UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de una aplicación móvil para la búsqueda de planes de entretenimiento en una ciudad

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Francisco de Borja Alonso Melero

Tutor: Manuela Albert Albiol

2019/2020

Dedicado a mis padres y a mis abuelos
por haber sido un punto de apoyo
en todo momento

Gracias a la música por haber hecho posible este trabajo.

Resumen

En esta memoria se presenta el resultado del desarrollo de una aplicación móvil para la búsqueda y realización de planes de entretenimiento en la ciudad en la que residas. Se pretende ver cuáles son las necesidades reales de los posibles usuarios para lanzar un producto al mercado con las funcionalidades que cubran las necesidades básicas de estos. Se ha seguido el método Lean Startup, una metodología que tiene como objetivo acortar los ciclos de desarrollo de productos.

Palabras clave: aplicación móvil, ocio, planes, Ionic, aplicación híbrida Firebase, NoSQL, database, MVP, Lean Canvas, Lean Startup, prototipo, metodología ágil

Abstract

This report presents the result of the development of an MVP for the search and realization of entertainment plans in the city in which you reside. It is intended to see results of the real needs of possible users to launch a product on the market with the functionalities that cover their basic needs. The Lean Startup method has been followed, a methodology that aims to shorten product development cycles.

Keywords: mobile app, leisure, plans, Ionic, hybrid app, Firebase, NoSQL, database, MVP, Lean Canvas, Lean Startup, prototype, agile methodology

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

Hoy en día, vivimos en un consumismo constante. Según el INE (Instituto Nacional de Estadística), una persona gastaba el 20% de sus ingresos hace 50 años. Ahora, el 40%. El doble. Esto no significa que una persona gaste el doble que hace 50 años. Simplemente hemos cambiado nuestra forma de gastar. Han aumentado los gastos en actividades de ocio, pero han disminuido en otros ámbitos, como en el de la alimentación o el de la ropa, donde hemos pasado de gastar un 15% a un 5%.

Volviendo un poco a la época actual, en 2018, las familias españolas tienen un gasto medio de 1.643 € en ocio y cultura, por encima de otros gastos importantes, como los 1.463 € en vestido y calzado o los 1.371 € en mobiliario.

Existen dos formas de diferenciar el ocio. Por una parte, el que se realiza en conjunto, en el cual destacan las actividades al aire libre o las que parten desde el domicilio familiar. En cambio, si hablamos del ocio individual, el *pódium* lo lideran las compras, los viajes y los restaurantes, seguidos de la lectura y escuchar música. Es extraño que, algo tan común como la salida a bares o discotecas, solo sea un 22,6% los que se decanten por este tipo de planes.

Hoy en día (yo lo vivo personalmente muy a menudo), la gente está falta de ideas a la hora de realizar actividades de ocio ya sea con su pareja, sus amigos, sus familiares, etc. Por otro lado, los negocios dedicados al ocio invierten una gran cantidad de dinero en publicitar sus servicios con el fin de conseguir clientes y, así, aumentar sus beneficios. En este proyecto se pretende desarrollar una aplicación que ayude a ambos perfiles; permitiendo a los negocios dedicados al ocio ofrecer sus actividades y ofreciendo a los usuarios una herramienta que ofrezca alternativas para el ocio.

1.1. Motivación

La idea del TFG que se presenta surge en una de las clases optativas que se cursa en 4º del Grado de Ingeniería Informática. Concretamente, en la asignatura Desarrollo de Aplicaciones para Dispositivos Móviles (DADM). La asignatura tenía como finalidad desarrollar un proyecto en grupo de una aplicación móvil en un tiempo de 3 meses. Éramos un equipo de 5 personas y el problema residía en qué desarrollar. Salieron varias ideas de cada uno. Una de las ideas del autor de este TFG fue la relacionada con el TFG que se presenta. Finalmente, el grupo optó por otra de las ideas que surgieron. Por lo que se decidió utilizar la idea en más adelante.

Yo me quedé con la idea en la mente. Día sí día también. Soñaba con ella. Sabía que se podía llevar a negocio. Era algo que tenía clarísimo. Empecé a pensar en cómo lo haría. Pensaba y pensaba en muchas cosas. "¿Cómo lo hago? ¿Crees que tienes los conocimientos suficientes? Yo la usaría sin lugar a duda, pero ¿y el resto?". Era la primera vez que empezaba algo totalmente de cero. No tenía ni idea de por dónde empezar. ¿Me lanzo a desarrollar como un loco como hizo el de Facebook a ver qué sale? Sí, pero ¿qué pasa si por hacerlo rápido y mal lo tiro "todo" al traste? Quería y tenía ganas de hacer las cosas bien desde el principio por primera vez en mi vida.

1.2. Objetivos

El objetivo principal del TFG es diseñar, desarrollar y lanzar al mercado una aplicación que permita al usuario estar al tanto de planes de ocio en su ciudad de residencia. Dentro de este objetivo, se identifican los siguientes objetivos específicos:

- Desarrollar una plataforma a la cual pueda acceder cualquier persona entre 16 y 60 años que resida en España.
- Permitir a los negocios aumentar su clientela publicando planes personalizados dentro de la plataforma.
- Monetizar la plataforma, ya sea cobrando una comisión a los usuarios o una cuota mensual a los negocios que se publiciten en ella.
- Comenzar a operar en la ciudad de Valencia y analizar la posible expansión a otras ciudades españolas e incluso a otros países europeos si los primeros experimentos en Valencia son satisfactorios.

1.3. Impacto esperado

Además de servirme para poner en práctica lo aprendido durante el grado y poder desarrollar nuevas habilidades a lo largo del proyecto, la aplicación desarrollada afectará positivamente a dos usuarios diferentes. Por una parte, a los ciudadanos habituales de una ciudad, ya que se les ofrecerá una alternativa para disfrutar de los diferentes eventos y planes de entretenimiento en la ciudad en la que residen.

Y, por otra parte, a los negocios, los cuales podrán publicitar sus actividades dentro de la plataforma, teniendo así otra vía por la que obtener clientes y aumentar sus beneficios.

1.4. Metodología

Para llevar a cabo este proyecto, se ha aplicado el método *Lean Startup*, explicado en detalle más adelante. Básicamente se centra en reducir los tiempos de desarrollo de productos adoptando una combinación de experimentación basada en hipótesis para medir el progreso, lanzamientos de productos iterativos para obtener retroalimentación por parte de los clientes y aprendizaje validado para medir cuánto se ha aprendido.

En primer lugar, se analizará el problema y se observará cómo está el mercado actualmente de aplicaciones de este estilo para ver cómo solucionan dicho problema, ver qué es lo que ofrecen a los usuarios y lo que nos puede diferenciar de ellas para obtener una ventaja competitiva.

Seguidamente, se lanzarán unas encuestas para obtener información más detallada acerca de las necesidades de los posibles usuarios y de cómo actúan.

A partir de los resultados de las encuestas, se creará un MVP para obtener *feedback* por parte de usuarios reales y así poder mejorar el producto en base a las necesidades del usuario final.

1.5. Estructura

Capítulo 1: En este capítulo se presenta la situación actual en la sociedad, la motivación que ha hecho posible el desarrollo de este proyecto, los objetivos que se pretenden alcanzar con el desarrollo de la aplicación, el impacto que se espera tener en los diferentes usuarios y la metodología a utilizar para que el proyecto sea lo más llevadero posible

Capítulo 2: En este capítulo se analizará cómo esta el mercado de aplicaciones de características similares a la que se va a plantear en este TFG. Se mencionarán las características de cada una de ellas y las diferencias que se observan con respecto a la aplicación planteada.

Capítulo 3: En este capítulo se presentarán los resultados que se han obtenido de las diferentes encuestas realizadas a posibles usuarios realizadas para obtener información acerca de sus necesidades básicas y, así, desarrollar un producto que cumpla sus deseos. También se presentará un resumen de los diferentes aspectos a tener en cuenta previos a realizar cualquier acción.

Capítulo 4: En este capítulo, se analizarán las diferentes tecnologías que se han valorado para el desarrollo de la aplicación y se explicará el por qué se han elegido las que se han elegido. También se mostrarán los bocetos, los prototipos y los diagramas realizados previos al desarrollo para que este sea más llevadero.

Capítulo 5: En este capítulo se explicarán los pasos que se han seguido para que el desarrollo fuera lo más limpio posible a la vez que se presentarán las diferentes partes de la aplicación y cómo se comunican entre sí.

Capítulo 6: En este capítulo se analizan las conclusiones obtenidas al finalizar este trabajo al igual que todo lo aplicado y aprendido a lo largo de este.

Capítulo 7: Por último, en este capítulo se presentan algunas de las acciones que se van a realizar en un futuro próximo.

2. Estado del arte

En esta sección se va a realizar una investigación sobre otras aplicaciones que existen en el mercado actualmente como competencia y se explicará qué es lo que hace cada una de ellas y las diferencias que tienen con la aplicación que se va a desarrollar en este proyecto.

Para considerar que una aplicación es competencia, se han estudiado varios aspectos de las distintas aplicaciones. En primer lugar, si dicha aplicación ofrece alternativas a la hora de buscar cosas que hacer en las distintas ciudades alrededor del mundo. En segundo lugar, si dicha aplicación es lo suficientemente popular

2.1. Fever

Fever[1] es una plataforma social que ofrece como servicio ofertas de ocio en tu ciudad o comunidad local a la vez que conecta a los usuarios con gustos parecidos. Dentro de la plataforma, puedes comprar *tickets* para eventos de diferentes categorías: festivales, teatros, gastronomía…

Actualmente, se encuentra operativa en distintas ciudades de todo el mundo, como Madrid, París o Los Ángeles. Entre todas estás ciudades, cuentan con más de 10.000 experiencias y una red de 12 millones de usuarios activos a la semana.

Como principales ventajas y características de la *app*, tenemos:

- Recomendaciones de experiencias personalizadas basadas en tus intereses.
- Potente filtrado para ver qué planes están cerca de ti y los próximos eventos disponibles a tu alrededor.
- Posibilidad de guardar tus planes favoritos
- Pago con facilidad y de manera segura.

2.2. Airbnb

Airbnb[2] es mundialmente conocida por ser una plataforma que permite a los usuarios publicitar sus propiedades para un posible alquiler por parte de los huéspedes a cambio de una remuneración económica.

Además del servicio mencionado, cuenta también con un apartado de experiencias dentro de la plataforma. A través de esta vertiente, la gente local tiene la posibilidad de diseñar y organizar una actividad que ofrece visitas y clases distintas a las habituales. De esta manera, muestra su ciudad, oficio o cultura a través de una experiencia.

El funcionamiento es simple. La gente local publica la experiencia que ofrece introduciendo una serie de datos, como la ciudad en la que se va a organizar dicha experiencia, el tema que engloba e información básica, como el idioma principal en el

[1] https://feverup.com/

[2] https://www.airbnb.es/s/experiences

que se va a llevar a cabo, el público al que va dirigido y tu experiencia previa. Por último, basta con subir unas cuantas imágenes para atraer la atención del posible público.

Por otro lado, el usuario que quiera disfrutar de una experiencia puede acceder tanto a la página *web* como a la aplicación móvil donde introducirá la ciudad donde quiere realizar la experiencia y las fechas. Así, se le listarán un listado de experiencias, donde tendrá la posibilidad de filtrar por categoría. Una vez seleccionada la experiencia deseada, lo único que faltaría es elegir la hora deseada y hacer el pago a través de la misma plataforma.

2.3. Meetup

Meetup, a diferencia de las dos plataformas mencionadas anteriormente, está más enfocada a grupos. Permite a sus miembros reunirse en la vida real por intereses en común, como deporte, cultura, tecnología…

El registro tanto en la plataforma *web* como en la aplicación móvil es gratuito para los usuarios. Los beneficios son obtenidos por la suscripción que se cobra a los organizadores de grupos.

Se puede hacer uso de Meetup de dos maneras: descubriendo grupos o creándolos. De la primera forma, destacar que la idea principal es encontrar eventos de grupos locales donde el usuario podrá conocer a gente, probar cosas nuevas o simplemente hacer más de lo que le gusta. En cuanto a la segunda, la creación de grupos de Meetup sirve para hacer nuevos amigos, perseguir pasiones y convertir aficiones en algo más profesional. Lo único que hay que hacer es crear el grupo indicando de qué va a tratar, quién debería unirse y de qué tratarán los eventos de dicho grupo. A partir de ahí, empezar a planificar el primer evento con ayuda de un equipo de organización.

2.4. Sharify

Sharify es una aplicación móvil donde el usuario puede descubrir qué pasa en su ciudad en tiempo real, encontrando gratis miles de eventos y personas con quien compartirlos.

En caso de que quieras buscar eventos, hay 2 opciones. La primera es visualizar en un mapa qué eventos se están llevando a cabo por el resto de los usuarios en ese momento con la posibilidad de unirse a dicho evento, pudiendo chatear con los asistentes. Como segunda opción, está el poder visualizar los eventos programados para hoy o que se realizarán próximamente.

Por otra parte, el usuario tiene la opción de conocer gente nueva. Como primera opción, el usuario puede publicar un plan para que otros usuarios de la plataforma se unan. Los pasos para hacerlo son sencillos, lo que permite publicar el plan en un par de *clicks*. Basta con indicar los detalles del plan: qué, cuándo y dónde y esperar a que otro usuario se una. Como segunda opción, la aplicación cuenta con Comunidades, es decir, grupos de usuarios que comparten el mismo *hobby* y organizan eventos para realizar entre todos los usuarios de esa comunidad.

Por último, Sharify también da la opción a negocios y a gente local que organiza eventos a promocionarse dentro de la aplicación, ampliando así la oferta dentro de la plataforma. Lo único que hay que hacer es registrar el negocio en la página *web* introduciendo una serie de datos de contacto y del negocio en cuestión.

2.5. Shoelace

Shoelace es una aplicación desarrollada por Area 120, taller experimental de Google. De momento, está operativa en la ciudad de Nueva York y su principal objetivo es el de encontrar actividades en grupo que compartan tus mismos intereses.

El hecho de que un gigante tecnológico como Google esté haciendo experimentos con este tipo de aplicaciones, nos dice que aún queda mucho terreno por explorar en este ámbito.

Shoelace también tiene un ámbito más pequeño y concreto para sus grupos, aunque la diferencia principal es que se centra en actividades y lugares concretos**.**

Actualización: a fecha 12 de mayo de 2020, tal y como está la situación con respecto al COVID-19, Google ha decidido cerrar el servicio de Shoelace al no superar la fase de pruebas. No hemos podido hacer uso de este servicio ya que solo estuvo disponible en la ciudad de Nueva York durante su corto año de vida. Debido a esto, no sabemos su funcionamiento real. Podemos asumir que, por las capturas de pantalla que se han visto en su página *web* (ya cerrada), funciona muy parecida a Sharify.

2.6. Couchsurfing

Couchsurfing es una plataforma donde un miembro (el anfitrión) aloja a gente en su casa cuando viajan a su ciudad. Así, los viajeros pueden compartir experiencias auténticas conociendo a los lugareños y a otros viajeros.

La propia plataforma cuenta con una vertiente llamada Couchsurfing Hangouts, donde se pueden encontrar fácilmente otros viajeros cerca de tu ubicación. Su funcionalidad es simple, puedes publicar qué planes tienes o buscar a gente a tu alrededor para solicitar unirte a su plan.

A través de una interfaz sencilla e intuitiva, puedes consultar quién está usando Hangouts, qué usuarios se reúnen cerca de tu ubicación, elegir una actividad para compartirla con otros viajeros, explorar qué planes están llevando a cabo los demás con la posibilidad de contactar con ellos para unirte y, una vez dentro, planear cuándo y dónde quedar.

3. Análisis del problema

En este apartado, se van a analizar los datos obtenidos a través de unas encuestas realizadas a posibles usuarios con los que podremos definir de manera más certera las funcionalidades básicas que va a tener la primera versión del producto y a qué tipo de usuario nos vamos a dirigir en el comienzo. Haremos también un análisis de los diferentes aspectos del producto, desde qué problema solucionamos hasta cómo vamos a monetizar. Además, nos vamos a enfocar en desarrollar los casos de uso que vamos a implementar en esta primera versión.

A través de las encuestas que se han realizado, se ha llegado a determinar el alcance del problema. Como conclusiones de dichas encuestas, muchas personas no saben qué planes realizar en la ciudad en la que viven y acaban optando por los planes de siempre. Los turistas que van a una ciudad sin conocerla, apenas tienen opciones donde poder elegir. Además, muchos negocios carecen de las habilidades para conseguir clientes. Esta sería una manera de proporcionarles una alternativa para conseguir clientes de una forma económica.

3.1. Análisis del perfil y las necesidades de usuario

El objetivo principal de las encuestas es obtener información para validar nuestra hipótesis de si el problema existe realmente y si la gente estaría dispuesta a pagar por una solución a ese problema.

Con los resultados obtenidos de las encuestas, se han definido unas funcionalidades básicas con las que se pretende desarrollar un producto mínimo viable (MVP, del inglés *Minimum Viable Product*), un producto con suficientes características para satisfacer las necesidades de los clientes iniciales y proporcionar retroalimentación para el desarrollo futuro.

También se han llegado a definir a los posibles usuarios de la aplicación. Por una parte, personas entre 18 y 25 años y entre 54 y 55 años serían las más propensas a hacer uso de la aplicación. Aunque se puede apreciar un rango bastante amplio de posibles usuarios. Podemos ver en el siguiente gráfico que, desde los más jóvenes hasta los más mayores, estarían dispuestos a disfrutar de más variedad de planes y eventos.

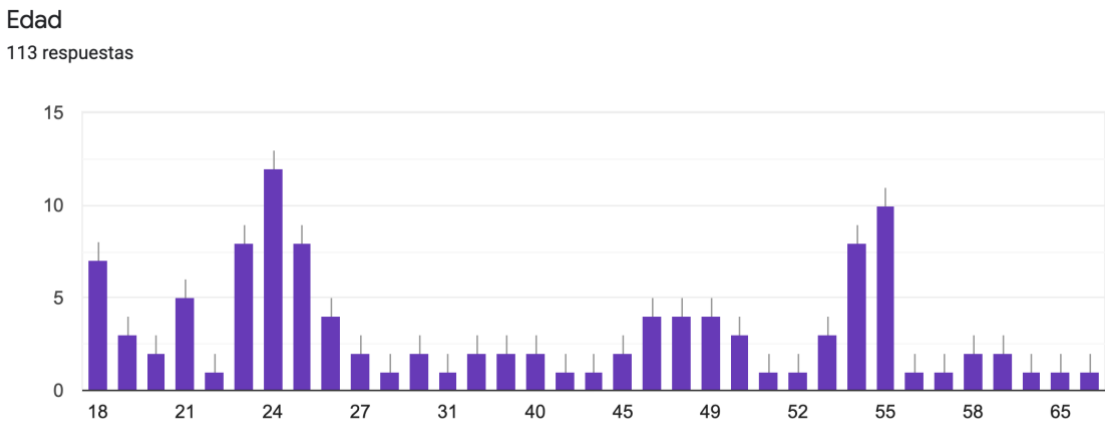


***Ilustración 1**: Gráfico de barras que representa edad de las personas*

En cuanto al sexo, contamos con una gráfica bastante equilibrada, con un 48,7% de mujeres y un 51,3% de hombres. A partir estos datos, podemos concluir que habrá que ofertar todo tipo de planes y eventos sin limitación por sexo para que tanto hombres como mujeres tengan multitud de opciones donde elegir.

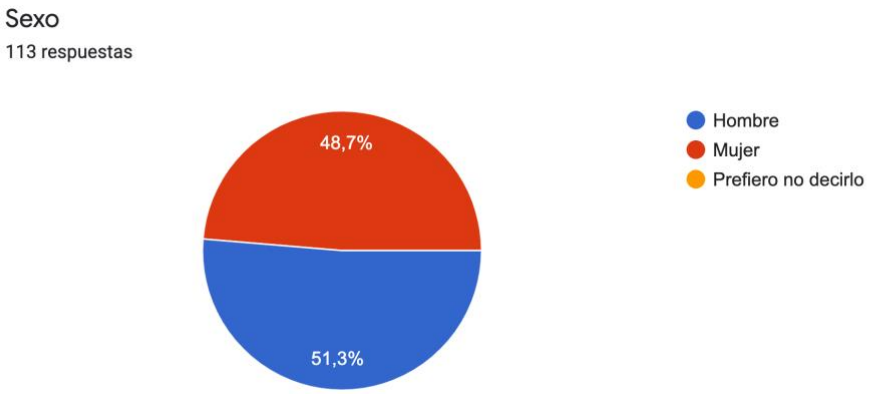


Ilustración 2: *Gráfico representando el sexo de los participantes*

De momento, se empezarán a ofertar planes en la ciudad de Valencia por dos razones: accesibilidad y conocimiento de la zona. Al tratarse del estado inicial del proyecto, los recursos son limitados y acceder a otras ciudades puede ser costoso ya que requeriría tanto inversión en publicidad para dar a conocer el servicio como en personal para llevar a cabo las diferentes acciones en otras ciudades. Por otro lado, la mayoría de mi red de contactos es de Valencia y, al ser una de las grandes ciudades de España, es una buena opción donde empezar a ofrecer cada vez más y más planes.

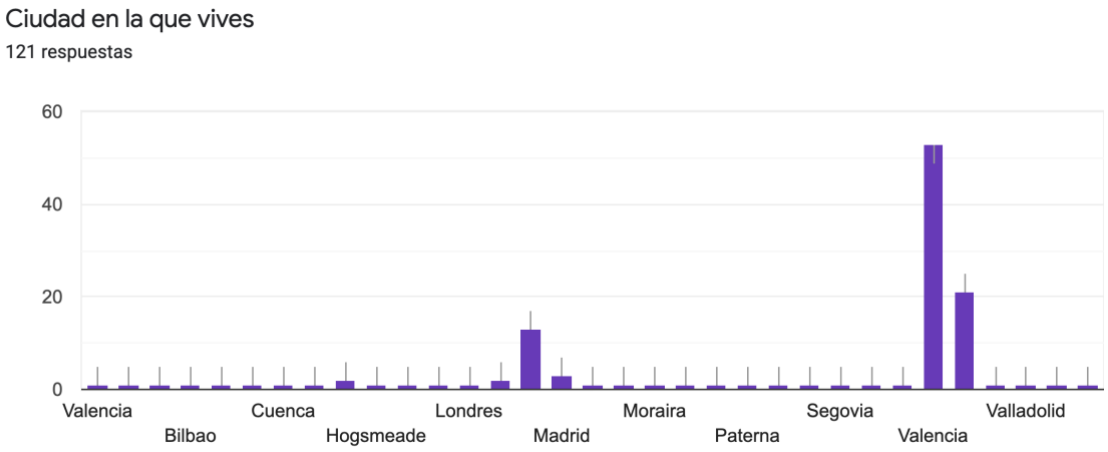


Ilustración 3: *Gráfico de barras de ciudades*

Los planes que se ofrecerían al principio serán más enfocados a realizarse con amigos, ya que el 53,7% de los encuestados respondieron que, con estos, son con los que suelen realizar más planes. A pesar de esto, no se dejarán de lado planes para realizar con familia o pareja, ya que es un gran porcentaje (43,8%) las personas que cuentan con ellos para sus planes.

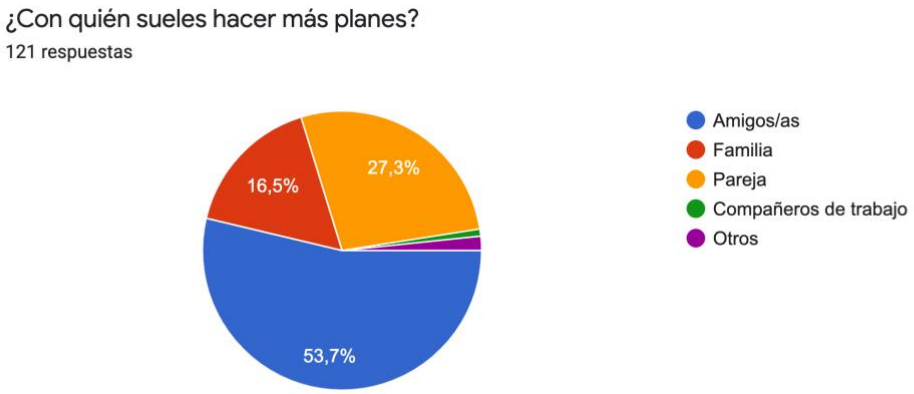


Ilustración 4: *Gráfico circular representando con quién hace más planes la gente*

Se preguntó también qué planes se realizaban con más asiduidad y cuáles eran los que echaban en falta y los que gustaría realizar con más frecuencia. Aquí se potenciarán las categorías más seleccionadas en ambos casos, siendo en el primer caso planes de gastronomía, nocturnos y en grupo, y viajes, conciertos y naturaleza en el segundo.

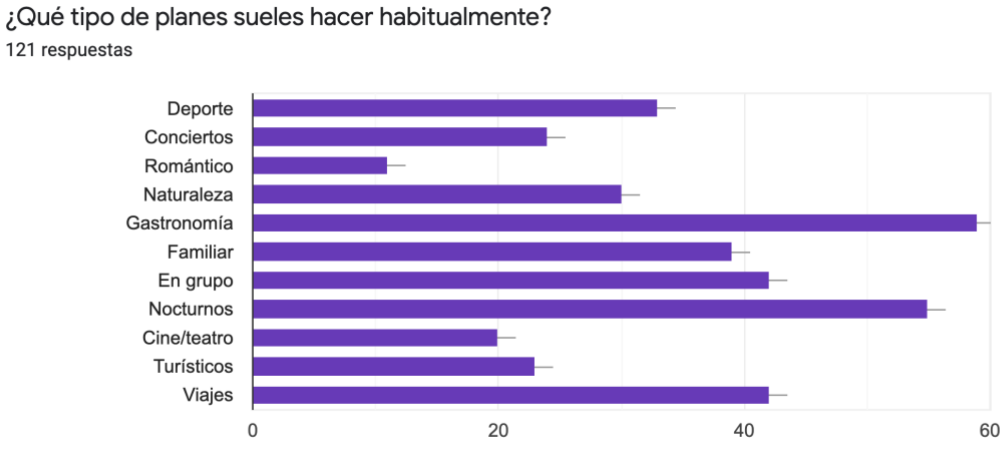


Ilustración 5: *Gráfico de barras que representa las categorías de los planes más realizados*

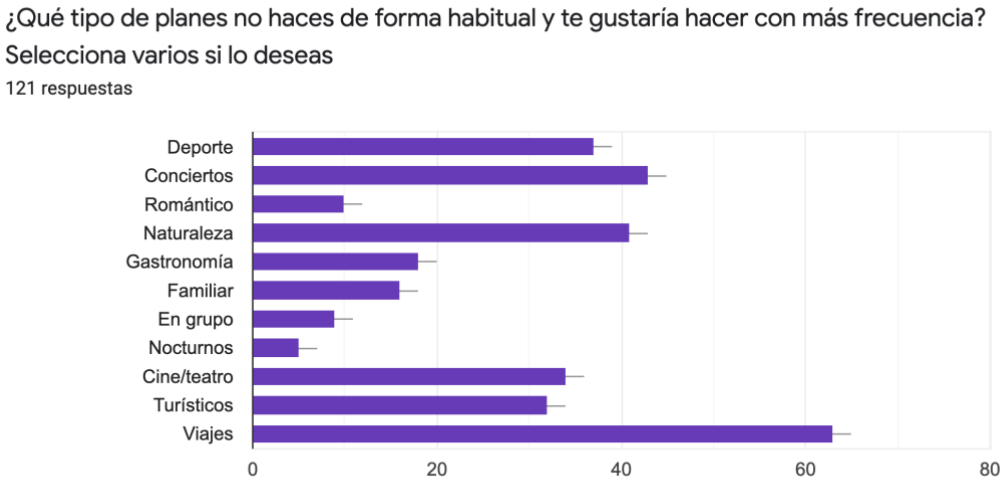


Ilustración 6: *Gráfico de barras que representa las categorías que más echa en falta la gente*

El 86% de los encuestados (104) gastan dinero a la hora de realizar planes. De ese 86%, el 19% (23) suele pagar más de 50 euros por cada plan, el 40,5% (49) entre 20 y 50 euros, el 33,1% (40) menos de 20 euros y es solo el 2,5% (3) los que no gastan nada. Esto nos indica que la gran mayoría está dispuesta a pagar por un plan que realice habitualmente o que no realice y desee vivir nuevas experiencias.

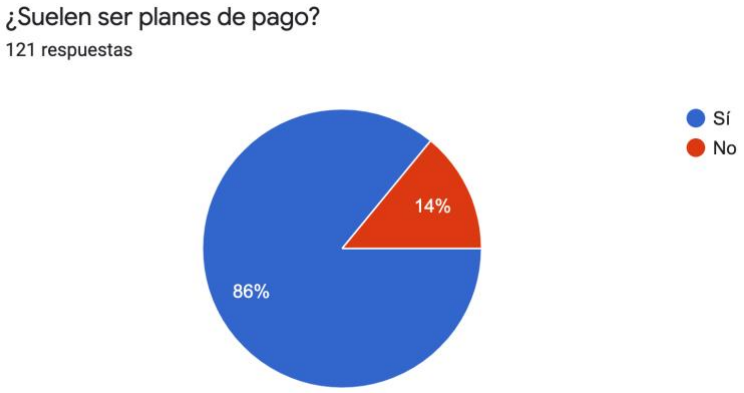


Ilustración 7: *Gráfico circular representando si los planes son de pago o no*

Ilustración 8: *Gráfico circular representando la cantidad aproximada que gasta la gente en los planes que realiza*

Por último, se planteó la posibilidad de realizar planes con gente desconocida. Fueron el 33,1% de los encuestados (40) los que sí estarían dispuestos y un 24% (29) los que no lo estarían. Por otra parte, tenemos al 43% restante (53) que tal vez sí que lo harían. Las respuestas a esta pregunta fueron dispares, lo que nos hace pensar que esta funcionalidad puede ser interesante no desarrollarla en una fase inicial del proyecto, ya que conllevaría un esfuerzo para desarrollar algo que no estamos seguros de si va a cumplir las necesidades del usuario.

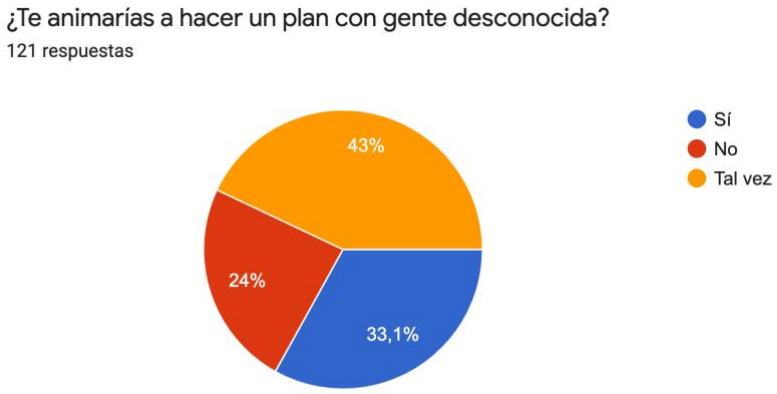


Ilustración 9: *Gráfico circular representando si la gente haría o no planes con gente desconocida*

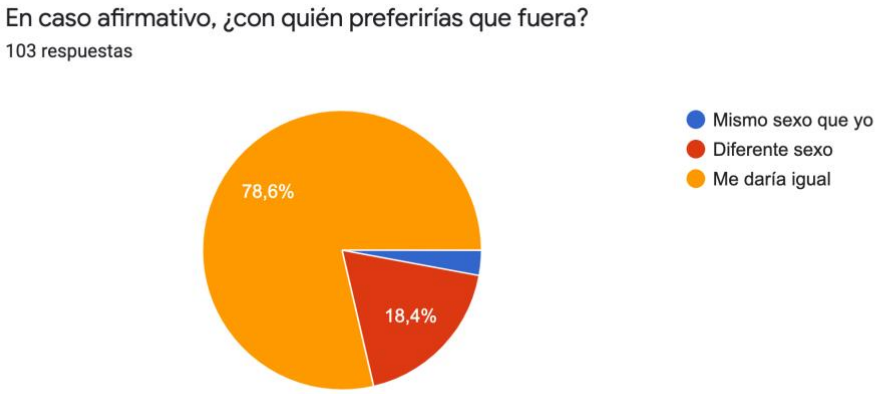


Ilustración 10: *Gráfico circular representando el sexo de las personas desconocidas con las que harían planes*

3.2. Producto mínimo viable (MVP)

En este apartado, se va a explicar en qué consiste un Producto Mínimo Viable (MVP) y qué pasos hay que seguir para llegar a lanzarlo al mercado y obtener información para ir mejorando de manera iterativa el producto para acabar ofreciendo un buen servicio de la manera más óptima posible.

Para el desarrollo del MVP, se va a seguir el método *Lean Startup* [1]. El método *Lean Startup* fue creado por Steve Blank cuando empezó a desarrollar una metodología de validación de productos basado en el desarrollo de cliente, que consiste en saber si un producto cubre las necesidades básicas o deseos de los clientes. Con este método se pretende invertir en recursos solamente cuando se ha obtenido el conocimiento del mercado. De esta manera, la cantidad de recursos que se invierten en la fase inicial de cualquier negocio se ven reducidas notablemente, permitiendo invertir estos recursos cuando hay más probabilidades de éxito y permitiendo pivotar (cambiar la estrategia que se había planteado inicialmente) para acercarse a la estrategia óptima.

Para llevar a cabo esta metodología, es necesario seguir unos pasos que nos llevarán a obtener un aprendizaje validado con el cual podremos adaptar mejor nuestro producto a lo que el cliente necesita realmente. Para ello, se hará uso del ciclo Crear-Medir-Aprender, el cual se pondrá en marcha una y otra vez para ir mejorando y afinando más nuestro producto para que cubra las necesidades de nuestros posibles consumidores.

Planteamiento hipótesis: se parte de que hay un problema a resolver y explicando por qué la gente estaría dispuesta a pagar por nuestro servicio. Se pueden realizar una serie de encuestas o entrevistas para identificar qué es lo que realmente les preocupa. Hay que tener claro desde un principio que el problema está ahí y que es lo suficientemente grande como para atacarlo.

Validación hipótesis: con el planteamiento realizado previamente, se desarrolla un MVP con las características básicas para ver si es lo que la gente quiere o si la gente pagaría por el servicio.

Medición hipótesis: una vez tengamos los datos de la validación, veremos si cumplen con lo que nosotros habíamos previsto. Así, podremos ir mejorando nuestro producto. Podemos ver desde si los canales de adquisición de usuarios han sido eficaces, hasta si la interfaz es suficientemente intuitiva para los usuarios o necesita cambios.

Aprendizaje validado: todo lo que se ha obtenido previamente es un conocimiento del mercado el cual hemos obtenido con la mínima inversión de esfuerzo, tiempo y dinero. A partir de este punto, habrá que aplicar todo lo aprendido, volver a lanzar una nueva versión de nuestro producto al mercado, volver a medir y volver a aprender.

Todo esto se consigue gracias al *feedback* de nuestro público objetivo, que es el que nos dirige, con sus reacciones a nuestro producto, hacia dónde nos tenemos que dirigir. Así podremos darnos cuenta si el cliente está dispuesto a pagar por el producto o necesitamos hacer cambios para satisfacer sus deseos.

3.3. Lean Canvas

El Lean Canvas [2] es una herramienta en la que se resume el modelo de negocio de manera visual. Está compuesto por unos bloques de análisis diseñados especialmente para empresas que están empezando. La idea es poder visualizar de una los diferentes aspectos del producto que se va a desarrollar: propuesta de valor, ingresos, gastos, canales de adquisición, entre otros.

1. **Clientes**

   Define quién sufre los problemas que se han detectado y a quién de estos puede llegar a dar una solución. Aquí hay que tener muy en cuenta quiénes son los *early adopters*, es decir, aquellos que son más susceptibles a hacer uso de nuestro producto o servicio para crear tendencia y atacar al resto del mercado.

2. **Problema**

   El problema que se resuelve con el producto que se va a desarrollar. El problema determina el por qué tu producto o servicio va a existir. Lo ideal es detectar los 3 principales problemas de tus posibles usuarios relacionados con nuestro campo de acción.

3. **Proposición de valor**

   ¿Qué se ofrece que no ofrezcan otras soluciones parecidas a la que vamos a desarrollar? ¿Por qué nos van a elegir a nosotros en vez de decantarse por otras soluciones similares? Es aquí donde se marca el valor diferencial que ofrece nuestro producto o servicio con respecto al mercado.

4. **Solución**

   Teniendo los problemas identificados, ¿cuál es la solución a estos problemas? Definiendo de manera correcta los problemas, podemos afinar la

solución mucho mejor. En fases tempranas, lo mejor es desarrollar las características de nuestro producto que permitan dar solución a los problemas detectados previamente y más adelante ir incluyendo más funcionalidades.

5. **Canales**

¿Cómo se va a acceder a los clientes en un inicio? ¿Cómo se va a dar visibilidad al producto o servicio? Dependiendo de los clientes que se hayan definido, hay que analizar cuáles serán los canales de venta y que queden bien especificados.

6. **Ingresos**

¿Cómo va a generar dinero nuestro producto o servicio? En este apartado, hay que plantear el modelo que se va a seguir para ganar dinero, ya sea por suscripción por parte de los usuarios, por pago mensual por parte de los negocios o cualquier otra forma para tener unos ingresos.

7. **Costes**

¿Qué costes vamos a tener al inicio y en el lanzamiento? Se deberá recoger todo aquel gasto fijo que se pueda necesitar para poner en marcha el negocio para estimar una inversión inicial. Entre otros, se pueden encontrar licencias de software, personal, material, servidores, bases de datos…

8. **Métricas**

Una vez se haya definido todo lo anterior, habrá que definir nuestros *Key Performance Indicators* (*KPIs*), datos que nos indicarán si vamos por el buen camino o es hora de pivotar. Como ejemplo, podemos establecer como *KPI* principal el número de usuarios que debemos tener a los 3 meses del lanzamiento.

9. **Ventaja competitiva**

La ventaja competitiva es lo que nos va a hacer estar por delante de nuestros competidores y que va a resultar difícil de copiar. Podría ser una patente, un diseño exclusivo, una tecnología específica, etc.

A continuación, presento el *Lean Canvas* relacionado con el proyecto presentado:

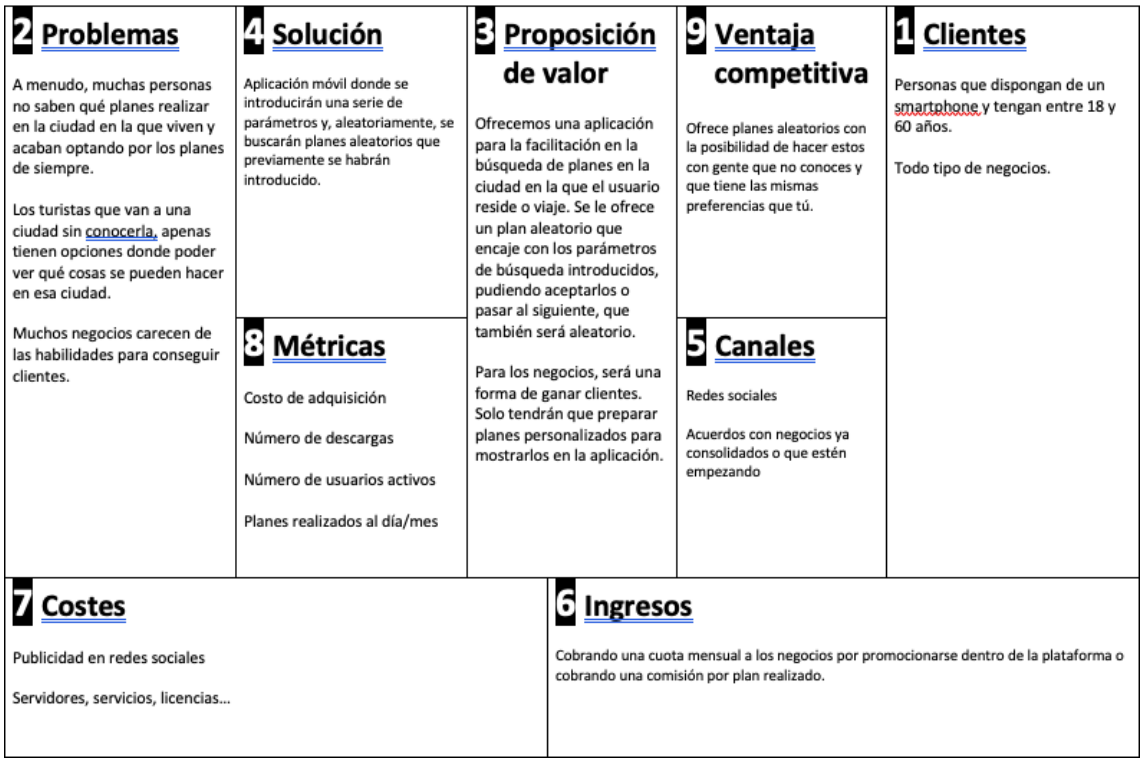


Ilustración 11: *Lean Canvas del proyecto presentado*

4. Diseño solución

Una vez obtenidos los datos a través de las diferentes encuestas realizadas y vistas las funcionalidades que harían falta para desarrollar el *MVP*, se pretende desarrollar un producto lo más simple posible y evitando un gran esfuerzo, aunque sin dejar de pensar en la escalabilidad del producto.

Partimos de haber tenido en cuenta que elegir la solución adecuada para el desarrollo depende de varios factores: presupuesto, experiencia con la tecnología y el público objetivo. Debido a ello, se va a desarrollar una aplicación híbrida, es decir, una aplicación *software* que combina elementos de las aplicaciones nativas y de las aplicaciones *web*. En las aplicaciones híbridas, el núcleo de la aplicación está escrito usando tecnologías *web* (HTML, CSS y JavaScript), que luego se encapsulan dentro de una aplicación nativa. La principal ventaja que tiene el desarrollo de una aplicación híbrida frente a una aplicación nativa (una aplicación que se ha desarrollado utilizando el lenguaje de desarrollo nativo y las herramientas específicas de esa plataforma) es que, al desarrollarse solamente en lenguaje *web* y compilar tanto en Android como en iOS, te ahorra mucho tiempo de desarrollo, sobre todo tratándose del estado inicial del proyecto, donde ya sabemos que el tiempo y los recursos son bien apreciados.

4.1. Tecnologías planteadas

Para desarrollar un MVP como el que se ha planteado, tenemos que contar con dos partes: la parte *front end* o frontal y la parte *back end* o servidor. Son dos términos muy utilizados en la informática para referirse a la separación de intereses entre una capa de presentación (*front end)* y una capa de acceso a datos (*back end*). En el caso del frontal, es la parte que interactúa con los usuarios y el servidor la que procesa la entrada desde el frontal. La idea general es que el frontal sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y los transforme ajustándolos a las especificaciones que demanda el *back end* para poder procesarlos, devolviendo generalmente una respuesta que el *front end* recibe y expone al usuario de una forma entendible para este.

A continuación, se presentan las diferentes tecnologías planteadas en ambas partes.

4.1.1. Front end

Como bien se ha comentado antes, el desarrollar una aplicación móvil nativa queda descartado debido al tiempo que hay que dedicar a desarrollar una aplicación para cada uno de los sistemas operativos líderes en el mercado actualmente: iOS y Android.

En 2017, Google introdujo el concepto de aplicaciones *web* progresivas (en inglés, *progressive web apps* o PWA), que permiten que este tipo de aplicaciones adopten más funciones similares a las de las aplicaciones, como iconos de aplicaciones, notificaciones, funcionamiento sin conexión y más. Microsoft también adoptó esta solución recientemente, convirtiendo este tipo de aplicaciones en ciudadanos de primera clase en su tienda de aplicaciones.

Debido al desconocimiento del desarrollo de una PWA, esto nos deja con la única elección de desarrollar una aplicación híbrida. En los últimos años, el desarrollo de aplicaciones híbridas ha crecido de manera notable. Estas son una gran opción para que las empresas puedan reducir tanto la cantidad de desarrolladores como el tiempo en el proceso de desarrollo.

Dentro del mundo del desarrollo de aplicaciones híbridas, contamos con diferentes *frameworks*, los cuales nos ayudan a diseñar y correr de manera rápida nuestra aplicación. Entre las muchas opciones, mencionamos los 3 más populares [3]:

- **Ionic Framework**

  Ionic Framework[3] es un kit de desarrollo software (en inglés, *software development kit* o SDK) de código abierto para desarrollar aplicaciones híbridas con experiencias nativas y web de alta calidad. Fue lanzado en 2013 y se construyó sobre AngularJS, un *framework* de JavaScript de código abierto que se utiliza para crear aplicaciones web, y Apache Cordova, un entorno de desarrollo de aplicaciones móviles utilizando CSS3, HTML5 y JavaScript. Sin embargo, la última versión se reconstruyó como un conjunto de componentes web, permitiendo al desarrollador decidirse entre otros *frameworks* como React o Vue.js.

  Es el más conocido de todos, ya que es de los que más años lleva en el ámbito del desarrollo híbrido. Al hacer uso de la tecnología *web* (HTML, CSS y JavaScript) y estar hacer uso de *frameworks* de JavaScript, son muchos desarrolladores los que se decantan por hacer uso de esta tecnología.

  De acuerdo con las estadísticas de las tiendas de los sistemas operativos iOS y Android, Ionic impulsa una gran cantidad de aplicaciones. Sin ir más lejos, un 9% en iOS y casi el 20% en Android, lo que dice mucho de este kit de desarrollo.

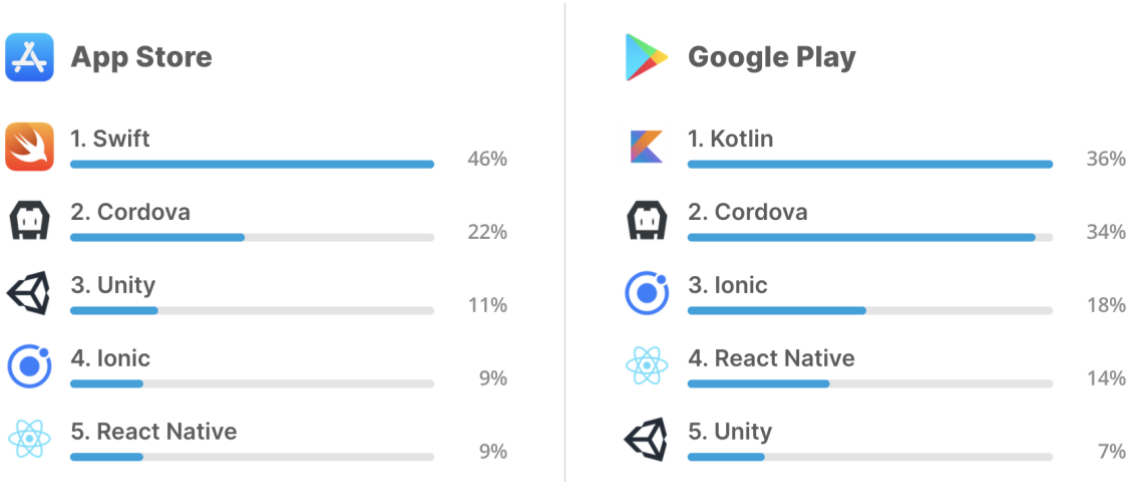


Ilustración 12: *Estadísticas de Appfigures sobre los SDK más utilizados en las aplicaciones publicadas en tiendas*

[3] http://ionicframework.com/

Son varias las razones por las que Ionic sería una buena elección para desarrollar el MVP. Entre ellas:

- Proporciona una gran colección de funcionalidades
- Basado en Angular; no hay necesidad de aprender un lenguaje nuevo
- Tiene multitud de componentes de interfaz de usuario y elementos predefinidos
- Tiene un sistema de recarga en vivo y una aplicación que permite obtener una vista previa de la aplicación directamente en el dispositivo
- Cuenta con un servidor de desarrollo incorporado
- Proporciona herramientas de depuración para agilizar nuestro desarrollo
- Contiene su propia interfaz de línea de comandos (Ionic CLI) que agiliza el ciclo de desarrollo
- Respaldado por una gran comunidad de desarrolladores

- **Flutter**

Flutter[4] fue lanzado en 2017 y destaca por ser el kit de herramientas de la interfaz de usuario Google para crear aplicaciones compiladas de forma nativa para dispositivos móviles, web y de escritorio a partir de un mismo código. Acorde con la encuesta anual realizada por StackOverflow[5] [4], página *web* de preguntas y respuestas para programadores profesionales y aficionados, Flutter es utilizado por un 7,2% de los 40,314 que respondieron a dicha encuesta (**Ilustración 13**).

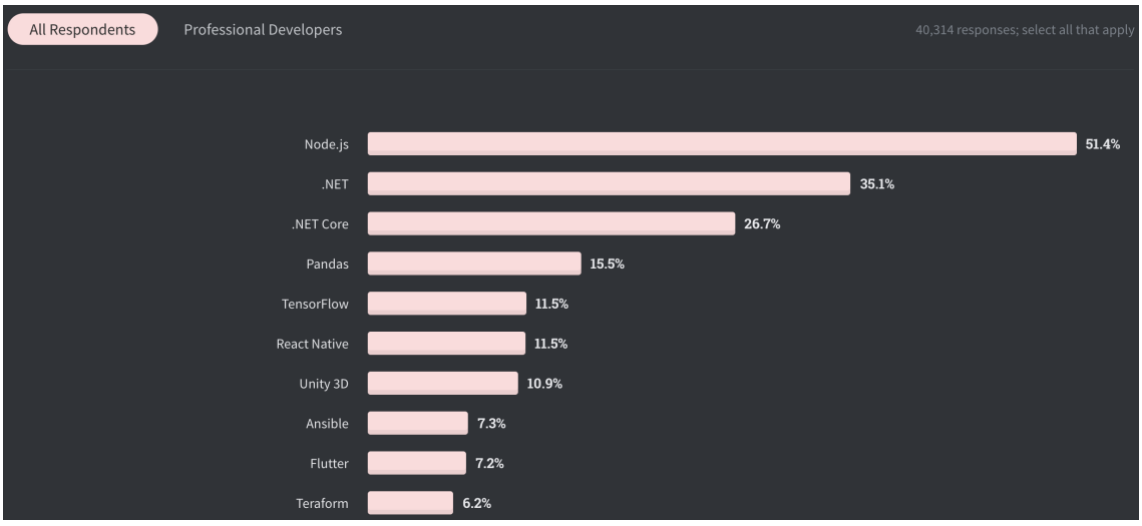


Ilustración 13: *Resultados encuesta sobre los frameworks más utilizados*

4 https://flutter.dev/

5 https://stackoverflow.com/

Este *framework* se caracteriza por un desarrollo rápido, interfaz de usuario expresiva y flexible y funcionalidad nativa. Con sus componentes de diseño flexibles, se puede crear una interfaz de usuario lo bastante sofisticada en muy poco tiempo. También, sus elementos de interactividad ayudan a crear una experiencia de usuario muy atractiva.

Otro de los elementos de por qué Flutter es uno de los *frameworks* más populares con respecto al desarrollo de aplicaciones híbridas es, sin duda, el lenguaje que utiliza: Dart, un lenguaje de programación de código abierto desarrollado por Google. Según la misma encuesta mencionada anteriormente, Dart ocupa el séptimo puesto de los lenguajes más queridos por los desarrolladores (**Ilustración 14)**.

Ilustración 14: *Resultados encuesta sobre los lenguajes de programación más utilizados*

- **React Native**

React Native es un *framework* de código abierto creado por Facebook, Inc. Se utiliza para desarrollar aplicaciones para Android, iOS y Web permitiendo a los desarrolladores utilizar React (biblioteca JavaScript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página) junto con funcionalidades de aplicaciones nativas.

Volviendo a la **Ilustración 13**, podemos observar que React Native es utilizado por el 11,5% de los desarrolladores que contestaron a la encuesta. Siendo un poco más precisos para ver el alcance de este *framework*, vemos en la **Ilustración 15** como ocupa el puesto 5 entre los más utilizados por desarrolladores profesionales.

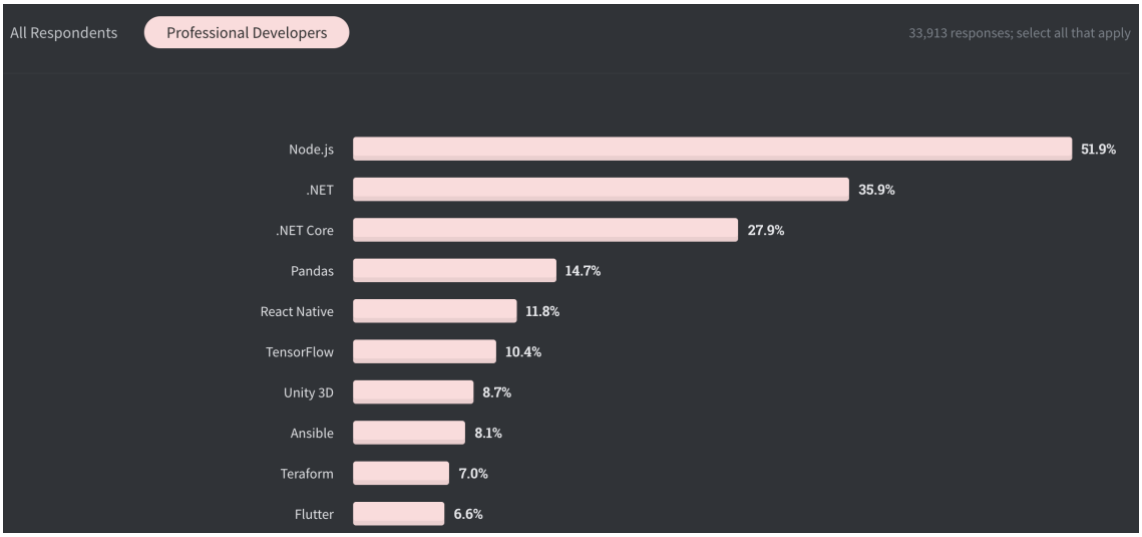


***Ilustración 15**: Resultados encuesta realizada a desarrolladores profesionales sobre los frameworks más utilizados*

Veamos las razones de por qué se trata de una de las tecnologías más utilizadas a lo hora de desarrollar aplicaciones híbridas:

- Alta potencia: renderizado con código nativo
- Reduce costes al reutilizar código
- Está respaldado por una gran comunidad de desarrolladores en la que se puede confiar
- Garantiza un ciclo de desarrollo más corto
- Cuenta con recarga en vivo: para actualizar el código y obtener una vista previa de esas actualizaciones en tiempo real
- Gran variedad de *plugins*

Pero hay algunas desventajas que lo colocan por debajo de los 2 anteriores:

- No es la mejor tecnología si se trata de una aplicación con múltiples cambios de pantalla, muchas interacciones y gran cantidad de pantallas
- La oferta de módulos personalizaos es bastante limitada
- Proporciona pocas funcionalidades nativas, lo que provoca perder tiempo adaptándolas por cuenta propia
- La curva de aprendizaje es bastante alta
- Necesidad de tener un equipo de desarrolladores con experiencia para crear una aplicación de alta potencia
- Presenta algunas deficiencias de los componentes de navegación

Una vez analizadas las 3 opciones, se ha decidido descartar los dos últimos *frameworks* mencionados y hacer uso de Ionic Framework. La razón principal es el ya conocimiento de él y el desconocimiento tanto del lenguaje utilizado en Flutter como la tecnología usada por React Native. También es razón de peso el que cuente con una gran comunidad de usuarios. Esto ayuda sobre todo a encontrar fácil solución a los problemas que vayan saliendo a lo largo del tiempo de desarrollo del producto.

4.1.2. Back end

Además de la parte *front end*, también debemos contar con una capa de acceso a datos, es decir, un servidor donde podamos recibir datos del frontal y devolver respuestas para que el usuario pueda visualizar de manera legible cualquier tipo de información.

Aquí tenemos diversas opciones para poner en marcha un servidor. En primer lugar, contamos con la más complicada y costosa de todas, donde tendríamos que montar nosotros la infraestructura y, además, escribir el código con todas las funcionalidades que queramos que tenga. Debido a que nuestra idea es desarrollar un MVP donde invertir el mínimo número de recursos posibles, vemos esta opción fuera de nuestro alcance.

En segundo lugar, una opción más asequible es la de elegir un servicio de terceros que cuente con infraestructuras de computación en la nube (en inglés, *cloud computing*), paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es internet. Esto nos permite dedicarnos a escribir el código sin tener que preocuparse por la infraestructura. Como ventajas principales en comparación con la computación tradicional, cabe destacar que no requiere el aprovisionamiento por adelantado ni la gestión de los servidores, que cuenta con capacidad de auto-escalado y que se paga únicamente por los recursos que se utilizan. Estas ventajas lo convierten en la opción ideal para muchos casos prácticos, desde aplicaciones HTTP a los *bots* de chat.

Como tres mejores opciones a la hora de elegir en servicio de computación en la nube, tenemos: Google Cloud Platform, Amazon Web Services (AWS) y Microsoft Azure. A continuación, entraremos un poco en detalle para ver qué ofrece cada uno de ellos:

- **Google Cloud Platform**

  Como el propio nombre indica, esta solución pertenece al gigante tecnológico Google. Es una plataforma utilizada para crear ciertos tipos de soluciones a través de la tecnología almacenada en la nube.

  Cuenta con diferentes productos para solventar y manejar cualquier tipo de proyecto. Entre ellos podemos encontrar de diferentes tipos: de computación, de almacenamiento y bases de datos, de redes, de seguridad…

  Entre los beneficios de esta plataforma, podemos encontrar:

  - Alta productividad
  - Colaboración rápida
  - Alta seguridad
  - Menor cantidad de datos almacenados en dispositivos vulnerables
  - Fiabilidad
  - Flexibilidad

- Rentabilidad

Entre los productos más destacados, tenemos:

- **Compute Engine:** máquinas virtuales que se ejecutan en el centro de datos de Google
- **Cloud Storage:** almacenamiento de objetos seguro, duradero y escalable
- **Cloud CDN:** red de distribución para publicar contenidos *web* y de vídeo
- **Dataflow:** analíticas de *streaming* para el procesamiento de *streaming* y por lotes

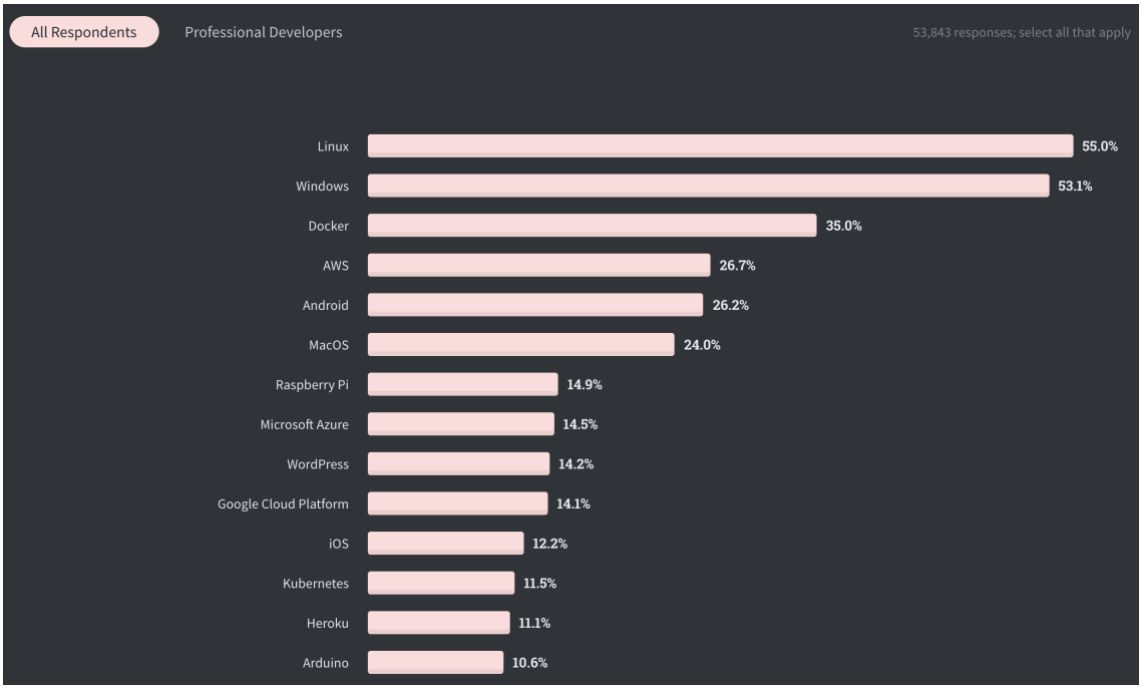


***Ilustración 16**: Resultados encuesta realizada a desarrolladores profesionales sobre las plataformas más utilizadas*

- **AWS**

Amazon Web Services es un conjunto de servicios de computación en la nube pública ofrecidos por Amazon.

Esta plataforma de Amazon ofrece casi todas las funcionalidades de la computación en la nube. Dispone de una gran cantidad de herramientas para la

gestión de los diferentes elementos de una empresa. Los servicios de AWS están totalmente preparados para ser escalables según la empresa que los utiliza va creciendo.

Podemos ver cómo AWS no pasa desapercibido en cuanto a las plataformas más utilizadas por los desarrolladores (**Ilustración 16**) ocupando el puesto nº 4.

Algunas de las categorías de los servicios que se ofrecen son las siguientes:

- ***Cloud computing*:** todo lo necesario para la creación de instancias y el mantenimiento o escalado de las mismas. Amazon EC2 es el rey indiscutible dentro de los servicios de computación en la nube de Amazon.
- **Bases de datos:** diferentes tipos de bases de datos mediante Amazon RDS, que incluye desde MySQL a NoSQL.
- **Almacenamiento:** está Amazon S3 como servicio principal en cuando a almacenamiento para almacenar todo tipo de archivos.
- **Seguridad:** compatible con 90 estándares de seguridad y certificaciones de conformidad y los 117 servicios de AWS que almacenan datos de los clientes ofrecen el cifrado de datos.

- **Microsoft Azure**

Microsoft Azure es la plataforma pública y privada de aplicaciones en la nube de Microsoft para crear, implementar y administrar aplicaciones.

Como podemos ver en la **Ilustración 16**, Azure ocupa un puesto notable dentro de las plataformas de servicios en la nube, incluso por encima de Google Cloud Platform.

Azure está centrada en las aplicaciones, lo que significa que administramos todo el ciclo de vida de nuestra aplicación, desde el diseño inicial, el desarrollo y las pruebas de la aplicación a su implementación en la nube. Con un simple botón, es posible supervisar y escalar dicha aplicación cuando se ejecuta en internet. También permite comprender y analizar la aplicación para que podamos crear una versión mejorada e implementarla en la nube sin que haya tiempo de inactividad.

Utiliza una tecnología conocida como virtualización. La virtualización separa el acoplamiento entre la CPU de un ordenador y su sistema operativo, utilizando una capa de abstracción llamada *hypervisor*, que emula todas las funciones de un ordenador real y su CPU en una máquina virtual. Puede ejecutar varias máquinas virtuales al mismo tiempo y cada máquina virtual puede ejecutar cualquier sistema operativo compatible, como Windows o Linux. Esta tecnología se repite a gran escala en los centros de datos de Microsoft.

Consta de 3 componentes principales:

- **Fabric:** es el conjunto abstracto de recursos de procesos del centro de datos

- **Storage:** ayuda a la aplicación a administrar todos sus datos de forma fiable y escalable

- **Experiencia:** empaquetamiento de Fabric, Storage y APIs para integrarlo con Visual Studio para ofrecerlo en forma de SDK

Y, en tercer lugar y la opción elegida, contamos con servicios plataforma como servicio (PaaS). Entre los mejores para el desarrollo de aplicaciones móviles encontramos 3, mencionando sus características a continuación:

- **Firebase**

Firebase[6] es un servicio *back end* en tiempo real respaldado por Google que permite a los desarrolladores crear aplicaciones web o móviles en tiempo real sin protocolos complejos o sin tener que escribir código desde cero o administrar la infraestructura. Utiliza un conjunto de herramientas para la creación y sincronización de proyectos que serán dotados de alta calidad, dejando espacio para la escalabilidad del producto en cuanto a número de usuarios.

Cuenta con una serie de servicios que permiten implementar de manera relativamente sencilla las principales características de una web o de una aplicación móvil, permitiendo así al desarrollador ahorra una gran cantidad de recursos. Entre ellos, podemos encontrar:

- **Autenticacón:** permite administrar usuarios de manera simple y segura. Ofrece varios servicios de autenticación (correo electrónico y contraseña, numero de teléfono…) proveedores externos como Google o Facebook, o

- **Cloud Storage:** almacenamiento de contenido generado por los usuarios, como imágenes, vídeos, audio, documentos. Los SDK de Firebase para Cloud Storage añaden la seguridad de Google a las operaciones de carga y descarga de archivos sin importar la calidad de la red.

- **Cloud Firestore:** almacenamiento y sincronización de datos entre usuarios y dispositivos a escala global mediante una base de datos NoSQL alojada en la nube. Cuenta con sincronización en tiempo real, soporte sin conexión y consultas eficaces.

- **Cloud Messaging:** envío de mensajes y notificaciones *push* a los usuarios de las diferentes plataformas (Android, iOS y la Web) de manera gratuita. Con posibilidad de enviar los mensajes a dispositivos individuales, a grupos de dispositivos o a segmentos de usuarios.

La gran ventaja de Firebase es que dispone del plan Spark, un plan gratuito para empezar a desarrollar. Dependiendo de la herramienta que vayamos a utilizar, podremos ver si es gratis o si tenemos unos límites establecidos. En la

[6] https://firebase.google.com/

Ilustración 17, observamos como los límites varían dependiendo del servicio. Por otro lado, tenemos el Blaze Plan, el cual, una vez sobrepasados los límites del plan Spark, empieza a cobrar dependiendo del uso que se dé a cada servicio.

Pruebas A/B	Sin cargo	
Analytics	Sin cargo	
App Distribution	Sin cargo	
App Indexing	Sin cargo	
Authentication		
Autenticación telefónica: Canadá, EE.UU. y la India	10,000 por mes	USD 0.01 por verificación
Autenticación telefónica: Todos los demás países	10,000 por mes	USD 0.06 por verificación
Otros servicios de autenticación	✓	✓
Cloud Firestore		
Datos almacenados	1 GiB en total	USD 0.18 por GiB
Salida de red	10 GiB por mes	Google Cloud pricing
Operaciones de escritura de documentos	20,000/día	USD 0.18 por cada 100,000
Operaciones de lectura de documentos	50,000/día	USD 0.06 por cada 100,000
Operaciones de eliminación de documentos	20,000/día	USD 0.02 por cada 100,000

Ilustración 17: *Precios de algunos servicios de los planes Spark y Blaze de la plataforma Firebase*

- **Heroku**

Heroku[7] es una de las soluciones plataforma como servicio (PaaS) basada en la nube más eficaces, que permite a los desarrolladores y empresas crear, implementar, administrar, mejorar y escalar sus aplicaciones de manera más eficiente. Proporciona una vía rápida para convertir una idea de aplicación en un producto funcional. Todos los productos de Heroku están destinados a facilitar un desarrollo más rápido y confiable, y un escalado más fácil.

Heroku funciona con Dynos, unidades que proveen capacidad de cómputo dentro de la plataforma y que están basados en contenedores Linux. Cada

7 https://www.heroku.com/

Dyno está separado del resto, por lo que las acciones que realices en uno, no va a afectar al resto

Como podemos observar en la **Ilustración 16,** Heroku es una de las plataformas más recomendables a la hora de elegir plataforma como servicio.

Las características principales son:

- **Contenedores inteligentes:** al estar separados los unos de los otros, en caso de haber fallos en la infraestructura de alguno, los demás contenedores no se ven afectados
- **Routing:** los routers hacen un seguimiento de la ubicación de los Dynos que estén corriendo y redirigen el tráfico de acuerdo a la misma
- **Elasticidad y crecimiento**: a través de línea de comandos o del panel de control, puedes cambiar los Dynos asignados a una aplicación
- **Integración continua:** integraciones automáticas a menudo para detectar cuanto antes los fallos
- **Alojamiento escalable:** después de la implementación, es posible que se requieran ajustes adicionales en los contenedores debido a una variedad de condiciones, como mayor tráfico o patrones de uso desiguales.

- **AWS Amplify**

Hasta ahora, Amazon ha traído servicios y productos para la web, móviles, redes de IoT, bases de datos, seguridad y mucho más. Sin embargo, ahora Amazon ofrece Amplify[8] una solución altamente efectiva para desarrolladores de aplicaciones web y móviles con funcionalidades listas para usar y muy innovadoras, ayudando a desarrollar e implementar de una forma más fácil, rápida y segura.

Amplify está compuesto por 3 componentes: bibliotecas, componentes de interfaz de usuario y una cadena de herramientas de la interfaz de línea de comandos. Las bibliotecas y los componentes de la interfaz de usuario se basan en los servicios de AWS. Son de código abierto y funcionan con los *frameworks* de front end existentes (Angular, React Native, Ionic…) así como con los entornos de desarrollo de iOS y Android. La interfaz de línea de comandos (CLI) de Amplify es una cadena de herramientas para crear y mantener back ends sin servidor en AWS.

Como productos destacados, podemos encontrar: autenticación, almacenamiento de datos, análisis de datos y notificaciones push.

Amplify tiene una serie de beneficios:

[8] https://aws.amazon.com/es/amplify/

- **Creación de aplicaciones innovadoras:** con las bibliotecas ofrecidas por Amplify, podemos mezclar características como la autenticación, los datos, la inteligencia artificial y el aprendizaje automático.

- **Configuración rápida:** proporciona un flujo de trabajo orientado para personalizar un back end en condiciones con tan solo unos simples comandos.

- **Implementación y escalabilidad fácil:** ofrece seguridad, confiabilidad y disponibilidad global. Flujo de trabajo basado en Git mediante la consola de AWS

La razón de haber elegido la tercera opción dentro de las tres planteadas es la falta de recursos en poder implementar una de las dos primeras. Con un PaaS, podemos disponer de las funcionalidades básicas que necesita nuestro MVP, pudiendo escalar la aplicación en caso de que tenga una buena aceptación en el mercado.

4.2. Prototipo de la aplicación

Hacer un prototipo de una aplicación móvil es una de las técnicas más utilizadas para ver cómo va a quedar dicha aplicación sin tener que desarrollar ninguna línea de código. Nos va a permitir ahorrarnos mucho tiempo y trabajo, ya que podemos optimizar el diseño, la apariencia y la usabilidad de la aplicación sin apenas esfuerzo.

Un buen prototipo es muy importante a la hora de desarrollar una aplicación por dos razones:

1. Evitar gastar tiempo innecesariamente. Si nos ponemos a desarrollar código sin tener claras las pantallas y la funcionalidad de la aplicación, lo más probable es que, mientras estemos desarrollando, nos demos cuenta de cosas que podríamos haber percibido de manera sencilla a la hora de hacer el prototipo. Esto nos va a ocasionar una pérdida de tiempo innecesaria debido a que nos va a tocar rehacer muchas cosas a medida que estamos desarrollando. Por lo tanto, antes de empezar con el código, debemos tener claro cómo va a ser todo para hacer el número mínimo de cambios posibles

2. Nos permite validar la idea desde un inicio. Al realizar un prototipo, podemos darnos cuenta de forma rápida de qué es lo que realmente funciona y qué hay que cambiar sin apenas esfuerzo. Puedes ver cómo va a ser el resultado final de la aplicación y hacer los cambios pertinentes sobre el prototipo, lo que es mucho más cómo que realizar cambios sobre código ya escrito.

En el ámbito del prototipado de aplicaciones, se nos vienen a la mente tres palabras: *wireframe, mockup* y prototipo. En muchas ocasiones, estas palabras son usadas indistintamente, pero realmente no son los mismo.

Un *wireframe* es un estilo de boceto de la aplicación móvil en el que se define, sin diseño ni colores, la organización de qué cosas van a aparecer en cada pantalla de nuestra aplicación. También se añade una explicación visual de cómo va a ser la

interacción y la navegación. Es importante que se trabaje bien desde un principio ya que es la base para acabar teniendo un buen diseño previo al desarrollo. La idea es que no lleve mucho tiempo y se vea de forma clara el esqueleto de la aplicación: incluso dibujar a mano puede valer para empezar. Se pueden utilizar en todas las fases del diseño, aunque se usan con mayor frecuencia en fases tempranas para definir la estructura básica.

Un *mockup* tiene como objetivo mostrar la parte más visual de la aplicación. Nos servirá para ver cómo va a estar estructurada la información, los contenidos y de qué funcionalidades va a disponer sin poder interactuar con él. Representan de una forma clara cómo va a ser la aplicación sin invertir tanto tiempo como al realizar el prototipo en sí. Al contrario que el *wireframe*, en el *mockup* ya se pueden observar colores, tipografía, sombras, etc. Es el punto donde el usuario o el cliente se hace una idea de la apariencia final del producto final. En resumen, si el *wireframe* asienta la base y estructura del diseño, el *mockup* define su apariencia.

Por último, un prototipo nos va a mostrar cómo será la interacción con la aplicación y cómo será finalmente en cuanto a apariencia visual. Son muy utilizados para testear la usabilidad del producto. Si el *wireframe* define la estructura y el *mockup* cómo es visualmente, un prototipo nos indica cómo se va a comportar el producto. Por ello, llegados a este punto, la interacción tiene que estar bien definida. Con un prototipo bien desarrollado, podemos evaluar con usuarios reales el comportamiento que tienen al interactuar con la aplicación. Existen múltiples herramientas con las que se pueden diseñar prototipos y trabajar de manera sencilla la interacción y navegación de la aplicación.

Debido a la temprana fase en la que nos encontramos y siguiendo los pasos del método *Lean Startup* mencionado anteriormente, hemos decidido simplemente diseñar unos *wireframes* para tener claro desde un principio cómo va a estar estructurada la aplicación y así poder testear con usuarios finales lo antes posible para ir iterando el producto a medida que vayamos recibiendo *feedback*. Para ello, hemos decidido hacer uso de Mockflow, una aplicación web que nos provee de herramientas de diseño de *wireframes*, *mockups* y prototipos.

En primer lugar, tendríamos la pantalla de inicio de sesión, donde el usuario podría introducir su correo electrónico y su contraseña para acceder a su cuenta en caso de no tenerla. También contamos con la posibilidad de iniciar sesión a través de redes sociales (**Ilustración 18)**:

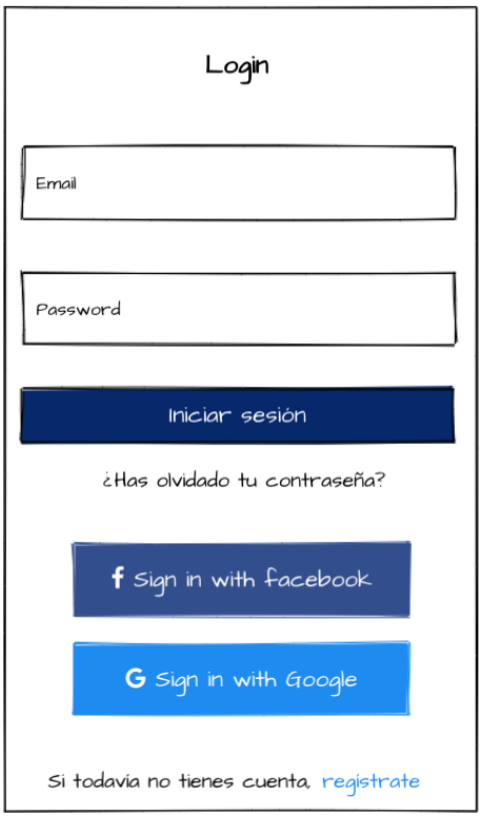


Ilustración 18: *Pantalla inicio de sesión*

En caso de no disponer de una cuenta, podemos hacer *click* en el botón de "regístrate" que podemos ver en la parte inferior derecha de la pantalla para navegar a la pantalla de Crear cuenta (**Ilustración 19).** El usuario tendrá que introducir una serie de datos personales o elegir una de las redes sociales proporcionadas para crearse una cuenta dentro de la plataforma:

Ilustración 19: *Pantalla para crear una cuenta de usuario*

Una vez dentro, nos encontramos directamente con un plan (**Ilustración 20)** el cual podemos reservar o descartar deslizando hacia la derecha o izquierda, respectivamente**:**

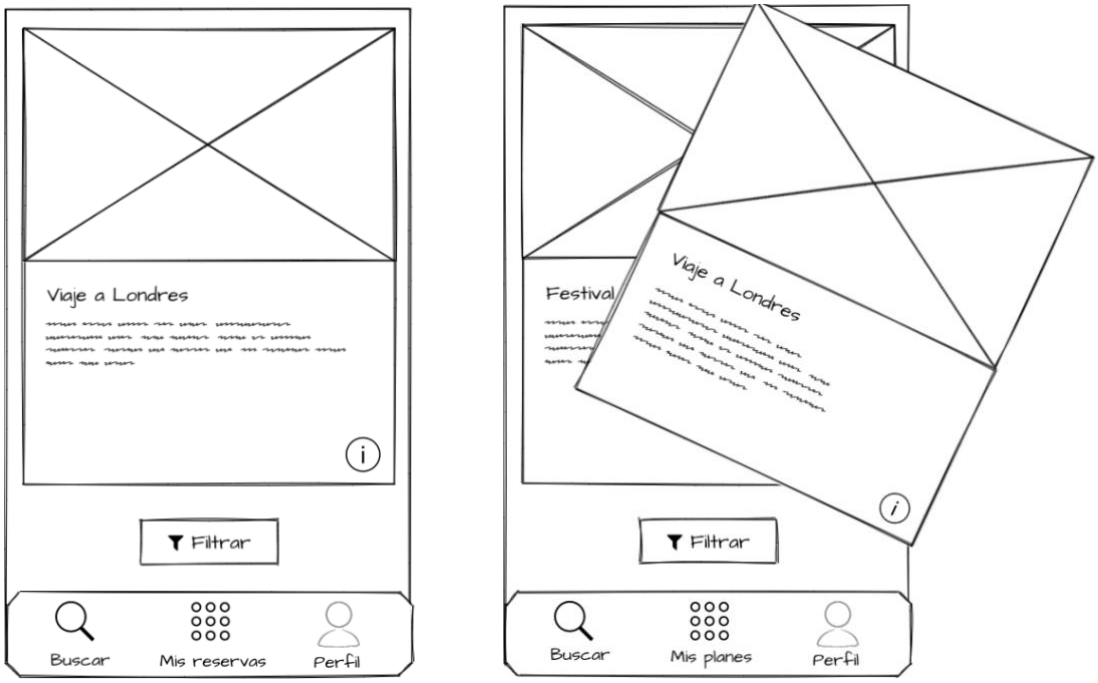


Ilustración 20: *Pantalla principal de planes.*

También podemos filtrar los planes que se nos muestran accediendo a la pantalla de los filtros (**Ilustración 21**), donde podremos filtrar por precio, fecha, categoría, etc:

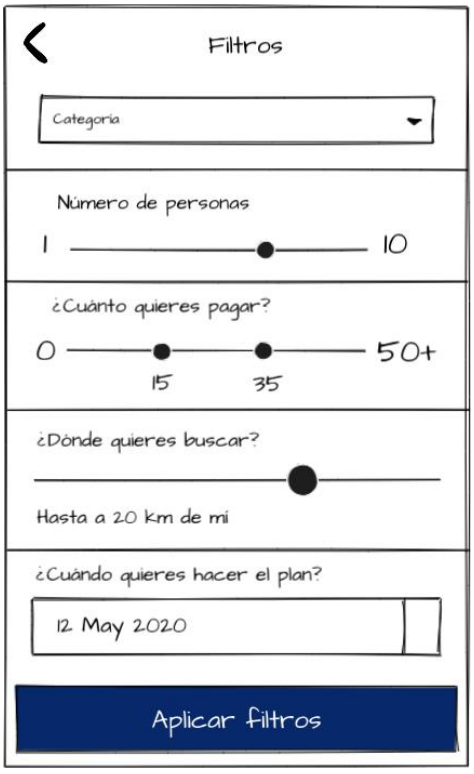


Ilustración 21*: Página de filtros*

Si queremos acceder a la pantalla de información del plan **(Ilustración 22)** mostrado en pantalla, deberemos hacer *click* en el botón de información situado en la parte inferior derecha dentro de la tarjeta del propio plan, desde la que podremos ver más imágenes, más información, compartir el plan y reservarlo:

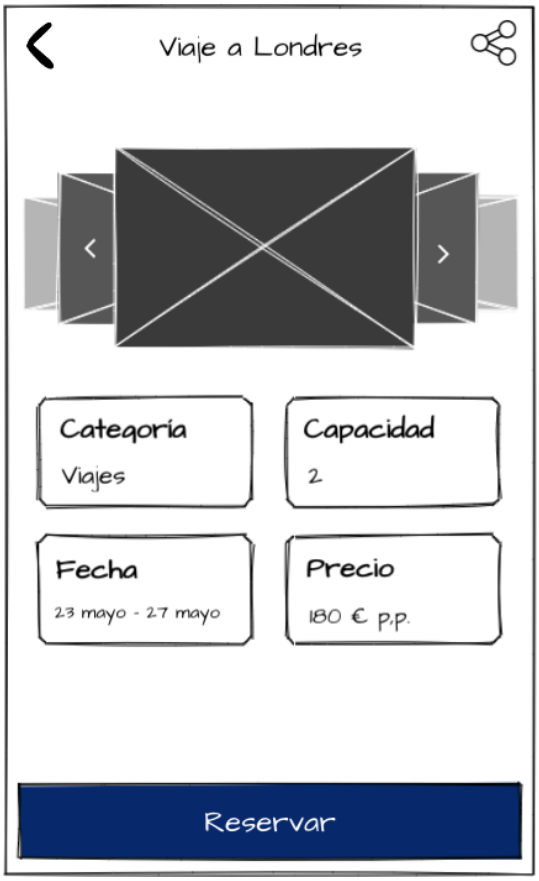


Ilustración 22: Pantalla de información del plan

Accediendo a la pestaña Mis reservas (**Ilustración 23)**, podremos ver los planes que tenemos reservados y que se van a realizar próximamente:

***Ilustración 23**: Pestaña de Mis reservas*

Por último, disponemos de la pestaña Perfil (**Ilustración 24**), donde podremos ver información nuestra, invitar a amigos y cambiar de idioma la aplicación:

Ilustración 24: *Pestaña de Perfil*

Con estos *wireframes*, ya disponemos de una idea básica de cómo va a ser nuestra aplicación. A partir de aquí, podemos empezar a desarrollar la aplicación teniendo claro hacia dónde vamos a dirigirnos en cuanto a diseño, evitando así muchos momentos de dudas y de tener que eliminar o rehacer código innecesariamente.

4.3. Diagramas

En esta sección, se pueden visualizar los diferentes diagramas de casos de uso y de clases según los diferentes componentes del sistema. Los casos de uso capturan los requisitos funcionales del sistema a desarrollar. Al tratarse del desarrollo de un MVP, los diagramas son muy básicos, aunque a medida que se vayan desarrollando más funcionalidades irán creciendo.

4.3.1. Diagramas de casos de uso

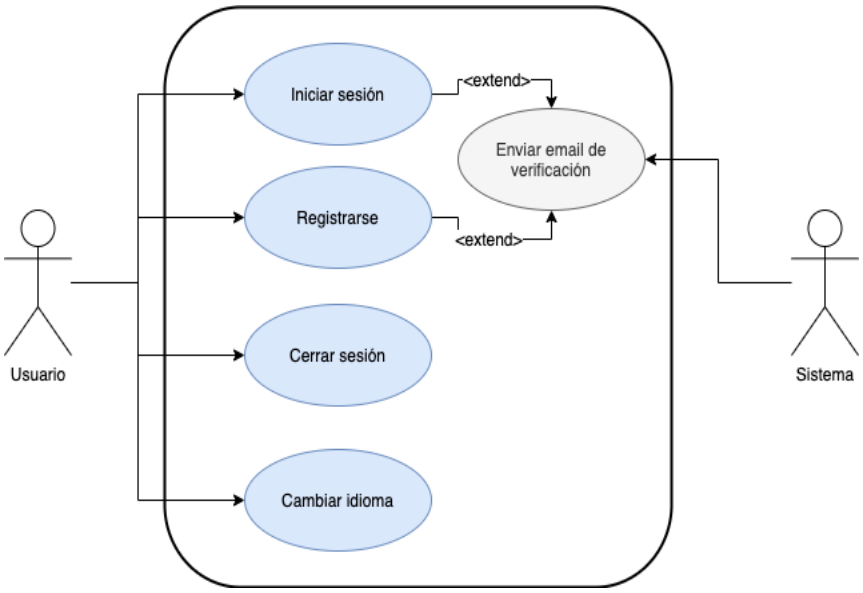
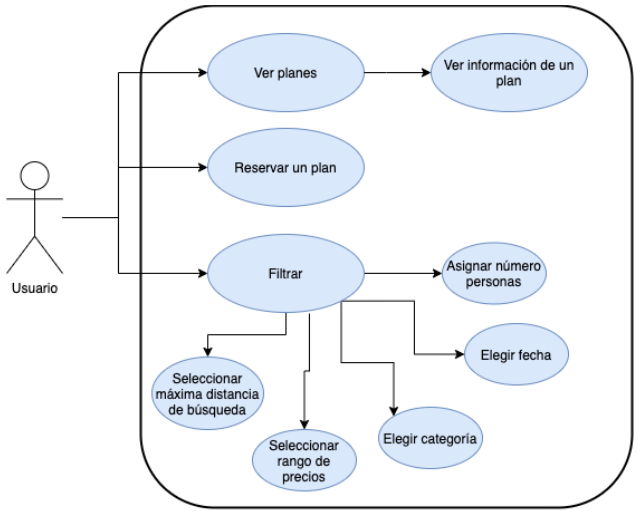


Ilustración 25: *Diagrama de casos de uso*



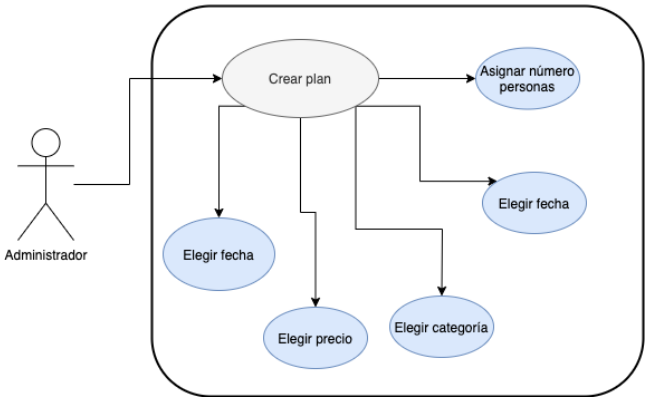


Ilustración 26: *Diagrama de casos de uso*

4.3.2. Diagrama de clases

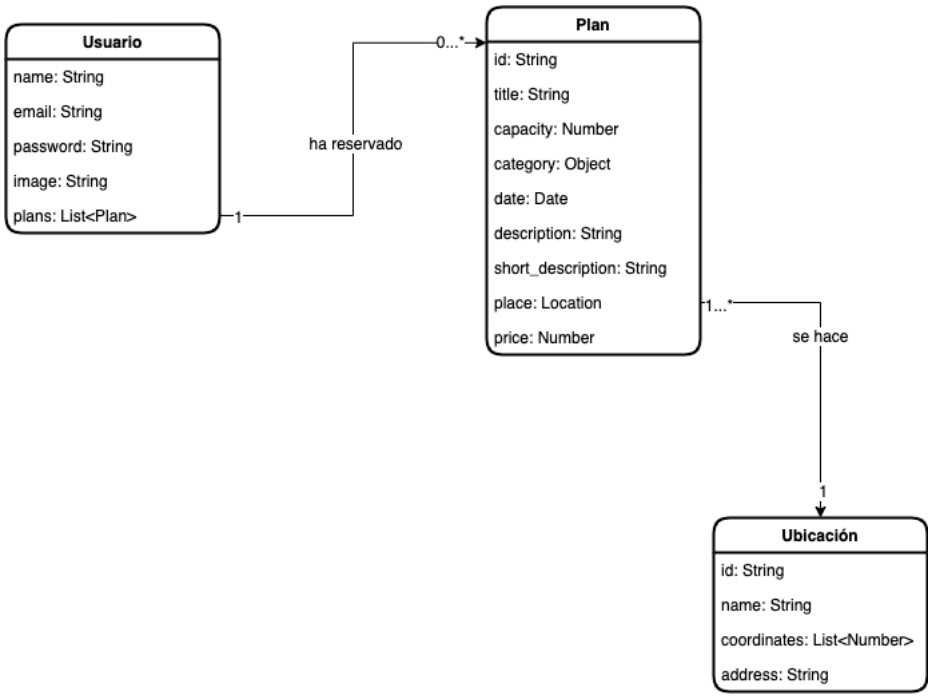


Ilustración 27: *Diagrama de clases*

5. Desarrollo

En este apartado se va a concretar qué metodología y qué flujo de trabajo se ha seguido y cómo se ha implementado la solución para llevar a cabo el desarrollo de la aplicación de este proyecto.

5.1. Metodología y flujo de trabajo

El presente trabajo se ha desarrollado bajo una organización de metodología ágil, donde se ha hecho una separación de la parte de desarrollo en pequeñas tareas para un mejor diseño, implementación y validación, reduciendo así el tiempo empleado en el futuro desarrollo de cada una de ellas.

El objetivo principal es acercar a todos una nueva forma de trabajo que permite afrontar la revolución digital en a que nos encontramos y que ha obligado a las organizaciones a innovar en todos los ámbitos.

Se podría definir como un conjunto de metodología destinadas a desarrollar productos y servicios de calidad que respondan a las necesidades de unos clientes cuyas prioridades cambian a una velocidad cada vez mayor.

¿Cómo se trabaja en la metodología ágil? El proyecto se trocea en pequeñas partes que tienen que completarse y entregarse en pocas semanas a la finalización de un período de tiempo o *sprint*. Estas partes concluyen en un mínimo producto viable o una versión inicial del producto con la funcionalidad necesaria que nos permita su lanzamiento al mercado para ser consumida por los clientes a un coste de producción bajo. Para ello, prevé la utilización de un conjunto de metodologías como *Design Thinking, Lean Startup*, Scrum o DevOps. Cada una de ellas se caracteriza por situar al cliente en el centro de todas las pruebas sobre el producto. Todo este trabajo lo realizan equipos multidisciplinares y autoorganizados en los que aparecen nuevos roles y funciones a través de un ciclo de vida iterativo e incremental. Una vez acabado nuestro primer MVP, se testea con los usuarios para recoger sus reacciones y opiniones. Los requerimientos extraídos de estas pruebas se integran en el segundo MVP a través de este proceso de adaptación rápida y constante que al mismo tiempo aumentar el compromiso e implicación de los componentes del equipo.

La metodología ágil se basa en los doce principios plasmados en el manifiesto del desarrollo ágil de *software* [4]:

1. Satisfacer al cliente por encima de todo a través de la entrega temprana y continua de software de valor.

2. Se aceptan cambios incluso en las etapas más tardías del desarrollo. Este tipo de metodología aprovecha el cambio constante para ofrecer una ventaja competitiva al cliente.

3. Entrega de software funcional frecuente, en un período de dos semanas hasta dos meses. Cuanto menor sea el tiempo de entrega, mejor.

4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.

5. Desarrollo de proyectos con personas motivadas, dándoles la oportunidad y el respaldo que necesiten y confiando en ellas para que realicen la tarea.

6. La conversación cara a cara es la manera más eficiente y efectiva de comunicar información dentro de un equipo de desarrollo.

7. El *software* funcional es la principal medida de progreso

8. Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica ensalza la agilidad.

10. Es esencial la simplicidad como arte de maximizar la cantidad de trabajo que se realiza.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan.

12. De vez en cuando, el equipo tiene que parar a reflexionar cómo ser más efectivo y actuar en consecuencia.

Como vimos en la sección 3.2, para el desarrollo de la aplicación se han seguido los principios del método *Lean Startup*. Para hacerlo posible, se ha hecho uso de una serie de herramientas que nos han permitido realizar un desarrollo organizado y lo que nos ha llevado a tener listo nuestro primer MVP.

- **Git**

Git es un sistema de control de versiones creado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Un sistema de control de versiones se encarga de mantener un registro de los cambios en el código fuente. Si una persona edita una línea de código, este sistema indicará el antes y el después de haber realizado este cambio y, en el mejor de los casos, indicará la fecha del cambio y el autor de este. Cada vez que realizo un cambio en el código y se publica en el repositorio (entiéndase repositorio como el lugar donde se guarda el código fuente), esto se conoce como un *commit* o un cambio. Los *commits* pueden venir acompañados de un mensaje en el cual se puede indicar qué se ha cambiado. En este sistema se puede ir al *commit* anterior en el caso de que el *commit* actual rompa el *software* o, sencillamente, no fue deseado.

Algo que caracteriza mucho a Git es su naturaleza, la cual sigue una estructura tipo árbol, es decir, que en un repositorio pueden existir diferentes ramas con diferentes cambios realizados. Se puede disponer de una rama *master* o rama principal y una rama de prueba. Si una persona se encuentra haciendo cambios en la rama de prueba y les dan el visto bueno a sus cambios, ésta puede incluir los cambios realizados sobre la rama de prueba en la rama *master*. De esta forma, un gran número de desarrolladores puede trabajar tranquilamente en un proyecto

teniendo una copia local del repositorio. Una vez listo los cambios, se pueden describir mediante un mensaje para posteriormente subirlos al repositorio remoto, sobre el cual trabajan los demás desarrolladores. Git también permite hacer un *pull* de los cambios, es decir, descargar los cambios que hayan hecho otros desarrolladores en el repositorio remoto a nuestro repositorio local para así tener el código lo más actualizado posible.

Por otra parte, Git es un sistema con una gran cantidad de utilidades. Todas las funciones de Git se deben hacer por un terminal o línea de comandos, aunque en la actualidad han salido varios clientes que ofrecen una interfaz gráfica para facilitar tareas, tales como crear un *commit,* ver el historial de los cambios, etc. A pesar de ello, los clientes con interfaz gráfica están bastante limitados en la mayoría de los casos ya que no ofrecen todas las funcionalidades de Git.

En la actualidad las páginas más famosas dedicadas al control de versiones son Github y GitLab, ambas basadas en este sistema.

- **GitLab**

Como ya se ha comentado antes, GitLab es un servicio de alojamiento para los proyectos que utilizan el sistema de control de versiones Git. Además, también ofrece alojamiento de wikis y un sistema de seguimiento de errores. Cubre el ciclo de vida de un proyecto de desarrollo y todo el ciclo de vida de DevOps.

Para este proyecto, se ha creado un repositorio privado para que cualquiera que tenga acceso se pueda descargar una copia local de todo el código almacenado en dicho repositorio.

Dentro de GitLab tenemos integradas diferentes herramientas entre las cuales podemos encontrar Tareas, *Merge Requests* y *CI/CD.*

Gracias a estas dos herramientas, hemos podido definir un flujo de trabajo, el cual nos va a permitir una fácil integración en caso de que se incorporen nuevos miembros al equipo de desarrollo en un futuro.

Con el flujo de trabajo, tenemos la posibilidad de organizar las tareas y recursos mediante reglas que facilitan y simplifican el control de un determinado proceso. El objetivo principal del flujo de trabajo es normalizar al máximo los procesos con control absoluto de todos y cada uno de los pasos de cada tarea.

En GitLab, además de poder almacenar el código fuente, tenemos la posibilidad de crear tareas asignándoles un miembro y una prioridad, entre otras. Una vez creadas las tareas y haberles asignado una prioridad, Hay que saber diferenciar entre las tareas qué vamos a meter en nuestra hoja de ruta las que se van a quedar en el *backlog* (lista de trabajo pendiente). Una vez definidas las que van a ser desarrolladas, las situamos dentro de nuestra hoja de ruta, donde podemos ver de forma visual qué tareas son las siguientes en desarrollarse y cuánto tiempo nos va a llevar.

A la hora de empezar a desarrollar una de las tareas, lo primero que se va a hacer es valorar el tiempo que nos puede llevar el desarrollo de dicha tarea y si tenemos que

dividir la tarea en tareas más pequeñas. De esta forma podremos saber cuánto tiempo nos va a llevar cada tarea para colocarla en nuestra hoja de ruta.

Una vez tengamos la valoración hecha, desde GitLab tendremos que crear una rama desde la rama principal. El nombre de esta rama creada tendrá que ser lo más descriptivo posible para que de un solo vistazo puedas ver qué se está desarrollando en esta rama. En el momento en el que se empieza el desarrollo, la tarea pasará al tablero Kanban, donde tendremos tres columnas con las tareas que hay por hacer, las que se están haciendo y las que se han terminado. Cada una de estas columnas visualiza una fase del proceso de una tarea. Al mismo tiempo cada tarea que entra en el flujo de trabajo aparece en el tablero como una tarjeta kanban. Como es de esperar, una tarea entrará por la columna "*To Do*" y pasará a la columna "*Doing"* una vez se decida empezar con el desarrollo de esta.

Una vez se ha terminado el desarrollo de una tarea y se han hecho los test pertinentes, la rama pasará a ser revisada mediante un *Merge request* contra la rama *master.* La persona asignada para la revisión tendrá que ver los cambios realizados para comprobar si se puede mejorar el código sobre todo en cuanto a legibilidad. Cuando el revisor o los revisores decidan que está todo correcto y aprueben el *merge request*, esa tarea podrá ser juntada con la rama *master*.

Todo lo comentado previamente es extrapolable a cualquier tarea de cualquier tipo lo cual es una buena ventaja porque esta forma de trabajo es independiente del número de personas que trabajen en el equipo de desarrollo.

5.2. Front end y back end

En este apartado se va a explicar con mayor detalle cómo se ha implementado la solución tanto en la parte *front end* como en la parte *back end*.

Ya se ha mencionado previamente en la sección 4.1 que, al tratarse del desarrollo de un MVP, se iban a utilizar las tecnologías Ionic Framework para el frontal y Firebase para el servidor.

5.1.1. Ionic Framework

Para empezar a crear una aplicación híbrida con Ionic, lo primero que hay que hacer es instalar el paquete de npm[9] `@ionic/cli` con el comando `npm i -g @ionic/cli`. Npm es el sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript.

Con el paquete ya instalado, podemos ejecutar diferentes comandos que nos permitirán realizar diferentes acciones dentro de nuestro proyecto, como por ejemplo generar nuevas páginas o compilar la aplicación en diferentes dispositivos y sistemas operativos.

9 https://www.npmjs.com/

Para empezar con nuestra aplicación, se ejecuta el comando `ionic start <name> <template>` donde el argumento `name` será el nombre de esta y el argumento `template` será la plantilla sobre la que queremos desarrollarla. En nuestro caso, el comando sería `ionic start plannie tabs`.

Hay dos *frameworks* sobre los que podemos desarrollar nuestra aplicación: React y Angular. En nuestro caso, hemos decidido hacer uso de Angular, ya que es lenguaje que ya conocemos y tiene un amplio soporte. Una vez creada, podemos empezar a escribir nuestro código HTML, TypeScript y SCSS para crear las funcionalidades y los componentes.

La mayoría de la aplicación va a ser desarrollada en el directorio `src/app/`:

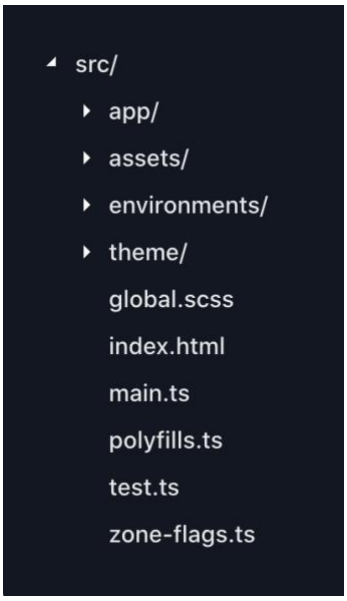


Ilustración 28: *Estructura inicial del proyecto*

El directorio `src/` tiene elementos como el archivo `index.html`, archivos de configuración para pruebas, una carpeta de recursos para imágenes y el directorio `app/` para el código de la aplicación.

Para correr la aplicación en el navegador, tenemos dos opciones. En primer lugar, si ejecutamos el comando `ionic serve` en la terminal dentro del directorio del proyecto, se empezará a correr un servidor de desarrollo al cual podremos acceder mediante la URL `http://localhost:8100`. Como segunda opción, podemos instalar un paquete de npm llamado `@ionic/lab`. Con el comando `ionic lab`, se nos abrirá la aplicación también en el navegador, pero en este caso en dos emuladores de Android e iOS (**Ilustración 29**), que nos permitirán ver el comportamiento de la aplicación en cada uno de ellos.

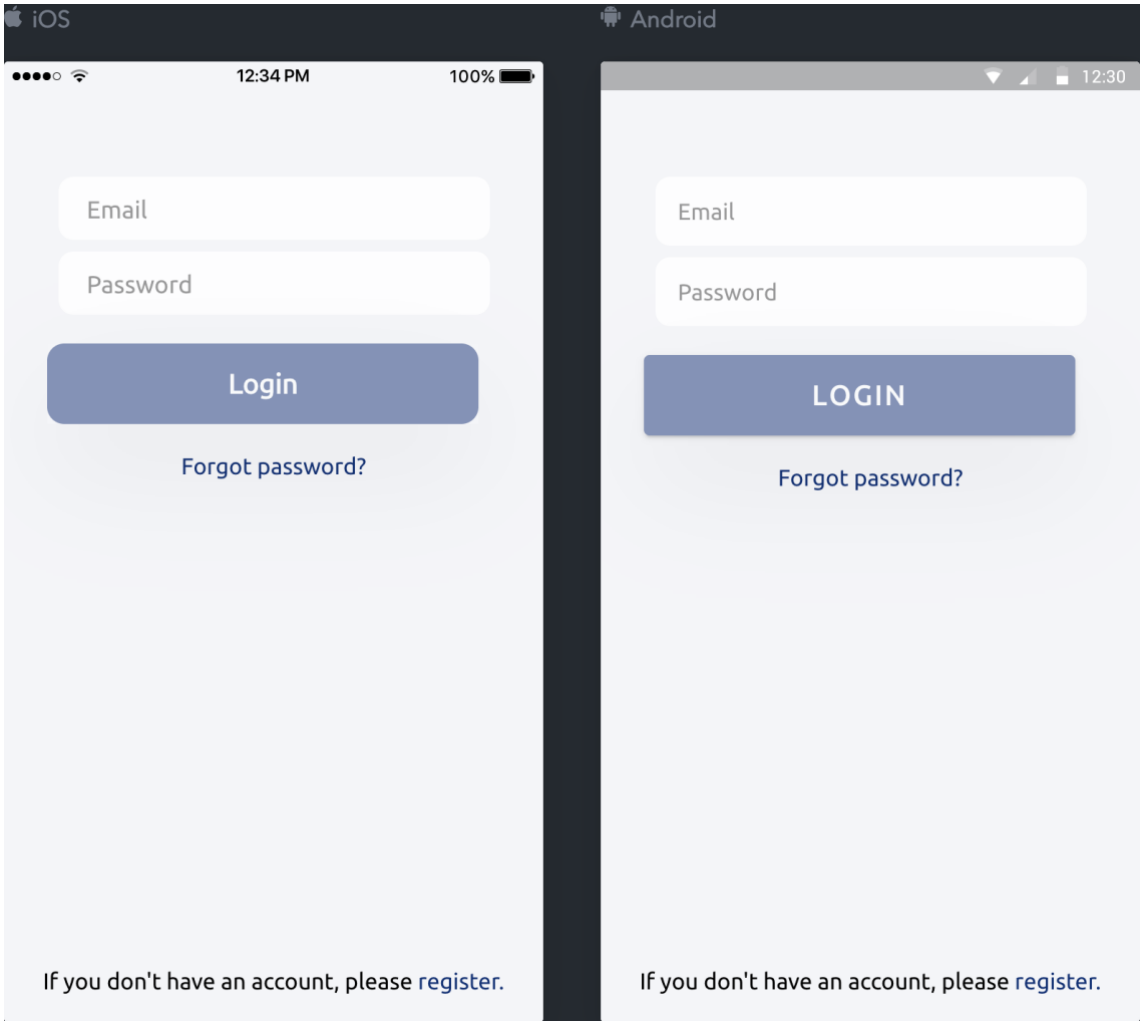


Ilustración 29: *Emuladores Android y iOS al ejecutar el comando* `ionic lab`

Una vez tengamos ya nuestra plantilla inicial corriendo en el navegador, podemos ir creando nuestros elementos dentro de la aplicación como páginas, servicios, componentes… con el comando `ionic generate <type> <name>`. En nuestro caso, hemos hecho uso de este comando para generar páginas, servicios e interfaces.

Se pueden definir las páginas como las pantallas de la aplicación. Cada página que se crea tiene su propio archivo `.html`, donde se escribirán los componentes de dicha página; un archivo `.ts`, donde escribiremos la lógica; y el archivo `.scss`, donde irán ubicados todos los estilos de la página.

Por otra parte, los servicios serán utilizados para conectarse con Firebase, para obtener la ubicación del usuario o para mostrar errores.

Se ha hecho uso de SaSS, un metalenguaje de CSS, para definir todos los estilos de la aplicación. Es un lenguaje de script que es traducido a CSS. Dentro de este lenguaje, vamos a aplicar la sintaxis SCSS, que usa el formato de bloques como CSS: llaves para separar bloques de código y punto y coma para separar las líneas dentro de un bloque. La gran diferencia entre SCSS y CSS es que en el primero se pueden anidar los selectores CSS. Aquí vemos un ejemplo de la diferencia:

SCSS Sass

```scss
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

CSS

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Para los estilos de la aplicación, se ha hecho uso de la metodología BEM[10]. BEM es una metodología de nomenclatura para definir las clases en los nodos HTML del documento, es decir, es una manera de nombrar las clases de los nodos de tu HTML para posteriormente atacarlos con CSS de una manera fácil, sencilla y clara. El objetivo de BEM es dar mucha más transparencia y claridad en tu estructura HTML y CSS. BEM te dice cómo se relacionan las clases entre sí, lo que es particularmente útil en secciones complejas del documento. Todas las clases del proyecto pueden encajar con esta filosofía.

BEM son 3 siglas: B de bloque, E de elemento y M de modificador (en inglés, *Block Element Modifier*). Un Bloque es una sección independiente que tiene significado propio por sí solo. Contiene todos los nudos HTML de una estructura a la que te estés refiriendo. Un Elemento es una porción más pequeña interna a un bloque. Se usa para ir dividiendo el bloque en segmentos más pequeños. Y un Modificador sirve para modificar algunas propiedades de un bloque o elemento.

Entrando más en detalle, un bloque es una entidad independiente con significado propio. Es una sección de código HTML. Puede ser simple o compuesto, es decir, que en su interior viven más bloques independientes. Para nombrar un bloque puedes utilizar letras, dígitos o guiones, pero tienes que tener en cuenta algunas restricciones a la hora del nombre un bloque. No se pueden usar mayúsculas, no se pueden usar dos guiones bajos seguidos porque está reservado para los elementos y tampoco se pueden usar dos guiones de seguidos porque está reservado para los modificadores.

En cuanto a un elemento, es una parte de un bloque y no tiene significado independiente. Cualquier elemento está semánticamente ligado a su bloque. Los nombres de los elementos pueden constar de letras, dígitos, guiones y guiones bajos. La clase CSS se forma como un nombre de bloque más dos guiones bajos más el nombre del elemento: `.block__elem`. Dentro de un bloque dado, todos los elementos son semánticamente iguales.

Por último, los modificadores se usan para cambiar la apariencia, el comportamiento o el estado de bloques o elementos. Los nombres de los modificadores pueden estar compuestos por letras, dígitos, guiones y guiones bajos. La clase CSS se forma como el nombre del bloque o elemento más dos guiones: `.block--mod` o `.block__elem--mod`. El modificador es un nombre de clase

[10] https://en.bem.info/methodology/

adicional que se agrega a un nodo DOM de un bloque o elemento. Se añaden las clases de modificadores manteniendo la clase original.

Este sería un ejemplo utilizado en el proyecto:

```
<ion-card class="plan" *ngFor="let plan of plans">
  <div class="plan__main">
    <ion-img src="{{ plan.image }}" class="plan__main-img"></ion-img>
    <div class="plan__main-info">
      <div class="plan__main-info--price">
        {{ plan.price }} € p.p.
      </div>
    </div>
    <div class="plan__title">{{ plan.title }}</div>
    <div class="plan__description">{{ plan.category.name }}</div>
    <div class="plan__capacity">
      {{ plan.capacity }}
      <ion-icon color="light" name="people"></ion-icon>
    </div>
    <div class="plan__location">
      <ion-icon color="light" name="pin"></ion-icon>
      {{ plan.place }}
    </div>
  </div>
</ion-card>
```

```
.plan {
  &__main {
    &-img {}
    &-info {
      &--price {}
    }
  }
  &__title {}
  &__description {}
  &__capacity {}
  &__location {}
}
```

Ilustración 30: *Código de ejemplo donde se aplica la metodología BEM*

El haber aplicado la metodología BEM tiene como objetivo principal mantener una estructura legible y escalable en los archivos de estilos.

Una pantalla de ejemplo para ver cómo ha quedado la aplicación después de aplicar los estilos:

Ilustración 31: *Pantalla principal una vez aplicados los estilos mediante metodología BEM*

Una funcionalidad importante que también se ha introducido es la internacionalización de la aplicación. Así, la aplicación está disponible en dos idiomas: español e inglés. Para hacer esto posible, se han instalado dos paquetes npm: `@ngx-translate` y `@ngx-translate/http-loader`. Una vez instalados, habrá que importarlos dentro del módulo principal de la aplicación, el archivo `app.module.ts`, y escribir una función que cree una instancia de lo que va a cargar los archivos de cada idioma, los cuales estarán situados en la carpeta `assets`. Al abrir la aplicación, esta se encargará de obtener el lenguaje guardado o, en su defecto, el lenguaje por defecto del navegador para obtener las traducciones solicitadas por cada página.

Cada archivo de traducción será un archivo JSON, donde se irán introduciendo las diferentes palabras o frases que se vaya a usar en la aplicación, por ejemplo:

```
"HOME": {
  "title": "Inicio",
  "filter": "Filtros"
},
```

Y en el HTML se usará de la siguiente manera:

```
<ion-button shape="round" class="filter" (click)="openFilters()">
  {{ 'HOME.filter' | translate }}
</ion-button>
```

5.1.2. Firebase

Como ya se ha mencionado en la sección 4.1.2, Firebase ofrece una multitud de servicios que permite implementar las principales características de una web o de una aplicación móvil. Gracias a los paquetes npm de angular `@angular/fire` y `firebase` se ha hecho posible la integración de Ionic con Firebase.

En este proyecto se han hecho uso de tres servicios de Firebase, los cuales se van a explicar a continuación:

- **Autenticación**

  Dentro del servicio de la autenticación, tenemos diferentes opciones para hacer que nuestros usuarios inicien sesión: correo y contraseña, teléfono, Google, Facebook… Por el momento, vamos a hacer uso de la primera de ellas.

  Lo primero que debemos hacer es importar el módulo de la autenticación (AngularFireAuthModule) dentro del módulo principal. Una vez importado, podemos hacer uso de las funciones de la autenticación de Firebase. En nuestro caso, desde el servicio `auth.service.ts` creado previamente, hacemos uso de varias. Entre ellas:

```
await this.afAuth.auth.createUserWithEmailAndPassword(
    credentials.email,
```

```
    credentials.password,
);
```

Con la cual creamos una cuenta de usuario mediante correo y contraseña en el sistema de autenticación. Para acceder a una cuenta de usuario:

```
await this.afAuth.auth.signInWithEmailAndPassword(
    credentials.email,
    credentials.password,
);
```

Los datos que se envían al servidor de Firebase son los que el usuario introduce en los formularios de registro y de inicio de sesión. Cada una de estas funciones almacena en la aplicación el usuario de Firebase con sus diferentes propiedades, como el email, número de teléfono, URL de la imagen, etc. Esta información será utilizada para mostrar en la pantalla de perfil.

- **Cloud Firestore**

Este servicio tiene como objetivo principal almacenar y sincronizar datos para el desarrollo del lado del cliente y del servidor.

Aquí es donde vamos a almacenar la información de todos los planes para que luego puedan ser leídos desde el lado del cliente. Para ponernos un poco en contexto, Firestore es una base de datos NoSQL que almacena los datos en documentos, organizados en colecciones. Los documentos pueden contener objetos anidados complejos, además de subcolecciones.

En nuestro caso, la colección principal será plans, donde cada documento de esta colección será un plan. Cada documento almacenará todas las propiedades de cada plan con sus respectivos valores.

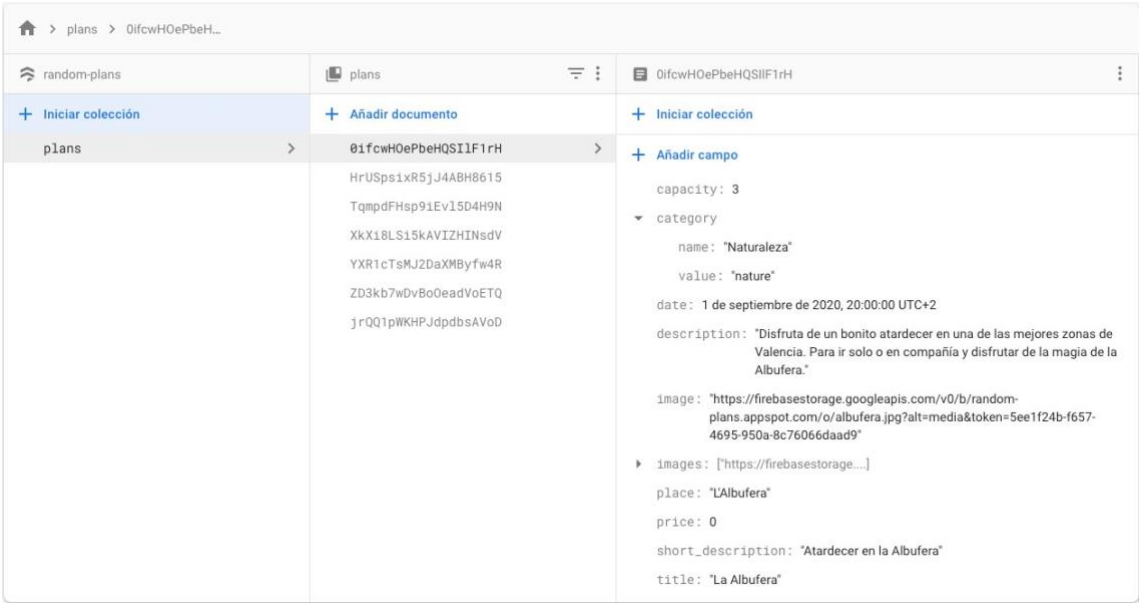


Ilustración 32: *Estructura base de datos de Cloud Firestore*

Para leer todos los planes, tenemos que importar el módulo de Firestore (AngularFiresoreModule) dentro del módulo principal para que se accesible desde cualquier parte de la aplicación. Desde el servicio `firebase-db.service.ts` llamamos a la función `get()` de la colección plans definida previamente:

```
this.plansCollection = this.db.collection<Plan>("plans");

getAllPlans() {
  return this.planCollection.get();
}
```

Esto nos devolverá una observable de tipo QuerySnapshot. Un QuerySnapshot contiene cero o más objetos DocumentSnapshot que representan los resultados de una consulta. Una vez obtenido el resultado de la consulta, podemos acceder a la información con un bucle forEach:

```
this.firebaseDbService.getAllPlans().subscribe((querySnapshot)
=> {
  querySnapshot.forEach(doc => {
    const plan = doc.data();
    plan.id = doc.id;
    this.plans.push(plan as Plan);
  });
});
```

Para obtener los planes con los filtros aplicados por el usuario, se utilizan las consultas compuestas, donde se encadenan varios métodos `where()` con el objetivo de crear consultas más específicas.

```
const plansCollection = this.db.collection('plans');

return plansCollection.ref
  .where('category.value', '==', filter.category)
  .where('capacity', '<=', filter.capacity)
  .get();
```

Por último, para obtener la información de un plan hay que llamar a la función `doc()` de la referencia de la colección pasando como parámetro el ID del plan que deseamos obtener.

```
getPlan(planId: string): AngularFirestoreDocument<Plan> {
  return this.db.collection('plans').doc(planId);
}
```

- **Cloud Storage**

Por último, utilizaremos el sistema de almacenamiento de archivos de Firebase para guardar las imágenes de cada plan. Cada imagen tendrá una URL, la cual estará almacenada en las propiedades `image` o `images` de cada plan, tal y como se muestra en la **Ilustración 32**.

6. Conclusiones

Para dar por finalizado este trabajo, he de decir que me ha hecho crecer mucho personalmente como profesionalmente. Han sido muchos los retos a los que me he tenido que enfrentar y solucionar por cuenta propia. El hecho de desarrollar una idea propia desde cero no es nada fácil, ya que son muchas las cosas que uno ha de tener en cuenta para tomar las decisiones correctas.

Además de haber aplicado algunos de los conocimientos aprendidos en el Grado de Ingeniería Informática, he aprendido mucho en cuanto a cómo llevar una idea a negocio desde el minuto cero, pasando por las diferentes fases hasta llegar a lanzar el producto al mercado. No solo he crecido técnicamente, sino que también he tocado otras áreas que me han abierto la mente para futuros proyectos.

Me he dado cuenta de que, para alcanzar los objetivos iniciales presentados al inicio del proyecto, había que realizar un laborioso estudio previo antes de ponerse directamente a desarrollar.

Por el momento y al tratarse de un MVP, no se ha llegado a contactar con negocios para incluir planes de pago en la aplicación. La idea es ver primero si la aplicación es bien aceptada por los usuarios ofreciendo planes gratuitos y si hay que hacer cambios para satisfacer las necesidades reales. Una vez se llegue a un producto bien asentado, se empezará a enriquecer la base de datos de planes que los negocios nos vayan proporcionando.

Por otra parte, a lo largo del desarrollo del proyecto se han aprendido nuevas tecnologías y metodologías que pueden ser aplicadas en cualquier proyecto en el que se partícipe en un futuro.

A partir de este punto y si los siguientes pasos que se van a realizar son satisfactorios, se empezaría a buscar gente para formar un equipo con diferentes perfiles.

7. Trabajos futuros

Como funcionalidades futuras, se ha pensado en que un usuario pueda publicar un plan que vaya a hacer para que el resto de los usuarios pueda filtrar por planes publicados por gente. De esta forma, se podrán hacer planes con gente desconocida. En consecuencia, se integrará un chat para que los usuarios que vayan a realizar un plan puedan hablar todos los detalles.

Algo importante también a implementar serán las notificaciones para fidelizar a los usuarios. En cuando a la fidelización de los negocios, automatizaremos el proceso de incorporación de planes dentro de la plataforma para que se haga de una forma rápida y sencilla.

En cuando a la automatización de procesos, se va a incluir la herramienta Ionic Appflow[11], la cual nos va a permitir publicar las aplicaciones tanto en Google Play Store como en la App Store con un simple *click* sin tener que seguir arduos procesos de despliegue o automatizar cada fase de entrega para ofrecer integración y entrega continua.

[11] https://ionicframework.com/appflow

8. Bibliografía

[1] J. C. Alcalde, «Economipedia,» [En línea]. Available: https://economipedia.com/definiciones/metodo-lean-startup.html.

[2] L. Ordoñez, «Oleoshop,» 26 julio 2016. [En línea]. Available: https://www.oleoshop.com/blog/que-es-el-lean-canvas-y-como-implementarlo.

[3] Optasy, «Medium,» marzo 2020. [En línea]. Available: https://medium.com/@OPTASY.com/what-hybrid-app-development-framework-should-you-use-for-your-projects-in-2020-top-3-999662b30cce.

[4] S. Overflow, «Stack Overflow,» febrero 2020. [En línea]. Available: https://insights.stackoverflow.com/survey/2020.

[5] «Principios del Manifiesto Ágil,» [En línea]. Available: https://agilemanifesto.org/iso/es/principles.html.