Trabajo Fin de Grado
Ingeniería de Telecomunicación

Desarrollo de una aplicación web de reservas y gestión hotelera

Autor: Sebastiá Guisasola Benítez

Tutor: Joaquín Rodrigo Fernández Valverde

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Ingeniería de Telecomunicación

Desarrollo de una aplicación web de reservas y gestión hotelera

Autor:
Sebastiá Guisasola Benítez

Tutor:
Joaquín Rodrigo Fernández Valverde
Profesor

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Desarrollo de una aplicación web de reservas y gestión hotelera

Autor: Sebastiá Guisasola Benítez

Tutor: Joaquín Rodrigo Fernández Valverde

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Resumen

La popularización y auge del turismo han facilitado el surgimiento de nuevos modelos de alojamiento, tales como la vivienda de uso turístico, y de nuevas plataformas que entran en el mercado ofreciendo este y otros servicios al turista.

Dentro de este contexto, se desarrolla una aplicación web llamada *WhatHostel*; que permita al usuario de dicha aplicación buscar, filtrar y reservar en *hostels*, tanto camas en habitaciones compartidas como habitaciones privadas.

Abstract

The tourism rise and popularization has ease the appearance of new accommodation models, so as the touristic use home, and of new platforms that enters the market offering this and other services.

In this context, a web application called WhatHostel is going to be devoloped, that allows the app's user to search, filter and book hostels, from beds in shared rooms to private dorms.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 OBJETIVOS

A continuación, se exponen los objetivos y alcance de este trabajo. Se indica cual es el fin último del proyecto, y se detallan los objetivos marcados para su consecución. Tras esto, se especifica el alcance marcado para este trabajo, concretando los resultados del mismo.

Se busca crear un portal de reservas enfocado en *hostels*; el cual sea intuitivo para el usuario, ofrezca una serie de funcionalidades útiles para el mismo, y ofrezca una navegación rápida.

Los objetivos técnicos vienen dados por la plataforma a la que están destinados:

- Aplicación web.

  Destinada al uso a través de navegadores. Esta aplicación se compone a su vez de dos aplicaciones íntimamente ligadas:

  - Portal de reservas
  - Intranet para la gestión de las propiedades.

- Aplicación móvil.

  Destinada al uso a través de un dispositivo móvil. Puede ser descargada e instalada por el usuario en su terminal.

1.1 Alcance

Este trabajo abarca la aplicación web; en concreto, desarrollar el portal de reservas. Para la creación del portal de reservas, denominado *WhatHostel*, se implementan los siguientes componentes:

- Servidor.

  El servidor de la aplicación, encargado de ofrecer servicios web al cliente y facilitarle recursos, además de conectarse y operar con la base de datos empleada.

- Cliente.

  Parte de la aplicación descargada por el usuario en su navegador. Hace uso del servidor para la obtención de información y recursos, y para realizar acciones sobre esta información.

- Base de datos.

  Contiene toda la información manejada por la aplicación, permitiendo su obtención y modificación por parte del servidor.

Además de estas tres partes principales, se hace uso de una plataforma como servicio para el despliegue de la aplicación. Esto consiste en un servicio de alojamiento y computación en la nube que externaliza la configuración del entorno del servidor.

2 INTRODUCCIÓN

La globalización y la reducción de precios en el transporte ha favorecido la creciente demanda turística en todo el mundo. Esta creciente demanda ha sido acompañada por un crecimiento de la oferta, los alojamientos dejan de ser ofrecidos únicamente por cadenas hoteleras y pequeños hostales para ser ofrecidos también por propietarios de viviendas. El precio se convierte en una parte clave de la competitividad y los albergues cobran un papel importante, ofreciendo camas individuales dentro de habitaciones compartidas.

Este modelo de alojamiento permite reducir los precios; además, el incentivo de tratar con otras personas y conocer gente de diferentes países ha propiciado también su popularización.

Dentro de este mercado fructífero tiene también su sitio la tecnología. Desde antes del 2000 han ido apareciendo nuevos portales web para ofrecer alojamiento, actuando de intermediarios entre el turista y el anfitrión. Estos portales han ganado poco a poco el terreno a las empresas físicas de turismo, dada su mayor capacidad de adaptación a los nuevos modelos; así como la expansión del comercio electrónico en general.

2.1 Motivación

Los portales de reservas existentes pertenecen a grandes empresas, enfocadas más en aspectos económicos que técnicos. Los cambios en la tecnología y estándares de diseño están siendo adoptados con lentitud por estas compañías, ofreciendo una oportunidad a nuevos portales para impresionar y atraer a los usuarios.

Una interfaz intuitiva, una navegación rápida, un fácil proceso de reserva; características que se pueden conseguir empleando las técnicas adecuadas y que ofrecen a *WhatHostel* una oportunidad.

2.2 Portal de reservas

Un portal de reservas se caracteriza por tres actores principales: el usuario, la aplicación web y la propiedad.

La aplicación web, o cualquier plataforma que permita la interacción entre el usuario y la propiedad, es el medio que emplea la empresa intermediaria para ofertar las habitaciones y camas de la propiedad al usuario.

La propiedad es dada de alta en la plataforma e introduce toda la información necesaria. La plataforma almacena esta información de forma persistente, por ejemplo, en una base de datos. Los precios y la disponibilidad son gestionados por la propiedad, quien indica un precio, del cual va a recibir el porcentaje acordado en el contrato de registro.

El usuario accede a la aplicación desde la herramienta correspondiente; por un navegador en el caso de una aplicación web, o descargando una aplicación móvil desde el catálogo del dispositivo. A través de esta aplicación, el usuario puede realizar búsquedas indicando detalles como el lugar de destino, la fecha de llegada y de salida o el número de huéspedes; además, se pueden aplicar filtros sobre los servicios ofrecidos por la propiedad, localización, etc.

Los resultados que obtiene el usuario a partir de esta búsqueda son los que la aplicación proporciona a partir de los datos almacenados de las propiedades. Se muestran imágenes de la propiedad, horarios, dirección y el precio, entre otras cosas. El precio mostrado no tiene, ni suele, corresponder con el indicado por la propiedad. Éste viene dado por las correspondientes modificaciones que le aplique el propio portal, rebajando o subiendo el mismo.

Son varias las posibles formas mediante las cuales el portal y la propiedad reciben su parte correspondiente del precio mostrado. Una vez el usuario realice la reserva, este puede haber pagado una parte del precio en el momento de la reserva. Dicha parte pertenece al portal, es la comisión que cobra por el servicio ofrecido; el resto puede pagarse también en ese momento y lo recibe la propiedad, o ser pagado en el momento de llegada al alojamiento.

Otra forma sería no pagar nada al realizar la reserva, pagando el precio íntegro a la llegada. En este caso la propiedad debe abonar a la empresa intermediaria la cantidad correspondiente en unos plazos acordados, normalmente, mensual.

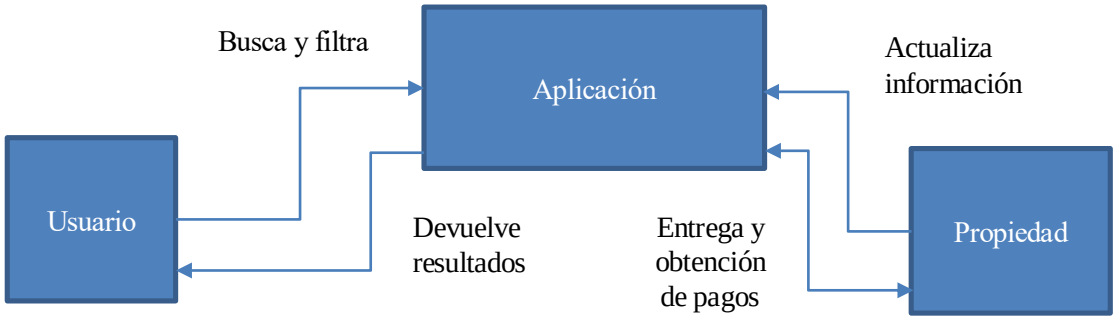


Figura 2-1. Estructura de un portal de reservas

3 ESTADO DEL ARTE

El proyecto pretende entrar en un sector ya poblado, donde existen actualmente portales web y aplicaciones que facilitan servicios similares. Por tanto, el primer paso es analizar esta población, estudiar su modelo de negocio y las tecnologías que emplean, así como diferenciar a los distintos competidores en función de su posición en el mercado.

3.1 Modelo de negocio

Las Agencias de Viajes Online (*OTAs* de su acrónimo en inglés) presentan tres modelos de negocio principales:

- Modelo de negocio por publicidad. *TripAdvisor*.

  En este modelo la empresa ofrece información o publicidad sobre una serie hoteles y otras propiedades turísticas en función de las preferencias marcadas por el usuario. Un ejemplo negocio que emplea este modelo como fuente de ingresos principal es *TripAdvisor*.

  TripAdvisor muestra a sus usuarios los resultados de su búsqueda; en el caso de hoteles, muestra su información y valoraciones del mismo. Además, ofrece enlaces disponibles a otras páginas donde poder realizar la reserva.

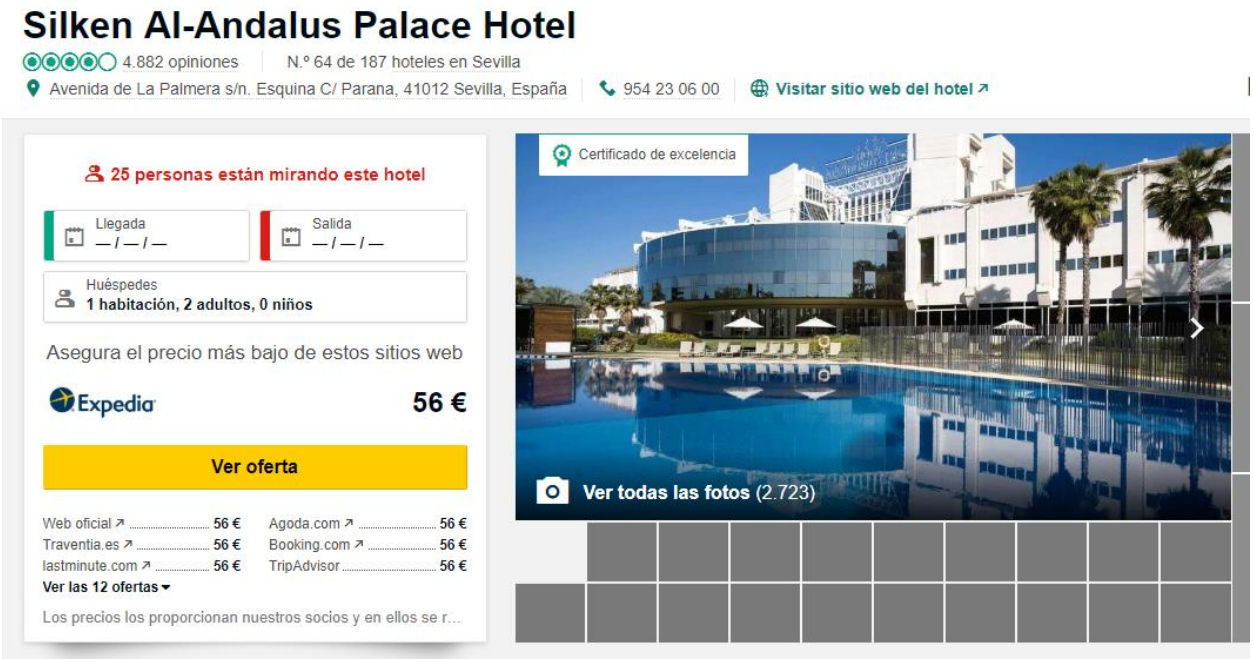


Figura 3-1. Interfaz de TripAdvisor

  La empresa recibe una compensación económica por cada *click* que un usuario haga a un enlace, por parte de la agencia propietaria de dicho enlace. Además, puede obtener una compensación mayor en función de lo lejos que el cliente llegue en el proceso de reserva.

- Modelo de negocio mercantil. *Expedia*.

  En el caso de *Expedia*, sus ingresos provienen principalmente de las reservas que se realizan a través de su portal. Su modelo de acción original se basa en adquirir paquetes de habitaciones y vuelos a hoteles

y compañías aéreas de todo el mundo, revendiendo dichos paquetes al usuario a través de su página. Además, ha adoptado el tercer modelo, el de agencia, debido a su mayor popularidad en Europa, Medio Oriente y Asia.

Figura 3-2. Interfaz de Expedia

- Modelo de negocio como agencia. *Booking.com*.

  El tercer modelo, representado por la agencia online perteneciente al grupo *Booking Holdings*, *Booking.com*; está basado también en la realización de reservas desde la propia página o aplicación. La diferencia respecto al modelo anterior es la no posesión de los productos que se venden. En el caso de *Booking.com*, no posee las habitaciones que ofrece, si no que cobra una comisión por la transacción.

  La ventaja de este modelo es que no existe necesidad de preocuparse por la gestión del inventario, se basa en contratos con los dueños de este inventario (los hoteles) con comisiones que varían desde el 10% al 30%, en función de la posición que estos desean tener en el portal de reservas y de su tamaño.

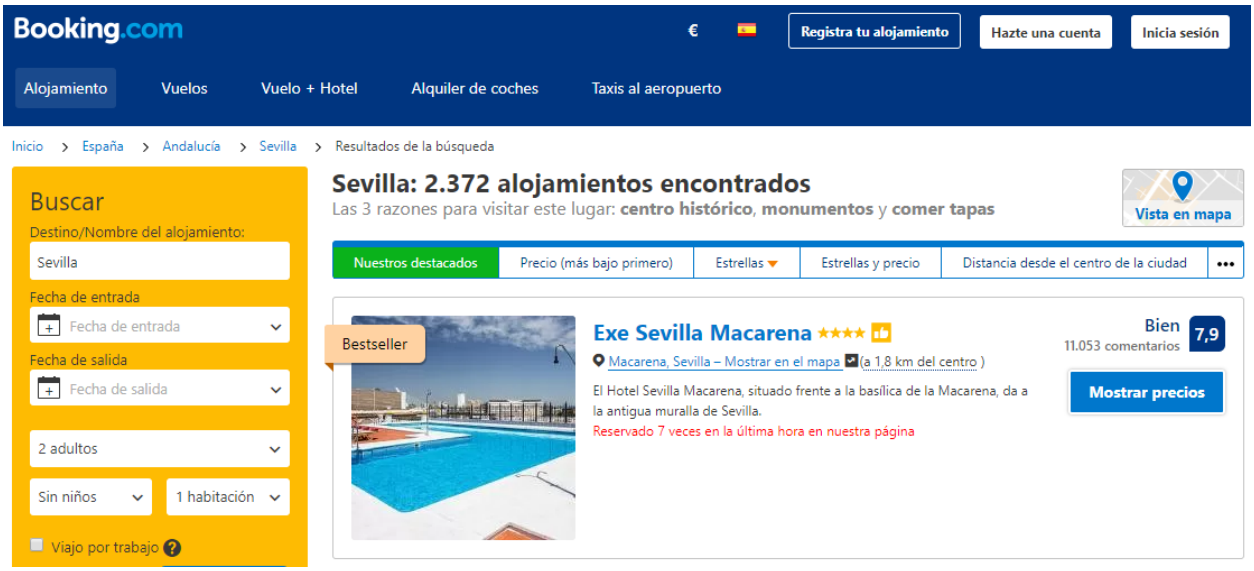


Figura 3-3. Interfaz de Booking.com

Este es el modelo a adoptar por la aplicación web, *WhatHostel*; exige una menor necesidad de capital inicial, ya que no se necesita disponer de inventario, y se basa principalmente en crear una interfaz práctica y que genere confianza en el usuario, así como entablar relaciones contractuales con los diferentes *hostels* que van a formar parte del catálogo.

3.2 Plataformas y tecnología

Las OTAs ofrecen sus servicios a través de diferentes plataformas, desde web hasta móvil. Para ello, deben seleccionar y emplear la tecnología adecuada para cada plataforma, invirtiendo los recursos necesarios en función de la rentabilidad que otorgue cada una. Atendiendo a la competencia basada en el tercer modelo de negocio explicado, el de agencia; se analiza su presencia en ambas plataformas mencionadas.

- Web.

  Todas las seleccionadas para el estudio, mostradas en el siguiente apartado según su posición, disponen de una aplicación web. Desde la aplicación web, adaptada también para ser visualizada desde el navegador del teléfono móvil, permiten acceder a los servicios ofrecidos.

  Se utilizan herramientas de programación y diseño web como HTML, CSS y JavaScript para el cliente; y herramientas para la programación y configuración de servidores y servicios web como Java, .NET, Python o JavaScript. Estas herramientas pueden ser utilizadas a través de *frameworks*, estudiados en la siguiente sección. También utilizan servidores, que pueden tanto estar gestionados por ellos como ser externalizados y gestionados en la nube; también se barajan las distintas opciones en la siguiente sección.

- Móvil.

  Aunque el uso de una web permite el conocimiento y uso de sus servicios, el hecho de disponer de una aplicación nativa instalada en el terminal del usuario favorece que este realice reservas posteriores. De las *OTAs* consideradas, únicamente la primera (*Hostelsclub.com*) no dispone de aplicación móvil.

  El desarrollo de una aplicación móvil, o dos si se quiere hacer nativa para iOS y Android, supone un gran coste y tiempo de trabajo.

3.3 Posición en el mercado

Para entender la progresión de los principales competidores, se han seleccionado cuatro portales de reserva para su estudio. De este estudio se ha extraído información sobre el total de visitas, el porcentaje de rebotes, ingresos y su correspondiente posición en el mercado. A continuación, se muestra en una tabla la información recogida:

Portal de reservas	Media visitas seis últimos meses	Porcentaje de rebote	Ingresos anuales estimados ($)	Posición en su categoría
Hostelsclub.com	305’74 mil	43’62%	< 1 millón	668
Hostelbookers.com	493’37 mil	49’61%	3’8 millones	416
Hostelworld.com	7’92 millones	38’94%	82’1 millones*1	22
Booking.com	428’17 millones	34’29%	3998 millones	1

Tabla 3–1. Principales competidores en orden ascendente

4 SELECCIÓN DE TECNOLOGÍAS

Una vez estudiado el estado del arte actual, se deben seleccionar las técnicas más adecuadas para el desarrollo del proyecto. Esto consiste en analizar las alternativas que ofrece la tecnología actual, para los diferentes componentes del sistema, con el fin de encontrar y utilizar aquella que permita resolver, con la mayor eficiencia, el problema.

Se parte de una explicación sobre la estructura y el funcionamiento de una aplicación web; a partir de dicha estructura, se contemplan las posibilidades de implementación de cada componente.

4.1 Aplicación web. Estructura y funcionamiento

Se describe la estructura y funcionamiento de una aplicación, actor principal de un portal de reservas. En concreto, dado el alcance del proyecto, se trata el caso de una aplicación web, indicando los distintos componentes y las interacciones entre ellos.

- Servicios web y servidor.

  El servidor es el encargado de realizar las tareas lógicas relevantes de la aplicación; autorización de recursos, autenticación, reservas, obtención y actualización de información de la base de datos.

  Para ello, el servidor ofrece los servicios web a través de una API REST o una API SOAP.

  Los servicios web se basan en una comunicación cliente-servidor a través de la web usando HTTP/S. HTTP es un protocolo de aplicación de comunicación sin estado que permite las transferencias de información en la web. HTTPS es una modificación de HTTP, apoyada en protocolos de seguridad (SSL/TLS) para encriptar la comunicación.

  REST es una representación de cambio de estado de los recursos basado en que Internet es una máquina de estados virtual donde el usuario avanza a través de una aplicación, atravesando estados, resultando en una página siguiente o nuevo estado de la aplicación. Este estilo de arquitectura de software presenta las siguientes restricciones:

  - Cliente-Servidor: separación en dos componentes independientes.
  - Sin estado: toda la información necesaria para comprender la petición está presente en cada mensaje HTTP. El cliente puede mantener una sesión a través de *cookies* o *tokens*.
  - Orientación a recursos: utiliza un identificador para acceder a los recursos, URI. Asocia los comandos HTTP con las funciones CRUD: POST-Crear, GET-Leer, PUT-Actualizar, DELETE-Borrar.

  SOAP es un protocolo estándar de transporte basado en XML que permite el intercambio de mensajes a través de Internet, mediante el protocolo HTTP. Está orientado a métodos, en lugar de a recursos como REST. Una petición SOAP tiene 3 partes principales:

  - Envoltorio: marco que define el contenido del mensaje y su procesamiento.
  - Reglas de codificación: indican como expresar las instancias de los tipos de datos definidos en la aplicación.
  - Convención: usado para representar las llamadas a procedimientos remotos y sus respuestas.

  El servidor como aplicación software, debe estar alojado en una máquina con conexión a Internet para poder recibir y procesar las peticiones de los clientes. Entre las opciones disponibles, podemos desplegarlo en una maquina propia, que debe ser configurada para el correcto funcionamiento del servidor; o emplear los servicios de *cloud computing* que ofrecen algunas empresas. Estos servicios de *cloud computing* o computación en la nube ofrecen alternativas de alojamiento en plataformas e

infraestructuras, dependiendo del tipo de servicio (PaaS, IaaS), en máquinas externas ya configuradas.

- Cliente.

  El cliente es la parte de la aplicación que se ejecuta en la máquina del usuario, en concreto, en un navegador web. El cliente se encarga de utilizar la API que ofrece el servidor para obtener información y mostrársela al usuario. Además, el cliente atiende las interacciones del usuario con la página y realiza acciones en consecuencia, tales como enviar formularios al servidor para obtener un resultado o mostrar nueva información en la página.

- Base de datos.

  En la mayoría de aplicaciones web es necesario almacenar información de manera persistente. Una de las opciones más utilizadas es el uso de una base de datos. Los sistemas gestores de bases de datos son herramientas que permiten la creación y gestión de una base de datos. Existen varios modelos de bases de datos dependiendo de su función y administración de los datos.

  - Jerárquicas: los datos se organizan en forma de árbol invertido, el nodo raíz es el nodo padre y puede tener hijos, un hijo sin hijos se conoce como hoja.
  - De red: se diferencia del anterior en que un mismo nodo puede tener varios nodos padres.
  - Orientada a objetos: se organiza en torno a objetos enfocándose en la integración con lenguajes de programación orientados a objetos como Java, C#, C++, etc.
  - Relacionales: basada en relaciones concebidas como tablas compuestas por registros, que representan las tuplas y campos (información) de la base de datos. El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL.
  - Documentales: almacenan los datos en forma de documentos. Cada documento representa información y su relación con otros elementos de datos, en forma de clave-valor.
  - Gráfica: utiliza la teoría de grafos para almacenar, mapear y consultar relaciones. Se basa en colecciones de nodos y bordes interconectados.

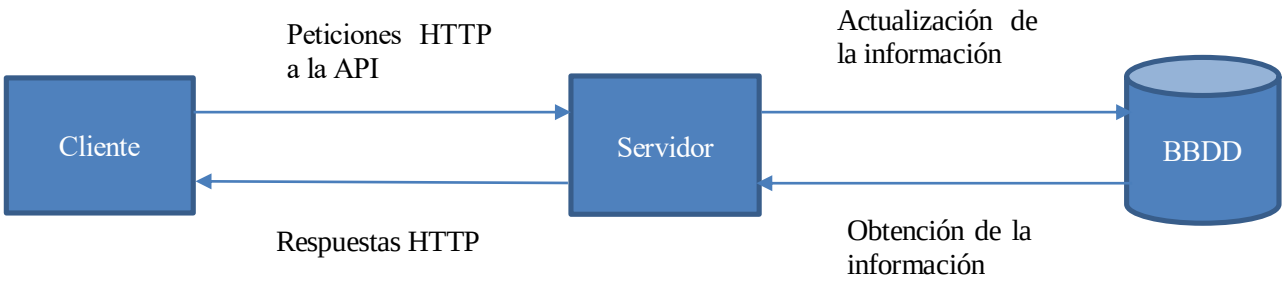


Figura 4-1. Diagrama de la estructura de una aplicación web

4.2 Servicios web y servidor

En primer lugar, se selecciona el tipo de servicio web o API a utilizar: SOAP o REST. A continuación, se barajan las alternativas para el servidor, tratando los lenguajes de programación disponibles y, en el caso correspondiente, el servidor web o entorno de ejecución complementario.

Tipo API	Formato Mensaje	Enfoque	Caché peticiones	Rendimiento

SOAP	XML	Basado en funciones	No	Más ancho de banda y recursos de computación
REST	Texto plano, JSON, HTML, XML, YAML, etc	Basado en datos	Sí	Menos recursos

Tabla 4–1. Comparativa tipos APIs para servicios web

Atendiendo a las necesidades del sistema; se requiere una gran escalabilidad y flexibilidad, la prioridad es la obtención de datos con una alta velocidad de respuesta. Por otro lado, se tiene una base de conocimientos personal previa mayor de JSON y HTML que de XML. Dado que REST permite que las peticiones se almacenen en caché y que SOAP consume un mayor ancho de banda; el primero resulta más favorable respecto a la escalabilidad y velocidad de respuesta. Además, permite el uso de JSON y HTML. Se elige REST como API para los servicios web de la aplicación.

En cuanto a infraestructura necesaria para almacenar la parte del servidor de la aplicación web, para la atención de peticiones y su procesamiento; se comparan las distintas alternativas prestando especial atención a los costes, tiempo de desarrollo y escalabilidad.

Según los costes, se pueden diferenciar dos tipos: CAPEX y OPEX:

- Inversiones en bienes de capital (CAPEX). Inversión para la compra de un activo del que se espera que otorgue beneficios a lo largo de su vida útil.
- Coste de funcionamiento (OPEX). Gasto recurrente o continuo, asociado generalmente a los costes de mantenimiento.

A continuación, se exponen las características de las dos alternativas consideradas: servidor propio y *cloud*.

Infraestructura	Costes CAPEX	Costes OPEX	Tiempo	Escalabilidad
Servidor propio	Servidores físicos Router, switches, cables, etc. Espacio físico ocupado Seguridad, extinción	Electricidad Conectividad Mantenimiento *hardware* Monitorización	Configuración de un entorno capaz de ejecutar la aplicación web, así como de tener una red segura y eficiente.	Se deben ampliar y configurar los nuevos recursos necesarios para atender la demanda
Cloud Computing	Ninguno	Recursos consumidos Servicios opcionales	Aprendizaje de la plataforma que ofrece el servicio *cloud*	Automático (asumiendo un posible aumento de los costes)

Tabla 4–2. Comparativa tipos APIs para servicios web

Debido a los recursos disponibles, la segunda opción permite un desarrollo más rápido y más económico en las primeras etapas del proyecto, aunque se debe tener presente la pérdida de privacidad que puede llegar a constituir el alojar la aplicación en servidores controlados por terceros.

Una vez seleccionado el tipo de infraestructura, se contemplan las plataformas que ofrecen los servicios

requeridos. En esta ocasión, Amazon Web Services y Heroku son las dos alternativas valoradas. El segundo permite el uso de un certificado SSL gratuito para hacer seguro el tráfico de la web, empleando HTTPS.

Finalmente, las tecnologías que se emplean se resumen en:

- Infraestructura *cloud* proporcionada por Heroku.
- Servicio web REST que ofrezca las funcionalidades de la aplicación.

4.3 Base de datos

Las dos alternativas a considerar para la creación de la base de datos son MySQL y MongoDB.

MySQL es un sistema de gestión de bases de datos relacional que permite el almacenamiento de datos estructurados. Ofrece una gran consistencia y la posibilidad de realizar transacciones. Esto lo hace menos flexible en un sistema con esquemas de datos no necesariamente estructurados y que pueden cambiar con frecuencia.

MongoDB es una base de datos no relacional documental, con una estructura de datos similar a JSON (BSON), lo cual permite una integración de los datos más rápida que en los modelos relacionales, pero pierde en cuanto a propiedades ACID. La alta velocidad de lectura y su flexibilidad la hacen adecuada para esta aplicación.

En la siguiente tabla se muestran las características de ambos:

Característica	MongoDB	MySQL
Cloud, SaaS, Web	Sí	Sí
Desarrolladores	MongoDB Inc.	Oracle Corporation
Esquema	Flexible	Rígido
Sistema Operativo	Multiplataforma	Multiplataforma
Almacenamiento de datos	JSON/BSON	Columnas y filas
Lenguaje de consulta	Sistema de ficheros de memoria volátil	SQL
MapReduce	Sí	No
Unicode	Sí	Sí
Respaldo	Sí	Sí
Creación/Desarrollo	Sí	No
Conversión de la base de datos	Sí	No
Monitorización	Sí	Sí
Análisis de rendimiento	Sí	No

Consultas	Sí	No
Interfaz relacional	Sí	No
Virtualización	Sí	No
Modelo de integridad	BASE	ACID
Atomicidad	Condicional	Sí
Consistencia	Sí	Sí
Aislamiento	No	Sí
Durabilidad del almacenamiento de datos	Sí	No
Transacciones	No	Sí
Integridad referencial	No	Sí
Índices secundarios	Sí	Sí
Claves compuestas	Sí	Sí
Búsqueda de texto	Sí	Sí
Índices geoespaciales	Sí	No
Soporte de grafo	No	No
CAP	CP	CA
Escalabilidad horizontal	Sí	Condicional
Replicación	Sí	Sí
Migración de datos	Sí	Sí
Modo replicación	Maestro-Esclavo	Maestro-Maestro/Esclavo
Fragmentación	Sí	Sí

Tabla 4–3. Comparativa MongoDB y MySQL

4.4 Lenguajes de programación

Una vez seleccionado el tipo de API y la infraestructura que almacenará nuestra aplicación web, comparamos los lenguajes de programación más utilizados para el desarrollo del lado del servidor y del cliente.

4.4.1 Lenguaje de programación en el lado del servidor

Dado que Heroku proporciona un servicio ya configurado, conocido como *Platform as a Service (PaaS)*, según la tecnología empleada para el desarrollo del servidor, se comparan los lenguajes de programación del lado del servidor en función de su rendimiento y escalabilidad, además de la documentación existente y la comunidad.

Lenguaje de programación	Año de lanzamiento	Comunidad	Sistema o *stack* usual	Rendimiento y escalabilidad
PHP	1995	Preguntas en StackOverflow: 1.112.866 Estrellas en GitHub: 12.670 Contribuidores: 485	LAMP: Linux Apache MySQL PHP	Mayor velocidad computacional
NodeJS (JavaScipt)	2009	Preguntas en StackOverflow: 187.917 Estrellas en GitHub: 38.398 Contribuidores: 1488	MEAN: MongoDB ExpressJS Angular NodeJS	Mayor escalabilidad: capaz de atender más peticiones simultáneamente con menos recursos

Tabla 4–4. Comparativa tipos APIs para servicios web

La elección de un lenguaje de programación no es determinante, si no que depende de las preferencias del equipo de desarrollo y, sobre todo, de su cooperación con los *frameworks* y demás componentes del sistema.

PHP es un lenguaje ampliamente conocido y utilizado para el desarrollo de páginas web dinámicas; NodeJS es un entorno de ejecución para JavaScript construido con el motor V8 de Google. El primero proporciona la confianza de una extensa documentación y comunidad detrás, mientras que el segundo dispone cada vez de más desarrolladores que crean módulos adicionales que añaden nuevas funcionalidades; su interés reside en que trabaja de forma asíncrona mediante un único hilo de ejecución, con una entrada y salida de datos en una arquitectura orientada a eventos y basada en *callbacks* para atender las peticiones, lo que permite una alta escalabilidad.

Un factor determinante es el uso de JavaScript para la creación de páginas dinámicas en el lado del cliente, lo cual favorece el uso de NodeJS con la intención de emplear un único lenguaje de programación para el desarrollo de toda la aplicación web. Además, se integra con MongoDB y permite un fácil uso de JSON para el intercambio de datos. Por lo tanto, se selecciona NodeJS como tecnología para el desarrollo de la lógica del servidor y del servicio web.

Para la construcción de la API REST, se emplea un *framework* ligero para Node, denominado ExpressJS. Este *framework* abstrae las funciones básicas de Node para un uso más sencillo, además de facilitar la construcción de la API mediante *callbacks* o llamadas a función que se realizan cuando llega una petición a una ruta determinada.

4.4.2 Lenguaje de programación en el lado del cliente

La parte del cliente de la aplicación web, corresponde a las páginas web y recursos que el usuario obtiene a través del navegador. Se emplean los siguientes lenguajes para la creación de estas páginas:

- HTML.

  HyperText Markup Language es un lenguaje de etiquetas empleado para la estructuración de páginas web, permitiendo definir los distintos componentes de manera estática.

- CSS.

  Cascading Style Sheets es un lenguaje de diseño gráfico para establecer el diseño visual de un documento estructurado con un lenguaje de etiquetas como HTML. Permite aplicar estilos a las páginas web.

- JavaScript.

  JavaScript es un lenguaje de programación interpretado, original del estándar ECMAScript. Es débilmente tipado y orientado a objetos. Su uso principal es el desarrollo de páginas web dinámicas, insertando lógica en el lado del cliente.

Existen *frameworks* que cambian el paradigma de desarrollo del lado del cliente de una aplicación web, ofreciendo patrones de software con los que han sido diseñados. Uno de estos *frameworks* es Angular.

Angular 7 es un framework para aplicaciones web desarrollado en TypeScript, superconjunto de JavaScript, y mantenido por Google. Permite la creación de aplicaciones web de una sola página (*SPA*), las cuales cargan todo el contenido (HTML, JavaScript, CSS) una vez al principio y sólo se solicitan los recursos dinámicos, ofreciendo una navegación más fluida. Angular enfatiza en el patrón Modelo Vista Controlador (MVC), siendo sus partes diferenciadas:

- Componentes.

  Los componentes en Angular son elementos con un ciclo de vida definido: se crean, cambian y se destruyen. Cada fase tiene asociado un método, el cual se puede implementar de manera opcional.

Método correspondiente a una fase del ciclo de vida	Llamada
ngOnChanges()	Se llama cuando cualquier propiedad de entrada al componente cambia y antes de iniciar el componente
ngOnInit()	Se llama una vez, tras la inicialización del componente al activarse el primer ngOnChanges()
ngDoCheck()	Es llamado durante cada fase de detección de cambios no controlados por Angular, para responder ante estos cambios
ngAfterContentInit()	Se llama una única vez tras el primer ngDoCheck()
ngAfterContentChecked()	Se llama después del anterior, y tras cada secuencia de ngDoCheck()
ngAfterViewInit()	Es llamado una única vez tras el primer ngAfterContentchecked(), para responder después de que Angular inicialice la vista del componente y sus hijos

ngAfterViewChecked()	Llamado tras ngAfterViewInit() y tras cada secuencia de ngAfterContentChecked(), para responder a cada comprobación de la vista del componente y de sus hijos
ngOnDestroy()	Se llama justo antes de que Angular destruya el componente, para dar de baja observables y eventos

Tabla 4–5. Fases del ciclo de vida de un componente

Cada componente está asociado a un módulo que lo declara, y tiene a su vez asociados un fichero HTML que describe la vista y un fichero CSS que describe los estilos. El fichero TypeScript es el que define el componente, como una clase; en el que se importan las clases necesarias y se definen los atributos y métodos del componente. Estos atributos y métodos pueden ser accedidos desde el HTML asociado, participando en el flujo de la página. Una ruta suele corresponderse con un componente.

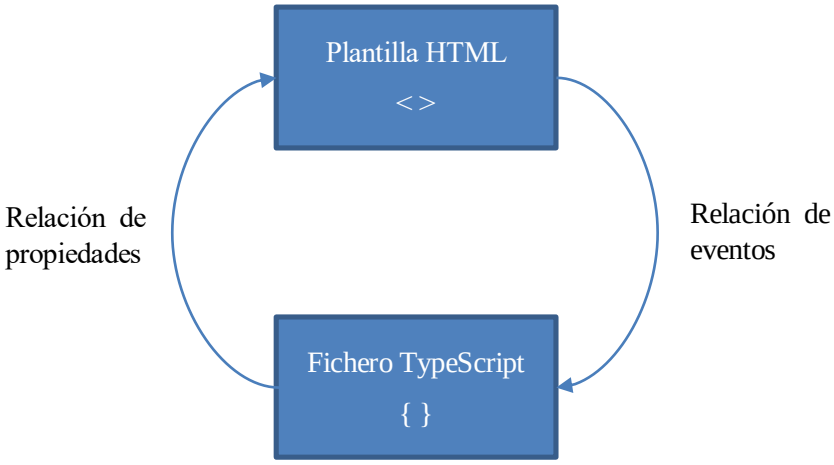
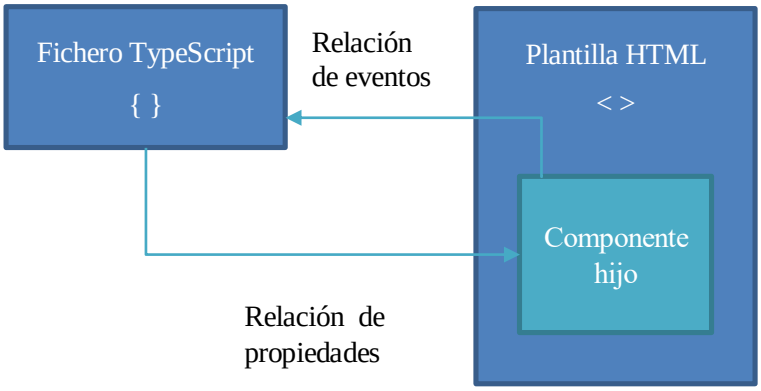


Figura 4-2. Relaciones de datos en componentes

Los componentes pueden a su vez, ser hijos de otros componentes. Esto sucede al incluir un componente en el HTML de otro, permitiendo el intercambio de datos mediante las relaciones de propiedades y de eventos



.

Figura 4-3. Componentes padre e hijo

- Servicios.

  A diferencia de los componentes, los servicios existen durante todo el funcionamiento de la página una vez hayan sido instanciados. Los servicios son instanciados e inyectados en los componentes que vayan a hacer uso de los mismos. Se emplean para tareas lógicas; como realizar una petición HTTP al servidor o facilitar la comunicación entre componentes.

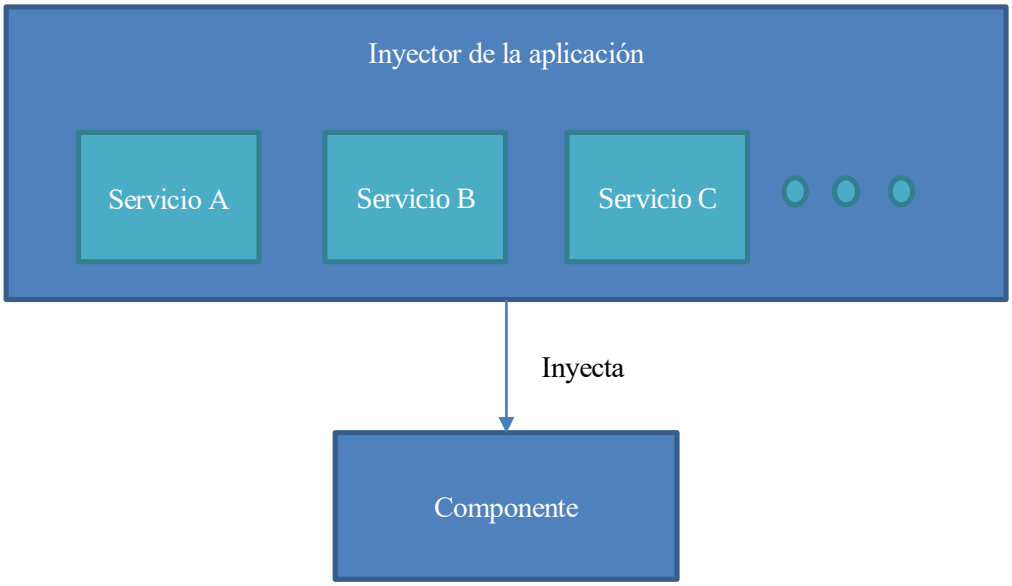


Figura 4-4. Importación de servicios en componentes

- *Guards*.

  Realizan acciones antes de navegar a otra ruta. Por ejemplo, impiden la navegación a rutas sin autorización.

- Filtros y tuberías (*Filters* & *Pipes*).

  Permiten filtrar o tratar la información, normalmente de manera asíncrona. Por ejemplo, ordenar listas o mostrar un formato de fecha específico

- Existen más elementos no tan fundamentales que permiten ampliar la funcionalidad de Angular.

De esta forma, se emplea NodeJS junto a ExpressJS, Angular 7 y MongoDB para el desarrollo de la aplicación web, la cual será desplegada en una instancia de la nube de Heroku.

5 DISEÑO E IMPLEMENTACIÓN

En esta sección, se muestra al lector los pasos seguidos para la consecución de los objetivos. Se desglosan estos objetivos en hitos y tareas, clasificadas según su fin, para el diseño e implementación de los servicios web y lógica que ofrece el servidor, de los esquemas y relaciones de los datos de la aplicación, y de los componentes y servicios que construyen la parte del cliente; por último, se detalla el proceso de despliegue de la aplicación en Heroku.

5.1 Funcionalidad del servidor

En primer lugar, se describe la API que debe emplear el cliente para acceder a los servicios. Seguidamente, se explica la lógica que se desarrolla en el servidor para satisfacer las necesidades planteadas por los servicios, así como módulos utilizados para determinadas acciones.

5.1.1 API

El servidor debe ofrecer una API con servicios que abarquen todas las funcionalidades de la aplicación. El cliente puede solicitar todas las funciones a través de peticiones HTTP (GET, POST, PUT, DELETE) a los siguientes *endpoints* o puntos finales de los servicios:

- Obtención de ciudades disponibles. *https://www.whathostel.com/api/cities (GET)*

  Este servicio permite que el cliente obtenga la lista de todas las ciudades presentes en los datos de la aplicación.

- Búsqueda de *hostels*. *https://www.whathostel.com/api/hostels (PUT)*

  Dado los identificadores de un conjunto de *hostels*, se obtiene toda la información que se posea de los mismos.

- Obtener opiniones de un *hostel*. *https://www.whathostel.com/api/reviews (GET)*

  Se obtienen todas las reseñas realizadas en un *hostel* concreto.

- Reserva de disponibilidad. *https://www.whathostel.com/api/new-checkout-session (POST)*

  Se realiza una reserva temporal de los artículos seleccionados por el usuario mientras este efectúa el pago. En caso de no realizar el pago, los artículos son liberados en un plazo de 10 minutos.

- Reserva de estancia y pago. *https://www.whathostel.com/api/checkout (POST)*

  Cuando el usuario realice el pago, se reservan definitivamente los artículos seleccionados y se procesa el pago correspondiente.

5.1.2 Lógica del servidor

Una vez atendidas las peticiones a cada ruta de la API, el servidor debe realizar una o varias acciones, devolviendo una respuesta al cliente. Para realizar estas acciones, puede ser necesario el uso de datos recibidos en la petición u obtenidos de la base de datos. Cada acción será definida en una función. Una petición a la API puede llamar a varias funciones, que van pasando la petición según un orden establecido.

- Obtención de ciudades de la base de datos. *getCities()*

  El servidor se comunica con la base de datos para que le proporcione toda la información disponible de las ciudades con las que trabaja la aplicación. Esta información es devuelta al cliente en la respuesta.

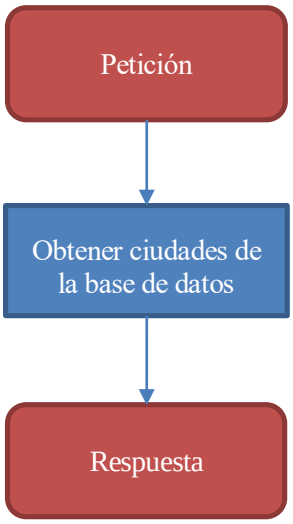


Figura 5-1. Diagrama de flujo de la obtención de ciudades

- Obtención de *hostels* de la base de datos. *getHostels()*

  Se utiliza la lista de identificadores contenida en la petición para obtener de la base de datos la información almacenada sobre los *hostels* indicados. Esta información es devuelta al cliente en la respuesta.

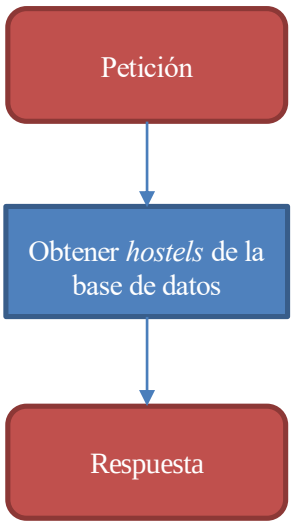


Figura 5-2. Diagrama de flujo de la obtención de *hostels*

- Obtención de opiniones de un *hostel*. *getReviews()*

  El servidor obtiene de la base de datos las opiniones realizadas por usuarios para un *hostel* indicado en la petición. Las opiniones son devueltas al cliente en la respuesta.

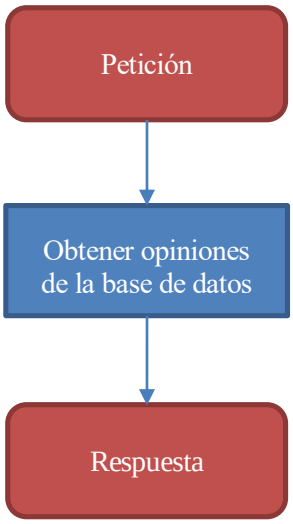


Figura 5-3. Diagrama de flujo de la obtención de opiniones

- Reserva temporal de disponibilidad. *newCheckoutSession()*

  Se disminuye la disponibilidad de la habitación indicada en la petición en función del número de huéspedes indicado también en la petición. Se emplea un temporizador para recuperar esa disminución al cabo de 10 minutos si no se ha realizado el pago. El servidor devuelve un *token* en la respuesta. Este token contiene la información de la reserva necesaria para que, en próximos accesos al servicio, se pueda comprobar el estado de la reserva, pudiendo realizar una nueva descartando la anterior en el caso de que cambiara su estado.

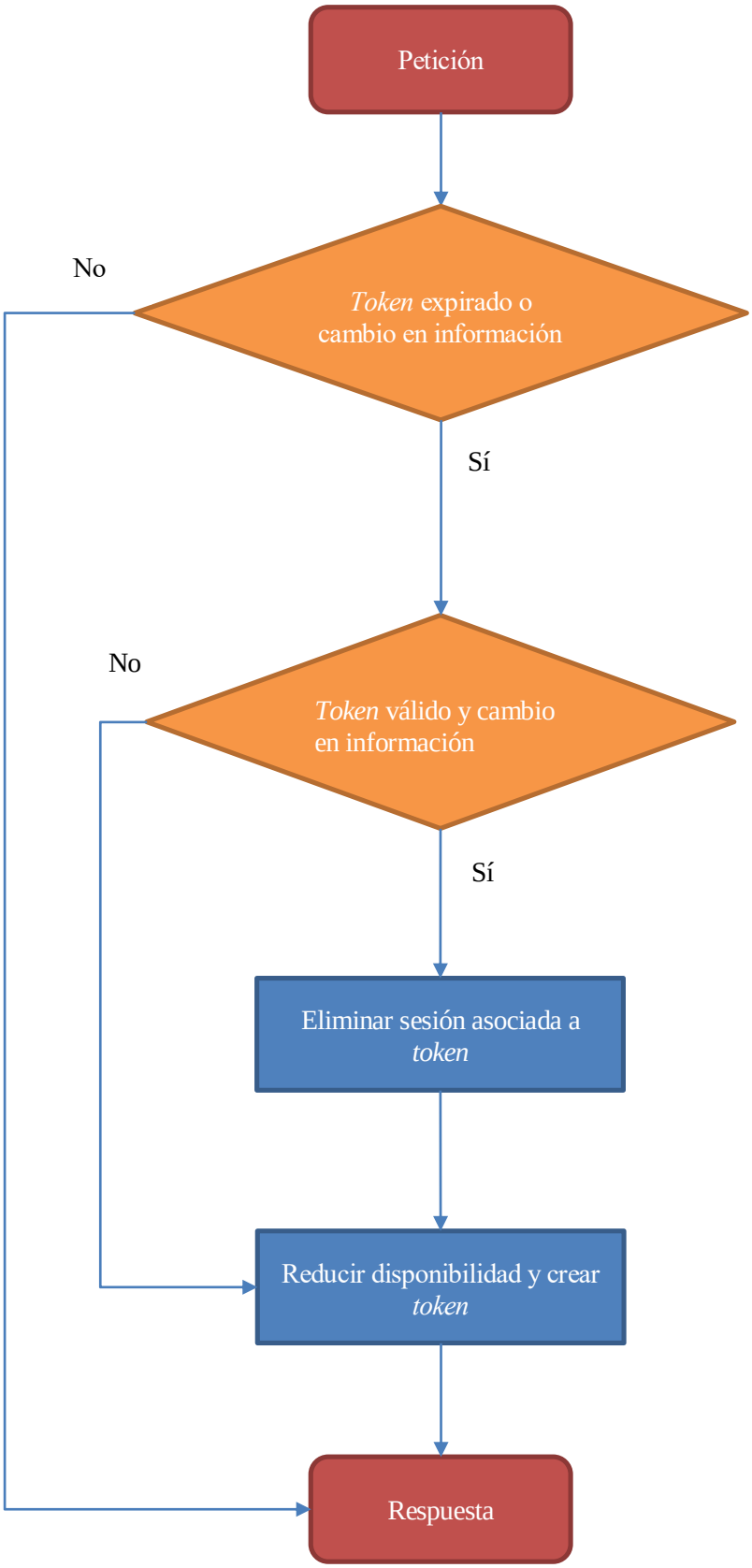


Figura 5-4. Diagrama de flujo de la reserva de disponibilidad

- Reserva definitiva. *checkCheckoutSession()*

  Se comprueba nuevamente la disponibilidad y el precio para evitar posibles fallos. Se anula la reserva temporal y se reserva nuevamente de manera definitiva. Se pasa la petición a la función del pago.

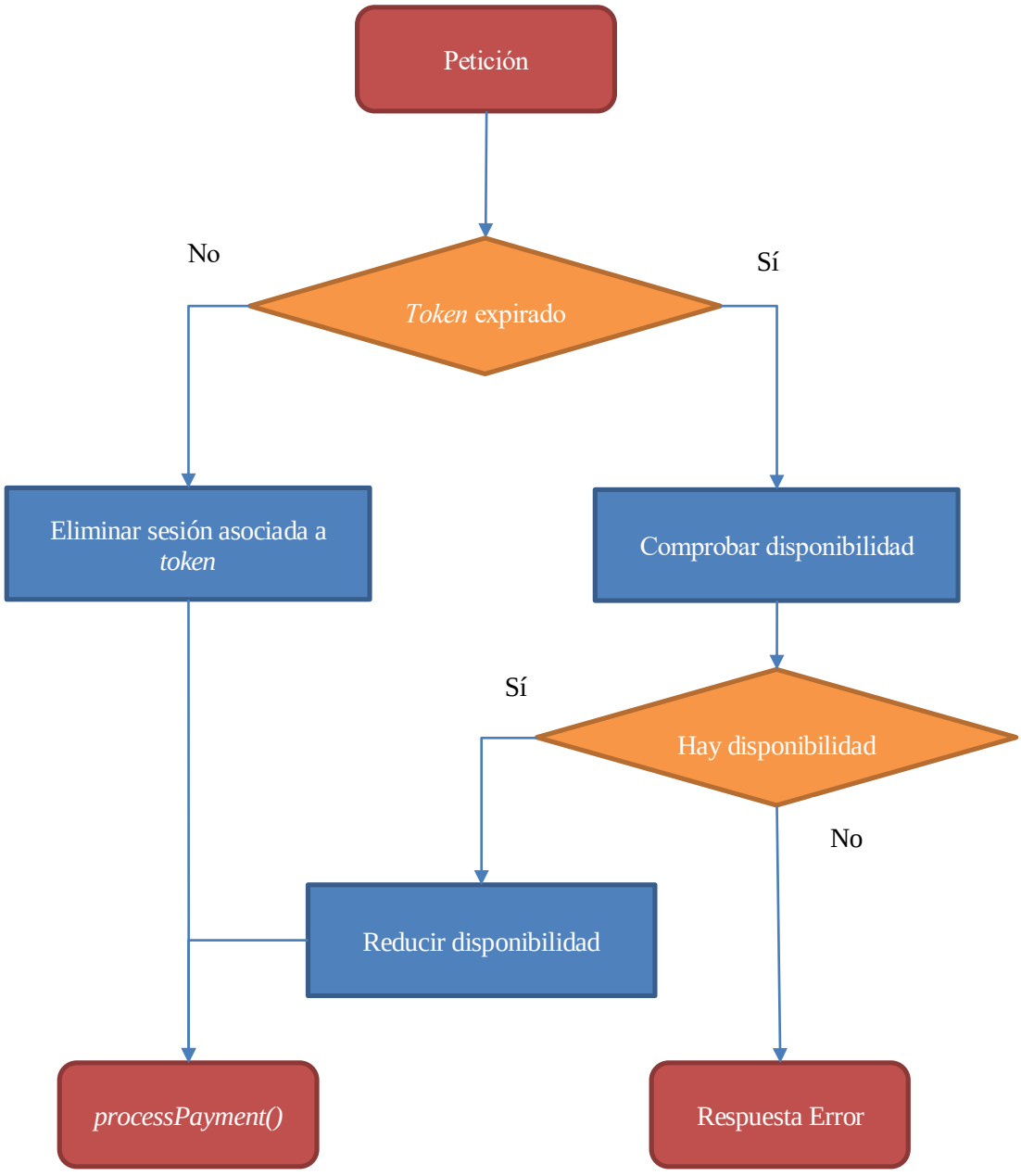


Figura 5-5. Diagrama de flujo de la reserva de disponibilidad

- Pago. *processPayment()*

  Se hace uso de un servicio de pago externo, el cual procesa la información de la tarjeta contenida en la petición para el cobro indicado. Se pasa la petición a la función de envió de email de confirmación en el caso de que se haya realizado con éxito; se devuelve error en la respuesta en caso contrario.

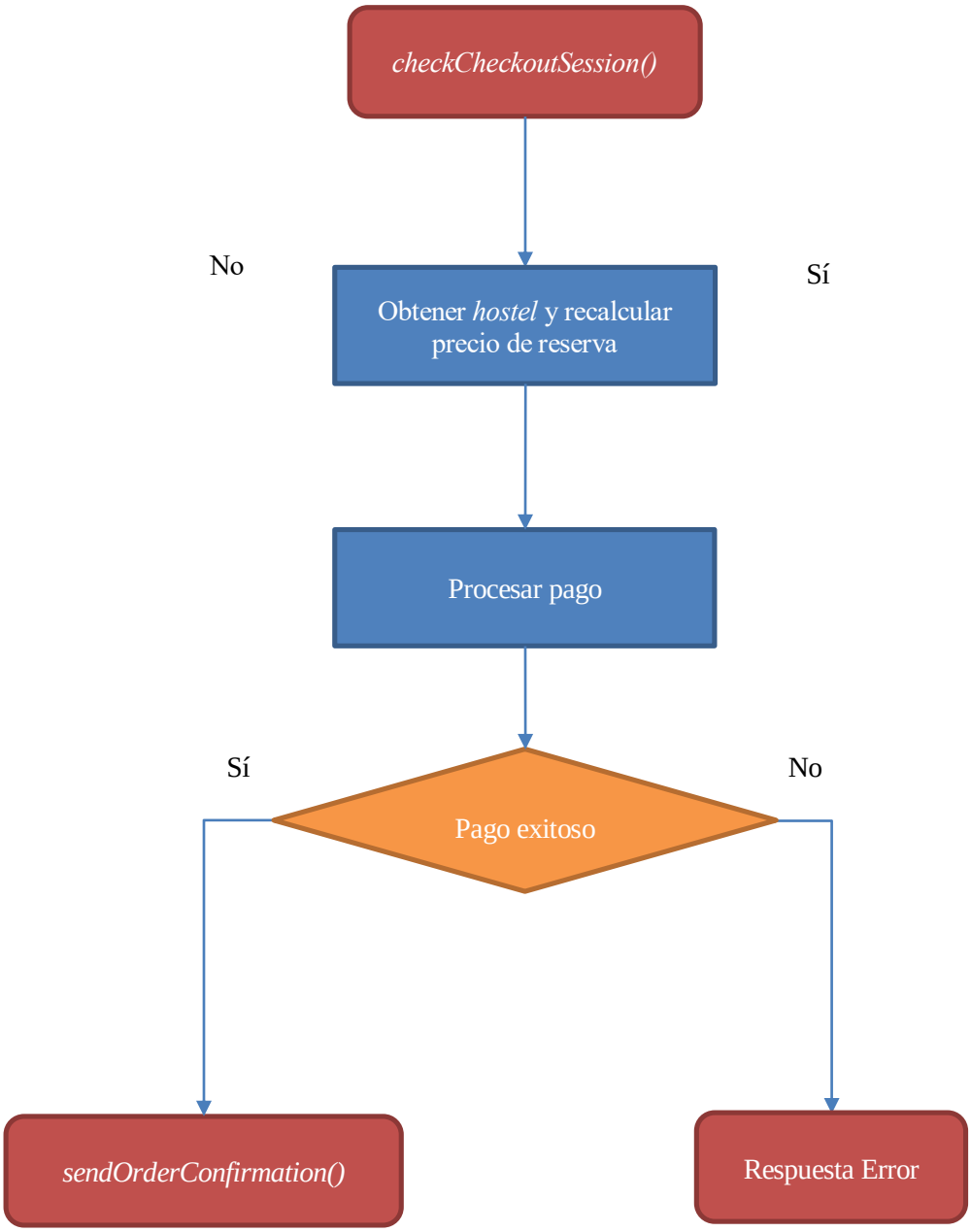


Figura 5-6. Diagrama de flujo del proceso de pago

- Envío de email de confirmación. *sendOrderConfirmation()*

  Se emplea un módulo denominado *sendinblue* para el envío de una plantilla de email generada dinámicamente a partir de los datos de la reserva y del cliente. Se devuelve éxito en la respuesta.

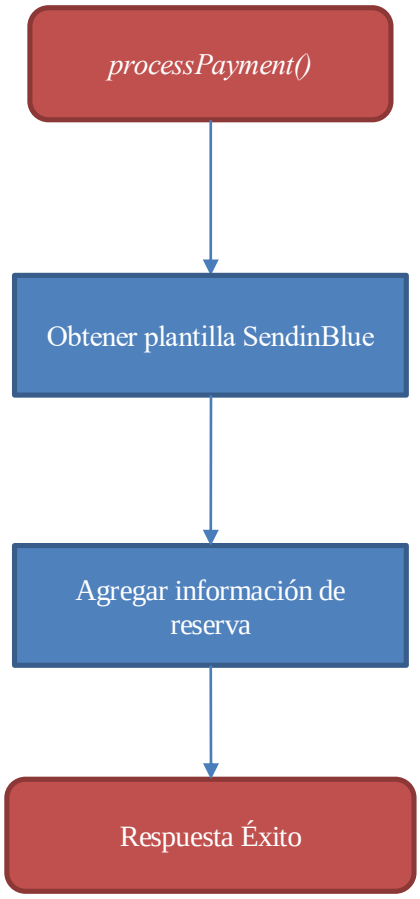


Figura 5-7. Diagrama de flujo del envío de correo electrónico de confirmación

5.2 Datos de la aplicación

Una vez analizados los servicios y funcionalidades de la aplicación, resulta sencillo identificar la clase de información que se emplea. Cada tipo de información se define en un esquema, este esquema representa sus atributos y las relaciones con otros componentes del modelo de datos.

5.2.1 Tipos de datos primitivos

Primero, se listan los tipos de datos primitivos soportados por MongoDB:

- Double: número con decimales
- Integer: número entero
- String: cadena de caracteres
- Object: objeto con varios atributos
- Array: tabla de elementos
- Object Id: identificador de MongoDB
- Date: fecha
- Boolean: verdadero o falso

Estos son algunos de los tipos de datos más usados permitidos en MongoDB. Para la conexión con la base de datos se usa un módulo de NodeJS llamado *mongoose*, que incluye los siguientes tipos de datos:

- String: corresponde al *String* de Mongo
- Number: corresponde al *Double* y al *Integer* de Mongo
- Date: corresponde al *Date* de Mongo
- Mixed: permite cualquier tipo (no se comprueba)
- Boolean: corresponde al *Boolean* de Mongo
- Array: corresponde al *Array* de Mongo y se representa con [*tipo*]. Siendo *tipo*, el tipo de los elementos de la tabla
- ObjectId: corresponde al *Object Id* de Mongo

5.2.2 Esquemas de datos

Los esquemas de información que componen el modelo de la aplicación son:

- City
    - _id (*ObjectId*): identificador asignado por la base de datos
    - name (*String*): nombre de la ciudad
    - country (*String*): nombre del país
    - latitude (*Number*): latitud geográfica de la ciudad
    - longitude (*Number*): longitud geográfica de la ciudad
    - images (*[String]*): rutas de imágenes de la ciudad
    - _hostels (*[ObjectId]*): identificadores de los *hostels* registrados en la ciudad

- Hostel
    - _id (*ObjectId*): identificador asignado por la base de datos
    - name (*String*): nombre del *hostel*
    - email (*String*): correo electrónico registrado
    - password (*String*): contraseña para acceder a la cuenta del *hostel*
    - salt (*String*): cadena empleada para el cifrado de la contraseña
    - id_hostel (*String*): identificador empresarial del *hostel*
    - inscription_date (*Date*): fecha de registro
    - consent (*Boolean*): consentimiento para completar el perfil del *hostel*
    - terms_agreement (*Boolean*): acuerdo de los términos de registro
    - city (*String*): nombre de la ciudad donde se encuentra
    - address (*String*): dirección postal de la propiedad
    - distance (*String*): distancia de la propiedad al centro de la ciudad
    - main_phone_number (*String*): número de teléfono principal de la propiedad
    - manager_phone_number (*String*): número de teléfono del encargado
    - manager_email (*String*): correo electrónico del encargado
    - brief_card_description (*String*): descripción breve de la propiedad
    - long_card_description (*String*): descripción extensa de la propiedad

- checkin_start (*String*): hora de inicio de llegada y entrada al *hostel*
- checkin_end (*String*): hora de fin de llegada y entrada al *hostel*
- checkout_start (*String*): hora de inicio de salida del *hostel*
- checkout_end (*String*): hora de fin de salida del *hostel*
- cancellation_policy (*String*): normas de cancelación del *hostel*
- _reviews (*[ObjectId]*): tabla con los identificadores de las reseñas realizadas sobre la propiedad
- rating (*Object*): puntuación desglosada, todos los tipos son *Number*
  - mean: media general
  - cleanliness: limpieza de la propiedad
  - location: localización de la propiedad
  - atmosphere: atmósfera o ambiente
  - facilities: servicios ofrecidos por la propiedad
  - staff: trato y trabajo de los trabajadores del *hostel*
  - valueformoney: calidad-precio de la estancia
- fee_list (*Object*): tarifas especiales de estancia
  - fee_name (*String*): nombre de la tarifa
  - fee_price (*Number*): precio ofertado
  - fee_cancellation (*String*): condiciones de cancelación
  - fee_breakfast (*Boolean*): incluye o no desayuno
  - fee_dinner (*Boolean*): incluye o no cena
- A continuación, se listan atributos que identifican características de la propiedad, según su nivel indicado por un número (tipo *Number*)
  - ambient: 0 – tranquilo, 1 – normal, 2 – fiesta
  - dinner: 0 – no hay cena, 1 – cena de pago, 2 – cena gratuita
  - breakfast: 0 – no hay desayuno, 1 – desayuno de pago, 2 – desayuno gratuito
  - tours: 0 – no hay guías turísticas, 1 – guías turísticas de pago, 2 – guías gratuitas
  - hair_dryer: 0 - no hay secador, 1 – secador de pago, 2 – secador gratuito
  - lockers: 0 – no hay taquilla, 1 – taquillas de pago, 2 – taquillas gratuitas
  - luggage_parking: 0 – no se guarda equipaje, 1 – se guarda equipaje, 2 – se guarda equipaje gratis
  - towel: 0 – no hay toallas, 1 – toallas de pago, 2 – toallas gratis
- Los siguientes atributos indican sí o no mediante un *Boolean*:
  - rooftop: hay terraza o patio
  - maps: hay mapas turísticos
  - laundry: hay servicio de lavandería
  - pets: permite mascotas
  - children: permite niños
  - coffee: hay cafetera
  - tea: hay tetera

        - bar: hay bar o restaurante
        - pool: hay piscina
        - snooker: hay billar
        - gym: hay gimnasio
        - games: hay consolas o juegos
    - bookings_list (*[Object]*): tabla de reservas realizadas en el *hostel*
        - id_booking (*String*): identificador de la reserva
        - customerName (*String*): nombre del cliente que ha realizado la reserva
        - people (*Number*): número de huéspedes
        - totalAmount (*Number*): precio total de la reserva
        - balanceDue (*Number*): cantidad que debe pagar el cliente a la llegada
        - rooms (*[Object]*): tabla de habitaciones reservadas
            - name (*String*): nombre de la habitación
            - price (*Number*): precio de la habitación
            - quantity (*Number*): cantidad de camas o habitaciones de este tipo reservadas
        - nights (*Number*): número de noches de la estancia
        - checkin (*Date*): fecha de llegada
        - checkout (*Date*): fecha de salida
        - reservationDate (*Date*): fecha de realización de la reserva
        - status (*Number*): 0 – llegada, 1 – salida, 2 – dentro
        - paymentInfo (*Object*): información de pago
            - nationality (*String*): nacionalidad del cliente
            - email (*String*): correo electrónico del cliente
            - phone (*String*): teléfono del cliente
            - titular_name (*String*): nombre del titular de la tarjeta
            - address (*String*): dirección de facturación
            - token (*Object*): elemento empleado para identificar el pago
        - _review (*ObjectId*): reseña asociada a la reserva
    - rooms_list (*[Object]*): tabla de habitaciones del *hostel*
        - name (*String*): nombre de la habitación
        - beds (*Number*): número total de camas
        - calendar_list (*[Object]*): calendario de disponibilidad de la habitación
            - date (*Date*): fecha del calendario
            - availability (*Number*): disponibilidad para esa fecha
            - price (*Number*): precio para esa fecha
            - occupied (*Number*): ocupación para esa fecha
        - Los siguientes atributos indican características de la habitación (*Boolean*)
            - ensuite: tiene baño privado

- aircon: hay aire acondicionado
- private_light: tiene lámpara personal
- private_plug: tiene enchufe personal
- wifi: hay Wi-Fi
- hair_iron: hay plancha para el pelo
- iron: hay plancha para la ropa
- linen: incluye la ropa de cama

- Review
  - highlight (*String*): título de la reseña
  - pros (*String*): cosas buenas de la propiedad
  - cons (*String*): cosas malas de la propiedad
  - hostelRes (*String*): respuesta del *hostel*
  - hostelName (*String*): nombre del *hostel* al que se hace referencia
  - checkoutDate (*Date*): fecha de salida de la estancia
  - reviewDate (*Date*): fecha de publicación de la reseña
  - customerName (*Stringt*): nombre del cliente que hizo la reserva
  - rating (*Object*): puntuación desglosada (*Number*)
    - mean: media
    - cleanliness: limpieza
    - location: localización
    - atmosphere: ambiente
    - facilities: servicios
    - staff: trabajadores
    - valueformoney: calidad-precio

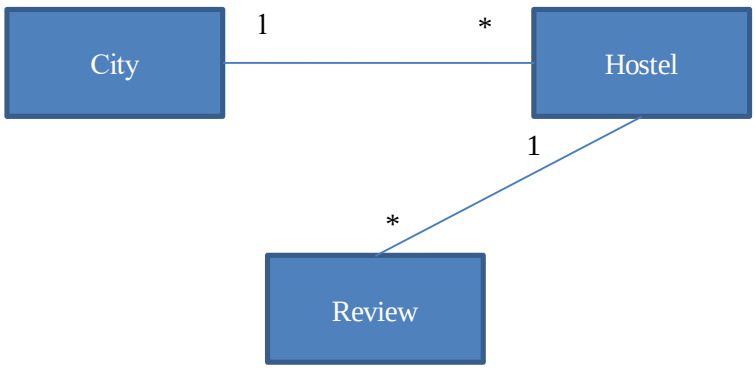


Figura 5-8. Relaciones entre esquemas de datos

5.3 Elementos del cliente

El lado del cliente está compuesto por una serie de componentes y servicios, cuya misión es ofrecer al usuario las funcionalidades de la aplicación, haciendo uso de la API desarrollada.

A continuación, se explica el flujo de la página web; se detallan los distintos componentes y servicios, indicando su función, la comunicación entre los mismos y las interacciones con el usuario.

5.3.1 Componentes

Como se ha dicho anteriormente, los componentes son los elementos fundamentales. Cada paso que se da en la página inicia una serie de componentes definidos y destruye los anteriores.

Los componentes que existen en la aplicación son:

- Home

  El componente *home* es el primero en mostrarse. En su HTML se detalla la vista y se importa otro componente, llamado *search-form*; este sería un componente hijo del componente *home*. Además, incluye dos botones para navegar a la página de registro e inicio de sesión de las propiedades, fuera del alcance del trabajo.

  En su CSS se da forma a los elementos y se adapta a escritorio y dispositivos móviles (*responsive*).

- Search-Form

  Este componente muestra un formulario. Antes de mostrar el formulario, el componente emplea un servicio llamado *city.service* para realizar una petición a la API y obtener la lista de ciudades disponibles.

  El primer campo del formulario, es un *input* o una entrada de texto para introducir la ciudad que se desea buscar. Se emplea una tarea asíncrona para filtrar la entrada del usuario y mostrar las coincidencias con la lista de ciudades.

  Los siguientes campos son para indicar la fecha de llegada y salida mediante un calendario. Finalmente, se muestra un *select* o elemento de selección para indicar el número de huéspedes, y un botón para realizar la búsqueda.

- Search

  Una vez pulsado el botón de búsqueda del *search-form*, se cambia el contenido de la página (es una aplicación de una sola página, por lo que solo se cambia el contenido, no se navega). El siguiente componente que se crea es el *search*, el cual obtiene un objeto con la información introducida en el formulario anterior y hace uso del servicio *search.service* para obtener mediante una llamada a la API la información de los *hostels* disponibles en el rango de fechas seleccionadas, para el número de huéspedes indicados, en la ciudad introducida.

  Con la información disponible, utiliza la API de Google para mostrar un mapa con los puntos de localización de las propiedades; y muestra tarjetas con la información resumida de estas propiedades.

  Las tarjetas se muestran mediante una lista dinámica, por cada elemento se accede a la ruta de la imagen, la cual esta almacenada en un servidor *cloud* llamado Cloudinary; y al resto de información disponible, como el precio mínimo o la puntuación. En los Resultados se mostrará en detalle la vista de los componentes.

  La lista de *hostels* puede ser filtrada haciendo uso de un *pipe* llamado *hostels.pipe*.

  El paso al siguiente componente se realiza al pulsar sobre alguno de los *hostels*.

- Hostel

  Este componente obtiene a través del servicio *hostel.service* la información de la propiedad seleccionada en el componente *search*, o del indicado en la URL.

  Aquí se muestran todos los datos del *hostel* disponibles. Se incluye una galería de imágenes, el desglose de la puntuación y las reseñas en una tarjeta. Además, se puede voltear dicha tarjeta para mostrar más

detalles como las horas de entrada y salida, la política de cancelación, una descripción larga e iconos que indican servicios de la propiedad.

En una tarjeta adyacente, se listan las habitaciones disponibles y se permite su selección y filtrado. Abajo, se puede ver el precio total de los elementos seleccionados, así como un botón para proceder a la reserva.

El pulsar este botón provoca el envío de una petición a la API para realizar la reserva de disponibilidad temporal, y se cambia al componente *payment*.

- Payment

  Este es el último componente de la página. En él, se presenta un formulario para introducir la información del usuario que realiza la reserva (nombre, teléfono, correo electrónico, etc) y la información de pago (número de tarjeta, nombre del titular, dirección de facturación, etc). A la derecha del formulario, se muestra un resumen de la reserva (noches, habitación, etc) y el precio total, la parte que se paga en el momento de reservar, y la que se paga a la llegada al *hostel*. Por último, se incluye el botón para efectuar la reserva.

  Cuando el usuario introduce los datos correctos y pulsa en el botón para realizar la reserva, se realiza otra petición a la API a través del servicio *payment.service*. Esta petición indica al servidor que realice la reserva y, si se diera el caso, el pago.

  Para el pago se utiliza una plataforma que permite hacer los cobros creando un token a partir de los datos de facturación, el cual identifica el pago.

5.3.2 Servicios

Ya se han mencionado los servicios utilizados, ahora se expone en detalle su funcionamiento.

- City Service

  Define métodos para realizar las peticiones a la API del servidor para obtener la lista de ciudades.

- Search Service

  Define métodos para realizar las peticiones a la API del servidor para obtener los *hostels* de una ciudad. Tiene un método que permite filtrar los *hostels* según su disponibilidad para las fechas y huéspedes seleccionados.

- Hostel Service

  Define métodos para realizar las peticiones a la API del servidor para obtener la información de un *hostel* concreto.

  Permite la comunicación entre el componente *search* y *hostel*; usa el patrón Observable para que el primero pase la información que ya posee sobre el *hostel* seleccionado al segundo, para que la muestre sin tener que realizar una petición HTTP.

- Payment Service

  Define métodos para realizar las peticiones a la API del servidor para hacer la reserva de disponibilidad temporal y para realizar la reserva definitiva.

  También hace uso del patrón Observable, para hacer posible que el componente *hostel* le pase al componente *payment* los datos de la reserva.

5.3.3 Otros elementos

Se ha señalado la existencia de otros elementos en una aplicación Angular. Uno de ellos ya se ha mencionado, y es el *hostel.pipe*; los otros dos empleados son módulos para las rutas y para elementos de terceras partes.

- Hostel Pipe

  Permite el filtrado de los *hostels* según el tipo de habitaciones disponibles y las características o servicios que ofrecen.

- Route Module

  Contiene todas las rutas de la aplicación Angular, cada una de ellas asociada a un componente. La etiqueta *router-outlet*, incluida en un componente raíz denominado *app.component*, indica que en esa posición el contenido dependerá de la ruta (componente). Se utiliza un objeto Router para la navegación de un componente a otro (de una ruta a otra).

- Material Module

  Material es un *framework* CSS de Google, que proporciona elementos con estilos aplicados siguiendo el patrón Material Design. Estos elementos son ampliamente utilizados en la página, aplicándoles estilos personalizados.

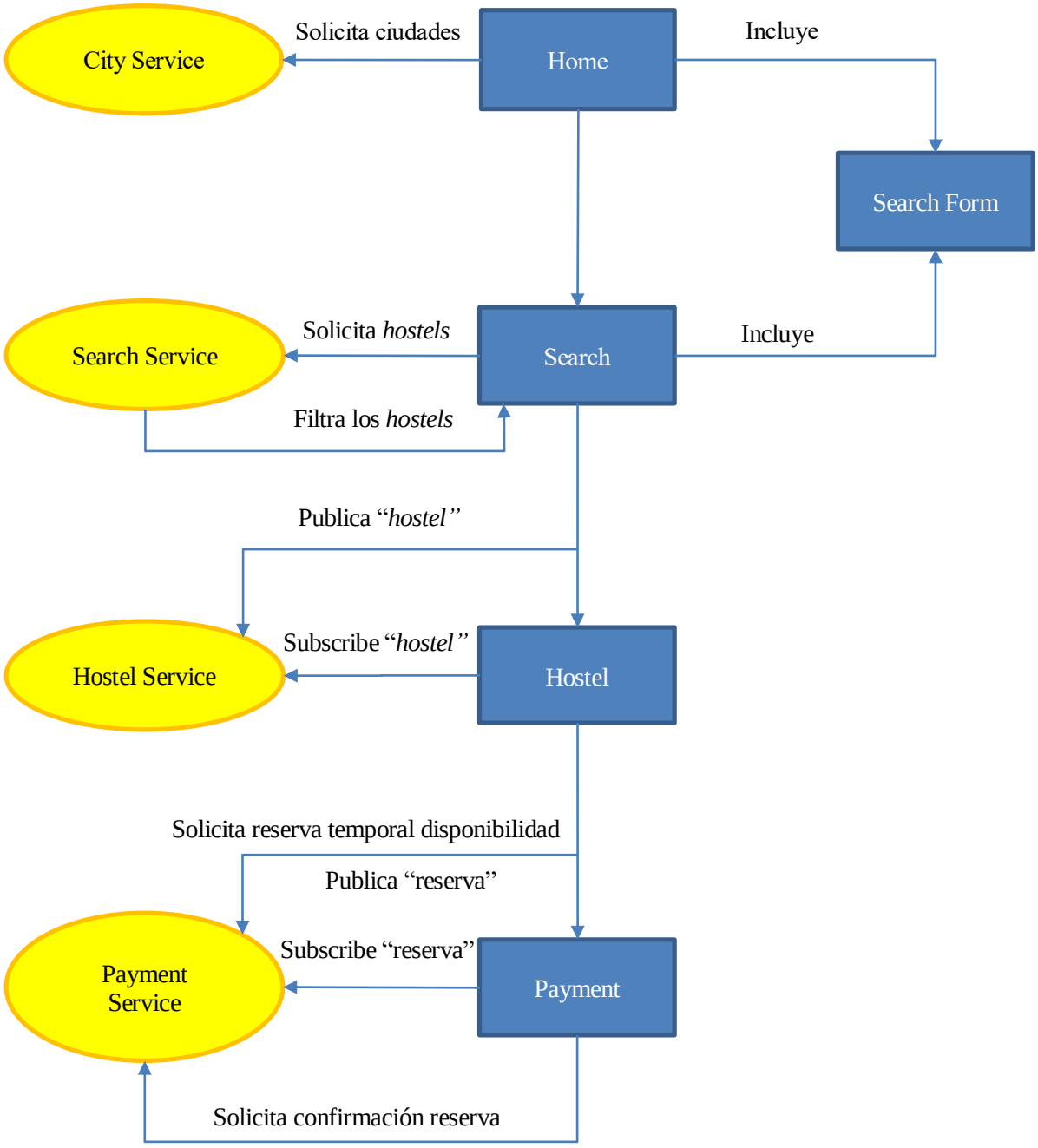


Figura 5-9. Diagrama de flujo del cliente

5.4 Despliegue en Heroku

Una vez desarrollados cliente y servidor, junto con la base de datos creada, se utiliza Git para desplegar la aplicación web en los servidores de Heroku. La aplicación consiste en un repositorio Git con las carpetas pertenecientes al servidor y con una carpeta resultante de una compilación AOT de la aplicación Angular, las cuales son subidas mediante un *git push*.

AOT (Ahead-of-Time) es una compilación de los ficheros HTML y TypeScript a código puramente JavaScript, mejorando la eficiencia y renderizado en el navegador. JIT (Just-inTime) es la otra opción, la cual compilaría a JavaScript en tiempo de ejecución en el navegador.

5.5 Metodología de desarrollo

En el desarrollo de software, existen numerosos enfoques, estilos o filosofías que guían el proceso. Basadas en el desarrollo ágil, destacan ciertas metodologías como *Scrum* o *Extreme Programming (XP)*. Esta última es en la que se ha basado el desarrollo de este proyecto.

La metodología de programación extrema fue formulada por Kent Beck, anteponiendo la adaptabilidad a la previsibilidad. Se apoya en la inmutabilidad de los requisitos de un proyecto, prefiriendo invertir en controlar sus cambios a definirlos al comienzo del proyecto.

La programación extrema se sostiene en cinco valores:

- Simplicidad.

  El diseño se simplifica con el fin de agilizar el desarrollo y facilitar el mantenimiento, siendo necesaria la refactorización del código. Lo mismo sucede con la documentación; el código debe estar autodocumentado, eligiendo correctamente los nombres de las variables, métodos y clases.

- Comunicación.

  La simplicidad del código mejora la comunicación. Los comentarios quedan desfasados con el código tras las sucesivas modificaciones, por lo que debe comentarse únicamente lo que no vaya a variar. Otra manera de comunicarse es el uso de pruebas unitarias, que describen el diseño de los componentes mostrando ejemplos de su funcionalidad.

- Retroalimentación.

  La integración del cliente en el proyecto reduce los ciclos de desarrollo, conociendo en tiempo real los requisitos. Las pruebas unitarias también son una fuente de retroalimentación al informar sobre posibles fallos en el código.

- Coraje.

  Se refiere al hecho de que el programador sea capaz de reconstruir su código, o incluso de desecharlo.

- Respeto.

  Se basa en las buenas relaciones entre los miembros del proyecto con el fin de mejorar la producción.

En el desarrollo del proyecto, se han aplicado estos valores. El proceso de diseño se ha limitado a describir los esquemas de datos de la aplicación, así como detallar las funcionalidades de la API y los distintos componentes del cliente. Durante la implementación, se ha refactorizado el código, modificándose el diseño original; por ejemplo, se han añadido nuevos atributos a los esquemas de datos iniciales.

6 RESULTADOS

Ya expuesto el diseño e implementación de la aplicación web, se presenta el resultado de la misma. En esta sección, se hace un recorrido a través de la página web, mostrando la vista o interfaz gráfica presentada en cada componente o paso de la navegación.

Como cada vista está asociada a un componente, se divide la sección en apartados respectivos a cada componente, explicando que interacciones son posibles y cuáles son las consecuencias de las mismas.

6.1 Home

Al acceder al enlace *https://www.whathostel.com* se encuentra el primer componente de la página. Este componente presenta una imagen de fondo que cubre toda la pantalla. En la parte superior, podemos localizar el logo y los botones que nos dirigen a la intranet de WhatHostel. En el centro de la pantalla, aparece el lema y el formulario correspondiente al componente *search-form*. Si hacemos *scroll* hacia abajo, vemos el *footer* de la página, con la información de contacto.

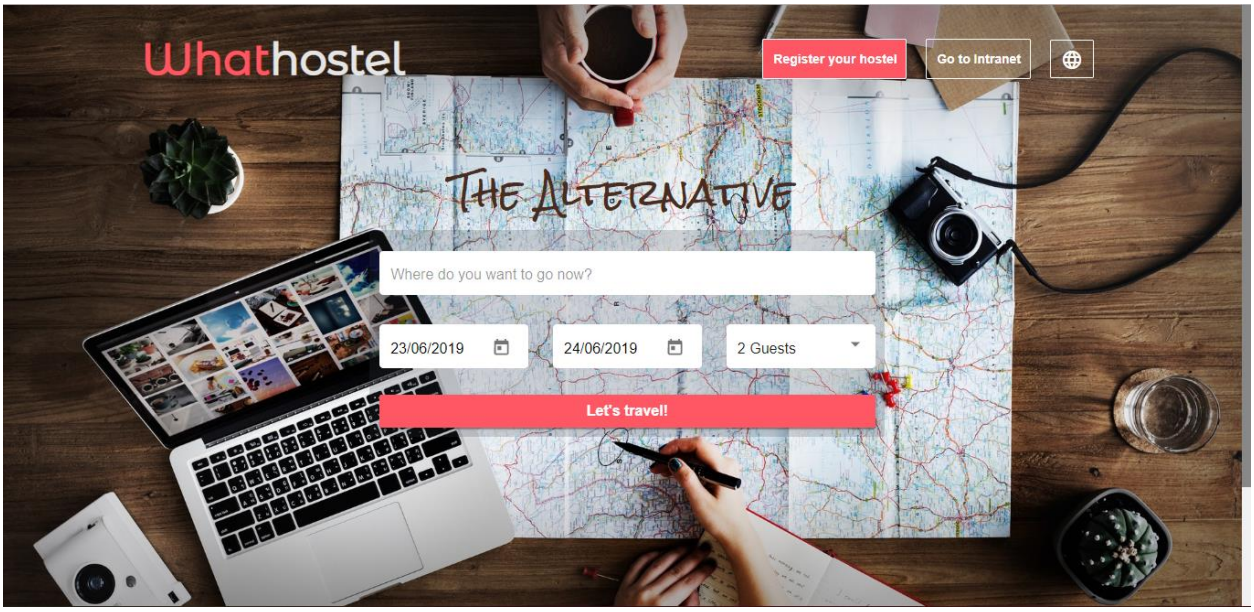


Figura 6-1. Página de *home* o de inicio

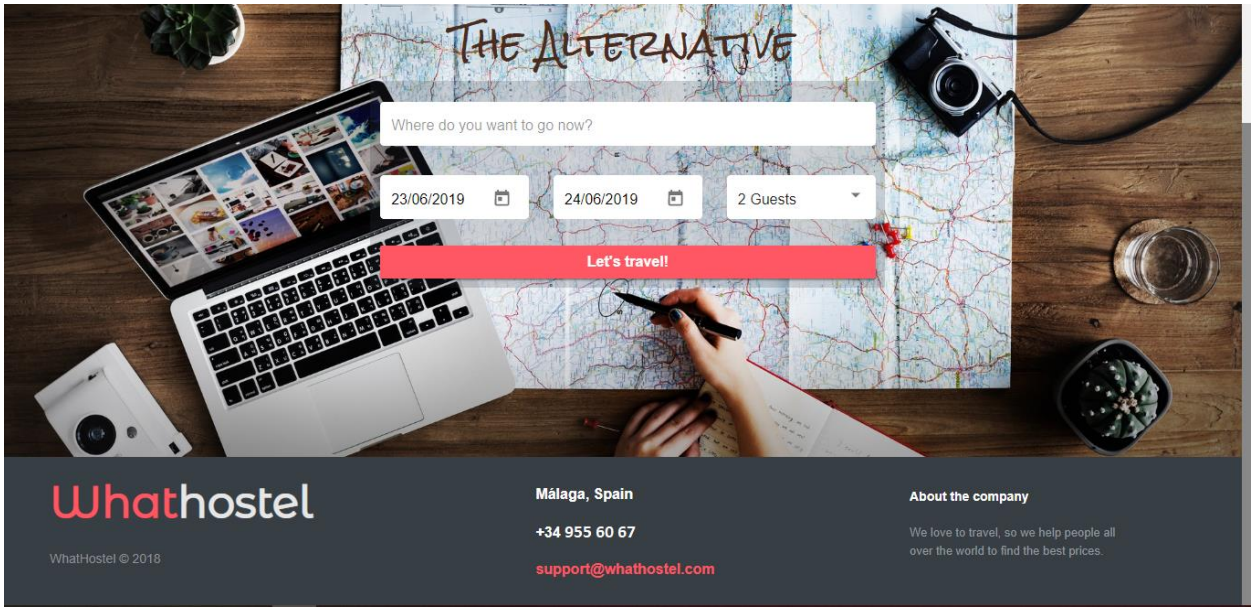


Figura 6-2. *Footer*

Se empieza a introducir los datos, y se ve como al escribir tres letras se ofrece sugerencias.

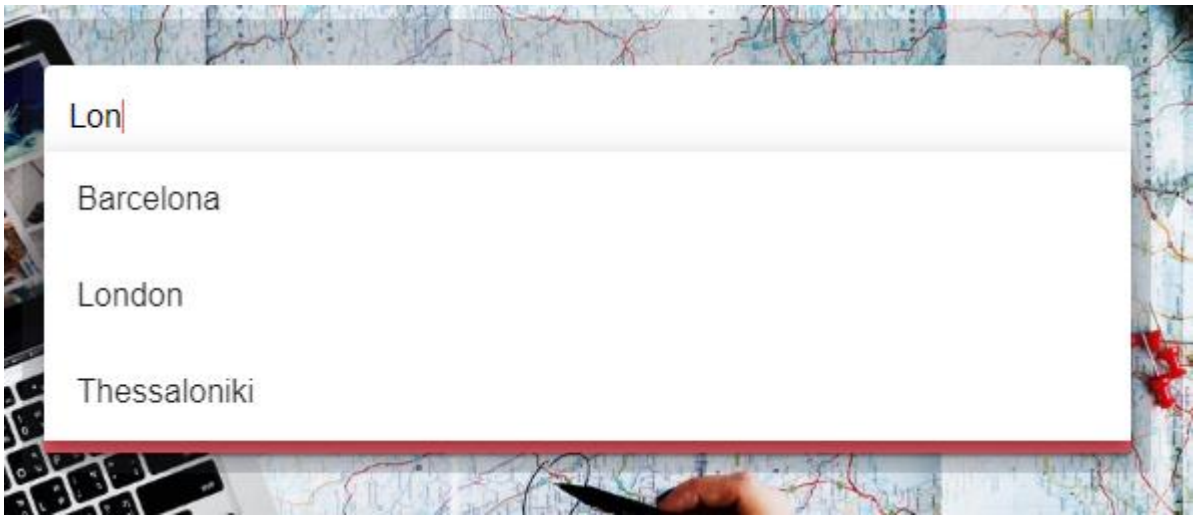


Figura 6-3. Sugerencias de ciudades

A continuación, se indican las fechas y el número de personas.

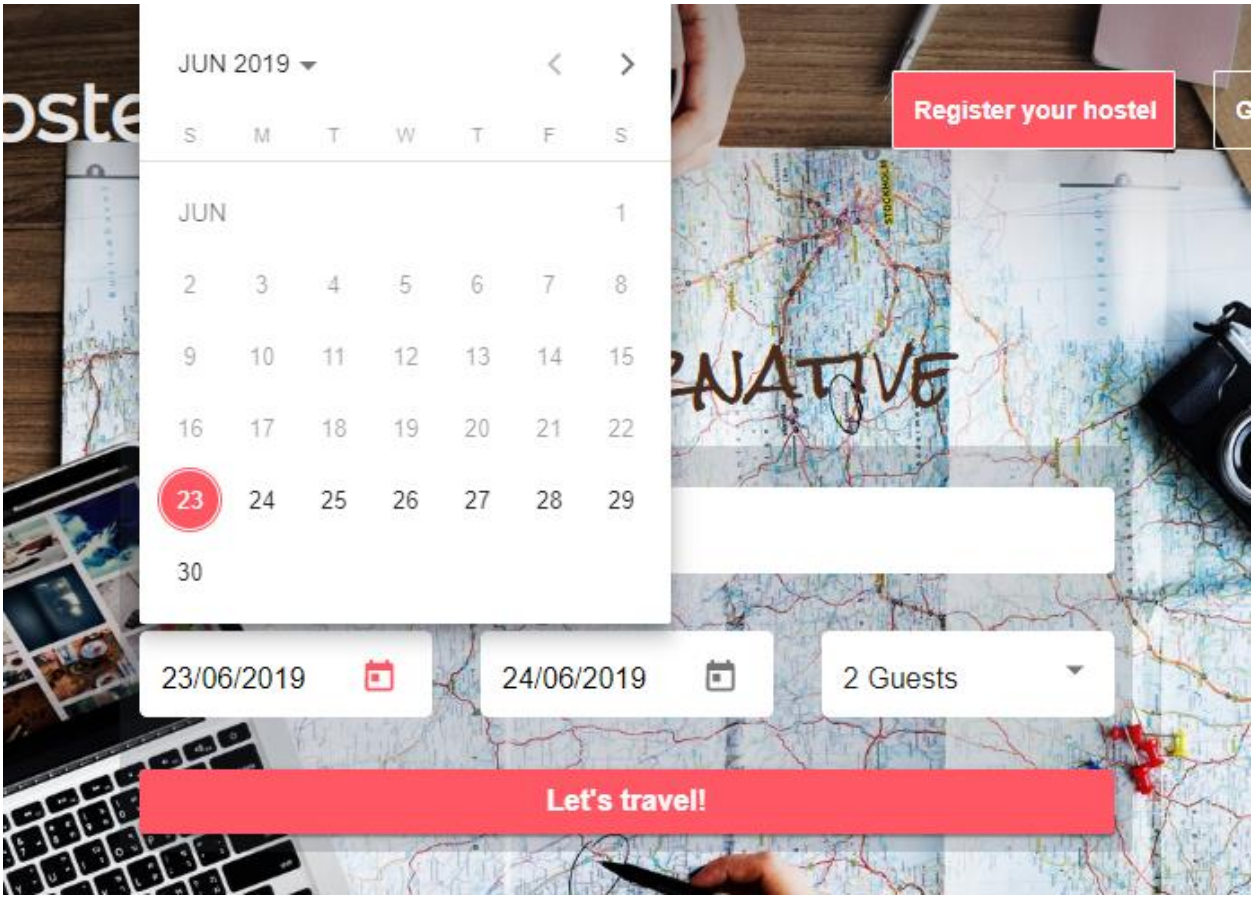


Figura 6-4. Selección de fechas

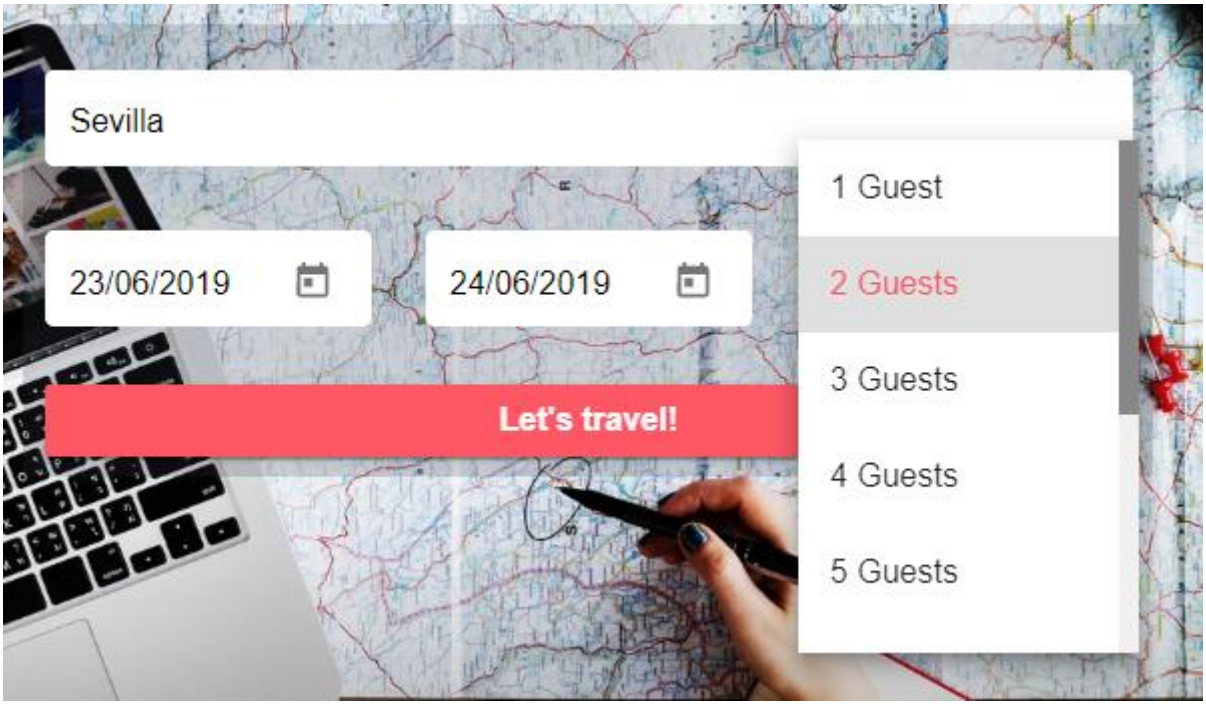


Figura 6-5. Selección de huéspedes

Si se pulsa en el botón *Let's travel!*, se pasa al siguiente componente con la información proporcionada.

6.2 Search

El siguiente componente, como se dijo en la sección anterior, es *search*. En la parte superior se puede observar el logo superpuesto con el mapa, en el que aparecen las localizaciones de los resultados de la búsqueda.

En la barra inferior del mapa, aparece nuevamente el componente *search-form* con los datos completados anteriormente. Seguidamente, se tienen dos *checkboxes* para elegir la preferencia de ordenación de los resultados. Por último, en la misma barra, se tienen las opciones de filtrado.

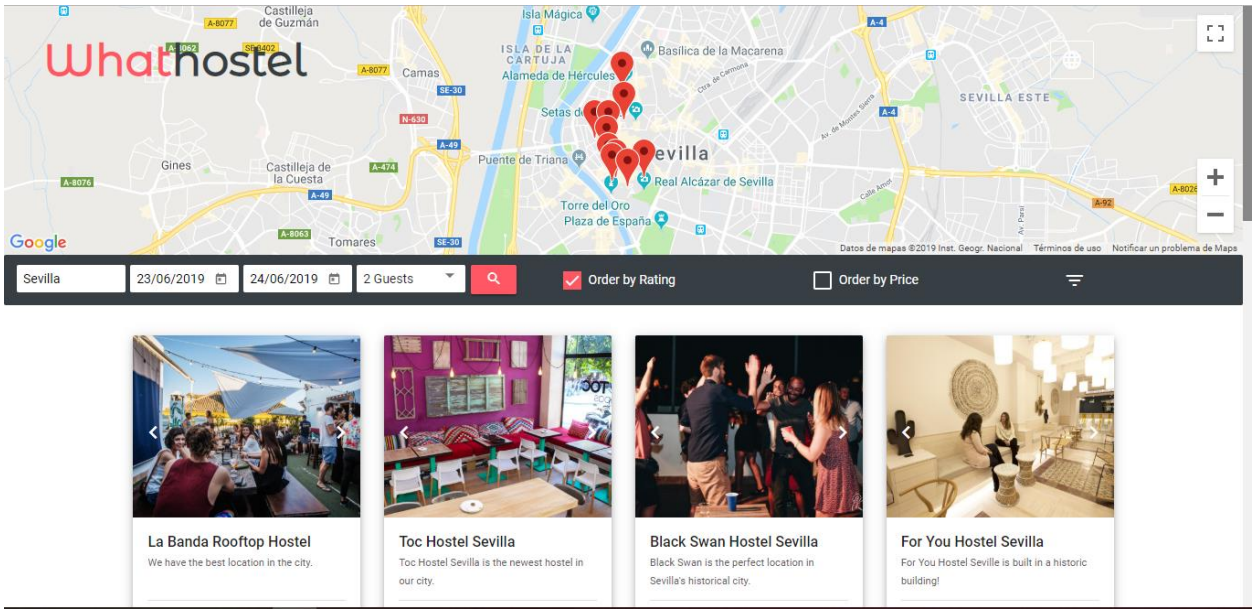


Figura 6-6. Parte superior de la página de resultados

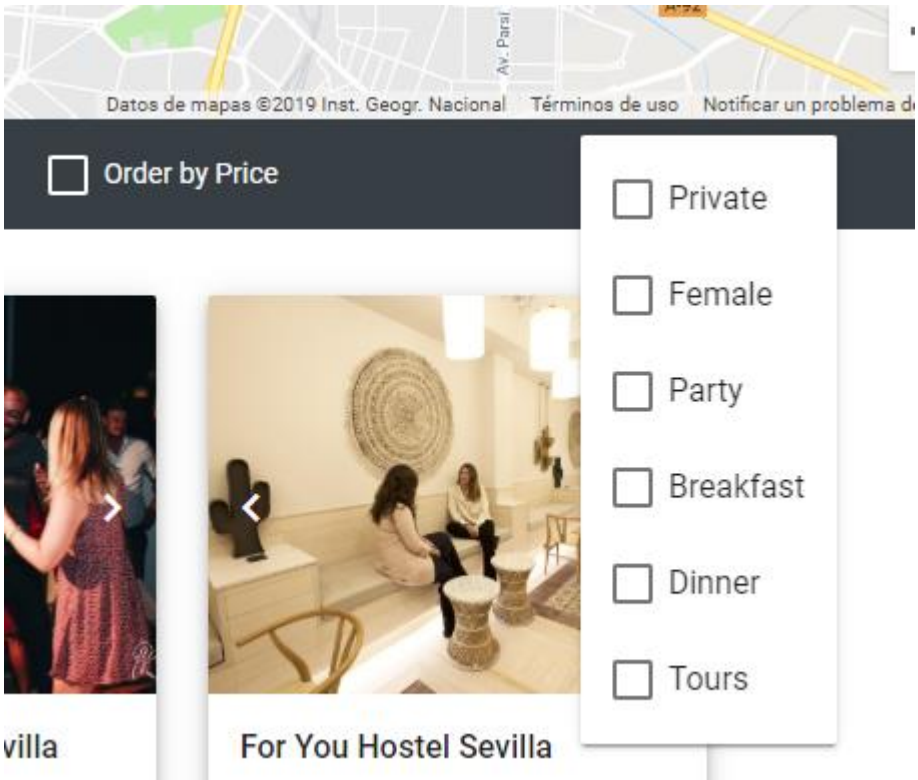


Figura 6-7. Opciones de filtrado

Bajo la sección del mapa, aparecen los resultados filtrados. Cada *hostel* válido para las opciones de búsqueda y filtros, se representa mediante una tarjeta.

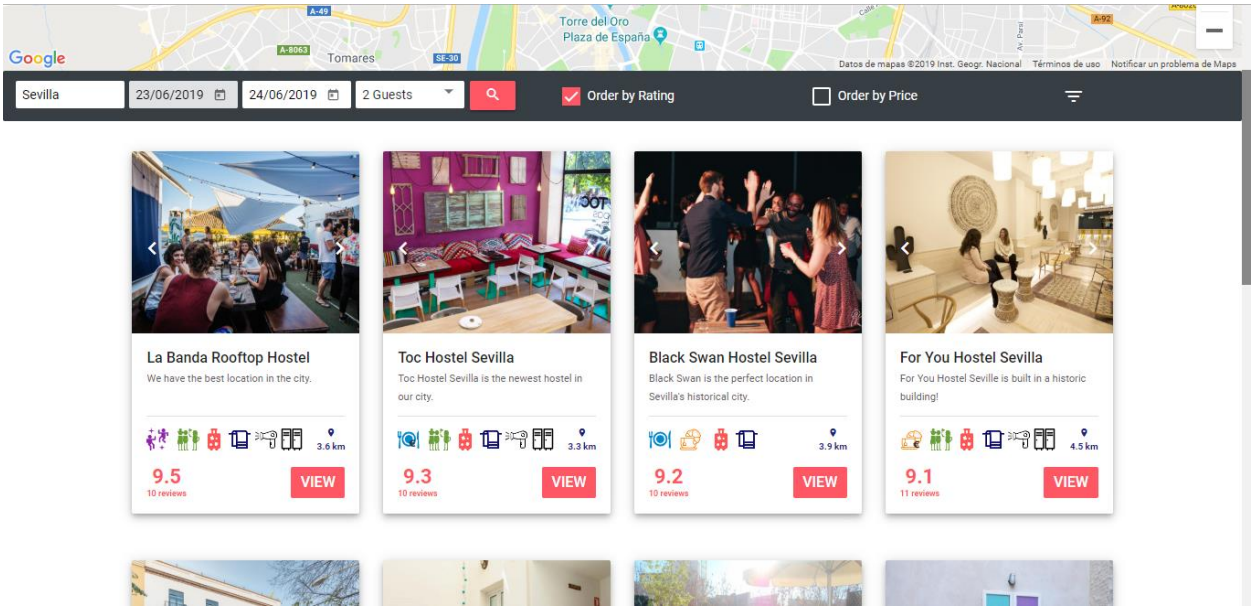


Figura 6-8. Resultados de la búsqueda

La tarjeta contiene una imagen de la propiedad, seguida de su nombre y una breve descripción. En la parte inferior, se muestran iconos que describen características y servicios del *hostel*; la puntuación, el número de reviews y el precio mínimo.

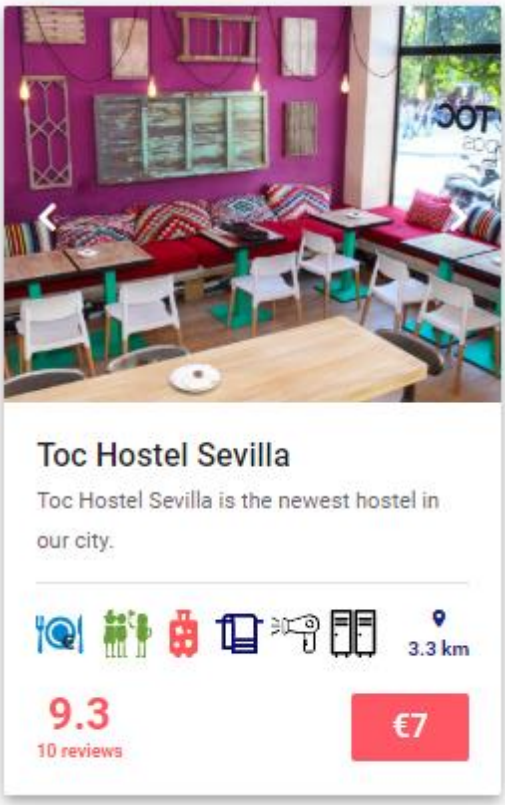


Figura 6-9. Tarjeta de la propiedad

Al pasar el cursor por encima de la tarjeta, esta se eleva y comienza a cambiar la imagen automáticamente; también se pueden pasar imágenes pulsando en las flechas que la superponen. Si se pasa el puntero por encima de uno de los iconos, se indica su significado.

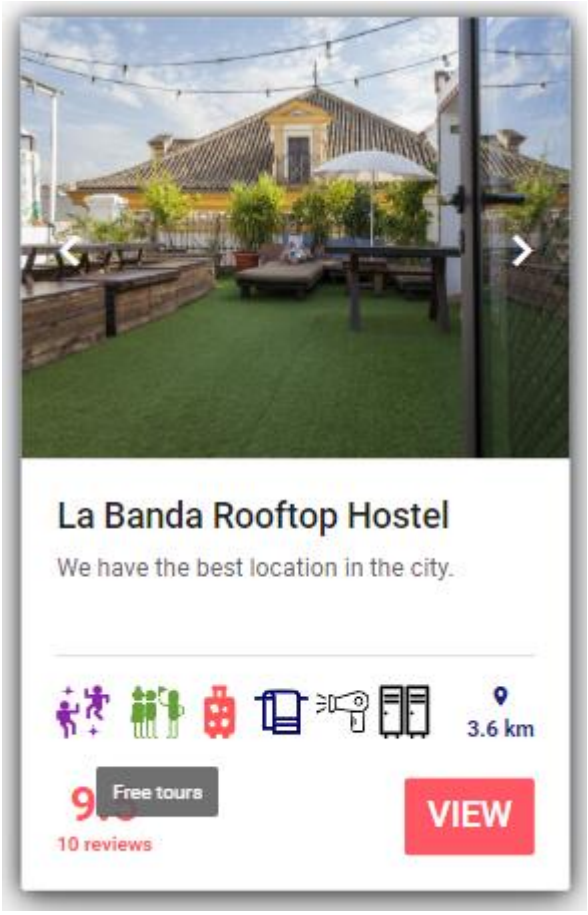


Figura 6-10. Ratón sobre icono de la tarjeta

Si se pulsa en una tarjeta, se pasa al siguiente componente con el *hostel* seleccionado.

6.3 Hostel

El componente *hostel* muestra la información detallada de la propiedad seleccionada, y permite elegir las habitaciones que se quieren reservar.

En la parte izquierda, se muestra la puntuación desglosada sobre la imagen de la propiedad. Además, se ven miniaturas de las otras imágenes disponibles, que se pueden ver pulsando sobre las flechas laterales o posando el puntero sobre una de ellas. Si se introduce el cursor en esta zona, la puntuación se esconde para poder ver completa la imagen.

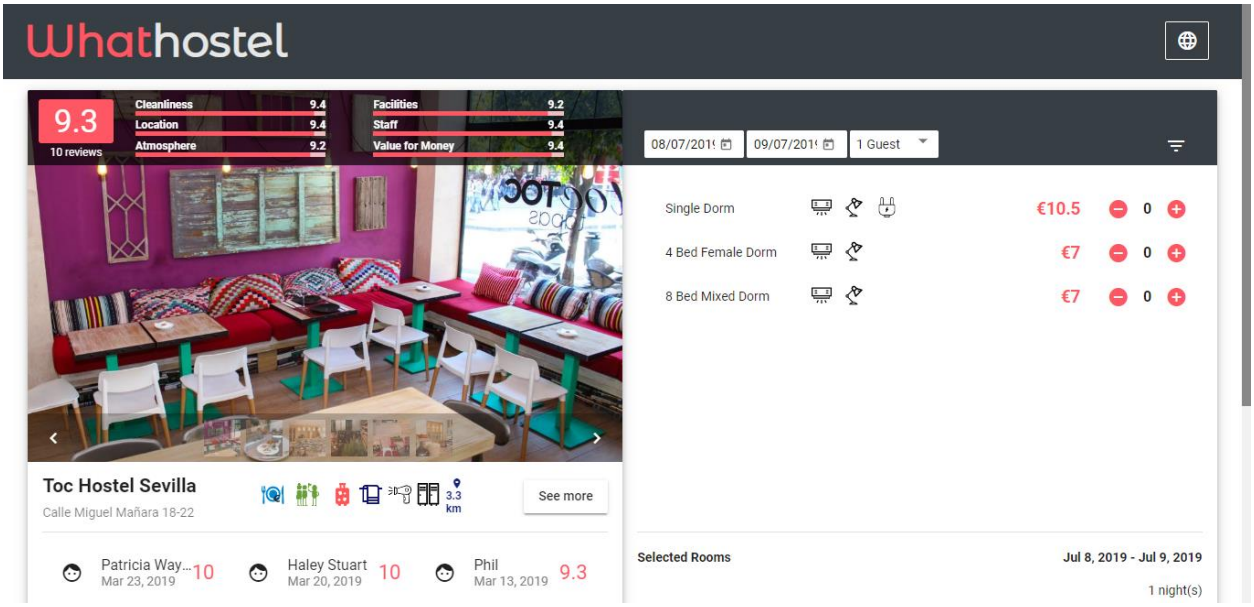


Figura 6-11. Vista del *hostel*

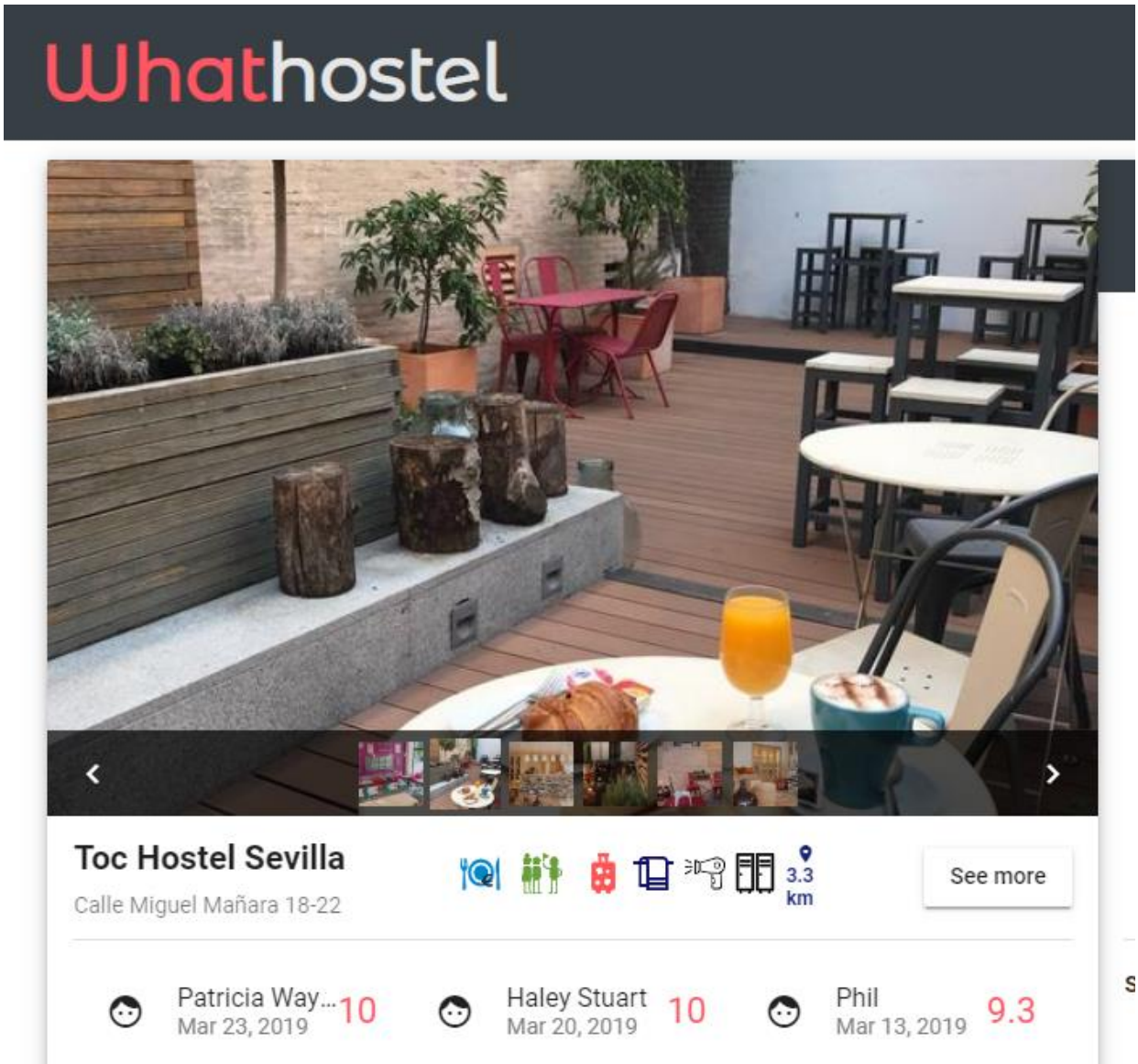


Figura 6-12. Cursor sobre imagen

Bajo la imagen, aparece nuevamente el nombre de la propiedad junto con los iconos descriptivos. A la derecha, se tiene un botón que da la vuelta a la tarjeta para ver más información.

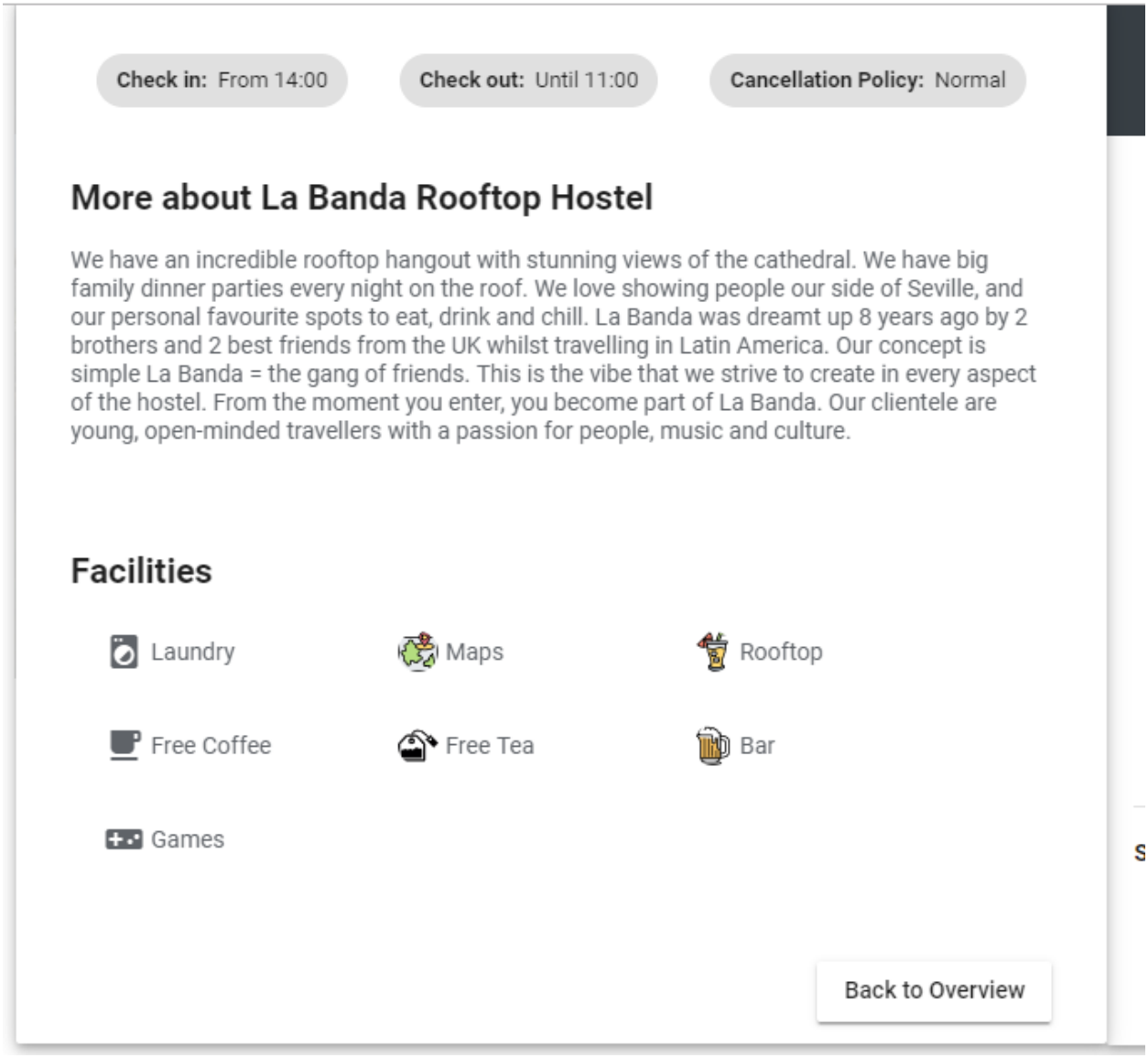


Figura 6-13. Parte trasera de la tarjeta izquierda

En la parte inferior de la tarjeta izquierda, aparecen las opiniones de tres en tres, pudiendo navegar por ellas a través de las flechas de la esquina inferior derecha. Para leer una reseña, basta con posar el puntero encima de ella.

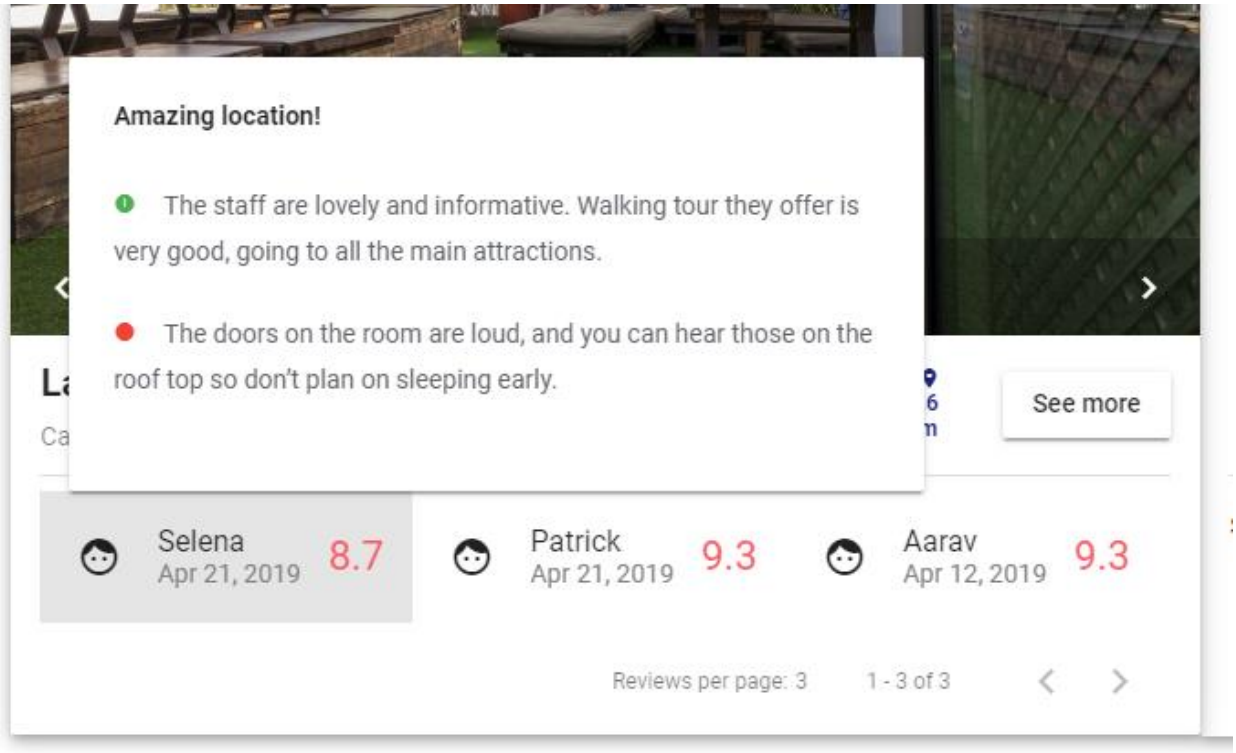


Figura 6-14. Puntero sobre reseña

En la parte derecha de la pantalla, se muestran las habitaciones que cumplen con los criterios de búsqueda, junto con iconos que describen utilidades y con los botones de selección. Abajo, aparece el resumen de las habitaciones y camas seleccionadas para proceder al pago.

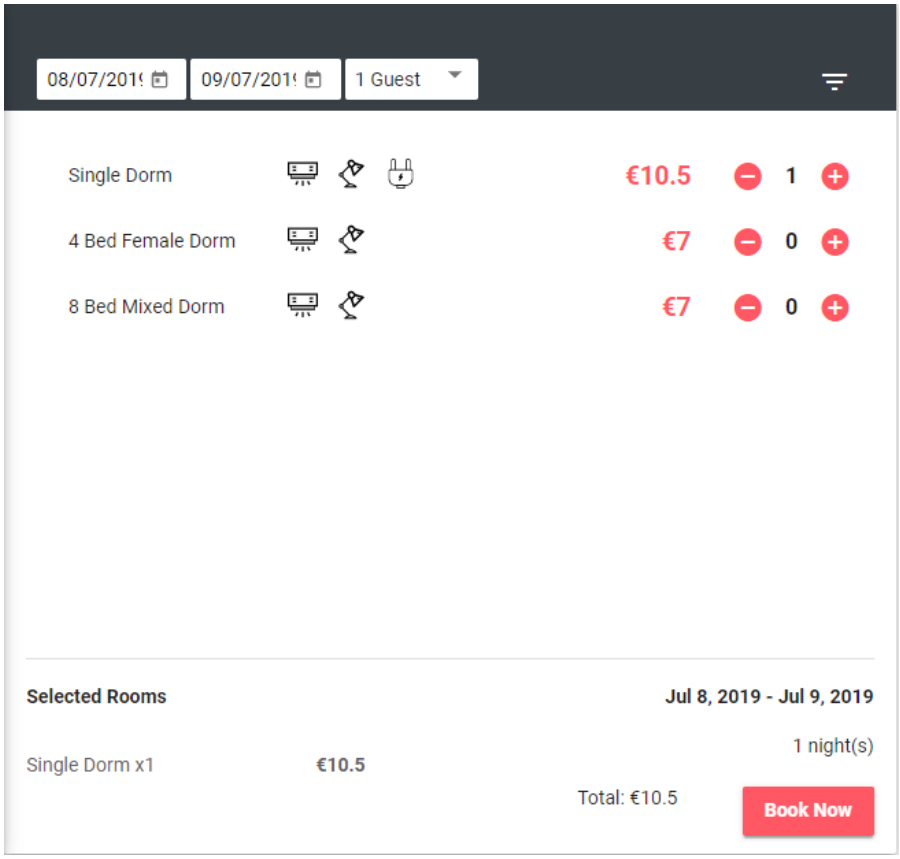


Figura 6-15. Tarjeta de selección

Se permite filtrar la lista por habitaciones privadas y femeninas.

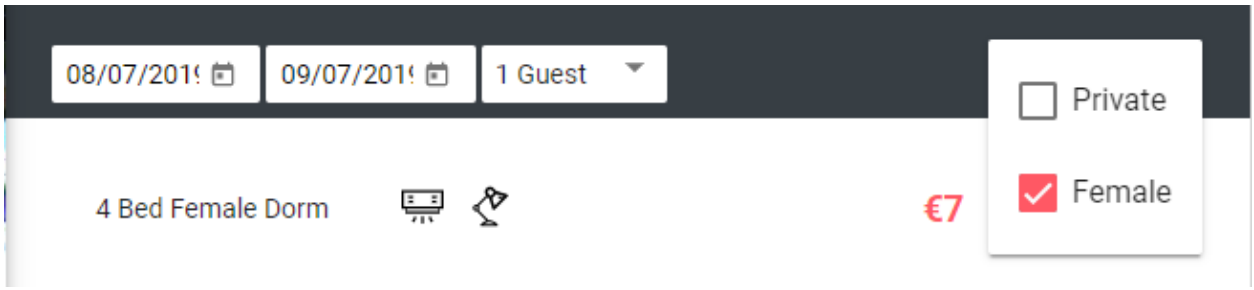


Figura 6-16. Filtro habitación femenina

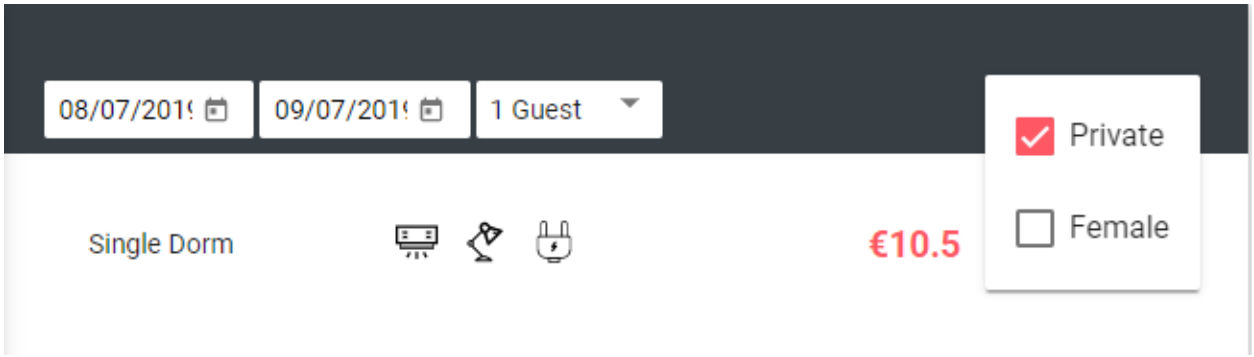


Figura 6-17. Filtro habitación privada

Si se pulsa el botón *Book Now*, se reserva temporalmente la disponibilidad y se procede al pago.

6.4 Payment

En este componente, se muestra el formulario de reserva e información de pago a la izquierda, y un resumen de la reserva a la derecha.

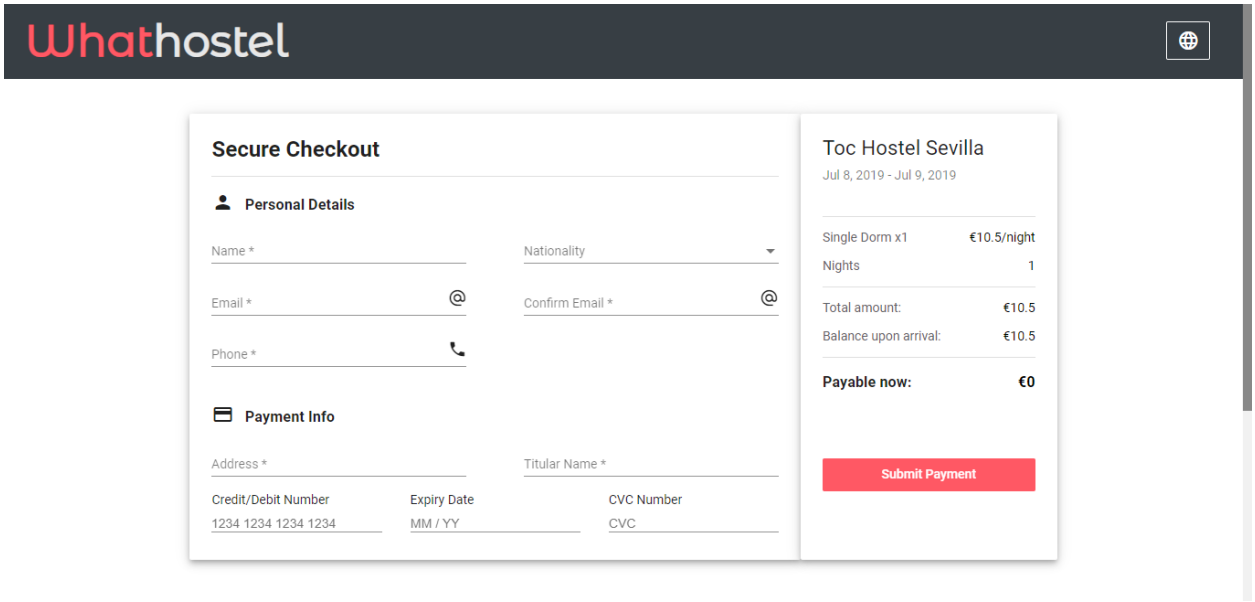


Figura 6-18. Formulario de pago y resumen de reserva

Tras introducir los datos y pulsar el botón *Submit Payment*, se validan y se completa el proceso de reserva.

7 CONCLUSIONES

Todo proyecto está sujeto a condiciones, riesgos e incertidumbre. El desarrollo de software requiere de creatividad y agilidad a la hora de solucionar los problemas que surgen durante sus fases. Desde el diseño hasta el mantenimiento de una aplicación web, los requisitos cambian ante la aparición de limitaciones; y el desarrollador debe ser capaz de adaptarse a estos cambios, de una forma eficiente.

En cada paso que se da, es crucial la evaluación de los resultados y, sobre todo, de los imprevistos a los que se ha tenido que hacer frente con el fin de aprender para el siguiente paso. Por ello, en esta sección se presentan los problemas acarreados durante el desarrollo, así como su solución; y las ventajas obtenidas de la metodología empleada.

7.1 Problemas encontrados

Ya sea por la tecnología empleada o por la naturaleza misma del objetivo del trabajo, han aparecido numerosas dificultades durante las distintas fases de la realización del proyecto.

7.1.1 Dificultades de diseño

- La fase de diseño implica conectar las distintas partes de la aplicación. Entender el funcionamiento de los protocolos de aplicación empleados en Internet, han facilitado la estructuración de los componentes; servidor y cliente como partes diferenciadas comunicadas por HTTP.
- La división de los datos en tres esquemas requiere de un diseño correcto en el que se especifique como estos están relacionados. Durante los primeros ciclos de la fase de diseño, se han formado bucles en estas relaciones hasta encontrar una estructura eficiente y útil.

7.1.2 Dificultades de desarrollo

- Para el desarrollo, el entorno debe estar configurado adecuadamente. NodeJS y Angular son tecnologías que cambian continuamente, por lo que se debe actualizar de manera progresiva, atendiendo a los cambios necesarios para retomar la codificación.
- El uso de lenguajes no tipados como JavaScript en el lado del servidor ha dificultado la depuración de la aplicación. La mayoría de errores son detectados en tiempo de ejecución, por lo que se debe recrear la situación hasta conseguir resolverla.
- Las tecnologías escogidas presentan una curva de aprendizaje grande, debido a la naturaleza asíncrona de las mismas. La adaptación de un paradigma de programación lineal a uno basado en eventos y *callbacks* resulta complicada en primera instancia.

7.1.3 Dificultades de implementación

- Para que las comunicaciones entre las partes sean seguras se usa SSL. La creación de un certificado autofirmado es insuficiente ya que no está reconocido por el navegador. Para que el usuario sea consciente de que las comunicaciones están cifradas mediante HTTPS se ha usado un certificado gratuito proporcionado por Let's Encrypt, y configurado en la plataforma de Heroku.

7.1.4 Dificultades de mantenimiento

- La existencia de numerosos módulos crea a su vez problemas de compatibilidad. Se deben corregir las versiones no compatibles y comprobar el funcionamiento de la aplicación tras las actualizaciones.

7.2 Aportaciones de la metodología y la tecnología empleadas

La programación extrema ha permitido desarrollar la aplicación en un tiempo relativamente corto, nueve meses. La brevedad del diseño ha facilitado las tareas de adaptación posteriores, permitiendo incorporar nuevos atributos a los datos sin alterar el resto de componentes.

El uso de lenguajes interpretados agiliza el proceso de desarrollo evitando las esperas de compilación. Además, es interesante el aprendizaje de un nuevo paradigma, ampliando las posibilidades en la realización de un proyecto.

En conclusión, la realización de un proyecto complejo debe ser analizada y dividida en partes con el fin de facilitar la consecución de los objetivos propuestos. El desarrollo de este trabajo muestra las dificultades que se pueden encontrar en un proyecto real, así como la existencia de múltiples alternativas para abordarlo, cada una con sus ventajas e inconvenientes.

8 LÍNEAS FUTURAS

Hasta aquí se ha cubierto el alcance del proyecto. En esta sección, se indican los siguientes pasos para concluir los objetivos restantes. Una vez cumplido el objetivo de desarrollar una aplicación web para la reserva de *hostels*, queda la creación o adaptación de dicha aplicación al dispositivo móvil.

La creación de aplicaciones móviles es una tendencia actual. El incremento del uso del móvil frente al uso de las aplicaciones de escritorio justifica la necesidad de aparecer también en esta plataforma.

Se analizan los tipos de aplicaciones móviles que se pueden desarrollar, con la intención de determinar la más apropiada para reutilizar la aplicación web creada.

8.1 Tipos de aplicaciones móviles

Cada tipo tiene sus ventajas e inconvenientes, y es necesario tenerlas en cuentas para decidir por cual decantarse.

Tipo de aplicación móvil	Ventajas	Inconvenientes
App Nativa	• Permite acceso completo al dispositivo • Mejor experiencia para el usuario • Permite el envío de notificaciones • Existencia en APP Store	• Diferencias de desarrollo para cada plataforma (IOs, Android) • Mayor coste asociado • No se puede reutilizar código entre plataformas
Web App	• El código es reutilizable para las diferentes plataformas • Menor coste asociado • No hay necesidad de autorización por las plataformas para su publicación • El usuario dispone de la última versión en todo momento • Se reutiliza la versión *responsive* de web correspondiente	• Se necesita conexión a Internet • El acceso a las características y hardware del dispositivo es muy limitado • La experiencia del usuario y el tiempo de respuesta es peor • No se encuentra en el APP Store
Web App Nativa	• Puede distribuirse en la APP Store • Se reutiliza el código • Permite el acceso a parte del hardware del	• La experiencia del usuario se aproxima más a la de una Web App • Los componentes visuales no están siempre relacionados con el

	dispositivo	sistema operativo nativo

Tabla 8–1. Tipos de aplicaciones móviles

8.2 NativeScript

NativeScript es un *framework* para el desarrollo de aplicaciones nativas, pero empleando tecnologías web. Tiene soporte directo para el desarrollo con Angular y permite la reutilización de código.

NativeScript y los complementos utilizados se instalan mediante *npm*, un gestor de paquetes. Las interfaces de usuarios se definen utilizando archivos XML, que son procesados por NativeScript para asociar a cada componente visual su correspondiente elemento de interfaz nativo para cada plataforma. La lógica puede ser compartida con la empleada por la aplicación web.

Emplea la reflexión para obtener información sobre las APIs de la plataforma nativa, y así poder hacer uso de las funciones agregadas.

Tras una configuración previa del entorno empleado para el desarrollo de la aplicación web, se pude integrar NativeScript para crear aplicaciones móviles nativas para cada plataforma, definiendo únicamente la interfaz gráfica mediante archivos XML y haciendo los cambios oportunos en los ficheros TypeScript; por ejemplo, para añadir módulos específicos de NativeScript.

REFERENCIAS

[1] Introduction to the server side. (2019, 19 marzo). Recuperado 3 junio, 2019, de https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

[2] Uenlue, M. (2018, 5 agosto). Business models compared: Booking.com, Expedia, TripAdvisor. Recuperado 5 junio, 2019, de https://www.innovationtactics.com/business-models-tripadvisor-booking-com-expedia/

[3] Schmidt, A. (2015, 28 septiembre). What Business Model Does Expedia Follow? - Market Realist. Recuperado 8 julio, 2019, de https://marketrealist.com/2015/09/business-model-expedia-follow/

[4] Cracan, M. (2018, 20 junio). Web REST API Benchmark on a Real Life Application. Recuperado 16 junio, 2019, de https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3

[5] Sarig, M. (2018, 29 noviembre). MongoDB vs MySQL: the differences explained. Recuperado 21 junio, 2019, de https://blog.panoply.io/mongodb-and-mysql

[6] Shah, M. (2019, 1 julio). MongoDB vs MySQL: A Comparative Study on Databases. Recuperado 4 julio, 2019, de https://www.simform.com/mongodb-vs-mysql-databases/

[7] Sánchez, R. (2018, 31 mayo). Comparing Node.js vs PHP Performance. Recuperado 16 junio, 2019, de https://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/

[8] ThinkMobiles Team. (2019, 25 febrero). PHP vs NodeJS Comparison and Benchmark. Recuperado 16 junio, 2019, de https://thinkmobiles.com/blog/php-vs-nodejs/

[9] Google. (s.f.). Angular. Recuperado 20 mayo, 2019, de https://angular.io/docs