Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Pack2Fly: Sistema Web de Paquetes Vacacionales

Pack2Fly: Vacational Packs Web System

Jesús Navarro Hernández

La Laguna, 10 de junio de 2021

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Pack2Fly: Sistema Web de Paquetes Vacacionales*"

ha sido realizada bajo su dirección por D. **Jesús Navarro Hernández**, con N.I.F. 51.152.195-B.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2021

Agradecimientos

Quisiera agradecer a todas las personas que desde pequeño han intentado inculcarme la educación como la mejor vía para lograr tus sueños y convertir este mundo en un lugar mejor.

A mis padres, que siempre me han animado a seguir adelante, a nunca rendirme.

A mi novia, por ser mi mayor pilar, y por hacerme soñar con que todo es posible, por muy difícil que parezca.

A todos mis seres queridos, por apoyarme y convertirme en quien soy.

Licencia

* Si quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:

Resumen

Viajar es uno de los mayores placeres de la vida. La emoción de enfrentar nuevas experiencias nos llena de motivación y constituye una de las mayores aspiraciones que una persona puede desear. Sin embargo, existe un problema en el proceso: La dificultad de organizar correctamente un viaje. Supone horas y horas de trabajo, buscando vuelos que se ajusten a los interesen personales, alojamiento que cumpla los criterios propios y que a su vez se ajusten al vuelo, actividades a realizar en el destino... ¿Y si fuera posible obtener todo ello con un simple click?

Pack2Fly es una aplicación que ayuda al usuario a encontrar su viaje soñado con la comodidad y la rapidez que permite al mismo centrarse en disfrutar sus vacaciones, sin quebraderos de cabeza.

Por motivos técnicos, se ha optado por una aplicación web, es decir, ejecutada en una página web, usando servidores locales. Estos servidores son proveídos por las propias tecnologías en las que ha sido desarrollado: Angular y Node.js. Además, se ha usado WebStorm como IDE de desarrollo debido a su gran soporte para desarrollo web basado en las principales tecnologías actuales. Esto ha permitido integrar desarrollo y despliegue en un mismo entorno. La aplicación se basa en una arquitectura Backend-Frontend, en la que ambas presentan arquitecturas distintas: Microservicios para el backend y MVC (Modelo Vista Controlador) para el Frontend.

En esta memoria, se describe paso a paso cada elemento del proyecto software, indicando metodologías, herramientas y resultados, de modo que se entienda a la perfección cómo se ha realizado este proyecto.

Palabras clave: Pack2Fly, Aplicación Web, Node.js, Angular, Paquetes Vacacionales

Abstract

Traveling is one of the greatest pleasures in life. The excitement of facing new experiences fills us with motivation and constitutes one of the greatest aspirations that a person can wish for. However, there is a problem in the process: The difficulty of correctly organizing a trip. It involves hours and hours of work. We need to look for flights that adjust to our personal interests. Then, accommodation that meets your own criteria and that, at the same time, it adjusts to the flight. Finally, activities to be carried out at the destination and more. What if it were possible to obtain all this with a simple click?

Pack2Fly is an application that helps users find their dream trip with the comfort and speed that allows them to focus on enjoying their vacations, without headaches.

For technical reasons, Pack2Fly is built as a web application. This means that it is executed on a web page, using local servers. These servers are provided by the own technologies in which it has been developed: Angular and Node.js. In addition, WebStorm has been used as a development IDE due to its great support for web development based on the main current technologies. This has made it possible to integrate development and deployment in the same environment. The application is based on a Backend-Frontend architecture, in which both have different architectures: Microservices for the backend and MVC (Model View Controller) for the Frontend.

In this report, each element of the software project is described step by step, indicating methodologies, tools and results, so that it is fully understood how this project has been carried out.

Keywords: Pack2Fly, Web Application, Javascript, Angular, Vacational Packs

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En el mundo en el que vivimos, donde la tecnología avanza más y más, la adaptabilidad a las mismas es prácticamente una necesidad. Es por ello que las empresas invierten una cantidad ingente de dinero en el desarrollo de servicios que se ajusten a las demandas de sus clientes potenciales.

Este proyecto de TFG nace como una oportunidad que busca fusionar las tecnologías actuales con las necesidades que se demandan en el sector turístico. Históricamente, el turismo se ha basado en la interración entre cliente y una agencia de viajes. Esta realiza las tareas de búsqueda de vuelos, hospedaje, excursiones, etc. A medida que han ido avanzando las distintas herramientas, el cliente ha comenzado a realizar las distintas actividades relacionadas con un viaje por su propia cuenta (haciendo uso Kayak, Booking, etc). Sin embargo, el peso de las agencias se mantuvo (con alguna bajada notoria) ya que muchos clientes aún preferían las comodidades que se le ofrecían, obteniendo todo organizado y preparado.

La idea principal del sistema web que se va a implementar es simplificar la tarea de organizar un paquete vacacional por parte del usuario de tal manera que con tan solo unos clicks, el cliente obtenga un paquete vacacional completo. Es decir, la aplicación actuará como una agencia de viajes, con todas las ventajas de las nuevas tecnologías (Instantaneidad, comodidad, sencillez...).

¿Por qué surge esta idea? Muchos de nosotros hemos realizado viajes recreativos o laborales a lo largo de nuestra vida. Viajar es, sin duda, una de las actividades que más nos motiva e ilusiona, ya que nos permite disfrutar de multitud de culturas, conocimientos, valores históricos, elementos artísticos, clima, paisajes, etc. Es decir, nos permite descubrir toda la belleza que ofrece nuestro mundo, ampliando nuestros horizontes. Aún así, el concepto del turismo implica una actividad tediosa y compleja: la organización. Esta actividad presenta una complejidad variable, ya que lógicamente dependerá de factores como la duración, número de vuelos, lugares a visitar, etc. Independientemente de esto último, por muy simple que parezca, la organización del viaje será, como mínimo, estresante.

Cuando pensamos en ello, surge casi al instante una duda: ¿Se podría simplificar este proceso? Al plantear la misma, surgió la idea: Crear una aplicación que realice todo el trabajo complejo, de tal manera que el usuario elimine la necesidad de hacerlo por su cuenta. Así, se simulará el trabajo realizado por las agencias de viajes, en la que el cliente les hace llegar sus intereses, así como sus requisitos, para obtener finalmente una experiencia de su agrado. Además, conforme avanza la tecnología, resulta cada vez más sencillo implementar un sistema capaz de lograr cumplir esta tarea, adaptándose a la evolución tecnológica de la población, que cada vez recurre más a las comodidades de las aplicaciones para ayudarse en sus tareas.

1.1. Evolución de las TIC en el Turismo

Resulta interesante poder verificar mediante datos reales esta tendencia que hemos comentado anteriormente. Para ello, vamos a proceder al análisis de datos procedentes del ISTAC [1] para conocer la información del uso de Internet a la hora de conocer Canarias como destino turístico por parte de los turistas (en términos generales, sin indagar en su procedencia). Así, comenzaremos con el primer gráfico que muestra los datos referentes al número de turistas según formas de conocer Canarias como destino turístico por países de residencia (véase figura A.3)

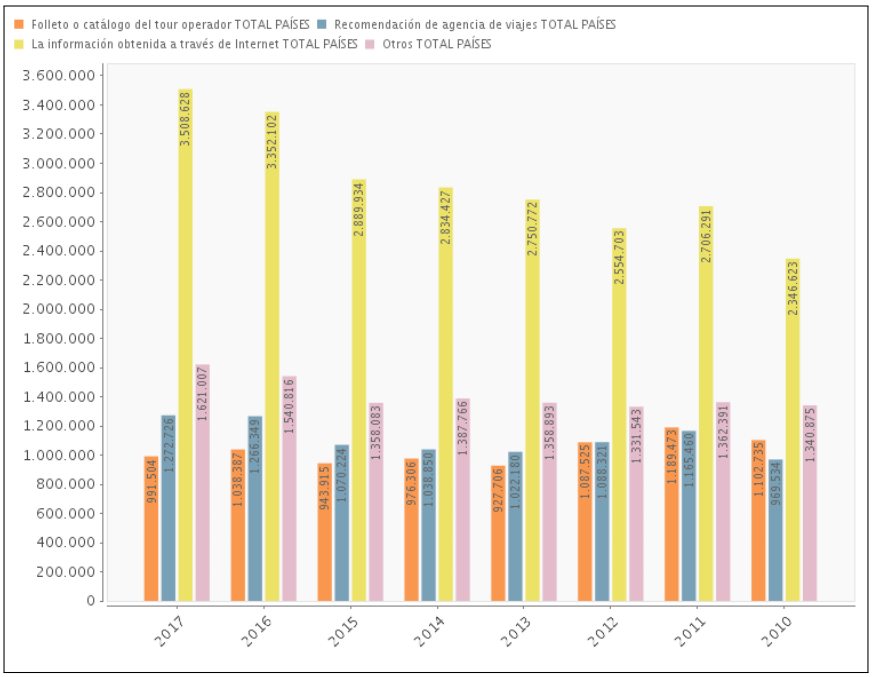


Figura 1.1: Evolución del número de turistas en Canarias por fuente de información

Estos datos muestran la evolución del número de turistas recibidos por año en canarias, desglosándolos según el medio utilizado (según encuestas) para obtener información sobre el archipiélago. Podemos observar como el grafo de color amarillo, correspondiente a "información obtenida a través de Internet" comienza con un valor de 2.346.623 en 2010. Vemos que presenta un crecimiento en todos los años (salvo en el año 2012, donde decrece ligeramente) hasta alcanzar, en el año 2017 un valor de 3.508.628. Esto se corresponde con la tendencia al uso de las nuevas tecnologías en lugar de las tradicionales, que podemos observar en la gráfica que presentan leves bajadas o una tendencia a mantenerse (véase el resto de grafos). Esta evolución se ha mantenido en progreso constante hasta nuestros días.

La situación a nivel nacional también ha presentado un crecimiento acorde. Según un artículo de Hosteltur [2], El 86 % de los españoles reserva sus viajes a través de Internet, a fecha de mayo del 2019. Este porcentaje habla por sí solo. Si pensamos en el número de clientes, vemos que por cada 100 clientes, 86 deciden reservar sus viajes de manera *online*, mientras que tan solo 14 lo hacen a través de una agencia. La diferencia entre este porcentaje y el de años anteriores es increíble. Para tener una idea del crecimiento, si nos fijamos en el año 2015, un artículo del periódico El País [3] recoge que el 66 % elige usar Internet para reservar sus viajes. Esto significa que en un periodo de tan solo 4 años, el porcentaje ha aumentado un 20 %. Sin embargo, también debemos darnos cuenta de que el porcentaje alcanzando en 2019 posiblemente se mantenga a lo largo de estos años a corto plazo, ya que ese 15 % aproximadamente de clientes difícilmente cambiarán sus hábitos turísticos a la hora de organizar sus viajes, debido a motivos como falta de dominio de las nuevas tecnologías, relaciones de confianza con sus agencias, precios especiales, etc. No será hasta

que se cree un equivalente a las agencias de viajes tradicionales en Internet que sea accesible de verdad a todo el mundo cuando realmente el porcentaje tienda a un valor próximo al 100 %.

1.2. El mercado de aplicaciones y *Web* turísticas

Internet está plagado de infinitas posibilidades. Existen una gran cantidad de páginas *web* que cubren todo tipo de temas. Es un espacio que ofrece unas posibilidades inimaginables.

Afortunadamente, existen muchísimas opciones de páginas y aplicaciones turísticas cuyo número sigue en aumento cada día. Esto implica una ventaja clara para el cliente: Mayor oferta, mejores promociones. Cada página o aplicación ofrece una serie de ventajas con el objetivo de atraer al usuario y que se decante por ellos. Por tanto, ahora más que nunca, podemos disfrutar de cientos de herramientas que facilitan la tarea de organizar un viaje.

A nivel nacional, podemos listar las principales *webs* y aplicaciones del 2019 en función de su cuota de uso (número de usuarios que han usado la aplicación [4]):

Aplicación	**Porcentaje**
Booking	39,7
Airbnb	13,7
Mi Nube	13,3
BlaBlaCar	12,1
TripAdvisor	10,4
Ryanair	9,9
Skyscanner	6,2
Goaz	6,1
Iberia	5,9
eDreams	5

Tabla 1.1: Procentaje de Usuarios por Aplicación en 2019

En la tabla 1.1 puede apreciarse que Booking, la aplicación / servicio *web* de reserva de alojamiento se mantiene como la más usada en España con una diferencia del 26 por ciento respecto a su inmediato perseguidor. Sin duda, está muy establecida entre los clientes, que la establecen como la favorita acorde a los datos. Nuevos servicios como Airbnb se han ganado un puesto en el mismo, debido a sus características, alzándose entre sus contrincantes. Por otro lado, aplicaciones / *webs* conocidas como TripAdvisor han sufrido caídas debido al auge de sus nuevos rivales, ya que la novedad siempre atrae. Además, podemos apreciar que los servicios de aerolíneas poseen una cuota bastante alta, siendo Ryanair e Iberia los máximos exponentes.

Resulta interesante comprobar cómo ha ido evolucionando el mercado de aplicaciones para entender la importancia de las mismas hoy en día. En la figura 1.2 se presenta una gráfica que representa el tráfico total de descarga de aplicaciones en todo el mundo.

Comenzando en el año 2016, se ve que el total de descargas de aplicaciones fue de 140,68 miles de millones (Billones en sistema americano). En tan solo dos años dicha cifra aumentó hasta los 192.45 miles de millones, lo que implica una tasa de crecimiento del 36 %. En el año 2019, se alcanzó 204 miles de millones de descargas, logrando mejorar la tasa un 6 %. Finalmente, en 2020 se lograrían las 218 miles de millones de descargas mejorando en otro 6 % la tasa de crecimiento.

Teniendo en cuenta esta información, nos damos cuenta del increíble potencial que posee el mercado de aplicaciones. Su crecimiento lineal está lejos de detenerse, por lo que seguirá aumentando el número de nuevas aplicaciones y, consecuentemente, el mercado potencial. Es una

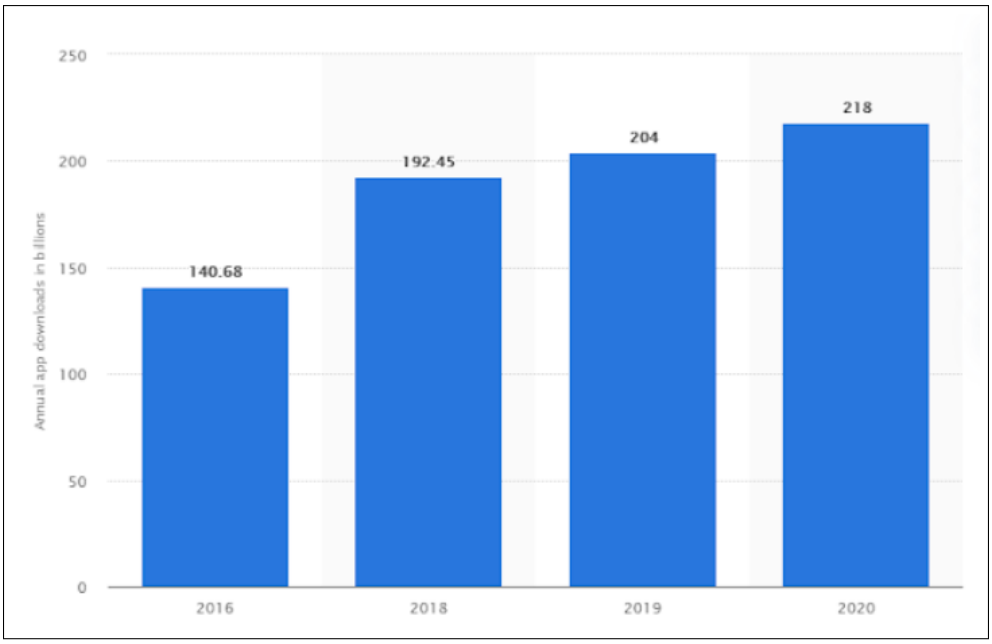


Figura 1.2: Evolución del número de Descargas de Aplicaciones en el Mundo.

oportunidad que el sector turístico no debe desaprovechar. Es por ello que **Pack2Fly** pretende aprovechar esta situación para alcanzar al mayor número de usuarios posibles, uniéndose a la imparable invasión de las nuevas tecnologías para conectar de una manera nunca vista al mundo entero a través del turismo.

Capítulo 2

Planteamiento de la Solución

En este capítulo, se va a profundizar en todos los aspectos referidos a la planificación de todos los aspectos necesarios para la correcta realización del proyecto. Es decir, el objetivo final del mismo será tener completamente claro que se pretende realizar y de qué manera. Para ello, lógicamente, será necesario llevar a cabo una serie de pasos que, en su conjunto, definirán el diseño de la aplicación.

2.1. Requisitos

El primer paso a la hora de realizar el planteamiento de la solución es establecer los requisitos a tener en cuenta en la aplicación. Estos serán:

Técnicos

1. Deberá contar con filtros que permitan refinar la búsqueda. Estos filtros serán:
    - Rango de Precio
    - Zonas
    - Hora de salida / llegada
    - Puntuación del Paquete Global
    - Puntuación de elementos del paquete: vuelo, alojamiento o actividades
2. Contará con unos campos iniciales, que serán, siguiendo el diseño de otros sitios web, los siguientes:
    - Lugar de estancia
    - Fecha de ida
    - Fecha de vuelta
    - Número de Personas (distinguiendo entre niños o adultos, en función de una edad a concretar).
3. La aplicación será diseñada en inglés para lograr una mayor difusión.
4. La aplicación ofrecerá al usuario el resultado de su búsqueda en forma de lista interactiva, de tal manera que el usuario podrá seleccionar un resultado para analizarlo mejor.
5. Cada paquete contendrá la información sobre el/los vuelo/s, el/los alojamiento/s y las actividades a realizar.

6. Permitirá ordenar los paquetes en función de los siguientes criterios:
    - Más barato
    - Mejor calidad-precio
    - Más caro
    - Mejor Puntuación
7. El usuario podrá iniciar sesión con sus credenciales, que serán usuario y contraseña.
8. Para registrarse, el usuario contará con un formulario donde introducirá sus datos.
9. La aplicación contará con un espacio dedicado al usuario, donde dispondrá de su información personal y de los paquetes que ha guardado.
10. La aplicación contará con una serie de recomendaciones de paquetes: paquetes recomendados y *top paquetes*.
11. La aplicación dispondrá de una página inicial de inicio de sesión, que además permitirá al usuario hacer uso de la misma sin iniciar sesión.
12. En la página de contacto, el usuario dispondrá de un formulario de contacto y de distintas opciones interactivas para acceder a las redes sociales de la aplicación.

No Técnicos

1. La aplicación debe ser fácil de usar, debido a que el mercado potencial está constituido por personas que no tienen porqué dominar las TIC.
2. La aplicación deberá contar con elementos de seguridad para garantizar la seguridad de los datos.
3. Es conveniente que la aplicación cuente con un diseño bien estructurado, para facilitar su flujo y la traza (y consecuente reparación) de errores.
4. La aplicación deberá, dentro de lo posible, mostrar los resultados en un periodo de tiempo igual o menor a 3 segundos.
5. La aplicación deberá estar disponible, al menos en su versión inicial, para *Google Chrome* y, si fuera posible, para el resto de navegadores, siendo compatible con sus versiones actuales.
6. Desplegable en versión web.
7. Interfaz sencilla.
8. Uso de lenguajes actuales para favorecer el rendimiento y mantenimiento.

2.2. Flujo de Usuario y Modelado

A la hora de planificar el comportamiento de la aplicación, resulta conveniente realizar una serie de diagramas y bocetos que nos permitan, en resumen, obtener una idea del flujo de datos y de usuario que nos ayudará enormemente a la hora de diseñarla.

El diagrama de flujo de usuario muestra las interacciones del usuario con la aplicación.

La página principal de la aplicación será el propio buscador. Así mismo, esta pantalla será la primera que vean los usuarios al entrar en la misma.

Si comenzamos por la parte superior del diagrama, encontraremos el camino en el que el usuario realiza una búsqueda. En primer lugar, introducirá el destino. A continuación, las fechas de ida y vuelta, el número de adultos y el número de niños a incluir. Pulsará buscar y se redirigirá a la página de resultados. En esta página, el usuario podrá seleccionar un filtro o realizar una ordenación, ofreciendo nuevos resultados o mostrándolos con un posicionamiento distinto, respectivamente. El usuario seleccionará un paquete. Si el paquete está guardado, mostrará el botón de eliminar, mientras que si no lo está, mostrará el botón de guardar junto con la información resumen. Si el usuario decide pulsar Erase, se eliminará el paquete de sus paquetes guardados. Si el usuario acciona el botón *save*, se comprobará si ya está guardado. En caso negativo, se guardará. Si por el contrario, ya está guardado, se mostrará un mensaje. Si el usuario pulsa *check*, será redirigido a la página de información del paquete. En esta página, el usuario podrá seleccionar información sobre el transporte, alojamiento, actividades o precios. En caso de elegirlo, se mostrará un resumen de el/los componentes. El usuario podrá seleccionar alguno de ellos, siendo redirigido a la página que ofrece dicha opción. Por último, el usuario podrá accionar el botón *get*, en cuyo caso será redirigido a la página correspondiente.

El segundo camino será el que corresponde a la situación en la que el usuario ha seleccionado el botón de *Top Packs*. En este caso, simplemente, se mostrará la página de resultados, con las acciones anteriormente expuestas.

El tercer camino se asemeja al anterior, y se despliega cuando el usuario selecciona el botón de *Recommended Packs*. En este caso, será redirigido a la página de recomendaciones. El usuario elegirá una temática y será redirigido a la página de resultados.

El cuarto camino contempla una nueva posibilidad. El usuario selecciona el botón de *Based on Your Likes*. Se comprobará si el usuario tiene una sesión iniciada. En caso afirmativo, se mostrarán los resultados en la pantalla conveniente. En caso negativo, se redirigirá al usuario a la página de inicio de sesión. Aquí el usuario podrá seleccionar varias opciones:

1. Selecciona no tengo cuenta: Será redirigido a la página de creación de cuenta. Aquí, el usuario introducirá el nombre de usuario, el email (repetidamente) y la contraseña (repetidamente). Pulsará a continuación crear cuenta. Se comprobará que todos los campos están rellenados y son válidos. En caso negativo, se informará al usuario y se le mantendrá en la página. En caso positivo, se le enviará un mensaje de verificación.

2. Selecciona no recuerdo la contraseña: El usuario introducirá un correo de recuperación. Si el correo está registrado, se le enviará un mensaje con su contraseña a ese correo introducido. En caso negativo, se le notificará al usuario.

3. El usuario introduce nombre de usuario y contraseña. Se comprobará que ambos campos son correctos. En caso afirmativo, se iniciará la sesión y se le redirigirá a la página anterior. En caso negativo, se le mostrará un mensaje de error.

El último camino envuelve al menú. El usuario seleccionará el icono de menú, desplegádose el mismo con las opciones que contiene.

1. Si el usuario pincha en buscador, será redirigido al mismo.

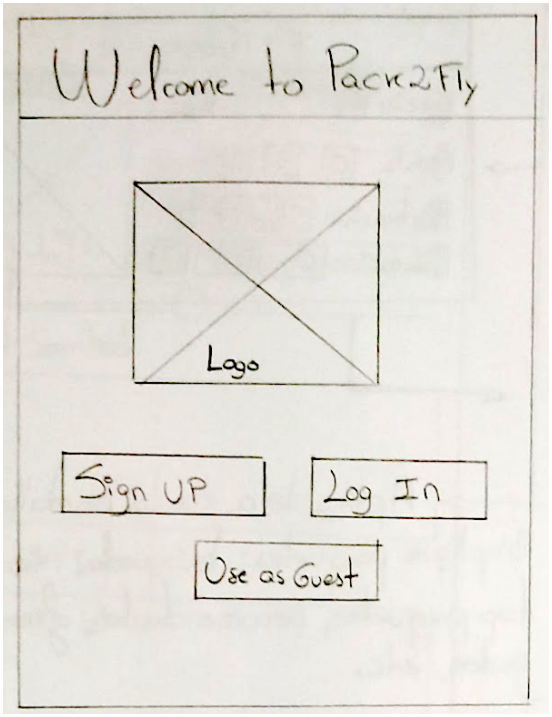
2. Si el usuario elige la opción de mi cuenta, se comprobará que esté logeado. En caso negativo, se redirigirá a la página de inicio de sesión. En caso afirmativo, será redirigido a la página de cuenta, donde podrá seleccionar su información personal, sus paquetes (siendo redirigido a la página de resultados) o la edición, habilitando la modificación de los campos.

3. Si el usuario elige sus paquetes, será redirigido a la página de resultados con sus paquetes guardados.

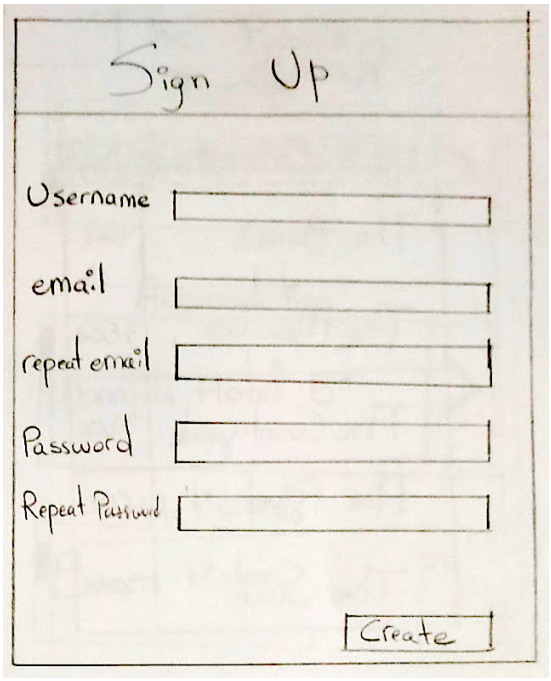
4. Si elige contacto, será redirigido a la página de contacto, que presenta 2 opciones: Introducir el título y mensaje del correo, y pulsar enviar, comprobando los campos y enviando el mensaje en caso de que todo esté correcto. En caso de fallo, se le enviará una notificación. Por otro lado, podrá seleccionar una red social, siendo redirigido a la misma.
5. Si el usuario selecciona Sobre Nosotros, será redirigido a la página pertinente.
6. Si el usuario selecciona desconectar, se comprobará si tiene una sesión iniciada. En caso afirmativo, se cerrará su sesión y se le notificará. Si no lo está, mostrará un mensaje de notificación al usuario.

Con todo esto, se ha analizado el flujo de un usuario que hace uso de la aplicación, abarcando todos los posibles caminos que pudiera tomar. Esto nos permitirá profundizar en la fase de diseño, cuyo siguiente paso consistirá en la creación de bocetos. El concepto de boceto refiere al esquema o el proyecto que sirve de bosquejo para cualquier obra. Se trata de una guía que permite volcar y exhibir sobre un papel una idea general antes de arribar al trabajo que arrojará un resultado final [5]. Así, se han planteado los siguientes bocetos para las distintas páginas que conforman la aplicación.

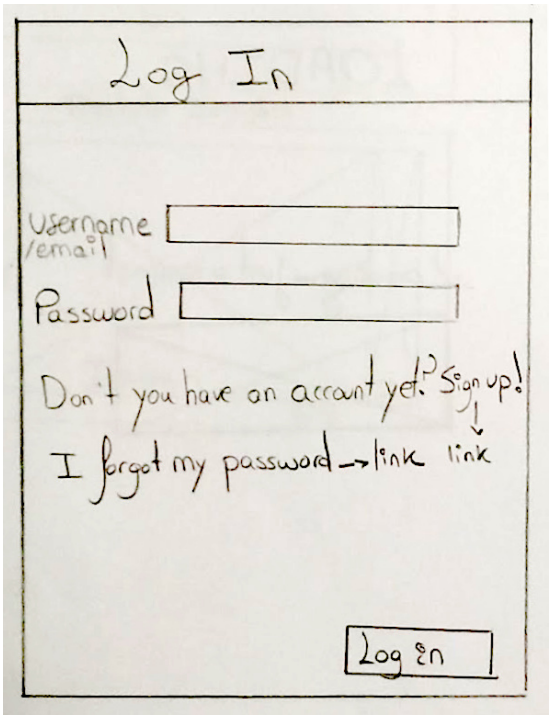
En los bocetos de la figura 2.1, se incluyen las páginas primaria, la de crear cuenta, la de inicio de sesión y el buscador.



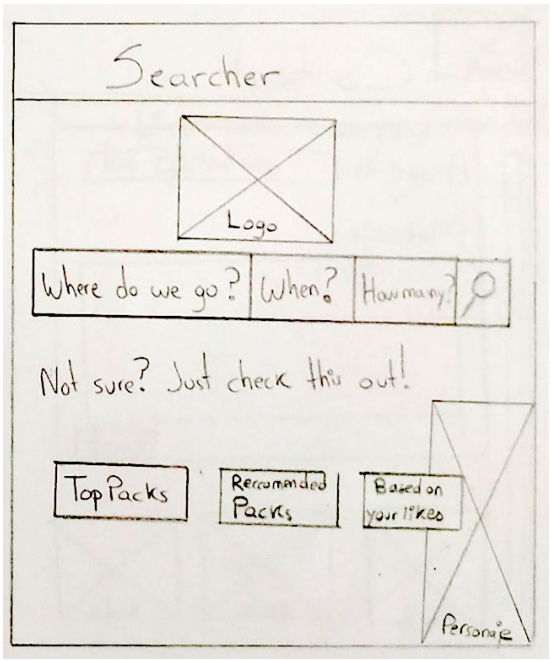
(a) Welcome



(b) Sign up



(c) Log in



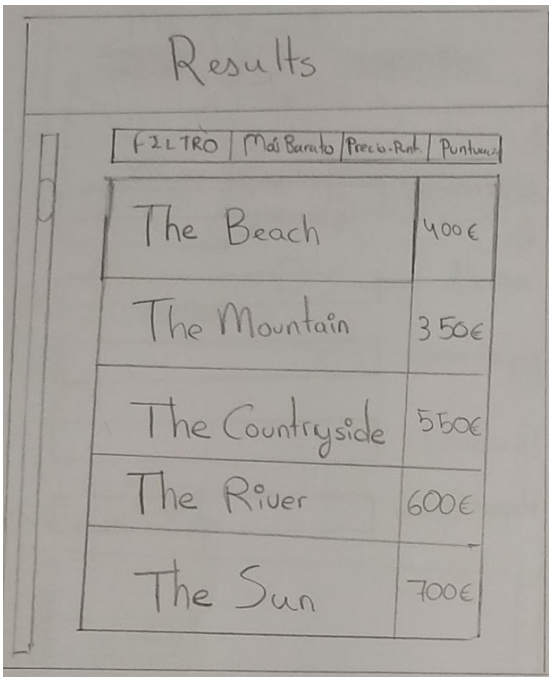
(d) Search

Figura 2.1: Boceto 1

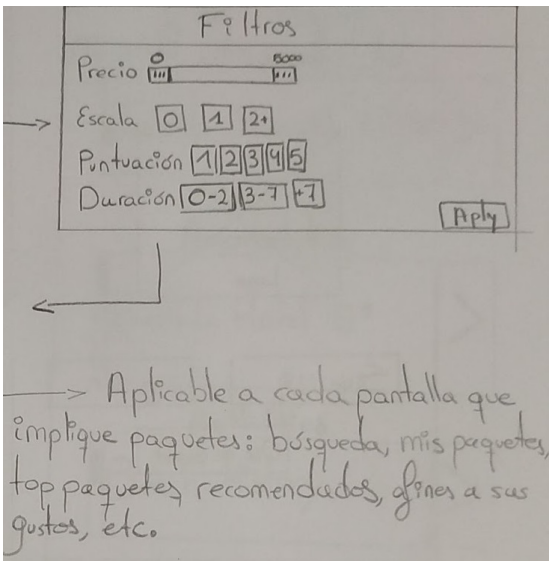
- **Página Primaria:** figura 2.1(a). Esta página será la primera que se mostrará al usuario. En ella, se le presentan 3 opciones, que serán iniciar sesión, crear cuenta o usar como invitado. Mostrará además el logo de la aplicación.
- **Página de Crear Cuenta:** figura 2.1(b). En esta página, el usuario dispondrá de los campos necesarios que deberá rellenar. Son nombre, *email* y contraseña, debiendo repetir estas dos últimas por motivos de seguridad.

- **Página de Inicio de Sesión:** figura 2.1(c). Esta página permitirá al usuario identificarse en la aplicación mediante su nombre o *email* más su contraseña. Además, dispondrá de dos mensajes. Uno para redirigirlo a la página de creación de cuenta, si no tuviera, y otro que lo redirigirá a una página adicional que le permitirá recuperar su contraseña.
- **Página de Buscador:** figura 2.1(d). Está página es la principal de la aplicación. En ella, se dispondrá en primer lugar de 3 campos que el usuario rellenará para realizar la búsqueda: destino, fechas del viaje y la cantidad de personas que son. Además, se colocará en esta pantalla 3 opciones adicionales de paquetes destacados: *top* paquetes, paquetes recomendados por el equipo y paquetes relacionados con los gustos del usuario. Este último campo será visible solo si se ha iniciado sesión.

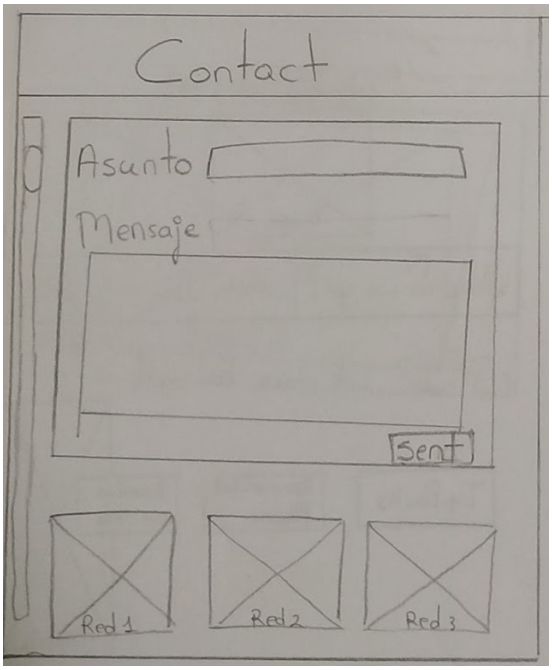
En el boceto 2.2, se incluyen las páginas de paquetes, los filtros, la de contactos y la de carga.



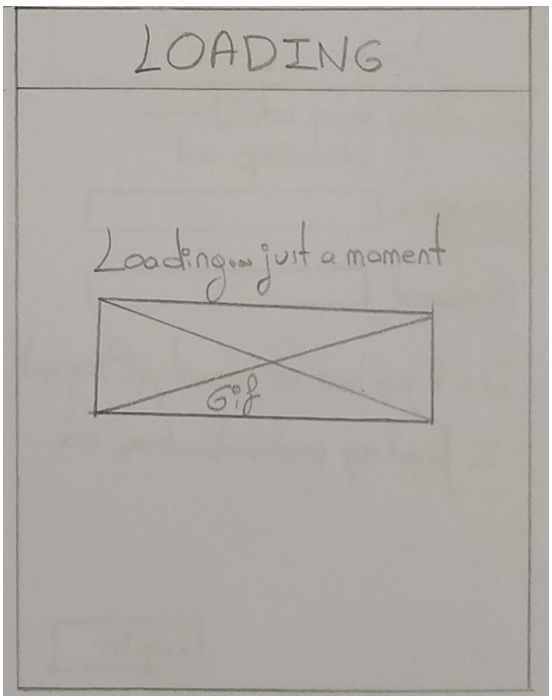
(a) Results



(b) Filters



(c) Contact



(d) Loading

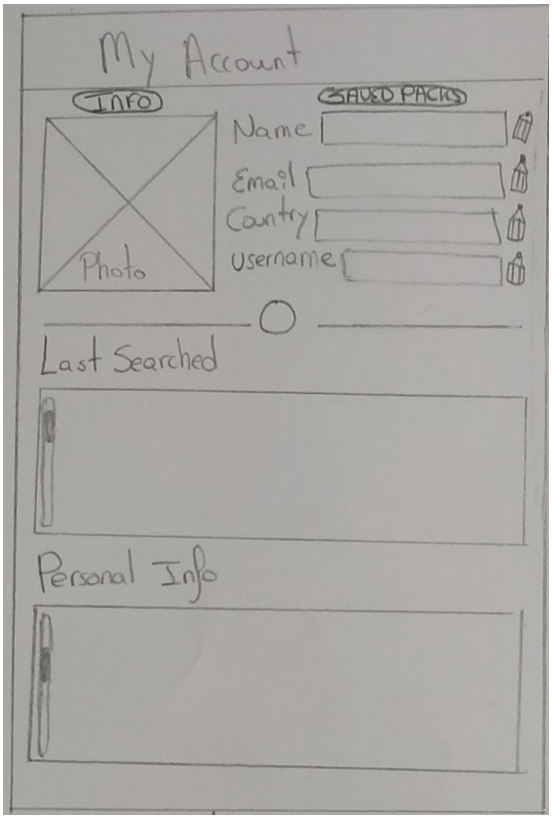
Figura 2.2: Boceto 2

- **Página de Paquetes:** Esta pantalla se usará como página para todas las acciones que muestren paquetes como resultado. En el ejemplo del boceto se ha seleccionado la acción de mostrar resultados de la búsqueda. En primer lugar, se mostrarán las distintas opciones a aplicar: filtros, ordenar por precio descendente, por calidad-precio o por puntuación. A continuación, se mostrarán los distintos paquetes construidos como una lista interactiva junto con su precio, de tal manera que el usuario pueda seleccionar un paquete y se despliegue su información.
- **Página de Filtro:** Esta página se mostrará cuando se seleccionen los filtros en la anterior

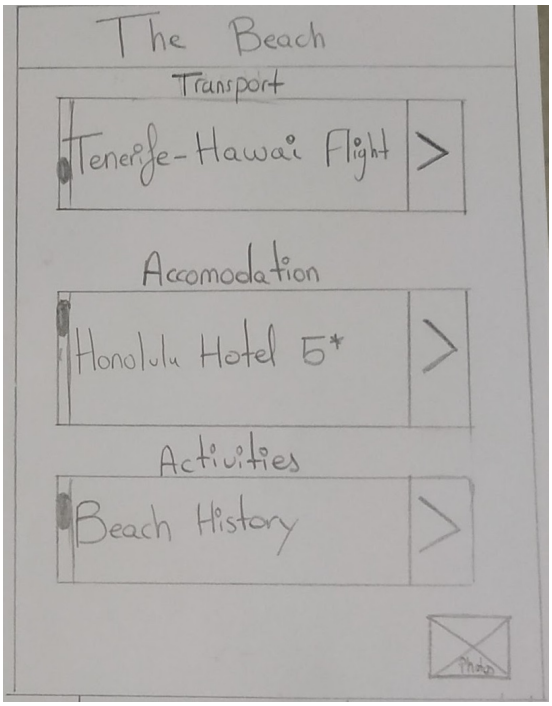
pantalla. Incluirá los distintos filtros que se contemplen: Precio, Escalas, Puntuación, Duración, etc. El botón Apply ejecutará los filtros, redirigiendo a la página de resultados con la filtración realizada.

- **Página de Contacto:** En esta página, se mostrará un formulario en primer lugar donde el usuario podrá rellenar un mensaje que desee enviar al equipo de soporte. Además, justo debajo, se mostrarán todas las opciones de contacto que están disponibles para el usuario (redes sociales, por ejemplo).
- **Página de Carga:** Esta pantalla se mostrará cuando se realice una búsqueda de paquetes. En ella, se ejecutará un gif con una imagen de carga.

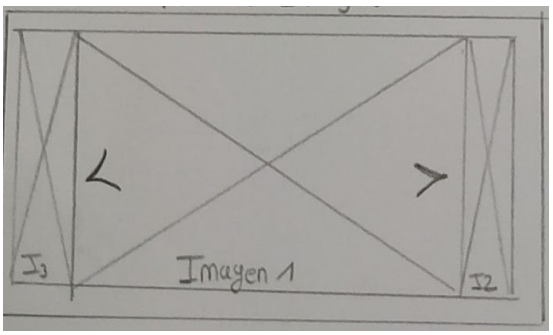
En el último boceto 2.3, se incluyen las páginas de Mi cuenta, de Paquete y de Imágenes.



(a) Results



(b) Filters



(c) Contact

Figura 2.3: Boceto 3

- **Página de Mi Cuenta:** En esta página, se dispondrá de dos botones. Uno permitirá ver la información personal del usuario, mientras que el otro presentará los paquetes guardados

por el usuario. Se mostrará una imagen de usuario, así como nombre, *email*, país, nombre de usuario y cualquier otra información de carácter personal a discutir más adelante. Además, se dispondrá de una serie de paneles que mostrarán información relacionada con el usuario, como puede ser las últimas búsquedas, información personal bibliográfica, métodos de pago almacenados, etc.

- **Página de Paquete:** Esta página mostrará la información en forma de paneles interactivos relacionada con el paquete: transporte, alojamiento, actividades, precios, etc. También se incluirán botones para guardar el paquete, adquirirlo y para consultar fotos relacionadas.
- **Página de Imágenes:** Esta página mostrará las imágenes relacionadas con los paquetes si se selecciona la opción.

Estos diseños "previos" nos serán de mucha utilidad a la hora de implementar un prototipo inicial que nos permita guiar el desarrollo de la aplicación.

2.3. Diagramas de Casos de Uso

Un diagrama de casos de uso es una herramienta gráfica que nos permite identificar las acciones que puede realizar cada tipo de usuario, denominado actor, en el sistema. Ayudará a comprender mejor las limitaciones que tendrán los distintos usuarios de la aplicación de cara a la implementación de la misma.

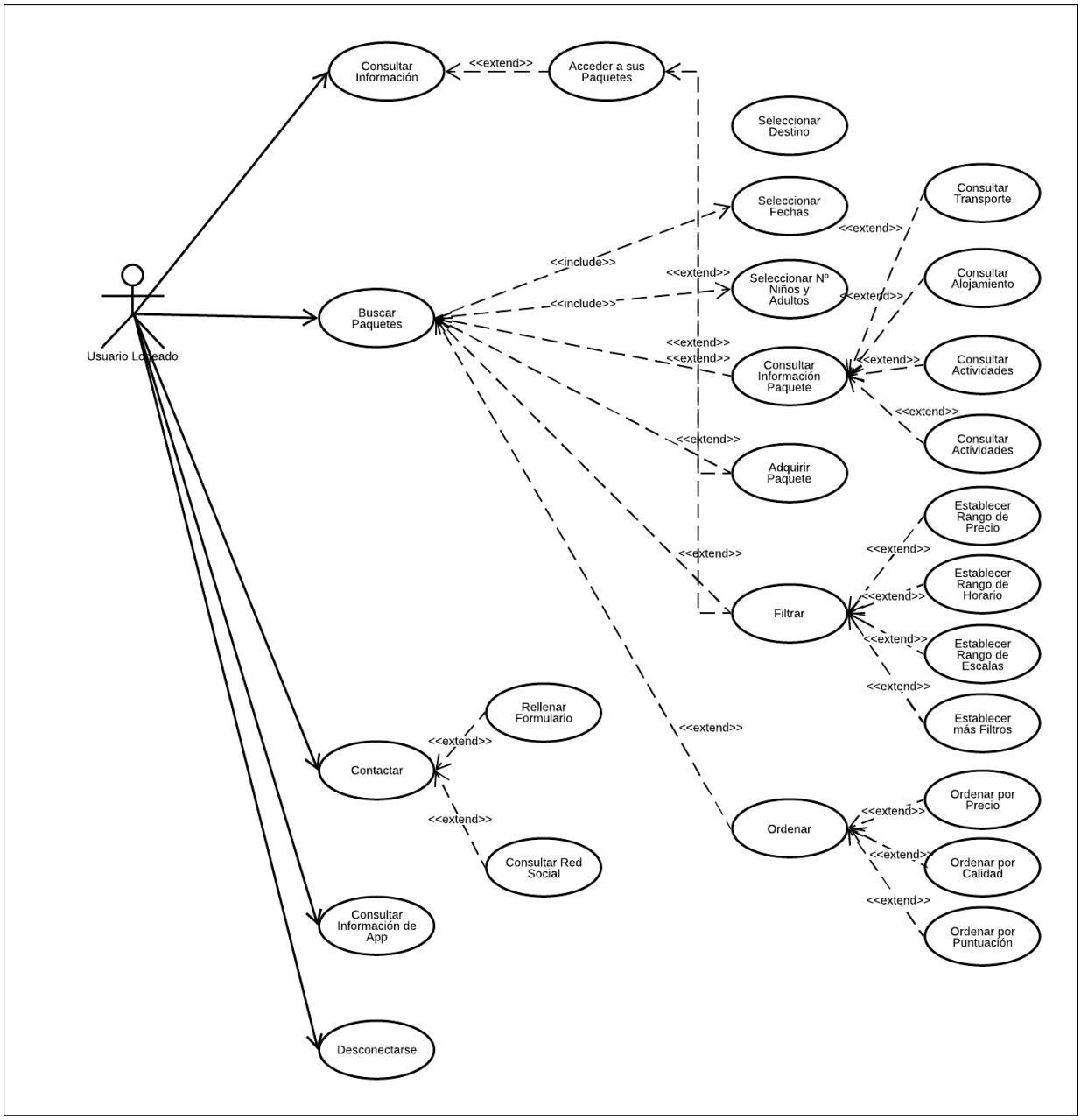


Figura 2.4: Diagrama de Caso de Uso para Usuario Logeado

En el diagrama 2.4 se incluyen las distintas funcionalidades asociadas a un usuario que ha iniciado sesión. Presenta muchas similitudes con el usuario sin sesión iniciada, con algunas diferencias. En primer lugar, este tipo de usuario puede consultar su información personal, pudiendo acceder a sus paquetes guardados. Además, lógicamente, puede desconectarse, cerrando su sesión.

Por motivos de espacio, se ha usado este diagrama como ejemplo para describir este tipo y su funcionalidad. Los demás diagramas se contemplan en el anexo correspondiente.

2.4. Prototipado

A la hora de realizar el prototipado, se ha optado por elegir la herramienta justinmind [6], debido varios factores:

- Es gratuita.
- Es sencilla de usar.
- Ofrece numerosas opciones para elaborar los prototipos, logrando un aspecto profesional.

El objetivo principal de un prototipo es, en resumidas cuentas, ofrecer una visión general del funcionamiento de la aplicación. Esto no significa que sea una versión funcional. El prototipo no es más que una simple simulación. Nos permitirá crear una idea de cómo actuará la aplicación en cada caso. Además, servirá para poder refinar nuestros diseños para lograr así un sistema más eficiente, elegante, profesional, etc, basándonos en retroalimentación.

Este tipo de herramientas nos resulta muy útil, ya que no es necesario realizar una versión funcional real para poder analizar la aplicación. Por ejemplo, en un proyecto más complejo, puede ocurrir que hayamos finalizado la aplicación cuando, al mostrarla al público, nos comenten que su interfaz es muy poco atractiva al uso. Lógicamente, los costes en tiempo y recursos aumentarán bastante si tuviéramos que implementar de nuevo una nueva versión. Es por ello que el prototipado es una solución tan buena. Podemos realizar múltiples estudios, refinando poco a poco el prototipo hasta lograr una versión que nos resulte satisfactoria, implementando la aplicación real basándonos en el mismo.

Aplicando esta herramienta al proyecto, se ha procedido a realizar un prototipo para móvil y para PC. En ambos se ha tenido en cuenta los requisitos anteriormente citados [véase 2.1] y el principio básico de sencillez en el diseño. El diseño final para dispositivos móviles se ha incluido en esta memoria [véase figuras B.1, B.2 y B.3]

2.5. Definición de Escenarios

Un escenario es una descripción parcial del comportamiento de la aplicación en un momento específico. La utilización de escenarios implica identificar distintas situaciones y describir la acción a llevar a cabo [7].

De esta manera, en la aplicación podemos establecer una serie de situaciones que nos permitan analizar con detalle el comportamiento específico que tendrá la aplicación frente a cada uno de ellas.

Situación 1: Buscador

1. **Escenario 1: Usuario Realiza una búsqueda**

   En este caso, el usuario introduce el destino, la fecha de ida, la fecha de vuelta, el número de adultos y el número de niños, pulsando el botón buscar. Se comprobará que todos los campos son correctos y están bien rellenados. Aquí se plantean varios escenarios

   - **Escenario 1.1: Destino Desconocido / No rellenado.** El usuario ha introducido un destino que no se reconoce en el sistema o no ha introducido información en el campo. En este caso, se le muestra un mensaje que informa al usuario de esta situación.
   - **Escenario 1.2 No introduce Fechas.** El usuario no introduce la fecha de ida y/o de vuelta. En este caso, se le muestra un mensaje que informa al usuario de esta situación.
   - **Escenario 1.3 No introduce Número de Personas.** El usuario no introduce el número de adultos y/o niños a incluir. En este caso, se le muestra un mensaje que informa al usuario de esta situación.

2. **Escenario 2: Usuario pulsa en Top Packs** El usuario es redirigido a la página de "Top Packs" con los resultados de los paquetes más vistos/reservados/etc.

3. **Escenario 3: Usuario pulsa Recommended Packs** El usuario es redirigido a la página de "Recommended Packs" con los resultados de los paquetes de moda.

4. **Escenario 4: Usuario pulsa Based on Your Likes** Se comprueba que el usuario esté logeado en la aplicación. Se pueden dar 2 posibles escenarios

    - **Escenario 4.1 Usuario Logeado.** El usuario es redirigido a la página de resultados con los paquetes que la aplicación le recomienda en función de los paquetes que ha guardado.
    - **Escenario 4.2 Usuario No Logeado.** El usuario es redirigido a la página de inicio de sesión. Se plantean 2 posibilidades:
        - **Escenario 4.2.1 Usuario decide iniciar sesión.** El usuario inicia sesión, siendo redirigido de nuevo a la página de los paquetes recomendados en función de sus gustos.
        - **Escenario 4.2.2 Usuario no inicia sesión.** El usuario decide no iniciar sesión en la página conveniente, volviendo a la página de búsqueda.

Situación 2: Página de Login

1. **Escenario 1: Usuario intenta iniciar sesión**
   El usuario introduce nombre de usuario y contraseña, pulsando iniciar sesión. En este caso, pueden darse 2 posibilidades:

    - **Escenario 1.1 Nombre de Usuario no válido.** El usuario introduce un nombre de usuario no reconocido en el sistema. Se le envía un mensaje de notificación.
    - **Escenario 1.2 Contraseña no válida.** El usuario introduce una contraseña no válida con respecto al correo introducido. Se le envía un mensaje de notificación.

2. **Escenario 2: Usuario pulsa en No Tengo Cuenta** El usuario pulsa sobre No tengo cuenta. Es redirigido a la página de Registro.

3. **Escenario 3: Usuario pulsa en Olvidé mi contraseña.** El usuario pulsa sobre olvidé mi contraseña. Se le envía un mensaje de notificación solicitando un correo de recuperación. Se busca ese correo en la base de datos y se enviará la contraseña asociada a ese correo.

Situación 3: Página de *Sign Up*

1. **Escenario 1: Usuario intenta crear cuenta** El usuario pulsa el botón de Crear Cuenta. Se pueden dar una serie de opciones:

    - **Escenario 1.1 Nombre de usuario no válido / no rellenado**: El usuario es notificado sobre esta situación.
    - **Escenario 1.2 Contraseña no válida / no rellenada**: El usuario es notificado sobre esta situación.
    - **Escenario 1.3 Repetición de Contraseña no válida / no rellenada**: El usuario es notificado sobre esta situación.
    - **Escenario 1.4 *Email* no válido / no rellenado**: El usuario es notificado sobre esta situación.
    - **Escenario 1.5 Repetición del *Email* no válido / no rellenado**: El usuario es notificado sobre esta situación

Situación 4: Página de Menú

1. **Escenario 1: Usuario pulsa en Buscador.**
   Es redirigido a la página de Buscador
2. **Escenario 2: Usuario pulsa en Mi Cuenta.**
   Se comprueba que el usuario tenga la sesión iniciada. Se pueden dar varias situaciones:
   - **Escenario 2.1 Usuario con Sesión Iniciada**: El usuario es redirigido a la página de su cuenta.
   - **Escenario 2.2 Usuario sin sesión iniciada**: El usuario es redirigido a la página de inicio de sesión. Se pueden dar 2 posibilidades:
     - **Escenario 2.2.1 Usuario sin sesión iniciada inicia sesión**: Es redirigido a su cuenta.
     - **Escenario 2.2.2 Usuario sin sesión iniciada no inicia sesión**: Vuelve al menú.
3. **Escenario 3: Usuario pulsa en Mis Paquetes** Se comprueba que el usuario tiene la sesión iniciada. Se pueden dar varias situaciones:
   - **Escenario 3.1 Usuario con sesión iniciada**: El usuario es redirigido a la página de sus paquetes.
   - **Escenario 3.2 Usuario sin sesión iniciada**: El usuario es redirigido a la página de inicio de sesión. Se pueden dar 2 posibilidades:
     - **Escenario 3.2.1 Usuario sin sesión iniciada inicia sesión**: Es redirigido a sus paquetes.
     - **Escenario 3.2.2 Usuario sin sesión iniciada no inicia sesión**: Vuelve al menú.
4. **Escenario 4: Usuario pulsa en Contacto** El usuario es redirigido a la página de contacto.
5. **Escenario 5: Usuario pulsa en Sobre Nosotros** El usuario es redirigido a la página de Sobre Nosotros.
6. **Escenario 6: Usuario pulsa en Desconectar** El usuario es notificado convenientemente y es redirigido al menú de nuevo.

Situación 5: Página de Resultados de Paquetes

1. **Escenario 1: Usuario elige un paquete.** El usuario es redirigido a la página de información del paquete.
2. **Escenario 2: Usuario filtra los resultados.** El usuario pulsa el botón de filtro. Se despliega la pantalla de filtros.
3. **Escenario 3: Usuario pulsa en pulsa ordenación por precio.** El usuario pulsa el botón de ordenación por precio. Pulsa una vez y los resultados aparecen de mayor a menor precio. Pulsa otra vez y se ordena de menor a mayor precio.
4. **Escenario 4: Usuario pulsa en pulsa ordenación por calidad.** El usuario pulsa el botón de ordenación por calidad. Pulsa una vez y los resultados aparecen de mayor a menor puntuación de calidad. Pulsa otra vez y se ordena de menor a menor a mayor.
5. **Escenario 5: Usuario pulsa en pulsa ordenación por puntuación.** El usuario pulsa el botón de ordenación por puntuación. Pulsa una vez y los resultados aparecen de mayor a menor puntuación. Pulsa otra vez y se ordena de menor a menor a mayor.

Situación 6: Página de Contacto

1. **Escenario 1: Usuario Rellena Formulario.** El usuario pulsa en enviar en el formulario de contacto. Se pueden dar 2 situaciones:
    - **Escenario 1.1: Titulo no válido / no rellenado**: Se le envía un mensaje de notificación.
    - **Escenario 1.2: Mensaje no válido / no rellenado**: Se le envía un mensaje de notificación.
2. **Escenario 2: Usuario accede a alguna red social** El usuario selecciona una red social de las habilitadas para el contacto. Es redirigido a la red social en cuestión.

Situación 7: Página de Filtro

1. **Escenario 1: Usuario aplica filtro**.

   El usuario pulsa en el botón de filtrar. Se aplican los filtros a los resultados, siendo redirigido a la página correspondiente.

Situación 8: Página de Mi Cuenta

1. **Escenario 1: Usuario accede a su información personal.** El usuario pulsa en su información personal. Se despliega la misma en la pantalla. El usuario ve su nombre, apellido, país, etc. Puede ver los paquetes vistos por última vez y los datos relacionados con su perfil. En este escenario puede darse un sub-escenario adicional:
    - **-Escenario 1.1: Usuario pulsa editar.** El usuario pulsa el botón de editar. Su información personal se habilita en modo edición. Pulsa intro tras introducir cada campo para guardar su contenido.
2. **Escenario 2: Usuario accede a sus paquetes guardados.** El usuario pulsa en sus paquetes guardados. Es redirigido a la página correspondiente de resultados.

Situación 9: Página de Información de Paquete

1. **Escenario 1: El paquete ya ha sido guardado.** Se muestran los botones de comprobar y de borrar de mis paquetes. En este caso, se dan 2 posibilidades:
    - **Escenario 1.1: Usuario pulsa en comprobar.** Es redirigido a la siguiente pantalla de información del paquete.
    - **Escenario 1.2: Usuario pulsa en eliminar paquete.** El paquete es eliminado de los paquetes del usuario.
2. **Escenario 2: El paquete no ha sido guardado** Se muestran los botones de comprobar y de guardar en mis paquetes. En este caso, se dan 2 posibilidades:
    - **Escenario 2.1: Usuario pulsa en comprobar.**¡ Es redirigido a la siguiente pantalla de información del paquete.
    - **Escenario 2.2: Usuario pulsa en guardar.** El paquete es guardado en los paquetes del usuario.

Situación 10: Página de Información de Paquete II

1. **Escenario 1: Usuario pulsa en Transporte.** El usuario pulsa en la opción del transporte. Es redirigido a la página concreta de información del transporte para el paquete.

2. **Escenario 2: Usuario pulsa en Alojamiento.** El usuario pulsa en la opción del alojamiento. Es redirigido a la página concreta de información del alojamiento para el paquete.
3. **Escenario 3: Usuario pulsa en Actividades.** El usuario pulsa en la opción de actividades. Es redirigido a la página concreta de información de actividades para el paquete.
4. **Escenario 4: Usuario pulsa en Precio.** El usuario pulsa en la opción de precio. Es redirigido a la página concreta de información del precio para el paquete.
5. **Escenario 5: Usuario pulsa en Guardar.** Se comprueba que el paquete ya está guardado. Se envía un mensaje y, en su caso, se guarda en los paquetes del usuario.
6. **Escenario 6: Usuario pulsa en Get** El usuario es redirigido a la/s página/s correspondientes.

Situación 11: Página de Información de Paquete Concreto III

1. **Escenario 1: Usuario pulsa en opción de transporte concreto.** El usuario es redirigido a la página pertinente para más información sobre la opción.

Situación 12: Página de Recomendación

1. **Escenario 1: Usuario selecciona una opción.** Es redirigido a la página de resultados filtrados por la temática seleccionada.5

Situación 13: Página de Sobre Nosotros

1. **Escenario 1: Usuario selecciona contacto.** Es redirigido a la página de contacto.

Situación Común 1: Selecciona Menú
El usuario selecciona el menú en cualquier momento. Es redirigido a la pantalla del menú.

Situación Común 2: Vuelve atrás
El usuario realiza la acción de volver a la página anterior, siendo redirigida a la misma.

Situación Común 3: Cierra la aplicación
El usuario cierra la aplicación. Se realiza una comprobación de integridad y se procede a su finalización.

2.6. Caracterización de Entidades

Una entidad es un objeto concreto o abstracto que presenta interés para el sistema y sobre el que se recoge información la cual va a ser representada en un sistema de base de datos. La mayoría de las entidades modelan objetos o eventos del mundo real, por ejemplo, clientes, productos o llamadas de pedidos [8].

De cara a la aplicación, la definición de entidades corresponde a definir y describir los objetos que son de interés para la misma. Así, se pueden identificar:

- Clientes: Son los usuarios que harán uso de la aplicación.
- Paquetes: Son los productos ofertados al cliente.

- Consulta: Mensaje enviado por el usuario
- Cuenta: Posee la información personal del cliente.
- Información Personal: Información propia de cada usuario.

Resulta muy útil crear un diagrama Entidad-Relación. Un modelo entidad-relación es una herramienta para el modelo de datos, la cual facilita la representación de entidades de una base de datos [9].

Este tipo de diagrama está constituido por una serie de elementos:

- **Entidades**
- **Relaciones:** Consiste en una colección, o conjunto, de relaciones de la misma naturaleza.
- **Atributos:** Los atributos son las características que definen o identifican a una entidad.
- **Cardinalidades:** La cardinalidad de la correspondencia indica el número de entidades con las que puede estar relacionada una entidad dada. Se distinguen:
  - Uno a Uno: (1:1) Un registro de una entidad A se relaciona con solo un registro en una entidad B.
  - Uno a Varios: (1:N) Un registro en una entidad en A se relaciona con cero o muchos registros en una entidad B. Pero los registros de B solamente se relacionan con un registro en A.
  - Varios a Uno: (N:1) Una entidad en A se relaciona exclusivamente con una entidad en B. Pero una entidad en B se puede relacionar con 0 o muchas entidades en A.
  - Varios a Varios: (N:M) Una entidad en A se puede relacionar con 0 o con muchas entidades en B y viceversa.

El diagrama implementado para la aplicación 2.5 cuenta con instancias propias de los elementos anteriores.

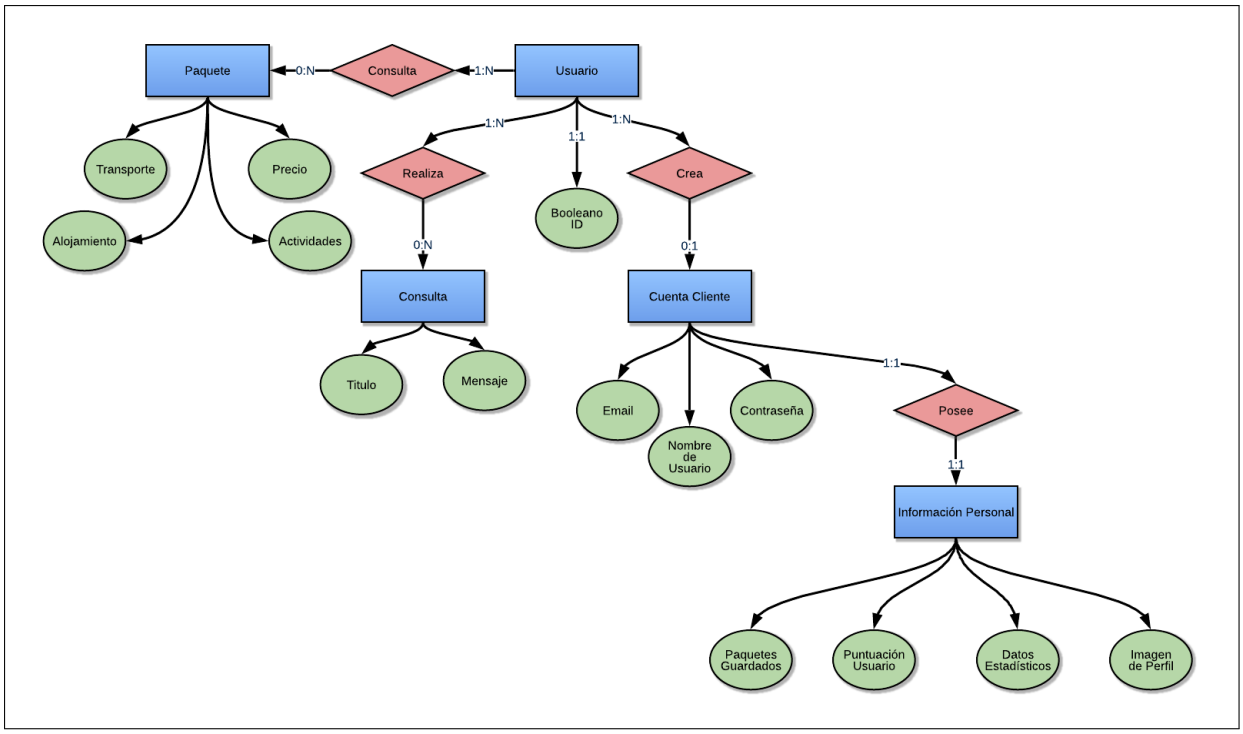


Figura 2.5: Modelo Entidad-Relación para la Aplicación

Comenzando por el usuario, en el sistema pueden existir una gran cantidad de ellos (N). El usuario puede crear una cuenta de cliente. Esta cuenta contará con *email*, nombre de usuario y contraseña. Además, la cuenta del cliente posee información personal, como paquetes guardados, puntuación del usuario, datos estadísticos e imagen de perfil. Además, el usuario dispone de un atributo del tipo booleano ID, que permite identificar si se trata de un usuario administrador o regular. El usuario puede realizar además una consulta, que posee un título y un mensaje. Por último, puede consultar todos los paquetes que quiera. Estos paquetes poseen como atributos el transporte, el alojamiento, el precio y las actividades.

2.7. Diseño de las Bases de Datos

Un modelo de base de datos muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se accede a ellos. [10]

En el apartado anterior [véase 2.6] se ha planteado un diagrama de entidad-relación que forma parte del modelo de datos. Permitirá acometer el siguiente paso: diseñar las bases de datos. Este paso es sumamente importante, ya que definirá la manera en la que se accede a la información.

Lógicamente, el sistema deberá contar con una base de datos propia para los usuarios existentes. El diseño para la misma se recoge en la tabla B.1 en el anexo de este documento.

El primero de los campos, ID, servirá para identificar el tipo de usuario. Es importante no confundirlo con un identificador de usuario. Si el valor es 0, indicará que se trata de un usuario regular. Si se trata de un 1, será un administrador. Ambos usuarios dispondrán de distintas funcionalidades, por lo que es importante diferenciarlos.

Los campos a continuación serán el *email*, el nombre de usuario y la contraseña del usuario. Respecto a este último campo, se hará uso de herramientas que permitan su codificación y de-

codificación segura para evitar problemas de seguridad. Respecto a los paquetes guardados, se almacenarán los ID de los paquetes almacenados en la base de datos pertinente, para poder acceder a su información cuando sea requerido. Los campos posteriores almacenarán información personal del usuario como puntuación, imagen de perfil, fecha de creación, última vez conectado, etc.

La base de datos que se va a diseñar a continuación es la de los paquetes. A fines de almacenamiento, los paquetes no siempre se almacenarán en la base de datos, sino que serán generados dinámicamente. Esto significa que cuando el usuario realice una búsqueda, el algoritmo se encargará de tomar los datos que el usuario ha aportado y generar todos los paquetes posibles. Se guardarán aquellos paquetes que el usuario decida almacenar en sus paquetes guardados. Así, se guardarán todos los paquetes que al menos hayan sido guardados una vez. De cara a la búsqueda de paquetes, se contemplan dos campos críticos: contador, para poder llevar un rastreo de los paquetes más veces guardados y *userIDs*, que permitirá saber que usuarios han guardado el paquete. De esta manera, podrán ser recuperados y accedidos cuando se requieran. El diseño para esta base de datos será:

ID	Transport ID	Accommodation ID	Activities ID	Price	Count	user IDs
1521	0000001, 0000002	0002412	0001241	450	4	052,501
1521	0000001, 0000002	0002412	0001241	450	7	052,501
1521	0000001, 0000002	0002412	0001241	450	8	052,501
1521	0000001, 0000002	0002412	0001241	450	9	052,501
1521	0000001, 0000002	0002412	0001241	450	2	052,501

Tabla 2.1: Diseño de Base de Datos para Paquetes

El ID del paquete identifica de manera única el paquete almacenado. A su vez, cada uno de los identificadores de los distintos campos permitirá acceder a la información concreta de cada uno de ellos. Por último, se almacenará el precio calculado y los campos anteriormente descritos.

Relacionada con esta base anterior, se contempla una base de datos por cada uno de los campos: Transporte, Alojamiento y Actividades. Al igual que ocurre con el caso anterior, solamente se almacenarían datos en caso que el usuario decida almacenar el paquete. Cada una de ellas dependerá de la información obtenida de las distintas *APIs* que se usarán [véase 2.8]

El diseño para la base de datos de transporte será la representada en la tabla B.2.

Por otro lado, el diseño para la base de datos del alojamiento será la representada en la tabla B.3. Por último, El diseño para la base de datos de las actividades será la reflejada en la tabla 2.2 a continuación:

ID	Lugar	Actividad
02013	Londres	Museo Británico
02023	Madrid	Museo del Prado
02035	Roma	El Coliseo

Tabla 2.2: Diseño de Base de Datos para Actividades

Como puede apreciarse, cada opción poseerá un ID propio para poder identificarlo únicamente, así como los campos establecidos.

Por último, se debe plantear una última base de datos: la de los formularios. En esta base, se

guardarán todas aquellas consultas realizadas a través del formulario de contacto. El diseño para esta base de datos es el siguiente:

ID	ID User	Title	Menssage
0512	XXXXXX	No funciona Filtros	No deja aplicar los filtros de ...
0512	XXXXXX	No funciona Filtros	No deja aplicar los filtros de ...
0512	XXXXXX	No funciona Filtros	No deja aplicar los filtros de ...
0512	XXXXXX	No funciona Filtros	No deja aplicar los filtros de ...
0512	XXXXXX	No funciona Filtros	No deja aplicar los filtros de ...

Tabla 2.3: Diseño de Base de Datos para Consultas de Formulario

Simplemente tendrá un identificador, el titulo de cada mensaje y el cuerpo del mismo, así como el identificador del usuario que ha realizado dicha consulta.

Teniendo en cuenta la estructura de las bases de datos, sería bastante útil optar por un modelo de datos relacional. Un modelo relacional consiste en representar datos por medio de tablas relacionadas cuyas filas se llaman tuplas y las columnas variables, conformando así una base de datos. Hay que tener en cuenta que las propias tablas ya establecen relaciones entre los datos, de ahí el nombre. En realidad se rige por unas normas sencillas, que son las siguientes:

- Las tablas son el centro del modelo y los datos deben ser representados en ellas. También se usan tablas cuando se calculan resultados de otras.
- El orden de cada columna viene determinado por el tipo de consulta que se realice. Por tanto, no es necesario un orden inicial, ya que cada relación la conforma un conjunto único de datos.
- Las filas son datos o casos, las columnas campos o variables. Por otro lado, cada celda es un registro que tiene dos dimensiones.
- Es imprescindible disponer de un identificador único (clave primaria) de cada registro. Este permite establecer relaciones entre dos o más tablas, usándolo como una clave externa.

Entre las ventajas podemos destacar las siguientes: Por un lado, tiene procesos que evitan la duplicidad de los datos.Por otro, garantiza la integridad referencial eliminando todo lo relacionado con un registro cuando sea necesario. Además, permite la normalización. [11]

2.8. *APIs* a Utilizar

Lógicamente, es necesario hacer uso de *APIs* de terceros que permita crear los paquetes que posteriormente serán ofrecidos al usuario. Es por ello que se debe realizar un estudio en profundidad de las distintas opciones de las que se disponen para obtener los datos necesarios.

Con este fin, se va a hacer uso de la herramienta *RapidApi* [12]. Es una página que ofrece una gran multitud de *APIs*, permitiendo probarlas en un entorno simulado. Así, se puede comprobar, entre otras cosas, que parámetros son necesarios, que opciones presenta cada *API*, la estructura del resultado de la operación y demás.

En primer lugar, debemos elegir una *API* que nos permita obtener información sobre los vuelos. Tras una búsqueda en profundidad, se ha optado por hacer uso de *SkyScanner API*. Esta *API* concreta ofrece una serie de opciones de uso. En particular, se hará uso de:

- *Get List Places*: Esta opción será usada para validar el lugar introducido por el usuario. Esto es muy importante, ya que nos permitirá comprobar que existe en la base de datos de *SkyScanner* y permitirá obtener los datos necesarios para realizar la búsqueda posterior. Con respecto a los campos requeridos, serán:

Campo	Tipo	Definición	Ejemplos
query	String	Lugar que se pretende validar en el sistema	Roma, Madrid
country	String	País en el que está el usuario	UK, USA
currency	String	Moneda para la expresión de precios	GBP, EUR
locale	String	Idioma para la expresión del resultado	en-EN, es-ES

Tabla 2.4: Parámetros para realizar una búsqueda de un lugar

A su vez, los datos que se deberán almacenar de los ofrecidos en el resultado de la petición serán:

Nombre Variable	Acceso	Ejemplo
PlaceId	Places[i].PlaceId	STOC-sky* siempre con -sky
PlaceName	Places[i].PlaceName	Stockholm
CountryName	Places[i].CountryName	Sweden

Tabla 2.5: Variables a considerar del resultado ofrecido

- *Get Browse Quotes*: Esta opción permitirá obtener el vuelo de coste mínimo entre dos lugares. Será el usado para crear el paquete.Los campos requeridos para esta funcionalidad son:

Campo	Tipo	Definición	Ejemplos
country	String	País en el que está el usuario	UK, USA
currency	String	Moneda para la expresión de precios	GBP, EUR
locale	String	Idioma para la expresión del resultado	en-EN, es-ES
originplace	String	Identificador del lugar de origen	SFO-sky, STOC-sky
destinationplace	String	Identificador del lugar de destino	SFO-sky, STOC-sky
outboundpartialdate	String	Fecha de Ida	2021-05-20, 2019-12-10
*inboundpartialdate	String	Fecha de Vuelta	2021-05-20, 2019-12-10

Tabla 2.6: Parámetros para realizar una búsqueda de un lugar

El símbolo * implica que el campo es opcional, es decir, que no es necesario incluirlo para realizar la búsqueda. A su vez, los datos que se deberán almacenar de los ofrecidos en el resultado de la petición serán:

Nombre Variable	Acceso	Ejemplo
MinPrice	Quotes[i].MinPrice	120
Direct	Quotes[i].Direct	True
DepartureDate	Quotes[i].OutboundLeg.DepartureDate	2021-09-01T00:00:00
Name	Carries[i].Name	Ryanair
Name	Places[i].Name	New York John F.Kennedy
CountryName	Places[i].CountryName	United States
CityName	Places[i].CityName	New York

Tabla 2.7: Variables a considerar del resultado ofrecido

Con estas dos funcionalidades, se podrá obtener el vuelo que se incluirá en el paquete que esté siendo creado.

A continuación, se debe plantear que *API* se podría usar para obtener información acerca del alojamiento. Tras realizar la búsqueda adecuada, se ha optado por hacer uso del *API Hotel.com Provider*. A su vez, puede ser utilizado para obtener los lugares más relevantes, lo que equivaldría a actividades que pueden realizarse. Al igual que ocurre con la *API* de *SkyScanner*, debemos tener en cuenta que opciones podemos usar, y de que manera.

- *Get Search Hotels*: Esta funcionalidad permitirá buscar hoteles en un lugar concreto. Para poder hacer uso de ello, se requieren de los siguientes campos:

Campo	Tipo	Definición	Ejemplos
sort-order	String	Orden en el que aparecerá el resultado.	STAR-RATING-HIGHEST
checking-date	String	Fecha de entrada.	2021-03-26
currency	String	Moneda para la expresión de precios	GBP, EUR
locale	String	Idioma para la expresión del resultado	en-EN, es-ES
adults-number	String	Número de adultos	1,2,3...
checkout-date	String	Fecha de Salida	2021-03-30
destination-id	String	Identificador del destino	1708350
*price-max	String	Precio Máximo	500
*landmark-id	String	Identificador de Zona	107842
*children-age	String	Edades de los niños a hospedarse	4,0,15
*page-number	String	Número de Página a buscar	1
*guest-rating	String	Puntuación mínima de los usuarios	4
*amenity-ids *theme-id	String String	Accesibilidad e Instalaciones Temáticas	527, 2063 14,27,25
*star-rating-ids	String	Estrellas de los hoteles	3,4,5
*price-min	String	Precio mínimo	100
*accomodation-ids	String	Tipo de alojamientos	20,8,15,5,1

Tabla 2.8: Parámetros para realizar una búsqueda de un lugar

A su vez, los datos que se deberán almacenar de los ofrecidos en el resultado de la petición serán:

Nombre Variable	Acceso	Ejemplo
value	query.value	Moscow,Russia
name	searchResults.results[i].name	Sadovnicheskaya Hotel
starRating	searchResults.results[i].starRating	5
streetAddress	searchResults.results[i].address.streetAddress	Sadovnicheskaya 20 str. 1
rating	searchResults.results[i].guestReviews.rating	4.3
scale	searchResults.results[i].guestReviews.scale	5
current	searchResults.results[i].ratePlan.price.current	169€
old	searchResults.results[i].ratePlan.price.old	200€

Tabla 2.9: Variables a considerar del resultado ofrecido

- *Get Search Destination by Query*: Esta funcionalidad devolverá los lugares almacenados relacionados con el nombre introducido. Por ejemplo, si se introduce Londres, devolverá todos los destinos que reconozca que se relacionen con él. Además, permitirá obtener los lugares de

interés relacionados con el lugar. Para poder hacer uso de ello, se requieren de los siguientes campos:

Campo	Tipo	Definición	Ejemplos
country	String	País en el que está el usuario	UK, USA
currency	String	Moneda para la expresión de precios	GBP, EUR
locale	String	Idioma para la expresión del resultado	en-EN, es-ES
query	String	Lugar a buscar	London, New York

Tabla 2.10: Parámetros para realizar una búsqueda de un lugar

A su vez, los datos que se deberán almacenar de los ofrecidos en el resultado de la petición serán:

Nombre Variable	Acceso	Ejemplo
name	suggestions[0].entities[i].name	London
destinationId	suggestions[0].entities[i].destinationId	549499
name	suggestions[2].entities[i].name	British Museum

Tabla 2.11: Variables a considerar del resultado ofrecido

- *Get Photos of The Hotel*: Esta última opción es también muy interesante, ya que permitirá mostrar al usuario fotos del sistema del hotel. Para poder hacer uso de ello, se requieren de los siguientes campos:

Campo	Tipo	Definición	Ejemplos
hotel-id	String	Identificador del Hotel	363464

Tabla 2.12: Parámetros para realizar una búsqueda de un lugar

A su vez, los datos que se deberán almacenar de los ofrecidos en el resultado de la petición serán:

Nombre Variable	Acceso	Ejemplo
mainUrl	[i].mainUrl	https://xxxxxx.yy.jpg

Tabla 2.13: Variables a considerar del resultado ofrecido

Capítulo 3

Implementación

En este capítulo se tratarán los aspectos propios de la implementación de la aplicación, utilizando la información recogida anteriormente en los distintos capítulos de diseño, siendo traducidos a la práctica. Se pretende recoger los distintos pasos que permiten convertir la idea en realidad, contemplando algoritmos, metodologías, herramientas de programación, etc. Se puede consultar el **repositorio de Github del proyecto** para acceder al código del mismo.

Una aplicación está compuesta, en líneas generales, de 2 elementos: *Frontend* y *Backend*. Cada uno de ellos tiene una serie de funciones diferenciadas. El *frontend* constituye la parte visual de la aplicación, es decir, lo que el usuario interpreta como aplicación. Por el contrario, el *backend* realiza las tareas invisibles a los mismos, pero necesarias para el funcionamiento correcto de la aplicación, obteniendo por ejemplo los datos de terceros.

3.1. *Backend*

El *Back-End* es la parte o rama del desarrollo web encargada de que toda la lógica de una página funcione. Consiste en el conjunto de acciones que pasan dentro de una web, pero que no podemos ver. [13]

Efectivamente, el *backend* de Pack2Fly será la encargada de realizar las operaciones funcionales de la aplicación, sin que el usuario sea consiente de ello. Las operaciones que realizará son:

- Interactuar con la base de datos: Entiéndase la creación de las tablas (solo la primera vez que se ejecuta), que son *Pack*, Transporte, *Accomodation*, *Activity* y Personas, así como las operaciones con las mismas (agregación, eliminación y consulta).
- Realizar las peticiones a las distintas *APIs* usadas para obtener la información necesaria con los datos proporcionados.
- Analizar la información obtenida y tratarla, almacenando únicamente la información relevante acerca del transporte, alojamiento y actividades.
- Crear los paquetes teniendo en cuenta los datos almacenados, guardando los mismos de manera que puedan ser accesibles.

La arquitectura del *backend* sigue el patrón de *microservicios*. Esto implica que ofrece muchas funcionalidades distintas de cara al *frontend*, de manera que se diferencien perfectamente y se adapten a las distintas necesidades de la aplicación.

A fines prácticos, el *backend* se ha dividido en una serie de módulos con funciones definidas:

- ***Module-Trans.js:*** Este módulo es el encargado de realizar la petición a la *API* de *SkyScanner* y del tratado de la información obtenida sobre el transporte para almacenar pertinentemente la misma en variables accesibles.
- ***Module-Accomo-Acti.js*:** Este módulo es el encargado de realizar la petición a la *API* de *Hotels.com* y del tratado de la información obtenida sobre el alojamiento y las actividades para almacenar pertinentemente la misma en variables accesibles.
- ***App.js*:** Este módulo se encargará de inicializar la conexión con el servidor *MySQL*
- ***Module-Pack-Generator.js*:** Este módulo será le encargado de crear los paquetes, haciendo uso del resto de módulos implementados.
- ***Modulo-Tablas.js*:** Este módulo será el encargado de la interacción directa con las tablas: eliminación, accesso y agregación.
- ***Module-Checker.js*:** Este módulo será el encargado de realizar las tareas de comprobación de la integridad de las variables.

Debido a la complejidad que presenta cada uno de los módulos, es necesario explicar en profundidad como realiza sus funciones cada uno de ellos.

3.1.1. *Module-Trans.js*

Como ya se comentó anteriormente, este módulo será el encargado de obtener la información del vuelo para los paquetes. Ya en el anterior capítulo se describió el diseño que se había hecho para la tabla de transporte. Basado en ello, se deberá tener una variable donde poder almacenar los datos relevantes: la variable transport

```
1
2 var transport = {
3     minPrice : String,
4     Direct : Boolean,
5     DepartureDate: String,
6     Airline: String,
7     LandMarkName_Origin: String,
8     LandMarkName_Destination: String,
9     Country_Origin: String,
10    Country_Destination: String,
11    City_Origin: String,
12    City_Destination: String
13 };
```

En esta variable almacenaremos toda la información que se obtendrá de la petición.

En el apartado 2.8, se comentó que debido a la peculiaridad de la *API SkyScanner*, es necesario conocer la ID particular del destino y del origen para poder realizar la búsqueda del vuelo. Para conseguir ambos, es necesario llamar a un método *GET* de la *API*. Lógicamente, resulta claro que se debe dividir el módulo en 2 métodos: Uno para la petición de las IDs (*initiate-ids*) y otro para la petición de búsqueda del vuelo (*initiate-tra*).

Afortunadamente, las propias *APIs* proporcionan el esqueleto para realizar la petición. En este caso, se dispone de una variable options, cuyo esqueleto (ya personalizado para *initiate-ids*) es:

```
1     const options = {        //variable para realizar la peticion, aportada por la API
2         "method": "GET",
```

```
        "hostname": "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com",
        "port": null,
        "path": get_place(), //"/apiservices/autosuggest/v1.0/US/USD/en-US/?query=London",
        "headers": {
            "x-rapidapi-key": "f1437027d0msh29390ccc3de5e5ep17f081jsndb3b1a2bd225",
            "x-rapidapi-host": "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com",
            "useQueryString": true
        }
```

Además, también se dispone del cuerpo de una función que realiza la llamada y obtiene la información. Por ejemplo, en el caso de *initiate-ids*, sería:

```
                const req = http.request(options, function (res) {
                    const chunks = [];

                    res.on("data", function (chunk) {
                        chunks.push(chunk);
                    });

                    res.on("end", function () {
                        const body = Buffer.concat(chunks);
                        resultado_id = body.toString();
                    });
                });
                req.end();
```

Un aspecto muy importante que se debe tener en cuenta es la sincronía. Esto afecta a cualquier petición. Cuando se realiza, el código sigue ejecutándose en paralelo a la petición, por lo que, si no lo tenemos en cuenta, provocará errores graves. Por ejemplo, si quisiéramos devolver, en el ejemplo anterior, resultado-id, para almacenarlo en una variable fuera del módulo, daría un valor *undefined*. Es por ello que se debe recurrir a una funcionalidad de JavaScript: **Las Promesas.** Una promesa es un objeto que representa un valor que puede que esté disponible «ahora», en un «futuro» o que «nunca» lo esté. [14] Esto significa que si se devuelve una promesa, no se resolverá la operación (de asignación, por ejemplo) hasta que dicho valor esté disponible. Esto permite solucionar un problema bastante molesto. Únicamente habrá que modificar ligeramente la operación de petición, quedando el ejemplo anterior de la siguiente manera:

```
    return new Promise((resolve, reject) => {
          try {
              const req = http.request(options, function (res) {
                  const chunks = [];

                  res.on("data", function (chunk) {
                      chunks.push(chunk);
                  });

                  res.on("end", function () {
                      const body = Buffer.concat(chunks);
                      resultado_id = JSON.parse(body);
                      resolve(getPlaceId());
                  });
              });
              req.end();
          }catch (error){
              throw error;
          }
      });
```

Otro aspecto a tener en cuenta que afecta al comportamiento del módulo es la forma en la que se presentan los datos de la petición. Si se observa el código inmediatamente anterior, en resultado-id se almacena el resultado convertido en un objeto a través de la función de parseo de JSON. Esto permite que el contenido se almacene de forma que se pueda acceder al mismo a través de índices, conociéndo la estructura de la petición. Por ello, se ha implementado la función get-place-id para el ejemplo de obtención de los *IDs*.

En este módulo se encuentra la otra función principal que lo compone, que es el método *get-transport-info*. Al igual que con la función de obtención del ID, accede a los campos correspondientes, almacenando en las campos propios de la variable *transport* la información. El código sirve para ver el comportamiento de acceso al objeto, por lo que puede ser usado como ejemplo:

```
1        function get_transport_info(){
2         transport.minPrice = resultado.Quotes[0].MinPrice;
3         transport.Direct = resultado.Quotes[0].Direct;
4         transport.DepartureDate = resultado.Quotes[0].OutboundLeg.DepartureDate;
5         transport.City_Origin = resultado.Places[1].Name;
6         transport.City_Destination = resultado.Places[2].Name;
7         transport.Country_Origin = resultado.Places[1].CountryName;
8         transport.Country_Destination = resultado.Places[2].CountryName;
9         transport.Airline = resultado.Carriers[0].Name
10        transport.LandMarkName_Origin = resultado.Places[1].Name;
11        transport.LandMarkName_Destination = resultado.Places[2].Name;
12        return transport;
13    }
```

Lógicamente, para conocer dónde está cada valor que interesa, se ha utilizado la opción de depuración del *IDE*, que permite visualizar el valor adoptado por la variable.

Para finalizar con el módulo, se debe establecer un método que llame a las dos funciones que se han comentado y almacene la información resultante:

```
1  exports.get_trans_pack= async (origin,destiny,check_in_date,check_out_date)  => {
2      try {
3          var id_destiny = await initiate_ids(destiny);
4          var id_origin = await initiate_ids(origin);
5
6          var tran_info = await initiate_tra(id_origin, id_destiny, check_in_date, check_out_date);
7          transport = tran_info;
8      }catch (error){
9          throw error;
10     }
11 }
```

Lógicamente, esta función será exportable, para poder ser llamada desde la aplicación principal. Como ya se ha comentado, el *await* permite que las variables obtengan su valor antes de que continue la ejecución.

3.1.2. Module-Accomo-Acti.js

Este módulo será el encargado de obtener las distintas opciones de alojamiento y las distintas actividades asociadas al destino. Se ha juntado ambas acciones debido a que se obtienen de la misma *API*. Al igual que ocurre con el módulo anterior, se debe disponer de una variable donde almacenar la información obtenida:

```
1 var resultado_hoteles; //!< Variable para almacenar el resultado final del alojamiento
2 var interest =[];//!< Variable para almacenar el resultado de las actividades
```

```
var hotel = {    //!< Variable para almacenar la informacion obtenida
                 // de la peticion por cada alojamiento
    id : String,
    name : String,
    Star_rating : String,
    address : String,
    guest_rating : String,
    scale : String,
    current_price : String
}
```

En este caso, hotel es un objeto que posee los atributos que interesan para el alojamiento. En la variable *resultado-hoteles*, se almacenarán las distintas opciones que se han conseguido. Finalmente, en la variable interest, se guardarán aquellas actividades que se consigan obtener.

Al igual que ocurre con el módulo de transporte, es necesario obtener el ID del lugar de destino, por lo que se deberá disponer de una función initiate-ids que se comportará de la misma manera. Además, adicionalmente, en la misma petición se puede obtener los lugares de interés. La función de obtención del ID es equivalente a la del módulo de transporte. Sin embargo, el resultado ofrece distintos lugares, por lo que difiere un poco:

```
    function getPlaceID(){
        for(let i=0;cuerpodeResultado.suggestions[0].entities.length;i++){
            if(cuerpodeResultado.suggestions[0].entities[i].name === place_to_search){
                return cuerpodeResultado.suggestions[0].entities[i].destinationId;
            }
        }
    }
```

Se debe iterar sobre las distintas entidades (que constituyen las distintas localizaciones) para comparar si el nombre de la misma corresponde con el lugar que estamos tratando, en cuyo caso devolvemos el identificador.

En el caso de la de obtención de las actividades, difiere ligeramente con la de obtención del vuelo, por lo que requiere de un análisis en profundidad:

```
 function getInterestingPlaces(){
        let latitude = "";
        let longitude = "";
        let name ="";

        for(let i=0;i<cuerpodeResultado.suggestions[2].entities.length;i++){
            name = cuerpodeResultado.suggestions[2].entities[i].name;
            longitude = cuerpodeResultado.suggestions[2].entities[i].longitude;
            latitude = cuerpodeResultado.suggestions[2].entities[i].latitude;

             var aux = {
                  name: name,
                  latitude: latitude,
                  longitude: longitude
              }
              interest.push(aux);
         }
```

En este caso, se deberá iterar sobre las entidades del tercer resultado de *suggestions*. Esto es debido a que el tercer resultado corresponde con "LANDMARK-GROUP", es decir, donde se almacenan los lugares de interés. Es por ello que, accediendo a las distintas entidades presentes

en ese grupo, se obtendrán el nombre del lugar, así como su localización.

La función de obtención de la información del hotel también presenta una estructura muy parecida a la de la obtención de la información del vuelo. Simplemente se accede a los campos correspondientes para obtener la información pertinente. Por último, al igual que ocurría con el módulo de transporte, se establece una función exportable que se encarga de obtener e inicializar la información completa del alojamiento y de las actividades:

```
exports.get_acco_pack = async (place,adults,ci,co) => {
    try {
        var id_destiny = await initiate_ids(place);
        resultado_hoteles = await initiate_hotel(adults, ci, co, id_destiny, "GUEST_RATING");
    }catch (error){
        throw error;
    }
}
```

La función exportable para obtener el alojamiento realiza la llamada a la función propia e inicializa la variable *resultado-hoteles*. Respecto a las actividades, no es necesaria una función propia, ya que con la llamada a la función initiate-ids ya se inicializaría el vector correspondiente, como se ha indicado.

3.1.3. Module-Checker.js

A lo largo del código, resulta indispensable comprobar el estado de las variables que forman parte de la búsquedas. El objetivo de esto no es evitar el error, sino notificarlo y manejarlo de manera adecuada. Este módulo presenta 2 funciones: *check-undefined* y *check-type*. Como sus nombres indican, estas funciones comprobarán si la variable que reciben están definidas (es decir, tienen un valor distinto de *undefined*) y que el tipo que presentan es igual al esperado (por ejemplo, si *origin* es del tipo *string*). Además, se creará la función *check-variable* del tipo exportable, que puede ser llamada desde cualquier módulo, que tiene el siguiente aspecto:

```
exports.check_variable = (variable,type) =>{
    try{
        check_undefined(variable)
        check_type(variable,type)
    }catch (error){
        throw error;
    }
}
```

3.1.4. Modulo-Tablas.js

Como ya se comentó anteriormente, este módulo realiza las tareas relacionadas con la base de datos y, más concretamente, con las tablas. En primer lugar, presenta una función que se encarga de crear (o comprobar si están creadas) las tablas. También, presenta funciones de borrado de información de la tabla pertinente. Por ejemplo, para la tabla de usuarios es:

```
exports.erase_user = (ID) =>{
    try {
        Personas.destroy({
            where: {
                id: ID
            },
            force: true
        })
```

```
    }catch (error){
        console.log(erro);
    }
}
```

Lógicamente, también contiene funciones encargadas de almacenar información en las tablas. Como ejemplo, se tomará la función de almacenamiento de paquetes:

```
exports.save_pack = async (pack) =>{
    var activities='';

    for(var i=0;i<pack.places.length;i++){
        activities+=pack.places[i].name;
        if(i<pack.places.length-1){
            activities+=',';
        }
    }
    /**
     * Almacenamos la informacion en la tabla con un formato transporte,alojamiento,
       actividades y precio
     * */
    const pack_to_save = Pack.create({
        transport: "Flight " + pack.flight.Airline + " from " + pack.flight.City_Origin +
           " to " + pack.flight.City_Destination + " at " + pack.flight.DepartureDate,
        //Vamos a identificar el vuelo con origen + destino + fecha ida
        accomodation: pack.hotel.name,
        activities: activities,
        price: pack.price
    })

    /**
     * Almacenamos a su vez cada uno de los componentes que componen el pack
     * */

    save_transport(pack.flight)
    save_accomodation(pack.hotel)
    save_activity(pack.places)
}
```

Primero obtiene el nombre de cada actividad y crea una cadena del tipo "xxxx,yyyy,zzzz". La información se guardará en el *pack* como identificadores. En el caso del transporte, una cadena que permita identificar adecuadamente el vuelo. El alojamiento con el nombre del mismo y las actividades como la cadena anteriormente especificada. Finalmente, se deberá almacenar individualmente cada opción. La función para almacenar los mismos tiene el siguiente esqueleto (ejemplo del transporte):

```
async function save_transport(transporte){
    const transport= Transporte.create({
        minPrice: transporte.minPrice,
        direct: transporte.Direct,
        departure: transporte.DepartureDate,
        airline: transporte.Airline,
        landmarkOrigin: transporte.LandMarkName_Origin,
        landmarkDestination: transporte.LandMarkName_Destination,
        countryOrigin: transporte.Country_Origin,
        countryDestination: transporte.Country_Destination,
        cityOrigin: transporte.City_Origin,
```

```
        cityDestination: transporte.City_Destination
    })
}
```

Simplemente almacena uno a uno en la tabla los distintos campos del objeto transporte pasado como argumento la función.

Lógicamente, también presenta funciones que permiten mostrar las tablas correspondientes. Además, para finalizar el módulo, presenta una función que se encarga de comparar paquetes en la tabla. Esto es importante, ya que se pretende que aquellos paquetes ya guardados que cumplan las condiciones de búsqueda del usuario también se ofrezcan dentro del resultado al usuario. La función es:

```
exports.compare_pack_table = async () => {
    const packs_table = await Pack.findAll();
    for (var i=0;i<packs_table.length;i++){
        var transport=packs_table[i].transport;
        var split = transport.split(" ");
        for(var z=0;z<split.length;z++){
            if(split[z]=="from"){
                if(split[z+1]==destiny){
                    packs.push(packs_table[i]);
                }
            }
        }
    }
}
```

En este caso, la única condición que se va a tener en cuenta es que el destino del paquete guardado coincida con el deseado por el usuario.

3.1.5. Module-Pack-Generator

Este módulo, como su nombre indica, llevará a cabo las tareas de generación de paquetes en función de los datos obtenidos del usuario. Para ello, hará uso de una serie de funciones. En primer lugar, se presenta la función *get-module-acco-acti-info*

```
async function get_module_acco_acti_info() {
    try{
        modulo_checker.check_variable(origin,"string");
        modulo_checker.check_variable(destiny,"string");
        modulo_checker.check_variable(adults,"number");
        modulo_checker.check_variable(check_in_date,"string");
        modulo_checker.check_variable(check_out_date,"string");
    }catch (error){
        throw error;
    }

    await modulo_acco_acti.get_acco_pack(destiny, adults, check_in_date, check_out_date);
    await modulo_transporte.get_trans_pack(origin, destiny, check_in_date, check_out_date);

    hotels = modulo_acco_acti.get_hotels();
    interested_places = modulo_acco_acti.get_interest();
    flight = await modulo_transporte.get_transport();

    try {
        modulo_checker.check_variable(hotels,"object")
```

```
21          modulo_checker.check_variable(interested_places,"object")
22          modulo_checker.check_variable(flight,"object")
23      }catch (error){
24          console.error(error);
25      }
26      /**
27       * Creamos el paquete por cada opcion de alojamiento
28       * */
29      for(var i=0;i<hotels.length;i++){
30          create_pack(hotels[i]);
31      }
32  }
```

En primer lugar, comprueba el estado de las variables necesarias para la búsqueda. A continuación, llama a las funciones de obtención de la información necesaria para el paquete, que inicializarán la información necesaria. Se obtendrá dicha información, almacenándola globalmente en las variables pertinentes y finalmente se llamará a la función de crear paquetes:

```
1  function create_pack(hotel){
2      var nuevo_pack = {
3          hotel:hotel,
4          flight:flight,
5          places:interested_places,
6          price:calculate_total_price(hotel)
7      }
8      packs.push(nuevo_pack);
9  }
```

Lo único que hace esta función es almacenar la información en una nueva variable *nuevo-pack*, realizando una operación de *push* en el vector *packs*. Por último, la función calculate-total-price se encarga de calcular el precio total del paquete:

```
1  function calculate_total_price(hotel){
2      var price;
3      var aux="";
4      for(var i=0;i<hotel.current_price.length;i++){
5          if(hotel.current_price[i]!="$"){
6              aux += hotel.current_price[i];
7          }
8      }
9      price = parseInt(aux);
10     price += flight.minPrice;
11     return price.toString();
12 }
```

Obtiene el precio del hotel, sumándole el del vuelo y devolviendo el valor en formato de cadena.

Esta breve explicación constituye una introducción al comportamiento completo del *backend*. Lógicamente, el objetivo del mismo es actuar tras las acciones realizadas por el usuario cuando sea necesario, siendo transparente ante la persona. El código al completo está disponible en el repositorio correspondiente en *Github* para un análisis más detallado.

3.2. Frontend

Un sistema de *frontend* es parte de un sistema de información al que el usuario accede directamente e interactúa para recibir o utilizar las capacidades de *backend* del sistema anfitrión. Permite a los usuarios acceder y solicitar las prestaciones y servicios del sistema de información subyacente. El sistema de *frontend* puede ser una aplicación de software o hardware o su combinación, así como recursos de la red. Un sistema de *frontend* se utiliza principalmente para enviar preguntas y solicitudes, y recibir datos desde el sistema anfitrión. Sirve o proporciona a los usuarios la capacidad de interactuar y utilizar un sistema de información. [15]

En el caso de Pack2Fly, el *frontend* será el encargado de interactuar con el usuario para obtener información del mismo. Esta información será necesaria para poder realizar las peticiones al sistema *backend*.

La arquitectura del *frontend*, por peculiaridad de Angular, se basa en el MVC o modelo Vista-Controlador. La idea principal de este modelo es dividir la aplicación en vistas, con su lógica asociada, o controladores, que se encargarán de realizar las operaciones necesarias cuando sean requeridas.

Las funciones que realizará el *frontend* son:

- Solicitar los datos de viaje del usuario: Se obtendrán el destino, el origen, las fechas de ida y vuelta y el número de viajeros.
- Ofrecer paquetes recomendados y *top*: En las páginas correspondientes, ofrecerá al usuario los paquetes afines a sus gustos y los paquetes más reservados.
- Permitir el contacto con la empresa: Dispondrá de un formulario y opciones de contacto para los usuarios.
- Ofrecerá un diseño simple y atractivo para una mejor experiencia de usuario, con elementos simplificados a la menor expresión.

  En este caso, se ha optado por Angular [16] para el desarrollo del *frontend*. Esta tecnología se caracteriza principalmente por los tipos de elementos que ofrece. En el caso de Pack2Fly, se va a hacer uso de los siguientes:
- Componentes: Son elementos que siempre van a estar, o que son comunes a todas las páginas. Por ejemplo, el menú de navegación y el pie de página.
- Páginas: Son los elementos que se compondrán las distintas páginas de la aplicación. Por ejemplo, la página de inicio, la de contacto.
- Servicios: Serán los encargados de la interacción con el *backend*. En este caso, realizar las peticiones para generar los paquetes, crear usuarios, etc.

Lógicamente, debemos profundizar en cada uno de estos elementos.

3.2.1. Componentes

Un componente en Angular es un bloque de código re-utilizable, que consta básicamente de 3 archivos: un CSS, un HTML (también conocido como plantilla o en inglés, *template*) y un TypeScript (en adelante, TS). Todo en angular son componentes. Sin embargo, por convenio, se consideran como componentes aquellos que serán comunes a toda la aplicación.

En el caso de esta aplicación, y como se ha comentado anteriormente, se contará únicamente con 2 componentes. Estos componentes son el *Footer* y el *Header*, ya que en ambos casos, se pretende que estén presentes en todas y cada una de las páginas.

Los componentes de este tipo son definidos en el app.component.html. Este código HTML es el que se ejecuta siempre al iniciarse la aplicación. Por tanto, siempre que se lance la misma, se mostrarán los componentes definidos en ella:

```
<app-navigation-bar></app-navigation-bar>
<router-outlet></router-outlet>
<app-footer></app-footer>
```

El componente *router-outlet* permitirá poder mostrar las distintas páginas que compondrán la aplicación. Para ello, se hace uso del sistema de *routing* de Angular: Se definen en el app-routing.module.ts las distintas direcciones o *paths* para las páginas que se contemplan en la aplicación. Así, se relacionará cada componente con una ruta específica. Estas direcciones serán incluidas en el vector de rutas del módulo, de tal manera que cualquier elemento podrá acceder a las mismas. La declaración para el caso de Pack2Fly es:

```
const routes: Routes = [
  {path: 'search-form', component: SearchFormComponent},
  {path: 'contact-form', component: ContactFormComponent},
  {path: 'about-us', component: AboutusComponent},
  {path: 'recommended-packs', component: RecommendedpacksComponent },
  {path: 'top-packs', component: ToppacksComponent }];
```

Esta funcionalidad es incluida al agregar el componente *router-outlet*. Además, estas rutas definidas serán usadas en el navegador de la página para redirigir[a]l usuario a la página pertinente.

Componente 1: Barra de Navegación

Este componente será el encargado de permitir al usuario navegar entre las distintas páginas que conforman la aplicación. Dado que es el primer ejemplo de componente (componente angular, no componente de la aplicación), es conveniente analizar adecuadamente el mismo.

Un componente en angular cuenta con 4 elementos: el HTML, el TS, el CSS y el .SPEC.TS. Cada uno de ellos permite agregar al componente una serie de funcionalidades, aunque los más importantes en el caso de esta aplicación son los siguientes:

HTML

contendrá el código para el navegador que se ejecutará en primer lugar. Por regla general, se agregarán los distintos elementos que, junto con el CSS, se mostrarán en la página. También es posible que se agregue código para modificar el comportamiento de los mismos. El código HTML para la barra de navegación es el siguiente:

```

<div class="topnav" id="myTopnav">
  <a mat-list-item routerLink="/search-form">Home</a>
  <a mat-list-item routerLink="/about-us">About Us</a>
  <a mat-list-item routerLink="/contact-form">Contact</a>
  <a mat-list-item routerLink="/recommended-packs">Recommended Packs</a>
  <a mat-list-item routerLink="/top-packs">Top Packs</a>
  <a mat-list-item routerLink="#">My Account</a>
  <a href="javascript:void(0);" class="icon" onclick="myFunction()">
    <i class="fa fa-bars"></i>
  </a>
</div>


```

```
<script>
  function myFunction() {
    var x = document.getElementById("myTopnav");
    if (x.className === "topnav") {
      x.className += " responsive";
    } else {
      x.className = "topnav";
    }
  }
</script>
```

En este caso, cuenta con una división que contiene los distintos *links* para cada página. Estos *links* son los proporcionados en el ruteo, como se ha comentado anteriormente. Se puede apreciar que una de las entradas es un logo, que si es pulsado activa el código JavaScript incluido dentro de la etiqueta*script*. Este código simplemente edita el nombre de la barra de navegación para agregar *responsive* en el caso conveniente.

CSS

CSS se usa para estilizar elementos escritos en un lenguaje de marcado como HTML, separando los estilos en un fichero propio. En él, se puede modificar la apariencia de cada elemento que compone el componente angular. El código CSS tiene una estructura como esta:

```
body {
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
  background: deepskyblue;
}


.topnav {
  overflow: hidden;
  background-color: deepskyblue;
}
...

@media screen and (max-width: 600px) {
  .topnav.responsive {position: relative;}
  .topnav.responsive .icon {
    position: absolute;
    right: 0;
    top: 0;
  }
  .topnav.responsive a {
    float: none;
    display: block;
    text-align: left;
  }
}
```

Se puede modificar de manera general o específicamente el fondo, el color de texto, la distribución del mismo, etc. Lo más destacable es, sin lugar a duda, @media. Estas instrucciones serán ejecutadas si se cumple una o más condiciones. Por ejemplo, vemos que en el caso de que la pantalla tenga una resolución menor de 600px, se modificará la barra de navegación, de tal manera que se ajuste adecuadamente a la misma. Esto es muy útil, ya que la aplicación debe poder visualizarse correctamente sin importar el dispositivo.

TypeScript (TS)

En este bloque de código, se define el componente angular como tal, estableciendo la información pertinente: ruta del HTML asociado, ruta del CSS asociado, atributos, funcionalidades, etc. La gran ventaja que presenta es que permite definir métodos que puede ser usado por el componente, como por ejemplo una función para obtener la información del usuario. El TS para la barra de navegación es el siguiente:

```
import { Component } from '@angular/core';
import { BreakpointObserver, Breakpoints } from '@angular/cdk/layout';
import { Observable } from 'rxjs';
import { map, shareReplay } from 'rxjs/operators';

@Component({
  selector: 'app-navigation-bar',
  templateUrl: './navigation-bar.component.html',
  styleUrls: ['./navigation-bar.component.css']
})
export class NavigationBarComponent {

  isHandset$: Observable<boolean> = this.breakpointObserver.observe(Breakpoints.Handset)
    .pipe(
      map(result => result.matches),
      shareReplay()
    );

  constructor(private breakpointObserver: BreakpointObserver) {}

}
```

Todos los componentes angular poseen el mismo esqueleto: la definición del componente con la sintaxis @Component. El selector realiza el papel de nombre del componente, que será el usado para utilizar el mismo (véase código 1). Posteriormente, se realiza la definición de la clase en sí, donde se agregarían tantas funcionalidades como se deseen.

En líneas generales, estos son los conceptos básicos de un componente de uso general en angular.

3.2.2. Servicios

Los servicios son clases que se encargan de acceder a los datos para entregarlos a los componentes [17]. ¿Qué implica esto? Pues básicamente, un servicio va a solicitar al *frontend* los datos necesarios para poder mostrar la información al usuario, a través del *API Rest*.

Al contrario que ocurre con los componentes, un servicio está compuesto únicamente por TypeScript. La sintaxis es muy parecida, salvo ligeras diferencias. Se considera el siguiente ejemplo, consistente en el servicio que obtiene los paquetes:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})

export class UsuariosService {
```

```
9
10   constructor(private http: HttpClient) { }
11
12   get_packs(Origin: string,Destiny: string,Checkin: string,CheckOut: string,people: string){
13     return new Promise((resolve, reject) => {
14       this.http
15         .get("http://localhost:8000/packs?Origin="+Origin+"&Destiny="+Destiny+
16         "&Checkin="+Checkin+"&CheckOut="+CheckOut+
17         "&people="+people)
18         .subscribe(res => {
19             resolve(res);
20           },
21           (err: any) => {
22
23             reject(err);
24           }
25         )
26     });
27   }
28 }
```

En primer lugar, puede apreciarse que tras las importaciones, se encuentra la sentencia @Injectable. Esto indica que el servicio podrá ser injectado cuando sea requerido. Dentro del servicio, se encuentra la función *get-packs*. Lo más interesante de esta función es su estructura. Lógicamente, las consultas no son instantáneas. Por ello, al igual que se hacía cuando se realizaba una petición a una API externa en el *backend*, se devolverá la promesa con el resultado de la consulta. Por lo general, en los servicios suele hacerse uso de HTTP para las peticiones.

Sin lugar a duda, lo más interesante de este método es el código a continuación. Haciendo uso del protocolo HTTP, se utiliza la llamada *get* para solicitar la información. En este caso, es necesario conocer la dirección a donde realizar la búsqueda, que es simplemente la dirección en la que se ejecuta el *backend*. Finalmente, el bloque suscribe se ejecutará cuando se complete la petición, devolviendo la información obtenida si es exitoso el proceso o un error.

Si se analiza mejor la dirección URL que se utiliza en la llamada *get*, se puede observar que tiene una estructura tal que http://localhost:8000/packs...Esto, en un principio, puede pasar desapercibido, pero es realmente importante. ¿De dónde sale es /packs? Simplemente, se está haciendo uso de la *API Rest*, que sirve como intermediario entre *Front* y *Back*.

API Rest

Cuando se habla de *Rest API*, implica utilizar una *API* para acceder a aplicaciones *backend*, de manera que esa comunicación se realice con los estándares definidos por el estilo de arquitectura *Rest* [18]. En el caso de Pack2Fly, se ha definido una *API Rest* implementado en JavaScript, denominado server.js. El código de la versión 1.0 es el siguiente:

```
1 let express = require('express')
2 let http = require('http')
3 let app = express()
4 paquetes = require('./Code/app')
5 var cors = require('cors')
6
7
8 app.use(function(req, res, next) {
9     res.header("Access-Control-Allow-Origin", "*");
10     next();
11 });
12
```

```
app.get('/packs',async (req, res) => {
    const Origin = req.query.Origin;
    const Destiny = req.query.Destiny;
    const Checkin = req.query.Checkin;
    const CheckOut = req.query.CheckOut;
    const people = req.query.people;

    let result = await get_data(Origin,Destiny,Checkin,CheckOut,people);

    res.send(result);
    return res;
})

async function get_data(origin,destiny,checkout,checkin,people){
    let result = await paquetes.get_packs(origin,destiny,checkin,checkout,parseInt(people));
    return result;
}

http.createServer(app).listen(8000, () => {
    console.log('Server started at http://localhost:8000');
});
```

Si se analiza detenidamente el código, se observa un elemento familiar: app.get('/packs). ¿Suena de algo? Efectivamente, como puede intuirse, cuando en el *frontend* se ejecuta la instrucción this.http.get("http://localhost:8000/packs...), la petición es pillada por ese método de la *API Rest*. En la API se pueden definir, por lo general, métodos tipo *GET,POST, UPDATE y ERASE*, aunque pueden definirse más. En la cabecera de la aplicación, lo más destacable es la ruta que, por así decirlo, escucha el método.

La función *get* para obtener los paquetes es muy sencilla: se llamará de manera asíncrona a la función get-data, que llama al método *get-packs* del *backend*, que ya se vió que devuelve los paquetes generados.

La*API Rest*tiene otra función muy importante: Inicializar el servidor propio para recibir las peticiones. Para ello, se utiliza el puerto 8000, por lo que la dirección es la que se ha visto. De esta manera, el servidor estará configurado para realizar escuchas y resolver las peticiones que reciba.

3.2.3. Páginas

Las páginas en angular son componentes en sí, pero con la peculiaridad de que representan vistas que tendrá la aplicación. La estructura que presenta es muy similar con respecto a los componentes que se definieron anteriormente. Sin embargo, estas páginas van a hacer uso de los servicios de la aplicación. Es la magia de angular. Para verlo, se considera, por ejemplo, el código TypeScript de la página de búsqueda:

```
import ...


@Component({
  selector: 'app-search-form',
  templateUrl: './search-form.component.html',
  styleUrls: ['./search-form.component.css']
})

export class SearchFormComponent implements OnInit {

  Origin: FormControl;
```

```
13   Destiny: FormControl;
14   CheckIn: FormControl;
15   Checkout: FormControl;
16   people: FormControl;
17
18   packs: any[] = [];
19   SearchForm: FormGroup;
20
21   constructor(public usuarioService: UsuariosService,private fb: FormBuilder) {
22           ...
23
24     this.SearchForm = this.fb.group({
25           ...
26     })
27   }
28
29   ngOnInit(): void {
30   }
31
32   async getPacks(): Promise<void>{
33    await this.usuarioService.get_packs(this.SearchForm.get('Origin')?.value,this.SearchForm.get('Destin
34      .then((res: any) => {
35         ...
36      })
37      .catch(err => console.error(err));
38   }
39  ...
40 }
```

Dentro de este código, se deben tener en cuenta 2 principales conceptos: El uso del servicio y la utilización de variables definidas en el formulario.

Para hacer uso de un servicio, es necesario llevar a cabo una serie de pasos. En primer lugar, se debe importar el servicio e inicializarlo en el constructor de la página. Así, podemos realizar llamadas a sus métodos. Una vez realizados estos pasos, simplemente se llamará al método *get-packs* del servicio. Con la sentencia *.then*, que se ejecutará cuando se obtenga el valor, se inicializará la variable *packs* con el valor del resultado. Lógicamente, se debe tener en cuenta algún posible error.

A la hora de poder usar las variables del formulario (comportamiento que será útil para todos los formularios de la aplicación) se debe tener en cuenta una serie de pasos. En primer lugar, se debe establecer un identificador para cada uno de los input del formulario en el HTML:

```
1 <input type="text" id="Origin"  placeholder="Enter Origin" formControlName="Origin">
```

Además, se deberá identificar convenientemente el formulario

```
1 <form class="form-inline" [formGroup]="SearchForm">
2 ...
3 </form>
```

Estos identificadores deben estar asignados al tipo *formControlName*. Una vez hecho esto, se debe modificar el código TypeScript para poder acceder a esos valores. En primer lugar, se importan las funcionalidades necesarias, como paso general.

```
1 import { FormControl, FormGroup, FormBuilder,... } from "@angular/forms";
```

A continuación, se definirán las variables propias en el .TS para almacenar el contenido de los valores introducidos por el usuario.

```
  Origin: FormControl;
  Destiny: FormControl;
  CheckIn: FormControl;
  Checkout: FormControl;
  people: FormControl;
```

Lógicamente, en el constructor se deberá inicializar estas variables para poder asignarles un valor. Para poder realizar la asignación, simplemente se accederá al formulario y se asignará directamente cada variable:

```
    this.SearchForm = this.fb.group({
      Origin: this.Origin,
      Destiny: this.Destiny,
      Checkout: this.Checkout,
      CheckIn: this.CheckIn,
      people: this.people
    })
```

donde fb es un objeto del tipo *FormBuilder* definido en el constructor.

```
constructor(public usuarioService: UsuariosService,private fb: FormBuilder)
```

Este esquema, como ya se ha comentado, es perfectamente aplicable a cualquier formulario que se desee implementar.

Con esto, se ha descrito en líneas generales el *Frontend* de la aplicación. Permitirá entender el comportamiento del resto del código, disponible en el repositorio de *Github* de *Pack2Fly*.

Capítulo 4

Conclusiones y líneas futuras

Un proyecto software en sí mismo supone un reto para cualquier estudiante sin demasiada experiencia. Nos enfrentamos a numerosos retos que exigen lo mejor de nosotros mismos como futuros ingenieros informáticos, aplicando los conocimientos adquiridos en el transcurso de la carrera y por voluntad propia.

No es un camino nada fácil. A lo largo del proyecto, se han producido muchos fallos propios de la falta de experiencia que sin duda han afectado a los tiempos planificados y a todo el proceso de organización del proyecto. Sin embargo, estos errores son parte de la experiencia. A fin de cuentas, el objetivo del TFG es simular un proyecto real, con todas sus implicaciones. Proporciona un primer contacto, permitiéndonos adquirir experiencia que sin duda resultará muy valiosa en el futuro. Por ello, los errores refuerzan el aprendizaje, ya que una vez que los cometes, resulta más difícil que se vuelvan a producir.

Conocer el funcionamiento de una aplicación web es realmente una utilidad realmente beneficiosa para mi futura carrera. Durante la carrera, se obtienen ligeras trazas de los componentes que, de manera conjunta, establecen una aplicación. Sin embargo, al menos en mi caso personal, no se habla de conceptos como *Backend*, *Frontend*, *API Rest*, etc. Al menos, no con la suficiente profundidad como se debería. Es indispensable conocer estos conceptos, ya que se ven en todo tipo de proyectos informáticos. Además, quizás más importante es conocer como interactúan entre sí y sus funciones. Es decir, saber que el *backend* se encarga de obtener los datos y tratarlos, la *API Rest* de interactuar con el *backend* para obtener o enviar datos para / desde el *frontend* y que el *frontend* constituye la interfaz con la que interactúa el usuario, con su lógica y comportamientos asociados. Son conceptos simples e interesantes, pero si nunca se han visto, supone una complicación bastante grande. En mi caso, un retardo de casi 1 mes por no acabar de entender como obtener los datos del *backend* para mostrarlos en el *frontend*.

A pesar de esto, y retomando las reflexiones anteriores, estas dificultades no traen solamente retrasos y estrés, sino también conocimiento indispensable. A fin de cuentas, ha supuesto todo un reto, del que estoy muy orgulloso. Me siento muy realizado de haber completado este último paso en la carrera de Ingeniería Informática, y de haber creado de la nada una aplicación web. A fin de cuentas, nuestro trabajo como ingenieros en precisamente ese: Crear cosas que solucionen problemas y necesidades reales. Esto supone, sin duda, una de las mayores satisfacciones que puede tener un profesional.

Creo firmemente que este proyecto puede estar involucrado en mi futuro. El tema principal me ha resultado muy interesante desde que por primera vez pensé en ello. ¿Qué pasaría si se pudiera realizar una aplicación completamente funcional y ambiciosa que usara sus propios datos y que permitiera adquirir en su propia plataforma los paquetes? ¿Podría llegar a ser una aplicación realmente potente? ¿Podría suponer un producto entorno al cuál se pudiera crear toda una

empresa?

En el momento de redacción de esta memoria, se ha logrado una aplicación que es capaz de generar los paquetes, que ofrece al usuario toda la información del mismo, permite a los usuarios seleccionar los paquetes top o los recomendados y contactar ante cualquier duda. Sin embargo, el objetivo final no se limita a eso. Se pretende que se cree un sistema de autenticación de usuarios que permita crear toda una comunidad de usuarios en la aplicación. Además, que se puedan guardar los paquetes cuando el usuario lo decida, que puedan comentar en los paquetes y asignarles una valoración y agregar filtros de refinamiento de búsquedas. Las posibilidades son infinitas, y sin lugar a dudas, el potencial de este proyecto es realmente enorme.

Capítulo 5

Conclusions and Future Work

A software project in itself is a challenge for any student without much experience. We face numerous challenges that demand the best of ourselves as future computer engineers, applying the knowledge acquired in the course of the career and of our own free will.

It is not an easy path. Throughout the project, there have been many failures due to lack of experience that have undoubtedly affected the planned times and the entire project organization process. However, these errors are part of the experience. After all, the objective of the TFG is to simulate a real project, with all its implications. It provides a first contact, allowing us to gain experience that will undoubtedly prove invaluable in the future. Therefore, mistakes reinforce learning, since once you make them, it is more difficult for them to happen again.

Knowing how a web application works is really a beneficial utility for my future career. During the race, slight traces of the components are obtained that, together, establish an application. However, at least in my personal case, concepts like Backend, Frontend, API Rest, etc. are not discussed. At least not deep enough as it should be. It is essential to know these concepts, since they are seen in all kinds of computer projects. Also, perhaps more important is to know how they interact with each other and their functions. That is, knowing that the backend is in charge of obtaining the data and treating it. Then, that the API Rest interacts with the backend to obtain or send data to / from the frontend. Finally, the frontend constitutes the interface with which the user interacts, with its associated logic and behaviors. They are simple and interesting concepts, but if they have never been seen, it is quite a complication. In my case, a delay of almost one month for not fully understanding how to get the data from the backend to show it on the frontend.

Despite this, and taking up the previous reflections, these difficulties do not only bring delays and stress, but also indispensable knowledge. At the end of the day, it has been quite a challenge, of which I am very proud. I feel very fulfilled to have completed this last step in the Computer Engineering degree, and to have created a web application from scratch. Ultimately, our job as engineers is precisely that: Creating things that solve real problems and needs. This is, without a doubt, one of the greatest satisfactions that a professional can have.

I firmly believe that this project can be involved in my future. The main theme has been very interesting to me since I first thought about it. What if you could build a fully functional and ambitious application that uses your own data and allows you to purchase packages on your own platform? Could it become a really powerful application? Could you suppose a product around which an entire company could be created?

At the time of writing this report, an application that is capable of generating the travel packages has been achieved. That is, it offers the user all the information about packs, allows users to select the top or recommended packages and contact them with any questions. However, the end goal is not limited to that. It is intended that a user authentication system be created that allows

the creation of a whole community of users in the application. In addition, the packages can be saved when the user decides, they can comment on the packages and assign them a rating and add search refinement filters. The possibilities are endless, and without a doubt, the potential of this project is truly enormous.

Capítulo 6

Presupuesto

Lógicamente, en el ámbito de este proyecto, todas las funcionalidades han sido creadas con el uso de herramientas de gratuitas. Es por ello que no se ha invertido una cantidad económica en el desarrollo de esta aplicación.

Teniendo en cuenta que *Pack2Fly* es una aplicación web que actualmente se ejecuta localmente, el primer servicio que se plantea contratar es una plataforma de servidores que permitan alojar la aplicación. Es un servicio que se conoce como *Hosting*. https://www.ciudadano2cero.com/que-es-un-hosting/ Se ha optado por elegir *Bluehost*, debido a su precio y todas las funcionalidades que ofrece. El precio según si sitio web [19] es de 9€/mes.

Otro gasto a tener en cuenta es el uso de las *APIs*. Actualmente, se está utilizando las versiones gratuitas de las mismas. Sin embargo, para un mayor tráfico de peticiones, se requeriría una suscripción *premium* en ambas. Así, el precio para la *API* de *Hotels.com* es de 40,07 €/mes. Por otro lado, la *API* de *Skyscanner* únicamente posee una opción gratuita, por lo que no supondría un gasto mensual.

Por último, es necesario tener en cuenta los gastos de desarrollo, es decir, el sueldo base que cobraría por el trabajo realizado. El sueldo básico de un programador junior sin experiencia es de aproximadamente 1.600€ mensuales aproximadamente [20]. Por tanto, el importe total por el trabajo de 4 meses sería de 6400€.

6.1. Desglose

Concepto	Coste Mensual	Precio Total
Desarrollo de la Aplicación	1600€	6400€
APIs	40,07	160,28€
Servicio de Hosting	9€	36€
Precio Total	**1649,07€**	**6.596,28€**

Tabla 6.1: Coste Total del Proyecto

Apéndice A

Manual de Uso

A.1. Manual de Uso

De cara al usuario, es conveniente la creación de un manual de uso para aprender todas las funcionalidades que ofrece la aplicación.

1. Cómo usar el Buscador
   El buscador constituye la principal funcionalidad de la aplicación. Su uso es bastante sencillo. En primer lugar, debe introducir el origen y el destino. Nótese que es posible que alguno o ambos no se encuentren en el sistema, de modo que no se arrojará un resultado.

   A continuación, introduzca las fechas de ida y vuelta y el número de viajeros. Para realizar la búsqueda, pulse en buscar.

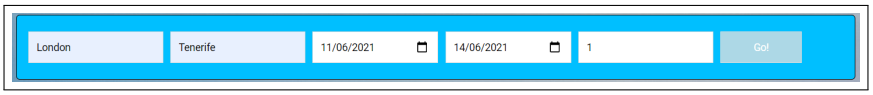


Figura A.1: Realizando una búsqueda

Una vez realizada, se mostrarán los resultados obtenidos. Aquí usted tiene 2 opciones:

- Guardar el Paquete: El paquete será guardado en el sistema, de manera que se ofrecerá en futuras búsquedas.
- Comprobar el Paquete: Mostrará toda la información del paquete en detalle.

Figura A.2: Opciones del paquete

Adicionalmente, se dispone de filtros para ordenar los resultados. Usted puede seleccionar entre:

- Más barato primero
- Calidad-Precio
- Más caro primero
- Mejor puntuación primero

2. Top Packs y Recommended Packs

   En este caso, funcionará muy parecido al buscador. Pulse el botón de búsqueda para obtener el listado de paquetes recomendados y más guardados cuando se hayan obtenido.

3. Contacto

   A la hora de realizar el contacto, existen 2 opciones:

   - Formulario: Para poder completar el formulario, se deberá introducir un título, el correo del usuario que está enviando el mensaje (asegurarse de que el correo es correcto) y el mensaje a enviar. A continuación, pulsar enviar.
   - Redes Sociales: En la propia página web, podrá acceder en cualquier momento a las redes sociales de la empresa. Para ello, pulse sobre el icono de la red social en la parte inferior de cada página.

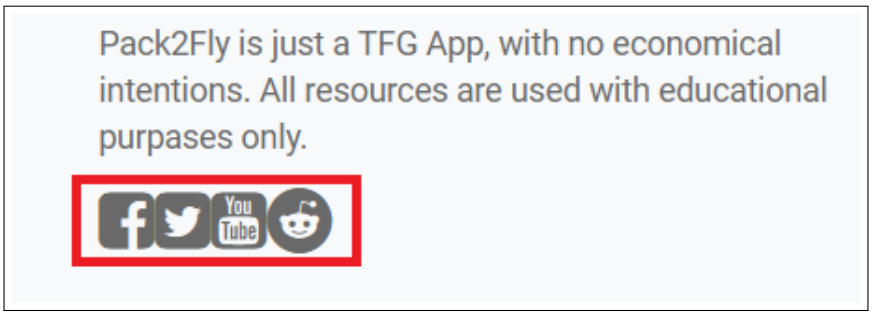


Figura A.3: Redes Sociales de la Aplicación

4. Sobre Nosotros Ponemos a disposición suya información sobre la empresa, por si fuera de su interés.

Apéndice B

Figuras y Elementos Gráficos

B.0.1. Diagramas de Caso de Uso

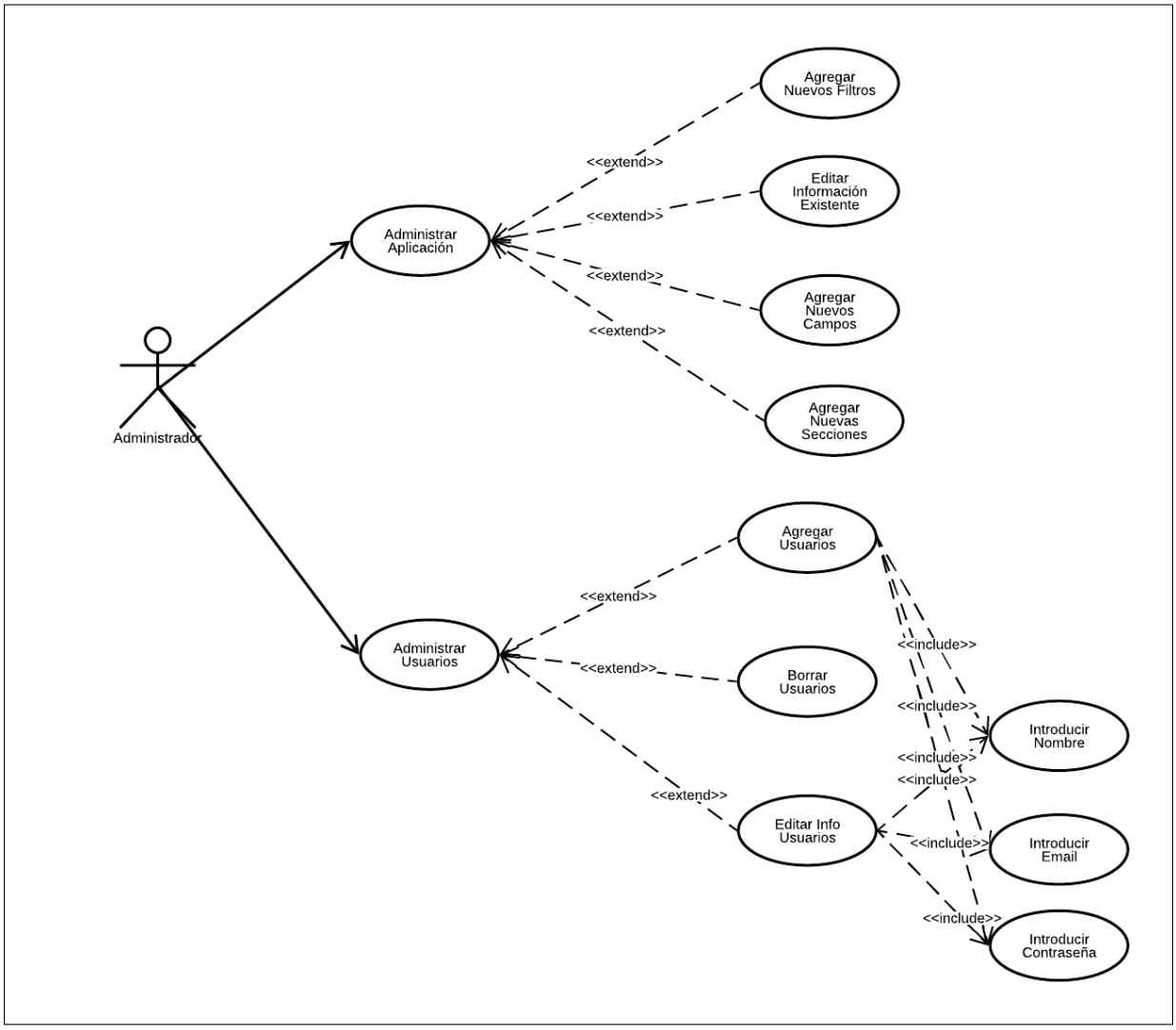


Figura B.4: Diagrama de Caso de Uso para Administrador

El primer diagrama B.4 representa las distintas acciones que puede llevar a cabo el administrador del sistema. Puede administrar la aplicación, agregando filtros, editando la información

existente, o agregando nuevos campos u acciones a los paquetes. Además, también se encarga de la administración de los usuarios. Esto implica la agregación y supresión de los mismos y la edición de su información: nombre, *email*, contraseña, etc.

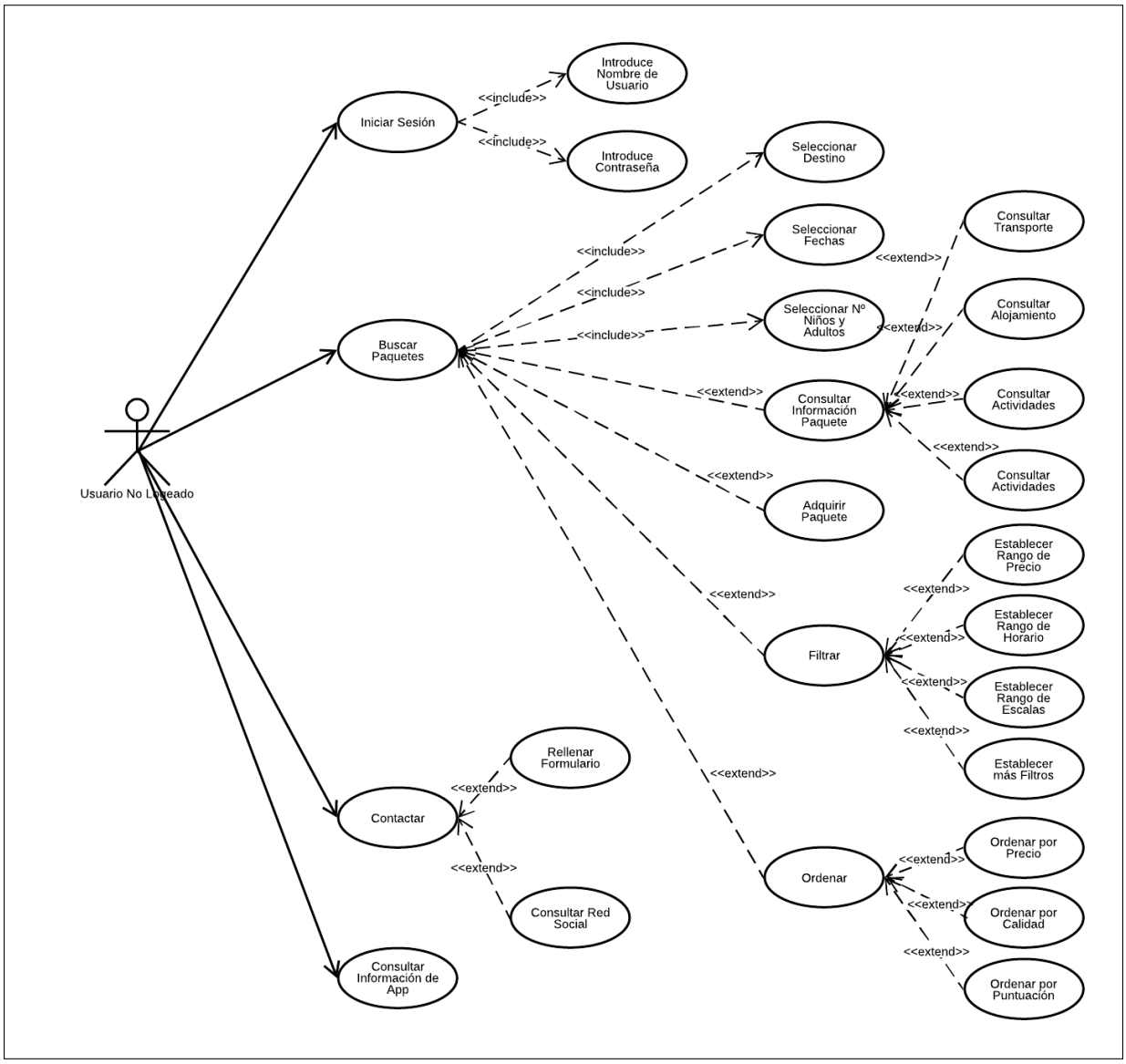


Figura B.5: Diagrama de Caso de Uso para Usuario No Logeado

El segundo diagrama B.5 representa las funcionalidades asociadas al usuario que no ha iniciado sesión en la aplicación. Este tipo de usuario puede iniciar sesión aportando su nombre y contraseña. También puede buscar paquetes aportando los campos necesarios para que pueda realizarse: destino, fechas, nºniños y nºadultos. Podrá consultar la información del paquete, adquirirlo, filtrarlos u ordenarlos. Además, podrá contactar en caso de duda o problema y acceder a la información sobre la aplicación.

Figura B.1: Pantallas del Prototipo Móvil 1

Figura B.2: Pantallas del Prototipo Móvil 2

Figura B.3: Pantallas del Prototipo Móvil 3

ID	Email	User Name	Password	Saved Packs	Puntuation	Profile Image	User Since	Last Seen	Nº Comments
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX
0	xxxx@yyy.zzz	XXXXX	YYYYY	151,512	9	yyy.xxx	DD MM YYYY	DD MM YYYY	XXX

Tabla B.1: Diseño de Base de Datos para Usuarios

ID	Precio	Directo	Salida	Compañía	Lugar de Salida	País	Ciudad	Lugar de Llegada	País	Ciudad
00121	120€	true	2021-09-01	Ryanair	Tenerife Norte	España	Santa Cruz	El Hierro	España	Valverde
00121	120€	true	2021-09-01	Ryanair	Tenerife Norte	España	Santa Cruz	El Hierro	España	Valverde
00121	120€	true	2021-09-01	Ryanair	Tenerife Norte	España	Santa Cruz	El Hierro	España	Valverde

Tabla B.2: Diseño de Base de Datos para Transporte

ID	Lugar	Nombre	Estrellas	Dirección	Puntuación	Escala	Precio Actual	Precio Anterior
00543	Moscú, Rusia	Moscu Hotel	5	Moscu 20 str. 1	4.3	5	169€	200€
00543	Moscú, Rusia	Moscu Hotel	5	Moscu 20 str. 1	4.3	5	169€	200€
00543	Moscú, Rusia	Moscu Hotel	5	Moscu 20 str. 1	4.3	5	169€	200€

Tabla B.3: Diseño de Base de Datos para Alojamientos

Bibliografía

[1] *ISTAC*. http://www.gobiernodecanarias.org/istac/. Accessed: 2021-03-14.

[2] Yolanda Acosta. *El 86 de los españoles reserva sus viajes a través de Internet*. https://www.hosteltur.com/comunidad/nota/018774_el-86-de-los-espanoles-reserva-sus-viajes-a-traves-de-internet.html. Accessed: 2021-03-14.

[3] Cristina Delgado. *Internet se come el viaje organizado*. https://elpais.com/economia/2015/08/23/actualidad/1440355929_902856.html. Accessed: 2021-03-16.

[4] Xavier Canalis. *Ranking de apps de viajes más usadas en España: quién sube y quién baja*. https://www.hosteltur.com/131315_ranking-de-apps-de-viajes-mas-usadas-en-espana-quien-sube-y-quien-baja.html. Accessed: 2021-03-16.

[5] Julián Pérez Porto y María Merino. *Definición de Boceto*. Accessed: 2021-03-25. URL: https://definicion.de/boceto/#:~:text=Derivado%20del%20t%C3%A9rmino%20italiano%20bozzetto,que%20arrojar%C3%A1%20un%20resultado%20final..

[6] Justinmind. *JustInMind*. https://www.justinmind.com/?utm_medium=cpc&utm_source=google&utm_campaign=1063145459&utm_term=justinmind_e&gclid=Cj0KCQiAyoeCBhCTARIsAOfpKxjIxs5uZJvcIGkaAs0CEALw_wcB. Accessed: 2021-03-16.

[7] Desconocido. *Definición de Escenario*. Accessed: 2021-03-28. URL: http://sedici.unlp.edu.ar/bitstream/handle/10915/4057/4_-_Escenarios.pdf?sequence=5&isAllowed=y#:~:text=Un%20escenario%20es%20una%20descripci%C3%B3n,acci%C3%B3n%20a%20llevar%20a%20cabo.&text=Los%20escenarios%20describen%20actores%2C%20objetivos%20y%20episodios..

[8] Lola Cárdenas Luque. *Definición de Entidad*. http://www.formauri.es/arrobamasmas/Cursos/index.php?apdo=05&curso=51&cap=2. Accessed: 2021-03-29.

[9] Desconocido. *Modelo Entidad-Relación*. https://es.wikipedia.org/wiki/Modelo_entidad-relación. Accessed: 2021-03-29.

[10] Lucidchart. *Qué es un modelo de base de datos*. https://www.lucidchart.com/pages/es/que-es-un-modelo-de-base-de-datos. Accessed: 2021-04-07.

[11] Enrique Rus Arias. *Modelo Relacional: Qué es, definición y concepto*. https://economipedia.com/definiciones/modelo-relacional.html. Accessed: 2021-04-08.

[12] *RapidApid*. https://rapidapi.com/. Accessed: 2021-04-08.

[13] Nestrategia. *Backend*. Accessed: 2021-04-24. URL: https://nestrategia.com/desarrollo-web-back-end-front-end/#:~:text=En%20otras%20palabras%2C%20el%20Back,la%20comunicaci%C3%B3n%20con%20el%20servidor..

[14] Jose Mª Baquero García. *Promesa*. Accessed: 2021-04-24. URL: https://www.arsys.es/blog/promesas-javascript/#:~:text=Una%20promesa%20es%20un%20objeto,que%20posponerse%20en%20el%20tiempo..

[15] *Frontend.Definición*. https://www.arimetrics.com/glosario-digital/frontend. Accessed: 2021-05-20.

[16] *Angular*. https://angular.io/docs. Accessed: 2021-06-05.

[17] *Servicios en Angular.Definición*. https://codingpotions.com/angular-servicios-llamadas-http. Accessed: 2021-05-20.

[18] *API Rest.Definición*. https://rockcontent.com/es/blog/api-rest/. Accessed: 2021-05-20.

[19] *Bluehost*. https://www.bluehost.com/special/homeaff?utm_source=www.top10.com&utm_medium=affiliate&utm_campaign=affiliate-link_naturalcpa_Directories. Accessed: 2021-05-31.

[20] *Sueldo de Programador Junior en España*. https://es.indeed.com/career/programador-junior/salaries. Accessed: 2021-05-31.