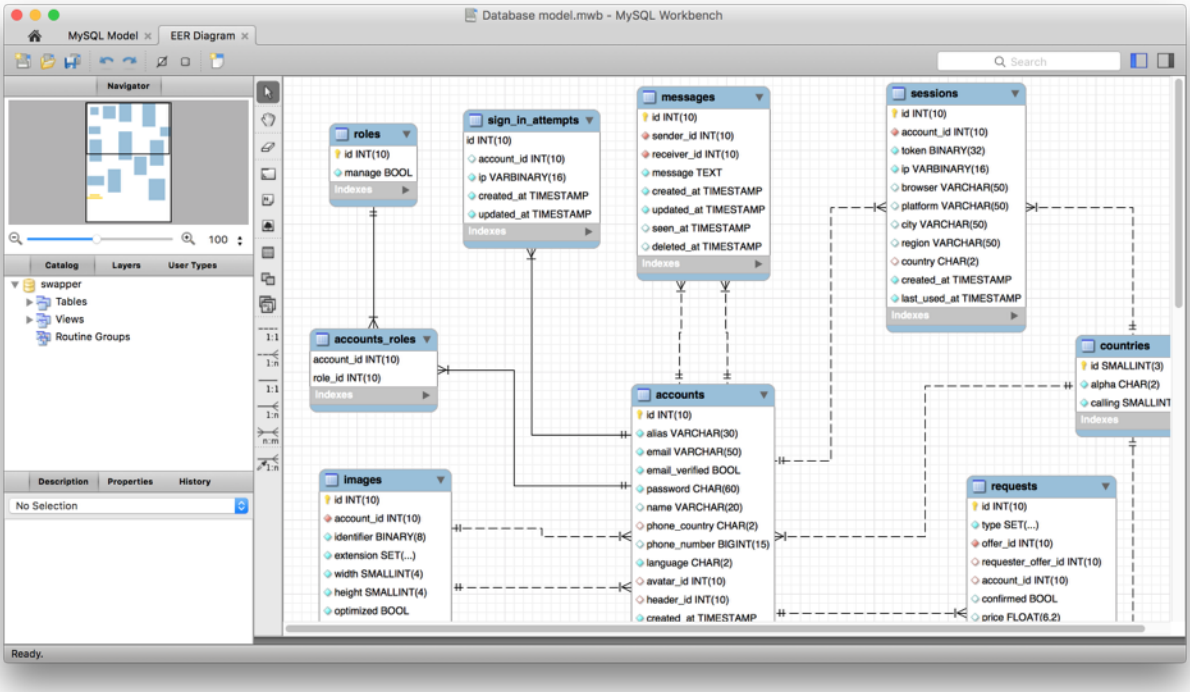

Figura 37. Captura de pantalla de MySQL Workbench.

Durante el desarrollo también se ha hecho uso de la aplicación Postman, que permite preparar peticiones a APIs para realizar pruebas.

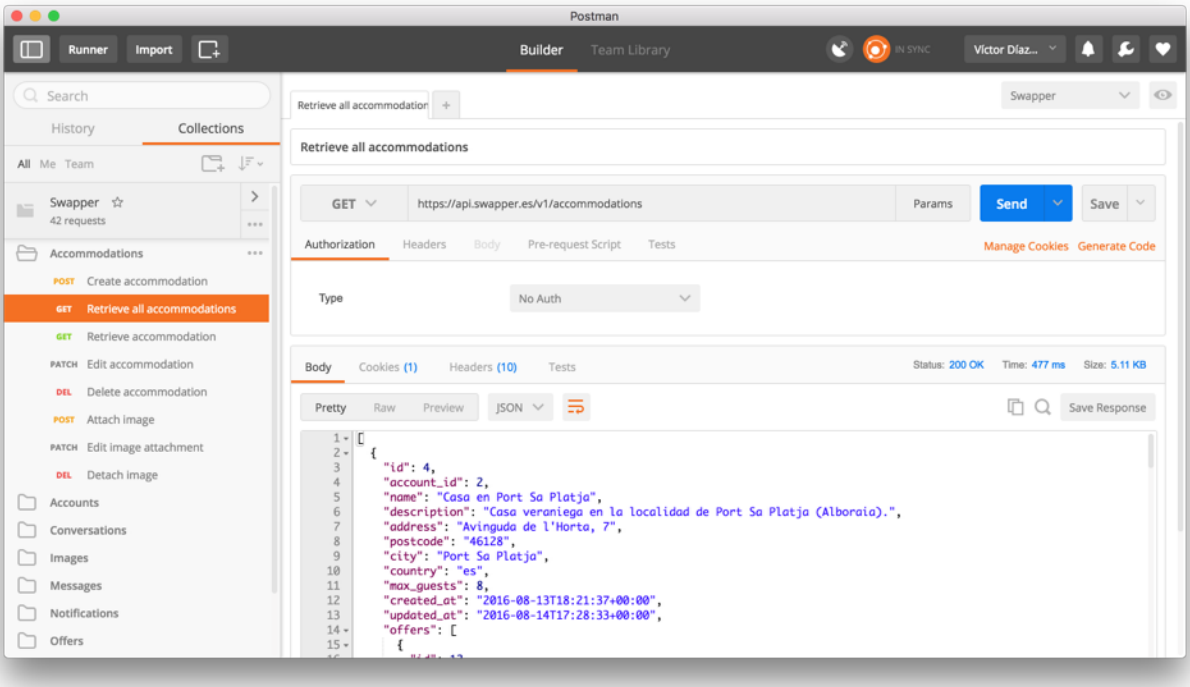

Figura 38. Captura de pantalla de Postman.

6.3. Control de versiones

Durante toda la fase de implementación se ha hecho uso del sistema de control de versiones Git. Se han creado tres repositorios: Web (para el *front-end*), API (para el *back-end*) y Database (para el modelo de la base de datos).

Por cuestiones de simplicidad se ha trabajado siempre en la rama *master* de todos los repositorios. En un contexto de trabajo en equipo se debe crear una rama por cada característica, la cual termina siendo unida a la rama *master* mediante la petición de una *pull request* bajo la supervisión del integrador de código. En este caso se ha encargado únicamente una persona de la implementación, por lo que no se ha considerado necesario seguir todo este proceso.

Además de Git, se ha utilizado la herramienta GitLab, que permite tener repositorios gratuitos en la nube, tanto para individuos como para equipos.

6.4. Estructura de directorios

Por último se describe la estructura de directorios y ficheros de ambos subproyectos.

6.4.1. *Back-end*

Directorio	Descripción
`/`	Es el directorio raíz del proyecto. Todo lo necesario está dentro de él, ya que el proyecto es autocontenido. No se depende de ningún directorio externo ni durante el desarrollo ni después del despliegue.
`/app`	Este directorio es el equivalente al *namespace* App de PHP. Todo componente del paradigma orientado a objetos de la aplicación se encuentra bajo este directorio.
`/app/Console`	Incluye todos los ficheros necesarios para la ejecución de los comandos Artisan definidos por la aplicación. Artisan es una interfaz de la línea de comandos que proporciona Lumen.
`/app/Events`	Contiene todos los eventos que se pueden generar en cualquier parte de la aplicación.
`/app/Exceptions`	Incluye el manejador de excepciones de la aplicación.
`/app/Http`	Contiene los controladores, *middleware* y rutas.
`/app/Http/Controllers`	Incluye todos los controladores de la aplicación.
`/app/Http/Middleware`	Contiene todo el *middleware* de la aplicación.
`/app/Listeners`	Incluye todos los oyentes que correspondan para cada evento definido.
`/app/Models`	Contiene los modelos de la aplicación, definidos en a partir del modelo base de Eloquent.
`/app/Policies`	Incluye todas las políticas de autorización de la aplicación. Cada política define las características de autorización para un modelo.

Directorio	Descripción
`/app/Providers`	Contiene todos los proveedores. Un proveedor es un *singleton* que proporciona un servicio concreto a la aplicación. Por ejemplo, existe un proveedor de servicio de autenticación.
`/bootstrap`	Incluye los *scripts* de arranque, el punto de entrada de la aplicación.
`/config`	Contiene ficheros de configuración de los que puedan depender algunas bibliotecas.
`/public`	Es el directorio público que expondrá el servidor web. Contiene un único archivo `index.php` que llama directamente a los *scripts* de arranque.
`/storage`	Incluye ficheros de caché y registros.

Tabla 29. Estructura de directorios del *back-end*.

6.4.2. *Front-end*

Directorio	Descripción
`/`	Es el directorio raíz del proyecto. Ocurre exactamente igual que con el directorio raíz del *back-end*.
`/app`	Contiene todos los componentes de la aplicación.
`/app/images`	Incluye todas las imágenes.
`/app/languages`	Contiene los ficheros de idioma con las traducciones para cada cadena de texto.
`/app/source`	Incluye todo el código JavaScript.
`/app/source/controllers`	Contiene todos los controladores.
`/app/source/directives`	Incluye todas las directivas.
`/app/source/factories`	Contiene todas las factorías.
`/app/source/filters`	Incluye todos los filtros.
`/app/styles`	Contiene todos los estilos CSS.
`/app/templates`	Incluye todas las plantillas HTML.
`/public`	Es el directorio público que expondrá el servidor web. Como se ha indicado previamente, mediante Gulp se definen tareas que permiten exportar a este directorio la aplicación ya construida.

Tabla 30. Estructura de directorios del *front-end*.

7. Despliegue

La fase de despliegue consiste en hacer que la aplicación ya desarrollada esté disponible para su uso por parte de los usuarios. Esta fase no forma parte del proceso de ingeniería de software, pero es fundamental para poder emprender la fase de pruebas.

7.1. Tecnologías utilizadas

Se ha dispuesto un servidor público que ha sido alquilado al proveedor de servicios OVH. Dicho servidor cuenta con una interfaz de red conectada directamente a Internet que tiene asignadas una dirección IPv4 y otra IPv6.

7.1.1. Sistema operativo

Como sistema operativo ha sido escogido CentOS 7.2. Esta distribución de Linux es la versión libre de Red Hat Enterprise Linux y está pensada para ser utilizada en entornos de producción con una alta exigencia de rendimiento. Se trata de una distribución muy estable y con un largo soporte. Por ejemplo, la última versión tiene un soporte de diez años desde la fecha de su lanzamiento. Desde 2010 es la distribución más utilizada para servidores web.

CentOS incluye el gestor de paquetes Yum, que permite instalar todo el software que se va a necesitar. Aunque los repositorios oficiales de CentOS no ofrecen las versiones más recientes de algunas dependencias, esto puede ser solventado mediante el uso de repositorios externos como EPEL o Remi. Incluso Nginx y MariaDB ofrecen repositorios para diversas distribuciones de Linux.

7.1.2. Servidor web

Los servidores web más utilizados en la actualidad son Apache y Nginx. A la hora de servir contenido estático Nginx es unas 2,5 veces más rápido que Apache, consumiendo además menos recursos. En cuanto a contenidos dinámicos, ofrecen prácticamente el mismo rendimiento. La ventaja de Nginx en este aspecto es la mejor gestión de los recursos.[5]

Por esta razón se ha escogido Nginx como servidor web.

7.1.3. Sistema de gestión de bases de datos

Como sistema de gestión de bases de datos se utilizará MariaDB. Tal y como se ha explicado anteriormente, es una bifurcación de MySQL y ofrece mejores resultados de rendimiento que este último.

7.2. Instalación de servicios

Antes de proceder a la configuración de los servicios es necesario instalar los que no vienen por defecto con la versión mínima de CentOS. Se ha aprovechado para instalar algunas utilidades necesarias para la administración del sistema.

Mediante la siguiente instrucción se han instalado el editor visual Nano y la utilidad Wget, que permite descargar archivos mediante el protocolo HTTP.

```
# yum install nano wget
```

A continuación se han instalado los repositorios EPEL y Remi.

```
# yum install epel-release
```

```
# wget http://rpms.famillecollet.com/enterprise/remi-release-7.rpm
# rpm -Uvh remi-release-7*.rpm
```

Seguidamente se han instalado los otros dos repositorios necesarios: Nginx y MariaDB.

En el primer caso se ha creado el fichero `/etc/yum.repos.d/nginx.repo` con el siguiente contenido:

```
1 [nginx]
2 name=nginx repo
3 baseurl=http://nginx.org/packages/mainline/centos/$releasever/$basearch/
4 gpgcheck=0
5 enabled=1
```

En el segundo caso se ha creado el fichero `MariaDB.repo`, en la misma ubicación que el anterior, con el contenido que sigue:

```
1 [mariadb]
2 name = MariaDB
3 baseurl = http://yum.mariadb.org/10.1/centos7-amd64
4 gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
5 gpgcheck=1
```

Para terminar con los repositorios se ha editado el fichero `remi-php70.repo`, que se encuentra en el mismo directorio que los dos anteriores, y se ha cambiado de `0` a `1` el valor del atributo `enabled` de la sección `[remi-php70]`. De esta forma se habilita el repositorio de PHP 7.

Por último se han instalado todos los servicios necesarios mediante los siguientes comandos.

```
# yum install nginx mariadb-server mariadb-client
```

```
# yum install php php-fpm php-mysql php-pdo php-mbstring php-mcrypt php-gd
```

7.3. Configuración de servicios

Una vez instalados todos los servicios adicionales se ha procedido a configurarlos.

En primer lugar se han configurado algunos servicios para arrancar junto con el sistema:

```
# systemctl enable nginx
# systemctl enable php-fpm
# systemctl enable mariadb
```

Para configurar Nginx, PHP-FPM y MariaDB ha sido necesaria la creación de los archivos que se listan a continuación. En caso de existir se han reemplazado completamente. Su contenido puede ser encontrado en el anexo correspondiente.

- `/etc/nginx/nginx.conf`. Es el fichero de configuración principal de Nginx. Es fundamental configurarlo adecuadamente por cuestiones de seguridad y eficiencia.
- `/etc/nginx/cloudflare.conf`. Permite que Nginx conozca la IP original de las peticiones cuando provienen de la red de distribución de contenidos CloudFlare.
- `/etc/nginx/conf.d/swapper.conf`. Define el servidor virtual para la aplicación web. Como los archivos de configuración de Nginx del *front-end* y *back-end* van incluidos en los propios repositorios, este archivo hace referencia a ellos.
- `/etc/nginx/cors`. Especifica la configuración necesaria para que las rutas de contenidos estáticos respeten el estándar CORS, ya comentado anteriormente.
- `/etc/php-fpm.d/swapper.conf`. Establece una jaula en la que se ejecutará todo el código PHP. De esta forma si un atacante consiguiera inyectar código no sería capaz de hacer nada más allá del directorio establecido.
- `/etc/my.cnf.d/server.cnf`. Modifica la configuración por defecto de MariaDB para que se utilice la codificación UTF8 en todas las comunicaciones.

También ha sido necesario crear el usuario `swapper`, el directorio `/home/swapper` y sus subdirectorios `web`, `api`, `cdn`. y `.tmp`.

Para el correcto funcionamiento de los protocolos HTTPS y HTTP 2 se han ubicado en `/etc/pki/tls` los certificados (firmados por una AC) y claves privadas correspondientes.

Antes de terminar se ha ejecutado el siguiente comando, que convierte en segura la instalación de MariaDB. De esta forma se ha establecido una contraseña de administrador y se han eliminado los datos de ejemplo que vienen con la instalación.

```
# mysql_secure_installation
```

Por último se han arrancado los servicios de Nginx, PHP-FPM y MariaDB.

```
# systemctl start nginx
# systemctl start php-fpm
# systemctl start mariadb
```

7.4. Despliegue de código

Como se ha trabajado con un sistema de control de versiones el despliegue de código es considerablemente sencillo.

Después de haber generado y añadido a GitLab las correspondientes claves SSH de despliegue, se han seguido los pasos que se indican a continuación. Una vez completados estos pasos y modificado el fichero de configuración (ubicado en `/home/swapper/api/.env`) se da por completado el despliegue, con lo que la aplicación queda a disposición de los usuarios.

7.4.1. *Front-end*

```
$ cd /home/swapper/web
$ git clone git@gitlab.com:swapper/web.git .
$ npm install
$ bower install
$ gulp build
$ systemctl reload nginx
```

7.4.2. *Back-end*

```
$ cd /home/swapper/api
$ git clone git@gitlab.com:swapper/api.git .
$ composer install
$ systemctl reload nginx
```

8. Pruebas

El objetivo de la fase de pruebas es comprobar que la aplicación cumple con lo establecido en las fases de especificación de requisitos y análisis, para lo que se realizarán tres tipos de pruebas. Con esta fase se dan por terminados el proceso de ingeniería de software y el propio proyecto.

8.1. Pruebas de usabilidad

La finalidad de las pruebas de usabilidad es medir la facilidad de uso de un producto. Para llevar a cabo dichas pruebas se ha seleccionado a una persona sin conocimientos técnicos en informática y se le ha solicitado que compruebe si es capaz de realizar lo que determina cada caso de uso, uno por uno.

El usuario en cuestión fue capaz de completar todos los casos de uso que establece la especificación, con mayor o menor dificultad. Por tanto, se concluye un resultado satisfactorio.

8.2. Pruebas funcionales

Las pruebas funcionales pretenden comprobar que la aplicación cumple todos los requisitos funcionales que se establecieron previamente a su implementación. Para llevarlas a cabo se han analizado los casos de uso uno por uno y se ha determinado si estos se satisfacen o no.

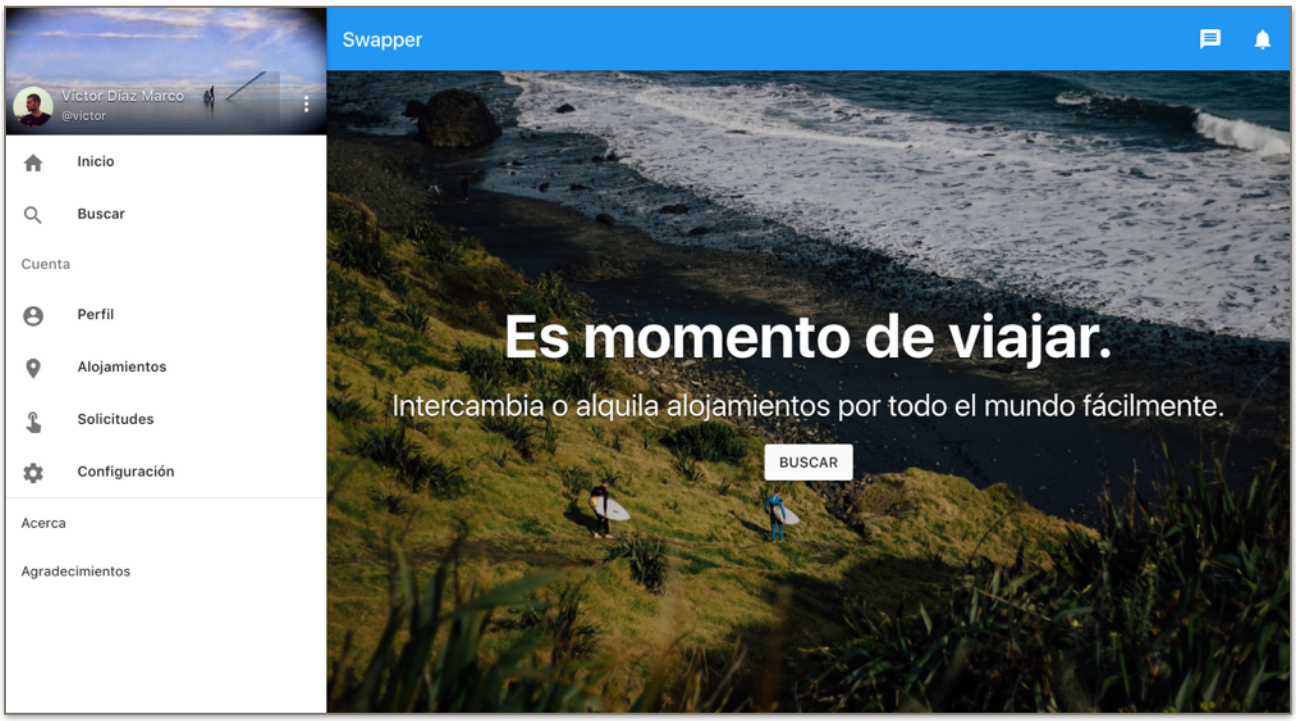

Figura 39. Captura de la vista de inicio.

A continuación se detallan las comprobaciones de los casos de uso más relevantes y los resultados obtenidos.

Primero se ha examinado la vista de búsqueda, relacionada con el caso de uso #7. Para ello se ha introducido una ciudad y las fechas de llegada y salida. El sistema ha devuelto, como cabía esperar, los alojamientos y ofertas que cumplen los requisitos de búsqueda.

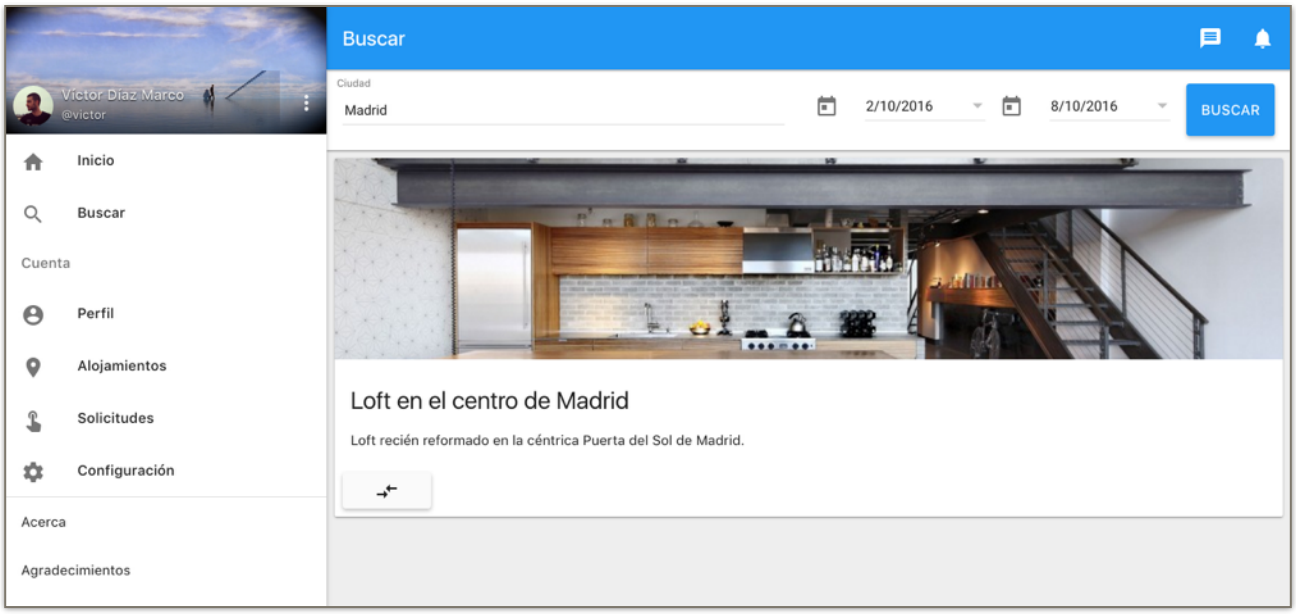

Figura 40. Captura de la vista de búsqueda.

A continuación se ha accedido a la vista de alojamiento (caso de uso #8), que permite consultar toda la información sobre el mismo, además de ver las ofertas disponibles.

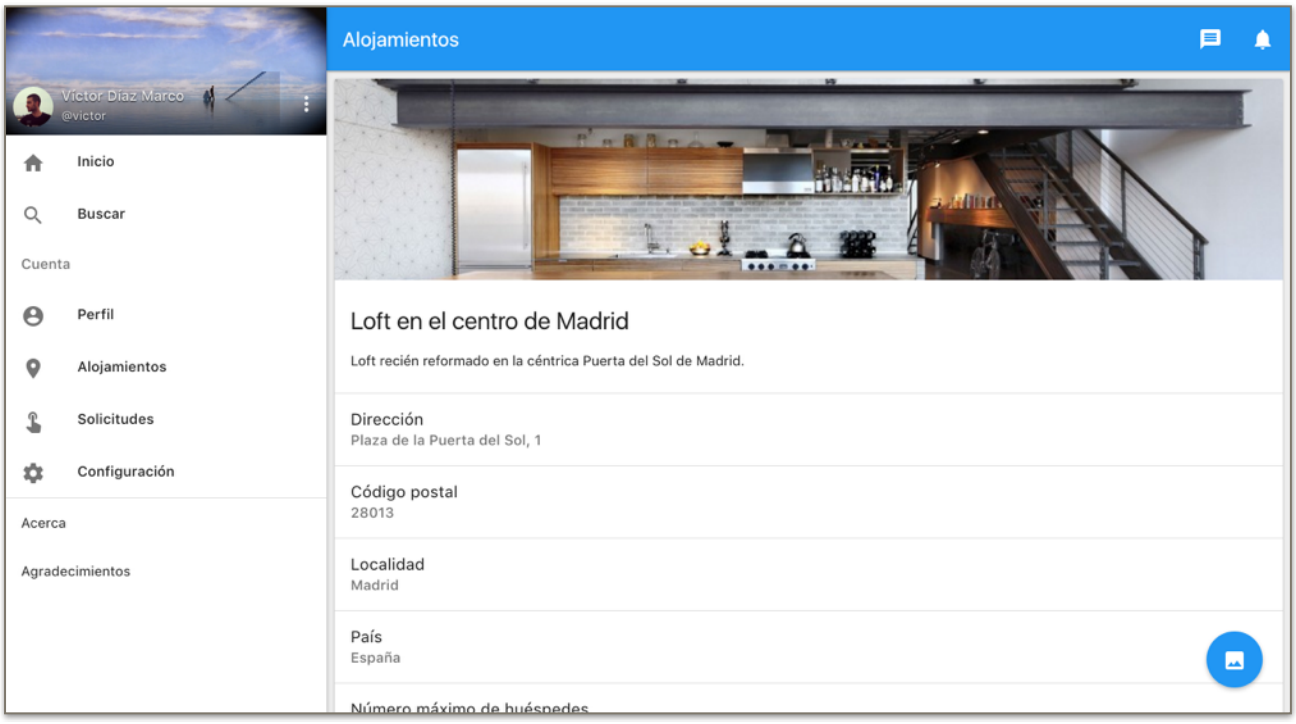

Figura 41. Captura de la vista de alojamiento.

Para comprobar que es posible realizar intercambios se ha realizado una solicitud, la cual ha sido creada sin ningún inconveniente. Con esto queda validado el caso de uso #15.

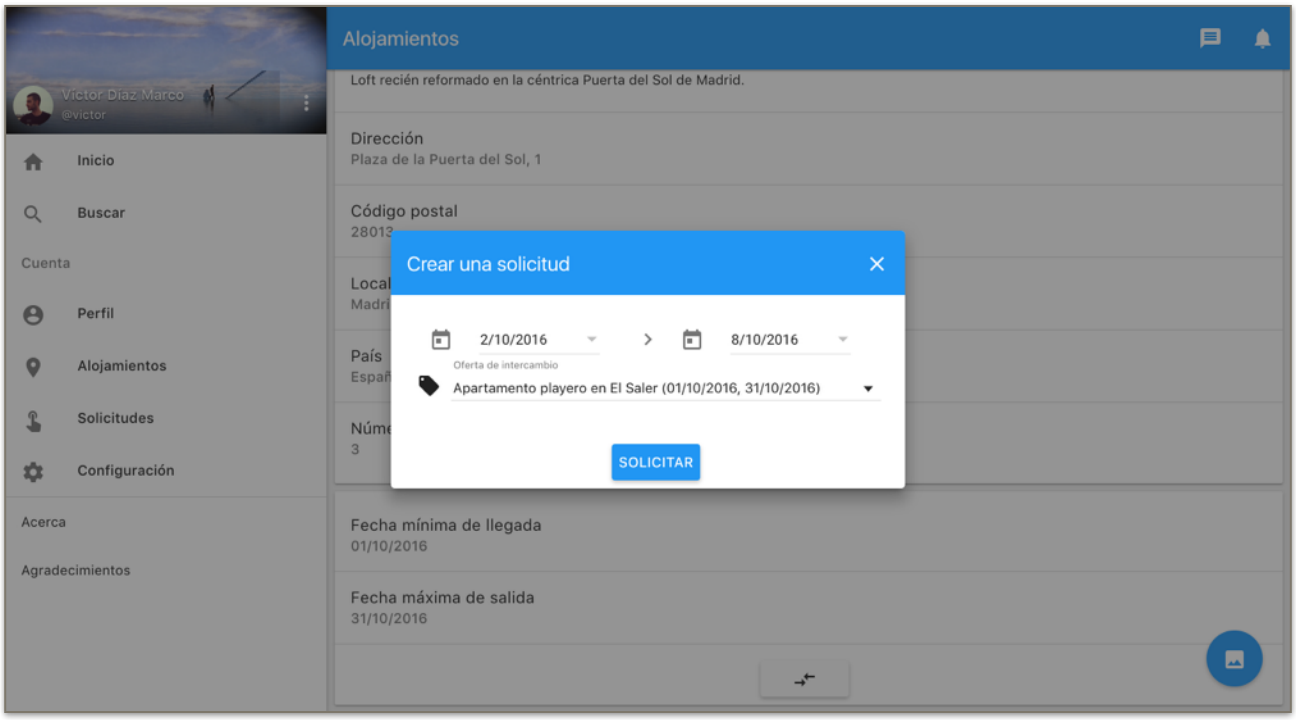

Figura 42. Captura de la vista de creación de solicitud de intercambio.

Posteriormente a la creación de la solicitud esta aparece reflejada en la vista correspondiente (caso de uso #16).

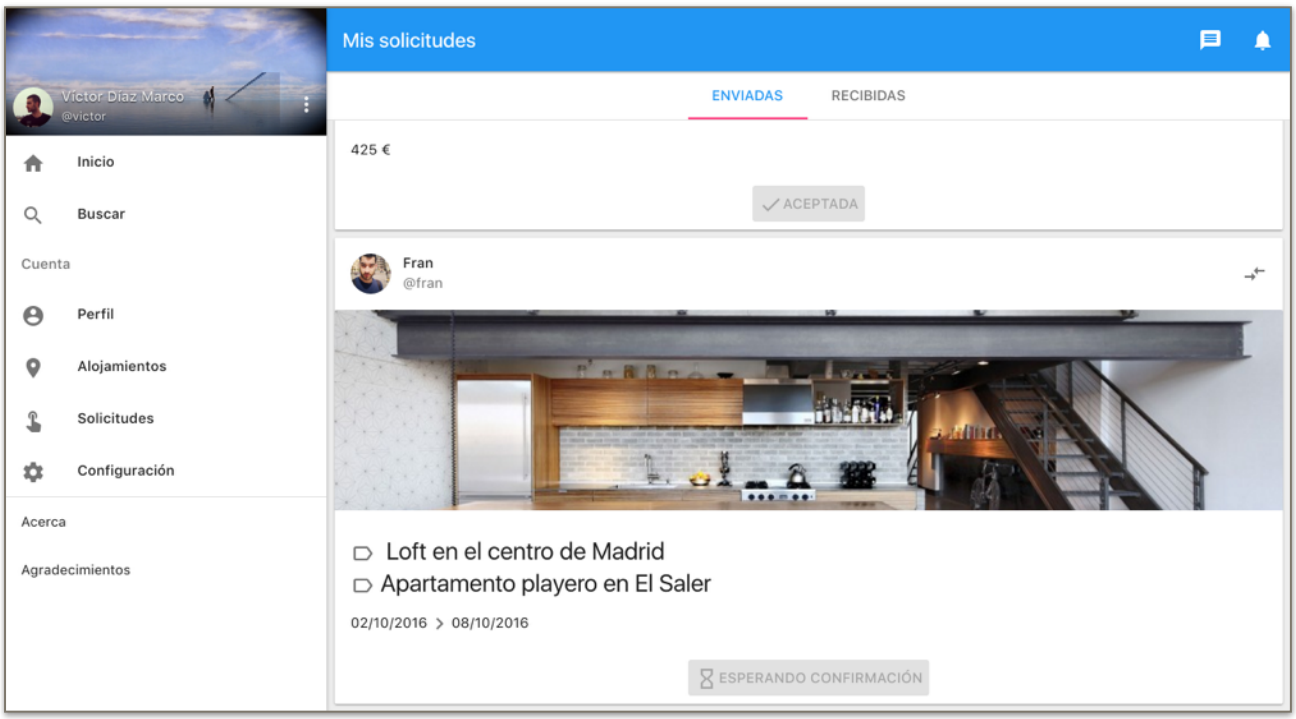

Figura 43. Captura de la vista de solicitudes (pestaña de enviadas).

Por otra parte, el usuario que recibe la solicitud también la ve reflejada y puede aceptarla o rechazarla (casos de uso #16, #17 y #18).

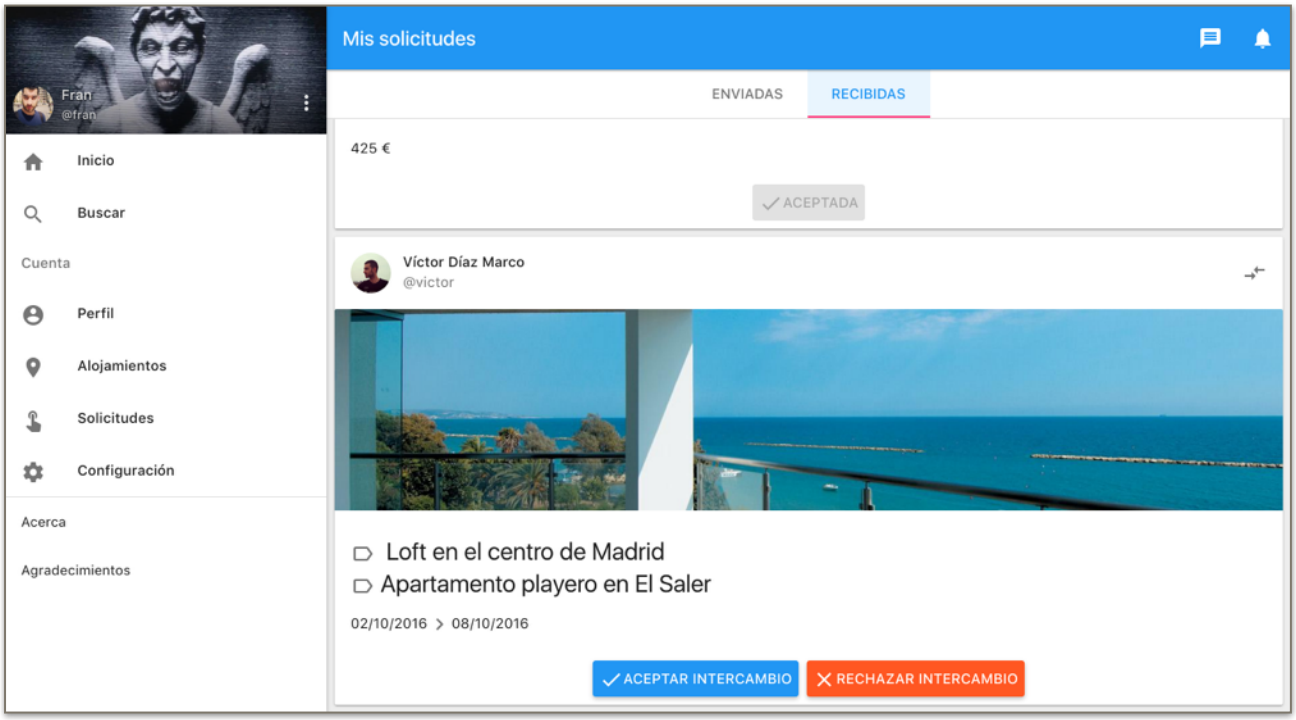

Figura 44. Captura de la vista de solicitudes (pestaña de recibidas).

Respecto a la edición de alojamiento, se ha comprobado la posibilidad de editar toda la información relacionada (caso de uso #9). A continuación se observa la vista de edición de los datos principales del alojamiento.

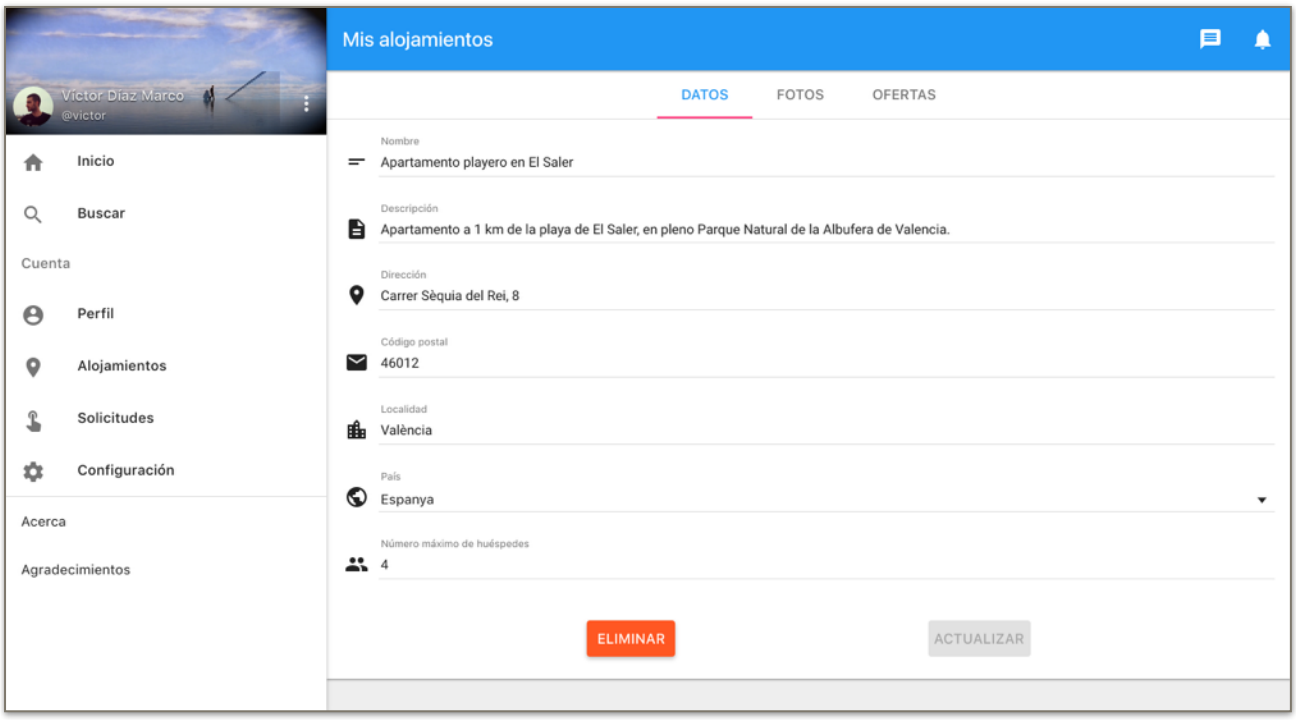

Figura 45. Captura de la vista de edición de alojamiento (pestaña de información).

La siguiente vista posibilita la adición y eliminación de fotografías a un alojamiento.

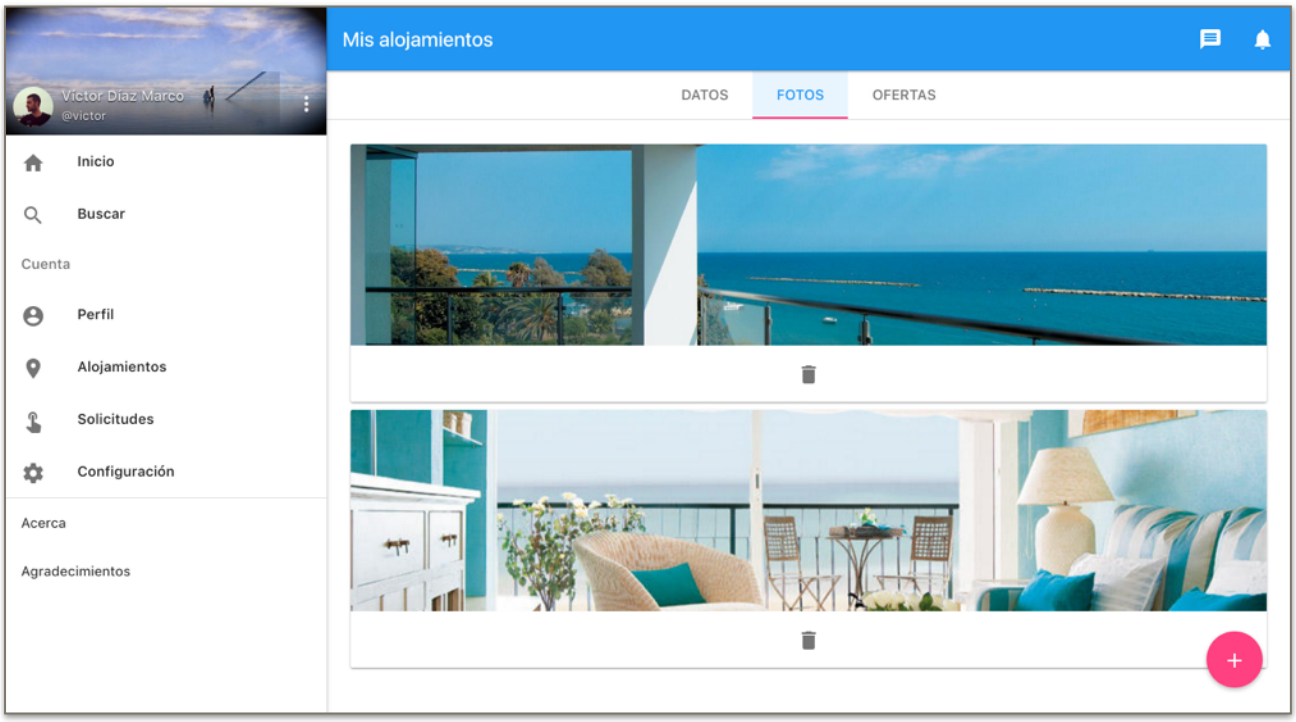

Figura 46. Captura de la vista de edición de alojamiento (pestaña de fotos).

Por último se muestra la vista de configuración, que permite editar toda la información relacionada con una cuenta (caso de uso #5).

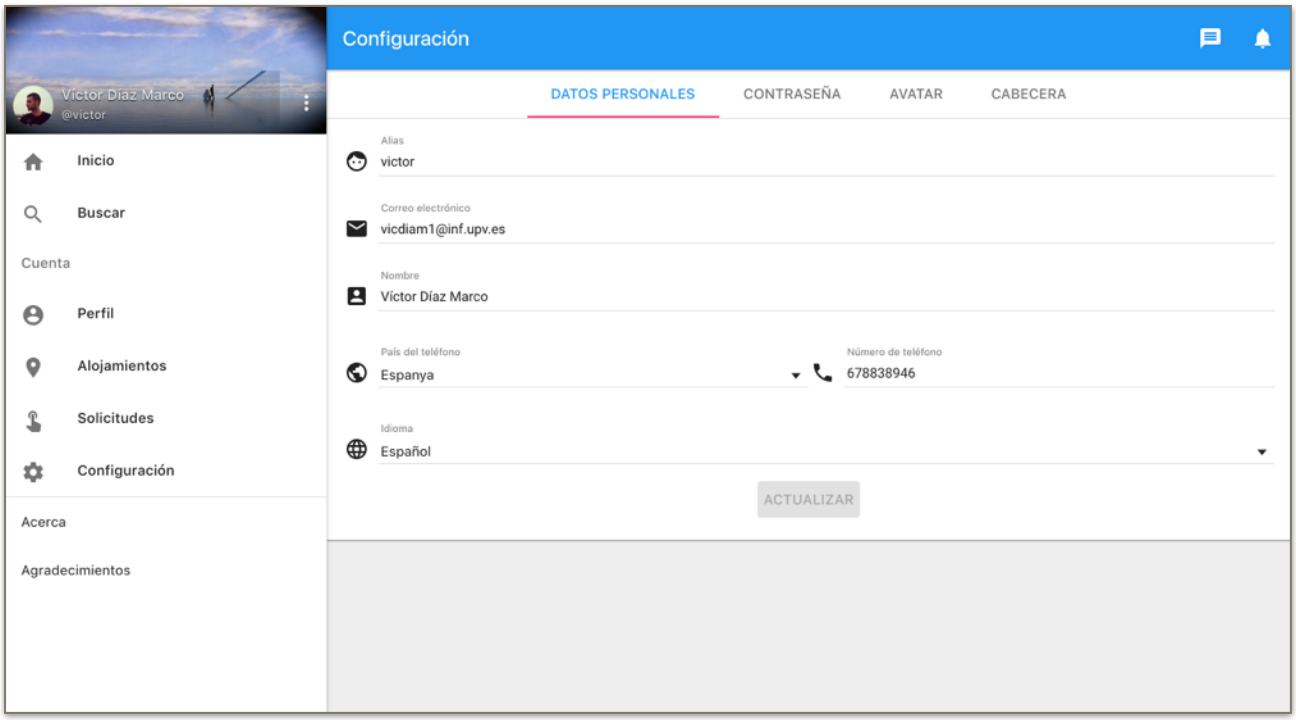

Figura 47. Captura de la vista de configuración (pestaña de datos).

Una vez analizados todos los casos de uso, de los cuales se han expuesto los más relevantes, es posible concluir que la aplicación cumple estrictamente con la funcionalidad especificada.

8.3. Pruebas no funcionales

Las pruebas no funcionales se encargan de verificar que el comportamiento de la aplicación es el que cabe esperar.

8.3.1. Pruebas unitarias

Las pruebas unitarias consisten en comprobar que diferentes fragmentos de la aplicación devuelven los resultados esperados en diferentes casos. Para llevarlas a cabo se ha utilizado la aplicación Postman, que permite asociar un conjunto de pruebas escritas en código JavaScript a cada una de las consultas a la API.

A continuación puede apreciarse el código fuente de la prueba unitaria para la creación de un alojamiento, es decir, para la petición HTTP `POST /v1/accommodations` al *back-end*.

```
1  tests['Response time is less than 200ms'] = responseTime < 200;
2
3  tests['Successful POST request'] = responseCode.code === 201 ||
   responseCode.code === 200;
4
5  var jsonData = JSON.parse(responseBody);
6  tests['The resource is returned successfully'] = jsonData.id;
```

La prueba se considera satisfactoria si la petición tarda menos de 200 ms, si se devuelve un estado `201` (creado) o `200` (correcto), y si el cuerpo de la petición incluye el campo `id`. Esto último es una de las muchas formas de comprobar si se está devolviendo correctamente el objeto creado en el cuerpo de la respuesta.

Este tipo de pruebas se han ido realizando durante toda la fase de implementación para ir obteniendo realimentación constante.

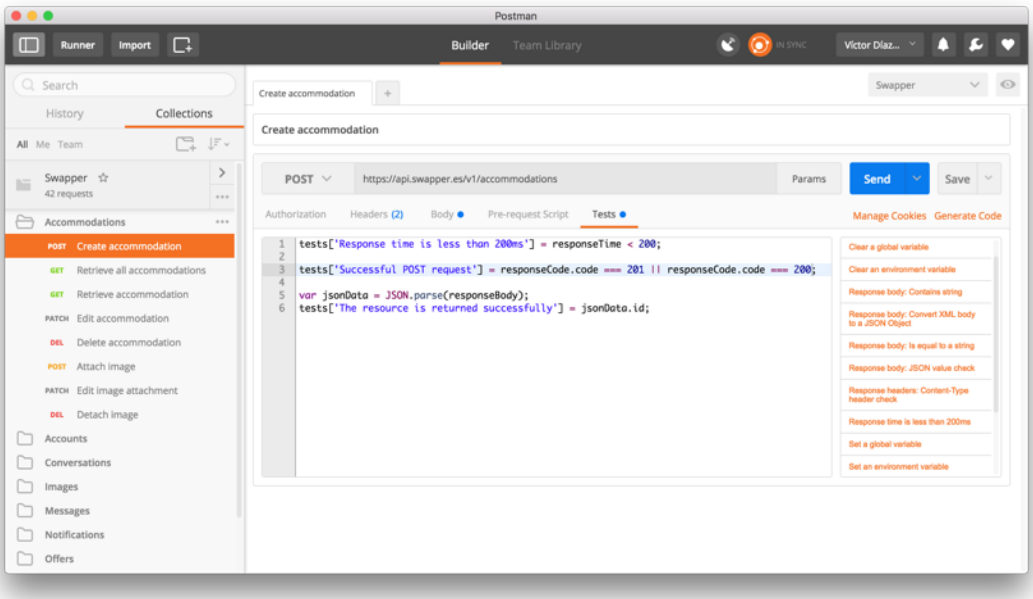

Figura 48. Captura de las pruebas unitarias de Postman.

8.3.2. Conformidad con los estándares del W3C

Al tratarse de una aplicación web basada en AngularJS, parte del código HTML 5 final es generado en tiempo de ejecución por dicho *framework*. Por tanto, no es posible utilizar el validador del W3C para comprobar directamente el código HTML del proyecto, ya que existirán algunos errores relacionados con las etiquetas y atributos personalizados de Angular. Estos son convertidos en tiempo de ejecución a código HTML 5 válido, tanto sintáctica como semánticamente.

Por esa razón se ha hecho uso del validador del W3C individualmente para cada fichero HTML del proyecto, descartando manualmente aquellos errores de validación relacionados con el uso de Angular. Esto ha permitido solucionar algunos errores, obteniendo como resultado un proyecto final que cumple con el estándar HTML 5.

8.3.3. Visualización en diferentes navegadores y plataformas

Una vez comprobado que el proyecto cumple los estándares cabría esperar compatibilidad absoluta con todos los navegadores que los respeten y estén al día con los mismos. Sin embargo, cada navegador hace su interpretación de los estándares y tiene sus peculiaridades. Por ese motivo se han realizado pruebas de visualización en diferentes navegadores.

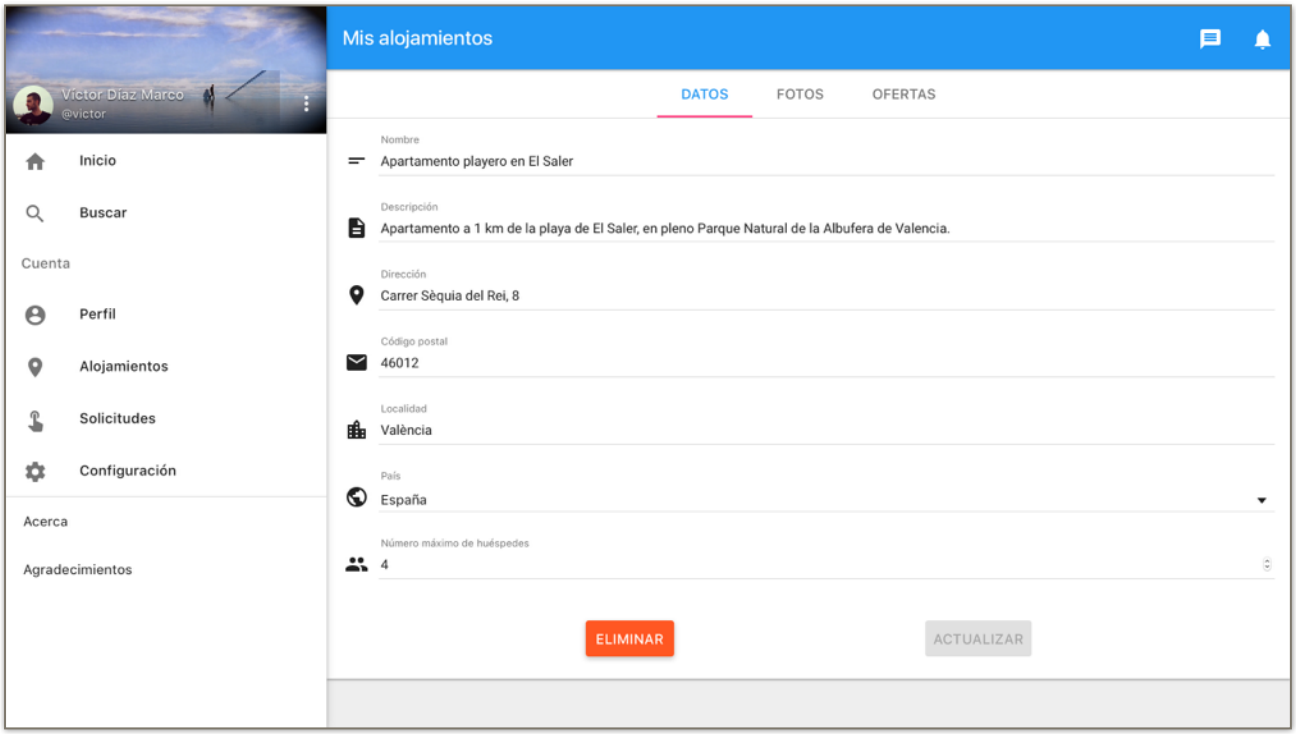

Figura 49. Prueba de visualización desde Firefox 48 (MacOS 10.11).

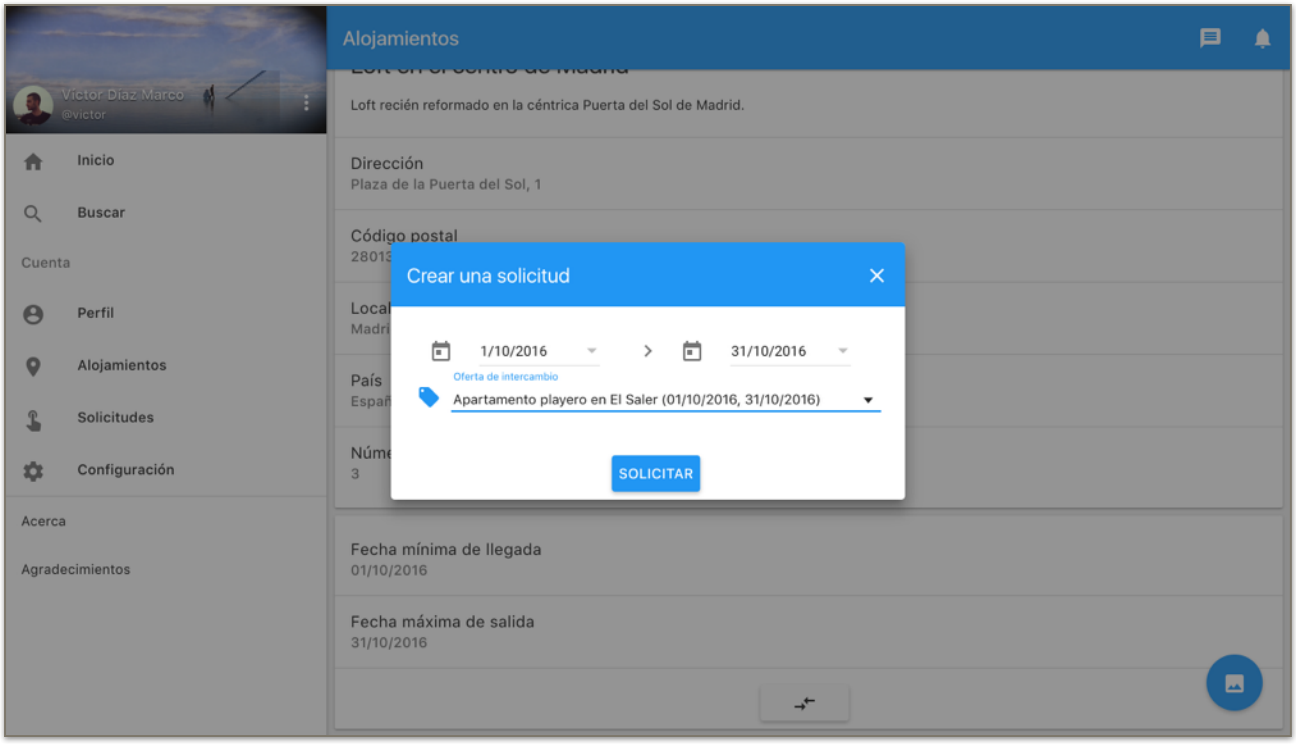

Figura 50. Prueba de visualización desde Opera 39 (MacOS 10.11).

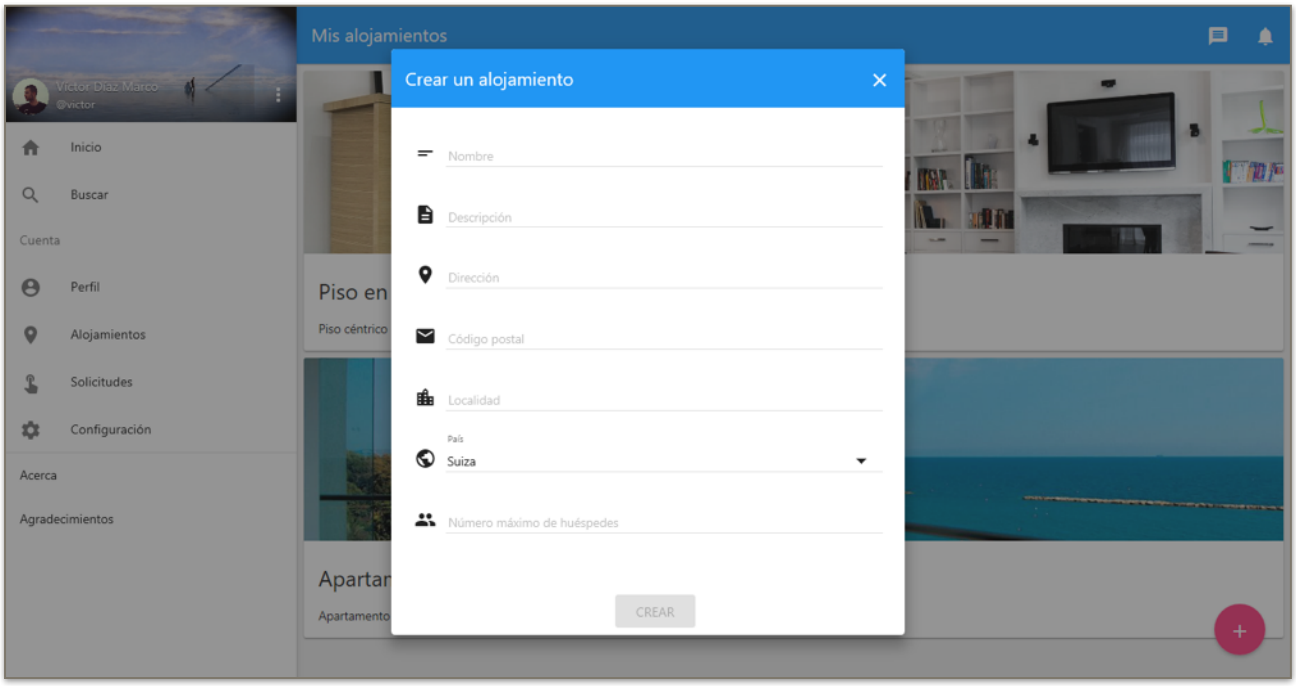

Figura 51. Prueba de visualización desde Edge 38 (Windows 10).

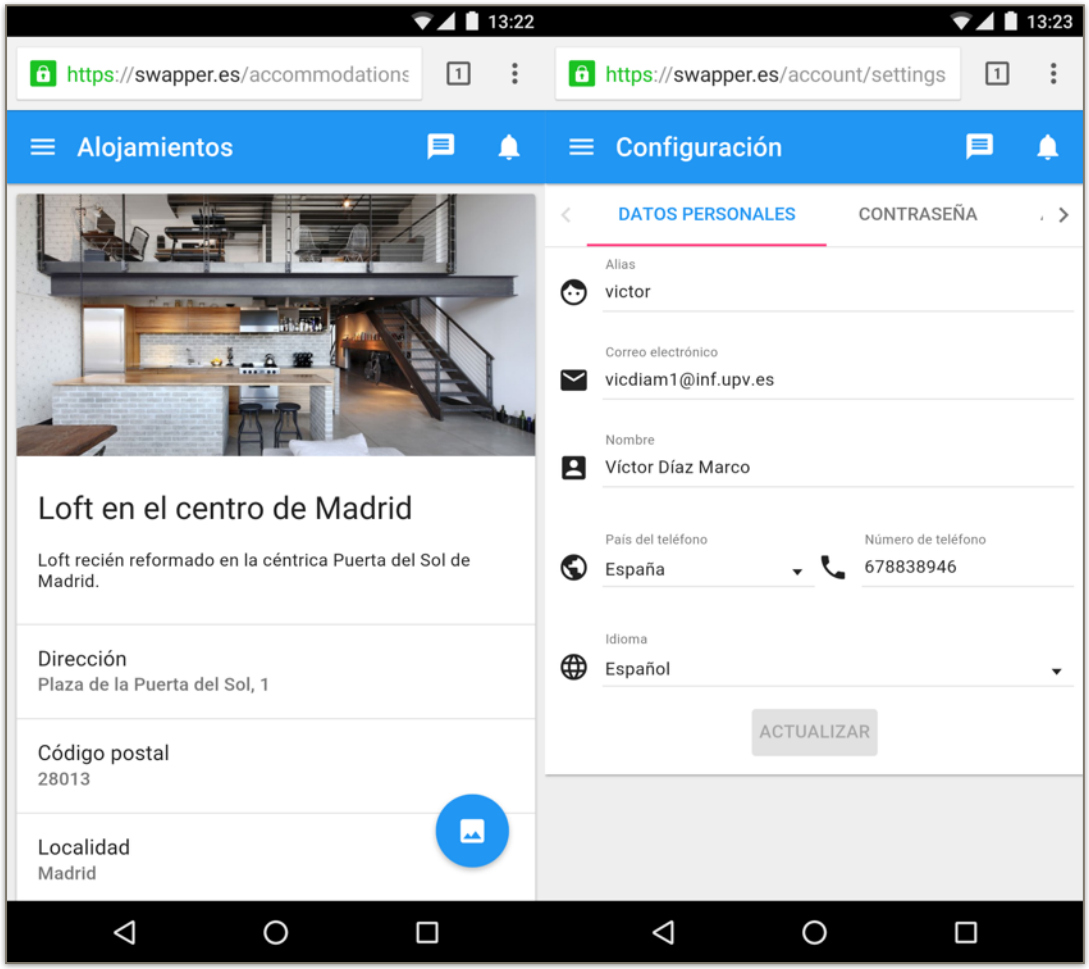

Figura 52. Prueba de visualización desde Chrome 52 (Android 6).

Como puede apreciarse, no existe ningún problema de visualización ni funcionamiento en los diferentes navegadores y sistemas operativos probados.

8.3.4. Realimentación de errores

Por último se ha comprobado que la realimentación de errores funciona adecuadamente. Esto permite que el flujo del usuario no se vea interrumpido de forma forzosa, sino que se le indique qué medidas debe tomar para poder continuar. A continuación se exponen algunos ejemplos de las comprobaciones realizadas.

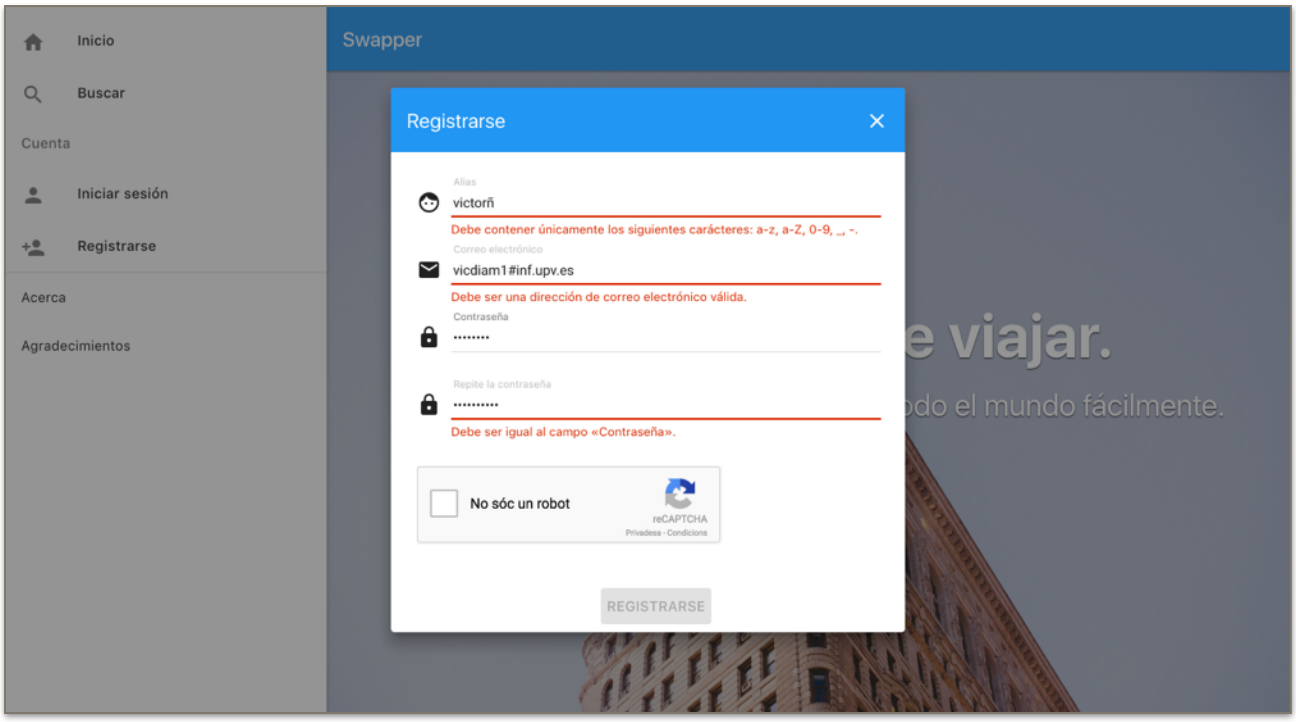

Figura 53. Prueba de realimentación de errores de validación.

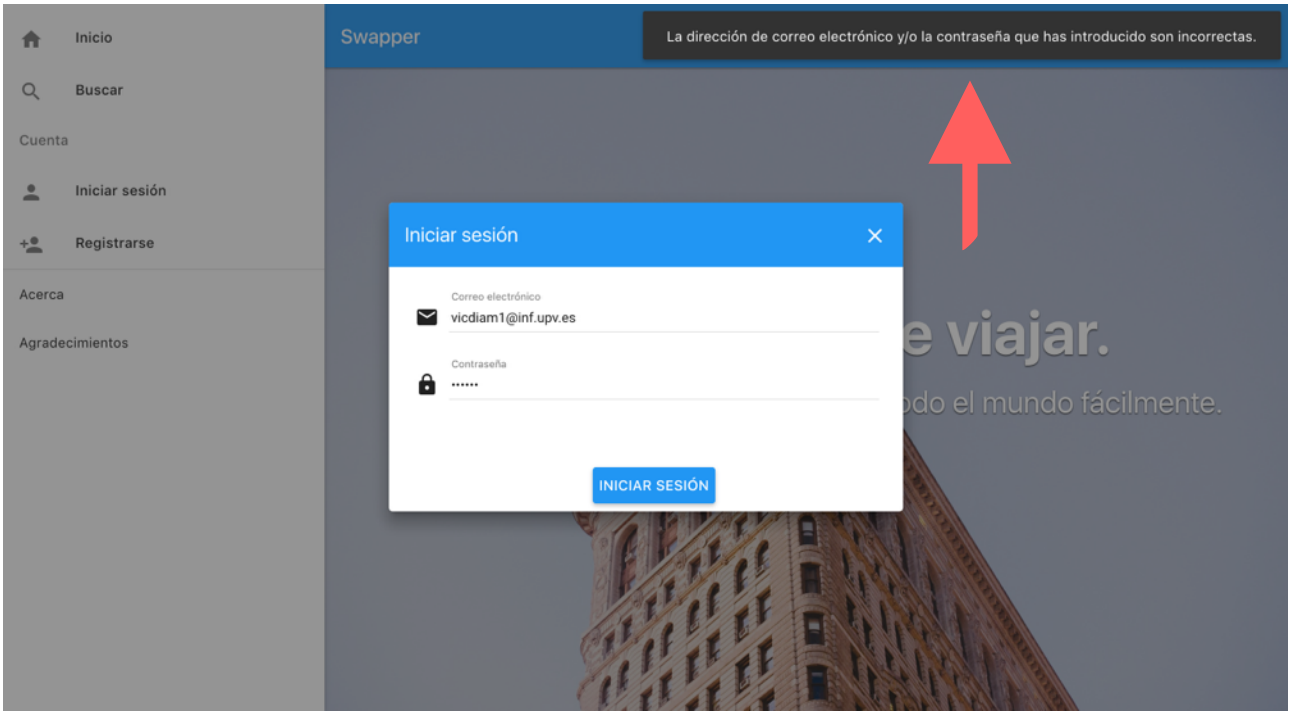

Figura 54. Prueba de realimentación de errores generales.

9. Conclusiones

El objetivo del trabajo era poner en marcha una aplicación web que permitiese el intercambio y alquiler de viviendas entre usuarios mediante la observancia de las disciplinas adquiridas como fruto de haber cursado la titulación.

El proyecto ha sido desarrollado completamente y de acuerdo con la especificación. Como resultado se ha obtenido una aplicación web plenamente funcional que no solo cumple con la propuesta inicial del trabajo, sino que también contempla otras características como el alquiler de viviendas (además del intercambio), el soporte a varios idiomas, un sistema de mensajería privada y un planteamiento que permite ofertar alojamientos por lotes.

El hecho de permitir alquileres hace que desde una misma aplicación se puedan explotar las dos ramas de la economía colaborativa que se exponían al principio del trabajo: la que consiste en aprovechar un recurso existente para ahorrar y la que se basa en la creación de un modelo de negocio para obtener ingresos extra. Además, gracias a la posibilidad de publicar ofertas por lotes se abre la puerta a que la aplicación sea utilizada también por negocios.

La parte más decisiva del proyecto ha sido el análisis a la hora de plantear las entidades de la aplicación y sus relaciones. El objetivo de dicho proceso era combinar de la forma más simple y versátil posible los alojamientos, las ofertas, los intercambios y los alquileres. Esta parte es clave en el desarrollo de la aplicación, ya que un mal planteamiento podría haber dificultado enormemente la fase de implementación.

La planificación y la definición de metas han sido los aspectos que más han contribuido a un buen desarrollo del proyecto. El hecho de no trabajar a ciegas y poder delegar ciertos aspectos en una planificación previamente establecida ha permitido finalizar el desarrollo en un tiempo razonable y ajustándose al alcance.

En cuanto a las dificultades más relevantes conviene destacar el planteamiento de la estructura de directorios y ficheros de AngularJS. Existen diversas guías de estilo y argumentaciones a favor de distintas propuestas, sin embargo fue necesario tener en cuenta numerosos puntos de vista para terminar creando un planteamiento diferente a lo que se proponía. Cada aplicación tiene sus peculiaridades y lo que podría considerarse una estructura excelente para una puede ser un mal planteamiento para otra.

Respecto a los conocimientos adquiridos, el desarrollo del proyecto ha permitido profundizar en la mecánica de una aplicación web distribuida, teniendo en cuenta todos los elementos por los que está formada y los agentes que intervienen en su funcionamiento. Además, la combinación de la arquitectura de tres capas y el patrón modelo-vista-controlador han permitido entender mejor la utilidad de estos planteamientos en el ámbito de la arquitectura de software y sus implicaciones para el resto de la aplicación.

Los diferentes planteamientos arquitectónicos que se exponen en este trabajo pueden ser tomados como base para el desarrollo de cualquier aplicación web, sea cual sea su naturaleza. Por otra parte, destaca la posibilidad de ampliación: convertir este proyecto en un negocio sería tan fácil como añadir una pasarela de pago y una comisión por trámite a cada solicitud aceptada.

Como valoración final, cabe destacar que desarrollar una aplicación web, cosa que a priori puede parecer trivial, implica muchas disciplinas, tales como la gestión de proyectos, la ingeniería de software, la administración de sistemas, la criptografía e incluso algunos aspectos legales. Es bien conocido el desproporcionado intrusismo profesional que existe en el ámbito de la ingeniería informática, en muchas ocasiones debido a la facilidad que existe en la actualidad a la hora de obtener información. Si bien es cierto que cualquier persona es capaz de desarrollar una aplicación web habiendo adquirido unos conocimientos básicos, es fundamental comprender que no por ello contemplará todo aquello que debe para garantizar la obtención de un producto final apropiado, fiable y seguro. La aportación de un profesional siempre es imprescindible y beneficia tanto al cliente de un producto como a la sociedad en su conjunto.

10. Bibliografía

(1) ACHOUR, M. et al. (1997). *PHP Manual*. The PHP Group. <http://php.net/manual/en/> [consulta: 5 de junio de 2016].

(2) BURBACK, R. T. (1998). “Software Engineering Phases” en *Software Engineering Methodology: The WaterSluice*. California: Stanford University. <http://infolab.stanford.edu/~burback/watersluice/node2.html> [consulta: 20 de agosto de 2016].

(3) DICKEY, J. (2014). *Best Practices for Building Angular.js Apps*. Medium. <https://medium.com/@dickeyxxx/best-practices-for-building-angular-js-apps-266c1a4a6917> [consulta: 5 de junio de 2016].

(4) FIELDING, R. T. (2000). “Representational State Transfer (REST)” en *Architectural Styles and the Design of Network-based Software Architectures*. California: University of California, Irvine. <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm> [consulta: 17 de agosto de 2016].

(5) FRANKEL, R. (2016). *NGINX vs. Apache (Pro/Con Review, Uses, & Hosting for Each)*. HostingAdvice. <http://www.hostingadvice.com/how-to/nginx-vs-apache/> [consulta: 23 de agosto de 2016].

(6) GOOGLE INC. *API Reference*. AngularJS. <https://docs.angularjs.org/api> [consulta: 5 de junio de 2016].

(7) GOOGLE INC. *API Reference*. Angular Material. <https://material.angularjs.org/latest/api> [consulta: 27 de julio de 2016].

(8) GOOGLE INC. *Developer Guide*. AngularJS. <https://docs.angularjs.org/guide> [consulta: 5 de junio de 2016].

(9) GOOGLE INC. *Material design*. <https://material.google.com> [consulta: 5 de junio de 2016].

(10) GOOGLE INC. *reCAPTCHA Developer's Guide*. Google Developers. <https://developers.google.com/recaptcha/intro> [consulta: 5 de junio de 2016].

(11) INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (2011). *29148-2011 - Systems and software engineering -- Life cycle processes -- Requirements engineering*. <https://standards.ieee.org/findstds/standard/29148-2011.html> [consulta: 5 de junio de 2016].

(12) JSON. *Introducing JSON*. <http://json.org> [consulta: 5 de junio de 2016].

(13) LARAVEL. *Laravel documentation*. <https://laravel.com/docs/master> [consulta: 5 de junio de 2016].

(14) LARAVEL. *Lumen documentation*. <https://lumen.laravel.com/docs/master> [consulta: 5 de junio de 2016].

(15) MARIADB CORPORATION. *MariaDB Documentation*. <https://mariadb.com/kb/en/mariadb/documentation/> [consulta: 5 de junio de 2016].

(16) MOMENT.JS. *Moment.js Documentation*. <http://momentjs.com/docs> [consulta: 5 de agosto de 2016].

(17) MOZILLA FOUNDATION. *HTTP access control (CORS)*. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS> [consulta: 5 de junio de 2016].

(18) NGINX. *Nginx documentation*. <https://nginx.org/en/docs> [consulta: 5 de junio de 2016].

(19) NIELSEN, J. (1993). *Response Times: The 3 Important Limits*. Nielsen Norman Group. <https://www.nngroup.com/articles/response-times-3-important-limits/> [consulta: 5 de junio de 2016].

(20) PAPA, J. *Angular 1 Style Guide*. Angular Style Guide. <https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md> [consulta: 5 de junio de 2016].

(21) PHOTOSWIPE. *API*. <http://photoswipe.com/documentation/api.html> [consulta: 14 de agosto de 2016].

(22) PRECHT, P. *Guide*. Angular Translate. <https://angular-translate.github.io/docs/#/guide> [consulta: 27 de julio de 2016].

(23) SAHNI, V. (2015) *Best Practices for Designing a Pragmatic RESTful API*. <http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api> [consulta: 5 de junio de 2016].

(24) SÁNCHEZ VICENTE, T. (2015). "La crisis, compartida, es menos crisis" en ABC. Madrid: Diario ABC. <http://www.abc.es/economia/20130615/abci-economia-colaborativa-201306101212.html> [consulta: 17 de agosto de 2016].

(25) SILES, R. (2011). *Session Management Cheat Sheet*. Open Web Application Security Project. <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet> [consulta: 5 de junio de 2016].

(26) VAN KESTEREN, A. (2014). *Cross-Origin Resource Sharing*. World Wide Web Consortium. <https://www.w3.org/TR/cors/> [consulta: 5 de junio de 2016].

(27) WIKIPEDIA. *Application programming interface*. <https://en.wikipedia.org/wiki/Application_programming_interface> [consulta: 5 de junio de 2016].

(28) WIKIPEDIA. *Collaborative consumption*. <https://en.wikipedia.org/wiki/Collaborative_consumption> [consulta: 17 de agosto de 2016].

(29) WIKIPEDIA. *File verification*. <https://en.wikipedia.org/wiki/File_verification> [consulta: 5 de junio de 2016].

(30) WIKIPEDIA. *PHP*. <https://en.wikipedia.org/wiki/PHP> [consulta: 18 de agosto de 2016].

(31) WIKIPEDIA. *Postal code*. <https://en.wikipedia.org/wiki/Postal_code> [consulta: 5 de junio de 2016].

(32) WIKIPEDIA. *SHA-2*. <https://en.wikipedia.org/wiki/SHA-2> [consulta: 5 de junio de 2016].

(33) WIKIPEDIA. *Software engineering*. <https://en.wikipedia.org/wiki/Software_engineering> [consulta: 17 de agosto de 2016].

(34) WIKIPEDIA. *Software requirements specification*. <https://en.wikipedia.org/wiki/Software_requirements_specification> [consulta: 5 de junio de 2016].

(35) W3TECHS. *Historical trends in the usage of client-side programming languages for websites*. <https://w3techs.com/technologies/history_overview/client_side_language/all> [consulta: 18 de agosto de 2016].

(36) W3TECHS. *Historical yearly trends in the usage of server-side programming languages for websites*. <https://w3techs.com/technologies/history_overview/programming_language/> [consulta: 18 de agosto de 2016].

(37) 20 MINUTOS. (2015). "Los españoles piensan viajar menos este verano: la intención de salir desciende del 60 al 49%" en 20 Minutos. <http://www.20minutos.es/noticia/2751607/0/espanoles-viajaran-menos-verano-2016-encuesta/> [consulta: 17 de agosto de 2016].

Anexos

A. Fichero «/etc/nginx/nginx.conf»

```
user nginx;
pid /var/run/nginx.pid;

worker_processes auto;
worker_rlimit_nofile 100000;

events {
	worker_connections 2048;
	use epoll;
	multi_accept on;
}

http {
	# Basic settings
	charset utf-8;
	index index.php index.html;
	types_hash_max_size 2048;

	# MIME
	include /etc/nginx/mime.types;
	default_type application/octet-stream;

	# Log
	access_log off;
	error_log /var/log/nginx/error.log emerg;

	# CloudFlare
	include /etc/nginx/cloudflare.conf;

	# Gzip
	gzip on;
	gzip_min_length 256;
	gzip_proxied any;
	gzip_comp_level 3;
	gzip_disable msie6;
	gzip_types text/plain text/css application/json application/x-
javascript text/xml application/xml application/xml+rss text/javascript;

	# TLS
	ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
	ssl_ciphers EECDH+CHACHA20:EECDH+AES128:RSA+AES128:EECDH+AES256:RSA
+AES256:EECDH+3DES:RSA+3DES:!MD5;
	ssl_prefer_server_ciphers on;
	ssl_session_cache shared:SSL:10m;

	# Performance settings
	open_file_cache max=10000 inactive=30s;
	open_file_cache_valid 60s;
	open_file_cache_min_uses 2;
	open_file_cache_errors on;
	keepalive_timeout 65;
	keepalive_requests 200;
	reset_timedout_connection on;
```

```
52          sendfile on;
53          tcp_nopush on;
54          tcp_nodelay on;
55          client_body_timeout 10;
56          send_timeout 2;
57
58          # DDoS protection settings
59          client_max_body_size 4M;
60          client_body_buffer_size 128k;
61          large_client_header_buffers 4 256k;
62          limit_conn_zone $binary_remote_addr zone=conn_limit_per_ip:1m;
63          limit_req_zone $binary_remote_addr zone=req_limit_per_ip:1m rate=80r/
    s;
64          limit_conn conn_limit_per_ip 10;
65          limit_req zone=req_limit_per_ip burst=20;
66
67          # Other
68          ignore_invalid_headers on;
69          server_tokens off;
70          server_name_in_redirect off;
71
72          # Includes
73          include /etc/nginx/conf.d/*.conf;
74  }
```

B. Fichero «/etc/nginx/cloudflare.conf»

```
set_real_ip_from 103.21.244.0/22;
set_real_ip_from 103.22.200.0/22;
set_real_ip_from 103.31.4.0/22;
set_real_ip_from 104.16.0.0/12;
set_real_ip_from 108.162.192.0/18;
set_real_ip_from 131.0.72.0/22;
set_real_ip_from 141.101.64.0/18;
set_real_ip_from 162.158.0.0/15;
set_real_ip_from 172.64.0.0/13;
set_real_ip_from 173.245.48.0/20;
set_real_ip_from 188.114.96.0/20;
set_real_ip_from 190.93.240.0/20;
set_real_ip_from 197.234.240.0/22;
set_real_ip_from 198.41.128.0/17;
set_real_ip_from 199.27.128.0/21;
set_real_ip_from 2400:cb00::/32;
set_real_ip_from 2405:8100::/32;
set_real_ip_from 2405:b500::/32;
set_real_ip_from 2606:4700::/32;
set_real_ip_from 2803:f800::/32;

real_ip_header CF-Connecting-IP;
```

C. Fichero «/etc/nginx/conf.d/swapper.conf»

```
# Web
include /home/swapper/web/nginx.conf;

# API
include /home/swapper/api/nginx.conf;

# CDN
server {
        listen 80;
        listen [::]:80;
        server_name cdn.swapper.es;

        rewrite (.*) https://cdn.swapper.es$1 permanent;
}

server {
        listen 443 ssl http2;
        listen [::]:443 ssl http2;
        ssl_certificate /etc/pki/tls/certs/swapper/all.crt;
        ssl_certificate_key /etc/pki/tls/private/swapper.key;

        ssl_verify_client on;
        ssl_client_certificate /etc/pki/tls/certs/cloudflare.crt;

        server_name cdn.swapper.es;
        root /home/swapper/cdn/public;

        location = / {
                return 301 https://swapper.es;
        }

        include cors;
}
```

D. Fichero «/etc/nginx/cors»

```
location / {
      set $allow_origin '*';
      set $allow_methods 'GET, OPTIONS';
      set $allow_headers 'Keep-Alive, Cache-Control, If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match';

      if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' $allow_origin;
            add_header 'Access-Control-Allow-Methods' $allow_methods;
            add_header 'Access-Control-Allow-Headers' $allow_headers;

            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            add_header 'Content-Length' 0;

            return 204;
      }

      if ($request_method = 'GET') {
            add_header 'Access-Control-Allow-Origin' $allow_origin;
            add_header 'Access-Control-Allow-Methods' $allow_methods;
            add_header 'Access-Control-Allow-Headers' $allow_headers;
      }
}
```

E. Fichero «/etc/php-fpm.d/swapper.conf»

```
[swapper]

user = $pool
group = $pool

listen = /var/run/php-fpm/$pool.sock
listen.owner = nginx
listen.group = nginx
listen.allowed_clients = 127.0.0.1

pm = ondemand
pm.max_children = 50

env[TMP] = /home/$pool/.tmp
env[TMPDIR] = /home/$pool/.tmp
env[TEMP] = /home/$pool/.tmp

php_value[open_basedir] = /home/$pool:/usr/share/php:/usr/share/pear
php_value[session.save_handler] = files
php_value[session.save_path] = /home/$pool/.tmp
php_value[soap.wsdl_cache_dir] = /home/$pool/.tmp
```

F. Fichero «/etc/my.cnf.d/server.cnf»

```
1 [mysqld]
2 character-set-client-handshake = FALSE
3 character-set-server = utf8mb4
4 collation-server = utf8mb4_unicode_ci
5 init-connect = 'SET NAMES utf8mb4'
6 ft_min_word_len = 3
7 default-storage-engine = MyISAM
```