Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Aplicación web para la gestión de hoteles de pequeño tamaño

Estudante: Rubén Vázquez Fariña
Dirección: Manuel Álvarez Díaz

A Coruña, setembro de 2021.

A todos los que han estado ahí y a los que no han podido estar

Agradecimientos

A mi familia por darme la oportunidad de llegar hasta aquí. Especialmente a los que me han dado fuerzas de forma no visible.

A los amigos que me llevo y que me han hecho pasar unos de los mejores años de mi vida.

A Manuel Álvarez Díaz por ayudarme a completar este proyecto.

A los que no han podido estar y sé que les habría gustado.

Resumen

Con este proyecto se pretende ayudar a mejorar la situación de muchos hoteles de pequeño tamaño que se encuentran con poca afluencia de clientes o aún se mantienen con la reserva telefónica como medio principal para la llegada de clientes. Por ello, se ha diseñado e implementado una aplicación web que permita facilitar la reserva de habitaciones y la gestión interna de este tipo de hoteles.

La aplicación permite el registro de hoteles de pequeño tamaño para facilitar a los usuarios individuales realizar búsquedas para localizar aquellos que mejor se ajusten a sus necesidades, así como realizar reservas o adquirir los productos y servicios que tienen disponibles. Una vez ejecutada una reserva, la aplicación permitirá también gestionar al personal del hotel la entrada y salida de los huéspedes así como la cuenta de gasto en función de los servicios del hotel que consuman durante su estancia.

La aplicación pretende reducir el esfuerzo que supone estar entrando en diferentes webs para localizar los hoteles y comparar los precios de estos. De este modo, un usuario puede filtrar los hoteles que desee, ya sea por el nombre del mismo o por la dirección de destino del viaje, desde un mismo sitio web. Además, permite proporcionar visibilidad en Internet a pequeños hoteles que pueden no tener presupuesto para un sitio web propio, además de facilitarle la gestión de sus reservas y estancias de clientes.

Para la realización de la aplicación se han utilizado tecnologías Java para el servidor y JavaScript/React para la capa web.

Abstract

The aim of this project is to help improve the situation of many small hotels that are with little influx of customers or are still using the telephone reservation as the main means for the arrival of customers. For this reason, a web application has been designed and implemented to facilitate the reservation of rooms and the internal management of this type of hotels.

The application allows the registration of small hotels to facilitate individual users to carry out searches to locate those that best suit their needs, as well as make reservations or purchase the products and services available to them. Once a reservation has been made, the application will also allow hotel staff to manage the entry and exit of guests as well as the expense account based on the hotel services they consume during their stay.

The application aims to reduce the effort involved in entering different websites to locate hotels and compare their prices. In this way, a user can filter the hotels they want, either by their name or by the travel destination address, from the same website. In addition, it allows

providing visibility on the Internet to small hotels that may not have a budget for their own website, as well as facilitating the management of their reservations and client stays.

To carry out the application, Java technologies have been used for the server and JavaScript / React for the web layer.

Palabras clave:

- Hotel
- Reserva
- React
- Aplicación Web
- Springboot
- JavaScript
- Redux

Keywords:

- Hotel
- Booking
- React
- Web Application
- Springboot
- JavaScript
- Redux

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La revolución tecnológica de los últimos años ha favorecido el aumento del consumo de las nuevas tecnologías en la sociedad actual, ya sea para realizar compras de primera necesidad, como para planificar sus vacaciones.

Como resultado de esta tendencia al alza, los hoteles de pequeño tamaño, han tenido que reinventarse, debido al aumento de carga de trabajo que produce la gestión manual, pasando a crear ellos mismos su propia página web en donde realizar las reservas de las habitaciones.

Favorecido por el aumento de la carga laboral de esta gestión, la dependencia de que haya alguien disponible en recepción continuamente para notificar al personal de limpieza las habitaciones ocupadas o libres, dificulta su trabajo y produce fallos de comunicación interna, siendo evitables mediante la delegación de parte del trabajo en una aplicación que facilita su gestión.

1.1 Determinación de la situación actual

Actualmente, existen diversos mecanismos a través de los cuales se puede realizar una reserva de habitación en un hotel determinado. El más habitual, es mediante una llamada telefónica al hotel, generando los problemas ya comentados anteriormente, aunque también está aumentando el uso de las plataformas de reserva online. Con independencia del tamaño de los hoteles, el medio más usado para realizar las reservas es Internet. Sin embargo, los hoteles de pequeño tamaño o los hostales, no pueden hacer uso de estas páginas web, ya sea por el coste que estas suponen o por la complejidad que tienen, obligando a esos hoteles u hostales a mantener la gestión manual o incluso las reservas telefónicas como único medio para poder desarrollar su actividad. Por otra parte, los hoteles que utilizan estos métodos, no alcanzan la visibilidad necesaria para poder tener un número de clientes que generen los beneficios suficientes. También hay que destacar que el personal de limpieza de estos hoteles, no tiene el listado de habitaciones ocupadas en tiempo real, lo que dificulta su trabajo.

Se hace necesario por tanto, la existencia de un sistema que proporcione servicio a los hoteles de pequeño tamaño y a los usuarios, permitiendo que:

- Diferentes hoteles de pequeño tamaño aumenten su visibilidad, ofertando sus diferentes habitaciones.
- Usuarios, que antes realizaban las reservas de forma telefónica, puedan pasar a utilizar las reservas online, agilizando así la comunicación entre los hoteles y los clientes.
- Mejorar la comunicación interna entre los componentes de los hoteles.

Por todo lo aquí presentado, se puede concluir la necesidad de un apoyo más completo, como una aplicación web para resolver esta problemática.

1.2 Alcance y objetivos

El objetivo principal de este proyecto es el desarrollo de una aplicación web que permita mostrar en un mismo lugar los hoteles de pequeño tamaño que hay disponibles, así como sus habitaciones. Adicionalmente, la aplicación va a permitir dar visibilidad a las pequeñas empresas del sector hostelero.

Esta aplicación va a permitir el registro de hoteles de pequeño tamaño y de usuarios individuales o clientes. Por una parte, los hoteles van a poder añadir las habitaciones que tienen disponibles, así como los productos y servicios de los que disponen para consumo del cliente. Además, mostrarán en cada uno de los hoteles, su listado de precios de los diferentes tipos de habitaciones de que disponen.

Por otro lado, el usuario individual o cliente, va a poder realizar reservas sobre el hotel que desee, entre los disponibles dentro de la aplicación, y adquirir los productos o servicios de los que este disponga.

Otro objetivo mayoritario es facilitar la comunicación entre los diferentes grupos de empleados del hotel, como por ejemplo recepción y el equipo de limpieza.

Se pretende que la aplicación permita el cambio, mediante la inclusión de nuevas funcionalidades, construyendo así un software más robusto y sólido. Mediante la aplicación de buenas prácticas y patrones de diseño, se intentará facilitar el proceso de evolución, minimizando el coste del mantenimiento.

Capítulo 2

Estado del arte

En la actualidad, existe diferente software empresarial con licencia tanto privativa como *opensource*, que ayuda a mantener un control sobre las reservas de las habitaciones a los hoteles. En estos productos software, el principal inconveniente para los hoteles de pequeño tamaño, es el elevado precio de las licencias, además de que solo cubren parcialmente la carga de trabajo que se genera en el momento de realizar un *check in*. Por otro lado, la interacción de los usuarios con este tipo de webs, es limitada, ya sea por el desconocimiento del producto o por la falta de experiencia de los hoteles.

Una de las aplicaciones que podemos destacar es, *Roomba*[1], que realiza la gestión de las reservas de los hoteles, sin importar el tamaño del mismo. Esta aplicación, tiene su código fuente disponible en Internet.

También cabe destacar a *Sirvoy* [2], software de reservas hoteleras y gestión de propiedades funcional en todo tipo de acomodaciones, como pueden ser hoteles, moteles, hostales o casas de huéspedes. Además esta opción contiene un motor de reservas integrable en la web del cliente. La apariencia de esta aplicación se puede ver en la figura 2.1.

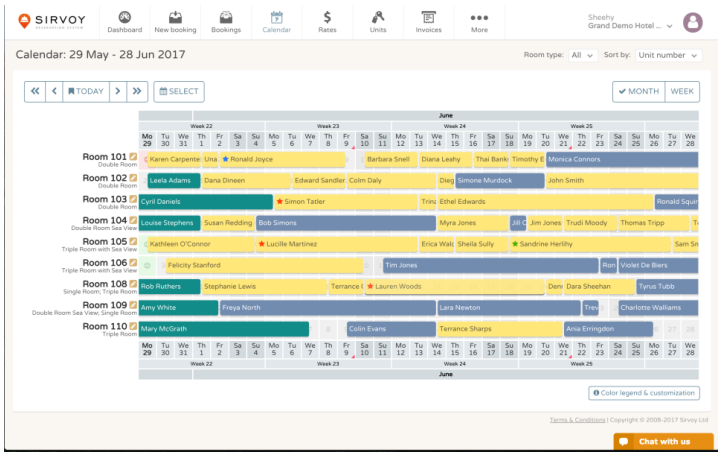


Figura 2.1: Apariencia Sirvoy

Por otro lado, nos encontramos con un software de fabricación nacional como es *GreenHotel* [3], el cual se encarga solo de la gestión de las reservas, sin hacerse cargo de las tareas propias de un hotel como puede ser la realizada por el equipo de limpieza ni tampoco ofrece los servicios y productos en un mismo punto de la aplicación.

Figura 2.2: Apariencia Green Hotel

Además de *GreenHotel*, otro software de creación nacional, es *Timon Hotel* [4], que a diferencia de este último, se encarga de la gestión completa de los hoteles, contando con una versión accesible desde cualquier punto geográfico sin necesidad de permanecer en el local. A mayores, se encarga de la gestión de los productos y servicios que un cliente adquiere durante su estancia.

Otro software a destacar es el software de *Admin Tour* [5], el cual al igual que el resto es un software de gestión hotelera, pero se diferencia por su modularidad y poder ser gestionado 100% online. Su apariencia se puede ver en la figura 2.3.

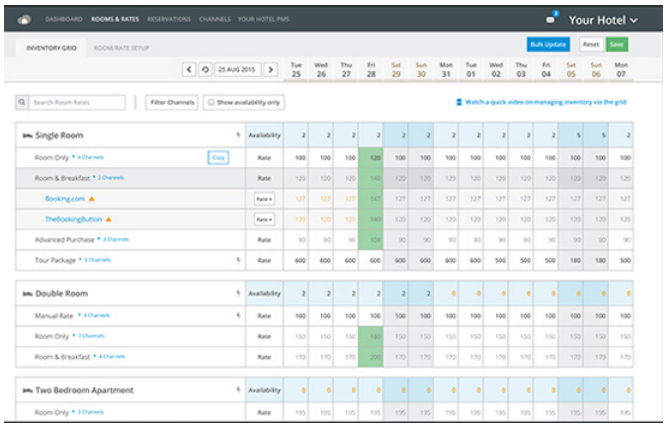


Figura 2.3: Apariencia Admin Tour

Por todo lo comentado anteriormente se decidió crear esta aplicación web para principalmente:

- Favorecer que hoteles de pequeño tamaño puedan dar el salto al mundo *online*.
- Facilitar el trabajo de los diferentes componentes del equipo de un hotel
- Ahorrar tiempo en la búsqueda de alojamientos en zonas parcialmente remotas o con pocos hoteles, que no reciben los suficientes ingresos para poder acceder a los contratos de las grandes empresas.

Capítulo 3

Base Tecnológica

En las siguientes secciones se describen los lenguajes y tecnologías usadas para llevar a cabo el proyecto, además de las herramientas empleadas para dicha tarea.

3.1 Lenguajes

A continuación se describen los lenguajes utilizados durante el desarrollo del proyecto:

Java [6]: Lenguaje de propósito general orientado a objetos. Permite el desarrollo de aplicaciones independientes de la arquitectura, gracias al motor de ejecución JRE (*Java Runtime Environment*), que permite ejecutar código fuente, ya compilado, en los diferentes entornos.

SQL [7]:*(Structured Query Language)*, es un lenguaje de programación diseñado para obtener, actualizar, y calcular información en bases de datos relacionales.

JPQL [8]: *Java Persistence Query Language*, se trata de un lenguaje de consulta utilizado por JPA (*Java Persistence API*) para obtener información de una base de datos. Permite la realización de consultas referenciando entidades del modelo de objetos de la aplicación en lugar de tablas (como puede ser el caso de SQL).

HTML [9]: *HyperText Markup Language*, es un lenguaje de marcado utilizado para determinar la estructura y contenido de las diferentes páginas de un sitio web.

CSS [10]: *Cascade Style Sheets*, es un lenguaje de estilos, que permite determinar la representación visual de los diferentes elementos declarados a través de un lenguaje de marcado como HTML.

JavaScript [11]: Se trata de un lenguaje de programación utilizado para proporcionar comportamiento dinámico a páginas web. Es un lenguaje interpretado, por lo tanto no necesita

ser compilado previamente. *JavaScript*, permite añadir una gran cantidad de características a un sitio web, como puede ser *routing* entre diferentes componentes, conexión con servidores o seguridad. Además, permite modificar dinámicamente los elementos visuales de una aplicación.

3.2 Frameworks y librerías

A continuación se describen los *Frameworks* y librerías utilizadas en la realización de este proyecto:

3.2.1 Core

Para el núcleo de la aplicación se ha usado:

SpringBoot [12]: Herramienta desarrollada para simplificar el proceso de creación de aplicaciones basadas en *Spring*. SpringBoot reduce la configuración necesaria para la ejecución del proyecto. Para ello autoconfigura todos los aspectos de la aplicación, minimizando las tareas de este tipo que debe realizar el desarrollador.

Spring [13]: Framework, de código abierto, para el desarrollo de aplicaciones Java. Dentro de este cabe destacar:

- Spring-Data[14]: Framework que tiene como objetivo simplificar la persistencia de datos contra distintos repositorios de información.
- Spring-Web[15] : Anotaciones para vincular solicitudes a controladores y métodos de manejo, así como para vincular parámetros de solicitud a argumentos de métodos.

3.2.2 Web

Para el desarrollo de la parte cliente se ha utilizado:

React [16]: Librería de *JavaScript* empleada para facilitar la construcción de interfaces de usuario interactivas. React se basa en la creación de componentes con estado y comportamiento propio. La construcción de aplicaciones se lleva a cabo mediante la creación de componentes complejos que se comunican entre sí y cuya apariencia varía en función del estado. Una de las características más beneficiosa de esta librería, es que simplifica el proceso de reutilización de componentes de forma que agiliza el proceso de desarrollo. Existen una gran variedad de librerías que complementan a React y que proporcionan componentes que reducen el tiempo de desarrollo. Las librerías más importantes utilizadas en este proyecto son:

  - react-intl [17]: Librería que proporciona un conjunto de componentes de React y una *API* que permite internacionalizar aplicaciones desarrolladas con React.
  - react-router-dom [18]: Proporciona componentes que permiten redirección entre componentes de una aplicación desarrollada con React.

Bootstrap [19]: Librería de *HTML, CSS* y *JavaScript* que aporta elementos visuales con estilo y comportamiento preestablecido, de forma que agiliza el proceso de desarrollo de sitios web.

Redux [20]: Contenedor de estado de aplicaciones que se puede utilizar en combinación con React o cualquier otro lenguaje de generación de vistas, que permite mantener un estado fácilmente accesible por los diferentes componentes de la aplicación, facilitando así la comunicación entre estos. Además, facilita el proceso de gestión del estado global de una aplicación.

Font Awersome Icons [21]: *Framework* de iconos vectoriales y estilos *css*.

3.2.3 Pruebas

Para probar el correcto funcionamiento del código se ha utilizado:

JUnit [22]: *Framework* basado en Java que permite la realización de pruebas, unitarias y de integración, sobre los distintos componentes de un sistema en desarrollo.

Insomnia [23]: Herramienta que permite el envío de peticiones o *requests* HTTP sobre una API *Rest*. Especialmente útil para el desarrollo de pruebas sobre una API Rest.

3.3 Herramientas de desarrollo

A continuación se muestran las herramientas de desarrollo utilizadas para la realización de este proyecto:

Apache Maven [24]: Herramienta de software utilizada para la gestión y construcción de proyectos *Java*. Proporciona un modelo de ejecución de proyectos simple, basado en un fichero XML (*Extensible Markup Language*) en el que se describen las características del software a construir así como sus dependencias y el orden de construcción de sus elementos. Su motor de ejecución accede a repositorios de los que se descargan las dependencias necesarias, simplificando el proceso de construcción y ejecución del código fuente. Permite gestionar proyectos software completos, desde la comprobación de que el código es correcto hasta que se despliega la aplicación, pasando por la ejecución de pruebas,

generación de informes y documentación. Todo esto se lleva a cabo mediante una serie de etapas de ejecución configurables de forma sencilla.

Git [25]: Sistema de control de versiones que facilita el proceso de seguimiento de cambios de los ficheros de un proyecto software. Se trata del software de control de versiones más extendido.

Eclipse [26]: IDE o entorno de desarrollo integrado que permite el desarrollo de aplicaciones en diferentes lenguajes. Proporciona una serie de módulos que permiten reducir el tiempo de desarrollo entre otras funcionalidades. Permite depurar el proyecto de forma que se reduce el tiempo de localización de errores, proporciona validación de código y autocompletado y se integra con otros sistemas como *Git*. También permite incluir una gran variedad de nuevas características mediante la instalación de *plugins*.

Visual Studio Code [27]: Se trata de otro entorno de desarrollo integrado (*IDE*), similar a Eclipse, que soporta muy bien el desarrollo con lenguaje *JavaScript*, entre otros.

Yarn [28]: Conjunto de herramientas para el desarrollo de *JavaScript*. Permite descargar una gran variedad de paquetes desarrollados por la comunidad y gestionar las dependencias de los proyectos. Además permite automatizar tareas rutinarias de desarrollo.

Babel [29]: Herramienta que permite transformar código *JavaScript* de última generación a un código que cualquier navegador o versión de *Node.js* pueda entender.

3.4 Servidores Web

Las tecnologías para la ejecución del proyecto varían entre la fase de desarrollo y la fase de producción:

- En **desarrollo**, la aplicación servidora se ejecuta a través de *SpringBoot*, que proporciona por defecto un servidor web *Tomcat*[30] embebido. El código *JavaScript* de la parte cliente, es proporcionado por el entorno *NodeJs*, que se apoya en *Webpack* [31], para generar los archivos *HTML*. Este código es descargado y ejecutado por un navegador web.

- En **producción**, la parte de la aplicación servidora, se ejecuta dentro de un servidor *Apache Tomcat*. En el *frontend*, después de compilar los archivos que conforman la aplicación web, se cuenta con los ficheros necesarios para el despliegue en cualquier servidor capaz de servir archivos *HTML* y *JavaScript*.

3.5 Sistemas de gestión de base de datos

El sistema de gestión de bases de datos utilizado en este proyecto es:

MySQL [32]: Sistema gestor de base de datos muy extendido y utilizado por su seguridad, escalabilidad, velocidad y facilidad de instalación y uso. Se trata del sistema de gestión de base de datos escogido para el despliegue final de la aplicación.

3.6 Protocolos

Los protocolos usados en la aplicación son:

HTTP [33](*HyperText Transfer Protocol*): Es un protocolo de nivel de aplicación que permite la transmisión de datos por la red siguiendo un modelo cliente-servidor.

REST : Es un paradigma de desarrollo de servicios web, orientados a recursos y que con frecuencia se implementan sobre el protocolo HTTP utilizando JSON como formato de transferencia de datos.

3.7 Motores de Ejecución

Los motores de ejecución usados en esta aplicación son:

Para el desarrollo de la parte servidora, se ha utilizado el *JRE* (*Java Runtime Environment*), que es un conjunto de utilidades que permiten el diseño y ejecución de las aplicaciones Java.

Para el desarrollo de la interfaz web, se ha utilizado el motor de ejecución de *node.js*, el cual está basado en el motor *opensource* de JavaScript de Google, que está diseñado para correr en un navegador y ejecutar el código *javascript* de una forma extremadamente rápida y eficiente.

Capítulo 4

Análisis de viabilidad

Antes de comenzar la realización del proyecto, se hizo un análisis de viabilidad para saber si era posible llevarlo a cabo, teniendo en cuenta las dimensiones temporales, económica y tecnológica.

En primer lugar, se debe analizar el **aspecto económico**, dado que al tratarse de un trabajo de fin de grado, los recursos disponibles son escasos. En este caso los recursos necesarios son principalmente, un ordenador y acceso a Internet. La disponibilidad de ordenador propio, solventa el coste de la obtención del mismo. Por otro lado, el acceso a Internet, tiene un coste muy reducido. En lo que a tecnologías se refiere, no supone un coste adicional al ser de tipo *open source*. Se puede determinar por tanto, que el aspecto económico no será un inconveniente para dudar de la viabilidad del proyecto.

Por otro lado, teniendo en cuenta la premisa de que es un proyecto de un trabajo fin de grado, el **aspecto temporal** no supone un inconveniente, puesto que la cantidad de funcionalidades a desarrollar, se adaptan para que sea posible desarrollarlas en el tiempo establecido, siendo un coste de trabajo equivalente a dos asignaturas de grado. Teniendo en cuenta lo citado, podemos concluir que el aspecto temporal no será un problema para la realización del proyecto.

Por último, es necesario determinar la **disponibilidad de tecnologías** que permitan la correcta realización del proyecto. En este caso, se plantea el desarrollo de una aplicación web, con dos partes claramente diferenciadas. Por una parte se requiere el desarrollo de una aplicación servidora que gestione la lógica de negocio, consulte y modifique la información de las bases de datos y ofrezca una interfaz con las funcionalidades planteadas. Para esto, analizando diferentes posibilidades, se ha llegado a la conclusión de que *SpringBoot*, ofrece la solución completa al problema planteado. Por otra parte, se requiere desarrollar una aplicación cliente, que utilizando la interfaz proporcionada por la parte servidora, le envíe peticiones a esta a partir de las acciones del usuario. Existen múltiples tecnologías que permiten este desarrollo. En este caso hemos optado por *React*, dado que ofrece una buena solución por su flexibilidad,

alcance y abundancia de librerías de desarrollo. Ambas se tratan de tecnologías ampliamente utilizadas para la creación de aplicaciones web, por lo que no suponen mayor riesgo que el derivado de la formación en estas tecnologías. Adicionalmente, relacionado con el aspecto tecnológico, se requiere un sistema de gestión de bases de datos para almacenar la información con la que trabajará la aplicación. Se ha determinado que *MySQL* es una buena opción por su fácil utilización, la abundancia de documentación y por ser uno de los más extendidos.

En definitiva, tras haber analizado los principales puntos que permiten determinar la viabilidad del proyecto, se ha llegado a la conclusión que no existe ningún inconveniente para la correcta realización del mismo en tiempo y coste, teniendo en cuenta los recursos disponibles y las restricciones que aplican en esta ocasión.

Capítulo 5

Introducción al desarrollo realizado

5.1 Introducción

En este trabajo de fin de grado se propone el diseño e implementación de una aplicación web que permita mejorar los procesos de gestión de hoteles de pequeño tamaño, así como el proceso de reserva de habitaciones. En el capítulo anterior se han enumerado las diferentes tecnologías y herramientas que han sido utilizadas durante el desarrollo del trabajo. En este capítulo se introduce la arquitectura de la solución indicando las tecnologías usadas en cada una de las capas. También se introduce la metodología utilizada y la división principal en iteraciones realizada.

5.2 Tecnologías

En esta sección, se comentan las tecnologías empleadas para el desarrollo del proyecto en base al diagrama de arquitectura de la aplicación que se muestra en la figura 5.1.

Antes de comenzar con las tecnologías utilizadas, cabe destacar que la aplicación desarrollada cuenta con dos partes claramente diferenciadas, una parte servidora o *backend*, encargada de realizar una serie de operaciones sobre diferentes fuentes de datos, y una parte cliente o *frontend*, encargada de realizar las peticiones al servidor y mostrar las respuestas de este.

Empezamos el repaso de las tecnologías aplicadas en el desarrollo del proyecto por la parte del servidor. El IDE empleado para el desarrollo fue Eclipse [26] y para poner a funcionar la capa del *backend* se usó el servidor *Tomcat* embebido dentro de *SpringBoot* [12]. Para la construcción del módulo, se empleó la herramienta software de Maven [24], diseñada para gestionar, empaquetar y construir proyectos Java, como es este. Como se indica en el diagrama, esta parte fue implementada mediante el framework *SpringBoot* para desarrollar toda la lógica del sistema. En primer lugar, se implementan las entidades que modelan la información a manejar por la aplicación, relativa a usuarios, hoteles, habitaciones o reservas entre otros

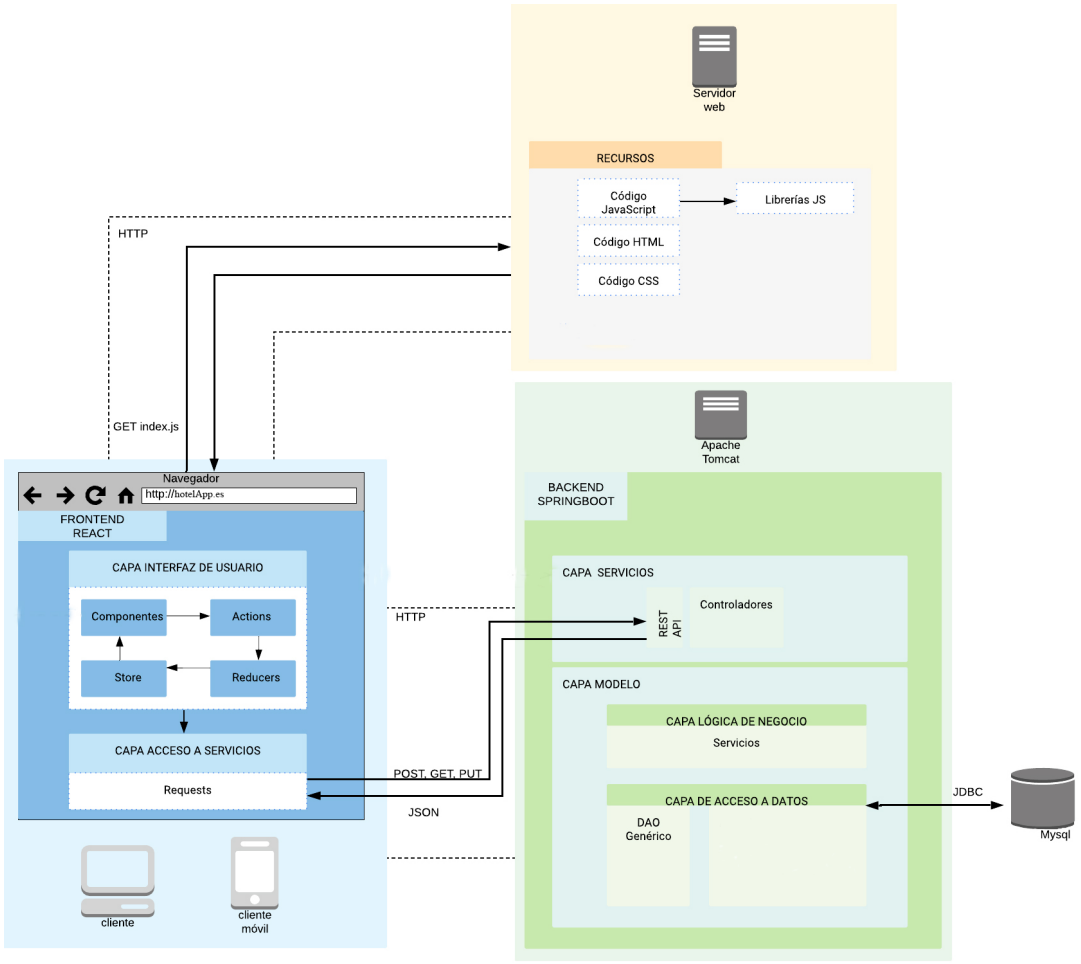


Figura 5.1: Diagrama de Arquitectura General

que van a persistir dentro de la Base de Datos mediante los DAOs(Objetos de Acceso a Datos), clases encargadas de la abstracción del acceso a las fuentes de datos. La BBDD (Base de Datos) usada es MySQL, que se accede mediante un conector *JDBC* [34]. Spring Data JPA [35] es el encargado de la comunicación con el Sistema de Gestión de Bases de Datos, que además ofrece una implementación para operaciones fundamentales sobre las entidades como son la creación, lectura, borrado o actualización. Para operaciones más complejas sobre la BBDD, ha sido necesaria la creación de consultas más específicas mediante *JPQL* [8]. Apoyándonos en esta API de persistencia, se desarrolló la capa de lógica de negocio, donde se implementan todas las operaciones de las que va a disponer la aplicación, que trabajan directamente con los DAOs. Para proporcionar acceso remoto a estas funcionalidades, se ha definido una capa adicional de servicios web. Esta última que recibe las peticiones, es una API REST, es decir, una interfaz entre sistemas que se comunican mediante el protocolo HTTP. Esta capa está formada por diferentes controladores, divididos según la funcionalidad que manejan, los cuales reciben

las peticiones realizadas por un cliente, acceden a la capa modelo a la espera de una respuesta que se enviará en un formato correcto.

Antes de comenzar a explicar el desarrollo del cliente, hay que destacar que este se trata de una SPA [36] (*Single Page Application*), es decir, una página web que contiene todo el código html, donde se puede navegar entre los componentes mediante la variación de la ruta de acceso, cargando los datos que se necesitan en cada instante de forma dinámica. Esta parte fue llevada a cabo mediante el *framework* para aplicaciones web *React* [16]. El IDE empleado para este desarrollo fue *Visual Studio Code* [27]. Como se puede apreciar en la imagen 5.1, el cliente obtiene los recursos del *frontend* y después, estos se comunicarán con la *API REST* para la realización de las operaciones necesarias en el *backend*. La capa web, sigue la arquitectura de *React*, *framework* basado en *JavaScript*[11] que permite la creación de componentes reutilizables, que mantienen su propia lógica y estado. Por lo tanto, para el desarrollo de las funcionalidades, ha sido necesaria la creación de componentes que reciban las entradas del usuario, hagan las peticiones necesarias a la parte servidora y muestren el resultado correspondiente. Como se ha dicho, muchos de estos componentes realizan peticiones a la aplicación servidor (*backend*). Para esto es necesario declarar funciones que realicen estas peticiones. Más concretamente, estas funciones realizan peticiones *HTTP* de tipo POST, GET, PUT o DELETE a la API REST del *backend*. En caso de ser necesario enviar información al servidor, esta es encapsulada en formato *JSON* [37] y enviada en la petición. Cabe destacar que, en alguna ocasión, diferentes componentes necesitan compartir cierta información para operar correctamente. Para solventar este problema se ha utilizado *Redux* [20], que permite desacoplar el estado global de una aplicación del apartado visual. Este utiliza unos elementos, conocidos como reductores, para mantener el estado compartido entre ciertos componentes. Para modificar este estado desde los componentes es necesario lanzar ciertas funciones (acciones). Para mejorar la apariencia de la interfaz web, se utilizó la biblioteca de *Bootstrap* [19] para los estilos y *Font Awersome Icons* para los iconos. Para todo el proyecto, como sistema de control de versiones se ha utilizado *Git*, concretamente *GitHub*.

5.3 Metodología e iteraciones

Como marco de desarrollo se ha utilizado una metodología basada en PUDS [38] (Proceso Unificado de Desarrollo de Software). Esta metodología facilita el desarrollo de una arquitectura comprensible, que permite la reutilización y los cambios en un futuro. Este método de desarrollo se guía por casos de uso, determinando el proceso a seguir para satisfacer los requisitos del cliente o usuario final. Se trata de una metodología iterativa e incremental basada, por tanto, en el desarrollo por iteraciones de las funcionalidades del sistema. El desarrollo de las distintas funcionalidades se ha dividido en 5 iteraciones incrementales precedidas por una

especificación de requisitos y formación. A continuación se determinan, de forma general, las tareas y funciones llevadas a cabo en cada etapa:

- Iteración 0: Especificación de requisitos y formación en las tecnologías utilizadas.
- Iteración 1: Registrar y modificar los perfiles de usuarios.
- Iteración 2: Añadir, modificar, consultar y eliminar los hoteles y las habitaciones de estos.
- Iteración 3: Añadir, modificar y consultar las reservas de cada usuario y hotel.
- Iteración 4: Registrar clientes y asignar las habitaciones a cada reserva.
- Iteración 5: Añadir, modificar, consultar y comprar productos o servicios del hotel.

En el capítulo 6 se analizan más en detalle cada una de las iteraciones comentadas.

Capítulo 6

Planificación y análisis de costes

Para la realización del proyecto se ha empleado la metodología PUDS (Proceso Unificado de Desarrollo Software). Según la aplicación de este modelo, se ha dividido el desarrollo de las funcionalidades en cinco etapas o iteraciones precedidas de una etapa adicional de especificación de requisitos y formación. La división del trabajo se ha llevado a cabo de tal forma que cada una de las iteraciones cuenta con tiempo similar. En las primeras iteraciones se llevan a cabo las funcionalidades más relevantes, dejando las de menor riesgo para las últimas etapas. El proceso de desarrollo de cada etapa, consiste en: Análisis de las funcionalidades a desarrollar, implementación, pruebas y documentación, aunque esta última, con diferente alcance en cada una de las iteraciones. Tras finalizar cada iteración se lleva a cabo una reunión con el jefe de proyecto para analizar las funciones desarrolladas y el programa del proyecto. A continuación, se detalla cada iteración:

Iteración 0: Se establecen los requisitos del sistema y las funcionalidades a desarrollar. Además se lleva a cabo un periodo de formación en las diferentes tecnologías y herramientas necesarias para el desarrollo.

Iteración 1: Se genera el proyecto *SpringBoot* base. Se realiza la configuración básica para que utilice *MySQL* como base de datos. En esta etapa, las funcionalidades a desarrollar son principalmente la creación y modificación de perfiles de usuario, personal del hotel, gerentes de los hoteles y el administrador de la aplicación. Las tareas concretas llevadas a cabo en esta iteración se pueden ver en la figura 6.1.

Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Prec	Nomb
	Definir Casos de Uso	24 horas	lun 01/06/20	mié 03/06/20		Analis
	Generar Especificación de Requisitos Software	8 horas	jue 04/06/20	jue 04/06/20	1	Analis
	Formación	32 horas	lun 01/06/20	jue 04/06/20		Desari
	v.1.0	**9 días**	**vie 05/06/20**	**mié 17/06/20**		
	Generar Proyecto SpringBoot	4 horas	vie 05/06/20	vie 05/06/20	3	Desari
	Capa Backend	**3,5 días**	**vie 05/06/20**	**mié 10/06/20**		
	Diseño de entidades	4 horas	vie 05/06/20	vie 05/06/20	5	Analis
	Crear Entidades	4 horas	lun 08/06/20	lun 08/06/20	7	Desari
	Crear Servicio Registro Usuarios	4 horas	lun 08/06/20	lun 08/06/20	8	Desari
	Crear Servicio Inicio de Sesion	4 horas	mar 09/06/20	mar 09/06/20	9	Desari
	Crear Servicio de Cambio de Password	4 horas	mar 09/06/20	mar 09/06/20	10	Desari
	Crear Servicio Actualizar Información	4 horas	mié 10/06/20	mié 10/06/20	11	Desari
	Probar Servicio de Usuarios	4 horas	mié 10/06/20	mié 10/06/20	12	Desari
	Generar Proyecto React	16 horas	jue 11/06/20	vie 12/06/20	6	Desari
	Capa Web	**3 días**	**lun 15/06/20**	**mié 17/06/20**		
	Crear Componentes Registro	8 horas	lun 15/06/20	lun 15/06/20	13;14	Desari
	Crear Componentes Inicio Sesion	8 horas	mar 16/06/20	mar 16/06/20	16	Desari
	Crear Componentes Modificar Perfil	8 horas	mié 17/06/20	mié 17/06/20	17	Desari

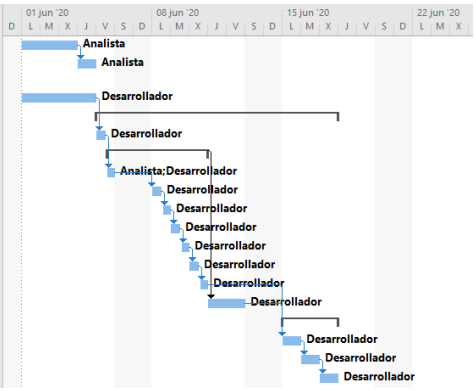


Figura 6.1: Iteraciones 0 y 1

Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Prec	Nomb
	v.2.0	**11,63 días**	**jue 18/06/20**	**vie 03/07/20**		
	Analisis de Funcionalidades	8 horas	jue 18/06/20	jue 18/06/20	18	Analis
	Capa Backend	**6,75 días**	**vie 19/06/20**	**lun 29/06/20**		
	Diseño de entidades y servicios	5 horas	vie 19/06/20	vie 19/06/20	20	Analis
	Construccion de entidades	4 horas	vie 19/06/20	lun 22/06/20	22	Desari
	Servicio Hotel	**2,25 días**	**lun 22/06/20**	**mié 24/06/20**		
	Añadir Hotel	4 horas	lun 22/06/20	lun 22/06/20	23	Desari
	Modificar Hotel	4 horas	lun 22/06/20	mar 23/06/20	25	Desari
	Buscar Hotel	4 horas	mar 23/06/20	mar 23/06/20	26	Desari
	Eliminar Hotel	6 horas	mar 23/06/20	mié 24/06/20	27	Desari
	Servicio Habitaciones	**2,38 días**	**mié 24/06/20**	**vie 26/06/20**		
	Añadir Habitacion	4 horas	mié 24/06/20	mié 24/06/20	24	Desari
	Buscar Habitación	4 horas	mié 24/06/20	jue 25/06/20	30	Desari
	Modificar Habitación	5 horas	jue 25/06/20	jue 25/06/20	31	Desari
	Eliminar Habitación	6 horas	vie 26/06/20	vie 26/06/20	32	Desari
	Prueba de Integración de Servicios	8 horas	vie 26/06/20	lun 29/06/20	29	Desari

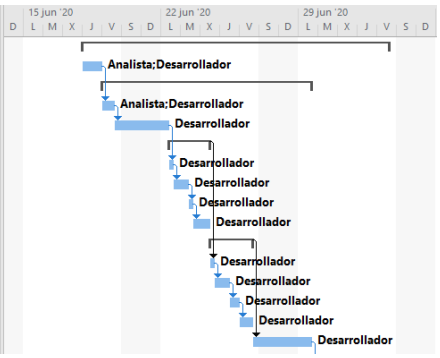


Figura 6.2: Iteración 2. Parte Backend

Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Prec	Nomb
	Capa Web	**3,88 días**	**lun 29/06/20**	**vie 03/07/20**		
	Crear Componentes Añadir Hotel	6 horas	lun 29/06/20	mar 30/06/20	34	Desari
	Crear Componentes Visualizar Hoteles	6 horas	mar 30/06/20	mié 01/07/20	36	Desari
	Crear Componentes Detalle Hotel	7 horas	mié 01/07/20	jue 02/07/20	37	Desari
	Crear Componentes Añadir Habitación	6 horas	jue 02/07/20	jue 02/07/20	38	Desari
	Listado Habitaciones Ocupadas	6 horas	jue 02/07/20	vie 03/07/20	39	Desari

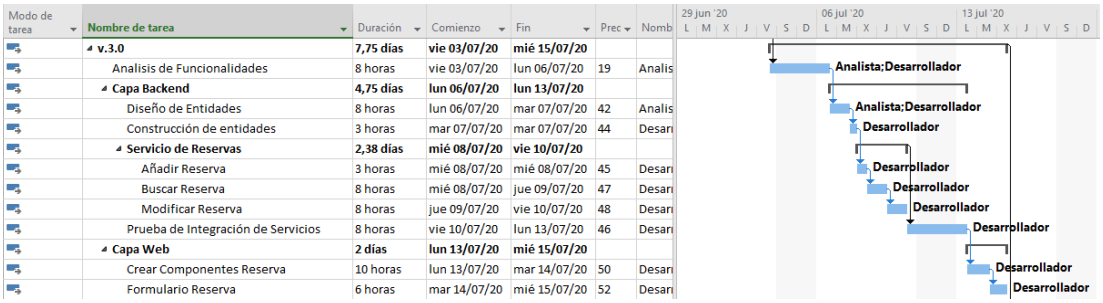
Figura 6.3: Iteración 2. Parte Frontend

Iteración 2: Como se puede observar en las figuras 6.2 y 6.3, en la segunda iteración, las tareas llevadas a cabo permiten al perfil de administrador, añadir, modificar y eliminar hoteles, y al perfil de gerente del hotel añadir las habitaciones correspondientes. Todo usuario, autenticado o no, puede ver el listado de hoteles disponibles, mientras que si el usuario está autenticado, puede ver toda la información del hotel que quiera.

Iteración 3: Los usuarios registrados, pueden añadir las reservas que requieran sobre el hotel que ellos deseen, pudiendo además, modificar la información sobre estas. Todos los usuarios pueden ver su listado de reservas, es decir, si el usuario logueado es parte del personal del hotel, puede ver las reservas realizadas sobre este y si el usuario logueado no forma parte del

personal del hotel, sólo puede ver las reservas que el mismo ha realizado. Las tareas llevadas a cabo se pueden ver en la figura 6.4.

Iteración 4: Como se puede observar en la figura 6.5, las tareas realizadas en esta iteración permiten que el personal del hotel registre los clientes que reciben a partir de la reserva que previamente han realizado. Así estos datos pueden ser almacenados para posteriormente transmitirlos a la policía o al organismo correspondiente. Además, se permite asignar una o varias habitaciones a las reservas que han sido recibidas.



Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Prec	Nomb
	v.3.0	**7,75 días**	**vie 03/07/20**	**mié 15/07/20**		
	Analisis de Funcionalidades	8 horas	vie 03/07/20	lun 06/07/20	19	Analis
	Capa Backend	**4,75 días**	**lun 06/07/20**	**lun 13/07/20**		
	Diseño de Entidades	8 horas	lun 06/07/20	mar 07/07/20	42	Analis
	Construcción de entidades	3 horas	mar 07/07/20	mar 07/07/20	44	Desarı
	Servicio de Reservas	**2,38 días**	**mié 08/07/20**	**vie 10/07/20**		
	Añadir Reserva	3 horas	mié 08/07/20	mié 08/07/20	45	Desarı
	Buscar Reserva	8 horas	mié 08/07/20	jue 09/07/20	47	Desarı
	Modificar Reserva	8 horas	jue 09/07/20	vie 10/07/20	48	Desarı
	Prueba de Integración de Servicios	8 horas	vie 10/07/20	lun 13/07/20	46	Desarı
	Capa Web	**2 días**	**lun 13/07/20**	**mié 15/07/20**		
	Crear Componentes Reserva	10 horas	lun 13/07/20	mar 14/07/20	50	Desarı
	Formulario Reserva	6 horas	mar 14/07/20	mié 15/07/20	52	Desarı

Figura 6.4: Iteración 3

Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Prec	Nomb
	v.4.0	**5,63 días**	**mié 15/07/20**	**mié 22/07/20**		
	Analisis de Funcionalidades	8 horas	mié 15/07/20	jue 16/07/20	41	Analis
	Capa Backend	**3,38 días**	**jue 16/07/20**	**mar 21/07/20**		
	Diseño de Entidades	8 horas	jue 16/07/20	vie 17/07/20	55	Analis
	Construcción de entidades	4 horas	vie 17/07/20	vie 17/07/20	57	Desarı
	Servicio de Clientes	**1,13 días**	**vie 17/07/20**	**lun 20/07/20**		
	Añadir Cliente	5 horas	vie 17/07/20	lun 20/07/20	58	Desarı
	Actualizar Cliente	4 horas	lun 20/07/20	lun 20/07/20	60	Desarı
	Prueba de Integración de Servicios	6 horas	mar 21/07/20	mar 21/07/20	59	Desarı
	Capa Web	**1,25 días**	**mar 21/07/20**	**mié 22/07/20**		
	Crear Componentes Clientes	5 horas	mar 21/07/20	mié 22/07/20	56	Desarı
	Crear Formulario Cliente	5 horas	mié 22/07/20	mié 22/07/20	64	Desarı

Figura 6.5: Iteración 4

Iteración 5: En esta iteración, como se puede ver en las figuras 6.6 y 6.7, se han realizado las tareas necesarias para que los usuarios autenticados puedan ver los servicios y productos que cada hotel ofrece, siendo estos añadidos por el perfil de gerente del hotel y además, los usuarios que han realizado una reserva sobre un hotel, puedan contratar los servicios o adquirir los productos que estos ofrecen.

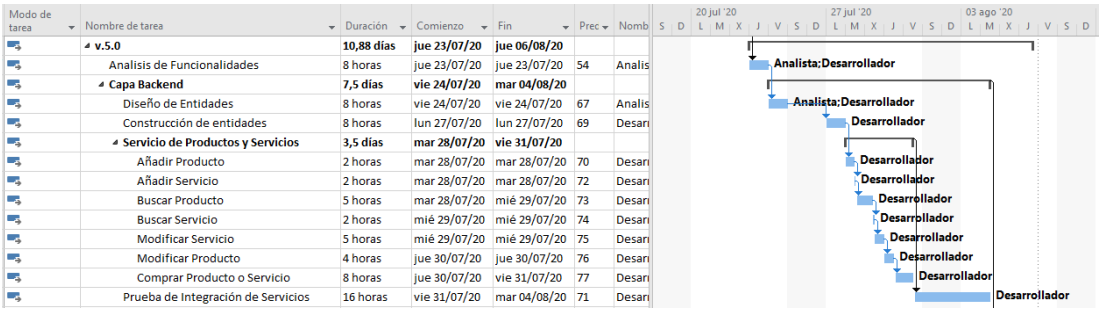


Figura 6.6: Iteración 5. Parte Backend

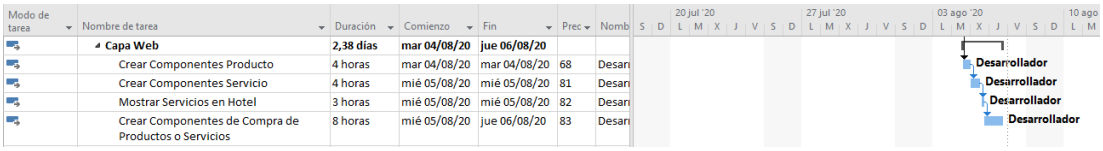


Figura 6.7: Iteración 5. Parte Frontend

Etapa	Recurso	Coste estimado	Coste Real	Desviación
Iteración 0	Analista	32 horas	32 horas	0 horas
Iteración 1	Analista	20 horas	20 horas	0 horas
	Desarrollador	61 horas	100 horas	39 horas
Iteración 2	Analista	16 horas	16 horas	0 horas
	Desarrollador	128 horas	150 horas	22 horas
Iteración 3	Analista	16 horas	16 horas	0 horas
	Desarrollador	72 horas	120 horas	48 horas
Iteración 4	Analista	16 horas	16 horas	0 horas
	Desarrollador	64 horas	80 horas	16 horas
Iteración 5	Analista	16 horas	16 horas	0 horas
	Desarrollador	120 horas	140 horas	20 horas
Total		561 horas	706 horas	145 horas

Tabla 6.1: Horas Planificadas y Consumidas

Como se puede observar en la tabla 6.1, aunque las funcionalidades fueron implementadas, el esfuerzo temporal, ha sufrido una serie de desviaciones, ya fuese por inexperiencia con las herramientas o por causas de fuerza mayor, afectando al proyecto en mayor medida esta dentro de las últimas dos iteraciones. Por lo tanto alguna de las tareas a realizar sufrió algún retraso por alguna de las causas anteriores, llegando a consumir un total de 706 horas.

Etapa	Recurso	Coste (€)
Iteración 0	Analista	384
Iteración 1	Analista	240
	Desarrollador	1000
Iteración 2	Analista	192
	Desarrollador	1500
Iteración 3	Analista	192
	Desarrollador	1200
Iteración 4	Analista	192
	Desarrollador	800
Iteración 5	Analista	192
	Desarrollador	1400
Coste Total Recursos		7292

Tabla 6.2: Coste Recursos

Respecto al **análisis de costes**, se ha elaborado una estimación del proyecto en un ámbito profesional. Para los cálculos reflejados en la tabla 6.2, se han tenido en cuenta los siguientes perfiles: el desarrollador, que realiza el proyecto, recibe un total de 10 € por hora y el analista percibe un total de 12 € por hora.

También cabe destacar que en la tabla, se realiza la suma de los salarios a percibir para obtener la cifra total de la iteración.

Concepto	Coste (€)
Desarrollo	7292
Electricidad	120
Conexión a Internet	240
Equipos Informáticos	80
Total	7732

Tabla 6.3: Costes Totales

Además del coste de desarrollo, se han tenido en cuenta los demás parámetros resaltados en la tabla 6.3. En lo respectivo al precio de los ordenadores, se ha tenido en cuenta el tiempo de uso de estos para el desarrollo, obteniendo así un precio adecuado, dado que se puede entender su uso más allá del de este proyecto gracias a que el periodo de amortización es mayor que la duración del mismo. En total el proyecto saldría por un total de **7732 €**.

Capítulo 7

Requisitos del sistema

7.1 Introducción

La aplicación aquí presentada y desarrollada, pretende dar servicio a los diferentes hoteles de pequeño tamaño, que quieran darse a conocer o dar el salto al mundo *online*. Por otra parte, también permite a los usuarios realizar las reservas de habitaciones de manera no presencial, sin tener como única forma de contacto el teléfono. A continuación se describen los requisitos de la aplicación.

Los hoteles que se registren, van a poder acceder al sistema para poner a la venta sus habitaciones, que pueden ser de diferentes tipos, como son *individuales, dobles, especiales, o triples*. También es posible realizar búsquedas de habitaciones de las que se mostrará el número, el estado, pudiendo ser este libre, ocupada, sin limpiar, o no utilizable, y el tipo. Al añadir las habitaciones, estas se crean en estado libre. Cuando se asignan a una reserva se actualiza su estado a ocupada y al realizar el *check out* en el hotel este estado pasa a ser sin limpiar. Mientras que si la habitación tiene algún problema grave que no permita su utilización, esta pasará a estado no utilizable.

Los hoteles que se han registrado previamente, se mostrarán en la página de inicio de la aplicación. Para cada uno se indican su nombre, el responsable, su dirección y el teléfono de contacto. Si el usuario que ha accedido, está autenticado en la aplicación, y accede a uno de los hoteles, va a poder ver en detalle la misma información ya conocida, pero además va a poder ver unas fotos, el listado de precios de los tipos de habitaciones, los servicios y los productos con su nombre, precio y una breve descripción. Todos estos servicios, productos y precios van a ser añadidos por los hoteles y también podrán ser adquiridos por los clientes.

Por otro lado, cuando un usuario realiza una reserva, se crea automáticamente una cuenta de gastos asociada a esta, donde el cliente va a poder almacenar los productos y servicios consumidos durante su estancia.

Además, estos hoteles, al recibir a los clientes que han hecho las reservas, van a poder

realizar todo el *check in* y asignar las diferentes habitaciones a la reserva correspondiente a través de la aplicación.

Por otro lado, el personal de los hoteles y los gerentes van a poder ver en un listado los clientes que han pasado por el hotel, mostrándose para estos su nombre, sus apellidos, un documento identificativo, un teléfono, su dirección postal y las fechas de entrada y salida del hotel.

Otra pequeña parte del personal del hotel, el encargado de la limpieza, va a poder ver en la aplicación, de una forma sencilla, las habitaciones que han sido ocupadas y cuales están libres para poder realizar su trabajo.

Los perfiles destacados de la aplicación son el admin, el manager del hotel, el personal del hotel y el usuario individual o cliente. Dependiendo del perfil de usuario que pase por el proceso de *login*, este podrá realizar una serie de acciones sobre los hoteles y habitaciones que se presentan. Si el usuario tiene el perfil o rol de *Admin*, este podrá crear las cuentas de usuario del *manager* de los hoteles, así como añadir los mismos.

Por otro lado, si el usuario tiene el rol de *manager*, este va a poder crear los productos y servicios del hotel, además de añadir y buscar las habitaciones, así como ver las reservas que hay para una fecha determinada o buscar los clientes que han pasado por el hotel. También es el encargado de subir las fotos que quiera mostrar en los detalles del hotel y añadir los precios de las habitaciones.

Otro perfil a destacar es el perfil de *personal del hotel*. Este podrá realizar las mismas acciones sobre la aplicación que el *manager*, con la excepción de que no podrá añadir productos, servicios y fotos. Este podrá actualizar el estado de las habitaciones, así como actualizar sus los precios, los productos y servicios.

Por otro lado, los clientes o usuarios que se registren van a poder ver toda la información de los hoteles disponibles, así como realizar las reservas de las diferentes habitaciones correspondientes a un hotel mediante su tipo. Los usuarios no registrados, también tienen acceso a la información de los hoteles pero no podrán realizar las reservas de manera *online*.

La aplicación debe adaptarse para poder ser usada por diferentes tipos de dispositivos.

7.2 Actores

A continuación se presentan los actores identificados en el sistema:

Usuario anónimo: Se trata de cualquier usuario que no ha pasado por el proceso de autenticación.

Usuario registrado: Se trata del usuario que ha pasado con éxito el proceso de autenticación, independientemente del rol que tenga.

Usuario individual o cliente: Se trata del usuario autenticado con el rol de usuario. Estos

tienen las mismas funcionalidades que el usuario anónimo, además de poder crear reservas sobre los hoteles que deseen.

Usuario Hotel: Se trata del usuario autenticado con rol de hotel. Estos tienen las funcionalidades de consultar y actualizar las habitaciones, actualizar la información del hotel al que pertenecen, actualizar los precios de los tipos de habitaciones, los productos y los servicios, a mayores de las funcionalidades del usuario individual, con la excepción de que estos no pueden realizar reservas.

Manager: Se trata del usuario autenticado con rol de manager del hotel. Estos tienen las funcionalidades de añadir y eliminar habitaciones, añadir y eliminar servicios y productos, crear la cuenta del personal del hotel (usuario hotel), a mayores de las funcionalidades de estos.

Admin: Se trata del usuario autenticado con el rol de *admin*. Estos son los encargados de crear las cuentas de los *manager* de los hoteles, además de crear los hoteles.

La relación entre los diferentes actores se puede ver en la figura 7.1.

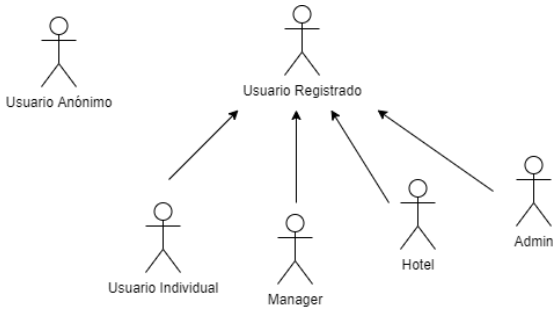


Figura 7.1: Jerarquía de actores

7.3 Casos de uso

A continuación se definen los casos de uso de esta aplicación:

CU - 01	**Registrarse**
Precondiciones	El usuario no está autenticado ni registrado
Flujo Normal	El usuario accede a la pantalla de *Iniciar Sesión* (CU-02),sigue el *link* a registro, introduce los campos requeridos y pulsa el botón de *Registrarse*.
Post-Condiciones	Se almacena la información del usuario en el sistema y este accede a la aplicación como usuario registrado.
Excepciones	El usuario no introduce la información obligatoria, el nombre de usuario ya existe o el correo electrónico no tiene el formato correcto.

CU - 02	**Iniciar Sesión**
Precondiciones	El usuario está registrado pero no autenticado
Flujo Normal	El usuario introduce el nombre y la contraseña y pulsa el botón de *Acceder*.
Post-Condiciones	El usuario accede a la aplicación como usuario registrado, con el rol correspondiente.
Excepciones	El usuario no introduce el nombre o la contraseña o alguno de los valores es inválido.

CU - 03	**Cerrar Sesión**
Precondiciones	El usuario está autenticado
Flujo Normal	El usuario selecciona la opción dentro del menú *Perfil – Cerrar sesión.*
Post-Condiciones	El usuario deja de estar autenticado en el sistema.
Excepciones	

CU - 04	**Modificar Contraseña**
Precondiciones	El usuario está autenticado y accede a la opción de *Actualizar Contraseña*
Flujo Normal	El usuario pulsa la opción *Cambiar contraseña* e introduce su contraseña antigua y la nueva dos veces para confirmar que no se equivoca.
Post-Condiciones	Se almacena la nueva contraseña del usuario en el sistema.
Excepciones	Contraseña antigua incorrecta, los dos campos de nueva contraseña no coinciden, la contraseña antigua y la nueva coinciden y la contraseña nueva no tiene el mínimo de caracteres exigidos.

CU - 05	**Actualizar Perfil**
Precondiciones	El usuario está autenticado y ha accedido a la página de *Actualizar Perfil.*
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar Información.* Automáticamente este es redirigido a la página principal de la aplicación.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar información* sin haber introducido todos los campos obligatorios o el correo electrónico no tiene un formato válido.

CU - 06	**Crear cuenta del Manager del Hotel**
Precondiciones	El usuario está autenticado y este tiene el rol de *Admin*.
Flujo Normal	El usuario accede a la opción del menú de *Crear Cuenta de Manager*, introduce los datos correspondientes y pulsa el botón de *Registrar*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria, el nombre de usuario ya existe o el correo electrónico no tiene el formato correcto.

CU - 07	**Crear cuenta del Personal del Hotel**
Precondiciones	El usuario está autenticado y este tiene el rol de *Manager*.
Flujo Normal	El usuario accede a la opción del menú de *Crear Cuenta de Empleado*, introduce los datos correspondientes y pulsa el botón de *Registrar*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria, el nombre de usuario ya existe o el correo electrónico no tiene el formato correcto.

CU - 08	**Añadir Hotel**
Precondiciones	El usuario está autenticado y este tiene el rol de *Admin*.
Flujo Normal	El usuario accede a la opción del menú de *Añadir Hotel*, introduce los datos correspondientes y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria o el nombre del hotel ya existe.

CU - 09	**Ver listado de Hoteles**
Precondiciones	
Flujo Normal	El usuario introduce los datos correspondientes para la búsqueda, mostrándose para cada una de las coincidencias, el nombre, la dirección, el teléfono de contacto y el gerente del mismo.
Post-Condiciones	
Excepciones	

CU - 10	**Ver detalle de un Hotel**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede al hotel que desea ver detallado, mostrándose el nombre, la dirección, el teléfono de contacto, el gerente del mismo, los precios de este, los productos y servicios, y una serie de fotos correspondientes al mismo.
Post-Condiciones	
Excepciones	

CU - 11	**Actualizar Información de un Hotel**
Precondiciones	El usuario está autenticado, ha accedido al *detalle del hotel* (CU - 10) y tiene el rol de *Manager u Hotel*.
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar*. Automáticamente es redirigido a la página principal de la aplicación.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar* sin haber introducido todos los campos obligatorios.

CU - 12	**Eliminar Hotel**
Precondiciones	El usuario está autenticado, este tiene el rol de *Admin* y ha accedido a la información detallada del hotel *(CU - 10)*.
Flujo Normal	El usuario accede a la opción del menú de *Eliminar Hotel*, y en el mensaje emergente, pulsa el botón de *Ok*.
Post-Condiciones	La información del hotel se elimina en el sistema.
Excepciones	El hotel no existe.

CU - 13	**Añadir Foto de Hotel**
Precondiciones	El usuario está autenticado, este tiene el rol de *Manager* y su dirección coincide con la dirección del hotel.
Flujo Normal	El usuario accede a la opción del menú de *Añadir Foto*, introduce el archivo correspondiente y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria.

CU - 14	**Añadir Precio de un tipo de Habitación del Hotel**
Precondiciones	El usuario está autenticado, este tiene el rol de *Manager* y su dirección coincide con la dirección del hotel.
Flujo Normal	El usuario accede a la opción del menú de *Añadir Precio*, introduce la información correspondiente y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema. El usuario es redirigido al detalle del hotel.
Excepciones	El usuario no introduce la información obligatoria.

CU - 15	**Añadir Producto**
Precondiciones	El usuario está autenticado y este tiene el rol de *Manager*.
Flujo Normal	El usuario accede a la opción del menú de *Añadir Producto*, introduce los datos correspondientes y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria o el producto ya existe.

CU - 16	**Ver listado de Productos**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede a la información detallada del hotel del que quiera ver los productos, introduce los datos correspondientes para la búsqueda, mostrándose para cada una de las coincidencias, el nombre y el precio.
Post-Condiciones	
Excepciones	

CU - 17	**Ver detalle de un Producto**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede al detalle del hotel que desea ver, después accede al producto de la lista y se muestra el nombre, la descripción, y el precio de este.
Post-Condiciones	
Excepciones	

CU - 18	**Actualizar Información de un Producto**
Precondiciones	El usuario está autenticado, ha accedido al *detalle del producto* (CU - 17) y tiene el rol de *Manager o Hotel*.
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar*. Automáticamente es redirigido a la página de detalle del hotel.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar* sin haber introducido todos los campos obligatorios.

CU - 19	**Eliminar Producto**
Precondiciones	El usuario está autenticado, este tiene el rol de *Manager* y ha accedido a la información detallada del producto *(CU - 17)*.
Flujo Normal	El usuario accede a la opción del menú de *Eliminar Producto*, y en el mensaje emergente, pulsa el botón de *Ok*.
Post-Condiciones	La información del producto se elimina en el sistema.
Excepciones	El producto no existe.

CU - 20	**Añadir Servicio**
Precondiciones	El usuario está autenticado y este tiene el rol de *Manager*.
Flujo Normal	El usuario accede a la opción del menú de *Añadir Servicio*, introduce los datos correspondientes y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria o el servicio ya existe.

CU - 21	**Ver listado de Servicios**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede a la información detallada del hotel del que quiera ver los servicios, introduce los datos correspondientes para la búsqueda, mostrándose para cada una de las coincidencias, el nombre y el precio.
Post-Condiciones	
Excepciones	

CU - 22	**Ver detalle de un Servicio**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede al detalle del hotel que desea ver, después accede al servicio de la lista y se muestra el nombre, la descripción, y el precio de este.
Post-Condiciones	
Excepciones	

CU - 23	**Actualizar Información de un Servicio**
Precondiciones	El usuario está autenticado, ha accedido al *detalle del servicio* (CU - 22) y tiene el rol de *Manager o Hotel*.
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar*. Automáticamente es redirigido a la página de detalle del hotel.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar* sin haber introducido todos los campos obligatorios.

CU - 24	**Eliminar Servicio**
Precondiciones	El usuario está autenticado, este tiene el rol de *Manager* y ha accedido a la información detallada del servicio *(CU - 22)*.
Flujo Normal	El usuario accede a la opción del menú de *Eliminar Servicio*, y en el mensaje emergente, pulsa el botón de *Ok*.
Post-Condiciones	La información del servicio se elimina en el sistema.
Excepciones	El servicio no existe.

CU - 25	**Ver listado de Precios**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede a la información detallada del hotel del que quiera ver los precios, mostrándose para cada una de las coincidencias, el tipo y el precio.
Post-Condiciones	
Excepciones	

CU - 26	**Ver detalle de un Precio**
Precondiciones	El usuario está autenticado y tiene el rol de *Manager u Hotel* y ha accedido al listado de precios *(CU - 25)*.
Flujo Normal	El usuario accede al precio de la lista y se muestra el tipo, y el precio de este.
Post-Condiciones	
Excepciones	

CU - 27	**Actualizar Información de un Precio**
Precondiciones	El usuario está autenticado, ha accedido al *detalle del precio* (CU - 26) y tiene el rol de *Manager u Hotel*.
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar*. Automáticamente es redirigido a la página de detalle del hotel.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar* sin haber introducido todos los campos obligatorios.

CU - 28	**Eliminar Precio**
Precondiciones	El usuario está autenticado, este tiene el rol de *Manager* y ha accedido a la información detallada del precio *(CU - 26)*.
Flujo Normal	El usuario accede a la opción del menú de *Eliminar Precio*, y en el mensaje emergente, pulsa el botón de *Ok*.
Post-Condiciones	La información del precio se elimina en el sistema.
Excepciones	El precio no existe.

CU - 29	**Ver Fotos de un Hotel**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede al detalle del hotel que desea ver, y se muestran las fotos de este.
Post-Condiciones	
Excepciones	

CU - 30	**Añadir Habitación**
Precondiciones	El usuario está autenticado y este tiene el rol de *Manager*.
Flujo Normal	El usuario accede al detalle del hotel *(CU - 10)*, después a la opción del menú de *Añadir Habitación*, introduce los datos correspondientes y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria o la habitación ya existe.

CU - 31	**Ver listado de Habitaciones**
Precondiciones	El usuario está autenticado, con el rol de *Manager o Hotel*.
Flujo Normal	El usuario accede a la opción del menú de *Habitaciones*, introduce los datos correspondientes para la búsqueda, mostrándose para cada una de las coincidencias, el número, el estado y el tipo de esta.
Post-Condiciones	
Excepciones	

CU - 32	**Ver detalle de una Habitación**
Precondiciones	El usuario está autenticado, tiene el rol de *Manager o Hotel* y ha accedido al listado de habitaciones *CU - 31)*.
Flujo Normal	El usuario accede a la habitación que desea ver de la lista, y se muestra el nombre, el tipo, y el estado de esta.
Post-Condiciones	
Excepciones	

CU - 33	**Actualizar Información de una Habitación**
Precondiciones	El usuario está autenticado, ha accedido al *detalle de la habitación* (CU - 32) y tiene el rol de *Manager o Hotel*.
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar*. Automáticamente es redirigido al listado de habitaciones.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar* sin haber introducido todos los campos obligatorios.

CU - 34	**Reservar**
Precondiciones	El usuario está autenticado y este tiene el rol de *User*.
Flujo Normal	El usuario accede al detalle del hotel *(CU - 10)*, después a la opción del menú de *Reservar*, introduce los datos correspondientes y pulsa el botón de *Reservar*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria o no hay habitaciones suficientes disponibles.

CU - 35	**Ver listado de Reservas**
Precondiciones	El usuario está autenticado, con el rol de *Manager, Hotel o User*.
Flujo Normal	El usuario accede a la opción del menú de *Reservas*, introduce los datos correspondientes para la búsqueda, mostrándose para cada una de las coincidencias, el identificador, el nombre del cliente, solo si el usuario tiene el rol de *Manager u Hotel*, el hotel, sólo si el usuario tiene el rol de *User*, la cantidad de habitaciones reservadas y las fechas de llegada y salida de esta.
Post-Condiciones	
Excepciones	

CU - 36	**Ver detalle de una Reserva**
Precondiciones	El usuario está autenticado, tiene el rol de *Manager, Hotel o User* y ha accedido al listado de reservas *CU - 35)*.
Flujo Normal	El usuario accede a la reserva que desea ver de la lista, y se muestra el identificador, el nombre del cliente, el hotel, la cantidad de habitaciones reservadas y las fechas de llegada y salida de esta.
Post-Condiciones	
Excepciones	

CU - 37	**Actualizar Información de una Reserva**
Precondiciones	El usuario está autenticado, ha accedido al *detalle de la reserva* (CU - 36).
Flujo Normal	El usuario modifica la información que desee y pulsa *Actualizar*. Automáticamente es redirigido al listado de reservas.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar* sin haber introducido todos los campos obligatorios.

CU - 38	**Eliminar Reserva**
Precondiciones	El usuario está autenticado, este tiene el rol de *Manager, Hotel o User* y ha accedido a la información detallada de la reserva *(CU - 36)*.
Flujo Normal	El usuario accede a la opción del menú de *Eliminar Reserva*, y en el mensaje emergente, pulsa el botón de *Ok*.
Post-Condiciones	La información de la reserva se elimina en el sistema.
Excepciones	La reserva no existe.

CU - 39	**Crear Cuenta de Gastos**
Precondiciones	El usuario está autenticado y este tiene el rol de *User*.
Flujo Normal	El usuario accede al detalle del hotel *(CU - 10)*, después a la opción del menú de *Reservar*, introduce los datos correspondientes y pulsa el botón de *Reservar*, al momento se dispara la opción de crear cuenta de gastos, obteniendo la información de la reserva.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria.

CU - 40	**Asignar Habitación a Reserva**
Precondiciones	El usuario está autenticado, tiene el rol de *Manager u Hotel* y ha accedido a la lista de reservas *(CU - 35)*.
Flujo Normal	El usuario accede al detalle de la reserva *(CU - 36)*, después a la opción del menú de *Asignar*, selecciona la habitación correspondiente y pulsa el botón de *Asignar*, saltando el mensaje de confirmación satisfactoria.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria.

CU - 41	**Añadir Cliente**
Precondiciones	El usuario está autenticado, tiene el rol de *Manager u Hotel* y ha accedido al detalle de la reserva *(CU - 36)*.
Flujo Normal	El usuario selecciona la opción del menú de *Añadir Cliente*, introduce los datos correspondientes y pulsa el botón de *Añadir*.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario no introduce la información obligatoria.

CU - 42	**Ver listado de Clientes**
Precondiciones	El usuario está autenticado, con el rol de *Manager u Hotel*.
Flujo Normal	El usuario accede a la opción del menú de *Clientes*, introduce los datos correspondientes para la búsqueda, mostrándose para cada una de las coincidencias, el nombre, los apellidos, el *DNI*, el teléfono de contacto, la dirección y las fechas de llegada y salida.
Post-Condiciones	
Excepciones	

CU - 43	**Añadir Producto a Cuenta de Gastos**
Precondiciones	El usuario está autenticado, con el rol de *User*.
Flujo Normal	El usuario accede al detalle del hotel *(CU -10)*, después selecciona el producto que quiere en la lista y al mostrarse el detalle de este, selecciona la cantidad y pulsa el botón de *Añadir*.
Post-Condiciones	
Excepciones	No existe el producto.

CU - 44	**Añadir Servicio a Cuenta de Gastos**
Precondiciones	El usuario está autenticado, con el rol de *User*.
Flujo Normal	El usuario accede al detalle del hotel *(CU -10)*, después selecciona el servicio que quiere en la lista y al mostrarse el detalle de este, selecciona la cantidad y pulsa el botón de *Añadir*.
Post-Condiciones	
Excepciones	No existe el servicio.

CU - 45	**Ver Cuenta de Gastos**
Precondiciones	El usuario está autenticado.
Flujo Normal	El usuario accede al detalle de la reserva *(CU - 36)*, después selecciona la opción de *Ver cuenta de Gastos*, mostrándose el precio total, y una lista con los productos y servicios que se han consumido, indicando para cada uno de estos, el nombre, la cantidad, el precio y el precio total.
Post-Condiciones	
Excepciones	

7.4 Modelo de casos de uso

Para comprender más en detalle la relación entre las diferentes funciones del sistema y los actores que las ejecutan, a continuación, se presentan diferentes modelos de caso de uso. En la figura 7.2 se muestra la relación de casos de uso comunes a todos los usuarios de la aplicación, una vez han pasado por el proceso de registro. Como se puede ver, únicamente se puede

registrar un usuario anónimo. Un usuario registrado, por otra parte, puede iniciar sesión con sus credenciales o cerrarla si se encuentra autenticado. Cualquier usuario registrado puede acceder a su información de perfil y modificarla. Estos pueden consultar los detalles de los hoteles, donde pueden ver todos los servicios y productos que ofrecen, además de las fotos. Cualquier usuario autenticado puede ver los detalles de los productos o servicios.

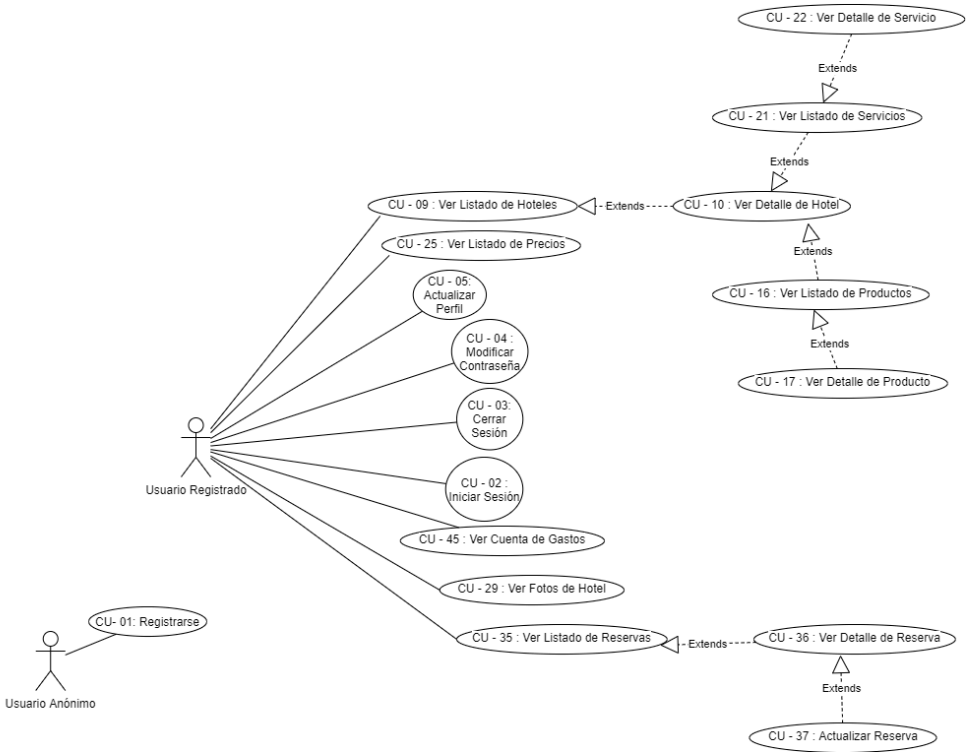


Figura 7.2: Modelo de casos de uso de usuario anónimo y los comunes a todos los registrados

Como se puede apreciar en la figura 7.3, la modificación de la información de los hoteles y habitaciones, sólo es posible para los actores con el rol de *Manager u Hotel*. Además, estos actores pueden ver los listados de clientes, así como el listado de habitaciones, el detalle de las mismas y el detalle de los precios de los tipos de habitaciones.

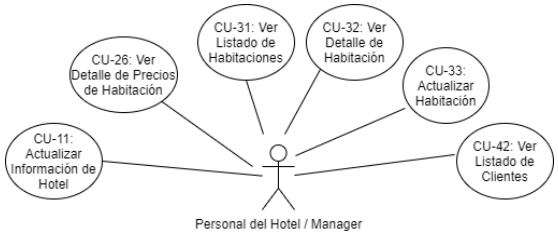


Figura 7.3: Casos de uso comunes a los perfiles de Hotel y Manager

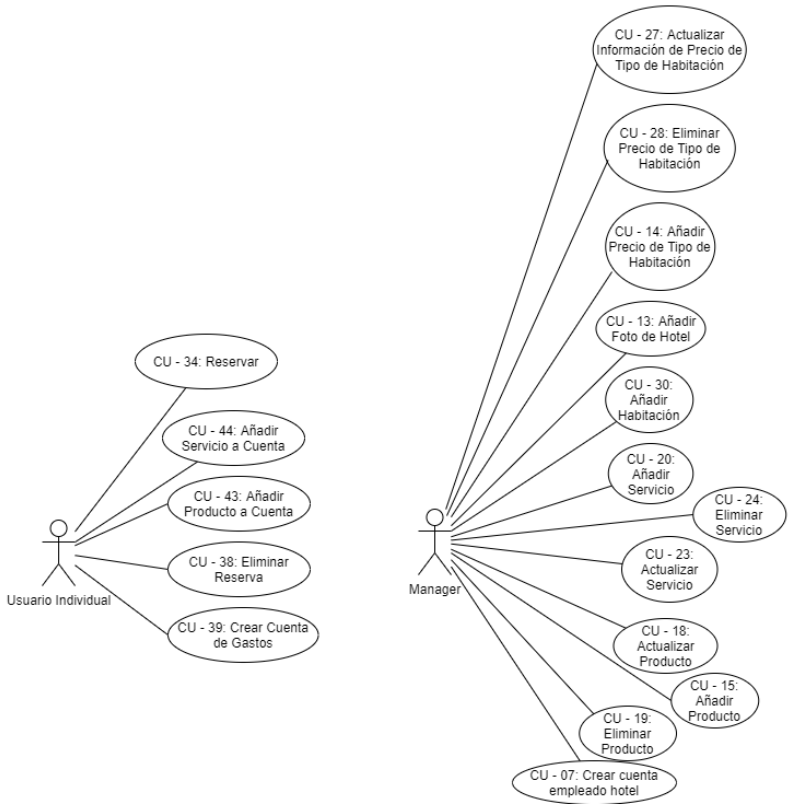


Figura 7.4: Modelo de casos de uso de Manager y usuario individual

El usuario con el rol de *Admin*, será el encargado de la creación y eliminación de los hoteles, así como de la creación de las cuentas de usuario de los *Managers* de cada hotel. Por otro lado, los usuarios con el rol de *Hotel* serán los encargados de añadir los clientes al fichero de clientes, así como de asignar una habitación a una reserva, todo esto se puede ver en la figura 7.5.

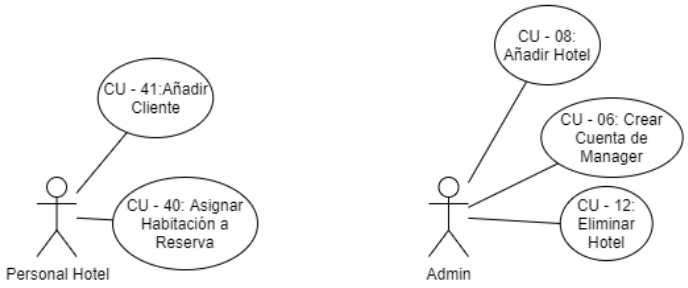


Figura 7.5: Modelo de casos de uso de Admin y Hotel

Aparte de los casos de uso comunes a los usuarios registrados, los usuarios individuales, pueden realizar reservas, crear una cuenta de gastos y adquirir productos o servicios que el

hotel tiene disponibles en cada momento, esto se puede ver en la figura 7.4 . Por otro lado, los *managers* o gerentes de los hoteles, van a poder añadir los precios de los tipos de habitaciones, los productos, servicios o las habitaciones, así como actualizarlos o eliminar los precios, productos o servicios. Además, van a poder crear la cuenta de los empleados del hotel.

7.5 Prototipado de la interfaz

Para llevar a cabo el desarrollo de las funcionalidades citadas, se ha creado previamente una serie de prototipos de alguna de las pantallas más importantes o complejas de la aplicación, que van a permitir cumplir con los requisitos de esta, así como con el diseño general de la misma.

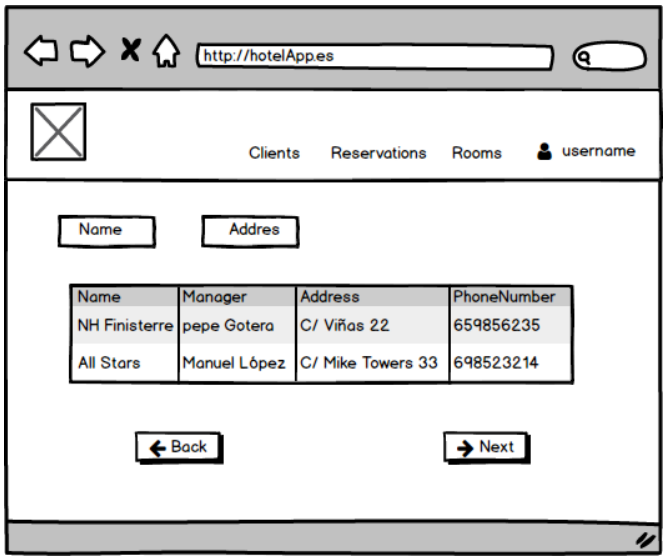


Figura 7.6: Listado de Hoteles

Al acceder a la aplicación, el usuario va a poder ver el listado de los hoteles, pudiendo filtrarlos según le interese. Este listado se puede observar en la figura 7.6. Además, se pueden ver los enlaces a las diferentes funcionalidades que un usuario tendrá dentro de la aplicación.

Una vez el usuario ha accedido al hotel que le interesa, se muestra el detalle de este, además de los precios de las habitaciones, los productos y servicios ofrecidos y una serie de fotografías del entorno exterior o interior del hotel, pudiendo verse esto en la figura 7.7.

Si el usuario desease realizar una reserva sobre este hotel, al acceder a la pantalla de realizar reserva, véase figura 7.8, el usuario puede cumplimentar los diferentes campos requeridos para que esta sea procesada correctamente.

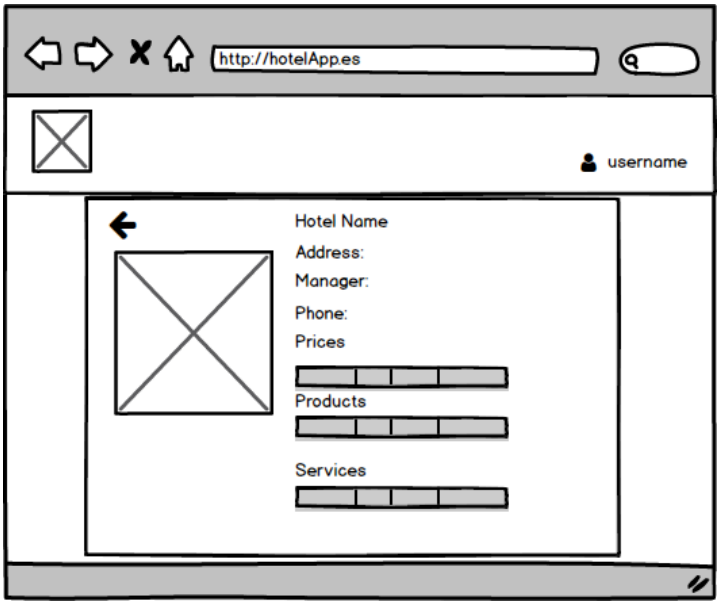


Figura 7.7: Detalle de un Hotel

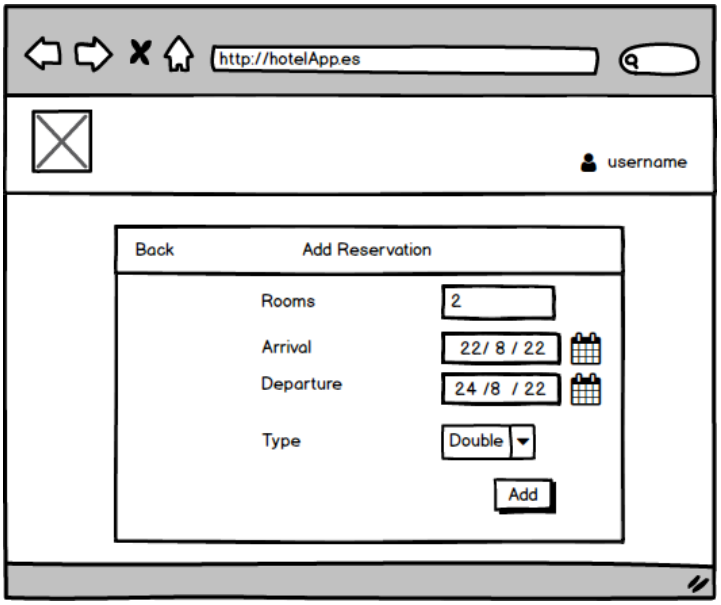


Figura 7.8: Añadir Reserva

Capítulo 8

Diseño

8.1 Introducción y objetivos

Cuando se procede a diseñar un sistema software, se determinan los componentes que lo conforman, indicando la estructura y la función de cada uno de estos. Se establecen además las diferentes iteraciones que se producen entre estos así como el medio a través del que se llevan a cabo. Durante el desarrollo de un proyecto software es inevitable la inclusión de cambios inesperados. Además, es necesario en ocasiones incluir funcionalidades que no estaban planeadas previamente en el sistema. Estas modificaciones implican un coste adicional. Un buen diseño, pretende reducir estos costes aumentando la flexibilidad y la robustez del sistema. En el software desarrollado se pretenden aplicar buenas prácticas de diseño que permitan reducir el tiempo de mantenimiento en el futuro.

Entre los objetivos que se pretenden conseguir, podemos destacar:

- Software poco acoplado, reduciendo las dependencias entre componentes.
- División de funciones de forma que aumente la facilidad de comprensión del sistema.
- Reducción de la complejidad.
- Desarrollo de software reutilizable.

8.2 Resumen de patrones usados

Para el cumplimiento de los objetivos previamente destacados, se han aplicado ciertos principios y patrones que se detallan a continuación.

Diseño por capas: Permite reducir las dependencias en el código, dividiendo el sistema en capas o niveles independientes entre sí pero comunicadas a través de unas interfaces. La

modificación en la implementación de alguna de estas capas puede hacerse sin cambios en las otras, donde cada una de estas tiene una funcionalidad diferente.

Uso de DAOs: el uso de DAOs (*Data Access Objects*) ofrece una capa más que abstrae y encapsula el acceso a las fuentes de datos, con la que se comunica la capa de lógica de negocio. En este caso proporciona una interfaz para la comunicación con una Base de Datos.

Uso de DTOs: En esta aplicación, la comunicación entre el *frontend* y el *backend* se lleva a cabo a través de la red, con los problemas inherentes que puede ocasionar por lo que es deseable realizar el menor número de peticiones posibles. Para solucionar esto, se utilizan *DTOs* (*Data Transfer Objects*) que recogen la información necesaria para una petición y la envían en un determinado formato.

Patrón Domain Model: Consiste en la creación de métodos de negocio que facilitan la implementación de capas superiores.

Uso de abstracciones: La inclusión de unos componentes en la implementación de otros se produce mediante el uso de abstracciones, evitando así referirse a implementaciones concretas. Esto es posible gracias a la inyección de dependencias que permite la creación de instancias de componentes a partir de la interfaz que estos implementan. Este método, reduce el nivel de acoplamiento entre las diferentes partes del sistema.

Adicionalmente, con el objetivo de facilitar la comprensión y el mantenimiento del sistema, se han aplicado los siguientes principios:

DRY (*Don't Repeat Yourself*): Cada parte del sistema tiene una función única y clara. La aplicación de este prinicpio provoca que cambios en una parte concreta del sistema no se propaguen hacia otras sin relación.

KISS (*Keep It Simple, Stupid!*): Reducir la complejidad del sistema para mejorar su funcionamiento y facilitar la introducción de cambios.

SOLID: Conjunto de principios que permiten un desarrollo fácil de extender.

YAGNI (*You Aren't Gona Need It*): Desarrollar funcionalidades que se vayan a usar en un futuro próximo, evitando el desarrollo de características prescindibles. Este principio permite reducir el tiempo de desarrollo, evitando la existencia de código innecesario que aumenta la complejidad del sistema.

8.3 Arquitectura general

El sistema desarrollado consta de dos partes claramente diferenciadas como se puede ver en la figura 8.1. Por una parte, cuenta con una aplicación servidor o *Backend* que se divide en tres capas. La capa de servicio, formada por una serie de controladores que agrupan funcionalidades comunes y proporciona una *API REST* que recibe las peticiones de la aplicación cliente, utilizando DTOs para el intercambio de datos. Esta redirige las peticiones a la capa de

lógica de negocio que implementa el comportamiento de cada una de las funcionalidades a través de una serie de servicios que a su vez agrupan una serie de funcionalidades interrelacionadas, mejorando la comprensión del sistema y su modificación. Esta a su vez interactúa con la capa de acceso a datos que se encarga de obtener y modificar la información de la Base de Datos, mediante la abstracción del acceso y la gestión de persistencia de los datos realizado a través de DAOs. La función general del *Backend* es, por tanto, recibir las peticiones del cliente y devolver una respuesta tras aplicar una lógica y consultar a la BD si es necesario. Por otra parte, el sistema incluye una aplicación cliente, *frontend*, que contiene la interfaz con la que interactúa el usuario final. Este subsistema cuenta además con una capa de acceso al servicio encargada de enviar las peticiones necesarias a la *API REST* del servidor.

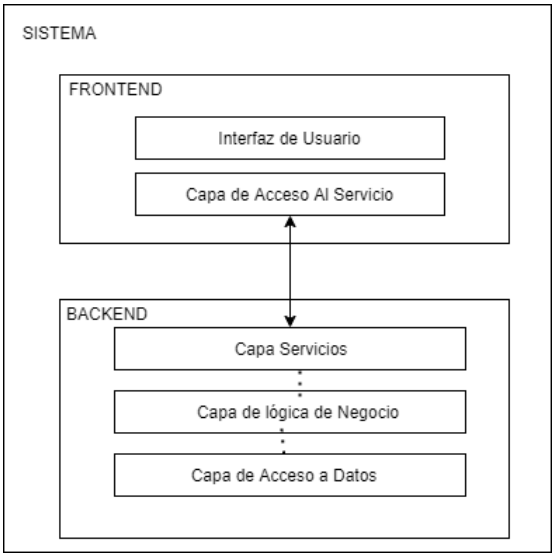


Figura 8.1: Diagrama de Arquitectura General del Sistema

8.4 Subsistema Backend

8.4.1 Objetivos

Este subsistema define la aplicación del lado servidor. Este ha sido desarrollado utilizando *SpringBoot* [12] y contiene la lógica del sistema. El principal objetivo e el desarrollo de esta parte es, diseñar un sistema dividido en capas independientes reduciendo la dependencia de implementaciones concretas. Con esto se pretende facilitar el mantenimiento de la aplicación y reducir el tiempo de desarrollo, haciendo una división clara y comprensible de las diferentes funcionalidades.

8.4.2 Arquitectura

La arquitectura de este subsistema se puede observar en la figura 8.2.

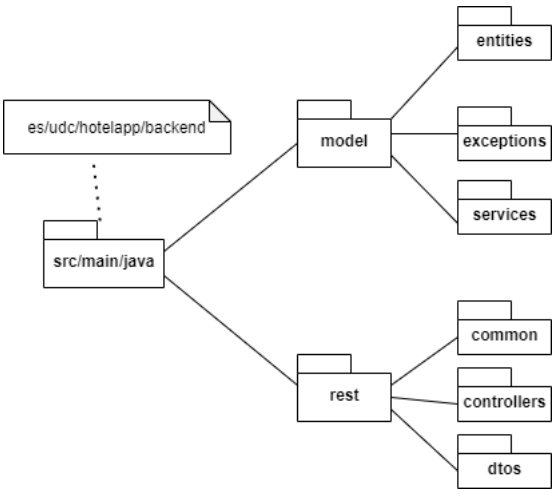


Figura 8.2: Jerarquía de ficheros del Backend

Como se puede apreciar, el sistema se divide en una serie de paquetes que agrupan elementos de las diferentes capas. El paquete *Entities* conforma la capa de acceso a datos. El paquete *services* está formado por los diferentes servicios que mantienen la lógica de negocio y en *rest* se encuentran los elementos que forman la *API REST*. A continuación se detalla, en general, cada una de estas capas y su objetivo dentro del sistema. En primer lugar se ha desarrollado la capa de acceso a datos. Para ello, ha sido necesario en primera instancia crear las entidades que utilizará la aplicación. Estas serán mapeadas a la Base de Datos (*BD*) de forma automática gracias a *Spring-Data* [14]. Para la creación, modificación y borrado de las entidades de la BD ha sido necesaria la creación de la otra parte que conforma la capa de acceso a datos, esto es, los DAOs (*Data Access Objects*). Por defecto, *Spring* proporciona una implementación genérica para estos. Sin embargo, se han realizado implementaciones propias con el objetivo de realizar consultas más complejas. Sobre la capa de acceso a datos, se encuentra la capa de lógica de negocio, en la figura el paquete de *services*. Esta capa está formada por una serie de servicios divididos por ámbito de funcionalidad y son los encargados de ejecutar la lógica del sistema realizando las peticiones pertinentes a la capa de acceso a datos, implementando los casos de uso del modelo. En última instancia, la capa servicio proporciona una *API REST* formada por una serie de controladores divididos, a su vez por el tipo de funcionalidad. Esta capa recibe las peticiones de los clientes y ejecuta las funciones de los servicios necesarias.

8.4.3 Modelo del dominio

Diagrama de entidades

En la figura 8.3, se pueden ver las diferentes entidades modeladas en la capa de acceso a datos de la aplicación del lado servidor. Existen dos tipos fundamentales de entidades: clases y enumerados. Estos últimos agrupan valores que representan características de las clases. Todas las entidades cuentan con identificadores autogenerados mediante secuencias independientes usando columnas contador en Base de Datos. Cada entidad se detalla a continuación:

- **User**: Mantiene la información personal de los usuarios, como son el nombre de usuario, la contraseña, el nombre, los apellidos, el *e-mail* y la dirección. Para controlar el rol de cada usuario se utiliza un enumerado con los valores de *USER, MANAGER, HOTEL, ADMIN*.
- **Hotel**: Mantiene la información de los hoteles, como son el nombre del hotel, el gerente, la dirección, una breve descripción y el número de teléfono.
- **Room**: Hace referencia a las habitaciones que van a ser almacenadas por cada uno de los hoteles.
- **RoomType**: Almacena los distintos tipos de habitaciones, principalmente mediante un nombre.
- **RoomTypeReservation**: Hace referencia a las diferentes reservas que va a realizar el usuario.
- **Account**: Almacena la información de los gastos que va a realizar el usuario, así como el precio total consumido.
- **AccountItem**: Hace referencia a cada uno de los productos o servicios adquiridos por el usuario.
- **RoomReservation**: Modela la asignación de las habitaciones a las reservas.
- **Product**: Almacena la información de los productos disponibles en cada *Hotel*, como es el nombre, el precio y una descripción.
- **Service**: Almacena la información de los servicios disponibles en cada *Hotel*, como es el nombre, el precio y una descripción.
- **Guest**: Almacena la información de los diferentes clientes que pasan por el hotel, como es el nombre, los apellidos, la dirección, un documento identificativo y el teléfono de contacto.

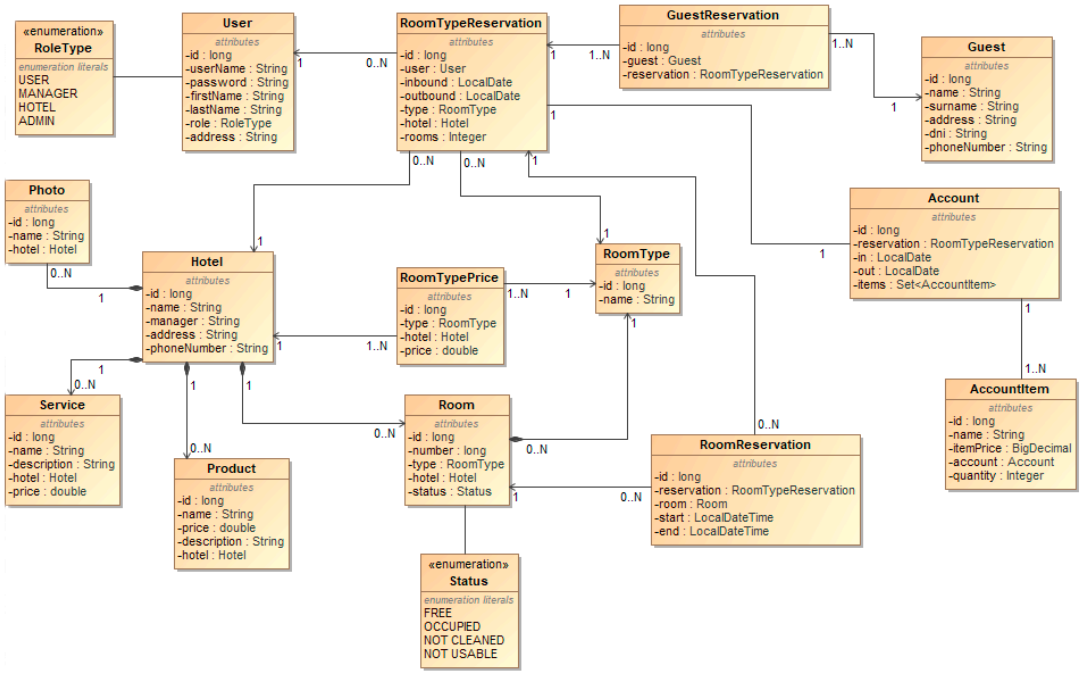


Figura 8.3: Diagrama de Entidades

- **GuestReservation**: Modela la entrada de un *Guest* en un *Hotel*.
- **Photo**: Almacena la información correspondiente a las diferentes fotos de un *Hotel*.
- **RoomTypePrice**: Representa los precios de los diferentes tipos de habitaciones de un hotel.

Dado que la relación entre la entidad *Guest* y la entidad *RoomTypeReservation*, es una relación *N:M*, se ha decidido crear la entidad *GuestReservation* que almacena esta información, obteniendo así una doble relación *1:N* con cada una de las entidades anteriormente mencionadas. Lo mismo ocurre con las entidades de *Room* y *RoomTypeReservation*, que se procedió de la misma forma creando la entidad de *RoomReservation*, dado que una reserva puede solicitar varias habitaciones. Por otro lado, en la entidad de *Account*, se ha procedido a crear los métodos de *AddItem* y *GetItem*, apoyándonos en el *Domain Model Pattern*, para poder añadir u obtener los elementos respectivamente. También se ha creado un método para obtener el precio total de la cuenta de gastos, ahorrando así recursos de otras capas de la aplicación

Modelo de datos

En la figura 8.4, se puede ver el modelo de datos de la aplicación, este representa el conjunto de tablas que mantienen información en la base de datos, donde cada una de estas equivale

a una entidad de la capa de acceso a datos. Mediante anotaciones, se define el modelado de las relaciones entre clases indicando a *Spring* la forma de llevarlas a cabo (e.g. columna de la tabla a la que referencia un atributo). Estas tablas mantienen información referente a cada entidad, incluyendo en cada caso un identificador único, que es autogenerado.

Spring construye las relaciones entre entidades partiendo de ciertas notaciones: *@OneToMany @OneToOne* y *@ManyToOne*. El uso de estas anotaciones depende de la multiplicidad de cada relación.

Al instanciar una entidad que participa de una relación *1:N*, se recupera información de las instancias con las que se relaciona dicha entidad. Esto se produce porque la política por defecto es *EAGER*, lo cual puede suponer un problema de eficiencia al recuperar información de forma innecesaria. Para evitar este problema se utiliza la propiedad ***fetchType = LAZY*** de *@OneToMany*, provocando que solo se instancien entidades relacionadas en caso de ser necesario. Aunque se pueden producir ineficiencias y pérdida de actualizaciones de datos se ha optado por no utilizar *@version* para solucionar la pérdida de actualizaciones y no utilizar un *@batchsize* determinado, para las posibles ineficiencias.

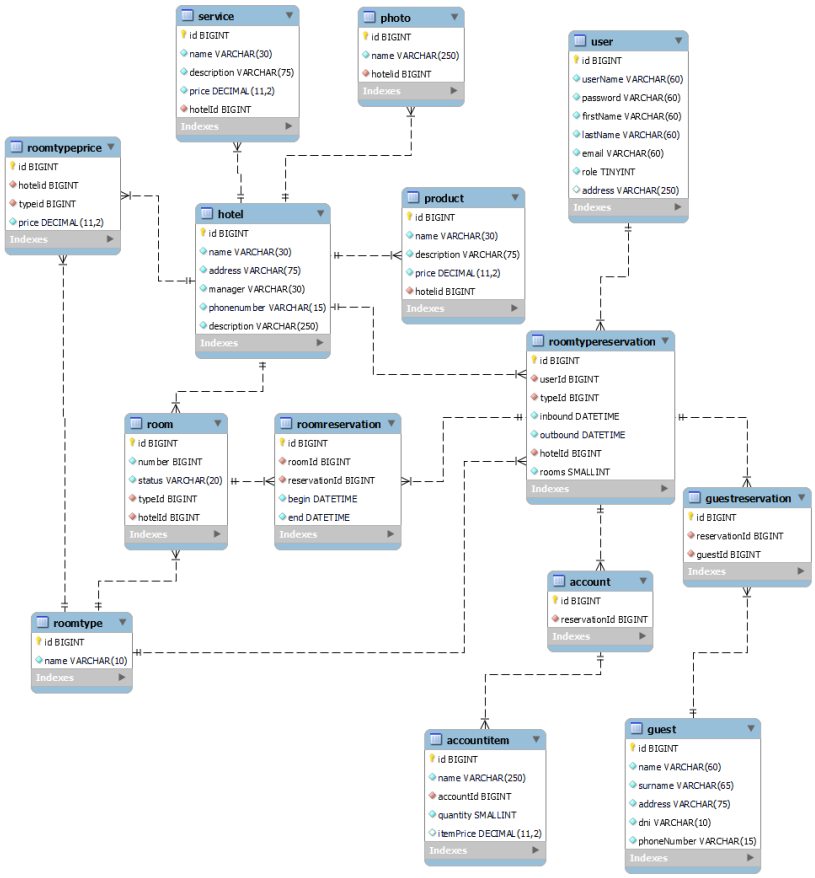


Figura 8.4: Modelo de datos

8.4.4 Capa de acceso a datos

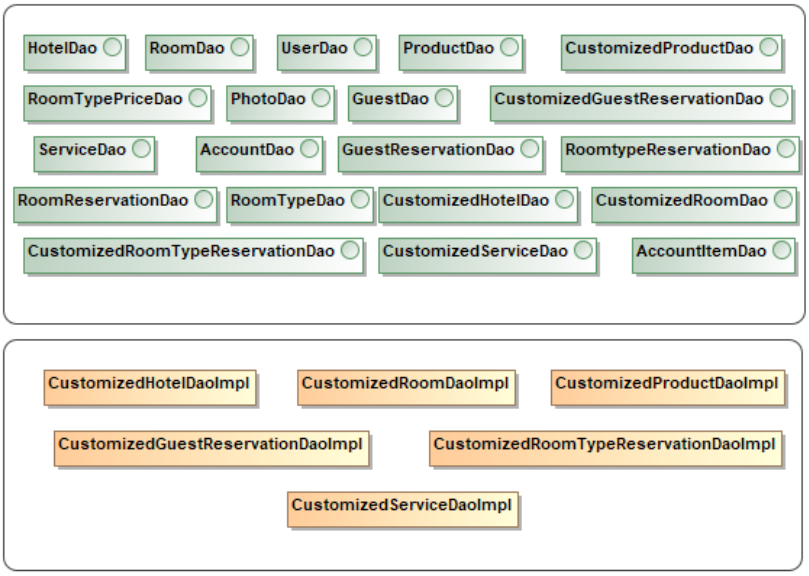


Figura 8.5: Diagrama de Capa de Acceso a Datos

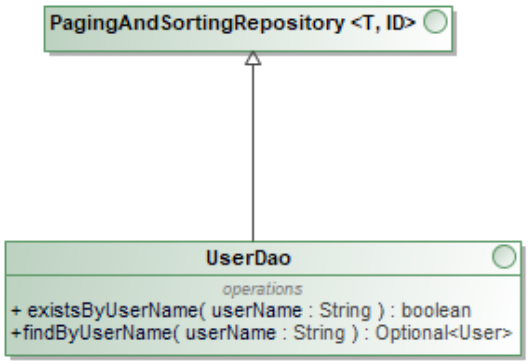


Figura 8.6: Arquitectura de un DAO

En la figura 8.5, se enumeran los DAOs de la capa de acceso a datos y las implementaciones que fueron necesarias en algunos casos para la realización de consultas que no eran proporcionadas por defecto por *Spring Data*. Estas hacen un uso de las facilidades que ofrece el framework de Spring extendiendo la clase *PagingAndSortingRepository* [39] que proporciona las funcionalidades necesarias para realizar las consultas y modificaciones contra el sistema de persistencia. En los casos donde se realizaron implementaciones adicionales para determinadas consultas sobre la BBDD, se utilizó la interfaz *EntityManager* de Java con la anotación de Spring *PersistenceContext* para asociarlo con la dependencia de la fuente de datos empleada.

Para mostrar la arquitectura de un DAO, dado que la estructura es similar en todos ellos, se muestra un ejemplo en la figura 8.6, en este caso de *RoomTypeDao*. Como se ha comentado, extiende la interfaz de *PagingAndSortingRepository*, que necesita como parámetros el tipo de dominio que se manejará (T) y el tipo del identificador de la entidad (ID).

La interfaz de *PagingAndSortingRepository*, extiende a su vez la interfaz de *CrudRepository*. Esta otra interfaz cuenta con las operaciones necesarias para devolver el número de entidades disponibles, borrar una o todas las entidades, hacer borrados a partir de un identificador, devolver todas las entidades de un tipo, buscar una instancia por *id*, almacenar una entidad y algunas otras operaciones similares con diferentes parámetros. Las operaciones añadidas por *PagingAndSortingRepository* son relativas al paginado y la ordenación de los elementos.

A mayores, como se puede observar en la figura, se pueden definir, por convención de nombrado, usando la semántica correcta y mediante el nombre del atributo, en este caso *name*, una nueva operación sin necesidad de que el programador la implemente. En esta aplicación, las otras operaciones definidas así fueron:

En *AccountDao*:

- *Optional<Account> findByReservationId (long id)*: si existe, devuelve la cuenta de gasto asociada a esa reserva.

En *GuestDao*:

- *boolean existsByDni (String dni)*: devuelve verdadero o falso si existe un Cliente con ese identificador oficial.
- *Optional<Guest> findByDni (String dni)*: si existe, devuelve el cliente con ese identificador oficial.

En *HotelDao*:

- *boolean existsByName (String name)*: devuelve verdadero o falso si existe un Hotel con el nombre indicado.

En *ProductDao*:

- *boolean existsByName (String name)*: devuelve verdadero o falso si existe un Producto con el nombre indicado.
- *Optional<Product> findByName (String name)*: si existe, devuelve el producto con ese nombre.

En *RoomDao*:

- *boolean existsByNumber (int number)*: devuelve verdadero o falso en función de si existe una Habitación con el número indicado.

- *Optional<Room> findByNumber(int number)*: si existe, devuelve la habitación con ese número.

En *RoomTypePriceDao*:

- *Optional<RoomTypePrice> findByTypeIdAndHotelId (long typeid, long hotelid)*: si existe, devuelve el Precio correspondiente a ese tipo de habitación dentro de ese hotel.

En *ServiceDao*:

- *Optional<Service> findByName (String name)*: si existe, devuelve el servicio con ese nombre.

En *UserDao*:

- *boolean existsByUserName (String userName)*: devuelve verdadero o falso si existe un Usuario con el nombre de usuario indicado.
- *Optional<User> findByUserName (String userName)*: si existe, devuelve el usuario con ese nombre de usuario.

Aún así, hay ciertas operaciones que debido a su complejidad, es necesario implementarlas por separado, es el caso de las operaciones contenidas en los ficheros con fondo naranja de la figura 8.5 que se detallan a continuación. En *CustomizedHotelDaoImpl* se definió una operación para poder realizar búsquedas de hoteles según el nombre y la dirección de este, además de paginarlos y ordenarlos por el nombre. En *CustomizedGuestReservationDaoImpl*, se ha implementado una operación para poder filtrar los resultados mediante el nombre del cliente, también ordenados y paginados. En *CustomizedRoomDaoImpl*, se ha implementado una operación para poder buscar habitaciones mediante el identificador del hotel, el tipo de estas y su estado, también paginando y ordenando en este caso por el número de la misma. La implementación contenida en *CustomizedServiceDaoImpl* y *CustomizedProductDaoImpl*, es similar dado que en ambos casos se ha implementado un método para buscar coincidencias a partir del identificador del hotel y un nombre, siendo este último un servicio o producto respectivamente, también paginados y ordenados. En *CustomizedRoomTypeReservationDaoImpl*, se han creado cuatro operaciones para buscar las reservas en base a unos criterios determinados, siendo tres de ellas paginadas y ordenadas, mientras que la última no se ha creído necesario paginarla por la cantidad de coincidencias que puede contener. Entrando más en detalle, todas ellas tienen como parámetro el identificador del hotel. Después, el resto de parámetros que reciben varía siendo, en una de ellas el nombre del usuario y una fecha. Otra de ellas, recibe el tipo de habitación y las fechas de entrada y salida. Una tercera, recibe también el tipo de habitación y la fecha de entrada al hotel. La última de ellas, recibe por parámetro el identificador del usuario y la fecha, siendo esta la única que no se ha paginado por los motivos anteriormente comentados.

8.4.5 Capa de lógica de negocio

Esta capa está formada por un conjunto de servicios, divididos por el tipo de funcionalidad que implementan. Se divide, más concretamente en interfaces de servicio y sus implementaciones concretas. La estructura de la capa se puede ver en la figura 8.7. Las interfaces declaran las funciones concretas que lleva a cabo cada servicio. Las clases de implementación aportan la lógica concreta de cada una de estas funcionalidades.

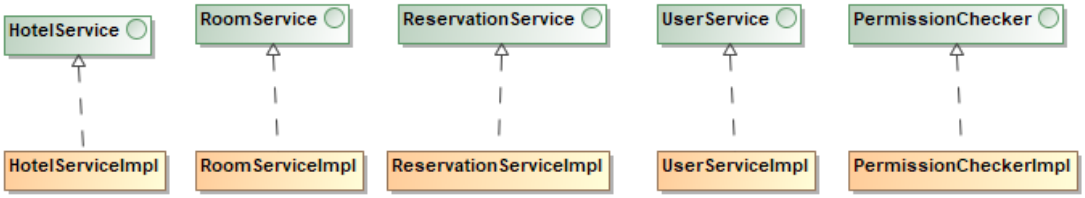


Figura 8.7: Diagrama de los servicios de la Capa Modelo

Los servicios reciben peticiones de la capa superior (*capa servicios*) y ejecutan una serie de operaciones en consecuencia. Entre estas se encuentran la consulta y/o modificación de información en la base de datos. Para ello se apoyan en las funciones que proporcionan los DAOs.

Spring inyecta la implementación concreta de estos servicios en la capa de servicios mediante la notación *@Service*. Esta incluye en cada implementación concreta y permite que sea instanciada conociendo únicamente la interfaz a la que implementa, suponiendo un diseño más modular y menos acoplado.

A continuación se detalla la función principal de cada servicio y las operaciones que lleva a cabo:

En **UserService**, servicio encargado de la gestión de usuarios podemos encontrar:

- *void* ***signUp****(User user) throws IncorrectLoginException*: Método que sirve para registrar a un usuario nuevo dentro de la aplicación.
- *User* ***login****(String username, String password) throws InstanceNotFoundException*: Inicia sesión para un usuario registrado.
- *User* ***loginFromId****(Long id) throws InstanceNotFoundException*:Inicia sesión a partir del identificador del usuario.
- *User* ***updateProfile****(Long id, String firstName, String lastName, String email, String address) throws InstanceNotFoundException*: Actualiza la información del perfil de usuario.

- *void* ***changePassword****(Long id, String oldPassword, String newPassword) throws InstanceNotFoundException, IncorrectPsswordException*: Cambia la contraseña del usuario.

HotelService es el servicio encargado de gestionar los hoteles, sus servicios y productos y los precios de los tipos de habitación, así como de subir una o varias fotos. En este podemos encontrar:

- *Long* ***createHotel****(Hotel hotel) throws HotelAlreadyExistsExeption*: Crea un hotel en la Base de Datos.
- *Hotel* ***findById****(Long id) throws InstanceNotFoundException*: Devuelve la información de un hotel a partir de su identificador.
- *Hotel* ***updateHotel****(Hotel hotel) throws InstanceNotFoundException*: Actualiza la información del hotel.
- *Block<Hotel>* ***findHotels****(String name, String address, int page, int size)*: Devuelve una lista paginada y ordenada con la información de los hoteles que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *void* ***removeHotel****(Long hotelid) throws InstanceNotFoundException*: Elimina la información de ese hotel dentro de la aplicación.
- *Long* ***addService****(Service service) throws InstanceNotFoundException, ServiceAlreadyExistsExeption*: Crea un servicio en la Base de Datos.
- *Service* ***findById****(Long id) throws InstanceNotFoundException*: Devuelve la información de un servicio a partir de su identificador.
- *Service* ***updateService****(Service service) throws InstanceNotFoundException*: Actualiza la información del servicio.
- *Block<Service>* ***findServices****(Long hotelid, String name, int page, int size)*: Devuelve una lista paginada y ordenada con la información de los servicios que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *void* ***removeService****(Long serviceid) throws InstanceNotFoundException*: Elimina la información de ese servicio dentro de la aplicación.
- *Long* ***addProduct****(Product p) throws InstanceNotFoundException, ProductAlreadyExistsExeption*: Crea un producto en la Base de Datos.
- *Product* ***findById****(Long id) throws InstanceNotFoundException*: Devuelve la información de un producto a partir de su identificador.

- *Product* ***updateProduct****(Product p) throws InstanceNotFoundException*: Actualiza la información del producto.
- *Block<Product>* ***findProducts****(Long hotelid, String name, int page, int size)*: Devuelve una lista paginada y ordenada con la información de los productos que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *void* ***removeProduct****(Long producid) throws InstanceNotFoundException*: Elimina la información de ese producto dentro de la aplicación.
- *Long* ***addPrice****(RoomTypePrice price)*: Crea un precio de un tipo de habitación en la Base de Datos.
- *RoomTypePrice* ***findPriceById****(Long id)*: Devuelve la información de un precio de un tipo de habitación a partir de su identificador.
- *RoomTypePrice* ***updateRoomTypePrice****(RoomTypePrice price) throws InstanceNotFoundException*: Actualiza la información del precio del tipo de habitación.
- *void* ***removePrice****(Long priceid) throws InstanceNotFoundException*: Elimina la información de ese precio del tipo de la habitación dentro de la aplicación.
- *boolean* ***uploadPhoto****(MultipartFile file, Long hotelid)*: Devuelve verdadero o falso en función de si ha sido posible subir el archivo al servidor.

En **PermissionChecker**, encargado de hacer validaciones reusadas en varios puntos de la aplicación, podemos encontrar:

- *User* ***checkUser****(Long userId) throws InstanceNotFoundException*: Comprueba si el usuario existe y devuelve la información de este.
- *boolean* ***checkIfPossibleToBook****(Long hotelid, String in, String out, String type, int quantity)*: Comprueba si es posible realizar la reserva de esa cantidad de habitaciones para evitar que se produzca un *overbooking*.

Dentro de **RoomService**, que gestiona todo lo relacionado con las habitaciones del hotel, podemos encontrar:

- *Long* ***addRoom****(Room room) throws InstanceNotFoundException, RoomAlreadyExistsExeption*: Crea una habitación en la Base de Datos.
- *Room* ***findRoom****(Long id) throws InstanceNotFoundException*: Devuelve la información de una habitación a partir de su identificador.

- *void* ***updateRoom****(Room r1) throws InstanceNotFoundException*: Actualiza la información de la habitación.
- *Block<Room>* ***findRooms****(String status, Long hotelid, String type, int page, int size)*: Devuelve una lista paginada y ordenada con la información de las habitaciones que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *void* ***removeRoom****(Room r1) throws InstanceNotFoundException*: Elimina la información de esa habitación dentro de la aplicación.
- *List<RoomType>****findAllRoomTypes****()*: Devuelve una lista con los tipos de habitaciones que se encuentran en la aplicación.

Dentro de **ReservationService**, encargado de gestionar lo referente a las reservas, a la asignación de habitaciones, la gestión de los clientes y las cuentas de gasto de cada una de las reservas, podemos encontrar los siguientes métodos:

- *RoomTypeReservation* ***addReservation*** *(RoomTypeReservation rt1) throws InstanceNotFoundException, ReservationException*: Añade una reserva dentro de la aplicación.
- *Block<RoomTypeReservation>* ***findReservations*** *(Long hotelid,String username, String date, int page, int size)*:Devuelve una lista paginada y ordenada con la información de las reservas que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *List<RoomTypeReservation>* ***findReservations*** *(Long hotelid, Long userid, String date)*: Devuelve una lista con la información de las reservas que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *void* ***updateReservation****(RoomTypeReservation rt2)*: Actualiza la información de la reserva.
- *RoomTypeReservation* ***findById****(Long id) throws InstanceNotFoundException*: Devuelve la información de una reserva a partir de su identificador.
- *void* ***removeReservation****(Long reservationid) throws InstanceNotFoundException*: Elimina la información de esa reserva dentro de la aplicación.
- *RoomReservation* ***assignReservation*** *(RoomReservation rr1, Long id) throws IncorrectReservationException*: Asigna una habitación a una reserva.
- *GuestReservation* ***addGuest*** *(GuestReservation gr1) throws IncorrectReservationException*: Añade un cliente dentro de la aplicación.

- *Block<GuestReservation>* ***findAllGuestReservations*** *(Long hotelid, String username, int page, int size)*: Devuelve una lista paginada y ordenada con la información de los clientes que coincidan con los parámetros de búsqueda o de todos si los parámetros están vacíos.
- *GuestReservation* ***findGuestReservationById*** *(Long id) throws InstanceNotFoundException*: Devuelve la información de un cliente a partir de su identificador.
- *Account* ***createAccount****(Account acc) throws IncorrectReservationException*: Crea una cuenta de gastos asociada a una reserva.
- *Account* ***findAccount*** *(Long reservationid) throws InstanceNotFoundException*: Devuelve la información de una cuenta de gastos a partir del identificador de la reserva.
- *Account* ***addToAccount*** *(Long productid, Long serviceid, Long reservationid, int quantity) throws InstanceNotFoundException*: Añade la cantidad de productos o servicios a la cuenta de gastos asociada a la reserva indicada.

Por último se explican las excepciones que aparecen en la firma de los métodos listados:

- *InstanceNotFoundException*: indica que la entidad referenciada no existe.
- *PermissionException*: indica que no se puede realizar la operación que se solicita.
- *HotelAlreadyExistsException*: indica que el hotel ya existe en el sistema.
- *ProductAlreadyExistsException*: indica que el producto ya existe en el sistema.
- *ServiceAlreadyExistsException*: indica que el servicio ya existe en el sistema.
- *RoomAlreadyExistsException*: indica que la habitación ya existe en el sistema.
- *IncorrectReservationException*: indica que la reserva referenciada no es correcta.
- *DuplicateInstanceException*: este error se produce al querer crear una entidad con iguales datos a otra ya existente, cuando esto no está permitido por cuestiones de unicidad.
- *IncorrectLoginException*: inicio de sesión incorrecto por proporcionar unas credenciales inválidas.
- *IncorrectPasswordException*: este error se produce cuando al introducir una contraseña, esta no coincide con la almacenada dentro de la aplicación.
- *ReservationException*: Indica que una reserva no se puede realizar.

8.4.6 Capa servicios

Esta capa tiene como objetivo fundamental recibir peticiones de la aplicación cliente y ejecutar funciones de los servicios en consecuencia. Para realizar esta tarea, la capa de servicio ofrece una serie de *endpoint*.

Cada uno de estos cuenta con una función asociada que ejecutará la lógica correspondiente a la petición. Para identificar dichos *endpoints* y la función que ejecutan, *Spring Web* [15] proporciona las notaciones *@PostMapping*, *@GetMapping*, *@PutMapping* y *@DeleteMapping*. Estas representan a los tipos de petición siguientes:

- **POST**: Creación de información.
- **PUT**: Modificación de información.
- **GET**: Consulta de información.
- **DELETE**: Borrado de información.

Ante una petición concreta, el controlador correspondiente ejecuta una o varias funciones de los servicios. Para ello *Spring* inyecta implementaciones concretas de los servicios en los controladores mediante la notación *@Autowired* sobre la declaración de las interfaces que implementan los servicios. Esto evita tener que instanciar implementaciones concretas reduciendo el acoplamiento del código.

La API REST implementada sigue la arquitectura RESTful de forma parcial, al cumplir que está orientada a recursos y utilizar de forma consistente las operaciones y códigos de respuesta HTTP para modelar las diferentes operaciones y estar diseñada para no mantener estado interno.

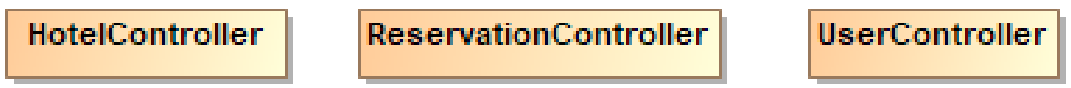


Figura 8.8: Controladores del Servicio Web

Los *endpoints* y las peticiones que reciben se detallan a continuación:

En **UserController**:

Recurso	Método	Excepciones
/users/signUp	*POST*	*DuplicateInstanceException*
/users/login	*POST*	*IncorrectLoginException*
/users/loginFromServiceToken	*POST*	*InstanceNotFoundException*
/users/{id}	*PUT*	*InstanceNotFoundException, PermissionException*
/users/{id}/changePassword	*POST*	*InstanceNotFoundException, PermissionException, IncorrectPasswordException*

Tabla 8.1: API de UserController

- */users/signUp* (*POST*): registra un nuevo usuario.
- */users/login* (*POST*): inicia la sesión de un usuario registrado.
- */users/loginFromServiceToken* (*POST*): inicia la sesión de un usuario registrado a partir de su identificador.
- */users/{id}* (*PUT*): Actualiza la información del perfil del usuario.
- */users/{id}/changePassword* (*POST*): Cambia la contraseña del usuario.

En el controlador de **HotelController** se han unificado los servicios de la capa inferior de *RoomService* y *HotelService*. Además, se ha creado un *endpoint* para obtener los clientes del hotel, quedando esto de la siguiente manera:

Recurso	Método	Excepciones
/hotels	*POST*	*HotelAlreadyExistsException*
/hotels	*GET*	
/hotels/{hotelid}	*GET*	*InstanceNotFoundException*
/hotels/{hotelid}	*PUT*	*InstanceNotFoundException*
/hotels/{hotelid}	*DELETE*	*InstanceNotFoundException*
/hotels/{hotelid}/upload-photo	*POST*	
/hotels/{hotelid}/rooms	*POST*	*RoomAlreadyExistsException, InstanceNotFoundException*
/hotels/{hotelid}/rooms	*GET*	
/hotels/{hotelid}/rooms/{id}	*GET*	*InstanceNotFoundException*
/hotels/{hotelid}/rooms/{id}	*DELETE*	*InstanceNotFoundException*
/hotels/{hotelid}/rooms/{id}	*PUT*	
/roomtypes	*GET*	
/hotels/{hotelid}/guests	*GET*	

/hotels/{hotelid}/prices	POST	
/hotels/{hotelid}/prices/{id}	GET	InstanceNotFoundException
/hotels/{hotelid}/prices/{id}	PUT	InstanceNotFoundException
/hotels/{hotelid}/prices/{id}	DELETE	InstanceNotFoundException
/hotels/{hotelid}/products	POST	ProductAlreadyExistsException, InstanceNotFoundException
/hotels/{hotelid}/products	GET	
/hotels/{hotelid}/products/{id}	GET	InstanceNotFoundException
/hotels/{hotelid}/products/{id}	DELETE	InstanceNotFoundException
/hotels/{hotelid}/products/{id}	PUT	InstanceNotFoundException
/hotels/{hotelid}/services	POST	ServiceAlreadyExistsException, InstanceNotFoundException
/hotels/{hotelid}/services	GET	
/hotels/{hotelid}/services/{id}	GET	InstanceNotFoundException
/hotels/{hotelid}/services/{id}	DELETE	InstanceNotFoundException
/hotels/{hotelid}/services/{id}	PUT	InstanceNotFoundException

Tabla 8.2: API de HotelController

- */hotels (POST)*: Añade un hotel
- */hotels (GET)*: Obtiene los hoteles a partir de los criterios clave (el nombre de este o la dirección)
- */hotels/{hotelid} (GET)*: Obtiene la información del hotel a partir del identificador
- */hotels/{hotelid} (PUT)*: Modifica la información del hotel
- */hotels/{hotelid} (DELETE)*: Elimina el hotel
- */hotels/{hotelid}/upload-photo*: Sube la foto del hotel
- */hotels/{hotelid}/rooms (POST)*: Añade una habitación
- */hotels/{hotelid}/rooms (GET)*: Obtiene la información de las habitaciones por criterios clave (identificador del hotel, el estado y el tipo de esta)
- */hotels/{hotelid}/rooms/{id} (GET)*: Obtiene la información de la habitación a partir del identificador
- */hotels/{hotelid}/rooms/{id} (PUT)*: Modifica la información de la habitación.

- */hotels/{hotelid}/rooms/{id} (DELETE)*: Elimina la habitación
- */roomtypes*: Obtiene todos los tipos de habitaciones
- */hotels/{hotelid}/guests*: Obtiene todos los clientes
- */hotels/{hotelid}/prices*: Añade un precio de un tipo de habitación
- */hotels/{hotelid}/prices/{id} (GET)*: Obtiene el precio de un tipo de habitación a partir de su identificador.
- */hotels/{hotelid}/prices/{id} (PUT)*: Modifica la información del precio del tipo de habitación determinado.
- */hotels/{hotelid}/prices/{id} (DELETE)*: Elimina el precio determinado
- */hotels/{hotelid}/products (POST)*: Añade un producto
- */hotels/{hotelid}/products (GET)*: Obtiene la información de los productos por criterios clave (identificador del hotel y el nombre)
- */hotels/{hotelid}/products/{id} (GET)*: Obtiene la información de un producto determinado
- */hotels/{hotelid}/products/{id} (PUT)*: Actualiza la información de un producto determinado
- */hotels/{hotelid}/products/{id} (DELETE)*: Elimina el producto
- */hotels/{hotelid}/services (POST)*: Añade un servicio
- */hotels/{hotelid}/services (GET)*: Obtiene los servicios por criterios clave (identificador del hotel y el nombre)
- */hotels/{hotelid}/services/{id} (GET)*: Obtiene un servicio determinado
- */hotels/{hotelid}/services/{id} (PUT)*: Modifica el servicio determinado
- */hotels/{hotelid}/services/{id} (DELETE)*: Elimina el servicio determinado.

En el **ReservationController**:

Recurso	Método	Excepciones
/reservations	*POST*	*InstanceNotFoundException, ReservationException, IncorrectReservationException*
/reservations	*GET*	
/reservations/{id}	*GET*	*InstanceNotFoundException*
/reservations/{id}	*PUT*	*InstanceNotFoundException*
/reservations/{id}	*DELETE*	*InstanceNotFoundException*
/resrvations/{id}/guests	*POST*	*IncorrectReservationException*
/reservaions/{id}/assignRoom	*POST*	*IncorrectReservationException, InstanceNotFoundException*
/reservations/{id}/account	*GET*	
/reservations/account	*GET*	
/reservations/{id}/account	*POST*	*InstanceNotFoundException*

Tabla 8.3: API de ReservationController

- */reservations* (POST): Añade una reserva.
- */reservations* (GET): Obtiene las reservas, opcionalmente a partir del identificador del hotel, el nombre del usuario o una fecha.
- */reservations/{id}* (GET): Obtiene la información de una reserva determinada
- */reservations/{id}* (PUT): Modifica la información de una reserva determinada
- */reservations/{id}* (DELETE): Elimina una reserva determinada
- */resrvations/{id}/guests*: Añade un cliente
- */reservaions/{id}/assignRoom*: Asigna una habitación a una reserva
- */reservations/{id}/account* (GET): Obtiene la cuenta de gastos de una reserva determinada.
- */reservations/{id}/account* (POST): Añade una cantidad de un producto o un servicio a la cuenta de gastos.
- */reservations/account*: Obtiene las reservas a partir del identificador del hotel, el id del usuario y una fecha.

Cada una de las excepciones indicadas previamente es capturada por un manejador definido en los controladores. *Spring* [13] proporciona la notación *ExceptionHandler* para indicarlos.

Estos capturan Excepciones concretas y devuelven un código de estado *HTTP* y un mensaje de error que serán enviados como respuesta a la petición cliente.

A continuación se muestran los mensajes de error HTTP asociados a cada excepción:

- PermissionException: **403 Forbidden**
- InstanceNotFoundException: **404 Not Found**
- IncorrectLoginException: **404 Not Found**
- ProductAlreadyExistsException: **409 Conflict**
- ServiceAlreadyExistsException: **409 Conflict**
- RoomAlreadyExistsException: **409 Conflict**
- HotelAlreadyExistsException: **409 Conflict**
- IncorrectPasswordException: **404 Not Found**
- IncorrectReservationException: **400 Bad Request**
- ReservationException: **400 Bad Request**

Por otro lado, para dotar a la aplicación de una cierta seguridad, en el archivo *SecurityConfig*, se han definido una serie de reglas que funcionan a modo de control de acceso. Esto es solo se permite el acceso y el correcto funcionamiento si se cumple alguna de las reglas que ahí están definidas, en caso contrario no se permite ni el acceso ni se produce el correcto funcionamiento de la aplicación. Para que esto se entienda, por ejemplo, un usuario con el rol de *USER* no puede acceder al listado de habitaciones de un hotel. Otro caso puede ser que un usuario no registrado intentase acceder a las reservas, produciendo un error con el código HTTP 403 *Forbidden*.

También cabe destacar que en la aplicación se han internacionalizado los mensajes de error, en nuestro caso se ha decidido por inglés, castellano y gallego, aunque se pueden internacionalizar a más idiomas, creando el fichero correspondiente con los pares clave-valor asociados a cada mensaje.

Por otro lado, se han realizado validaciones básicas comprobando que se envían los campos requeridos en cada petición al servidor, así como la representación del objeto dentro de la misma.

8.5 Subsistema Frontend

8.5.1 Objetivos

Este subsistema representa la aplicación cliente desarrollada en *React* [16]. Proporciona la interfaz de usuario y realiza peticiones a la aplicación servidora. Al desarrollar esta parte del sistema se pretende cumplir con los objetivos de diseño previamente establecidos. Se busca aplicar buenas prácticas que permitan facilitar el propio proceso de desarrollo del sistema así como su mantenimiento, además de facilitar la comprensión de los componentes. Por tratarse de una aplicación que realiza peticiones al *Backend* se pretende, minimizar el número de solicitudes reduciendo así la carga de trabajo del servidor.

8.5.2 Arquitectura

La arquitectura general del sistema se puede ver en la figura 8.9. Esta cuenta con dos partes principales claramente diferenciadas: la capa de interfaz de usuario que contiene los diferentes componentes que conforman el apartado visual y la capa de acceso a servicios que mantiene las peticiones que el subsistema realiza al *Backend*. Esta estructura facilita la comprensión del sistema y el proceso de modificación de sus componentes. La capa de acceso al servicio, está formada por una serie de funciones que contienen las peticiones que realiza el cliente al *Backend*. Estas construyen peticiones *HTTP* que serán enviadas a la capa de servicio del servidor. La información que es necesario enviar en peticiones de tipo *POST* o *PUT* es encapsulada en formato *JSON*. Tras enviar una petición, las funciones de la capa de acceso a datos interceptan la respuesta del servidor para que su información pueda ser utilizada por los componentes.

La capa de interfaz de usuario está compuesta por una serie de componentes. Estos se distribuyen en módulos como se puede apreciar en la figura 8.10. Cada uno de estos agrupan componentes que comparten ámbito de funcionalidad. A continuación se explica el contenido de cada uno de los módulos:

- **App**: Componentes que se muestran en la totalidad de la aplicación, como puede ser el menú superior.
- **Common**: Componentes que se reutilizan en diferentes módulos.
- **Guest**: Componentes que conforman todo lo referente a los clientes del hotel.
- **Hotel**: Componentes que modelan lo referente a los hoteles, incluyendo las fotos y los precios de los mismos.
- **Product**: Componentes que forman todo lo referente a los productos del hotel.

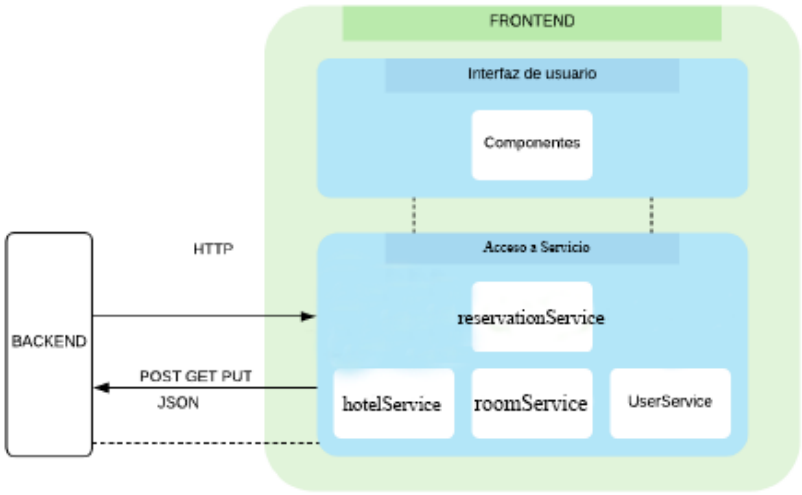


Figura 8.9: Diagrama de Arquitectura Frontend

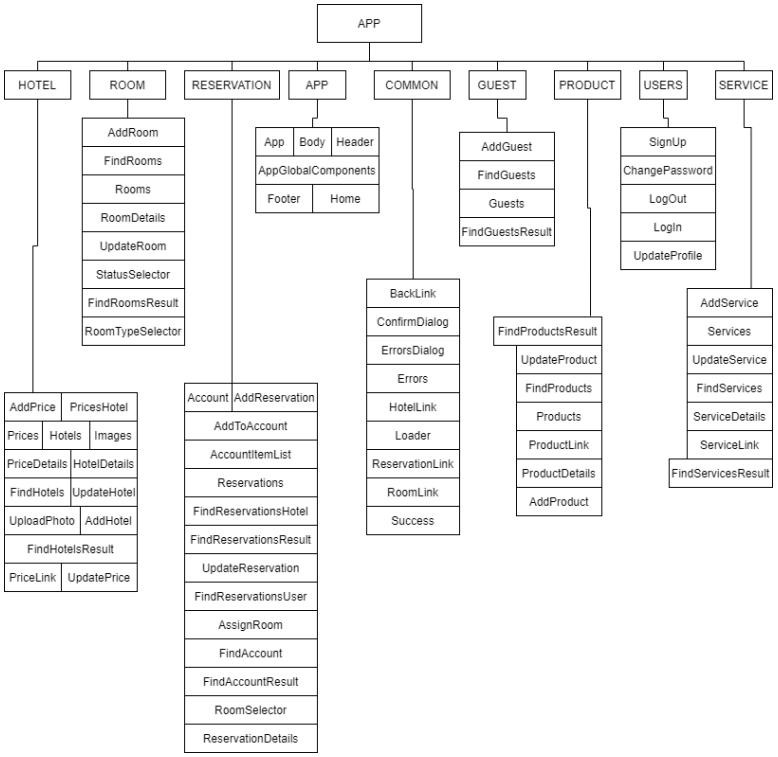


Figura 8.10: Diagrama de Componentes del Frontend

- **Reservation**: Componentes encargados de gestionar todas las acciones sobre las reservas de un usuario o de un hotel.

- **Room**: Componentes que modelan la interfaz de aquellas funciones necesarias para gestionar las habitaciones.
- **User**: Componentes que conforman la interfaz de aquellas funciones que puede llevar a cabo cualquier perfil.

Además, esta capa de interfaz de usuario contiene una serie de archivos dentro del directorio *i18n* para facilitar la traducción a diferentes idiomas de los diferentes mensajes. Por otro lado, también contiene un fichero de configuración inicial *index.js* en donde se configura el *proxy* para la conexión con el *backend* y se añade un manejador de eventos o *listener*, encargado de renderizar la aplicación completa o los componentes necesarios. También cuenta con un módulo especial, llamado *store*, en donde *redux* almacena todo el estado de la aplicación.

Cuando un usuario accede a la aplicación, se obtiene un componente principal *App*. Este contiene un elemento *Router* que permite asociar rutas o *URLs* a cada componente. El direccionamiento desde los componentes se produce mediante la modificación de estas rutas de acceso.

8.5.3 Capa de acceso al servicio

Como se ha comentado previamente, esta capa está compuesta de una serie de ficheros *javaScript* que se encargan de realizar las peticiones a la parte servidora. Los ficheros *javaScript* de esta aplicación se explican a continuación:

- **hotelService**: encargado de todas las peticiones referidas a los hoteles, incluyendo los productos y servicios que este contiene.
- **appFetch**: se encarga de configurar las diferentes peticiones a la parte servidora. Además también es el encargado de parsear el cuerpo de las peticiones/respuestas a JSON.
- **roomService**: encargado de todas las peticiones referidas a las habitaciones, incluyendo las peticiones de los tipos de estas.
- **networkError**: se encarga de lanzar un error cuando algo ha salido mal.
- **reservationService**: encargado de lo referente a las reservas, así como de asignar una habitación u añadir un cliente.
- **userService**: se encarga de todo lo relacionado con los usuarios.

8.5.4 Capa de componentes de interfaz

Como se ha indicado previamente, la interfaz de usuario está formada por una serie de componentes. Estos son ficheros *JavaScript* que contienen la parte visual de la aplicación y en

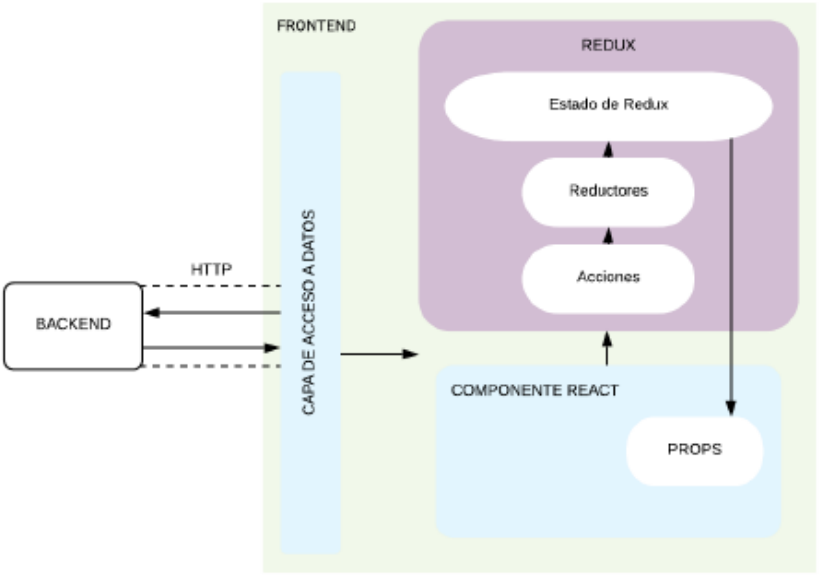


Figura 8.11: Diagrama de React con Redux

algunos casos una parte lógica. Cada componente contiene una función *render* que muestra los elementos visuales. Estos son declarados fundamentalmente usando código *HTML* al que se le aplican unos estilos *CSS*. Cuando se recupera un componente se ejecuta su método *render*, mostrando su contenido. Es posible incluir unos componentes dentro de otros permitiendo así su reutilización. En la aplicación, los componentes del módulo *Common* son reutilizados en distintos puntos de esta forma.

En la aplicación, existen componentes que modifican y/o obtienen información de la base de datos como puede ser el caso de *UpdateRoom* o *FindGuests*. Estos necesitan realizar peticiones al *Backend* por lo que realizan acciones, que su vez importan funciones declaradas en la capa de acceso a servicios del *Frontend*. De esta forma pueden ser ejecutadas por el propio componente ante ciertas acciones del usuario o como parte de su ciclo de vida.

Cada componente mantiene un estado formado por información utilizada por este. En el caso, por ejemplo, de *AddHotel* el estado lo conforman los diferentes valores que han sido introducidos en el formulario. Esta información parte de un estado inicial y se modifica con las acciones del usuario, pudiendo ser además compartida entre componentes. Para ello, es posible insertar el estado de un componente en las propiedades de otro. En ocasiones, es necesario compartir esta información con varios componentes. La utilización de propiedades, en este caso dificulta el proceso y la comprensión del código. Para solventar esta problemática se ha optado por incluir *Redux*.

Redux permite mantener un estado global en la aplicación. Este se encuentra distribuido

en los diferentes componentes en unos archivos conocidos como reductores. Esta información es accesible por cualquier componente. Los reductores que se han implementado son:

- En el módulo **Hotel**, mantiene el listado de hoteles y el hotel seleccionado.
- En el módulo **Room**, mantiene el listado de habitaciones, la habitación seleccionada y la lista de tipos de habitaciones.
- En el módulo **Guest**, mantiene el listado de clientes.
- En el módulo **User**, mantiene la información del usuario autenticado para ser usada en diferentes puntos de la aplicación.
- En el módulo **Product**, mantiene el listado de productos de un hotel, así como el producto seleccionado y la reserva que hay activa en ese preciso instante.
- En el módulo **Service**, mantiene el listado de servicios de un hotel, así como el servicio seleccionado y la reserva que hay activa en ese preciso instante.
- En el módulo **Reservation**, se mantiene la información del listado de reservas y la reserva seleccionada.
- En el módulo **App**, se mantiene la información de los errores, si la aplicación se encuentra en proceso de carga o ha terminado de cargar.

Para modificar el estado en *Redux* es necesario lanzar unas acciones que normalmente realizan peticiones al *Backend* y envían la respuesta a los reductores. Estos interpretan esta información y modifican su estado en consecuencia.

Los componentes se vinculan al estado global de la aplicación mediante la función *combineReducers*, que se encarga de gestionar el estado de la aplicación.

Además de proporcionar una forma de compartir un estado global ente componentes, *Redux* permite mantener cacheados resultados de peticiones al *Backend*, evitando tener que repetir peticiones concretas.

Cabe destacar el uso de la librería *react-intl* para facilitar la internacionalización de la aplicación. Esta incluye el componente *<FormattedMessage>* que permite añadir textos en los componentes, obtenidos de los diferentes ficheros de traducción.

Capítulo 9

Implementación

9.1 Software requerido

En esta sección se detalla el software requerido para el correcto desarrollo de esta aplicación:

Para la construcción del subsistema *Backend*:

- El paquete para desarrolladores **Java SE Runtime Enviroment 8+**
- Un sistema de bases de datos **Mysql 8+.**
- La herramienta **Apache Maven 3+.** para la construcción de proyectos

Para el desarrollo del subsistema *Frontend*:

- El administrador de paquetes **Yarn 1.21.1+.**
- Un entorno de ejecución **node.js 12.14.0+.**

9.2 Estructura

La estructura de ficheros del proyecto se divide en dos partes claramente diferenciadas. Por una parte encontramos el directorio *hotelapp/src/backend*, que contiene los ficheros y directorios correspondientes al subsistema *Backend*. Los ficheros de configuración del *Frontend* se encuentran almacenados en el directorio *hotelapp/src/frontend*. A continuación se describe cada uno de los directorios.

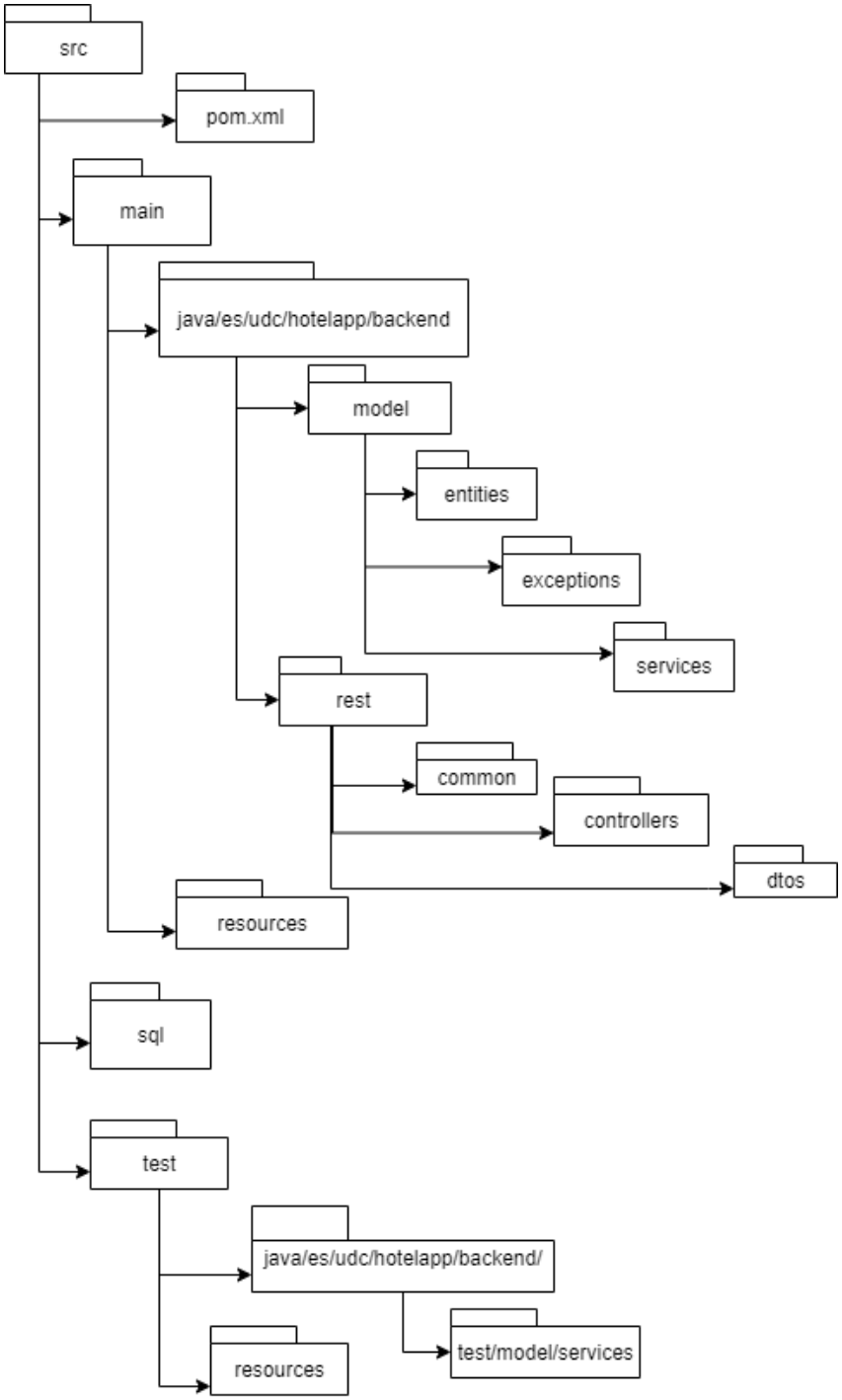


Figura 9.1: Estructura de Ficheros Backend

En lo que se refiere al *Backend*, al tratarse de un proyecto *maven*, la estructura de directorios está predefinida por el patrón de configuración de este. Por eso la estructura debe tener la forma que este determina, mostrado en la figura 9.1, aunque cabe destacar que el directorio *sql*

no es estándar, es recomendable hacer esa separación para agrupar los ficheros de creación de las tuplas y relaciones en BBDD.

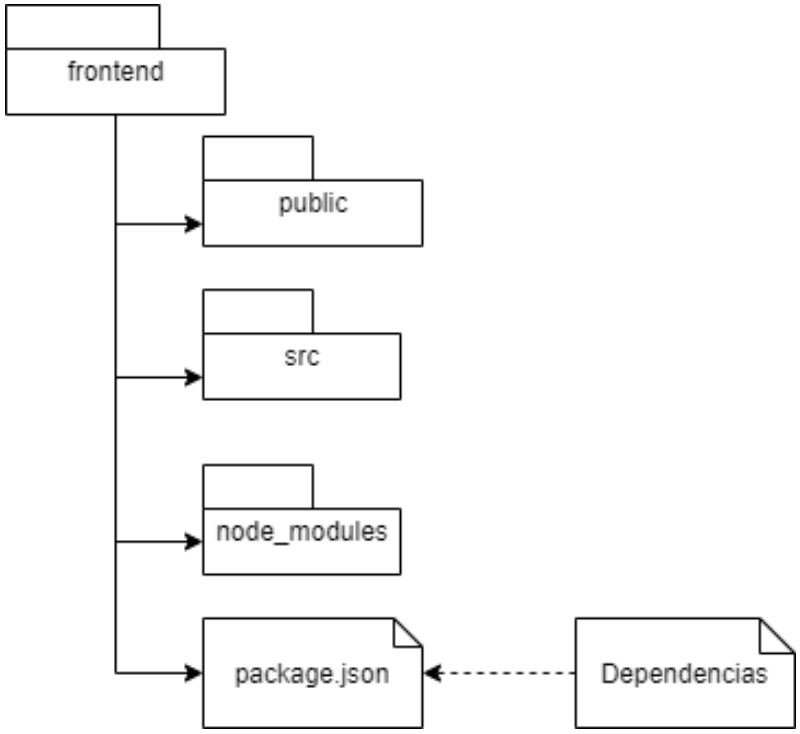


Figura 9.2: Estructura de Ficheros Frontend

En lo que se refiere al *Frontend*, ver figura 9.2, cuenta en primer lugar, con un directorio *node modules* donde se descargan las dependencias del proyecto, indicadas en el fichero *package.json*. El directorio *src* contiene los ficheros que construyen la capa servicios y la interfaz de usuario del *frontend*.

9.3 Instrucciones de compilación

Para la compilación y ejecución del proyecto es necesario, en primer lugar, la creación de la base de datos necesaria. Los pasos a seguir son los siguientes:

1. Si el gestor *Mysql* no está en ejecución, se ejecuta el comando **$mysqld** en un terminal.

2. Se accede al gestor de bases de datos mediante el comando **$mysqladmin -u root -p**, si no se ha indicado una contraseña durante la instalación podemos obviar la opción *-p*.

3. Se crea la base de datos mediante los siguientes comandos:

- **create hotelapp**
- **create user ”hotelapp@localhost” identified by hotelapp**
- **grant all privileges on hotelapp to hotelapp@localhost with grant option**

4. Ahora debemos ejecutar la creación de las tablas de las base de datos con el comando **$ mvn sql:execute**.
5. Ahora debemos generar el fichero *.jar* con el comando **$ mvn package**.
6. Se procede a la ejecución del *Backend*, mediante el comando **$ mvn spring-boot:run**, desde la carpeta */hotelapp/src/backend*.
7. Para finalizar, se ejecuta el *Frontend*, desde la carpeta */hotelapp/src/frontend*, mediante el comando **$ yarn start**.

Capítulo 10

Pruebas

10.1 Introducción

Durante el proceso de desarrollo de cualquier aplicación o sistema software, es importante verificar que el software que se está construyendo satisfaga los requisitos indicados previamente por el usuario. Además, resulta de vital importancia buscar e identificar errores y regresiones de manera temprana. Con este objetivo, resulta importante realizar pruebas que localicen estos errores permitiendo a los desarrolladores subsanarlos o implementar algún tipo de mejora. Existen diferentes tipos de pruebas en función del momento de desarrollo en que se realicen y la parte del sistema que se esté probando. En este apartado se describen todos los tipos de pruebas utilizados durante el desarrollo de este proyecto.

10.2 Pruebas de integración

Durante la implementación de la capa de servicios de lógica de negocio del *Backend* se han implementado pruebas de integración de cada caso de uso. Estas operan sobre los servicios de la capa modelo comprobando que sus funciones se ejecutan correctamente, devolviendo el resultado esperado. Estas pruebas son de especial importancia por dos motivos principales:

- Comprueban que la comunicación entre los servicios del modelo y las capas inferiores funciona correctamente. Además se verifica el correcto funcionamiento de los diferentes niveles.

- Facilitan la detección de errores de manera temprana, además de verificar si algún cambio en la implementación de algún módulo en la capa modelo puede producir errores a otra parte del sistema.

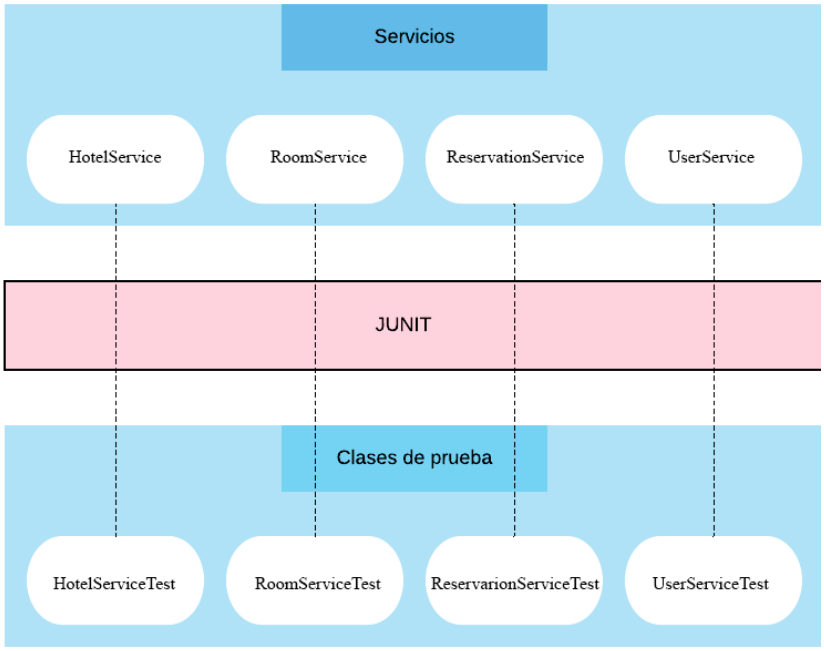


Figura 10.1: Clases de prueba y Servicios probados

es.udc.hotelapp.backend.model.entities	93,8 %	2.032	135
CustomizedRoomTypeReservationDaoI	90,5 %	523	55
CustomizedRoomDaoImpl.java	91,8 %	169	15
CustomizedGuestReservationDaoImpl.	90,8 %	119	12
CustomizedProductDaoImpl.java	90,8 %	118	12
CustomizedServiceDaoImpl.java	90,8 %	118	12
CustomizedHotelDaoImpl.java	93,7 %	134	9
AccountItem.java	86,4 %	51	8
Guest.java	93,3 %	56	4
GuestReservation.java	86,7 %	26	4
User.java	96,7 %	117	4
Account.java	100,0 %	79	0
Hotel.java	100,0 %	103	0
Photo.java	100,0 %	30	0
Product.java	100,0 %	50	0
Room.java	100,0 %	65	0
RoomReservation.java	100,0 %	50	0
RoomType.java	100,0 %	20	0
RoomTypePrice.java	100,0 %	40	0
RoomTypeReservation.java	100,0 %	70	0
Service.java	100,0 %	50	0
Status.java	100,0 %	44	0
es.udc.hotelapp.backend.model.services	97,8 %	1.560	35
PermissionCheckerImpl.java	91,2 %	186	18
ReservationServiceImpl.java	97,6 %	524	13
Block.java	95,5 %	85	4
HotelServiceImpl.java	100,0 %	504	0
RoomServiceImpl.java	100,0 %	152	0
UserServiceImpl.java	100,0 %	109	0

Figura 10.2: Cobertura de las pruebas de integración

Para implementar estas pruebas en la aplicación se ha utilizado **JUnit**. Este conjunto de librerías permite que las pruebas se ejecuten de forma automática. Se han creado las clases de prueba que se muestran en la figura 10.1. Estas comprueban el correcto funcionamiento de los servicios. Se anotan con *@Test*, para ser identificadas por *JUnit* y permiten verificar el resulta-

do esperado ante la correcta ejecución de un servicio. También es necesario comprobar casos de fallo provocando ejecuciones que generan excepciones. En estos casos se añaden casos de prueba en donde se comprueba mediante un *assertThrows* que se lanza la excepción correcta, indicando dentro del *assert* la clase de excepción que se espera que se lance. Durante la ejecución de cada caso de prueba se genera un entorno completo de prueba, abriendo una nueva transacción en la Base de Datos (BBDD). Para esto se anota la clase como *@Transactional*, para que al terminar la ejecución y validación de cada caso de prueba se realice una llamada de *Rollback* de la transacción, dejando así la BBDD en el estado anterior a esta ejecución.

Se ha analizado el nivel de cobertura de los tests, intentando cubrir el mayor número de casos posibles para detectar los posibles errores y en la figura 10.2 se pueden ver los resultados, aquí podemos observar el porcentaje de líneas de código cubiertas por las pruebas de integración verificando así lo correcto, confiable y efectivo que es el código desarrollado. Además, se puede observar que todos los servicios probados y las entidades tienen una cobertura superior al 86%, que se considera relativamente alta.

10.3 Pruebas sobre API Rest

Con el objetivo de probar el correcto funcionamiento de la *Api Rest* se utiliza la herramienta *Insomnia*. Proporciona una interfaz gráfica que permite simular peticiones *HTTP* sobre el servidor. De esta forma se comprueba que el resultado de la ejecución es el esperado. Probando la *Api Rest*, se prueban de forma indirecta las capas inferiores del *Backend*, comprobando su correcto funcionamiento. En la figura 10.3, se puede ver el resultado de una prueba realizada con *Insomnia*, donde se requieren todas las habitaciones que están ocupadas de un hotel determinado. Estas pruebas son realizadas manualmente.

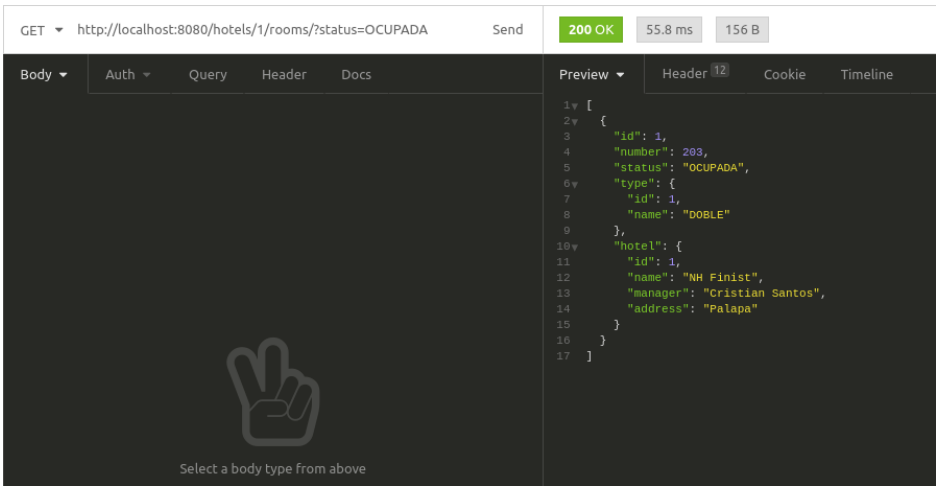


Figura 10.3: Petición GET sobre HTTP en Insomnia

10.4 Pruebas de Aceptación

Tras completar el desarrollo de la aplicación, es necesario comprobar el correcto funcionamiento de esta y el cumplimiento de los requisitos marcados por el cliente. Este último suele ser el encargado de la realización de las pruebas de aceptación. Estas son realizadas mediante la interfaz de usuario de forma manual y abarcan todas las funcionalidades desarrolladas dentro de la aplicación. Algunos ejemplos de las pruebas realizadas son las siguientes:

- Añadir un hotel
  - Se accede con la cuenta del administrador
  - Se crea la cuenta del gerente o responsable del hotel
  - Se pulsa el botón añadir hotel sin introducir ningún dato, recibiendo un error con los campos requeridos.
  - Se cumplimenta el formulario de forma correcta y se añade el hotel.
  - Se cierra sesión
  - Se registra un cliente y se accede al sistema, visualizando todos los hoteles disponibles.
- Registrar un usuario individual
  - Se pulsa el botón de *Registrarse*
  - Se pulsa el botón de *Aceptar*, sin introducir ningún dato.
  - Se muestra un error por la falta de los datos requeridos.
  - Se añaden los datos sin añadir una *@* dentro del *email*, recibiendo el error por la falta del *@* en el *email*.
  - Se añaden correctamente los datos.
  - Se accede con el usuario recién registrado a la aplicación.
  - Se pulsa el nombre de usuario para acceder a la opción de *Actualizar perfil*.
  - Se comprueba que toda la información es correcta.
  - Se cierra sesión.

Capítulo 11

Conclusiones y futuras líneas de trabajo

En este trabajo de fin de grado se ha diseñado e implementado una aplicación web para la gestión de hoteles de pequeño tamaño.

11.1 Conclusiones

Para concluir se analizan los objetivos que se pretendían alcanzar con la realización de este proyecto y su nivel de cumplimiento. En particular, con la aplicación desarrollada se han logrado las siguientes metas:

- Favorecer la comunicación entre los clientes potenciales y los hoteles de pequeño tamaño
- Agrupar en una única aplicación la gestión de las reservas y las habitaciones de los hoteles.
- Aumentar la visibilidad de los hoteles de pequeño tamaño, aunque a veces los gerentes o propietarios de los mismos sean reticentes a entrar en el mundo *online*.
- Facilitar la adquisición de los productos o servicios que un hotel de estas características ofrece a los huéspedes.

Las funcionalidades planteadas e implementadas en el sistema desarrollado, están enfocadas a satisfacer los objetivos previamente establecidos. Puesto que todas las funcionalidades nombradas han sido desarrolladas completamente, se puede concluir que los requisitos funcionales se han cumplido. Adicionalmente se pretendía desarrollar un sistema modular, con

componentes independientes que facilitase la introducción de cambios en un futuro, el mantenimiento y la comprensión del propio sistema. Estos objetivos se han tenido en cuenta en la fase de diseño del sistema y se han aplicado en su desarrollo.

11.2 Futuras líneas de trabajo

Partiendo de las funcionalidades desarrolladas y teniendo en cuenta las carencias de los hoteles de pequeño tamaño, surgen nuevas funcionalidades que se pueden incorporar para satisfacer estas carencias. En particular se han valorado las siguientes:

- La integración de algunos sistemas de pago, como pueden ser Paypal u otros similares.
- La integración con servicios externos, como por ejemplo la notificación de clientes a la Policía o Guardia Civil.
- Permitir a los gerentes de los hoteles la incorporación de las habitaciones mediante el volcado de datos incluido dentro de un fichero *csv*. De este modo se facilitaría el migrado de datos desde otras aplicaciones a la aplicación aquí desarrollada.

Apéndices

Apéndice A

Material adicional

A.1 Instalación de Software

En este apartado se detalla el proceso de instalación y ejecución del software desarrollado. Para la ejecución de la aplicación es necesario tener instalado en primer lugar:

- Un sistema gestor de Bases de Datos MySql 8.0 o superior
- Un servidor Apache

A continuación se crea la Base de Datos necesaria para la ejecución de la aplicación. Para ello se realizan los siguientes pasos:

1 Ejecutar en un terminal: **$mysqladmin -u root -p** e introducir la contraseña del usuario root si se ha especificado durante la instalación de *Mysql*.

2 Ejecutar **$ create hotelappproject**

3 Ejecutar **$ create user ”hotelapp@localhost” identified by ”hotelapp”**

4 Ejecutar **$ grant all privileges on hotelappproject to hotelapp@localhost with grant option**

5 Ejecutar source /path/to/1-MySQLCreateTables.sql

6 Ejecutar source /path/to/2-MySQLCreateData.sql

Para el despliegue del *Backend*:

- Abrimos un terminal con la ruta donde descargamos el archivo *hotel-app-backend-1.0.0.jar*
- Ejecutamos el comando **$ java -jar hotel-app-backend-1.0.0.jar**

Para realizar ahora el despliegue del *Frontend*, añadimos todos los archivos que lo conforman al directorio */var/www/html* de la máquina linux usada. A mayores, es necesario añadir en el archivo */etc/apache2/sites-enabled/000-default.conf* lo siguiente:

<Directory "/var/www/html">
AllowOverride All
</Directory>

Por último, se deben reiniciar los servicios de apache, con los comandos **$ sudo a2enmod rewrite** y **$ sudo service apache2 restart**.

Para acceder a la aplicación, es suficiente con abrir un navegador y acceder a **localhost**.

A.2 Manual de Usuario

En este apartado se describen los pasos a llevar a cabo para realizar las distintas funcionalidades que proporciona el sistema. Para ello, estas se han dividido en seis grupos:

- Acceso y Registro: Funcionalidades de registro y acceso a la aplicación
- Gestión de Hoteles: Funcionalidades de creación, búsqueda, modificación y eliminación de los hoteles.
- Gestión de Habitaciones y Precios de Habitaciones: Gestión de la creación, búsqueda, modificación y eliminación de los precios, así como de la creación, búsqueda y modificación de la información de las habitaciones.
- Gestión de Reservas y Clientes: Funcionalidades de creación y búsqueda tanto de reservas como de clientes, así como modificación de estas.
- Gestión de Productos y Servicios: Añadir, consultar, modificar, eliminar y adquirir productos y servicios pertenecientes a los hoteles.
- Gestión de fotos: Añadido de fotos para los hoteles.

A.2.1 Acceso y Registro

En la figura A.1 se puede ver la pantalla inicial de la aplicación. Se puede iniciar sesión, mediante la opción de *Autenticarse*, si ya se ha creado, o nos han creado una cuenta cómo se puede ver en la figura siguiente A.2. También hay la opción de registrarse desde esta pantalla mediante el botón de *Registrarse*.

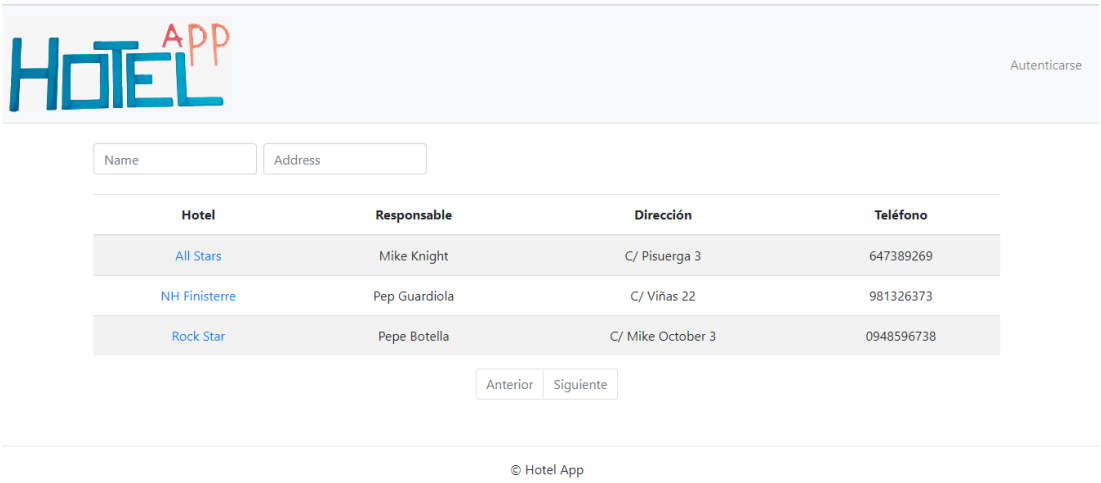


Figura A.1: Página principal

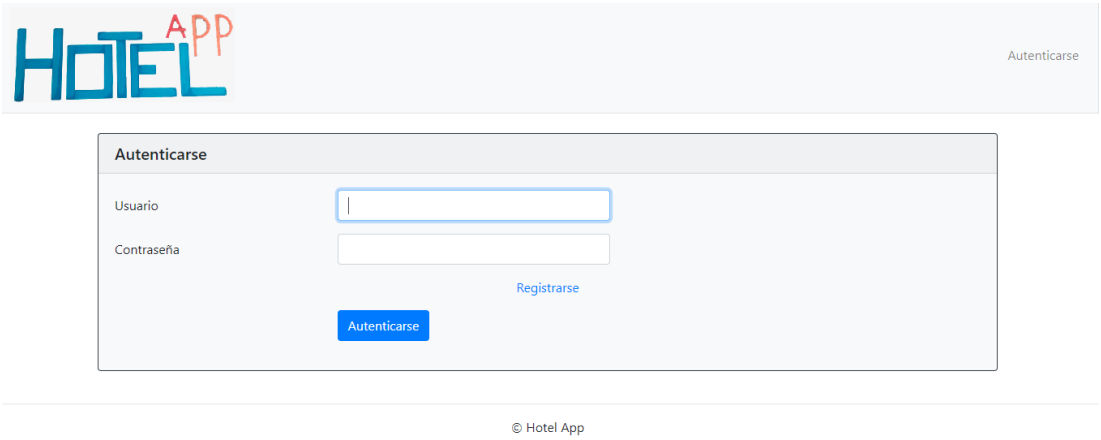


Figura A.2: Inicio de Sesión

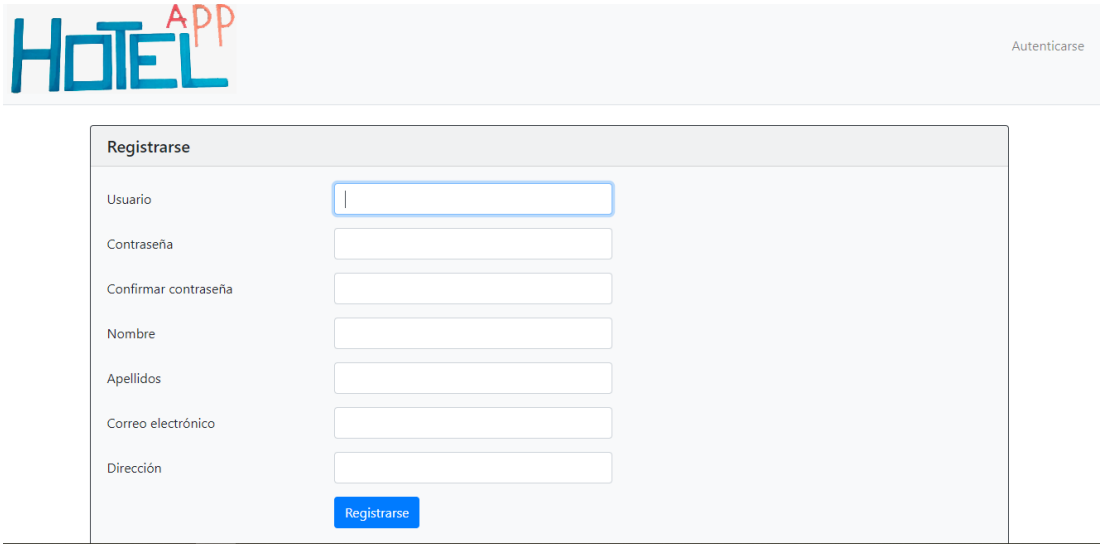


Figura A.3: Registro Individual

En la figura A.3 se muestra la pantalla de registro de un usuario individual, dado que las credenciales del personal del hotel deben ser proporcionadas por el *administrador* de la aplicación, el cual va a crear la cuenta de usuario de los gerentes del hotel al hacer click en el enlace *añadir manager* que se muestra en el menú del administrador. Por otro lado, el manager va a crear las cuentas del personal del hotel, del mismo modo que el administrador. Para registrarnos o que alguien nos registre, es necesario completar toda la información que se nos requiere, como se puede ver en la figura A.3.

Tras haber añadido la información requerida, el sistema automáticamente, nos redirige a la página inicial de la aplicación, en donde podemos observar los diferentes menús, pudiendo ser para un usuario anónimo A.7, para un usuario individual A.6, para un usuario del personal del hotel A.5, para un gerente A.8 y para el administrador A.4.

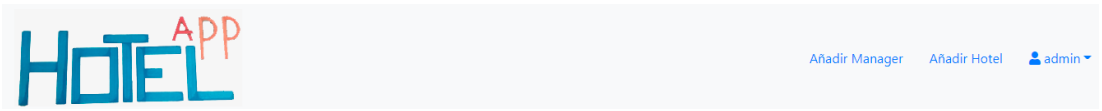


Figura A.4: Menú usuario admin

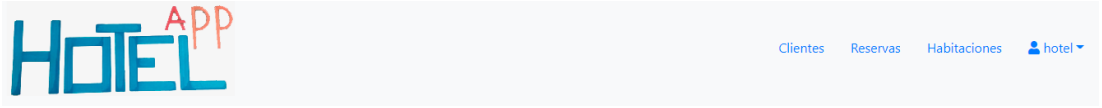


Figura A.5: Menú usuario hotel

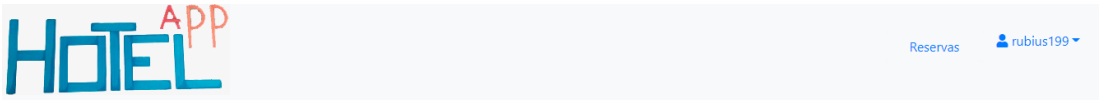


Figura A.6: Menú usuario individual

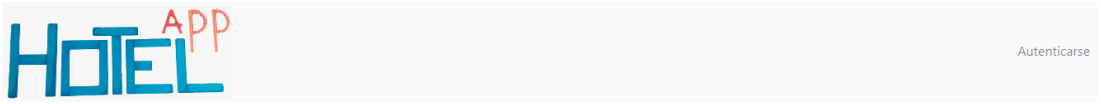


Figura A.7: Menú usuario anónimo

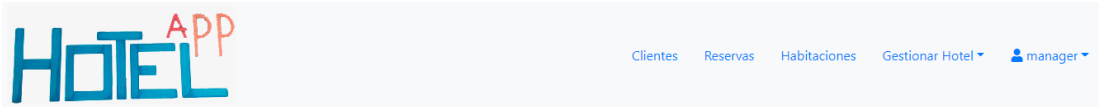


Figura A.8: Menú usuario manager

Para poder modificar cualquier dato erróneo de nuestro perfil, debemos acceder al nombre de usuario del menú y en el desplegable, seleccionar *Actualizar perfil* o si deseamos cambiar nuestra contraseña, seleccionamos *Cambiar contraseña*. En esta figura A.10, se muestra la información que consta en el sistema actualmente para nuestro usuario, pudiendo ser modificada y almacenada de nuevo mediante el botón *Actualizar*. En la imagen A.9 se muestra la información necesaria para realizar el cambio de contraseña, siendo similar a la actualización del perfil.

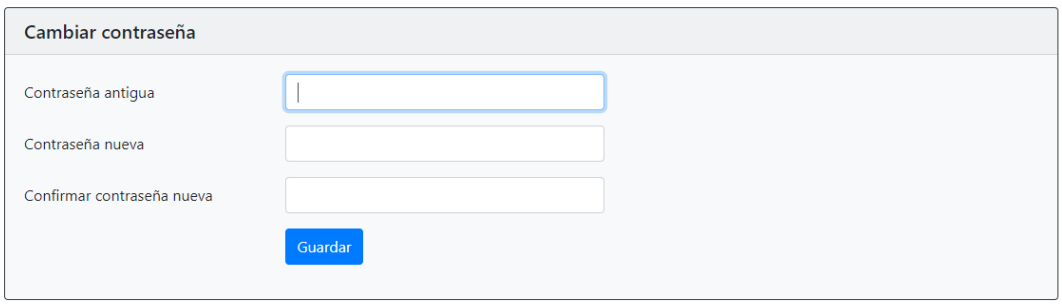


Figura A.9: Cambio de Contraseña

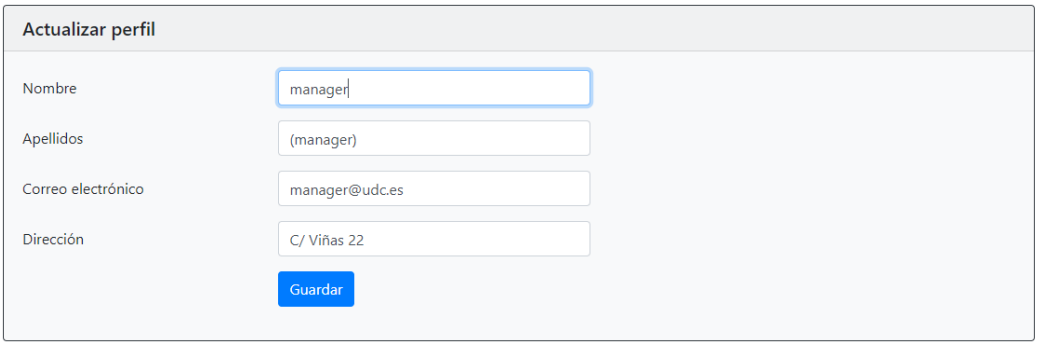


Figura A.10: Actualizar Perfil

A.2.2 Gestión de Hoteles

Para poder añadir un hotel, es necesario iniciar sesión con un perfil de *administrador*, dado que son los únicos usuarios que pueden realizar esta operación. Una vez autenticados, seleccionamos en el menú superior la opción de *Añadir Hotel*. Se muestra el formulario para añadir un hotel, véase figura A.11, en donde debemos cumplimentar los campos requeridos y una vez hayamos terminado pulsar en el botón de *Registrar*.

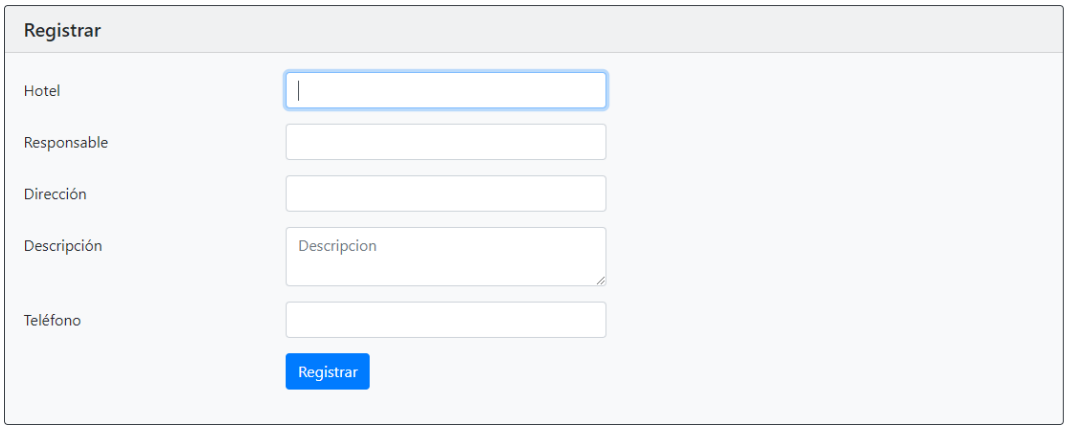


Figura A.11: Añadir Hotel

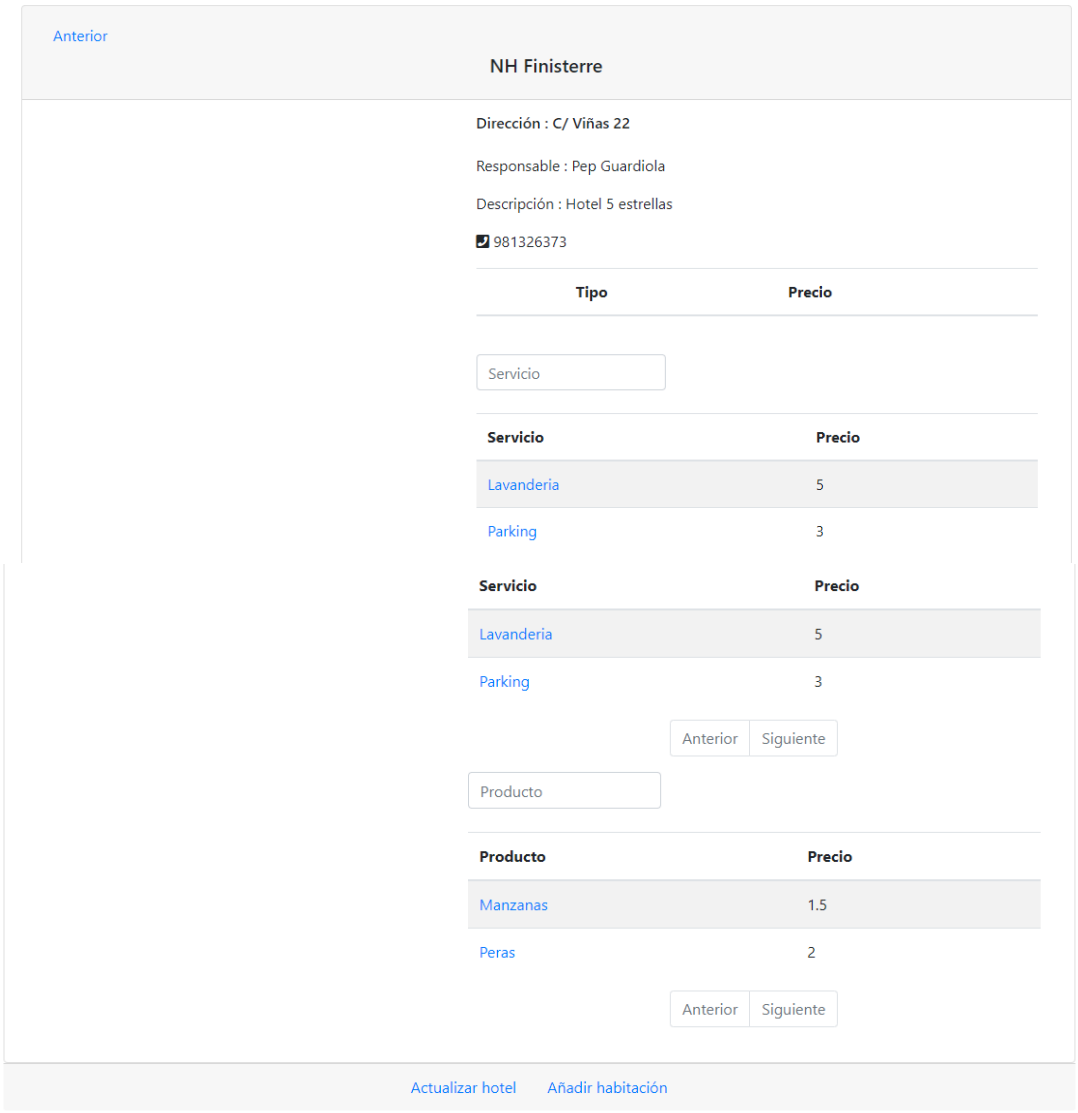


Figura A.12: Detalle Hotel

Para poder editar o eliminar los hoteles, debemos acceder al detalle del cual deseamos modificar o eliminar, véase figura A.12, y seleccionamos una de las opciones que nos aparecen en el menú inferior. Si hemos seleccionado la opción de editar, se nos mostrará el formulario que ya conocemos, esta vez cumplimentado con la información que hay actualmente en la aplicación, como se puede ver en la figura A.13.

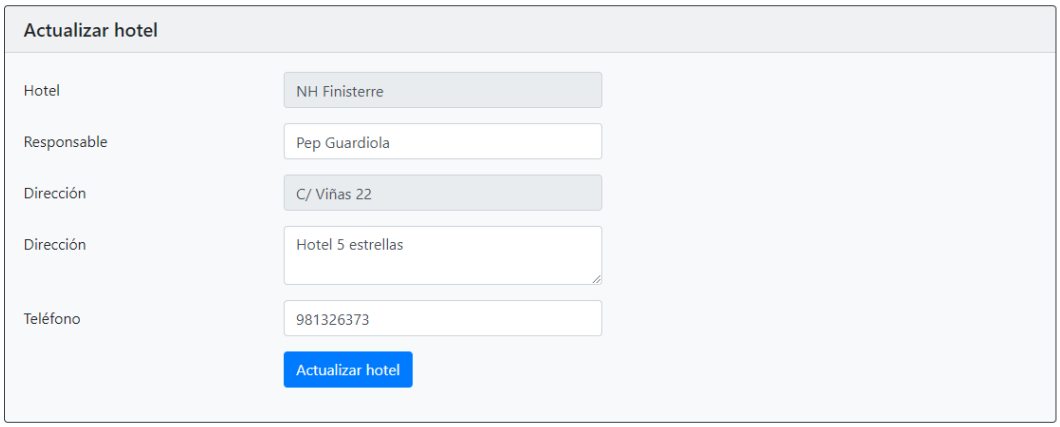


Figura A.13: Actualizar Hotel

Si por el contrario, hemos decidido eliminar el hotel en cuestión, se nos mostrará un mensaje emergente en donde se nos pide la confirmación para la eliminación de este, véase figura A.14.

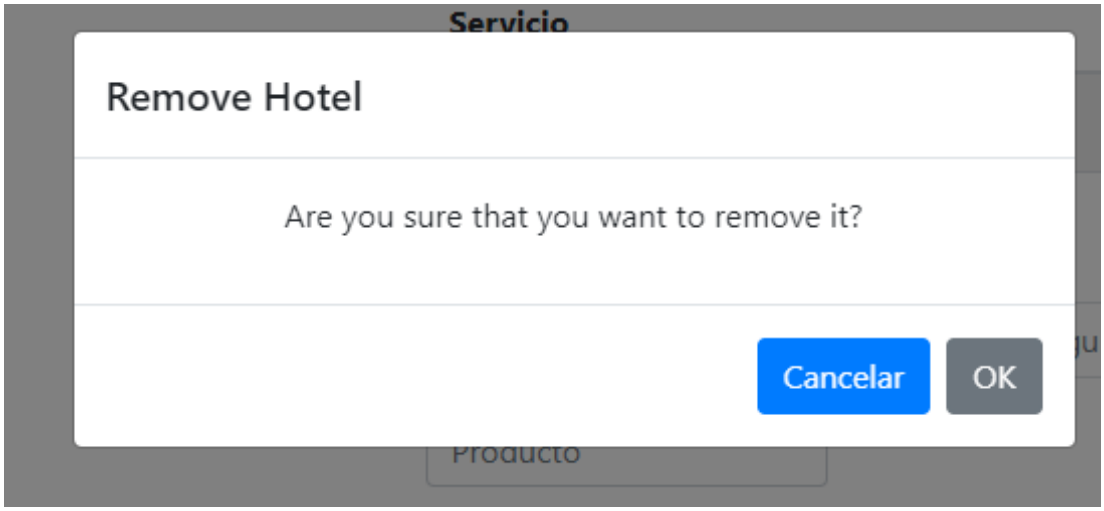


Figura A.14: Confirmación eliminación

En caso de aceptar este mensaje o de almacenar las modificaciones en la información del hotel, la aplicación nos redirige a la página inicial.

Sin importar el perfil del que disponemos, se van a poder filtrar los hoteles, tanto por el nombre de este, como por la dirección, como se puede ver en la imagen A.15.

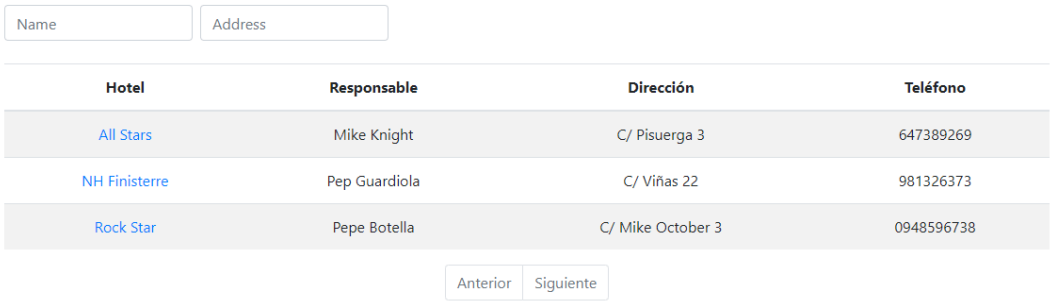


Figura A.15: Listado Hotel

Para ver el detalle de un hotel, debemos estar autenticados, sin importar el perfil, y pulsar sobre el nombre del hotel. Se mostrará una pantalla donde aparecerá una foto del hotel, el listado de precios, la información ya conocida y los productos y servicios disponibles, como se puede ver en la figura A.12.

A.2.3 Gestión de Habitaciones y Precios de Habitaciones

Para poder añadir tanto las habitaciones como los precios del tipo de estas, es necesario tener un perfil de *manager* al acceder a la aplicación. Una ve autenticados, accedemos al detalle del hotel, dentro de la lista que se muestra y seleccionamos la opción de *Añadir Habitación* en el menú inferior. Se mostrará el formulario de añadir habitación, véase figura A.16, donde debemos cubrir todos los datos y pulsar el botón de *Añadir Habitación*. En caso de no producirse ningún error, aparece el listado de todas la habitaciones, véase figura A.17.

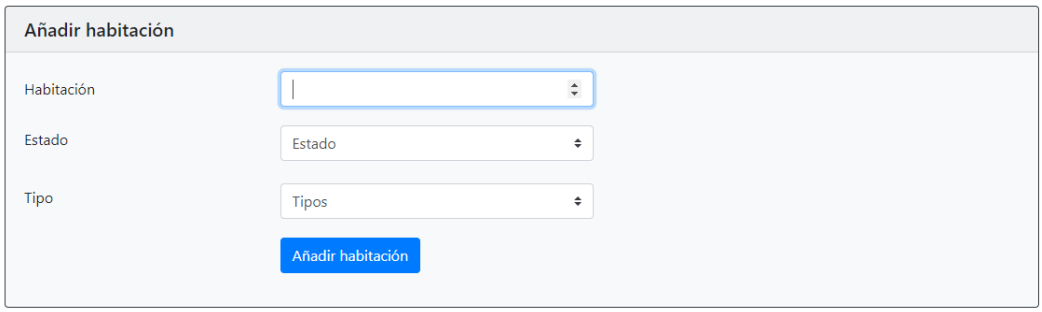


Figura A.16: Añadir Habitación

Figura A.17: Listado Habitaciones

Si por el contrario, deseamos añadir un precio de un tipo de habitación, debemos acceder a *Gestionar Hotel - Añadir Precio*, dentro del menú superior. Al acceder, se mostrará el formulario para añadir los precios, véase figurar A.18, en donde debemos seleccionar el tipo de habitación e introducir el precio del mismo. Al terminar debemos pulsar *Guardar*. Si no se producen errores, aparece el detalle del hotel en el que trabajamos, véase figura A.12.

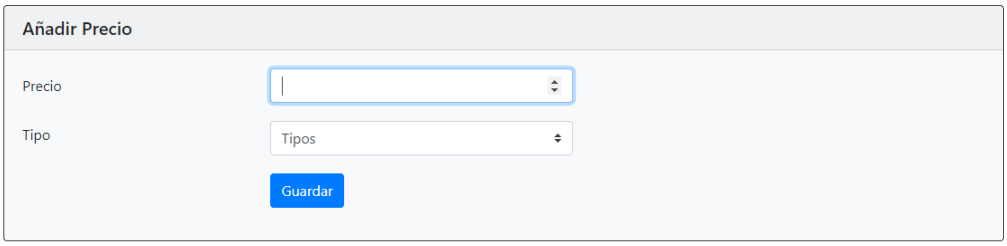


Figura A.18: Añadir Precio

Figura A.19: Detalle Habitación

Para consultar la información de las habitaciones, debemos acceder desde el menú superior a *Habitaciones*, donde aparecerá un listado, donde se podrá filtrar por el estado de las

mismas, véase figura A.17. Si detectamos cualquier anomalía en las habitaciones mostradas, podemos acceder al detalle pulsando en el número de la misma. Se mostrará la información de la habitación, véase figura A.19, junto con un menú en donde se podrán realizar modificaciones o eliminaciones, si la habitación ha dejado de existir por reformas.

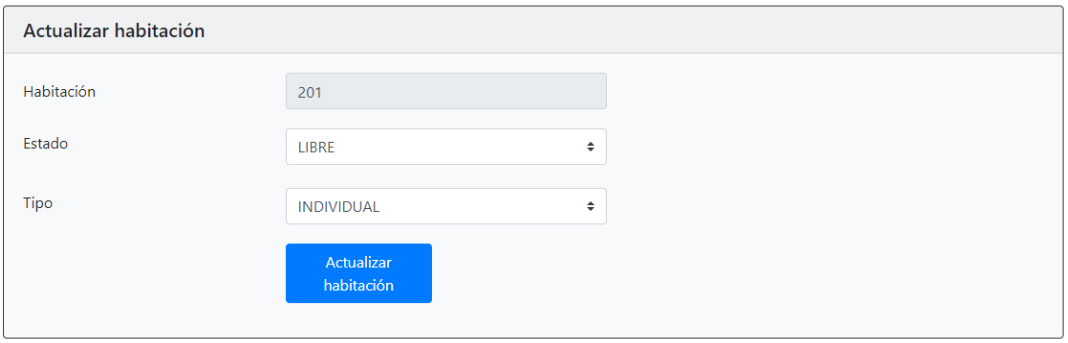


Figura A.20: Actualizar Habitación

Si accedemos a la opción de modificación, similar a la modificación de los hoteles, se mostrará el formulario cumplimentado de añadir habitación, véase figura A.20. Si por el contrario deseamos realizar la eliminación, el proceso será similar a la eliminación de un hotel.

Para poder consultar el listado de precios de los tipos de habitación, debemos acceder al detalle del hotel, como se indicó en el apartado anterior.

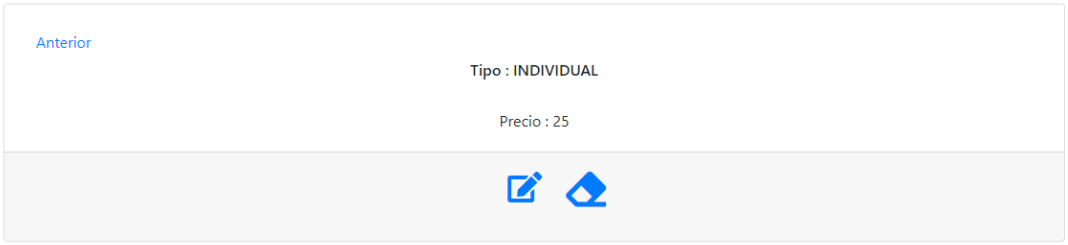


Figura A.21: Detalle Precio Habitación

Para poder modificar o eliminar estos precios debemos estar autenticados con un perfil de *manager u hotel* y pulsar sobre el identificador que aparece en la tabla. Se mostrará la misma información en la pantalla que aparece junto con el menú inferior ya conocido A.21. Este menú tiene las mismas funcionalidades que el menú de las habitaciones o el de los hoteles. Si accedemos a la opción de modificar, se muestra el formulario de añadir precio cumplimentado, donde solo se puede modificar el precio, no el tipo de habitación, véase figura A.22.

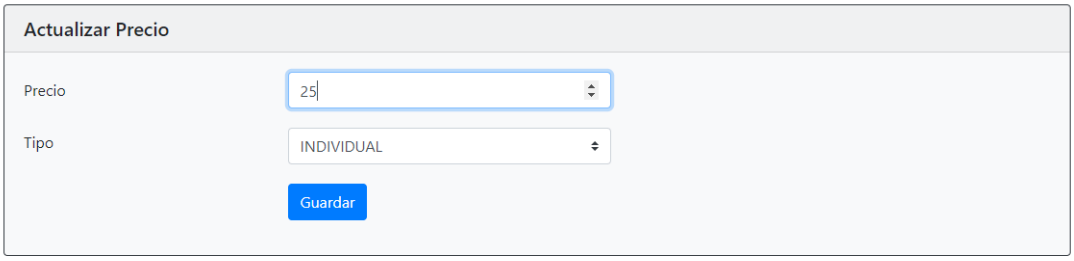


Figura A.22: Actualizar precio

Si por el contrario deseamos eliminar ese precio, por no tener más ese tipo de habitaciones, el proceso es igual al de eliminar hoteles, con la excepción de que este nos devuelve al detalle del hotel.

A.2.4 Gestión de Reservas y Clientes

Para realizar una reserva, debemos estar autenticados como *usuario individual*, acceder al detalle del hotel en el que deseamos alojarnos y seleccionar en el menú inferior la opción de *Reservar*, aparecerá un formulario, véase figura A.23, que debemos cumplimentar y al terminar pulsar en *Reservar*. Si no se producen errores, se nos mostrará el listado de reservas que hemos realizado A.24. Pudiendo ver el detalle de estas, accediendo al identificador, como se puede ver en la figura A.25, donde podremos modificar la reserva A.26 o ver nuestra cuenta de gastos A.27.

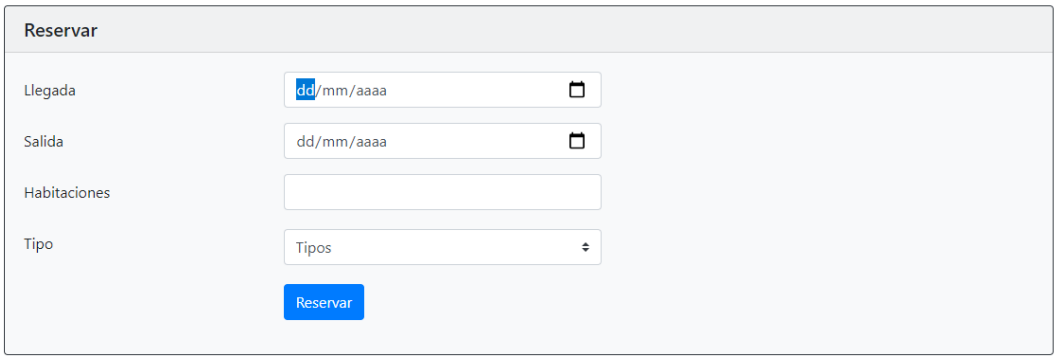


Figura A.23: Reservar

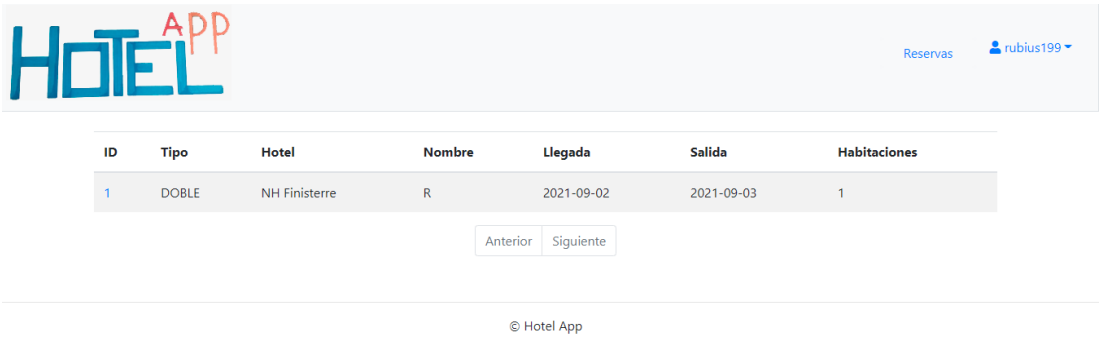


Figura A.24: Listado Reservas Cliente

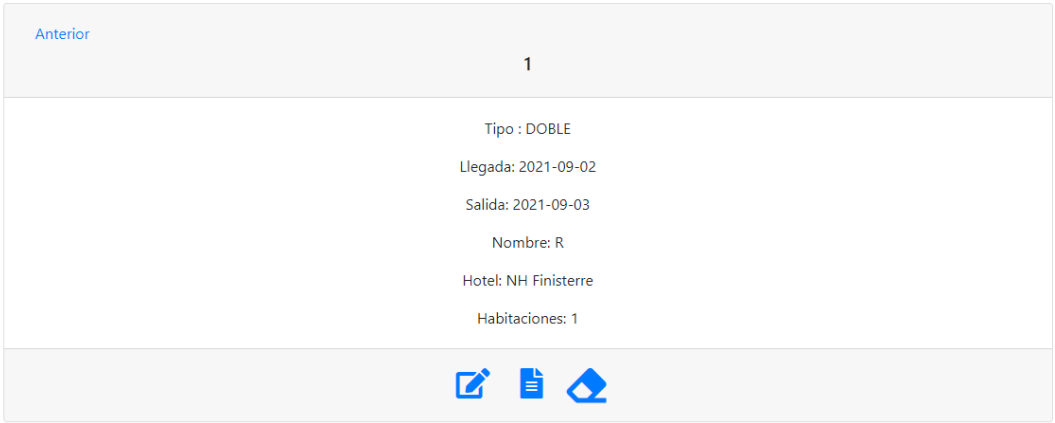


Figura A.25: Detalle Reserva Cliente

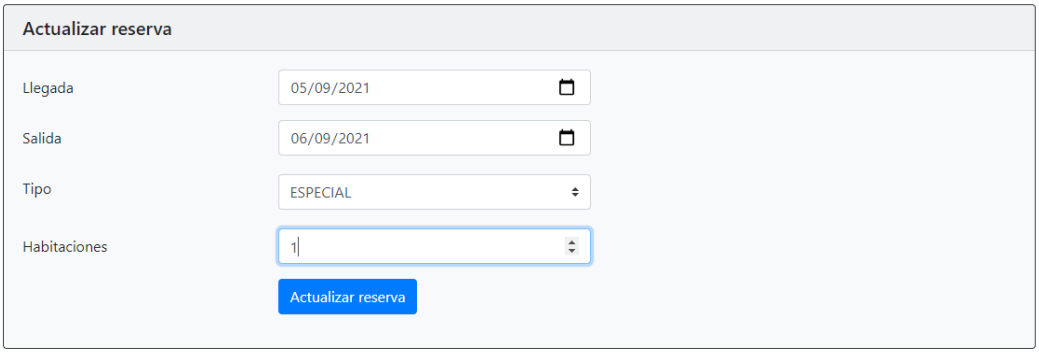


Figura A.26: Actualizar Reserva

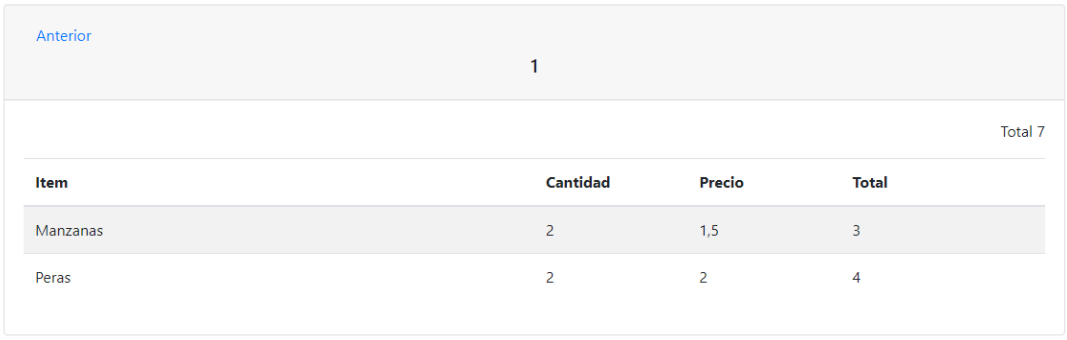


Figura A.27: Detalle Cuenta de Gastos

Si por el contrario, estamos autenticados como *manager u hotel*, para ver el listado desde el menú superior en la opción de *Reservas*, aparecerán además dos cuadros de búsqueda que permitirán filtrar las reservas por el nombre del cliente o por la fecha que se desee, véase figura A.28.

Figura A.28: Listado Reservas Hotel

Por otro lado, si accedemos del mismo modo que un usuario individual al detalle de una reserva, nuestro menú contendrá dos opciones más como son la de asignar una habitación y la opción de añadir los clientes, véase figura A.29.

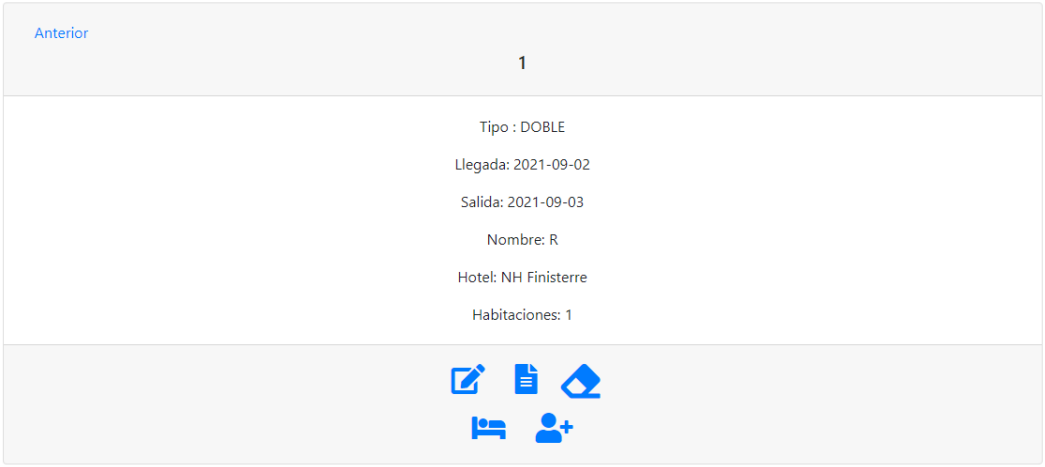


Figura A.29: Detalle Reserva Hotel

Figura A.30: Asignar Habitación

Al intentar asignar una habitación, se mostrará un selector con las habitaciones disponibles en ese instante, véase figura A.30, donde debemos elegir la que deseamos y posteriormente pulsar en el botón *Guardar*. Si todo ha ido bien, nos devuelve al listado de reservas.

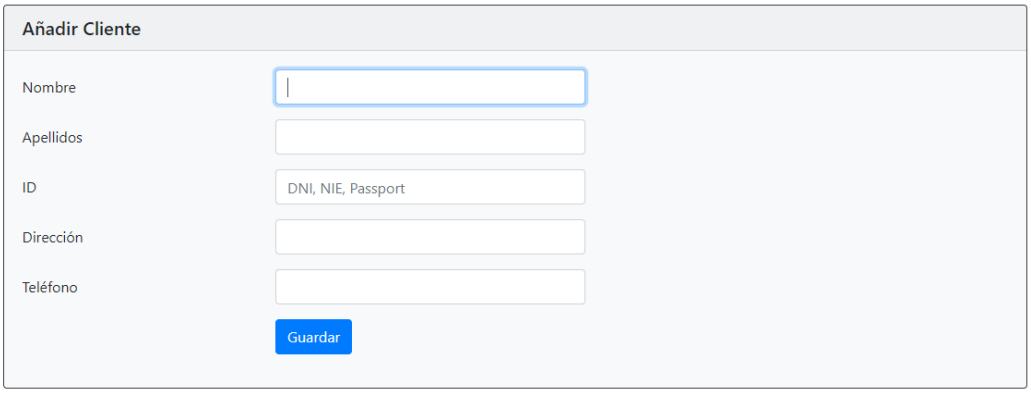


Figura A.31: Añadir Cliente

Cuando vamos a añadir a un cliente, pulsando en la opción del usuario con el "+", se mostrará el formulario, véase A.31, a rellenar. Al terminar, debemos almacenar los datos en la aplicación mediante el botón de *Guardar*. Si todo ha ido bien, deberíamos ver el listado de los clientes, donde podemos filtrar por el nombre del mismo, véase figura A.33.

Cuando vamos a ver el gasto que tuvimos en una estancia, se mostrará un listado de los productos o servicios adquiridos, el precio de la estancia en la habitación y el precio total de gasto, vease figura A.27. En cambio si no se ha producido gasto, se mostrará el mensaje de la imagen A.32.

Figura A.32: Cuenta Vacía

Para acceder al listado de clientes, siendo *manager u hotel*, debemos acceder desde el menú superior en la opción de *clientes*, véase figura A.33.

Figura A.33: Listado Clientes

A.2.5 Gestión de Productos y Servicios

Las operaciones para estos son similares, por eso solo se va a comentar para los productos.

Para añadir productos o servicios, debemos estar autenticados con un perfil de *manager*. Para añadir un producto, debemos acceder al menú superior *Gestionar Hotel - Añadir Producto*. Aparecerá un formulario, véase figura A.34, que deberemos cumplimentar y pulsar el botón de *Guardar*. Si no recibimos ningún error, se mostrará el detalle del hotel en el que trabajamos.

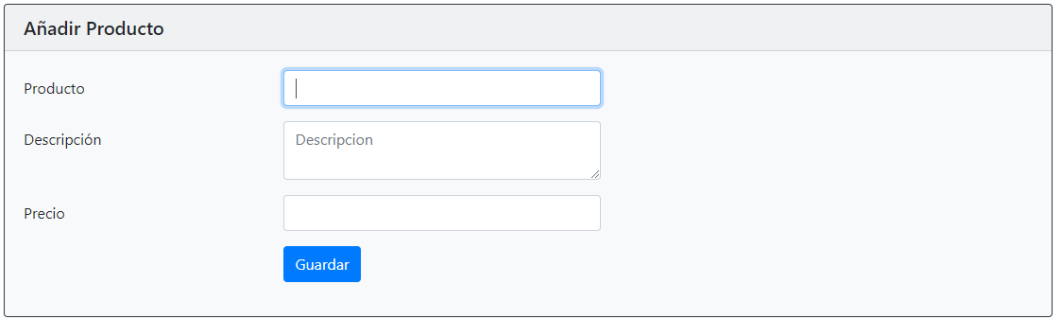


Figura A.34: Añadir Producto/Servicio

Si deseamos ver el listado de productos o servicios, debemos acceder al detalle del hotel explicado en el apartado A.2.2.

Para acceder al detalle de cada producto o servicio, basta con pulsar en el nombre de este, y se mostrará la información que ya conocemos junto a la descripción y en caso de ser un usuario *individual*, con reserva activa en ese hotel, se podrá realizar la adquisición de este producto o servicio, indicando la cantidad y pulsando en *Añadir a cuenta*, véase figura A.35.

Figura A.35: Detalle Producto/Servicio

En caso contrario, solo aparecerá la información y el menú ya conocido de modificar y eliminar. El funcionamiento de la modificación y eliminación es igual al de los precios de los tipos de habitaciones, solo varía la información que se muestra en caso de modificar los productos, como se puede ver en la figura A.36.

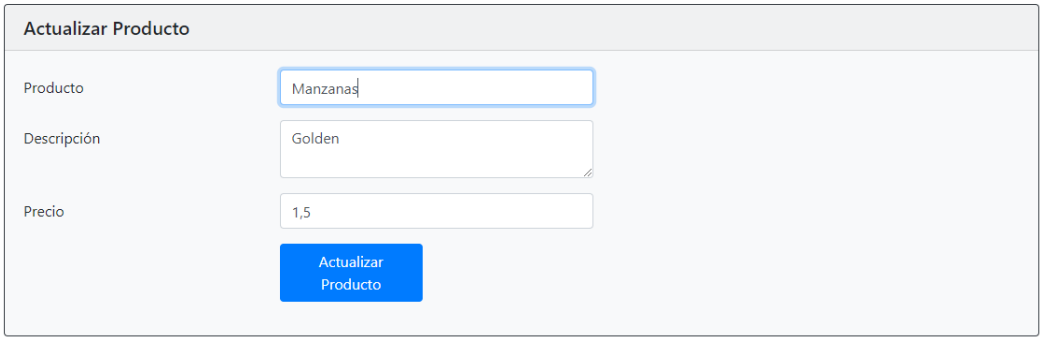


Figura A.36: Actualizar Producto/Servicio

A.2.6 Gestión de fotos

Para añadir una foto al hotel, debemos estar autenticados con el perfil de *manager*. Una vez hemos accedido, debemos seleccionar en el menú superior la opción de *Gestionar Hotel - Añadir Foto*, donde nos aparecerá un formulario para que seleccionemos la foto y al terminar pulsemos el botón de *Guardar*, como se puede ver en la imagen A.37. Si todo sale bien, nos redirige al detalle del hotel.

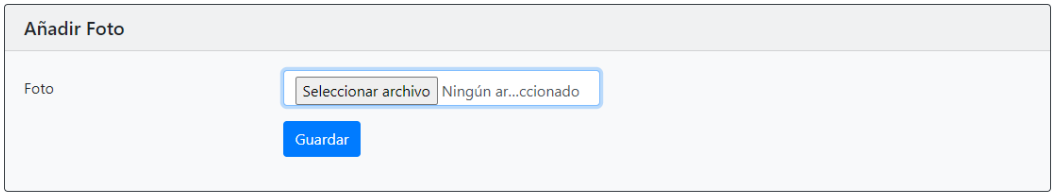


Figura A.37: Añadir Foto

Lista de acrónimos

HTTP *HyperText Transfer Protocol.*

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

DTO *Data Transfer Object.*

DAO *Data Access Object.*

JDBC *Java DataBase Connectivity.*

JSON *JavaScript Object Notation.*

JPQL *Java Persistence Query Language.*

IDE *Integrated Development Environment.*

BBDD *Base de Datos.*

API *Application Programming Interface.*

SQL *Structured Query Language.*

JRE *Java Runtime Environment.*

XML *Extensible Markup Language.*

REST *Representational State Transfer.*

PUDS *Proceso Unificado de Desarrollo de Software.*

SPA *Single Page Application.*

Glosario

Bytecode Código independiente de la máquina que generan compiladores de determinados lenguajes (Java, Erlang,...) y que es ejecutado por el correspondiente intérprete.

Frontend Aplicación del lado cliente de una aplicación web.

Backend Aplicación del lado servidor de una aplicación web.

Framework Estructura conceptual y tecnológica de soporte definido, con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

Bibliografía

[1] Roomba. (Accedido el: 05-09-2021). [En línea]. Disponible en: https://sourceforge.net/projects/roomba/

[2] Sirvoy. (Accedido el: 31-08-2021). [En línea]. Disponible en: https://sirvoy.com/

[3] Green hotel. (Accedido el: 31-08-2021). [En línea]. Disponible en: https://www.greensoft.es/greenhotel/

[4] Timon hotel. (Accedido el: 31-08-2021). [En línea]. Disponible en: https://timonhotel.com/

[5] Admin tour. (Accedido el: 31-08-2021). [En línea]. Disponible en: https://admintour.com/modulos-y-herramientas-para-la-hoteleria/pms-hotelero-software-para-hoteles/

[6] Oracle. Introduction to java. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://docs.oracle.com/javase/tutorial/

[7] Sql. (Accedido el: 30-08-2021). [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Glossary/SQL

[8] Jpa query structure (jpql / criteria). (Accedido el: 20-08-2021). [En línea]. Disponible en: https://www.objectdb.com/java/jpa/query/jpql/structure

[9] Html. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML

[10] Css. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS

[11] Javascript. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

[12] Spring. Springboot overview. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://spring.io/projects/spring-boot

[13] Spring. (Accedido el: 20-08-20219. [En línea]. Disponible en: https://spring.io/

[14] Spring-data. (Accedido el: 30-08-2021). [En línea]. Disponible en: https://spring.io/projects/spring-data

[15] Spring-web. (Accedido el: 30-08-2021). [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/package-summary.html

[16] React. Introduction to react. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://es.reactjs.org/docs/getting-started.html

[17] react-intl. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://github.com/formatjs/formatjs

[18] react-router-dom. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://www.npmjs.com/package/react-router-dom

[19] Bootstrap. Getting started. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://getbootstrap.com/docs/4.5/getting-started/introduction/

[20] Redux. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://es.redux.js.org/

[21] Font awersome icons. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://fontawesome.com/v5.15/icons?d=gallery&p=2

[22] Junit. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://junit.org/junit5

[23] Insomnia. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://insomnia.rest/

[24] Apache maven. (Accedido el: 20-08-2021). [En línea]. Disponible en: http://maven.apache.org/what-is-maven.html

[25] Git. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://git-scm.com/

[26] Eclipse. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://www.eclipse.org/

[27] Visual studio code. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://code.visualstudio.com/

[28] Yarn. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://yarnpkg.com/

[29] Babel. (Accedido el: 22-08-2021). [En línea]. Disponible en: https://babeljs.io/

[30] Apache tomcat. (Accedido el: 30-08-2021). [En línea]. Disponible en: http://tomcat.apache.org/

[31] Webpack. (Accedido el: 22-08-2021). [En línea]. Disponible en: https://webpack.js.org/

[32] Mysql. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://www.mysql.com

[33] Mozilla. http. (Accedido el: 30-08-2021). [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP

[34] Que es jdbc? (Accedido el: 20-08-2021). [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSGU8G_11.70.0/com.ibm.jdbc_pg.doc/ids_jdbc_011.htm

[35] Spring-data-jpa. (Accedido el: 30-08-2021). [En línea]. Disponible en: https://spring.io/projects/spring-data-jpa

[36] Spa. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://desarrolloweb.com/articulos/que-es-una-spa.html

[37] Json. json. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://www.json.org/json-en.html

[38] Puds. (Accedido el: 20-08-2021). [En línea]. Disponible en: https://sites.google.com/site/businesscontrolesi/productos-software/metodologia-del-trabajo/pud

[39] Paging and sorting repository. (Accedido el: 29-08-2021). [En línea]. Disponible en: https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html