UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Aedo, Aplicación de experiencias turísticas entre usuarios: Backend

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Mut Portes, Andreu

Tutor/a: Bort Mir, Lorena

CURSO ACADÉMICO: 2022/2023

[Título]

El proyecto “AedoApp” ha sido realizado en conjunto por cuatro alumnos de la Escuela Técnica Superior de Ingeniería Informática (ETSINF) de la Universidad Politécnica de Valencia en el campus de Vera durante el curso 2022/2023. Este proyecto se ha desarrollado en el marco de Start.inf el espacio de emprendimiento de la ETSINF.

Siguiendo las recomendaciones que ofrece la ETSINF para este tipo de proyectos de emprendimiento, el trabajo se ha realizado de manera conjunta, pero cada alumno presenta de forma separada su TFG asociado a la parte técnica del trabajo en la que ha sido más protagonista. Sin embargo, por ser el mismo proyecto y para que la memoria esté autocontenida, varios apartados del TFG son comunes y han sido elaborados de forma colaborativa. Ellos son: evaluación y desarrollo de la idea de negocio y los anexos “Manual de uso de la aplicación”, “Preguntas de la encuesta a los usuarios”, “Experimentos realizados” y “Objetivos de desarrollo sostenibles”.

Los alumnos participantes en el proyecto “AedoApp” y las áreas de trabajo que cada uno presenta en su TFG son:

- Hernández Bonet, Álvaro: Inteligencia artificial
- Matarredona Coloma, Joan: Frontend de la aplicación móvil
- Mut Portes, Andreu: Backend de la aplicación
- Palacios Martínez, Diego: Frontend de la aplicación web

C2 – Uso Restringido



# Resumen

En el presente documento se expone el proceso de emprendimiento por el que se va a desarrollar una aplicación móvil con su aplicación web para concebir una idea de negocio aplicada al sector turístico. La idea surge de la necesidad de diferenciar el turismo convencional de otros tipos de turismo, como el turismo alternativo, el cultural e incluso el turismo lento; intentando que los usuarios/as obtengan unas experiencias más personalizadas que podrán disfrutar y recordar. La aplicación fomenta la movilidad y el turismo por localidades de la España profunda. En el transcurso de la memoria se detalla la evaluación y el desarrollo de la idea de negocio para poder llevar a cabo la aplicación, y cómo se ha implementado el *backend*, la parte de una aplicación que no queda visible de cara al usuario.

**Palabras clave:** turismo, experiencias personalizadas, movilidad, España profunda, *backend.*

# Abstract

The following document presents the entrepreneurial process followed to develop a business idea applicable to the tourism sector, in which a mobile and a web application will be developed. The idea emerged from the need to find an alternative to conventional tourism from other kinds, such as cultural tourism, rural tourism, or mountain tourism in which users are able to enjoy and remember customized experiences. The app promotes personal mobility and visiting locations of the deep rural Spain. This report details the evaluation and development of the business idea that was needed to build the app and how the backend, that is, the part of the app that users are not aware of, has been implemented.

**Keywords:** tourism, customized experiences, personal mobility, deep rural Spain, backend.

# Índice general

**Contenido**

# Índice de figuras

# Índice de tablas

# 1. Introducción

## 1.1 Introducción

Cuando se piensa en una ciudad o una región, se suelen destacar una serie de lugares o experiencias de mayor calado turístico que la gran mayoría de las personas conoce, pero ¿qué ocurre con todas las tradiciones y la cultura que se esconde tras esos sitios?

Actualmente, existen en el mercado numerosas aplicaciones para ayudar al viajero/a que no sabe qué actividades realizar en la localidad de destino a la que se dirige. Estas aplicaciones ofrecen, por lo general, una serie de tours convencionales que suelen dejar una imagen sesgada de la cultura, las tradiciones o la realidad de la sociedad de una ciudad o una región. Un/a turista que visita la ciudad de Valencia suele ir a la Ciudad de las Artes y las Ciencias, a la Catedral, a la Plaza del Ayuntamiento, a probar una paella en la playa... pero ¿de dónde viene ese arroz que se ha comido en la paella, qué importancia tiene la preparación de la paella para la cultura valenciana, o qué rito gastronómico supone la típica reunión familiar de los domingos para degustar ese plato? ¿Qué supone el "esmorzaret" para un valenciano? ¿Cuál es un típico "esmorzaret"? Todas estas cuestiones son las que al final marcan el carácter y la tradición de una sociedad, y estos elementos no tienen cabida en la típica visita a la ciudad.

Aedo nace de la necesidad y de la importancia de poder ofrecer unas actividades turísticas de calidad a los viajeros/as, que permitan poner en valor las tradiciones y la cultura de cualquier lugar, por pequeño o inhóspito que sea. Además, ofrece una plataforma a todas estas localidades para poder publicitar la riqueza de sus tierras, atrayendo a nuevos visitantes con todas las ventajas que esto puede suponer.

## 1.2 Objetivos

La finalidad de este trabajo es realizar un proyecto de emprendimiento que ofrezca como resultado una aplicación que pueda ser usada por cualquier usuario/a y en la que pueda encontrar actividades (gastronómicas, culturales, al aire libre...) que se ofrezcan en distintas poblaciones de España realizadas por gente local. Para conseguir este producto se han definido una serie de objetivos comunes a todos los miembros del equipo que habría que alcanzar durante el desarrollo de la aplicación:

- Elaborar, desarrollar y evaluar una idea de negocio teniendo en cuenta el mercado actual.
- Obtener una aplicación que cumpla con las características establecidas por el equipo de desarrollo.
- Comprobar mediante experimentos que el producto software resulta del agrado de los usuarios finales (y sería usado por los mismos).

Por otra parte, se presentan a continuación los objetivos propios del autor de este documento, que son:

- Generar una estructura de código de calidad adecuada a las necesidades de la aplicación.
- Utilizar en el proyecto tecnologías existentes para facilitar la tarea de programación.
- Conseguir eficiencia en las consultas sobre la base de datos y la interacción de la aplicación con la capa de persistencia.
- Ofrecer una correcta cobertura y deuda técnica reducida utilizando herramientas de mantenimiento de software y análisis de código.

## 1.3 Estructura de la memoria

El presente documento expone una idea de negocio y el desarrollo posterior de una aplicación que va a ser el producto software resultante. Los distintos apartados de la memoria van a ser los siguientes:

- Apartado 2: Se realiza el resumen de la aplicación junto a un estudio de la idea de negocio y de mercado para comparar la aplicación con el *software* competidor. Además, se presenta el modelo de negocio, una proyección económica, en análisis de la matriz DAFO y una serie de conclusiones.
- Apartado 3: Se desarrolla la idea de negocio con los dos productos *MVP (Minimum Viable Product)* y sus respectivos experimentos.
- Apartado 4: Explica los distintos aspectos técnicos de la aplicación, como el modelo de datos, las herramientas, la arquitectura, patrones, pruebas y desafíos de programación.
- Apartado 5: Expone la cronología seguida durante los meses empleados para la realización del trabajo.
- Apartado 6: Aparecen las conclusiones del trabajo realizado en base a los objetivos propuestos, así como las limitaciones del mismo y el posible trabajo futuro que se podría realizar con la aplicación.

El documento finaliza con las referencias bibliográficas consultadas durante el trabajo y un apartado de anexos sobre el funcionamiento general de la aplicación, las preguntas de las encuestas, los experimentos realizados y los objetivos de desarrollo sostenibles.

# 2. Evaluación de la idea de negocio

## 2.1 Resumen de Aedo

Antes de explicar el contenido de la aplicación, es importante explicar la terminología utilizada en el ámbito del proyecto. Empezando por el propio nombre, Aedo proviene del vocablo griego que hacía referencia a los personajes conocidos en la cultura occidental como juglares, y se ha utilizado como un símil a esa tradicional vía oral de transmisión de información, de noticias, tradiciones etc. En este marco histórico, el usuario que utilice la aplicación se convertirá en un "Odiseo", el cual deberá juntarse con el Aedo para que le transmita ese conocimiento ancestral, y en el momento en el que el Odiseo haya protagonizado esa aventura del saber, podrá convertirse a su vez en un Aedo, al publicar sus propias experiencias.

Aedo es una aplicación tanto móvil como web que permite a cualquier usuario publicar y reservar experiencias turísticas. Estas experiencias pueden ser de cualquier calado, aunque la idea subyacente es que se realicen actividades que pongan en valor las tradiciones, la cultura, la gastronomía, etc. de los lugares en los que se realizan.

El principal atractivo de la aplicación es la posibilidad de realizar actividades ofertadas por personas locales de los lugares que se desean visitar, ayudando a paliar problemáticas como la despoblación, las desigualdades laborales o la perdida de trabajos artesanales.

## 2.2 Idea de negocio

La idea de negocio se trata del producto o el servicio que se ofrece al mercado. Dicho producto puede existir en el mercado actual o ser creado. Después se le aporta una propuesta de valor para el mercado al cual se dirige, es por ello que dicha solución puede generar beneficios para el impulsor del producto y otros beneficios para el usuario final de la aplicación.

Para poder elaborar la idea de negocio de una manera más sencilla y visual, se ha utilizado le técnica del *Lean Canvas* (Maurya, 2022) *ya* que el *Lean Canvas* es una herramienta de visualización de modelos de negocio rápida porque en una única hoja, de un vistazo, se dispone de toda la información importante y necesaria para llevar a cabo la idea de negocio. El tablero *Lean Canvas* se divide en nueve apartados: clientes, problema, proposición de valor, solución, canales, ingresos, costos, métricas y la ventaja competitiva.

Como resultado de la aplicación de esta técnica, se obtiene la Figura 1, la cual recoge todos los apartados mencionados anteriormente. Del tablero cabría destacar los dos puntos que resultan más interesantes: la proposición de valor y la ventaja competitiva, ya que tienen un contenido común, las experiencias. La ventaja competitiva de Aedo son las experiencias personales que se transmiten de persona a persona junto a la gamificación y el chat interno, mientras que la proposición de valor contempla la realización de experiencias para aprender tradiciones de unos a otros para que estas no se pierdan lo cual fomenta el turismo de zonas despobladas, que es donde suelen estar las tradiciones.

Esta versión del Lean Canvas de la Figura 1 corresponde a la etapa inicial de la idea de negocio de la aplicación y podría sufrir variaciones a lo largo del tiempo, ya que la industria del *software* está en constante cambio y es muy posible que una idea que a priori resultaba innovadora después sea implementada por otras aplicaciones competidoras. Así pues, la ventaja competitiva planteada inicialmente podría dejar de serlo.

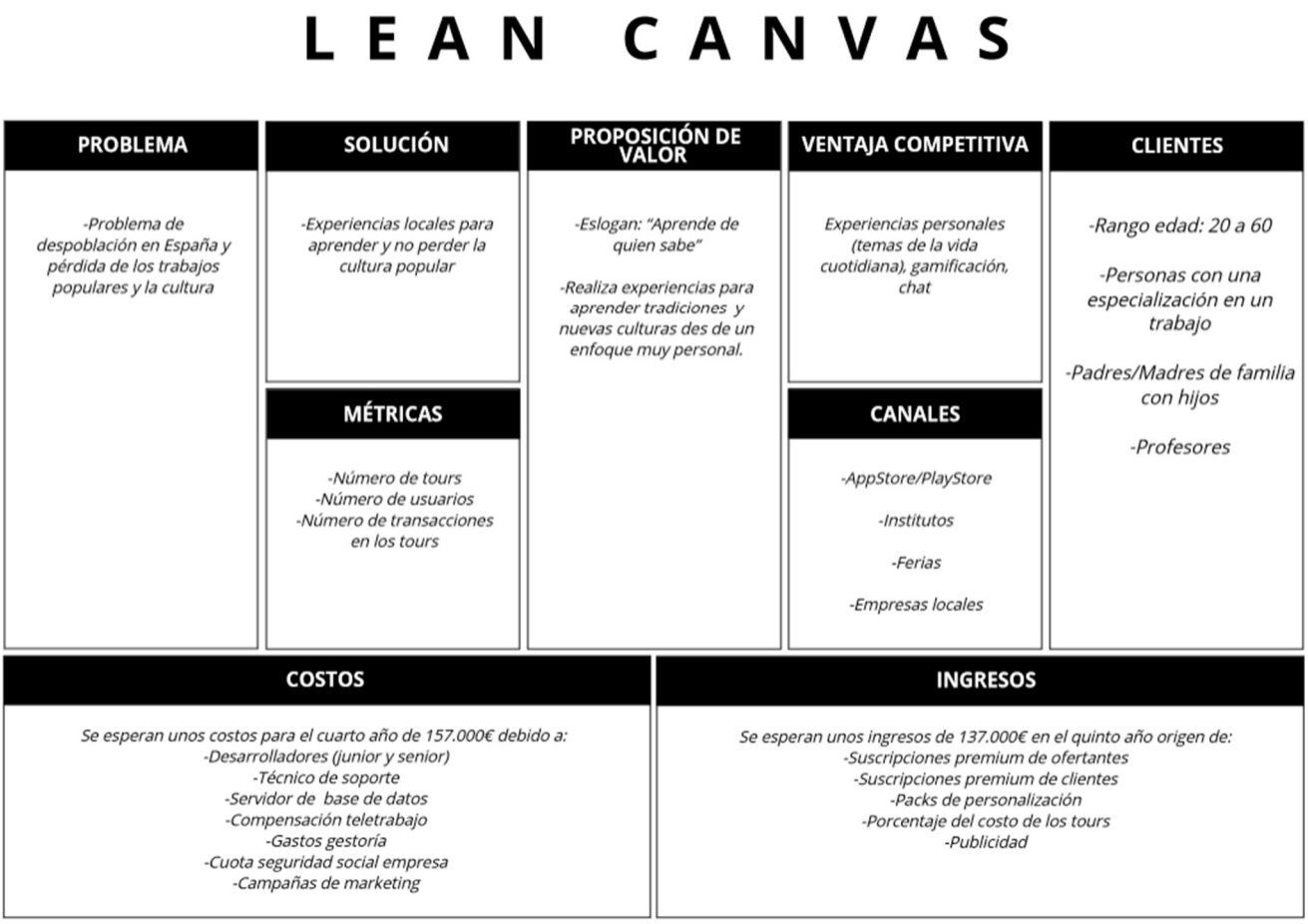



***Figura 1:*** *Lean canvas Aedo App (Fuente: Elaboración propia)*

## 2.3 Estudio de mercado

Tras observar las características principales de la aplicación, se realizó un estudio de las posibles alternativas competidoras. Se ha hecho una elección basándose en la similitud del objetivo final de la herramienta, destacando los puntos en común y las diferencias respecto a Aedo.

### 2.3.1 Aplicación competidora: Civitatis

Esta aplicación (Figura 2) está centrada en destinos de viajes a las principales ciudades y destinos de las mismas. Es una aplicación que tiene más de 500 mil descargas en Google Play;

esto la convierte en uno de los competidores más importantes. Además, también incorpora la opción de buscar tours cerca de la localización actual.

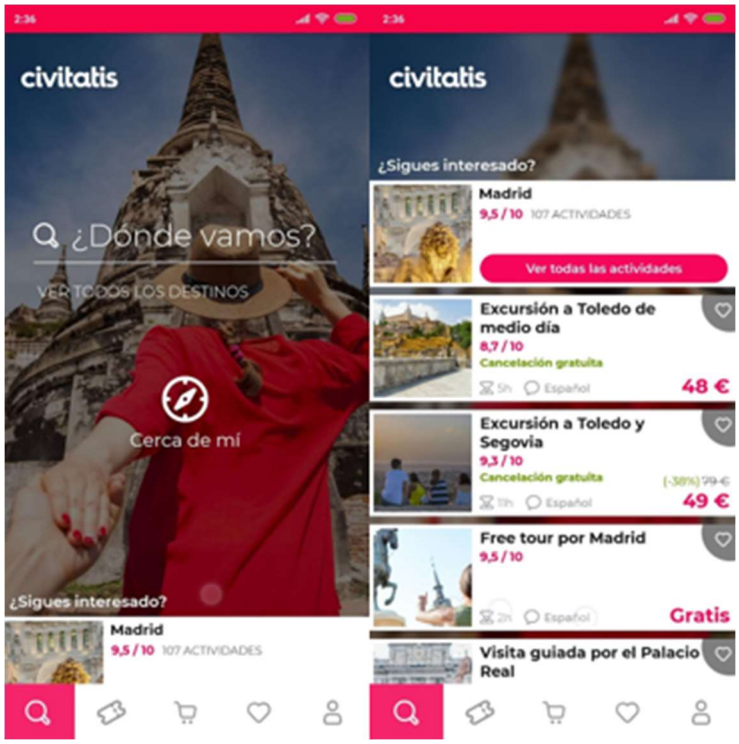



***Figura 2**: Web Civitatis (Fuente: civitatis.com/es/)*

Civitatis cuenta con una aplicación específica con guías para varias ciudades europeas para centrarse más en el destino elegido por los usuarios (Figura 3).

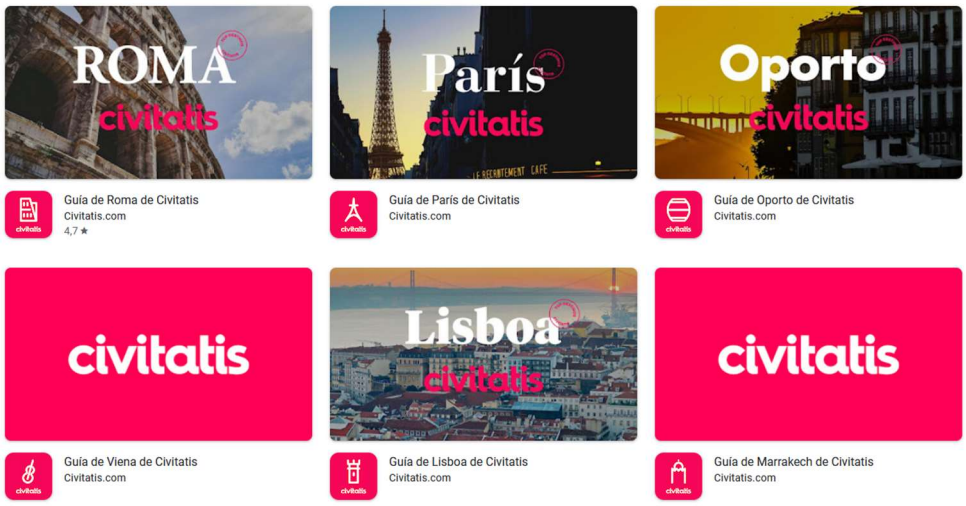



***Figura 3**: Web Civitatis (Fuente: civitatis.com/es/)*

### 2.3.2 Aplicación competidora: Guruwalk

Guruwalk (Figura 4) está centrada en ***free tours*** por las principales ciudades del mundo. Según ellos, se encuentran más de **2300** *free tours* en español, inglés y otros idiomas. Un *free tour* es un tipo de visita turística en la que el viajero paga lo que quiere al final del recorrido, según su satisfacción y presupuesto.

Esta app establece un sistema de propinas: los/as turistas pueden dar una propina a su guía turístico/a al final del tour si están satisfechos/as con la experiencia. Además, fomenta el turismo responsable, sostenible y cultural apoyando a los guías locales, promoviendo el contacto directo y contribuyendo a la diversificación y la desestacionalización de la oferta turística, al ofrecer *free tours* en lugares menos masificados y durante todo el año.


***Figura 4**: Captura de GuruWalk (Fuente: Google play GuruWalk)*

### 2.3.3 Aplicación competidora: Freetour.com

FreeTours.com (Figura 5) es una plataforma online que ofrece tours gratis o de bajo coste con guías locales en más de 120 países. Tiene un sistema de valoración y comentarios de los usuarios/as que ayuda a elegir el mejor tour y a mejorar la calidad del servicio.

Según SimilarWeb, FreeTour.com tiene un tráfico mensual estimado de 1.3 millones de visitas en los últimos 6 meses, con un promedio de 3.5 páginas vistas por visita y un tiempo de permanencia de 4:28 minutos.

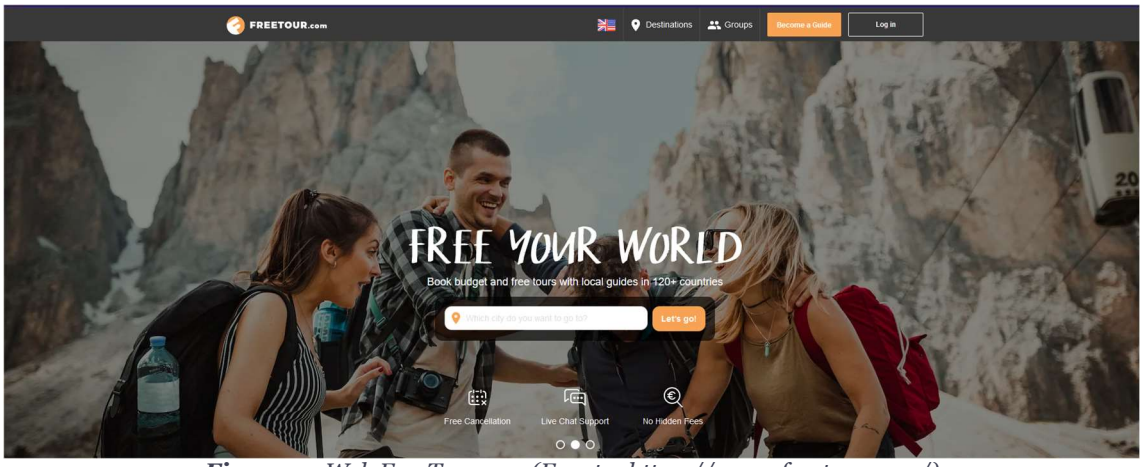



***Figura 5:*** *Web FreeTour.com(Fuente: https://www.freetour.com/)*

Además, también dispone de una versión móvil (Figura 6) con más de 100k descargas en Google play.

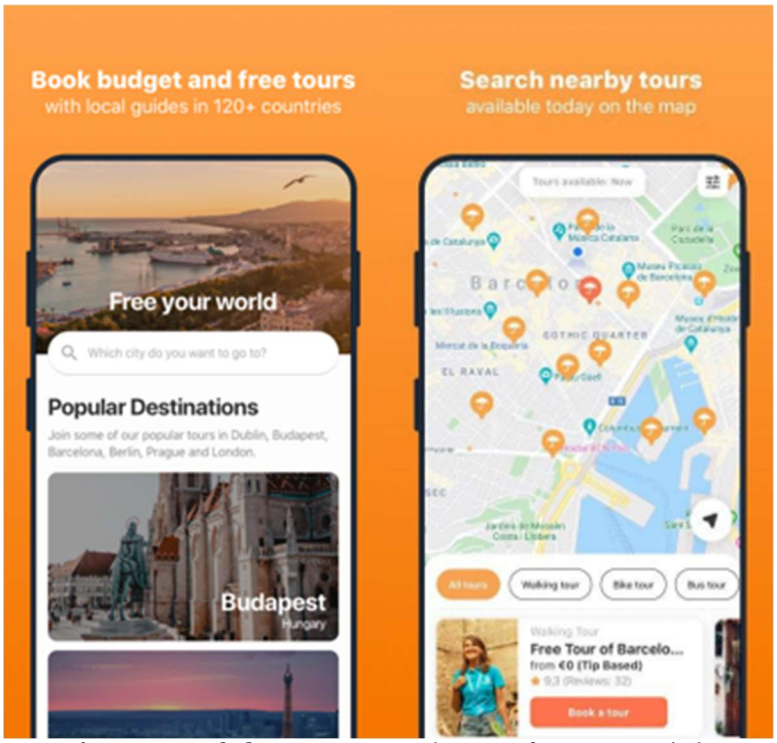



***Figura 6****: Web de FreeTour.com (Fuente: freetour.com/es)*

### 2.3.4 Aplicación competidora: Identify

La *app* de turismo cultural Identify (Figura 7) es una guía digital que permite descubrir más de 300.000 puntos culturales en todo el mundo. Se puede personalizar la experiencia según

los intereses y la geolocalización del usuario/a, así como participar en un juego donde se ganan recompensas y premios por identificar elementos culturales y compartirlos con otros usuarios/as.


***Figura 7**: Web Identify (Fuente: identifytravel.app)*

### 2.3.5 Aplicación competidora: Withlocals

Withlocals (Figura 8) es una plataforma que conecta a viajeros/as con gente local que ofrece comida y experiencias únicas. Se puede disfrutar de una ciudad como un lugareño, conocer su cultura y sus secretos, y apoyar el turismo sostenible. Hay más de 100 destinos disponibles en todo el mundo. Las experiencias son guiadas por expertos locales que conocen la historia, la cultura y las normas sanitarias de cada lugar.


***Figura 8**: Withlocals (Fuentewithlocals.com/es/)*

### 2.3.6 Tabla comparativa

Tras analizar las distintas aplicaciones competidoras, se ha confeccionado una tabla comparativa (Tabla 1) en la que se comparan una serie de funcionalidades típicas en este tipo de aplicaciones y se tiene un registro de, por cada programa, cual implementa las características analizadas.

***Tabla 1**:Tabla comparativa Aedo con otras aplicaciones. (Fuente: Elaboración propia)*

| **Característica:** | **Civitatis** | **Guruwalk** | **FreeTour.com** | **Identify** | **Withlocals:** | **Aedo** |
|---|---|---|---|---|---|---|
| Buscador | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Usuarios registrados consumidores | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Usuarios registrados productores | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Filtros | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Compras integradas | ✓ | | | | ✓ | ✓ |
| Soporte técnico | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Diferentes idiomas de interfaz | ✓ | | | ✓ | ✓ | Español, Inglés |
| Diferentes idiomas del tour | | ✓ | ✓ | | ✓ | ✓ |
| Distintos tipos de tours | Entradas, espectáculos, escapadas gastronómicas, transportes | Alternativo, arte urbano, bici, leyendas ... | Tours gratis, diarios, urbanos, bares, bicicleta, gastronómicos | Sólo muestra puntos de interés | Gastronómicas, bici, arte, música, nocturnas, ... | ✓ |
| Sin gastos de reserva | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Servicio de la APP gratuito | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Ver fotos, reseñas, detalles, descripciones y disponibilidad del tour | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| | | | | | | |
|---|---|---|---|---|---|---|
| (quien oferta el tour) | | | | | | |
| Calificar y comentar el tour/guía | ✓ | ✓ | ✓ | | ✓ | ✓ |
| *Wishlist* de tours | ✓ | | ✓ | | | ✓ |
| Encontrar tours por cercanía (a la localización actual) | ✓ | | ✓ | ✓ | | ✓ |
| Oferta de entradas a espectáculos | ✓ | | | | | |
| Recursos para estudiantes y centros educativos | | | | ✓ | | ✓ |
| Rutas personaliza-das | | | | ✓ | | ✓ |
| Verificación de cuenta | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Valoración del ofertante sobre los clientes | | | | | | ✓ |
| Chat Bot | | | | | | ✓ |
| Gamifica-ción | | | | | | ✓ |

Así pues, se puede observar en la Tabla 1 que hay una serie de características comunes a todas las aplicaciones, como pueden ser buscador, registro de usuarios, soporte técnico, ver fotos y reseñas y la verificación de cuenta. Después, existen algunas características que están en prácticamente todas las aplicaciones como pueden ser los filtros, las compras integradas, uso gratuito de la app o publicar comentarios, por ejemplo. Finalmente están las características propias de la aplicación Aedo, las cuales no están contempladas en la competencia en la actualidad y son: la gamificación, un chat interno, un chat bot para soporte técnico y la valoración por parte de la persona que explica la experiencia sobre los clientes que asisten. Además, Aedo ofrece recursos para estudiantes y centros educativos y rutas personalizadas, característica solo ofrecida por Identify. Sin embargo, si se tuviera que remarcar el elemento diferenciador de Aedo, sería la posibilidad de que cualquier usuario pueda publicar actividades, y que no se restrinja la creación de estas a la empresa propietaria de la aplicación. Se ha considerado que esta es la puesta en valor del proyecto respecto a la competencia, y también la característica que más puede favorecer a los objetivos de desarrollo sostenible perseguidos por la aplicación.

## 2.4 Modelo de negocio

Existen distintos modelos de negocio que se podrían aplicar en Aedo pero, al tratarse de una aplicación móvil, algunos de ellos no serían bien recibidos. Es por esa razón que, de antemano, el modelo de negocio tradicional, basado en la venta de licencias en la que se adquiere el producto por un precio y se dispone de total libertad para utilizarlo, no encaja con la aplicación. La razón por la que se descarta es que Aedo necesita de una amplia base de datos de usuarios para poder ofertar experiencias por parte de las personas que quieren ofrecerlas y de clientes que asistan a dichas experiencias. Si los/as usuarios/as no utilizan la aplicación, no hay tránsito y por tanto la aplicación no aumentará el número de usuarios activos. Muchos de estos usuarios rehusarían de la aplicación si tienen que pagar por ella, además que probablemente otra empresa podría vislumbrar el potencial de la aplicación y la implementaría con otro modelo de negocio.

Es por esta razón que la opción que se considera acertada para el proyecto es un modelo *freemium,* en el que los usuarios pueden descargar el *software* de manera gratuita y utilizarlo para, posteriormente, adquirir los servicios *premmium* si así lo desean. Entre las distintas posibilidades de opciones de compra se han pensado las siguientes:

- **Suscripción *premmium* para los ofertantes**: Se trata de un servicio para los/as usuario/as que ofertan experiencias. Mediante este servicio se podrían posicionar las experiencias en el buscador de otra manera. A su vez, permitiría añadir más imágenes a la hora de crear las experiencias para que los usuarios puedan tener más información sobre lo que se van a encontrar. Además, desaparecen los anuncios publicitarios.
- **Suscripción *premmium* para los clientes**: Un cliente *premmium* no tiene anuncios publicitarios en su cliente móvil y podría tener prioridad a la hora de acceder a una experiencia para un día dado si dicha experiencia tiene el cupo de usuarios completo. Un usuario *premmium* se beneficia de prioridad en la cola. También se anularía la publicidad para este tipo de usuarios.
- **Packs de personalización**: Es común en las aplicaciones dar la posibilidad a los/as usuarios/as de adquirir elementos para su perfil de usuario, que son visibles desde la aplicación y los distinguen de otros usuarios que no han adquirido la membresía *premmium* como pudiera ser un icono o alguna insignia.

Por otra parte, también se consideraría como fuentes externas de ingresos la implementación de anuncios publicitarios en la aplicación y una pequeña comisión del precio de las experiencias ofertadas en la plataforma cada vez que un cliente se apunta a las mismas.

## 2.5 Proyección económica a 6 años

Para realizar la estimación de ingresos y gastos durante los seis primeros años, se ha dividido este período de tiempo en doce semestres. Se parte de una pequeña cantidad de usuarios para el primer semestre estimada en 500 hasta llegar a 60.000 usuarios activos en el semestre doceavo.

La Figura 9, muestra la proyección económica preestablecida para el proyecto en cuestión, mostrando los diferentes gastos desglosados y los previstos ingresos según la fuente proveniente.

Por una parte, los **ingresos** provienen, como se ha explicado en el apartado de **modelo de negocio** de este documento, de distintas fuentes a ser las suscripciones *premmium* para ofertantes y clientes, los packs de personalización, las comisiones de las experiencias y la publicidad. Cabe destacar que no se prevén ingresos publicitarios hasta el semestre décimo momento en el cual ya se dispone de una base de usuarios activos suficiente como para ser rentable que los anunciantes publiciten sus productos. Es por esta razón que resulta tan complicado adquirir ese punto de equilibrio en el que se empiezan a generar beneficios.

Por otra parte, los **gastos** se deben a distintos servicios de *hosting* necesarios para el funcionamiento de la aplicación, a la adquisición de los dispositivos necesarios para realizar el desarrollo del *software* y los salarios de los trabajadores, además de campañas de *marketing*.

Analizando la Figura 9 se puede apreciar que el primer año los gastos van a ser constantes durante los dos semestres y los ingresos van a ser prácticamente nulos debido a la carencia de usuarios, es por ello por lo que se optará por una campaña de publicidad más agresiva de cara al segundo año.

En el segundo año, el número de usuarios se incrementa, y con ello los ingresos. Los gastos también aumentarán debido a los costes de gestoría y a la campaña publicitaria. Dicha campaña publicitaria se refleja en el tercer año, en el que se multiplica por cinco la cantidad de usuarios respecto a inicios del segundo año. Esto permite aumentar aún más los ingresos, pero a su vez se requerirá de la contratación de un desarrollador senior.

A partir de este momento, cuarto año, se prevé que junto a campañas publicitarias y el boca a boca de los usuarios el número de usuarios activos seguirá aumentando y en este momento la empresa empezará a generar beneficios semestrales, pero no será hasta el quinto año donde se equilibren los gastos y beneficios.

Como se puede apreciar en la Figura 9, la previsión establece un *bypass* semestral entre ingresos y gastos a partir del segundo semestre del cuarto año. Esta situación se generalizaría en el acumulado a partir del segundo semestre del quinto año, obteniendo unos beneficios aproximados en dicho plazo de unos 82.000€. En la tabla también aparecen desglosados los diferentes gastos e ingresos proyectados para dicho proyecto y que se han considerado estándares para un trabajo de esta complejidad.

| Usuarios activos | 500 | 1000 | 2000 | 6000 | 10000 | 15000 | 20000 | 30000 | 40000 | 50000 | 55000 | 60000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | |
| **Número de Semestre** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| **Tipos de suscripciones** | **Año 1 / S1** | **Año 1 / S2** | **Año 2 / S1** | **Año 2 / S2** | **Año 3 / S1** | **Año 3 / S2** | **Año 4 / S1** | **Año 4 / S2** | **Año 5 / S1** | **Año 5 / S2** | **Año 6 / S1** | **Año 6 / S2** |
| Suscripción premium ofertantes | 1500 | 3000 | 6000 | 18000 | 30000 | 45000 | 60000 | 90000 | 120000 | 150000 | 165000 | 180000 |
| suscripción premium para clientes | 200 | 400 | 800 | 2400 | 4000 | 6000 | 8000 | 12000 | 16000 | 20000 | 22000 | 24000 |
| Packs de personalización | 50 | 100 | 200 | 600 | 1000 | 1500 | 2000 | 3000 | 4000 | 5000 | 5500 | 6000 |
| porcentaje odiseas | 90 | 180 | 360 | 1080 | 1800 | 2700 | 3600 | 5400 | 7200 | 9000 | 9900 | 10800 |
| publicidad | | | | | | | | | | 33600 | 36960 | 40320 |
| | | | | | | | | | | | | |
| **Total Ingresos Semestrales** | **1840** | **3680** | **7360** | **22080** | **36800** | **55200** | **73600** | **110400** | **147200** | **217600** | **239360** | **261120** |
| | | | | | | | | | | | | |
| **Gastos Anuales** | **Año 1 / S1** | **Año 1 / S2** | **Año 2 / S1** | **Año 2 / S2** | **Año 3 / S1** | **Año 3 / S2** | **Año 4 / S1** | **Año 4 / S2** | **Año 5 / S1** | **Año 5 / S2** | **Año 6 / S1** | **Año 6 / S2** |
| Servidor BBDD | 100 € | 200 € | 400 € | 1.200 € | 2.000 € | 3.000 € | 4.000 € | 6.000 € | 8.000 € | 10.000 € | 11.000 € | 12.000 € |
| mobiles IOS / Android | 800 € | 0 € | 0 € | 0 € | 800 € | 0 € | | | | | | |
| Compensación teletrabajo | 3.600 € | 3.600 € | 3.600 € | 3.600 € | 7.200 € | 7.200 € | 7.200 € | 7.200 € | 7.200 € | 7.200 € | 7.200 € | 7.200 € |
| Gestoría | 0 € | 0 € | 1.000 € | 1.000 € | 2.000 € | 2.000 € | 2.000 € | 2.000 € | 2.000 € | 2.000 € | 2.000 € | 2.000 € |
| Desarrollador Senior Android / IOS | 0 € | 0 € | 0 € | 0 € | 20.000 € | 20.000 € | 20.000 € | 20.000 € | 20.000 € | 20.000 € | 20.000 € | 20.000 € |
| Desarrollador junior | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € |
| Desarrollador junior | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € | 12.500 € |
| Tecnico de soporte | 0 € | 0 € | 0 € | 0 € | 10.000 € | 10.000 € | 10.000 € | 10.000 € | 10.000 € | 10.000 € | 10.000 € | 10.000 € |
| Cuota seguridad social empresa | 300 € | 300 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € | 1.200 € |
| Campaña marketing | 2000 | 2000 | 4000 | 6000 | 6000 | 5000 | 5000 | 5000 | 5000 | 5000 | 5000 | 5000 |
| | | | | | | | | | | | | |
| **Total Gastos** | **31.800 €** | **31.100 €** | **35.200 €** | **38.000 €** | **74.200 €** | **73400** | **74400** | 76400 | 78400 | 80400 | 81400 | 82400 |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| **Resultado Semestral** | -29.960 € | -27.420 € | -27.840 € | -15.920 € | -37.400 € | -18.200 € | -800 € | **34.000 €** | 68.800 € | 137.200 € | 157.960 € | 178.720 € |
| | | | | | | | | | | | | |
| **Resultado Semestral Acumulado** | -29.960 € | -57.380 € | -85.220 € | -101.140 € | -138.540 € | -156.740 € | -157.540 € | -123.540 € | -54.740 € | 82.460 € | 240.420 € | 419.140 € |

***Figura 9**: Proyección económica (Fuente: Elaboración propia.)*

La Figura 10 muestra el balance de pérdidas/beneficios acumulados por semestre que se han explicado en el documento. En ella se pueden observar las pérdidas de los primeros semestres hasta llegar al punto de equilibrio en el semestre diez (segundo semestre del quinto año), donde la aplicación empezará a generar beneficios.


***Figura 10:** Resultado semestral acumulado (Fuente: elaboración propia)*

## 2.6 Análisis DAFO

La Tabla 2, conocida como matriz DAFO[1], establece las Debilidades, las Amenazas, Fortalezas y Oportunidades que se han observado para el proyecto. En este caso se debe destacar la poca experiencia del equipo, equilibrada por el potencial del proyecto para diferentes sectores de mecenazgo de la sociedad (instituciones, organizaciones etc.) Además, se ha considerado positivo el hecho de contar con un equipo reducido para potenciar la facilidad en la toma de decisiones.

***Tabla 2:*** *Matriz DAFO (Fuente: Elaboración propia.)*

| **Debilidades** | **Amenazas** |
|---|---|
| • Equipo junior con poca experiencia.<br>• Poca financiación en el proyecto inicial.<br>• Equipo reducido.<br>• Dirección estratégica inicial poco establecida. | • Entrada de empresas competidoras con mayor financiación.<br>• Cambios demográficos intensos.<br>• Crecimiento lento del mercado.<br>• Futuros cambios en la política económica de la empresa.<br>• Políticas adversas en otros países. |
| **Fortalezas** | **Oportunidades** |
| • Nicho con poca competencia.<br>• Funcionalidad novedosa.<br>• Capacidad de decisión limitada a un equipo pequeño.<br>• Políticas afines.<br>• Basado en ODS con posible financiación europea.<br>• Flexibilidad organizativa.<br>• Equipo joven con ideas modernas y atractivas. | • Afinidad con diferentes sectores de la población.<br>• Afinidades políticas con instituciones.<br>• Aprovechamiento del potencial turístico del país.<br>• Políticas territoriales alineadas con la visión de la empresa.<br>• Financiación a nivel estatal y europea.<br>• Crecimiento rápido de la popularidad de la aplicación. |

## 2.7 Conclusiones de la evaluación

A grandes rasgos, el presente documento ha pretendido mostrar la factibilidad de un proyecto con un gran nicho de mercado, con gran potencial y elementos distintivos, que, a pesar de requerir de un presupuesto inicial elevado, debido a la complejidad de este, el riesgo que ello supone es limitado en relación con las posibles oportunidades observadas.

El hecho de aunar en una aplicación diferentes características tan atractivas para diferentes sectores de la sociedad, como actividades de ocio, tiempo libre, medio ambiente, puesta en valor de tradiciones, cultura, etc.., se ha visto como un potente aliciente por parte del equipo para confiar en la buenaventura del proyecto y el compromiso con el mismo.

[1] DAFO: Debilidades, Amenazas, Fortalezas, Oportunidades

# 3. Desarrollo de la idea de negocio

A continuación, se muestra el mapa de características de la aplicación.

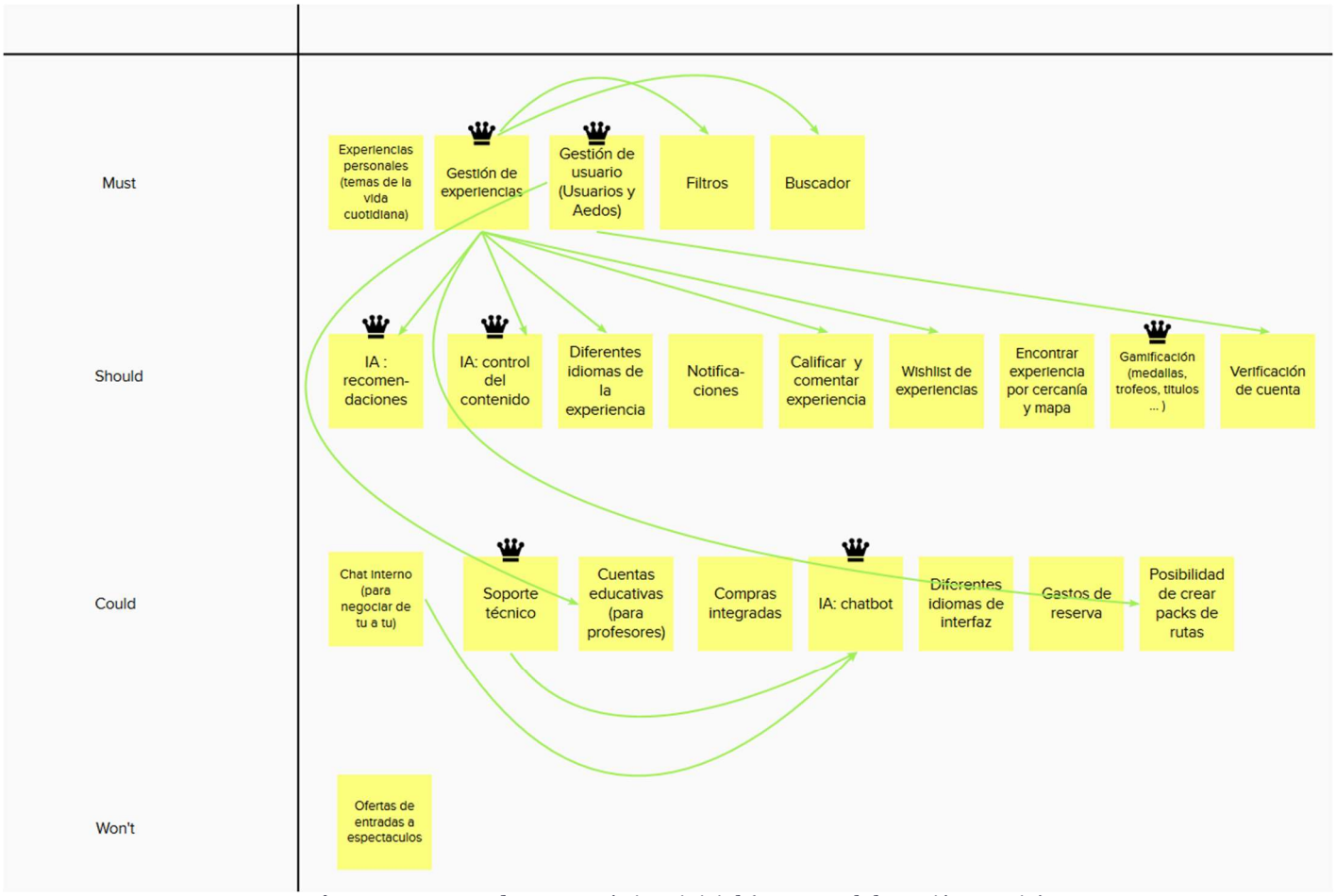



***Figura 11:*** *Mapa de características inicial (Fuente: Elaboración propia)*

En la Figura 11 podemos observar los diferentes elementos que se han considerado importantes a implementar en la aplicación durante su desarrollo hasta consolidarse como un producto maduro. También se ha realizado un análisis MOSCOW (Kuhn, 2009) para priorizar las diferentes características, basándose en criterios de funcionalidad y usabilidad.

Destacar funciones básicas como la gestión de experiencias o usuarios, que establecen la base de la aplicación, u otras más avanzadas como la IA de recomendaciones o de control de contenido, que darían un toque distintivo a la misma, mejorando la experiencia de uso de los usuarios.

## 3.1 Desarrollo del primer MVP

El desarrollo de la aplicación Aedo empezó en el marco de la asignatura de Proyecto de Ingeniería del Software, con una metodología preestablecida por el profesorado de esta. En ese ámbito se utilizó la herramienta Worki para gestionar el esfuerzo dedicado al proyecto y el trabajo individual de cada miembro. Se realizaron 3 Sprints de un mes aproximadamente, en el que se implementaron varias funcionalidades básicas del proyecto para posteriormente presentarlo en la Feria de Proyectos de la ETSINF. La elección de dichas funcionalidades se basó en un criterio MVP según las características que se consideraron básicas de la aplicación según la Figura 11.

El diseño de la aplicación fue desarrollado por el grupo de estudiantes de Bellas Artes integrados en el grupo de desarrolladores de software, técnica novedosa llevada a cabo en dicha asignatura para incrementar la cooperación entre estudiantes de diferentes facultades y crear sinergias de trabajo similares a una experiencia laboral real.

### 3.1.1 Experimento 1: Feria de proyectos

En la Feria de Proyectos se presentó la aplicación, y se estableció contacto con empresas y otros desarrolladores que la analizaron y opinaron sobre sus funcionalidades. De esta experiencia se extrajo el aprendizaje de la importancia de la viralización en una aplicación de este tipo y de las diferentes posibilidades de realizar experiencias en grupo, por personas desconocidas entre sí, y que decidieran conocerse realizando dicha actividad.


***Figura 12**: Fotografía del stand y el equipo de Aedo (Fuente: Fotografía de la Etsinf)*

## 3.2 Desarrollo del segundo MVP

El segundo *MVP* estuvo enfocado a aumentar las funcionalidades de la aplicación, elaborar una inteligencia artificial para las recomendaciones de experiencias, elaborar la página web de la aplicación para las gestiones administrativas y tener una versión básica de la aplicación en una versión de navegador. De entre las funcionalidades implementadas, cabría destacar el apartado educativo de la aplicación. Tras el segundo *MVP,* la aplicación permitía disponer de cuentas educativas y de generar experiencias colectivas para clases de alumnos enteras. También se incluyeron mejoras de interfaz y funcionalidades para conocer, de una manera más clara, cuántos usuarios/as estaban apuntados a una experiencia, mejoras en el perfil, categorías extra, etc.

Para el segundo y tercer experimento, tras tener la aplicación con las últimas funcionalidades implementadas, se presentó la aplicación a dos perfiles diferentes, pero a su vez enfocados en la creación de experiencias.

### 3.2.1 Experimento 2: Presentar la aplicación a un trabajador del campo.

Con el objetivo de conocer la opinión de un potencial creador de contenido de la aplicación, se contactó con un agricultor interesado en realizar talleres con personas aficionadas a las labores del campo. En este caso, la persona seleccionada era un conocido de uno de los miembros del equipo de desarrollo, con un perfil joven e interesado en la temática de la aplicación, con intención real de aportar contenido a la misma.

Teniendo en cuenta que uno de los objetivos de desarrollo sostenible defendidos por la cultura de la aplicación era el de “Ciudades y Comunidades Sostenibles”, se consideró un posible usuario muy apropiado para los intereses del proyecto.

De esta forma, se le presentó la aplicación y se le realizaron una serie de preguntas orientadas a conocer, de primera mano, los puntos positivos y negativos que un usuario primerizo podría observar de la aplicación. Las preguntas se realizaron verbalmente *in situ* al interesado, no obstante, también se le pidió que contestara a las preguntas de un cuestionario *online* a través de un enlace para tener el registro de las respuestas. El Anexo 2 recoge las preguntas del formulario y el Anexo 3 fotos del experimento. El resultado fue satisfactorio, destacando, por parte del encuestado, la interfaz intuitiva y la facilidad de uso. De igual forma, se recalcó la importancia de disponer de un chat de usuarios en el que poder responder a las dudas y cuestiones que le realizaran los posibles clientes. Dicha funcionalidad ya había sido valorada por el equipo de desarrollo, pero no se ha dispuesto de tiempo suficiente para presentarla en la presente versión.

### 3.2.2 Experimento 3: Presentar la aplicación a un profesor

Ya que uno de los objetivos perseguidos durante la segunda fase del desarrollo de la aplicación fue la implementación de un tipo de cuentas educativas, se consideró oportuno contactar con un profesor de secundaria para conocer su opinión al respecto. La elección de dicha persona vino condicionada por ser excompañero de clase de unos de los miembros del equipo y ser una persona joven con facilidad en el uso de tecnologías.

Se le explicó el proyecto y se le presentaron las funcionalidades de este, destacando el uso de las cuentas educativas y las opciones para reservar actividades para grupos escolares. Ya que el proyecto pretende poner en valor las tradiciones y la cultura, el encuestado recalcó la importancia de este tipo de actividades dentro del ámbito educativo de los centros de secundaria. También sugirió la posibilidad de exportar las descripciones de las actividades a pdf para poder compartirlas con los padres de los alumnos, y firmar los permisos para que los alumnos las realicen. El equipo de desarrollo valoró dicha opción, y se tiene pensado implementar un sistema de exportación a diferentes formatos y de compartir las actividades a través de aplicaciones de terceros en futuras versiones.

Se le facilitó al docente un cuestionario *online* (Ver Anexo 2) para conocer su opinión y registrar sus posibles sugerencias. Las imágenes del experimento pueden verse en el Anexo 3.

# 4. Aspectos técnicos

Se debe hacer una consideración inicial antes de empezar con la parte específica del *backend*. Como ya se ha comentado, el proyecto de Aedo nació en la asignatura de proyecto de ingeniería del software (PIN). Fue un proyecto multidisciplinar entre la escuela de informática y la escuela de bellas artes, esto es importante tenerlo presente de cara a la consideración.

Fue decidido por el equipo de programadores utilizar una terminología basada en la antigua Grecia para la aplicación de Aedo, es por ello que los usuarios mencionados hasta este momento en el documento, en el código se les conoce como “odiseos” y las experiencias en el código aparecerán como “odiseas”. Por tanto, siempre que aparezca la palabra odiseo en este presente documento, se refiere a los usuarios de la aplicación (los que no son aedos, los aedos son los usuarios que ofrecen las experiencias al resto de usuarios) y cuando aparezca odisea, realmente se refiere a las experiencias a las que los usuarios se inscriben.

Esta discrepancia de nombres tiene un motivo: A la hora de presentar el proyecto en la Feria de Proyectos se mostró la aplicación a los alumnos de bellas artes con unas semanas de antelación y se propuso cambiar el nombre de “odisea” por algo más amigable ya que la palabra odisea se relaciona con algo muy complicado o imposible de hacer y puede dar una mala impresión al usuario sobre a qué actividad se están inscribiendo. Como se tuvo que cambiar la palabra odisea, no tenía sentido mantener la palabra odiseo para los usuarios, sin embargo, todo el apartado de *backend* de la aplicación, como puedan ser las *APIs*, los modelos y los objetos, siguen manteniendo esta terminología antigua de odiseos y odiseas.

## 4.1 Modelo de datos

En la Figura 13 se puede ver representado el modelo de datos del proyecto, protagonizado por las Experiencias como pilar fundamental de la aplicación, con una serie de relaciones con las demás clases de este. Esta estructura está orientada a la optimización en la utilización de la base de datos no relacional *Firestore* (detallada en la sección 4.1), y a las necesidades mismas del correcto funcionamiento de la aplicación. Como se puede observar, todos los elementos propios de una aplicación de este tipo estarían representados, destacando las reservas, los comentarios, los usuarios, los diferentes idiomas disponibles, etc.

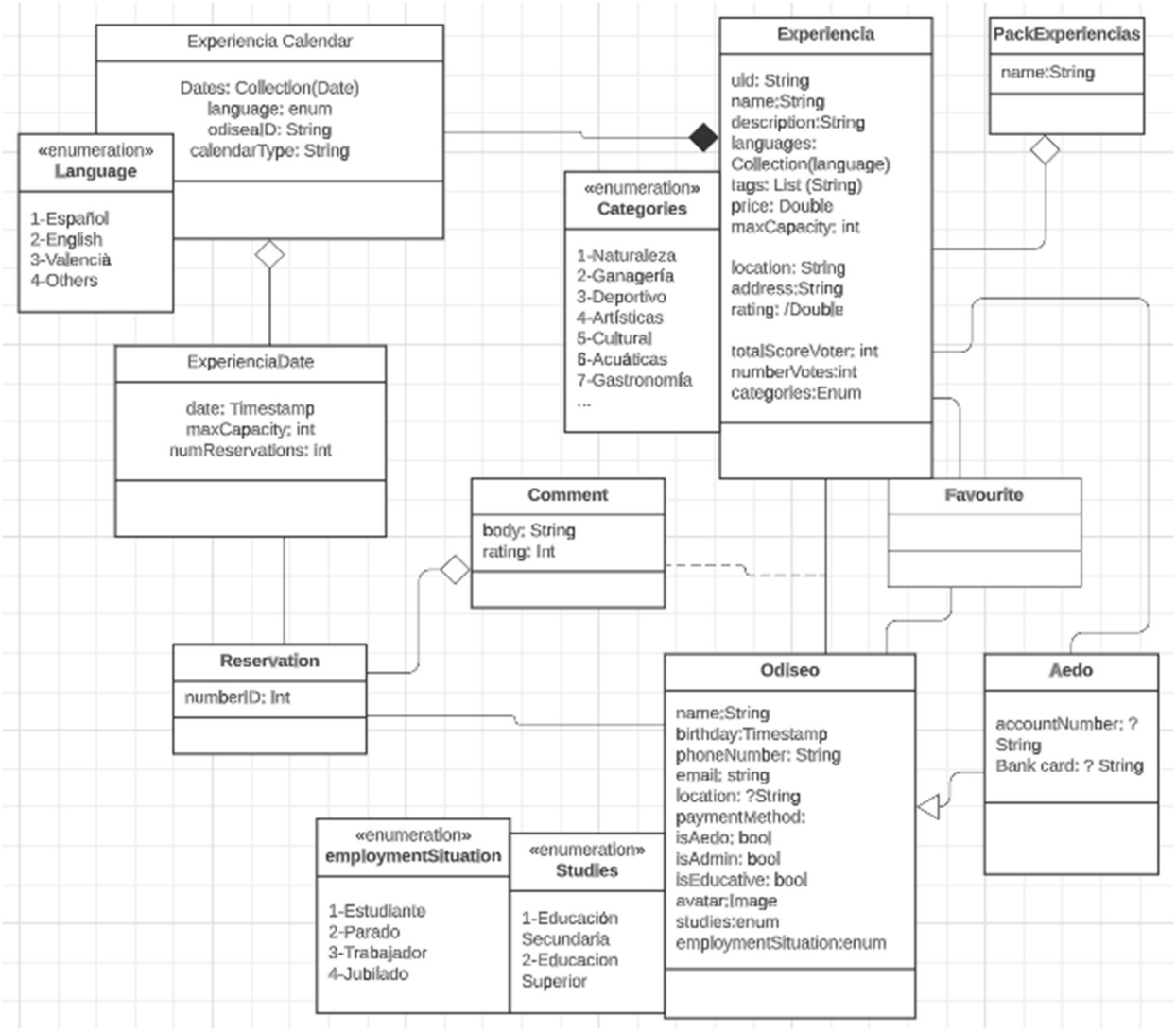



***Figura 13**: Modelo de datos (Fuente: Elaboración propia)*

## 4.2 Herramientas utilizadas

Las herramientas que se van a exponer en este apartado han sido imprescindibles para el desarrollo de la aplicación. Se va a explicar desde la base de datos escogida hasta las pruebas de cobertura, pasando por los lenguajes de desarrollo y el entorno de programación, la paquetería Npm de Node.js o la generación de documentación mediante JSDoc, entre otros.

### 4.2.1 *Firebase*

*Firebase*[2] es una plataforma para desarrollar aplicaciones respaldadas por Google. Dispone de distintas funcionalidades que ayudan a programar las aplicaciones sin tener que preocuparse por ciertos apartados, ya que ofrece una infraestructura de *backend* completamente administrada.

Se ha querido implementar *Firebase* en Aedo por las facilidades que aporta a la hora de tratar con el almacenamiento de imágenes, la autenticación de los usuarios, el servicio de *hosting* y la base de datos no relacional incluida en el servicio. *Firebase* dispone de un plan gratuito[3] de 50.000 lecturas diarias o 20.000 operaciones de escritura y/o borrado al día también. Es una cantidad más que suficiente para desarrollar la aplicación y para un número de usuarios pequeño; en el futuro se puede actualizar a otros planes cuando la aplicación disponga de más usuarios.

En cuanto a las razones por las cuales se ha escogido una base de datos no relacional como es *Firebase,* comentar que aunque la aplicación es de pequeño tamaño y en estos casos no tiene mucha influencia el que sea o no relacional, sin embargo, sí es cierto que para una aplicación con mayor número de lecturas que escrituras, una base de datos no relacional es más eficiente; y ese es el caso de Aedo en el que los usuarios realizan en su mayoría lecturas, y solo realizan escrituras cuando se registran ellos o experiencias, cuando guardan una experiencia a favoritos o cuando reservan. Por lo general, la mayoría de las operaciones son de lectura. Además, hasta la fecha sólo se había trabajado con bases de datos relacionales y se quería tener un otro punto de vista y una experiencia diferente.

A continuación, se van a explicar más detalladamente los servicios empleados de *Firebase* en la aplicación:

1.*Firestore Database*

*Firestore* (*Cloud Firestore, s.f.)* se trata de una base de datos NoSQL flexible, escalable y que está en la nube para poder almacenar y sincronizar datos tanto en la parte cliente como en la parte del servidor. Es por lo que **se ha escogido *Firestore* como sistema de base de datos común para la versión móvil y la versión web de la aplicación**.

Los datos se almacenan dentro de documentos; dichos documentos a su vez se organizan en distintas colecciones, y el contenido de estos documentos puede ser desde campos simples como pueda ser un número o una cadena de texto, hasta objetos anidados complejos.

La Figura 14 muestra en la columna de la izquierda las distintas listas de la aplicación Aedo, las cuales coinciden con el modelo de datos de la Figura 13; en la columna central aparecen los distintos documentos pertenecientes a la colección, y en la columna derecha los datos asociados a dicho documento.

Además, se pueden realizar consultas para recuperar documentos individuales dentro de colecciones específicas como recuperar todos los elementos de una colección que coinciden con los parámetros de la consulta.

[2] https://firebase.google.com/
[3] https://firebase.google.com/pricing

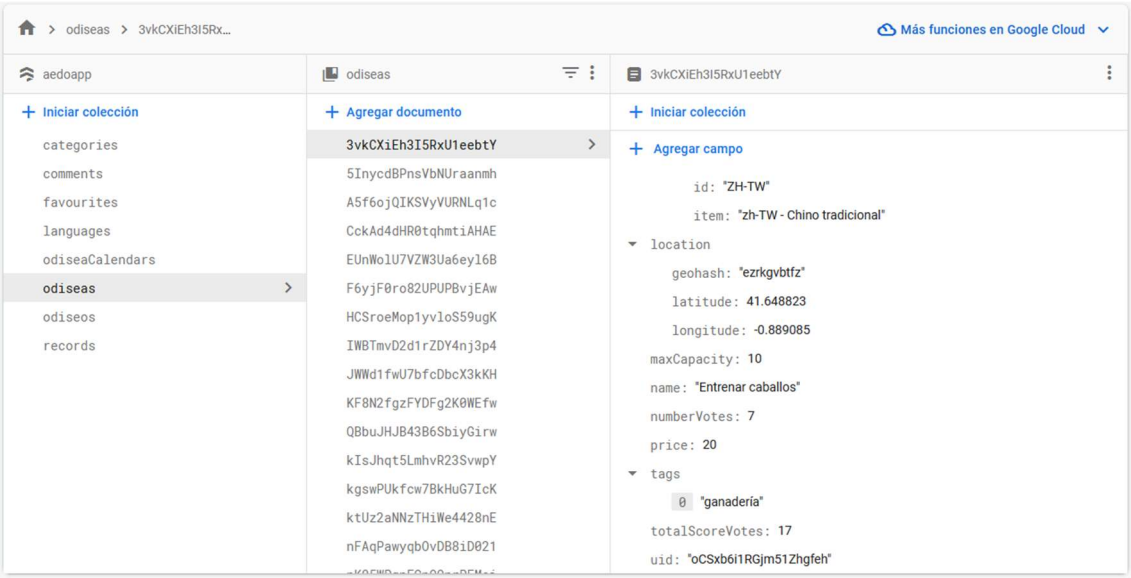



***Figura 14**: Colecciones y documentos de Aedo (Fuente: Elaboración propia)*

Los datos se actualizan en tiempo real por lo que cualquier dispositivo conectado a la base de datos dispondrá de los datos actualizados en todo momento.

Tiene la característica de la asistencia sin conexión, con la cual *Firestore* almacena en memoria caché los datos. Esto permite que, si un dispositivo se queda sin conexión, en el momento en que la recupere, escribirá los datos que tiene almacenados en su caché dentro de la base de datos.

2.*Authentication*

La segunda funcionalidad empleada del *Firebase es el* autentificador. Aedo necesita identificar a los usuarios para ofrecerles una experiencia óptima y poder brindar todas las funcionalidades; es por ello que se ha optado por usar el *Authentication* (*Firebase Authentication, s.f.)* ya que entrega servicios para el *backend* así como bibliotecas para la interfaz de usuario ya elaboradas.

Esta funcionalidad admite distintos tipos de autentificación: mediante correo, números de teléfono, proveedores federados (como Google, Facebook o Twitter), etc. El punto interesante de utilizar el servicio es que el *Authentication* gestiona los usuarios por su cuenta, el sistema guarda las contraseñas, por lo que los desarrolladores no se tienen que preocupar por la manera en la que se guardan las contraseñas o el encriptado de las mismas.

Mediante el uso de las *APIs* del autenticador, los desarrolladores pueden implementar un cambio de contraseña, un cambio de correo, tramitar una baja de un usuario, todo de manera transparente. El servicio del *Authentication* se encarga automáticamente de enviar correos electrónicos a los usuarios para realizar las gestiones como una recuperación de contraseña o una baja de la cuenta.

3.*Storage*

La tercera funcionalidad es el *Storage* (*Cloud Storage para Firebase, s.f.)* o almacenamiento. Esta funcionalidad se emplea para almacenar y entregar contenido generado por usuarios, como puedan ser fotos, audio, videos o cualquier tipo de contenido. *Storage* agrega seguridad a las operaciones de carga y descarga de ficheros sin que importe la calidad de la conexión; si se interrumpe una carga o una descarga, después la aplicación continúa desde el punto en el que estaba para ahorrar ancho de banda. En el *storage* se guardan las imágenes de las categorías, las imágenes de perfil de cada usuario o las fotos de les experiencias cuando se crean desde el gestor de experiencias.

4.*Hosting*

La última característica empleada es el *Hosting* (*Firebase Hosting, s.f.)* que permite alojar la aplicación web en un servidor el cual acepta contenido estático y dinámico. La página web se entrega al usuario de manera segura mediante SSL (*Secure Socket Layer)*. Es necesario el servicio para la aplicación porque se va a ofrecer una versión web de esta, desarrollada en Angular.

### 4.2.2 Lenguaje de desarrollo

La aplicación está escrita en dos lenguajes de programación distintos: JavaScript y TypeScript. Desde un primer momento se tuvo claro que la aplicación iba a ser nativa dadas sus características, por tanto, hubo que escoger un *framework* compatible con dispositivos móviles, a ser posible uno que fuera concordante con Android y iOS. Dado que en el grupo de desarrolladores ninguno tenía problema en estudiar y aprender nuevas tecnologías, se escogió React Native[4] como *framework frontend* (*framework* específico para la interfaz gráfica de la aplicación). Este marco de trabajo es capaz de trabajar con diferentes lenguajes de programación como Objective-C, Swift o Java, pero también JavaScript, y dado que el objetivo era aprender nuevas tecnologías, se optó por JavaScript. El lenguaje es interpretado, por tanto, no se compila y es orientado a objetos, característica afín al proyecto ya que se van a generar objetos con atributos; es débilmente tipado, es decir, a la hora de declarar una variable no se indica el tipo de dato, esta característica no es deseable. Además, no tiene atributos de visibilidad, por tanto, los *getters* o *setters* no son necesarios.

Por otra parte, la aplicación dispone de una versión web escrita en Angular. Se estuvieron valorando los tres *frameworks* más potentes para el *frontend* web a ser Angular[5], React[6] y Vue[7] y como se quería aprender lo máximo posible, se descartó React por su similitud a React Native y se adoptó Angular de Google porque se desarrolla utilizando TypeScript. Este lenguaje funciona en los mismos entornos en los que se puede emplear JavaScript pero tiene la particularidad de que

[4] https://reactnative.dev/
[5] https://angular.io/
[6] https://es.react.dev/
[7] https://vuejs.org/

las variables son *tipadas*, aquí sí que cada variable tiene un tipo de dato preestablecido y admite atributos de visibilidad.

### 4.2.3 Entornos de desarrollo

Una vez se escogió el lenguaje de programación con el cual se iba a desarrollar la aplicación, hubo que designar un IDE, del inglés *Integrated Development Environment*. El mercado actual ofrece numerosos entornos de desarrollo para JavaScript como puedan ser: Visual Studio Code, Átomo, Webstrom, IntelliJ IDEA, Sublime Text, ActiveState Komodo… Cada uno con unas características, unos *pros* y unos *contras* (*Mahendra, s.f.)* en su uso.

La elección de un entorno de desarrollo es una decisión más personal. Por las características que ofrecen, se valoró entre el Visual Studio Code (VSC) y el IntelliJ IDEA, ambos muy completos y potentes. Al final se optó por el Visual Studio Code por todas las extensiones que ofrece para ayudar al desarrollo, así como los *snippets,* que son una especie de atajos de teclado mediante los cuales, escribiendo, por ejemplo, solo cinco caracteres y una combinación de teclas, genera un componente entero de React Native.

Otra funcionalidad del VSC que se empleó mucho, sobre todo al inicio del proyecto en el que se tenían escasos conocimientos sobre las tecnologías, fue la opción de *LiveShare,* mediante la cual un usuario compartía su código de sesión de programación con un enlace y los demás podían unirse a programar colaborativamente. Esta funcionalidad fue especialmente útil durante esa etapa de *pair-programming* (programación colaborativa) en la que todos los miembros del equipo estaban de aprendizaje, tanto para React Native como para Angular. También se empleó el *LiveShare* en distintas sesiones de depuración en las que se intentaban resolver unos fallos a la hora de transmitir datos entre los distintos componentes de React Native.

De entre todas las extensiones que se estuvieron empleando cabría destacar el uso de dos extensiones más. Por una parte, el “To-Do Tree” con la cual se marcaban partes de código por hacer, pero lo más importante es que se marcaban puntos de interés en el código donde podían existir errores para su posterior solución. Por otra parte, “Android iOS Emulator” como la joya de la corona de las extensiones para el proyecto de Aedo. Se trata de una extensión que conecta con el emulador móvil de Android o iOS que ofrece el “Virtual Device Manager” de Android Studio (otro entorno de desarrollo para aplicaciones Android). Al final, desde un único punto como es el Visual Studio Code, se disponía de todas las herramientas para poder desarrollar de la manera más eficiente posible, incluso disponía de la consola de Git para el trabajo colaborativo con control de versiones.

### 4.2.4 Herramientas colaborativas de programación

Programar es una tarea que se realiza en equipo. Puede ser un equipo de mayor o menor tamaño; en el caso de Aedo, es un equipo de cuatro programadores *full stack*. Los profesionales deben utilizar herramientas para el trabajo colaborativo. El equipo empleó Git como sistema de control de versiones. Git es de los sistemas más utilizados actualmente por todo el mundo y se trata de un sistema de control de versiones distribuido, es decir, no está ni en un único ordenador,

ni en un único servidor, sino que el código está ubicado en distintos servidores en la nube accesibles a través de cualquier cliente que tenga la aplicación de gitBash instalada (y por supuesto, disponga de credenciales). De entre las distintas opciones entre las cuales se puede escoger para almacenar el código en la nube como repositorio, se escogió GitHub, ya que todos los miembros disponían de cuenta y se había estado utilizando en otros proyectos durante la carrera. Sin embargo, se analizaron también otros servicios como GitLab[8] o Codeberg[9]; todos estos servicios tienen un funcionamiento similar, desde línea de comandos o desde la interfaz gráfica del Visual Studio Code, ya que Git es un estándar y los comandos son comunes independientemente de la plataforma que se utilice, simplemente cambia en qué repositorio se está trabajando.

Además, el proyecto se ha servido de una metodología de trabajo ágil y para ello se empleó la herramienta de *Worki Tune Up*[10] durante el transcurso de la asignatura de PIN. Esta herramienta captura características de distintas metodologías ágiles como puedan ser *Scrum (Drumond, s.f.), ExtremeProgramming (Bello, 2021)* o *Kanban* (*Radigan, s.f.*), y las combina todas en una única página web para que los desarrolladores puedan trabajar de manera colaborativa cumpliendo a su vez con las necesidades de la asignatura. *Worki* ofrece un tablero *Kanban* con un flujo de trabajo personalizado; algunas de las fases son la especificación de requisitos, la programación y las pruebas de aceptación. Una unidad de trabajo (UT) es una funcionalidad que se debe implementar en una aplicación, dicha UT pasará por las distintas fases del tablero *Kanban*, desde que son registradas hasta que se realizan las pruebas de aceptación y se da por concluida. Esas unidades de trabajo están priorizadas y deben ser abordadas en dicho orden para evitar problemas de dependencias o satisfacer las necesidades del *Product Owner*. Al tratarse de un equipo *full stack,* cualquier miembro puede realizar las distintas unidades. La Figura 15 muestra un listado con las unidades de trabajo priorizadas. Una característica de *Scrum* es el trabajo con *sprints,* que son unos ciclos de tiempo, normalmente tres o cuatro semanas, en las cuales se desarrollan unas funcionalidades para la aplicación según la capacidad del equipo y de esta manera el *software* se elabora de una manera incremental en cada uno de estos *sprints*.

Introducidas estas dos herramientas (Git y Worki), la metodología de trabajo fue crear las unidades de trabajo por cada funcionalidad de la aplicación, a partir de los requisitos que se decidieron a inicios del proyecto durante la especificación de requisitos. Por cada UT se creó una rama de Git a partir de la rama principal en la cual se implementaba la funcionalidad de la unidad de trabajo. Una vez la UT llegaba al final del ciclo *Kanban* y pasaba las pruebas de aceptación, se realizaba un *merge* (unir) de la rama secundaria con la funcionalidad ya implementada sobre la rama principal para que esta funcionalidad estuviera disponible.

[8] https://about.gitlab.com/
[9] https://codeberg.org/
[10] https://cliente.tuneupprocess.com/web/#/login

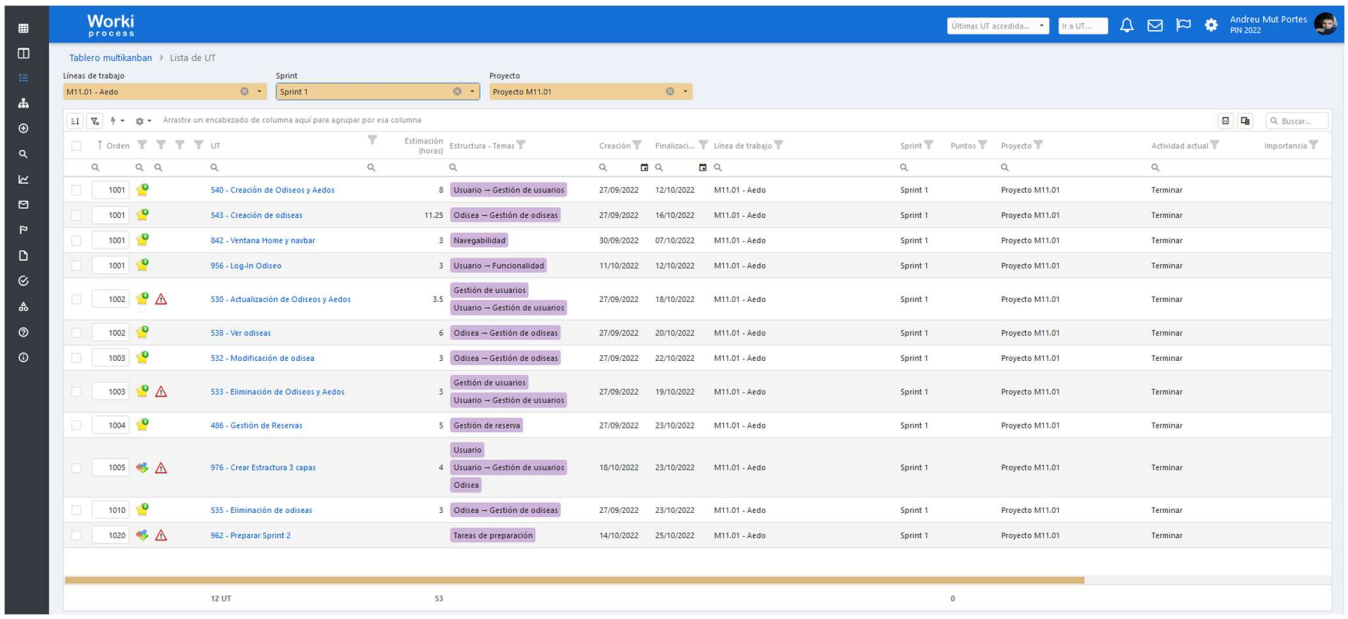



***Figura 15**: Lista de unidades de trabajo en Worki (Fuente: Elaboración propia)*

Tras la asignatura de Proyecto de Ingeniería del Software (PIN), el equipo de desarrollo apostó por otra herramienta que se utiliza en el mundo empresarial similar al *Worki*. Se trata de la herramienta de *Jira software*[11] la cual también dispone del listado de unidades de trabajo (ver Figura 16) en Jira, llamadas "historias de usuario" por la terminología de *Scrum,* aunque en esencia son lo mismo; también dispone de un tablero *Kanban* por el cual van pasando las historias de usuario en sus distintas fases. Una funcionalidad de Jira que resultó interesante es la integración con Microsoft Teams, pudiendo generar mensajes automáticamente en canales autorizados con información sobre las historias de usuario de Jira. Además, Jira permite crear ramas automáticamente en el repositorio de Git cuando se generan las historias.

Tras este tiempo trabajando con ramas de Git, y diversos sucesos y problemáticas con las mismas, se obtuvo una valiosa experiencia a su vez que una necesidad, y es la de haber trabajado con una rama de "producción" (que podría ser perfectamente la rama principal) y una rama secundaria de "preproducción" (creada a partir de la rama de producción) desde la cual nacieran las distintas ramas para elaborar las funcionalidades y realizar la unión (*merge)* de las funcionalidades sobre esa rama de preproducción. De esa manera, en todo momento se dispondría de una rama cien por cien funcional y estable para el cliente (la rama de producción). Se tuvieron problemas al realizar unas implementaciones días antes de una presentación de producto y podría haber acabado en una catástrofe si no se llega a solucionar; siempre se podría haber restaurado una versión anterior de Git como última instancia.

[11] https://www.atlassian.com/es/software/jira

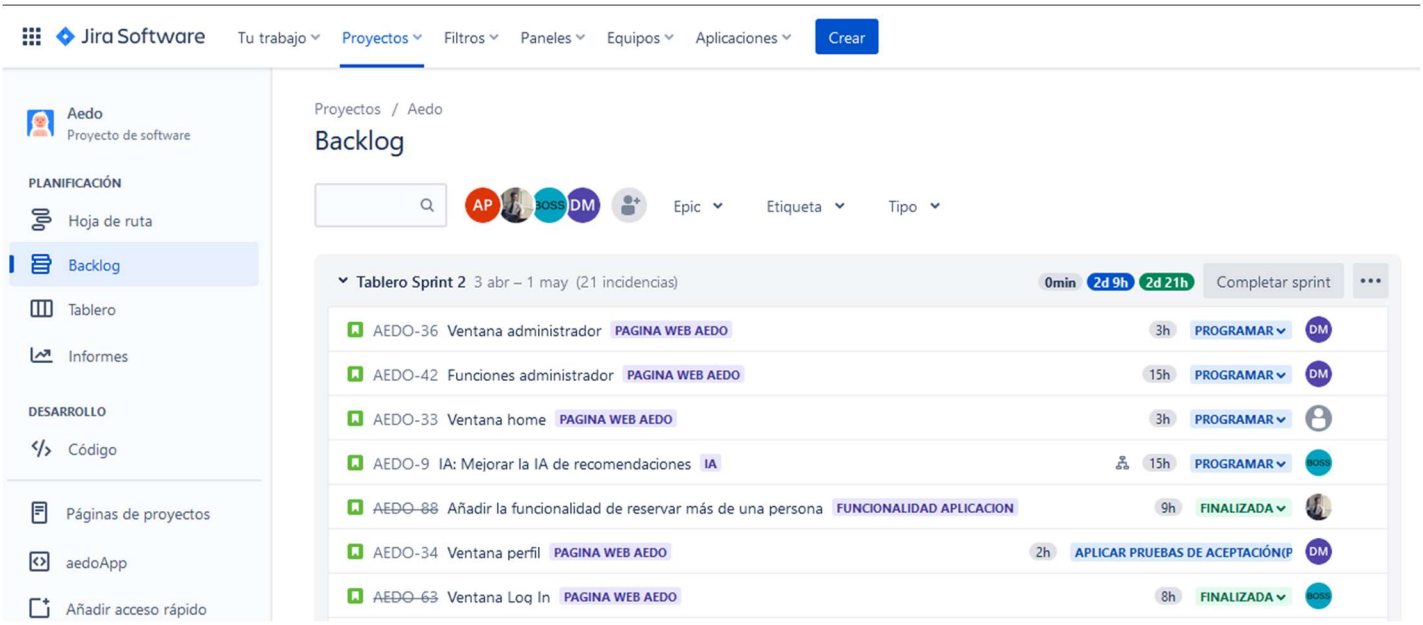



***Figura 16**: Lista de unidades de trabajo en Jira (Fuente: Elaboración propia)*

### 4.2.5 Node.js y Npm

Al haber escogido JavaScript como lenguaje de programación, se requiere un entorno de ejecución para dicho lenguaje y ahí es donde entra en juego Node.js (*Lucas, 2019*). Así pues, Node.js es un entorno de ejecución para ejecutar un programa que esté escrito en JavaScript. Aunque JavaScript se utilizaba inicialmente en entornos web, Node.js permite ir más allá de los sitios web y le otorga la capacidad a este lenguaje de hacer cosas igual que hacen otros lenguajes de secuencia como Python. JavaScript se ejecuta en el motor en tiempo de ejecución, este motor lo que hace es convertir el código JavaScript a código máquina, sin tener que interpretarlo primero, ignorando la compilación y por tanto aumentando la velocidad. Node.js utiliza un modelo de entrada y salida sin bloqueo donde las entradas son solicitudes y las salidas son respuestas por parte del servidor. La finalidad de Node.js no es la de realizar operaciones con mucha carga de procesador, y si se hace, de hecho, elimina casi todas las ventajas del entorno, ya que donde Node realmente destaca es en la creación de aplicaciones de red rápidas, lo que se acaba traduciendo en una alta escalabilidad.

Explicado el entorno, se debe hablar ahora sobre el sistema de paquetería que utiliza éste mismo, conocido como Npm (*Hernández, 2021*) que responde a *Node package manager*. Este gestor es la herramienta por defecto usada en JavaScript para instalar y compartir paquetes. Npm se compone de dos partes principales: por una parte, un repositorio de paquetes de *software* libre en línea, los cuales se pueden utilizar en proyectos Node.js, por otra parte, ofrece una herramienta para la terminal que permite interactuar con el repositorio el cual ayuda a instalar las utilidades, manejar las dependencias y publicar paquetes. Las dependencias se guardan en el fichero package.json del proyecto, el cual recoge todos los paquetes instalados en el proyecto mediante la herramienta Npm así como la versión de cada uno de estos paquetes instalados.

Gracias a Npm, se han podido desarrollar rápidamente numerosas funcionalidades dentro de la aplicación por la preexistencia de paquetes realizados por otros programadores con dicha funcionalidad. Especialmente ha sido útil en el apartado del *frontend* con React Native, ya que había paquetes con componentes ya creados que permitían trabajar fácilmente en los formularios de la aplicación y otras funcionalidades como el mapa en el cual presentar las experiencias, los calendarios para la selección de fechas, los comentarios con sus puntuaciones y una larga lista de funcionalidades más.

### 4.2.6 Expo

Expo[12] es un *framework* cuyo objetivo es el de facilitar el desarrollo de aplicaciones React Native. Gracias a Expo, se pueden crear aplicaciones nativas sin tener que lidiar con los problemas derivados del uso de Android Studio.

La herramienta de Expo facilitó la creación de la *apk* y el poder compartirlas entre los desarrolladores, ya que Expo genera la aplicación *apk* en línea y la deja disponible para los usuarios, los cuales deben disponer de permisos para poder descargarla. Además, permite generar los archivos *aa*b (App Bundle) que son requeridos por las tiendas como Google Play.

La principal diferencia entre un fichero *apk* y un fichero *aab* es que el *apk* es el ejecutable de una instalación que contiene todos los recursos de la aplicación, como puedan ser los idiomas o los textos, mientras que una *aab*, la cual se publica en la tienda de Google, no contiene todos los recursos, solo instala lo necesario y a consecuencia de esto ocupará menos espacio en los dispositivos.

### 4.2.7 ChatGPT

La inteligencia artificial de ChatGPT fue utilizada también en el proyecto para prestar asistencia en la generación de contenido para la base de datos. A la hora de realizar las demostraciones en los experimentos uno y dos, anteriormente mencionados en este documento en los apartados “3.1.1 Experimento 1” y “3.2.1 Experimento 2”, se necesitaba disponer de una base de datos lo suficientemente poblada como para que la aplicación pareciera que estaba siendo usada en un entorno real; es para ello que se creó un *script* de generación de datos. Este *script* generaba usuarios, experiencias con nombre, descripción, idioma, categoría, incluso una ubicación.

La aplicación, actualmente, permite crear experiencias que se pueden englobar en doce categorías distintas, como por ejemplo acuáticas, artesanía, culturales, gastronomía o historia (entre otras). La inteligencia artificial facilitó el obtener nombres y descripciones para dichas experiencias de cada una de las categorías. Con una consulta como la siguiente: “Dame un diccionario de JavaScript cuya clave sean actividades de cocina y el valor una descripción de la actividad”, la inteligencia devolvía dichos diccionarios. La Figura 17 muestra un ejemplo de un

[12] https://expo.dev/

diccionario con las actividades de gastronomía, aunque en la figura se pueden apreciar los otros diccionarios para el resto de las categorías de la aplicación.

Además, también se empleó ChatGPT para solicitar las coordenadas de latitud y longitud de distintas ciudades y poblaciones de España para poder asignar una ubicación a las experiencias, para generar usuarios a partir de los nombres y apellidos más comunes del país, y para realizar comentarios de contenido genérico asociado a una puntuación. La información siempre se recogía en la misma estructura de datos, el diccionario, para poder trabajar con ella de una manera eficiente.

```
export const actividadesCultura = {...};
export const actividadesGanaderia = {...};
export const actividadesHistoria = {...};
export const actividadesArtesanales = {...};
export const actividadesAcuaticas = {...};
export const actividadesAgricultura = {...};
export const actividadesAdrenalina = {...};
export const actividadesArtisticas = {...};
export const actividadesNaturaleza = {...};
export const actividadesDeportivas = {...};
export const actividadesGastronomia = {
  "Cocinar al horno": "Cocción de alimentos con calor seco en un
horno.",
  "Freír patatas":
    "Técnica culinaria que consiste en freír patatas en aceite
caliente.",
  "Hacer sushi":
    "Preparación de rollos de arroz, pescado y otros ingredientes.",
  "Asar carne":
    "Técnica de cocción en la que se cocina carne a altas
temperaturas en una parrilla o sartén.",
  "Preparar ensaladas":
    "Mezcla de vegetales, frutas y otros ingredientes para crear una
ensalada.",
      ...
};
```

***Figura 17**: Diccionarios de actividades por categoría (Fuente: Elaboración propia)*

### 4.2.8 SonarQube

La calidad del *software* es imprescindible dentro de la industria del desarrollo. Es por lo que emplear herramientas para analizar y evaluar el código se convierte en un elemento fundamental para garantizar que la tarea del desarrollo se efectúa como es debido y se aplican buenas prácticas. Mediante SonarQube[13] se consigue tal fin: poder escribir código más limpio y seguro.

SonarQube (*Sentrio,2021)* es una plataforma de código abierto para la inspección continua de la calidad del código mediante el análisis estático (no en ejecución) de código fuente.

[13] https://www.sonarsource.com/products/sonarqube/

La herramienta proporciona métricas que ayudan a los programadores a mejorar la calidad del código, permitiendo a los equipos el poder hacer seguimiento y detectar errores junto a vulnerabilidades de seguridad y así poder mantener el código limpio. Por eso es una herramienta esencial durante la fase de pruebas (*testing)* y las auditorías de código. La herramienta guía a los equipos en las revisiones de código sobre qué hay que mejorar y qué se debe evitar.

SonarQube está desarrollado en Java, pero admite los lenguajes de programación más populares: Java, C, C++, C#, JavaScript, Python, COBOL… Por ello se ha podido emplear esta plataforma en el proyecto Aedo.

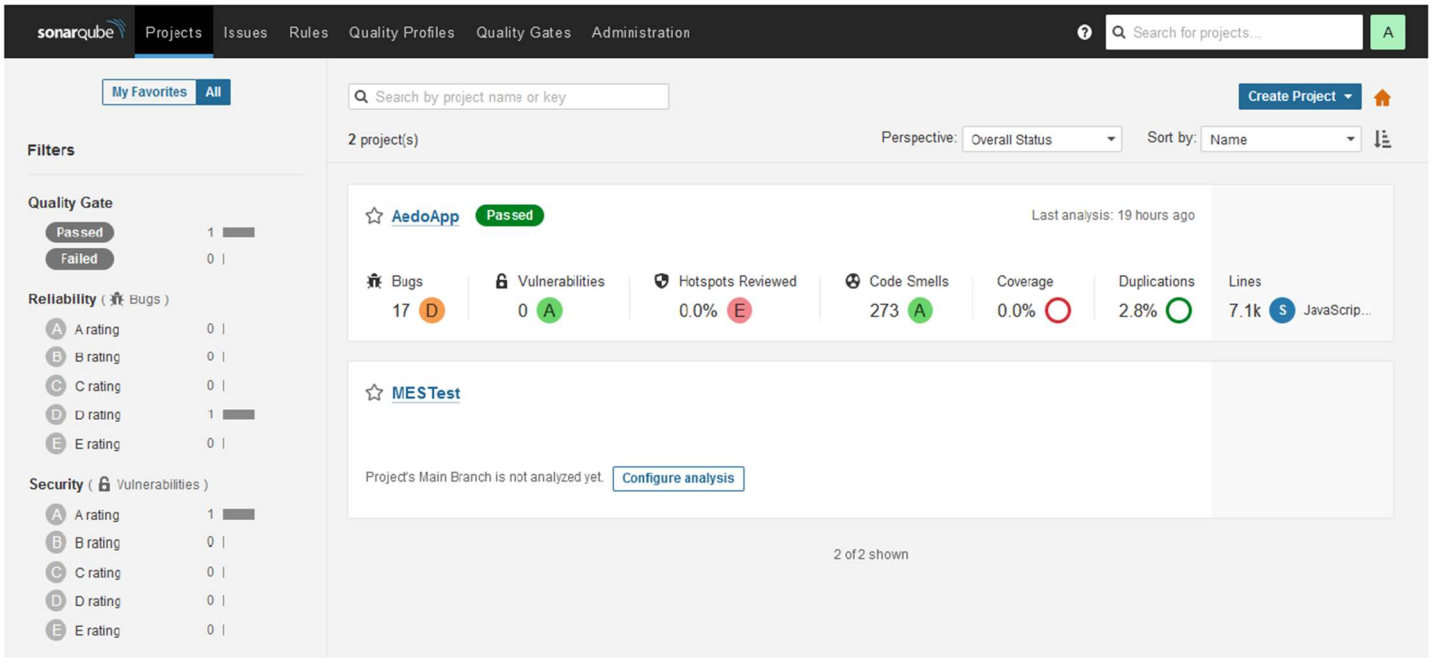



***Figura 18**: Panel de SonarQube en el navegador (Fuente: Elaboración propia)*

El proceso de instalación y configuración es relativamente sencillo mediante la documentación oficial (*SonarQube Documentation, s.f.)*. Básicamente la instalación consiste en descargar una copia local de la herramienta, ejecutarla en consola y acceder mediante el navegador a la dirección de localhost:9000 para poder cambiar la contraseña. Se accede a la opción de crear un proyecto local y se obtiene una *project key* asociada al proyecto.

Una vez instalado, se crea un archivo de propiedades del análisis dentro de la carpeta del proyecto. La configuración de este archivo puede variar según las necesidades, sobre todo son opciones enfocadas en excluir carpetas del proyecto o tipos de fichero concretos. La información que sí debe contener el archivo es una variable con la misma *project key* asociada al proyecto. Después de configurar el fichero, es necesario descargar y ejecutar el escáner de SonarQube. Tras realizar el escaneo del código, este se presenta en el navegador (Figura 18). Toda esta información del procedimiento está recogida en la documentación oficial.

El resultado del análisis presenta información sobre distintos apartados, como puedan ser los *bugs, code smell,* vulnerabilidades y reglas de seguridad. Además, presenta la deuda técnica de la aplicación. Este concepto responde a la cantidad de tiempo natural necesario para reparar los errores de un *software;* cuanto menor sea la deuda técnica mejor. En la Figura 19 se puede

apreciar el análisis más detallado de los *bugs* de la aplicación. A su vez, estos *bugs* se clasifican en distintas categorías, entre ellas bloqueantes, críticos, mayores o menores, por lo que la aplicación ayuda a identificar qué se debe corregir primero. Es por todo esto que la herramienta de SonarQube se utiliza en tareas de mantenimiento de software.

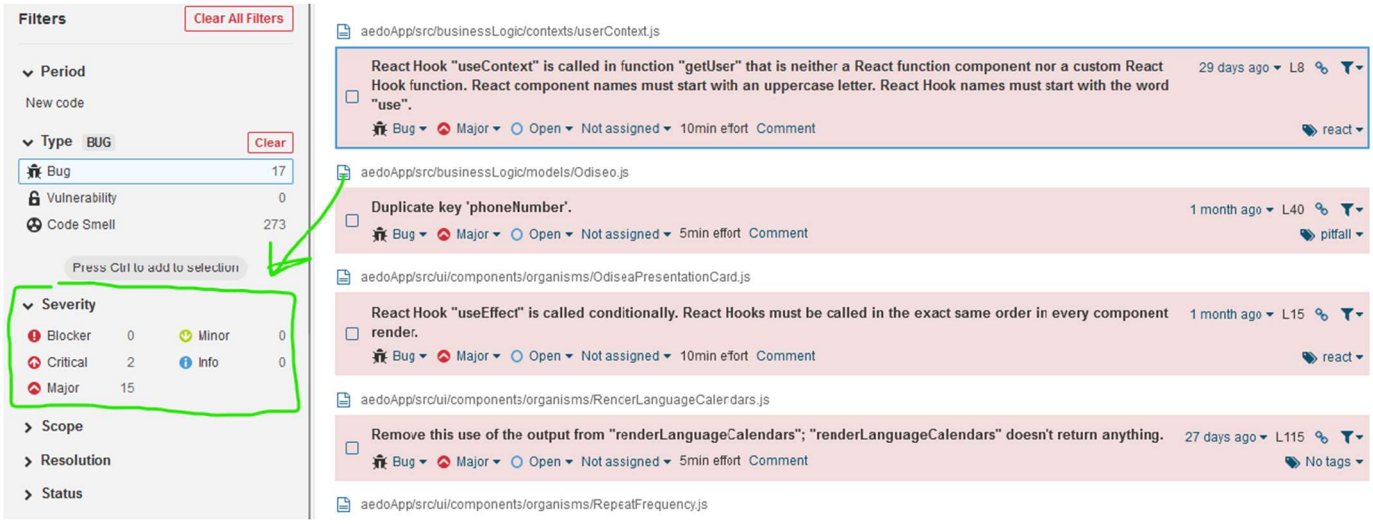



***Figura 19**: Bugs detectados por SonarQube. (Fuente: Elaboración propia)*

### 4.2.9 Jest

La herramienta Jest se ha utilizado para realizar las pruebas en la aplicación. Se hablará más al respecto en el apartado "4.6. Pruebas realizadas.", pero a modo introductorio, es un *framework* para realizar pruebas en el idioma de JavaScript. Permite probar componentes de React así como probar la cobertura de código.

### 4.2.10 JSDoc: clean-jsdoc-theme

Generar documentación es otro punto importante a tener en cuenta si se quieren aplicar las buenas prácticas de programación, ya que cuando un programador realiza su trabajo, no lo está haciendo para sí mismo, caben altas posibilidades que en el futuro otra persona distinta vuelva a revisar el código que se hizo en su momento para poder reutilizarlo en otra parte del programa, o para entender el funcionamiento interno. No siempre se trata de reutilizar funciones, otras veces se tiene que acceder a una aplicación desde el exterior de esta. En el caso de las *APIs* externas resulta crucial tener una correcta documentación que explique cómo utilizar los *endpoints* del programa.

En la aplicación de Aedo, se ha utilizado la librería "Clean-jsdoc-theme" (*Clean-jsdoc-theme, s.f.*) que es una librería específica para el lenguaje de programación JavaScript. De hecho, se sirve en la paquetería Npm de Node.js. Algunas de sus particularidades es que es compatible con muchos tipos de pantallas (ordenador de sobremesa, portátil, *tablet* y dispositivos móviles), dispone de un tema oscuro y un tema claro; cuando se genera la documentación en formato

HTML, minimiza información innecesaria y esto permite que se ahorre espacio en la unidad de almacenamiento. También tiene una función de búsqueda y no desperdicia recursos, por lo que su rendimiento es alto.

*Clean-jsdoc-theme* se ha empleado para generar la documentación de los componentes de React Native y para las *APIs* internas de la aplicación, la sección del documento "4.4.2: APIs de la aplicación móvil" explicará más al respecto sobre estas.

Para generar la documentación se utiliza un sistema de etiquetas antes de cada función que se desee documentar. La estructura que sigue es la mostrada en la Figura 20. Antes de la función se genera un comentario en bloque de JavaScript cuya primera línea es una descripción de qué hace la función, después recibe una etiqueta según el tipo de clasificación de la documentación (para una *API* se utiliza "@external", para los componentes de React Native se emplea "@module") y una ruta que es donde está ubicada la documentación. A continuación, recibe los distintos parámetros (@param) de entrada que tiene la función y lo que retorna dicha función (@return). Opcionalmente, y es recomendable, se puede utilizar la etiqueta @example para mostrar una porción de código de ejemplo de utilización de la función en la documentación.

```
/**
   * Crea Odisea, crea OdiseDates de la Odisea, sube las imagenes al
firebase
   * @external OdiseaApi/create()
   * @param {*} uid identificador del usuario que crea la odisea
   * @param {*} name nombre de la odisea
   * @param {*} description descripcion de la odisea
   * @param {*} images fotos de la odisea
   * @param {*} languages lenguaje/s de la odisea
   * @param {*} data valor de la fecha introducida
   * @param {*} selectedDatePicker tipo de fechas a introducir
   * @param {*} location localizacion de la odisea
   * @param {*} maxCapacity capacidad maxima
   * @param {*} tags lista con las categorias de la odisea
   * @returns una referencia de la odisea
   * @example
   * import { OdiseaApi } from "ROUTE/OdiseaApi";
   * await OdiseaApi.create(
          uid, name, description, images,
          price, languages, data, selectedDatePicker,
          location, maxCapacity, tags
      );
*/
```

***Figura 20**: Fragmento de código para documentar la creación de una odisea en Clean-JSdoc-Theme. (Fuente: Elaboración propia)*

Una vez se han generado los comentarios en bloque antes de cada función a documentar, se debe rellenar el fichero de configuración *jsdoc.conf.json* que tiene las configuraciones de la herramienta. En este archivo se puede escoger qué carpetas incluir en la documentación, donde guardar los artefactos generados (carpeta de salida) y otras opciones que quedan explicadas en la documentación oficial de la herramienta. Tras la configuración del fichero, se ejecuta el comando "npm run docs" y se genera la documentación de la aplicación Aedo en la carpeta de salida especificada. La carpeta de salida se puede ejecutar con el *LiveServer* del Visual Studio Code para

visualizar en el navegador la documentación en local, o se puede exportar a un servidor en internet para que pueda ser accesible por cualquier usuario.

La Figura 21 muestra un ejemplo en el navegador web de la documentación de la *API* de creación de usuarios (Odiseos). En la figura aparece el ejemplo de utilización de código, y en la izquierda se pueden observar las distintas *APIs* en el desplegable *Externals*. Se puede consultar el código fuente de la función y, específicamente en la línea 20 que describe la Figura 21, es donde estaría la información de cada uno de los parámetros de entrada (@params) y la salida (@return) de la función.

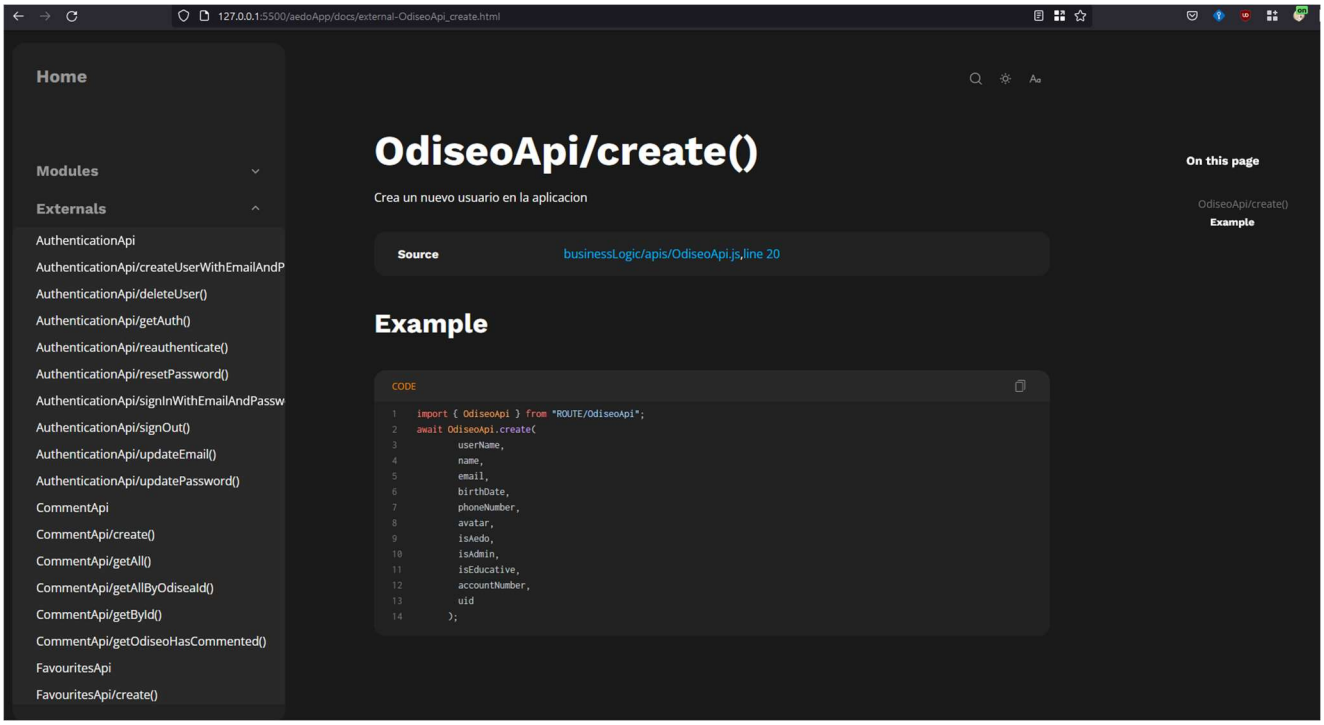



***Figura 21**: Documentación de OdiseoApi/create. (Fuente: Elaboración propia)*

## 4.3 Arquitectura

En esta sección se van a explicar aspectos técnicos sobre la arquitectura de la aplicación. Se distinguen dos apartados: Los modelos de la aplicación y los patrones de programación aplicados.

### 4.3.1 Modelos

JavaScript tiene la particularidad de no ser un lenguaje en el que se asignen los tipos de datos a las variables. Es por esa razón que en una variable declarada “x” se puede guardar cualquier tipo de dato, desde una cadena de texto o un número hasta una lista o un objeto de JavaScript. Para controlar significativamente este problema, se decidió implementar modelos. Un modelo vendría a emular el funcionamiento de una clase de Java, un fichero donde se decide cómo

es un objeto, cómo es el constructor y una serie de métodos asociados a dichos objetos. La Figura 21 muestra los distintos tipos de modelos implementados en la aplicación; existe uno por cada tipo de objeto representable en el modelo de datos (Figura 13).

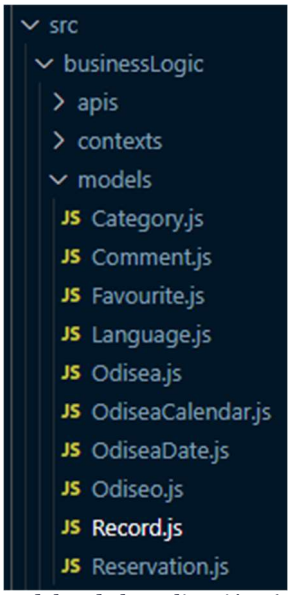



***Figura 22**: Ficheros modelos de la aplicación. (Fuente: Elaboración propia)*

La Figura 23 muestra un fragmento de código del modelo "Odisea.js" en la que se define un constructor de odiseas (experiencias) y tiene funciones como el *toString()* para comprobar los atributos del objeto, un cálculo de la media de las puntuaciones de los comentarios o una lista de OdiseaDate asociada a dicha odisea. En la sección "4.4.2: APIs de la aplicación móvil", concretamente en los ejemplos, se explica el funcionamiento de la OdiseaAPI. En dicho ejemplo se muestra cómo se instancian los objetos del modelo Odisea mediante la palabra reservada "new".

```
import {OdiseaDate} from "./OdiseaDate.js";
const geofire = require("geofire-common");

export class Odisea {
  constructor(uid, name, description, images, price, languages, location,
    maxCapacity, totalScoreVotes = 0, numberVotes = 0, tags = []
  ) {
    this.uid = uid;
    this.name = name;
    this.description = description;
    // Crea un geoHash dada una localización
    const geoHash = geofire.geohashForLocation([
      Number(location.latitude),
      Number(location.longitude),
    ]);
    this.location = { ...location, geoHash: geoHash };
    ...
```

```
  }

  toString() {
    ...
  }

  calculateAverage() {
    ...
  }

  // Crea una coleccion de odiseasDate asociados a la odisea
  createOdiseaDateCollection(odiseaId) {
    ...
  }
}
```

***Figura 23**: Fragmento del modelo Odisea.js. (Fuente: Elaboración propia)*

Nada asegura que una variable declarada como odisea en la que se ha instanciado un objeto Odisea (new Odisea) vaya a ser usada para guardar otro tipo de dato, pero de esta manera se tiene un cierto control y se puede realizar la llamada a funciones a partir del objeto creado.

### 4.3.2 Patrones de diseño.

"Los patrones de diseño o *design patterns*, son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error." (Martínez, 2020, párrafo 2).

En esta sección se ha empleado el libro *Learning Patterns* (Hallie y Osmani, 2023) distribuido bajo licencia *Creative Commons Atribution-NonComercial* como fuente de información para la implantación de algunos de los patrones.

En la aplicación Aedo, se han utilizado tres patrones de diseño diferenciados para poder ayudar a dar la funcionalidad de la aplicación, aumentar la escalabilidad y capacidad de modificación, y fomentar la reutilización del código.

1. *Facade pattern*:

El patrón fachada ha sido el pilar fundamental sobre el que se ha construido la aplicación junto al modelo de tres capas. Ofrece al usuario un punto de acceso a la aplicación y, junto al modelo de tres capas, la aplicación se encarga de dar la funcionalidad. Esto permite que se puedan realizar cambios en una de las capas sin alterar la funcionalidad del resto de ellas. Se explicará más detalladamente en la sección "4.4 Implementación *facade pattern* y el modelo de tres capas".

2. *Singleton pattern*:

El patrón *singleton* es un patrón de diseño creacional que permite tener una única instancia de un objeto disponible en toda la aplicación. En React Native se ha implementado mediante los "contextos"; se explica más detalladamente en la sección "4.5 Implementación del patrón *singleton"*.

3. *Module pattern:*

La finalidad de este patrón es la de dividir el código en partes más pequeñas para que se puedan reutilizar. React Native trabaja con componentes, por tanto, si una funcionalidad se utiliza en distintas pantallas de la aplicación, resulta interesante extraer la funcionalidad como componentes y así llamar al componente en las pantallas que sea necesario y se evita la reescritura de código.

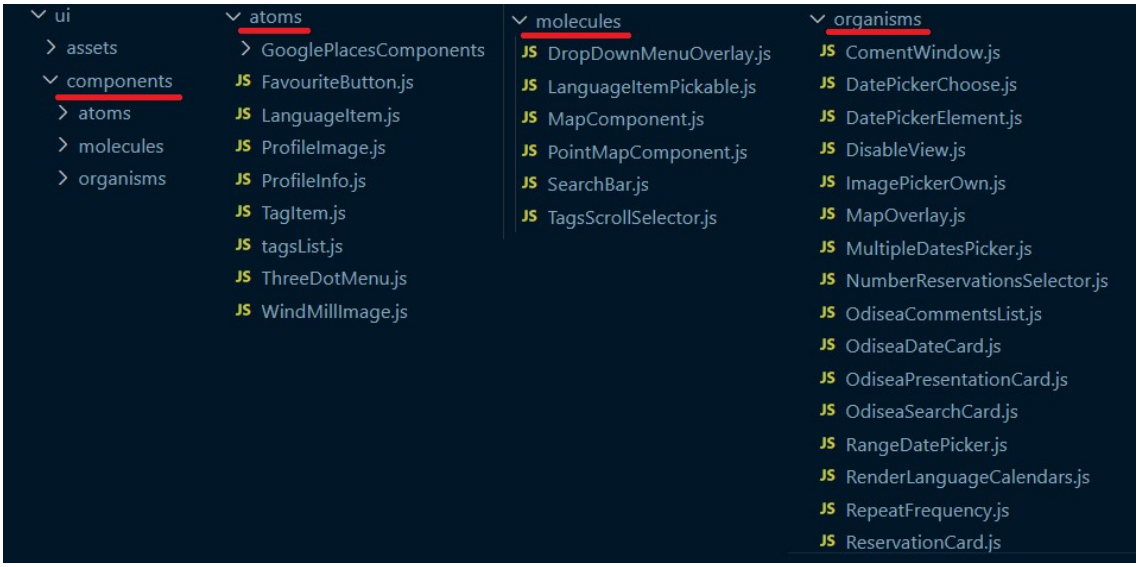



***Figura 24**: Componentes de la aplicación. (Fuente: Elaboración propia)*

En la aplicación se han dividido los componentes como átomos, moléculas y organismos (ver la Figura 24). Asignarlos a un grupo u otro depende del tamaño y de las funcionalidades que ofrece el componente. Si es muy simple, se asignaría a un átomo, mientras que si el componente utiliza otros componentes y además ofrece funcionalidades, podría asignarse a un organismo. Con esta separación hecha, siempre que haga falta un componente en una ventana dada, se puede llamar y renderizar sin tener que repetir líneas de código.

## 4.4 Implementación *facade pattern* y el modelo tres capas

El patrón fachada marca un punto de acceso en la aplicación y trabaja en conjunto con el modelo de tres capas. Un usuario, cuando utiliza la aplicación, navega por unas interfaces de pantalla, y llegado cierto punto, puede querer interaccionar para introducir u obtener información

de la aplicación. Para ello, puede utilizar los campos de los formularios o gestos sobre la pantalla. La pantalla es la capa de interfaz de usuario y esa capa se tiene que conectar con las otras dos capas restantes: la de lógica de negocio y la de persistencia. La interfaz puede realizar una conexión directa con la capa de persistencia, pero esto no es correcto y puede dar problemas futuros. Lo que se debe buscar en una aplicación *software* es aumentar lo mayor posible el desacoplamiento para poder facilitar el mantenimiento, la reutilización y las pruebas del código. El patrón fachada es una capa situada entre la interfaz de usuario y la capa de la lógica de negocio, es el punto de entrada de la aplicación a través de la interfaz de usuario. Dicho acceso se ofrece mediante las *APIs* y se obtiene una aplicación con las tres capas separadas, ya que el *front* pasa la información a las *APIs,* las *APIs* realizan la lógica de negocio y se conectan con el fichero de configuraciones del *Firebase,* y este fichero ya realiza la conexión con la base de datos (Figura 25).

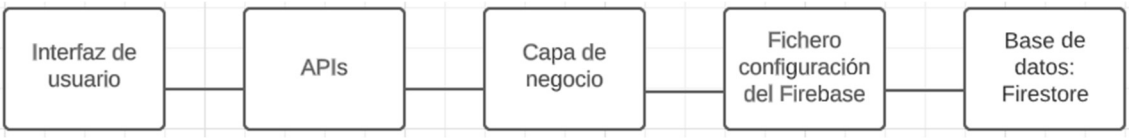



***Figura 25:*** *Esquema modelo de tres capas. (Fuente: Elaboración propia)*

Este modelo permite que, si en algún momento se tiene que realizar un cambio en la capa de negocio, solo se modifique la *API* afectada que interacciona con la capa de negocio modificada. Esto facilita que si hay distintas interfaces de usuario que apunten a la misma funcionalidad de la *API*, realizando el cambio en un único sitio (la *API)* se corrija la funcionalidad en todos los demás sitios (las distintas interfaces de usuario). De igual manera, con el sistema de base de datos, si en el futuro se decidiera cambiar la base de datos del *Firestore* a otra tecnología, simplemente habría que modificar la capa intermedia entre la capa de negocio y la capa de persistencia de la base de datos y la capa de negocio seguiría funcionando igual.

### 4.4.1 Fichero *Firebase.*

Se va a explicar a continuación un fichero específico del proyecto que es el fichero de configuraciones del *Firebase*. Es importante mencionar las distintas partes porque en él se incluye la conexión al servidor externo de base de datos y las distintas funciones para que la aplicación pueda ofrecer el servicio.

#### *4.4.1.1 Conexión al Firebase*

Para poder empezar a utilizar el servicio es necesario crear un proyecto *Firebase* siguiendo la documentación (Primeros pasos con Cloud Firestore, s.f.). Una vez creado, desde la consola en la página web se puede acceder a la "Configuración del proyecto" en la que en la pestaña General tiene la clave *Api* (Figura 26) que se emplea en el fichero de configuración (Figura 27).

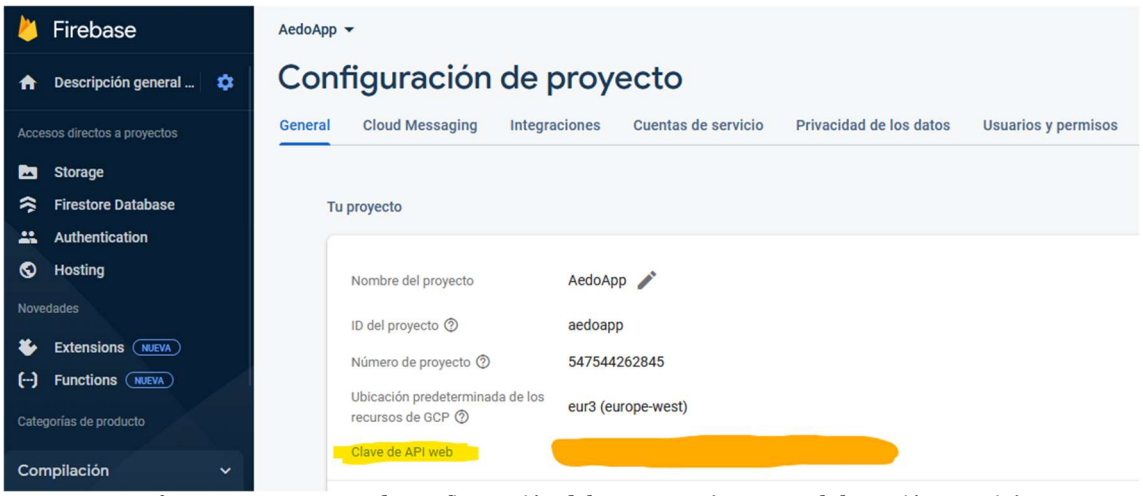



***Figura 26:*** *Captura de configuración del proyecto. (Fuente: Elaboración propia)*

La figura 27 presenta el fragmento de código que toda aplicación debe contener para poder implementar el servicio. Como campo destacado mencionar la línea "apiKey" que es donde se coloca la llave API que proporciona el *Firebase* al generar el proyecto según se ha explicado. Además, se han realizado los *imports* de los servicios que se van a emplear: el *authentication* y el *storage*.

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import {
  addDoc, collection, deleteDoc, doc, endAt, getDoc, getDocs, getFirestore,
limit,
  orderBy, query, setDoc, startAfter, startAt, where,
} from "firebase/firestore";
import { getDownloadURL, getStorage, ref, uploadBytes } from
"firebase/storage";
const geofire = require("geofire-common");

const firebaseConfig = {
  //Aedo App firebase:
  apiKey: //Your API key here
  authDomain: "aedoapp.firebaseapp.com",
  projectId: "aedoapp",
  storageBucket: "aedoapp.appspot.com",
  messagingSenderId: "547544262845",
  appId: "1:547544262845:web:18bcc1f6f3522be9c7afd4",
  measurementId: "G-PJGG5SR7T5",
};
// Initialize Firebase
const app = initializeApp(firebaseConfig);
const db = getFirestore();
export const auth = getAuth(app);
```

***Figura 27****: Fragmento de código del fichero firebase.js (Fuente: Elaboración propia)*

#### *4.4.1.2 Operaciones CRUD*

Una vez se ha realizado la conexión, hay que trabajar con el servicio. Una operación *CRUD* proviene de la sigla inglesa *create, read, update and delete*, son las distintas operaciones que se pueden realizar en una base de datos. Al estar utilizando *Firestore*, la base de datos se organiza en colecciones y documentos asociados a dichas colecciones. Un fragmento de código para crear un documento en la colección "odiseas" podría ser el de la Figura 28.

```
let item = {name:"Pescar", description:"Aprender a pescar"...}


export const createExperiencia = async (item) => {
  return await addDoc(collection(db, "odiseas"), Object.assign({}, item));
};
```

***Figura 28**: Creación de una experiencia sin genericidad. (Fuente: Elaboración propia)*

De esta manera se crean los documentos en las distintas colecciones utilizando la *API* del *Firestore*. Las operaciones restantes del *CRUD*, es decir *read, update* y *delete* son similares en sintaxis a la Figura 28. La función recibe un objeto como parámetro y después se realiza la acción.

Aquí se presenta un problema, y es que, por cada colección o clase de objeto que se quiera tratar en la aplicación, se va a requerir de cuatro funciones adicionales, una para cada acción del *CRUD*. Esto resulta en una explosión de funciones en el fichero del *Firebase*.

Para evitar cargar con tal cantidad de funciones el código se buscó una solución alternativa y al final se optó por emplear la genericidad. En lugar de utilizar una función por cada colección, se han utilizado cuatro funciones genéricas, tal y como se muestra en la Figura 29.

```
export const create = async (coll, item) => {
  return await addDoc(collection(db, coll), Object.assign({}, item));
};

export const getById = async (coll, id) => {
  const docRef = doc(db, coll, id);
  const docSnap = await getDoc(docRef);

  return { ...docSnap.data(), id: docSnap.id };
};

export const update = async (coll, item, id) => {
  return await setDoc(doc(db, coll, id), Object.assign({}, item));
};

export const remove = async (coll, id) => {
  return await deleteDoc(doc(db, coll, id));
};
```

***Figura 29:** Funciones genéricas para las operaciones CRUD. (Fuente: Elaboración propia)*

Estas cuatro funciones genéricas reciben un parámetro “coll’ que es la colección con la que se quiere trabajar con la base de datos y otros parámetros según necesidad de la operación. De esta manera se obtiene un fichero más limpio y sin tantas funciones y se deja la funcionalidad de escoger a qué colección llamar y qué información pasar como parámetros a las diferentes *APIs* internas de la aplicación.

El fichero anterior por supuesto dispone de otras funciones necesarias para el correcto funcionamiento de Aedo, así como una función, también genérica, que recibe por parámetros la información necesaria para hacer las distintas consultas en la base de datos y obtener información de acuerdo a unos parámetros que se emplean en los campos *where*.

### 4.4.2 *APIs* de la aplicación móvil

Este apartado tratará de explicar las distintas *APIs* de la aplicación. Se ha generado un fichero por cada tipo de objeto definido en el modelo de datos de Aedo y un par más que han surgido por necesidades específicas de la aplicación. La Figura 30 muestra esos ficheros.

En general, todos los archivos tienen un funcionamiento similar: recibir información de la interfaz gráfica mediante el paso de parámetros, generar un objeto de la lógica de negocio perteneciente a dicha entidad y llamar a la función pertinente del archivo del *Firebase* para que traslade la petición a la base de datos. No obstante, aunque tengan un comportamiento similar en general, en algunos de los ficheros su contenido puede resultar interesante. Procedemos a explicarlo a continuación.

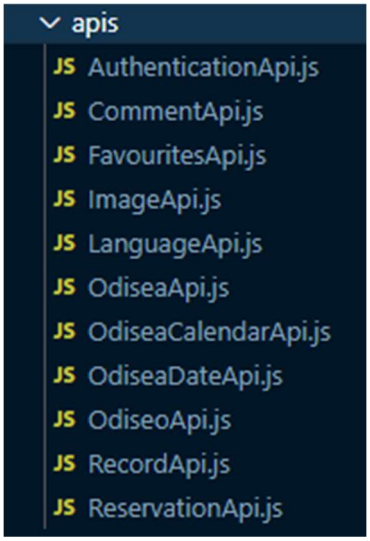



***Figura 30:*** *APIs de la aplicación. (Fuente: Elaboración propia)*

1.AuthenticationApi.js

Este fichero implementa toda la funcionalidad relativa a la información de inicio de sesión de los usuarios: creación de usuario mediante correo electrónico y contraseña, inicio de sesión

con el correo electrónico y la contraseña, restaurar contraseña, actualizar correo electrónico o la contraseña, cerrar sesión o borrar el usuario.

Para ofrecer tal funcionalidad se sirve de la librería *firebase/auth* del *Firebase Authentication* (Firebase Authentication, s.f.). La Figura 31 muestra un fragmento de código para la generación de un usuario mediante correo electrónico utilizando el servicio del *Authentication* de *Firebase.* Se puede apreciar que, tras crear el usuario en el servicio, se envía un *email* de confirmación al cual no se le permite realizar un inicio de sesión hasta que no es aceptado.

Estas funcionalidades proporcionadas por la librería han agilizado las tareas de programación, ya que *Authentication* se encarga de gestionar el inicio de sesión, cambios de contraseña, validaciones, incluso de almacenar la contraseña del usuario.

```
import {
  createUserWithEmailAndPassword,
  sendEmailVerification,
  ...
} from "firebase/auth";
import { auth, remove } from "../../database/firebase";

export const AuthenticationApi = {
  ...
  createUserWithEmailAndPassword: async function (auth, email, password) {
    //Crear el usuario en el auth de google
    await createUserWithEmailAndPassword(auth, email, password).then(
      async () => {
        //Enviarle el email de confirmacion
        await sendEmailVerification(auth.currentUser);
      }
    );
  }
  ...
}
```

***Figura 31**: Fragmento AuthenticationApi. (Fuente: Elaboración propia)*

2.OdiseoApi.js

Esta *API* ofrece las siguientes funciones: getAll(), que devuelve todos los usuarios; create(*params*…), para crear un usuario; getByUserName(*userName*), para obtener la referencia a un usuario dado su nombre de usuario; getById(*uid*), similar al anterior, pero obtiene la referencia al usuario según la *id* del *Firestore*; update(*params*…), para actualizar los datos de un usuario, updateField(*uid*, *field*, *value*), actualiza un campo en concreto de un usuario (el teléfono por ejemplo) dada la identificación del usuario a modificar con el *value* dado y, finalmente, la función remove(*uid*) que borra el usuario según su *id* del *Firestore* también.

Estas funciones se repiten en mayor o menor medida en el resto de *API*, solo que en lugar de obtener o modificar objetos de la lista “odiseos”, crea, modifica, obtiene o borra los de la colección perteneciente a la *API* en concreto.

La Figura 32 presenta un fragmento de código de OdiseoApi, específicamente la función de create(*params*…), que se utilizará para ejemplificar modelo de tres capas. El módulo de React encargado de la creación de usuarios llama a la función *create* de la clase OdiseoApi suministrando los parámetros necesarios a partir de la información incluida por el usuario en la aplicación. En OdiseoApi se crea el objeto Odiseo con la palabra reservada *new* y se llama a la función *createSetDoc* del fichero *Firebase* (importado como "server" en el código) que recibe como parámetro la lista en la que crear el objeto (odiseos), el objeto odiseo y el identificador con el que se guardará en la base de datos. Coincide que el *uid* es el identificador creado por el *Authentication* de *Firebase*, por tanto, realmente antes de crear el objeto odiseo en la base de datos, se ha llamado a *Authentication* con el correo electrónico y la contraseña para generar al usuario en este servicio.

La figura 32 no devuelve ningún objeto del *Firebase,* pero si se llamara la función getById(*uid)* mencionada anteriormente, realizaría un *return* a la interfaz gráfica, cerrando el ciclo de datos mostrado en la Figura 25.

```
import * as server from "../../database/firebase";
import { Odiseo } from "../models/Odiseo";

export const OdiseoApi = {
    ...

  create: async function (userName, name, email, birthDate,
    phoneNumber, avatar, isAedo, isAdmin, isEducative,
    accountNumber, uid
  ) {
    const odiseo = new Odiseo(
      userName, name, email, birthDate, phoneNumber, avatar,
      isAedo, isAdmin, isEducative, accountNumber
    );
    await server.createSetDoc("odiseos", odiseo, uid);
  },

  ...

}
```

***Figura 32:*** *Fragmento de OdiseoAPI.js (Fuente: Elaboración propia)*

3.OdiseaApi.js

Este fichero resulta interesante también por toda la lógica que tiene a la hora de crear una experiencia en la aplicación. Como se muestra en la Figura 33, tras la llamada del *create* y el paso de parámetros, se comprueba qué tipo de calendario se utilizará (un único día, un rango de fechas, múltiples fechas o ciertos días de la semana como frecuencia); se crea el objeto new Odisea y se guarda en el *Firebase.* Después, con el ".then" se crean los calendarios por cada idioma facilitado en la base de datos y, finalmente, se cargan las imágenes en el servidor llamando a *uploadFile*.

```
import * as server from "../../database/firebase";
import { Odisea } from "../models/Odisea";
import { OdiseaCalendarApi } from "./OdiseaCalendarApi";

export const OdiseaApi = {
  ...
  create: async function (
    uid, name, description, images, allowEducative, price, languages,
    data, selectedDatePicker, location, maxCapacity, tags
  ) {
    /**
     * Tratar las dates segun el calendario seleccionado:
     * selectedDatePicker = 0 -> Single Day
     * selectedDatePicker = 1 -> Range of Days
     * selectedDatePicker = 2 -> Multiple Days
     * selectedDatePicker = 3 -> Frequency
     */
    let odiseaCalendarType;
    if (selectedDatePicker == 0) {
      odiseaCalendarType = "singleDateCalendar";
    }
    if (selectedDatePicker == 1) {
      odiseaCalendarType = "rangeDatesCalendar";
    }
    if (selectedDatePicker == 2) {
      odiseaCalendarType = "multipleDatesCalendar";
    }
    if (selectedDatePicker == 3) {
      odiseaCalendarType = "frequencyDatesCalendar";
    }

    const odisea = new Odisea(
      uid, name, description, images, allowEducative, price,
      languages, location, maxCapacity, 0, 0, tags
    );

    // Crea la odisea
    return (
      server.create("odiseas", odisea)
        .then((ref) => {
          //Crear el odisea Calendar para cada idioma
          odisea.languages.forEach(async (language) => {
            let languageID = language.id;
            await OdiseaCalendarApi.create(
              data[languageID], ref.id, language, odiseaCalendarType
            );
          });
        })
        // Sube las imagenes
        .then((ref) => {
          odisea.images.forEach(async (element) => {
            await server.uploadFile("images/" + element.assetId,
element.uri);
          });
        })
    );
```

```
  }
  ...
}
```

***Figura 33**: Fragmento de crear odisea en OdiseaApi (Fuente: Elaboración propia)*

Además, OdiseaApi incluye funciones que realizan llamadas de obtención de experiencias a la base de datos limitadas en tamaño, esperando a que el usuario oprima "Cargar más experiencias" al visualizarlas en la pantalla principal de la aplicación. Esto se implementó de esta manera para evitar lecturas excesivas. Si la base de datos tiene mil experiencias registradas, es mejor cargarlas de diez en diez a petición del usuario que sobrecargar el cliente móvil con todas las experiencias que no se irán a visualizar.

Esta *API* también se encarga del cálculo de la media de los comentarios, realizando una llamada a la base de datos para obtener la experiencia en concreto, y después con esa experiencia se llama a la función de *calculateAverage()* del modelo Odisea (Figura 23) en la capa de lógica de la aplicación. Este valor se devuelve del modelo a la *API* y de la *API* a la interfaz gráfica, mostrando por pantalla al usuario para que sepa la puntuación media que tiene basándose en las puntuaciones de otros usuarios.

Para finalizar, en la Figura 23 del modelo de Odisea, aparecía el mismo fragmento de código representado en la Figura 34. La función *geoHash* (Consultas de ubicación geográfica, s.f.) merece una explicación, ya que es la manera en la que *Firebase* guarda la localización.

```
// Crea un geoHash dada una localización
    const geoHash = geofire.geohashForLocation([
      Number(location.latitude),
      Number(location.longitude),
    ]);
    this.location = { ...location, geoHash: geoHash };
```

***Figura 34**: Fragmento geohash en el constructor del modelo Odisea. (Fuente: Elaboración propia)*

Cuando un usuario selecciona en el mapa una posición donde se realizará la experiencia, se guarda una longitud y una latitud, pero *Firestore* solo permite una única consulta compuesta; eso se traduce en que no se pueden realizar consultas de ubicación geográfica utilizando la latitud y la longitud como campos independientes. Para eso se utiliza el *geohash,* que se trata de un sistema para codificar la tupla (longitud, latitud) en una única cadena de caracteres en base32. En el sistema *geohash,* el mundo se divide en una cuadrícula rectangular y cada carácter de la cadena *geohash* especifica una de las 32 subdivisiones del *hash* del prefijo. Por ejemplo, el hash "abcd" es uno de los 32 *hashes* de cuatro caracteres que está integrado en el *hash* más grande "abc".

Cuanto más largo sea el prefijo entre dos *hashes*, más cerca están el uno del otro. "abcdef" está más cerca de "abcdeg" que "abcdff" sin embargo, lo contrario no ocurre igual. Dos áreas pueden estar muy cerca entre sí y tener *geohashes* muy distintos, tal y como muestra la Figura 35. Se pueden utilizar los *geohash* para almacenar y consultar documentos por posición en *Firestore*

con una eficacia razonable, y así se soluciona el problema ya que tiene un único campo en la consulta.


***Figura 35:*** *Ejemplo cuadrícula geohash. (Fuente: https://firebase.google.com/docs/firestore/solutions/geoqueries)*

4.ImageApi.js

El último archivo que merece la pena ser comentado es el de las imágenes, ya que esta *API* es la encargada de realizar la subida de ficheros al servicio del *Storage*. Igual que las otras *APIs*, recibe información de la interfaz de usuario, en este caso las fotos seleccionadas por el usuario, y las transmite al fichero del *Firebase* tal y como muestra la Figura 36, llamando a la función *uploadImage*. Este fichero también permite descargar imágenes o los iconos de las categorías que están implementadas en la aplicación; simplemente varía en la carpeta a la que apunta al realizar el *downloadFile*.

```
import * as server from "../../database/firebase";
export const ImageApi = {
  uploadImage: async function (imageName, uri) {
    return await server.uploadFile("images/" + imageName, uri);
  },
  downloadImage: async function (imageName) {
    return server.downloadFile("images/" + imageName);
  },
  downLoadCategoryIcon: async function (iconNameID) {
    return server.downloadFile("icons/" + iconNameID);
  },
  getCategoriesUris: async function (category) {
    return server.downloadFile("icons/" + category.name + "/" +
category.id);
  },
};
```

***Figura 36****: Código de ImageApi.js (Fuente: Elaboración propia)*

La Figura 37 presenta un fragmento del código del fichero de *Firebase,* y es la pieza del rompecabezas al que se conecta la *API* anterior (Figura 36) y sobre la que realiza la llamada de *uploadFile* y *downloadFile*. Este fichero se conecta al servicio del *Storage* previamente importado y carga o descarga las imágenes. Hay que mencionar que *Storage* trabaja con *URIs*, no utiliza ficheros propiamente dichos. Por tanto, a la hora de mostrar una imagen en la aplicación, esta se descarga el enlace de la imagen, lo asigna al componente que vaya a mostrarla y después el componente ya se descarga la imagen mediante el enlace asignado.

```
import { getDownloadURL, getStorage, ref, uploadBytes } from
"firebase/storage";
...
export const uploadFile = async (fileName, uploadUri) => {
  const storage = getStorage();
  const reference = ref(storage, fileName);
  const file = await fetch(uploadUri);
  const bytes = await file.blob();
  await uploadBytes(reference, bytes);
};

export const downloadFile = (fileName) => {
  const storage = getStorage();
  const starsRef = ref(storage, fileName);
  return getDownloadURL(starsRef);
      };
```

***Figura 37**: Fragmento de código de Firebase.js (Fuente: Elaboración propia)*

## 4.5 Implementación del patrón *Singleton*

El patrón *singleton* permite una única instancia de un objeto. Realmente en la aplicación no se ha implementado este patrón como tal (ya que no son objetos), pero sí que se ha hecho algo similar. Debido al funcionamiento de JavaScript y React Native, la información entre distintos componentes no se actualiza en tiempo real. Si un componente padre facilita a sus hijos una variable "usuario", cada componente hijo tiene una copia de "usuario"; por tanto, si el usuario de la aplicación cierra sesión en un componente, en los otros podría estar aún con la sesión abierta. Para solucionar este problema, se utilizan los llamados "contextos" de React.

Los contextos (Passing Data Deeply with Context, s.f.) es la manera que tiene React de pasar información entre componentes y que esta se mantenga actualizada. *Context* permite que el padre ofrezca información disponible y actualizada a todos los componentes que dependen de él sin tener que pasar la información como "props". Los *props* son la manera habitual que tiene React de compartir información entre componentes. Básicamente, el componente padre facilita al hijo cierta información como parámetros, y en el componente hijo se rescata y asigna en variables dicha información. Hay que mencionar que no se debe abusar de los contextos porque actúan como envoltorios y pueden llegar a saturar la aplicación.

En Aedo se han utilizado dos contextos por necesidad que están disponibles en toda la aplicación: el usuario activo y los favoritos del usuario. La Figura 38 muestra el fichero App.js que es el fichero principal de la aplicación. En el "return" se puede observar el componente "*NavigationContainerScreen"* que está envuelto por los dos contextos: *UserProvider* y *FavouritesProvider*.

```
import React from "react";
import {LogBox} from "react-native";
import UserProvider from "./src/businessLogic/contexts/userContext";
import FavouritesProvider from "./src/businessLogic/contexts/userFavourites";
import NavigationContainerScreen from
"./src/ui/navigations/NavigationContainer";

export default function App() {
  LogBox.ignoreAllLogs();
  return (
    <UserProvider>
      <FavouritesProvider>
        <NavigationContainerScreen />
      </FavouritesProvider>
    </UserProvider>
  );
}
```

***Figura 38**: Fichero App.js (Fuente: Elaboración propia)*

*UserProvider* está situado en la parte más externa y se encarga de facilitar el usuario autentificado en todo momento a todos los componentes descendientes. Algunas funciones requieren disponer del usuario identificado, como pueda ser reservar una experiencia, ver las recomendaciones personalizadas de la inteligencia artificial o visualizar las reservas. Si cada componente no dispusiera de la información actualizada de la variable "user", podría ser un desastre porque, por ejemplo, al cerrar sesión, este podría seguir reservando experiencias.

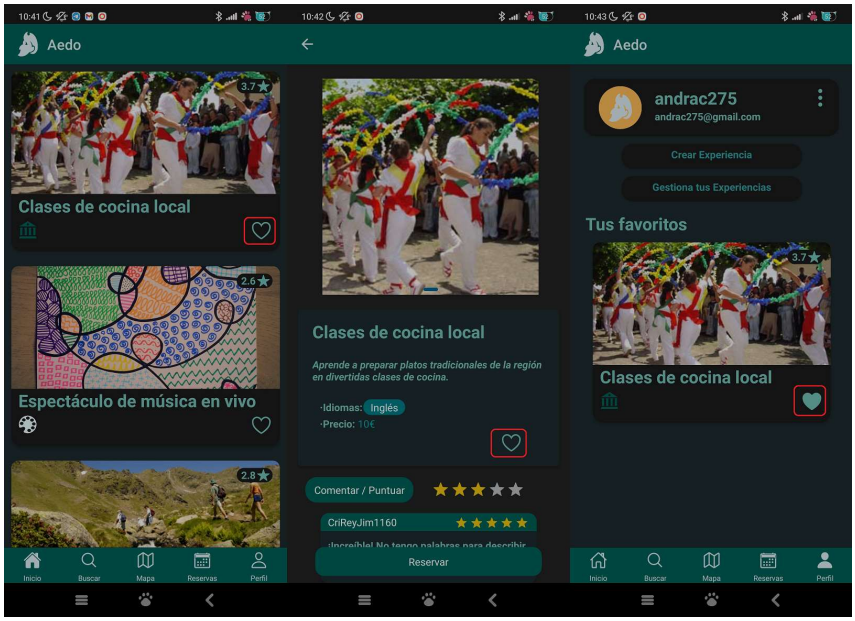



***Figura 39:** Pamtallas donde interactuar con los favoritos (Fuente: Elaboración propia)*

También fue necesario llevar constancia de los favoritos del usuario. Un usuario de Aedo puede registrar en sus favoritos las experiencias desde distintas partes de la aplicación. Puede guardar un favorito desde la página principal donde aparece el listado de experiencias, puede asignar un favorito en la página del buscador por proximidad de ubicación, o puede añadirla también en la página propia de la experiencia al consultarla. La Figura 39 muestra las distintas pantallas o componentes donde interaccionar con los favoritos. El contexto se ha implementado para que la información sobre los favoritos del usuario esté actualizada en todos y cada uno de los componentes dentro de la aplicación.

La Figura 40 es un fragmento del contexto, dispone de funciones para añadir y borrar favoritos a la lista del usuario, pero además tiene la definición del "FavouritesProvider", que es el contexto. En el fichero se observa el useState de "[userFavourites, setUserFavourites]" que son las variables compartidas a nivel global. Así se puede acceder a los favoritos del usuario en cualquier punto y se pueden cambiar también (añadir y borrar). En el apartado *UserFavouritesContext.Provider* del *return* se incluyen todas las variables que se quieren compartir, que en este caso es el listado de favoritos, el *setUserFavourites* y las dos funciones para añadir y borrar los favoritos.

El archivo del contexto de usuario es similar en funcionamiento, por lo que no se va a compartir el código fuente.

```
import React, {useContext, useEffect, useState} from "react";
import {FavouritesApi} from "../apis/FavouritesApi";
import {UserContext} from "./userContext";

export const UserFavouritesContext = React.createContext();

export const addFavourite = async (userID, odiseaID, userFavourites,
setUserFavourites) => {
  ...
};
export const removeFavourite = async (odiseaID, userFavourites,
setUserFavourites) => {
  ...
};

const FavouritesProvider = ({ children }) => {
  const user = useContext(UserContext);
  const [userFavourites, setUserFavourites] = useState([]);
  useEffect(() => {
    loadFavourites();
  }, [user]);

  const loadFavourites = async () => {
    async function getUserFavourites(userID) {
      await FavouritesApi.getAllByOdiseoId(userID).then((favourites) => {
        setUserFavourites(favourites);
      });
    }
    if (user) {
      await getUserFavourites(user.uid);
```

```
    }
  };

  return (
    <UserFavouritesContext.Provider
      value={{userFavourites, addFavourite, removeFavourite, setUserFavourites}}
    >
      {children}
    </UserFavouritesContext.Provider>
  );
};

export default FavouritesProvider;
```

***Figura 40:*** *Fragmento de código de FavouritesContext (Fuente: Elaboración propia)*

## 4.6 Pruebas realizadas

Para las pruebas se ha utilizado Jest[14]. Se trata de un *framework* específico para realizar pruebas en el lenguaje de JavaScript; además, también funciona en Node.js, React, React Native, Angular y otros más. Requiere de pocas configuraciones y ejecuta cada uno de los casos de prueba de una manera aislada, pero en ejecución paralela, lo que permite que realice rápido los *tests* porque se realizan a la vez. La herramienta de Jest se tiene que utilizar en conjunto con la librería de "testing-library" (*Testing Library Documentation, s.f.*) de React Native. Ambas trabajan a la vez para poder ofrecer las funcionalidades.

La Figura 41 muestra un ejemplo de un test realizado sobre la clase/componente de React Native "Create odisea screen", que es el componente utilizado para insertar las experiencias en la aplicación. En la prueba, Jest y *testing-library* renderizan el componente, rellenan los campos con valores y después comprueban que los campos realmente tienen el valor esperado; de esta manera, se puede comprobar que el componente almacena bien la información antes de ponerse en contacto con la capa lógica de la aplicación. Se pueden realizar distintas pruebas en un componente concreto. Todos se crean con la estructura "test" seguido de un nombre representativo.

```
import React from "react";
import { fireEvent, render, screen } from "@testing-library/react-native";
import CreateOdiseaScreen from
"../../../src/ui/screens/odisea/CreateOdiseaScreen";

describe("Create odisea screen", () => {
  test("Crear odisea", () => {
    render(<CreateOdiseaScreen />);
    const nombre = screen.getByPlaceholderText("Llegar a Itaca ...");
    fireEvent.changeText(nombre, "Elaboracion de fartons");
    expect(nombre.props.value).toBe("Elaboracion de fartons");
```

[14] https://jestjs.io/es-ES/

```
    const descripcion = screen.getByPlaceholderText(
      "Vuelve a tu hogar y descansa ..."
    );
    fireEvent.changeText(
      descripcion,
      "En 2 horas te enseño como se elaboran los fartons de manera
tradicional"
    );
    expect(descripcion.props.value).toBe(
      "En 2 horas te enseño como se elaboran los fartons de manera
tradicional"
    );

    const aforo = screen.getByPlaceholderText(
      "N° máximo de compañeros... (10 por defecto)"
    );
    fireEvent.changeText(aforo, 20);
    expect(aforo.props.value).toBe(20);
  });
});
```

***Figura 41**: Fragmento de Código de OdiseaTest.js. (Fuente: Elaboración propia)*

Cuando se ejecutan las pruebas mediante el comando pertinente, la consola de comandos muestra los distintos *tests* ejecutados y el resultado de si ha sido satisfactoria (o no) su ejecución (ver Figura 42), así como el tiempo necesario para realizarla. Si hay algún fallo durante la prueba, aparece un mensaje en la línea de comandos indicando qué prueba ha fallado y se informa en qué línea se tiene el error, de tal manera que se dispone de un punto de partida desde el cual empezar a corregir la prueba o el código.

```
Test Suites: 6 passed, 6 total
Tests:       6 passed, 6 total
Snapshots:   1 file obsolete, 0 total
Time:        6.237 s
Ran all test suites.
PS C:\Users\Andrac\Desktop\pinProject\aedoApp>
```

***Figura 42**: Resultado de las pruebas. (Fuente: Elaboración propia)*

Jest, además, ofrece un análisis de la cobertura del código en base a las pruebas realizadas. Las pruebas que se han automatizado son para las operaciones de crear y modificar usuarios y experiencias, que son el pilar fundamental de la aplicación, y tras realizar el análisis, Jest muestra la cobertura de los distintos ficheros y componentes de la aplicación. En la Figura 43 se enseñan dichos ficheros representados con barras de colores indicando la cantidad de líneas de código probadas. Se puede observar que la clase de odisea y odiseaManage tiene una cobertura superior al 60%; sin embargo, la cobertura de las APIs o a la base de datos no se pueden probar porque realizan conexiones al *Firebase* y no hubo manera de *mockear* las conexiones a la base de datos.

| File | | Statements | | Branches | | Functions | | Lines | |
|---|---|---|---|---|---|---|---|---|---|
| businessLogic | | 50% | 3/6 | 100% | 0/0 | 0% | 0/3 | 50% | 3/6 |
| businessLogic/apis | | 6.5% | 11/169 | 0% | 0/24 | 2.29% | 2/87 | 6.5% | 11/169 |
| businessLogic/models | | 9.21% | 7/76 | 0% | 0/4 | 0% | 0/27 | 9.21% | 7/76 |
| database | | 25% | 19/76 | 0% | 0/2 | 0% | 0/21 | 25% | 19/76 |
| ui/assets/images | | 100% | 3/3 | 100% | 0/0 | 100% | 0/0 | 100% | 3/3 |
| ui/components | | 83.33% | 5/6 | 50% | 1/2 | 100% | 1/1 | 83.33% | 5/6 |
| ui/components/molecules | | 0% | 0/31 | 0% | 0/5 | 0% | 0/13 | 0% | 0/30 |
| ui/components/organisms | | 40% | 42/105 | 9.09% | 2/22 | 27.58% | 8/29 | 40.77% | 42/103 |
| ui/hooks | | 42.85% | 6/14 | 16.66% | 1/6 | 60% | 3/5 | 46.15% | 6/13 |
| ui/screens/odisea | | 66.66% | 14/21 | 50% | 4/8 | 63.63% | 7/11 | 65% | 13/20 |
| ui/screens/odiseaManage | | 64.51% | 20/31 | 57.14% | 4/7 | 50% | 8/16 | 65.51% | 19/29 |
| ui/screens/profile/login | | 48.14% | 13/27 | 20% | 2/10 | 63.63% | 7/11 | 48.14% | 13/27 |
| ui/screens/profile/login/register | | 42.68% | 35/82 | 25% | 10/40 | 50% | 13/26 | 42.68% | 35/82 |
| ui/screens/profile/modifyProfile | | 57.89% | 22/38 | 0% | 0/6 | 46.66% | 7/15 | 57.89% | 22/38 |
| ui/styles | | 82.69% | 43/52 | 0% | 0/6 | 33.33% | 2/6 | 82% | 41/50 |

***Figura 43**: Análisis de cobertura de Jest. (Fuente: Elaboración propia)*

Por supuesto, quedan muchas partes de la aplicación por automatizar las pruebas, pero este es un buen punto de partida teniendo en cuenta que ya se comprueba de manera automática la creación de usuarios, la operación de *login* y la creación o modificación de las experiencias, que es lo más importante de la aplicación, y algo que debe funcionar en todo momento.

---

A parte de las pruebas realizadas en la aplicación mediante la herramienta *Jest*, recordar que se realizaron dos experimentos ya explicados en los apartados "3.2.1: Presentar la aplicación a un trabajador del campo" y "3.2.2: Presentar la aplicación a un profesor" de este documento. Las preguntas que se formularon durante los experimentos quedan recogidas en el Anexo 2. En general, algunas preguntas están enfocadas a la usabilidad y calidad del *software*, y otras están relacionadas con la interfaz, la ubicación de los elementos, los tiempos de carga, servicios ofrecidos, etc.

Posteriormente, las mismas preguntas del cuestionario fueron presentadas a distintos conocidos del equipo de desarrollo con un rango de edad comprendido entre 20 y 30 años para conocer su opinión; respondieron once personas en total. Las puntuaciones para las distintas preguntas obtenidas fueron superiores a ocho puntos sobre diez de media, siendo, en su mayoría, puntuaciones superiores a nueve puntos. La pregunta con menor puntuación fue de 8,36 en la que algunos encuestados opinan que la aplicación no brinda la información suficiente sobre cómo proceder cuando existe algún problema. Sin embargo, estos usuarios no han mencionado nada al respecto en el apartado de comentarios del cuestionario. Por otra parte, el apartado de interfaz ha recibido una puntuación media de diez sobre diez, seguido por 9,82 puntos en la pregunta de la creación de experiencias. En el anexo 2, se comparten los datos de la encuesta y los resultados.

Cuatro de los encuestados compartieron sugerencias para mejorar la aplicación. Entre ellas sugirieron implementar un chat interno en la aplicación, exportar la información de las

experiencias a distintos formatos, la posibilidad de compartirlas en redes sociales y la posibilidad de añadir una hora a la experiencia y no solo el día. Todas estas sugerencias han sido tomadas en cuenta y serán implementadas en versiones futuras de la aplicación.

## 4.7 Desafíos de programación

Este apartado engloba distintos problemas que se han tenido durante el desarrollo de la aplicación y están relacionados con el *backend* de la misma. Siempre se ha tratado de obtener la solución al problema que fuera lo más técnica posible.

### 4.7.1 Problema 1: Los contexts.

El primer problema ya se ha explicado en este documento (en la sección "4.5 Implementación del patrón *singleton*"), de modo que se hará un resumen. Se requería tener actualizado en todo momento en todos los componentes de la aplicación el usuario registrado. Cuando no se conocía bien el funcionamiento de React, esta particularidad dio muchos problemas porque no se entendía cómo era posible que, al cerrar sesión de usuario, éste siguiera estando activo y pudiera seguir reservando experiencias.

La solución fue el uso del *context,* explicado en el apartado 4.5 del documento. Más tarde, se vio la necesidad de aplicar un segundo contexto en la aplicación, que fue el de los favoritos de los usuarios.

### 4.7.2 Problema 2: Las fechas de Android y iOS. Clase Date para regular las fechas.

La aplicación trabaja con fechas en distintos apartados. Se registra la fecha de nacimiento, se asignan fechas a las experiencias, se reserva para una fecha concreta… La aplicación es multiplataforma y no se comporta igual en *Android* que en *iOS* a la hora de guardar las fechas.

Realizando pruebas, nos percatamos de que algunas fechas disponibles a la hora de reservar estaban disponibles en un sistema *Android* pero no en un sistema *iOS*. Investigando, se llegó a mirar la manera en la que se guardaba la información en el *Firebase,* que es un *TimeStamp* con la siguiente apariencia: "15 de diciembre de 2022, 12:00:00 UTC +1". Una vez se disponía de esta información, se realizaron comprobaciones creando experiencias y se observó cómo se almacenaban los *TimeStamp* en la base de datos, llegando a la conclusión de que en *Android* y *iOS* no se guarda la zona horaria de la misma manera, por tanto, crear una experiencia en un dispositivo *iOS* a partir de las 22:00 resultaba en la creación para el día posterior porque sumaba dos a la zona horaria, mientras que en *Android* no se añadía ningún valor.

Esta problemática se solucionó comprobando el sistema operativo del dispositivo justo en el momento antes de crear la fecha, y si era un *iOS* se substraían dos horas. Estas líneas de

código se implementaron en todas las partes de la aplicación en las que se trabajara con fechas, una solución que funcionaba, pero era poco elegante.

En una etapa del desarrollo más tardía, se optó por incluir una librería de fechas (*date-fns documentation, s.f*) junto a una clase de utilidades llamada *DateUtils* para solucionar el problema. A partir de ese momento, todas las líneas de código relacionadas con las fechas atravesaban la clase *DateUtils* para obtener la fecha correcta y formateada y que estuviera todo centralizado en un único sitio para que, de tener la necesidad de realizar cambios, no tener que hacerlo en distintos componentes y clases de JavaScript, sino únicamente en la clase de utilidades de la fecha. La Figura 44 muestra las distintas funciones que tiene la clase de utilidades de fechas: dispone de los formateadores de fecha según el tipo de calendario; también una función para realizar un reinicio de la hora, minutos y segundos; además, devuelve si las fechas están disponibles en un calendario a la hora de realizar reservas de una experiencia; etc. La lógica ahora está recogida en esta clase y no en distintos componentes de React, donde se necesitaban estas funciones y estaban duplicadas.

```
import { set, sub, add } from "date-fns";
import { Platform } from "react-native";
import { OdiseaDateApi } from "../apis/OdiseaDateApi";

export const formatDate = (date) => {
  ...
};

export const formatMultipleDates = (dates) => {
  ...
};

export const formatRange = (range) => {
  ...
};

export const getButtonLabel = (date) => {
  ...
};

const resetHourMinSec = (date) => {
  ...
};

export const setHoursMins = (date, hours, minutes) => {
  ...
};

export const dateIsDisabled =
(date,odiseaCalendar,odiseaCalendarType,fullDates,odisea) => {
  ...
};

const checkEducativeOptions = async (date, odisea) => {
  ...
};
```

***Figura 44**: Funciones de la clase de utilidades DateUtil. (Fuente: Elaboración propia)*

### 4.7.3 Problema 3: Estructura de calendarios, fechas y reservas.

La Figura 45 es una versión simplificada del modelo de datos de la Figura 13 y se va a utilizar para explicar la problemática que se tuvo y cómo se solucionó. La aplicación se ha ido desarrollando de manera incremental por *sprint*s. Inicialmente, una experiencia era de un único idioma y solo se ofrecía en un día concreto; por tanto, un *Odiseo* realizaba una *Reservation* sobre una experiencia. En *sprints* posteriores se implementaron los diferentes idiomas en los que se podía ofrecer una experiencia y se permitía ofrecerla en distintos días. El modelo de datos antiguo había quedado obsoleto y necesitaba ser actualizado para que pudiera albergar las nuevas funcionalidades. Tras el análisis del problema, se optó por la solución presentada en la Figura 45 en la que, al generar una experiencia, se crea un calendario asociado a dicha experiencia por cada lenguaje, de esta manera se puede ofrecer cada experiencia en distintos idiomas y además permite que cada idioma distinto se pueda impartir en fechas distintas. Además, permitió el uso de distintos tipos de calendarios como única fecha, rango de fechas, días múltiples o una periodicidad por días.

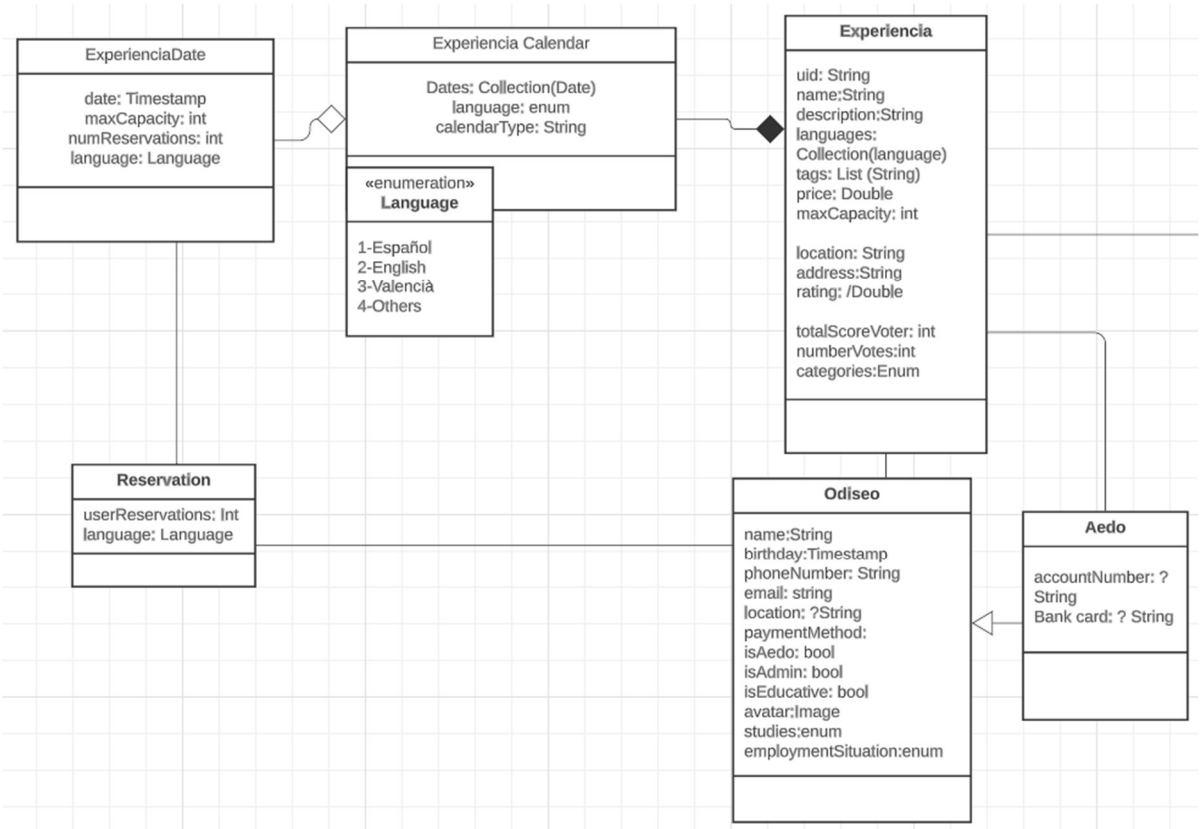



***Figura 45**: Modelo de datos simplificado con las experiencias, calendarios, fechas y reservas. (Fuente: Elaboración propia)*

Los objetos *ExperienciaDate* se crean sólo cuando un usuario reserva para un día en concreto. Estos objetos tienen un idioma asignado según lo escogido por el usuario, así que se enlazan con el *ExperienciaCalendar* en el idioma concreto. De esta manera, se tiene también un registro del aforo máximo por días.

El campo *maxCapacity*, que está presente en *ExperienciaDate* y *Experiencia*, está duplicado. Duplicar campos no es una solución técnica en programación, pero se optó por esta implementación para evitar un número de consultas excesivas sobre *Firebase*. La razón es que el camino a recorrer desde *Reservations* a *Experiencia* pasa por dos clases intermedias (*ExperienciaDate* y *ExperienciaCalendar*) y para transitar de una clase a la siguiente hay que realizar consultas *where* con los distintos identificadores de los documentos (objetos) relacionados. En la base de datos se almacenan numerosas *ExperienciaDate* y *ExperienciaCalendar* por tanto el número de consultas a realizar eran muy elevadas buscando el "camino" correcto. Con este campo duplicado, se tenía a mano en seguida la capacidad máxima, ya que una reserva se hace para una única *ExperienciaDate*.

El campo *numReservations* de *ExperienciaDate* también sería prescindible ya que se pueden contar el número de reservas enlazadas a cada *ExperienciaDate* pero se optó por guardarlo como un atributo más por la misma razón que antes: evitar llamadas excesivas. Cuando un usuario intenta reservar para un día concreto, en el calendario aparecen las fechas habilitadas o deshabilitadas según si la persona que imparte la experiencia permite o no que se realice ese día la experiencia, pero además, también se deshabilita el día también si el aforo está al máximo. Por esta razón, hay que comprobar para cada *ExperienciaDate* del calendario si hay disponibilidad, y resulta que se efectúan un menor número de llamadas comprobándolo directamente sobre el día en el campo *numReservations*.

### 4.7.4 Problema 4: Pasar categorías de padres a hijos para evitar tantas llamadas

Inicialmente las categorías de las experiencias se almacenaban en la aplicación local. Existía una carpeta con los distintos iconos asociados a las categorías y en el código de los componentes, se cargaba de un listado preestablecido. Esto no ocurre en la versión actual, pues las categorías se guardan en una colección del *Firebase* y los iconos en el *Storage*. Estas categorías son editables desde el panel de administración de la página web de la aplicación.

Lo que en un principio funcionaba, dejo de funcionar cuando se migraron las categorías a la nube en la base de datos. Es por lo que se tuvo que crear una *API* para las categorías y empezar a realizar llamadas para obtener los nombres y los iconos de las categorías. Nos percatamos de que el número de llamadas se había incrementado con creces porque la aplicación tiene muchas ventanas y muchos componentes. Por poner un ejemplo, cada experiencia que se muestra en la ventana de inicio de la aplicación es un componente distinto y cada componente realiza una consulta en la base de datos para obtener las categorías que tiene asociadas, y después los iconos para cada categoría. Se ha puesto el ejemplo de la ventana de inicio, pero los iconos y los nombres de la categoría se muestran en distintas ventanas y muchos componentes, como ya se ha comentado.

Para evitar este problema, ahora se realiza la llamada de las categorías y las imágenes sobre la base de datos en el componente más externo, el componente padre, que son las ventanas principales de la aplicación: Inicio, buscar, mapa, reservas y perfil, y se suministran a todos los componentes descendientes mediante los *props* de React Native. Los componentes hijos tan solo tienen que rescatar el nombre y la *URI* de la imagen facilitada por el padre.

Se valoró la opción de realizar un contexto para las categorías, pero como hay que evitar crear contextos de más por temas de rendimiento y las categorías no es un atributo que se modifique a menudo (tener en cuenta que se crean o modifican desde la página web cuando los administradores lo deciden, no es una operación que se realice frecuentemente), al final se descartó la idea del contexto para las categorías y no implementarlo fue la opción más viable en términos de rendimiento. Como mucho puede pasar que al acceder a una ventana no se obtenga el listado actualizado de categorías porque en ese momento se ha realizado un cambio en la web, pero tras cambiar de ventana se actualizaría. No es un factor crítico como el usuario autentificado o los favoritos en los que sí que se tuvo que implementar el contexto.

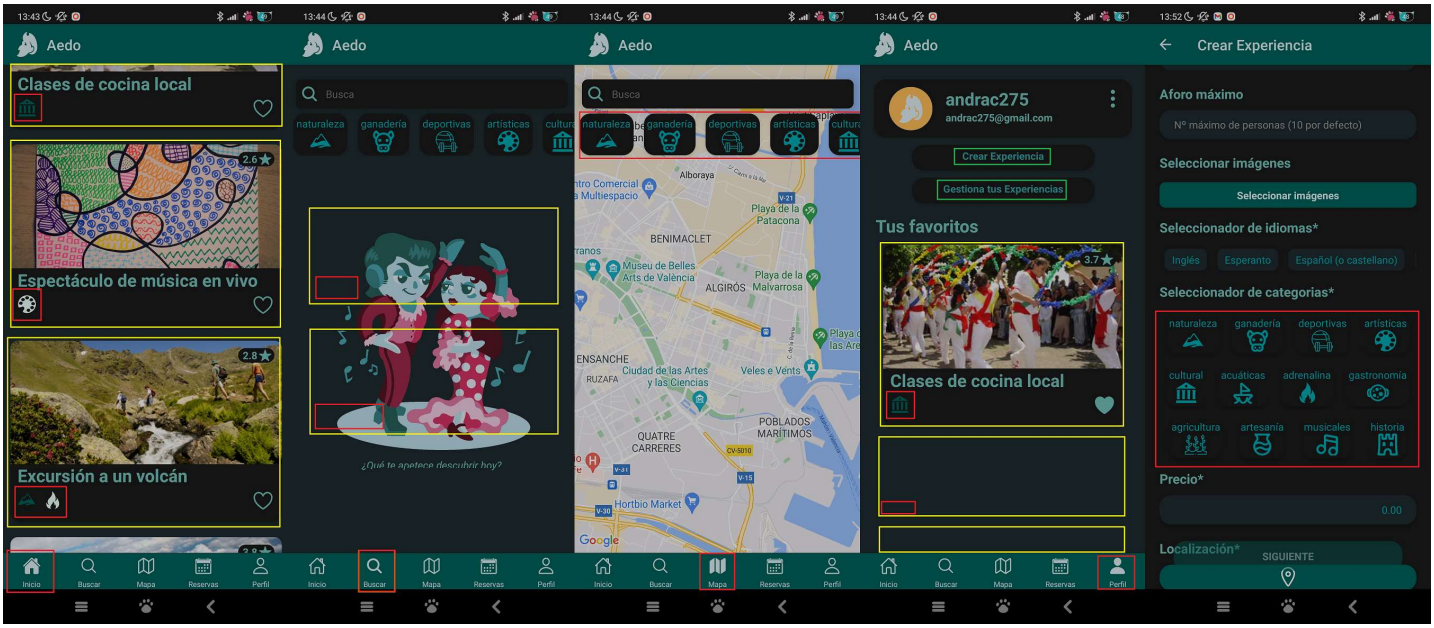



***Figura 46**: Pantallas de la aplicación donde se muestran las categorías. (Fuente: Elaboración propia)*

La Figura 46 muestra las distintas ventanas de la aplicación en las que se dibujan las categorías. Los recuadros rojos son las categorías en cada una de las distintas ventanas. Los recuadros amarillos representan componentes (en la ventana de "buscar" aparecen tarjetas de experiencias por proximidad, no hay cerca de la ubicación del dispositivo desde el cual se han realizado las capturas de pantalla, es por lo que aparece el dibujo de los bailarines en su lugar, pero ahí deberían aparece más tarjetas con sus categorías asociadas). Cada componente en amarillo realizaría llamadas para obtener las categorías, y pueden llegar a ser muchos ya que depende de la cantidad de experiencias registradas en la base de datos y la navegación que realice el usuario. Por eso la opción de recibir lo necesario desde el componente padre ha sido la implementada.

# 5. Cronología del TFG

La cronología de la aplicación se divide en seis *sprints* de programación y un séptimo inicial que se empleó para definir la idea de la aplicación, evaluar la idea de negocio, realizar el estudio de mercado, mapa de características, modelo de negocio y proyección económica. Todo el procedimiento está relacionado con los proyectos de emprendimiento. También se tuvo que elaborar un *pitch* de la idea de negocio, preparar el entorno de desarrollo y formarse los miembros para no afrontar el proyecto sin tener ningún conocimiento.

A continuación, se explican los seis *sprints* principales y en qué se enfocó el equipo de desarrollo en cada uno de ellos. Cabe mencionar que el trabajo realizado por los miembros fue el de *full stack*. Aunque este documento esté enfocado al *backend,* todos realizaron tareas de *full stack.* Por tanto, la información mostrada en los *sprints* siguientes serán tareas de todo tipo.

## 5.1 Sprint 1 (PIN: Proyecto de Ingeniería del *software*)

Durante el primer *sprint* se implementaron las funcionalidades básicas para que la aplicación tuviera usuarios, se pudieran registrar experiencias y se pudieran apuntar. Para ello, se programaron las distintas ventanas necesarias como la barra de navegación inferior, la pantalla de inicio donde aparecían listadas todas las experiencias y la posibilidad de acceder a las mismas a ver la información y poder reservar. También se implementaron las opciones de modificación y borrado, tanto de las experiencias como de los usuarios.

## 5.2 Sprint 2 (PIN: Proyecto de Ingeniería del *software*)

En el segundo *sprint* se tuvo que adaptar el proyecto al modelo de tres capas que se ha comentado en el documento (sección “4.4 Implementación facade pattern y el modelo tres capas”) y se implementó el contexto de usuario que era necesario también para la aplicación. En esta iteración se implementó, además, la posibilidad de registrar experiencias en diferentes idiomas y diferentes calendarios, lo que obligó a reestructurar el modelo de datos. Se puso en funcionamiento también el mapa, el buscador, el control del aforo máximo de las experiencias y poder comentar en ellas.

## 5.3 Sprint 3 (PIN: Proyecto de Ingeniería del *software*)

El tercer *sprint* fue relajado en cuanto a funcionalidades. El equipo se dedicó a pulir aspectos de la aplicación ya implementados y pulir errores y excepciones para dejar la aplicación funcional de cara a la Feria de Proyectos. Hubo distintas reuniones con los alumnos de Bellas Artes y se implementaron las interfaces gráficas de las distintas pantallas según el material

facilitado por los estudiantes. Como no se disponía de maquetas de todas las ventanas de la aplicación, el resto se tuvo que implementar con una estética similar a la proporcionada por los compañeros de Bellas Artes.

En cuanto a funcionalidades, se implementó una versión muy básica de la inteligencia artificial; más que una inteligencia artificial era un algoritmo de recomendaciones según las experiencias que se habían consultado hasta el momento. También se trataron los filtros de las experiencias en la pantalla del buscador y la ventana del mapa.

## 5.4 Sprint 4

El objetivo de la cuarta iteración fue empezar con la página web para realizar las tareas de administración y otras tareas que desde el dispositivo móvil pudieran resultar más tediosas de realizar. Se configuró el servicio de *hosting* de *Firebase* para la página web, se empezó con una visión básica del inicio de sesión y el perfil de usuario y la posibilidad de añadir y borrar idiomas y categorías.

Se empezaron a investigar más a fondo algoritmos para elaborar la inteligencia artificial.

## 5.5 Sprint 5

La quinta iteración se centró en seguir añadiendo funcionalidades básicas como las del dispositivo móvil a la página web, programar la inteligencia artificial e implementar la posibilidad de tener cuentas educativas para los profesores. También se implementó y mejoró el *script* de generación de datos en el *Firebase,* el cual era necesario para la inteligencia artificial.

## 5.6 Sprint 6

Durante el sexto y último *sprint,* el equipo se centró en seguir añadiendo funcionalidades a la página web y acabar de programar y pulir la inteligencia artificial. Se implementó también la posibilidad de poder realizar múltiples reservas desde una misma cuenta. También se realizaron distintos *refactorings* tanto de *front* como de *back.* Se generó la *APK* de la aplicación y se publicó en la tienda de Google.

# 6. Conclusiones y trabajo futuro

## 6.1 Conclusiones

Tras llegar al final del segundo *MVP* de la aplicación y tener que presentar el trabajo de final de grado, se deben realizar las conclusiones y no hay mejor manera de hacerlo que comparando el trabajo realizado con los objetivos definidos en el apartado "1.2 Objetivos"; de entre todos ellos, se han cumplido prácticamente todos:

- Se ha conseguido elaborar, desarrollar y evaluar la idea de negocio teniendo en cuenta el mercado actual y todas las aplicaciones disponibles.
- Se ha comprobado mediante experimentos que el producto *software* resulta del agrado de los usuarios. Las encuestas realizadas lo han corroborado.
- Se ha generado una estructura *backend* de código de calidad, utilizando el modelo de tres capas y reutilizando todo el código posible en forma de componentes de React Native.
- Se han empleado tecnologías, herramientas y librerías existentes para facilitar las tareas de programación. En el apartado "4.2 Herramientas utilizadas" se habló al respecto.

Sin embargo, el objetivo que no se ha cumplido al cien por cien ha sido el de conseguir una aplicación que cumpliera con las características establecidas por el equipo de desarrolladores. Es cierto que la aplicación incluye numerosas funcionalidades y las funciones básicas esperadas de una aplicación de esta índole, pero también es verdad que un elemento diferenciador de la aplicación respecto a sus competidores iba a ser un chat interno, y esa característica no se ha implementado. Teniendo como referencia la Tabla 1, tampoco se ha implementado la interfaz de usuario en inglés, ni la valoración de los consumidores de experiencias por parte de los ofertantes, ni tampoco los elementos de gamificación. Todo esto quedará en el *backlog* para iteraciones futuras de programación. Un *software* que dispone de contenido en el *backlog* es una buena señal porque indica que siempre hay nuevas funcionalidades por añadir y está "vivo" el proyecto.

Por otra parte, consideramos que hay que seguir trabajando en el apartado de deuda técnica y la cobertura del código mediante pruebas automatizadas. Quedan muchos elementos los cuales deberían de automatizarse para evitar tener que realizar las pruebas de manera manual cada vez que se realizan cambios en la aplicación. Para finalizar, la deuda técnica, a medida que se genera código, siempre es posible seguir reduciéndola y para ello está la herramienta SonarQube.

## 6.2 Trabajo futuro

Han quedado muchas funcionalidades de la aplicación en el tintero, como se ha comentado en el apartado de conclusiones, que no han podido ser implementadas, con lo cual la aplicación tiene trabajo futuro. Entre ellas se destacan las siguientes:

- Mejorar el sistema para recomendar experiencias que estén a cierta distancia de la ubicación actual pero no muy lejos, para fomentar la movilidad y hacer que los usuarios visiten las zonas menos pobladas.
- Implementar un sistema de pago mediante el dispositivo móvil, ya sea para las experiencias como para los futuros servicios *premium*.
- Implementar el chat interno de la aplicación para que los usuarios y los aedos que imparten las experiencias puedan comunicarse sin necesidad de compartir el número de teléfono.
- Sistema de notificaciones. No se implementó ninguno y sería interesante disponer de uno que avise cuando una reserva está próxima o si se cancelan las reservas. La funcionalidad exacta habría que discutirla.
- Seguir introduciendo la funcionalidad del móvil faltante en la página web.
- Autentificación con distintas cuentas: Google, Facebook.
- Posibilidad de compartir experiencias en redes sociales.
- Tareas de mejora continua de código. Seguir reduciendo la deuda técnica de la aplicación y generando más pruebas automatizadas.
- Implementar distintos idiomas de interfaz de usuario. Al estar enfocada en España, el idioma de la aplicación es el español, aunque se permita impartir experiencias en otros idiomas.
- Definir el trato de datos de carácter personal según la Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD) antes de publicar la aplicación.

# 7. Glosario de términos

*API: Application Programming Interface.* Una API actúa de intermediario entre dos partes. Pueden ser dos aplicaciones distintas o dentro de la misma aplicación. Por ejemplo, en una aplicación separada por capas, se utilizan APIs para conectar una parte de la aplicación con la otra capa.

Apk: *Andoid Application Package. (*Aguilar, 2020) Packete de instalación que contiene los datos de una aplicación y permite su instalación en dispositivos Android.

Aplicación nativa: Una aplicación nativa se trata de un tipo de *software* desarrollado para ser utilizado en dispositivos móviles, tanto en el sistema Android (teléfonos o *tablets*), como Apple (IPhone y IPad)

*Backend:* Complementario al *frontend*. El *backend* (*Machuca, 2022*) está formado por todos aquellos elementos de la página web o de la aplicación los cuales no son accesibles para los usuarios finales de la aplicación.

*Bug:* Concepto que se utiliza en el ámbito de la informática para referirse a errores que se han producido en un programa.

*Code Smell:* Los *code smell* son una serie de síntomas en el código que indican que las cosas puede que no se estén haciendo como es debido, lo que puede desencadenar en errores y problemas futuros.

*Framework:* Un *framework* (*Framework, 2022*), o marco de trabajo, tiene como objetivo facilitar las tareas de desarrollo que surgen a la hora de programar, estos aceleran el proceso de programar gracias a que facilitan las tareas propias por tanto su objetivo al final es facilitar las tareas a los programadores.

*Frontend:* Complementario al *backend*. El *frontend* (*Chapaval, 2018*) está formado por la parte visible de la aplicación, sea una interfaz web, una aplicación móvil o de escritorio. Es la parte de la aplicación con la que los usuarios interactúan. Las letras, los colores, elementos… Que los usuarios pueden visualizar.

*Full stack*: Programador que realiza las tareas tanto de *frontend* como de *backend*.

*Getters o Setters:* Funciones mediante las cuales se consulta el valor de atributos privados de una clase desde una clase externa (*get)* como se asigna un nuevo valor a dicho atributo privado desde la clase externa (*set)*.

*MVP: Minimum Viable Product.* (*Gasbarrino s.f.)* Producto viable mínimo. En términos de desarrollo de aplicaciones, se trata de un producto con un número de características suficientes como para satisfacer las necesidades iniciales de los clientes.

*Product Owner:* Dentro de las metodologías ágiles, se trata de una figura dentro de la empresa que actúa de representante entre la parte cliente y el equipo de programadores. Es el responsable del producto final ya que dispone de la visión del cliente.

*Software*: Conjunto de reglas o programas que dan instrucciones a un ordenador ( u otro dispositivo electrónico) para que realice unas tareas específicas.

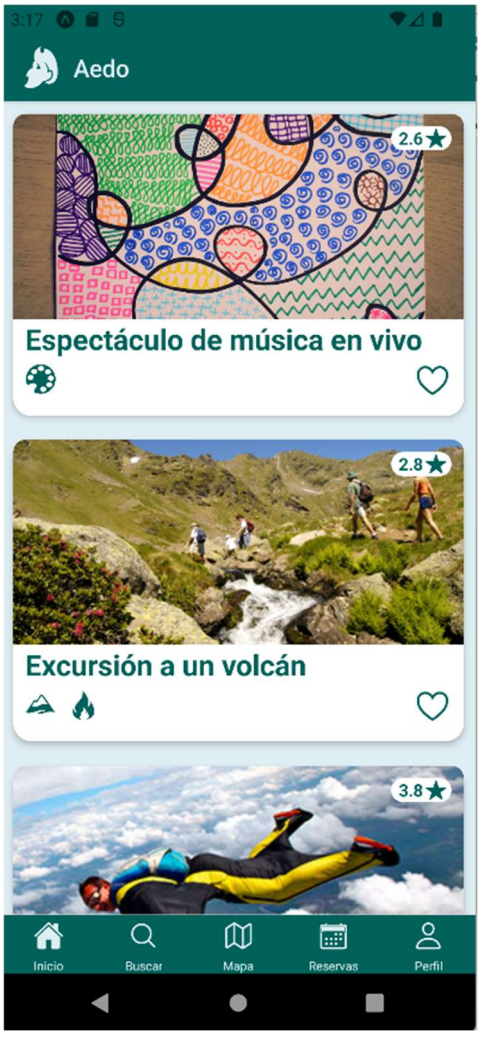
# 8. Bibliografía


- Aguilar, R. (17 de Julio, 2020). Qué es un APK de Andoid, cómo se instala y diferencias con las apps normales. *Xatakandroid.* https://www.xatakandroid.com/aplicaciones-android/que-apk-android-como-se-instala-diferencias-apps-normales Consultado en Mayo 2023.
- Bello, E. (28 de Abril, 2021). Descibre qué es el Extreme Programming y sus características. *IEBSchool*. https://www.iebschool.com/blog/que-es-el-xp-programming-agile-scrum/ Consultado en Abril 2023.
- Chapaval, N. (2018). Qué es Frontend y Backend: diferencias y características. *Platzi.* https://platzi.com/blog/que-es-frontend-y-backend/ Consultado en Mayo 2023.
- Clean-jsdoc-theme. (s.f.). Clean-jsdoc-theme. *Clean Jsdoc Theme.* https://github.com/ankitskvmdam/clean-jsdoc-theme Consultado en Febrero 2022.
- Cloud Firestore. (s.f.). Cloud Firestore. *Firebase.* https://firebase.google.com/docs/firestore Consultado en Octubre 2022.
- Cloud Storage para Firebase. (s.f.). Cloud Storage para Firebase. *Firebase.* https://firebase.google.com/docs/storage Consultado en Noviembre 2022.
- Consultas de ubicación geográfica. (s.f.). Consultas de ubicación geográfica. *Firebase.* https://firebase.google.com/docs/firestore/solutions/geoqueries Consultado en Diciembre 2022.
- Date-fns documentation. (s.f). Date-fns documentation. *Date-fns.org.* https://date-fns.org/docs/ Consultado en Abril 2022.
- Díaz, C. (5 de Agosto, 2019). Code smells y deuda técnica. *Openwebinars.* https://openwebinars.net/blog/code-smells-y-deuda-tecnica/ Consultado en Mayo 2023.
- Drumond, C. (s.f.). ¿Qué es scrum?. *Atlassian.* https://www.atlassian.com/es/agile/scrum Consultado en Abril 2023.
- Firebase Authentication. (s.f.). Firebase Authentication. *Firebase.* https://firebase.google.com/docs/auth Consultado en Octubre 2022.
- Firebase hosting. (s.f.). Firebase hosting. *Firebase.* https://firebase.google.com/docs/hosting Consultado en Enero 2023.
- Framework. (26 de Julio, 2022). Framework. *Edix*. https://www.edix.com/es/instituto/framework/ Consultado en Mayo 2023.
- Gasbarrino, S. (s.f.). MVP: qué es el producto mínimo viable, cómo hacerlo y ejemplos. *Hubspot.* https://blog.hubspot.es/sales/producto-minimo-viable Consultado en Mayo 2023.
- Hallie, L y Osmani,A. (2023). *Learning Patterns.* https://archive.org/details/learning-patterns/learning-patterns-final-v1.1/ Consultado en Abril 2023.
- Hernández, M. (8 de Febrero, 2021). ¿Qué es NPM?. *Freecodecamp.* https://www.freecodecamp.org/espanol/news/que-es-npm/ Consultado en Mayo 2023.
- Kuhn, J. (2009). Decrypting the MoSCoW analysis. *The workable, practical guide to Do IT Yourself*, *5*.
- Lucas, J. (4 de Septiembre, 2019). Qué es NodeJS y para qué sirve. *Openwebinars.* https://openwebinars.net/blog/que-es-nodejs/ Consultado en Mayo 2023.
- Machuca, F. (22 de Mayo, 2022). Backend: ¿qué es y para qué sirve este tipo de programación?. *Crehana.* https://www.crehana.com/blog/transformacion-digital/que-es-el-backend-y-como-usarlo/ Consultado en Mayo 2023.
- Mahendra, S. (s.f.). ¿Cuál es el mejor IDE para desarrollar JavaScript en 2021?. *Developers.dev.* https://www.developers.dev/tech-talk/es/javascript/what-is-the-best-ide-to-develop-javascript-in-2021.html Consultado en Mayo 2023.
- Maurya, A. (2022). *Running lean*. O'Reilly Media, Inc.
- Martínez, M. (24 de Junio, 2020). *¿Qué son los patrones de diseño de software?.* https://profile.es/blog/patrones-de-diseno-de-software/ Consultado en Mayo 2023.

- Objetivos de desarrollo sostenible. (25 de Setiembre, 2015). Objetivos de desarrollo sostenible. *ONU*. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ Consultado en Abril 2023
- Passing Data Deeply with Context. (s.f.). Passing Data Deeply with Context. *React.* https://react.dev/learn/passing-data-deeply-with-context Consultado en Marzo 2023.
- Primeros pasos con Cloud Firestore. (s.f.). Primeros pasos con Cloud Firestore. *Firebase.* https://firebase.google.com/docs/firestore/quickstart Consultado en Abril 2023.
- Radigan, D. (s.f.). ¿Qué es kanban?. *Atlassian.* *https://www.atlassian.com/es/agile/kanban* Consultado en Abril 2023.
- Sentrio. (15 de Diciembre, 2021). Qué es SonarQube: Verifica y analiza la calidad de tu código. *SentrioIO.* https://sentrio.io/blog/que-es-sonarqube/ Consultado en Mayo 2023.
- SonarQube Documentation. (s.f.). SonarQube Documentation. *SonarQube.* https://docs.sonarqube.org/latest/ Consultado en Diciembre 2022.
- Testing Library Documentation. (s.f.). Testing Library Documentation. *Testing Library.* https://testing-library.com/docs/react-native-testing-library/intro/ Consultado en Marzo 2023.

# Anexo I: Manual de uso de la aplicación

**Capturas de pantalla de la aplicación móvil**




Pantalla de inicio:

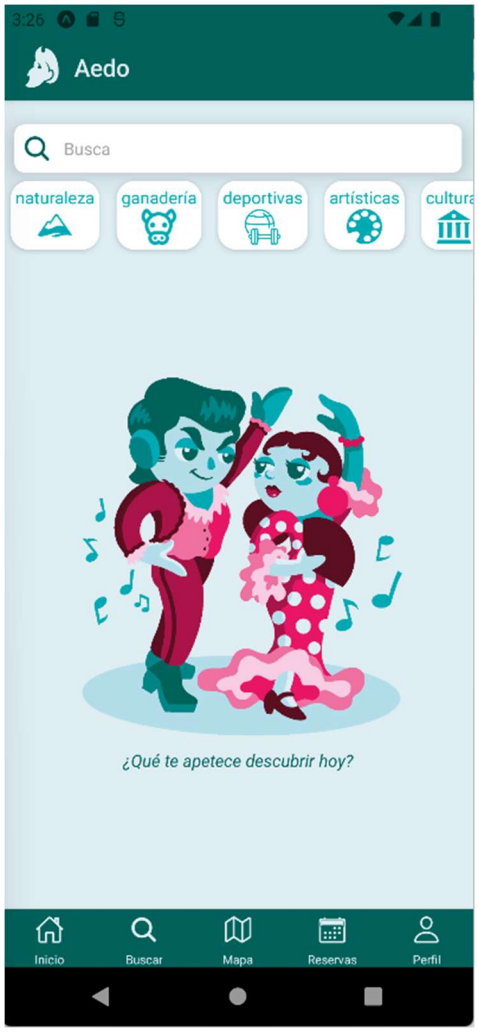
En esta pantalla aparecen todas las experiencias propuestas por los distintos usuarios en la aplicación. En la parte inferior aparecen recomendaciones personalizadas por una inteligencia artificial.


Pantalla de información de una experiencia:

Aparece la información de la experiencia, así como los comentarios y valoraciones dados por otros usuarios de la aplicación. Además, se puede reservar la experiencia.

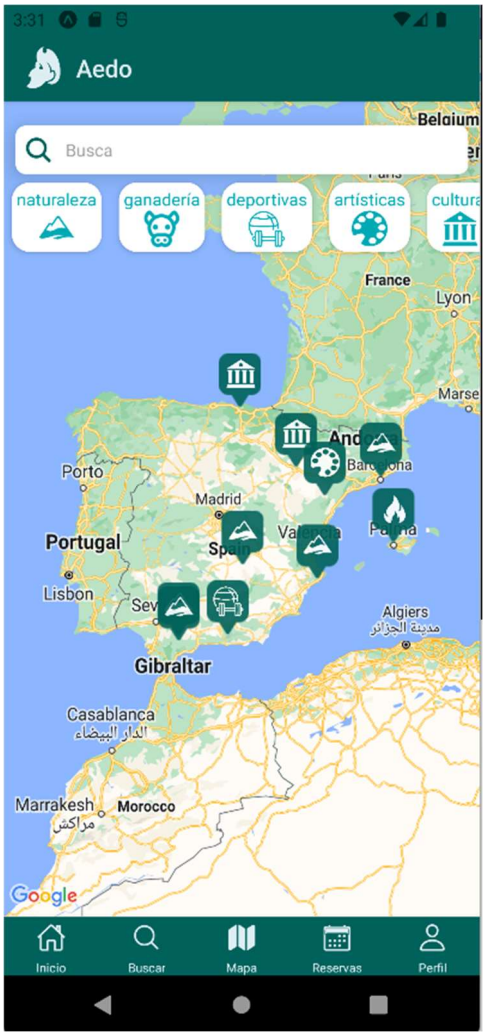

Pantalla de selección de fecha de reserva:

Al intentar reservar una experiencia, aparecen los distintos idiomas disponibles y un calendario para poder escoger la fecha de la reserva.



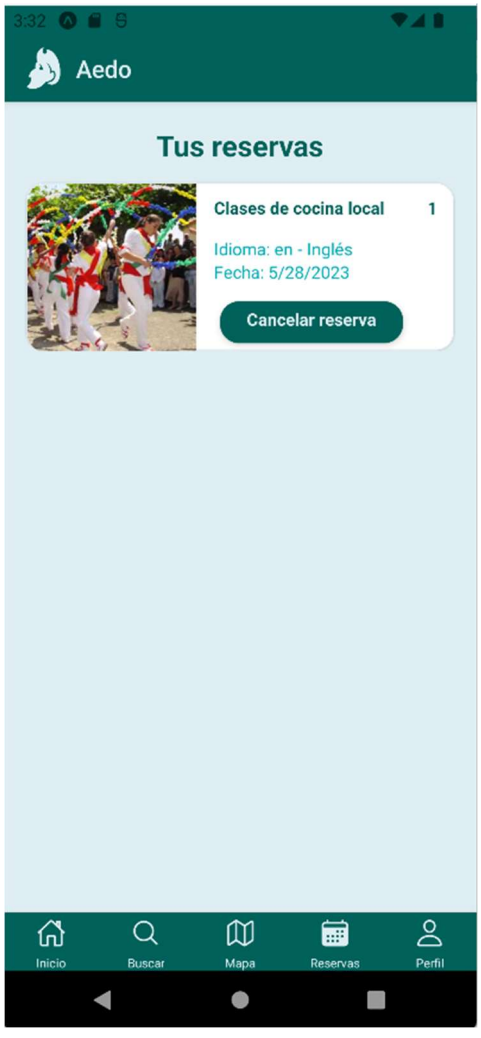

Pantalla de buscar:

Esta pantalla muestra experiencias próximas a la ubicación del dispositivo. Además, ofrece un buscador para buscar por localidad y un sistema de filtros por categorías de experiencias.



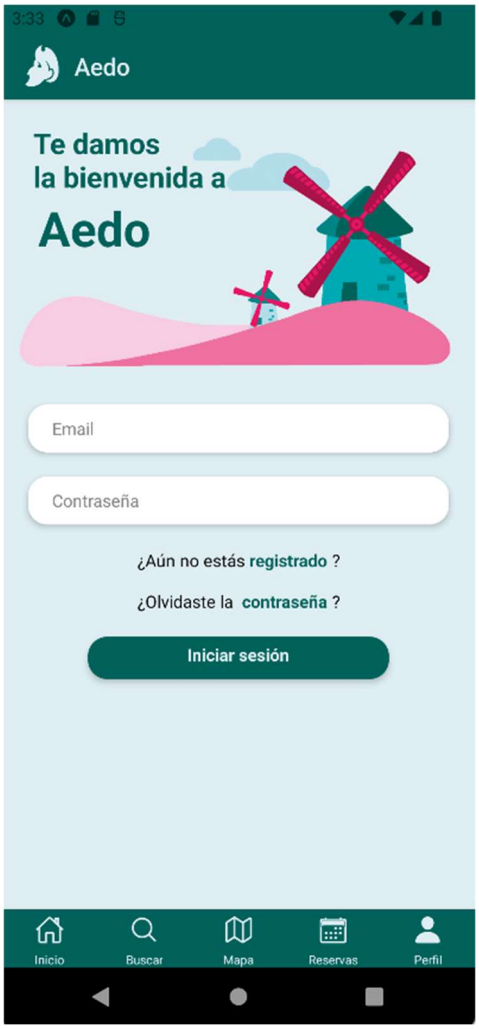

Pantalla del mapa:

En este elemento aparece el mapa y todas las experiencias clasificadas con sus categorías y la ubicación. Se puede acceder a la "pantalla de información de una experiencia" tocando el icono y el nombre en el mapa.



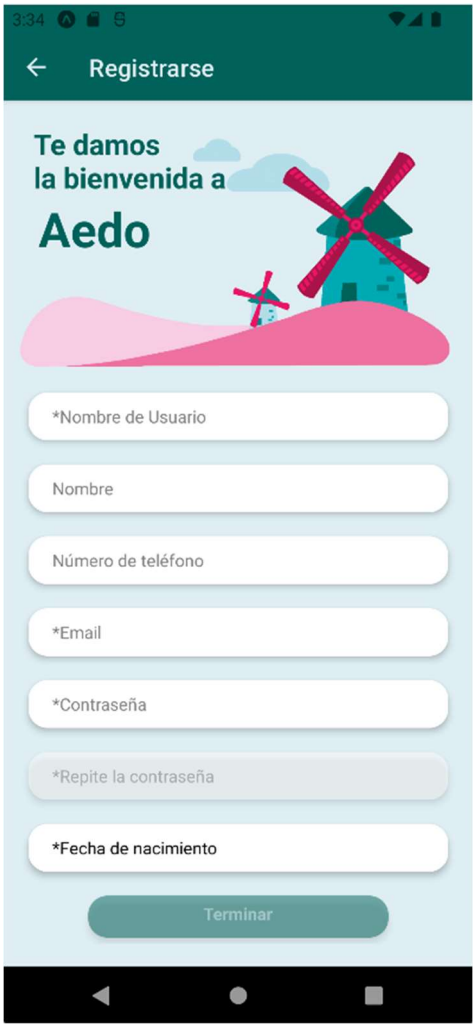

Pantalla de tus reservas:

En esta interfaz aparecen las reservas realizadas por el usuario que ha iniciado sesión y la información de estas como la fecha, el idioma y el número de reservas realizadas. Permite cancelar la reserva también.




Pantalla de inicio de sesión:

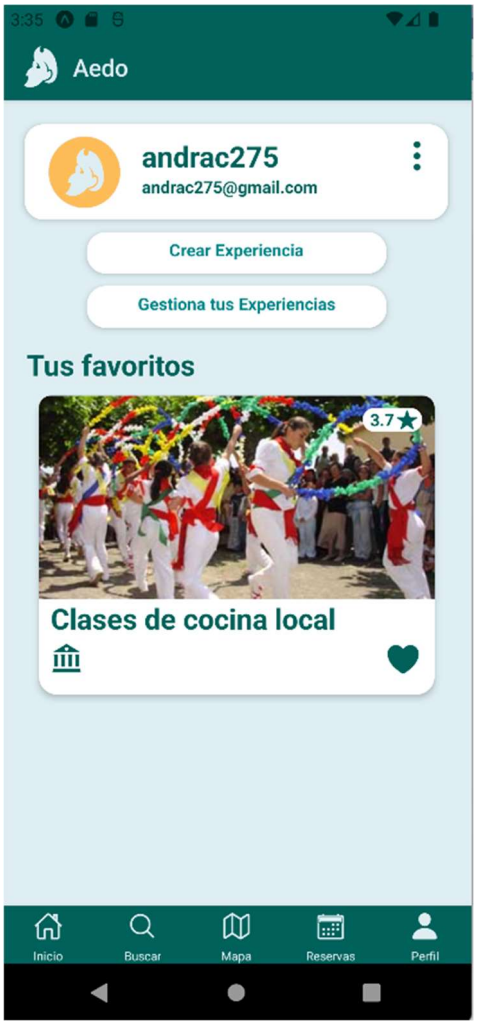
Interfaz para iniciar sesión con el correo electrónico y la contraseña. Además, permite registrarse o restablecer la contraseña.




Pantalla de registro:

Incluye el formulario con los datos necesarios para realizar el registro en la aplicación. Los campos marcados con un asterisco son obligatorios.

Al registrarse se debe confirmar el correo electrónico.

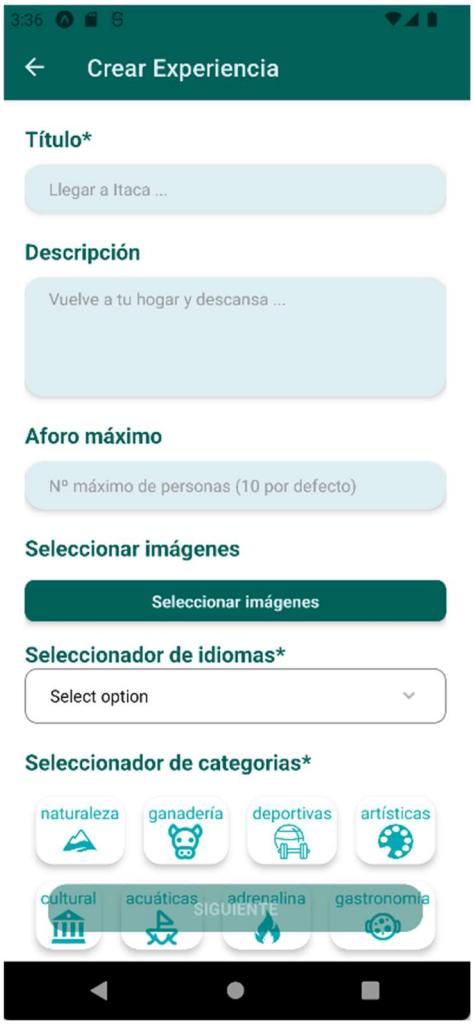



Pantalla de perfil de usuario:

La pestaña del perfil con el usuario autentificado se convierte en la pantalla de perfil, aquí se pueden crear experiencias o gestionarlas, además aparecen las experiencias favoritas del usuario y permite acceder a la modificación de datos personales, correo y contraseña.

Pantalla modificar perfil:

Esta interfaz se utiliza para modificar los datos del usuario facilitados durante el registro en la aplicación. Permite además modificar el correo electrónico y la contraseña.

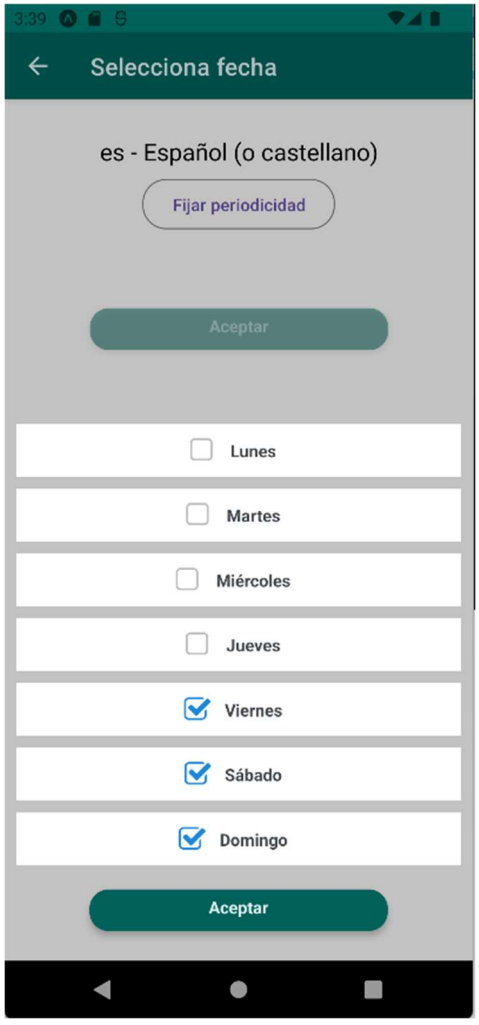


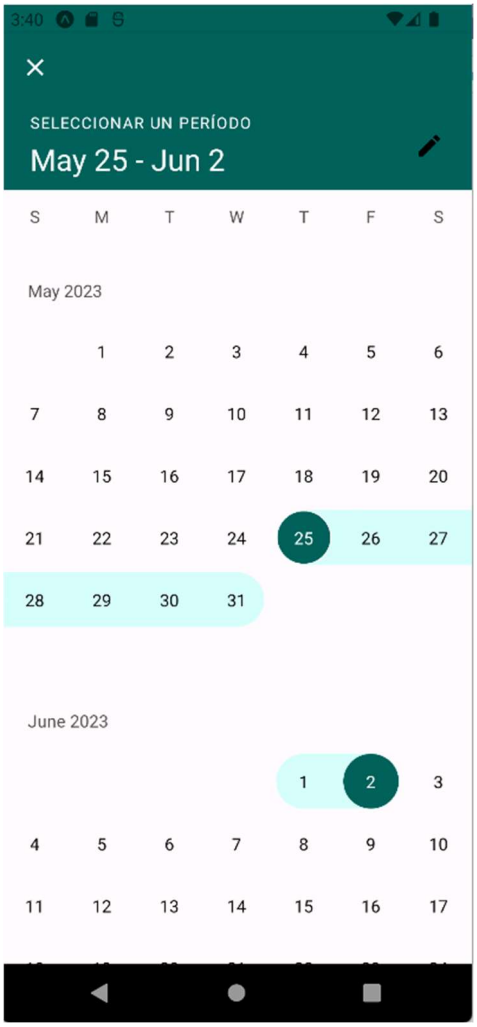

Pantalla crear experiencias:

Interfaz para crear experiencias en la aplicación. Permite introducir un título, una descripción, el aforo máximo, las imágenes, la categoría, el/los idioma/s y la localización.

Pantalla periodicidad:

La ventana anterior lleva a la selección de fechas o periodicidad de la experiencia por cada idioma.



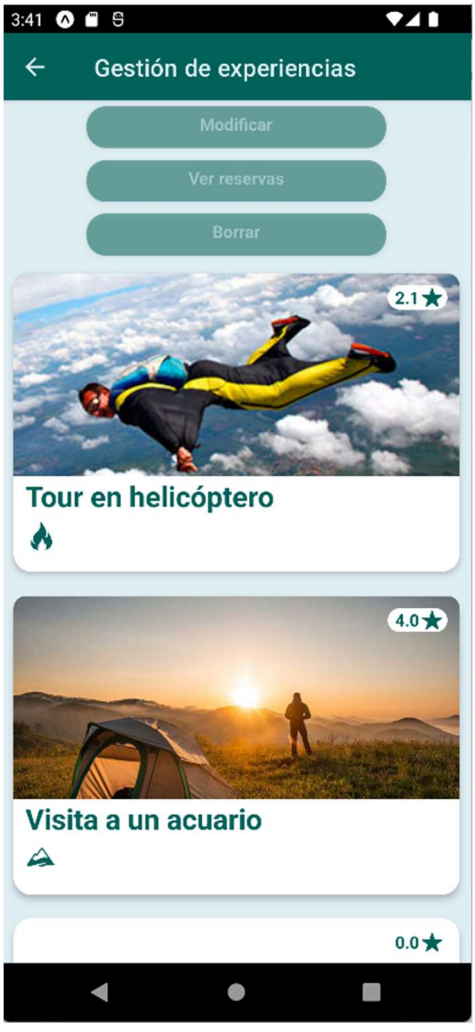



Pantalla selección de rango de fechas:

La pantalla de creación de experiencias lleva como paso final a la selección de fechas en las que se ofrece la experiencia.

Pantalla gestión de experiencias:

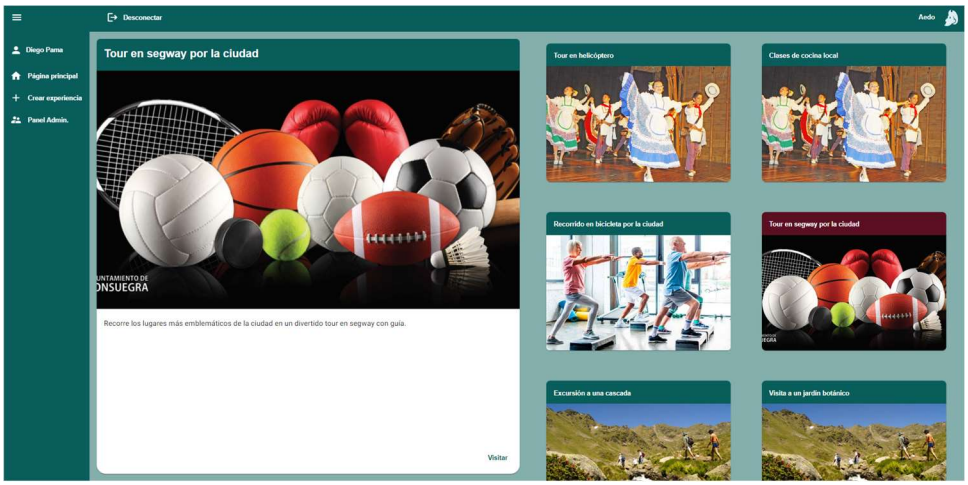
Aquí se presentan las distintas experiencias ofrecidas por el usuario registrado. Permite modificar la información, borrarlas o ver otros usuarios que se han apuntado y para qué fecha lo han hecho.



**Capturas de pantalla de la página web**

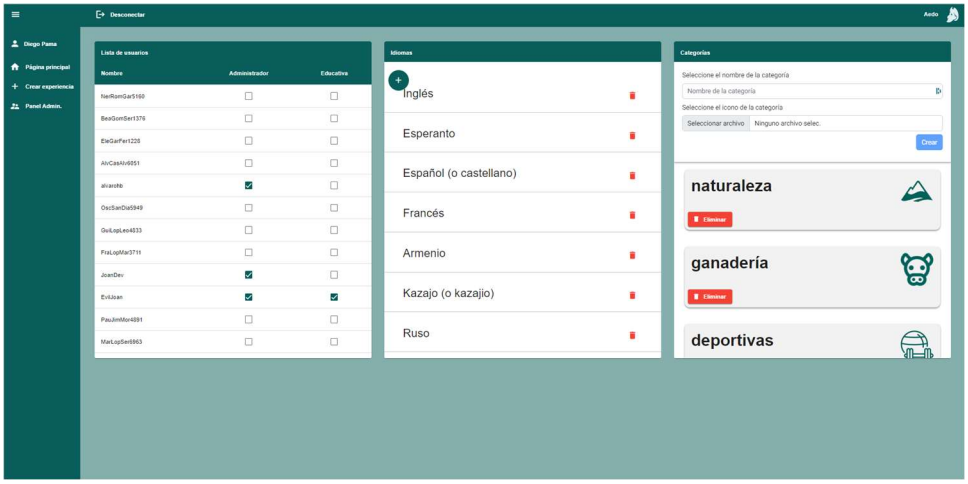
Pantalla principal:

Primera pantalla que ve el usuario al entrar a la web. En ella se muestran las distintas actividades que están disponibles y permite seleccionarlas para ver más información sobre ellas e ir a la página de reserva.



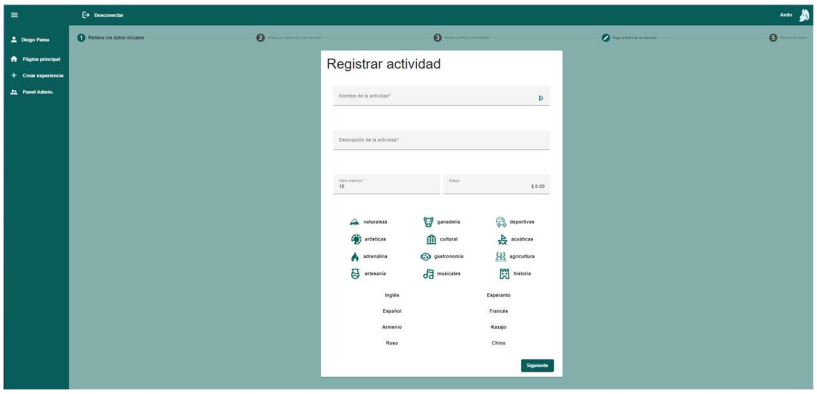

Pantalla de administración:

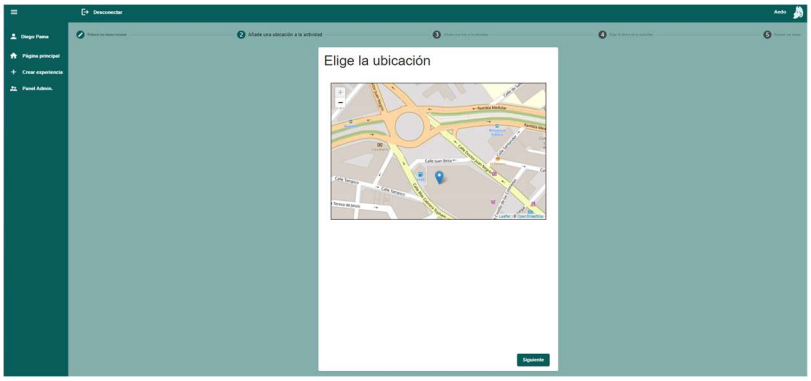
Pantalla disponible para los usuarios con el rol de administrador, en ella se muestran en distintas funcionalidades de gestión. No es accesible para otros tipos de usuarios.

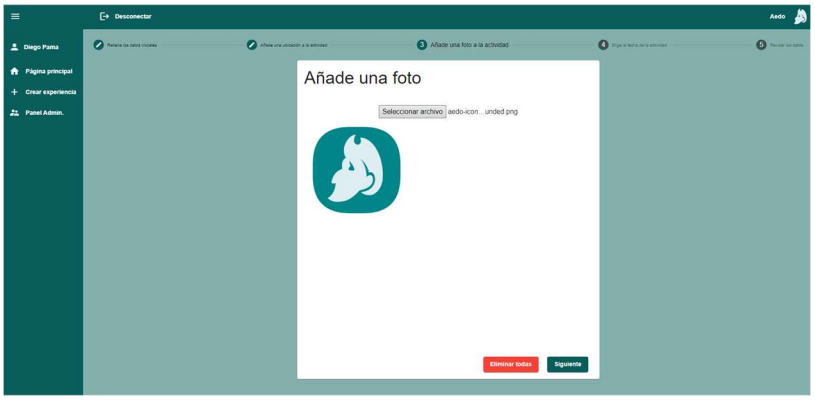


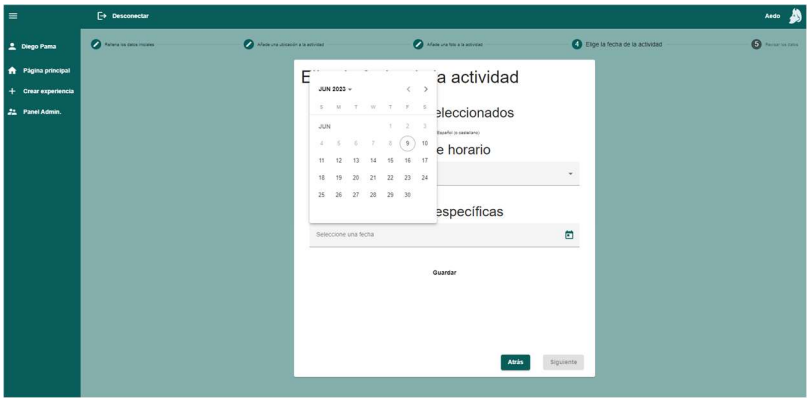

Pantalla de registro de actividades:

Pantalla disponible para los usuarios registrados. Permite introducir un título, una descripción, el aforo máximo, las imágenes, la categoría, el/los idioma/s y la localización.


Desconectar
Aedo
Diego Pama
Página principal
Crear experiencia
Panel Admin.
Registrar actividad
Siguiente

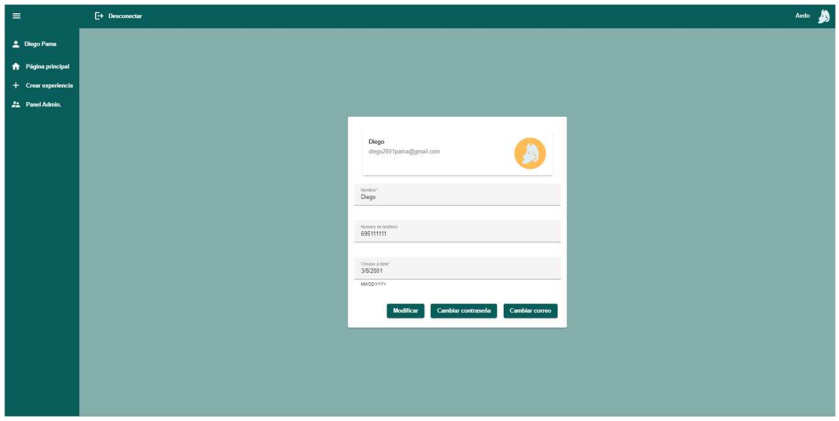

Desconectar
Aedo
Diego Pama
Página principal
Crear experiencia
Panel Admin.
Elige la ubicación
Siguiente


Desconectar
Aedo
Diego Pama
Página principal
Crear experiencia
Panel Admin.
Añade una foto
Seleccionar archivo
Eliminar todas
Siguiente


Desconectar
Aedo
Diego Pama
Página principal
Crear experiencia
Panel Admin.
Guardar
Atrás
Siguiente

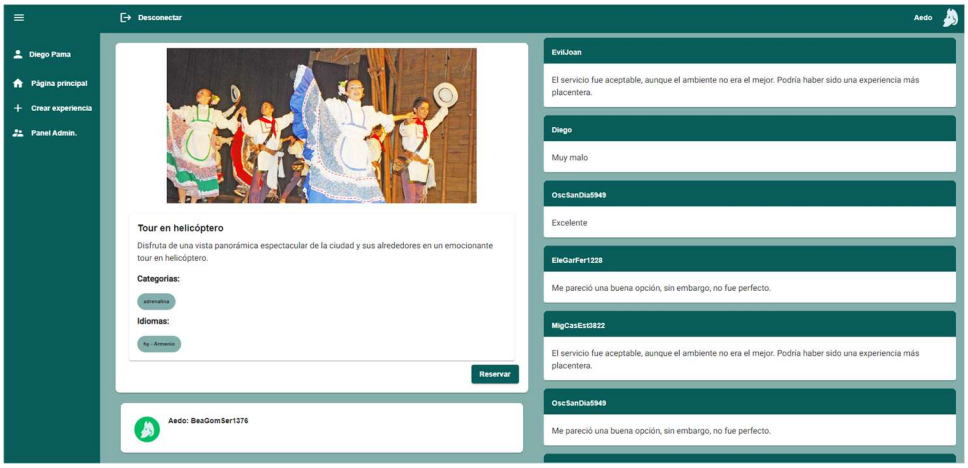
Pantalla de perfil del usuario:

Muestra la información del usuario que está conectado y a su vez permite la modificación de nombre del usuario, su número de teléfono, su fecha de nacimiento, contraseña y correo.



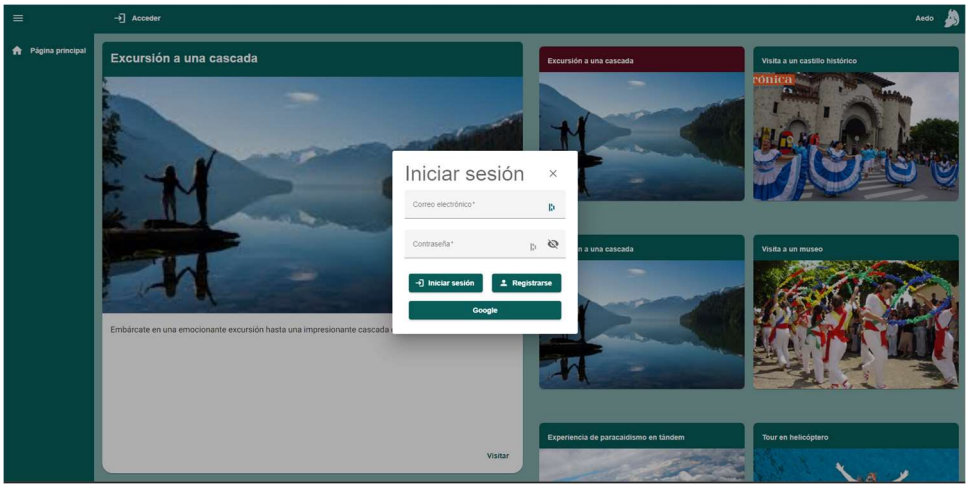

Pantalla de información de una experiencia:

Aparece la información de la experiencia, así como los comentarios dados por otros usuarios de la aplicación. Además, permite al usuario crear una reserva para dicha experiencia.



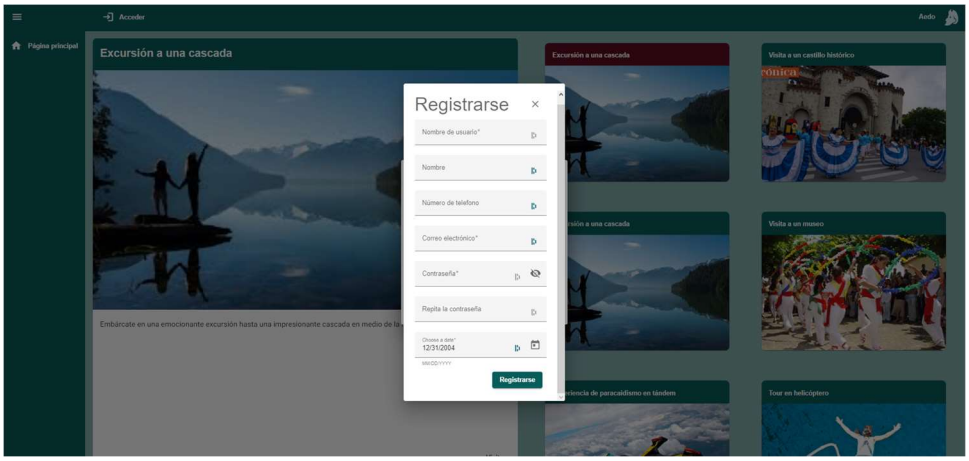
Pantalla de inicio de sesión:

Permite el inicio de sesión de usando la información de un usuario registrado y también da acceso a la pantalla de registro de un usuario.




Pantalla de registro:

Pide al usuario un nombre de usuario, su nombre real, su número de teléfono, correo electrónico, fecha de nacimiento y contraseña con el fin de registrar el usuario en la aplicación.



# Anexo II: Preguntas de la encuesta a los usuarios

Se realizaron dos encuestas distintas. Una encuesta era para un usuario/a que fuera a utilizar la aplicación y a realizar una experiencia simple (Experimento 2, ver sección "3.2.1 Experimento 2: Presentar la aplicación a un potencial creador de contenido"); la otra encuesta (Experimento 3, sección "3.2.3 Experimento 3: Presentar la aplicación a un profesor") iba enfocada al ámbito educativo, permitiendo reservar una experiencia para una clase entera de alumnos. A continuación, se exponen las preguntas generales para un usuario/a estándar que desee realizar una experiencia. La encuesta "educativa" incluye las preguntas generales y dos preguntas específicas.

**Preguntas generales:**

| **Numero** | Pregunta | Tipo de pregunta |
|---|---|---|
| **1** | Grado de satisfacción con la aplicación | Escala lineal de 1 a 5 |
| **2** | ¿Qué posibilidades hay de que recomiende Aedo a un conocido? | Escala lineal de 1 a 5 |
| **3** | ¿Considera que fue sencillo aprender a utilizar la aplicación? | Escala lineal de 1 a 5 |
| **4** | Considero que la aplicación me informa en cómo proceder cuando existe algún problema. | Escala lineal de 1 a 5 |
| **5** | ¿Considera que siempre ha sabido como proceder en lo que deseaba hacer? | Escala lineal de 1 a 5 |
| **6** | ¿Considera que la interfaz de la aplicación es placentera? | Escala lineal de 1 a 5 |
| **7** | ¿La navegación de la aplicación le pareció intuitiva? | Escala lineal de 1 a 5 |
| **8** | ¿Considera que la aplicación cumple con las características esperadas de una aplicación para tal fin? | Escala lineal de 1 a 5 |
| **9** | ¿La velocidad de respuesta de la aplicación le parece adecuada? | Escala lineal de 1 a 5 |
| **10** | ¿Qué le ha parecido el proceso de creación de experiencias? | Escala lineal de 1 a 5 |
| **11** | ¿Qué le ha parecido la administración de experiencias y de usuarios apuntados a las | Escala lineal de 1 a 5 |

| | | |
|---|---|---|
| | mismas? ¿Considera que la información es suficiente? | |
| **12** | ¿Sugerencias de la aplicación? | Abierta |

**Preguntas específicas de la encuesta educativa:**

| **Número** | Pregunta | Tipo de pregunta |
|---|---|---|
| **13** | ¿Qué le ha parecido el proceso de inscripción a experiencias Educativas? | Escala lineal de 1 a 5 |
| **14** | ¿Qué le ha parecido la administración de experiencias educativas y de usuarios apuntados a las mismas? ¿Considera que la información es suficiente? | Escala lineal de 1 a 5 |

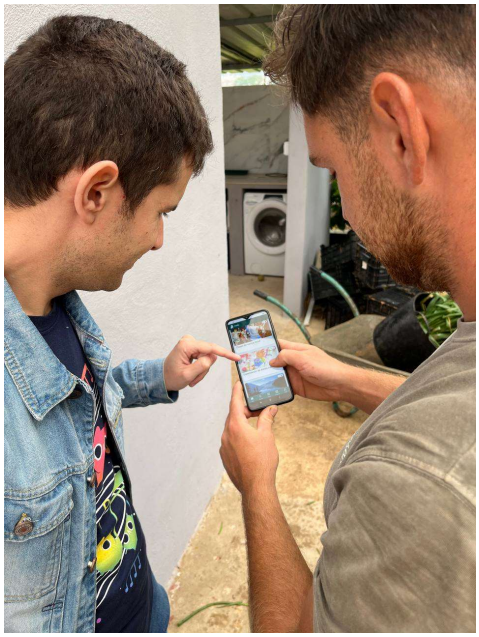
| Marca temporal | Pregunta 1 | Pregunta 2 | Pregunta 3 | Pregunta 4 | Pregunta 5 | Pregunta 6 | Pregunta 7 | Pregunta 8 | Pregunta 9 | Pregunta 10 | Pregunta 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5/27/2023 18:12:18 | 5 | 5 | 5 | 4 | 4 | 5 | 4 | 4 | 4 | 5 | 4 |
| 6/10/2023 14:58:26 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6/12/2023 8:42:34 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 4 | 5 | 5 | 5 |
| 6/12/2023 8:43:20 | 5 | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6/12/2023 8:44:06 | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6/12/2023 16:32:29 | 4 | 4 | 5 | 3 | 4 | 5 | 5 | 4 | 4 | 5 | 4 |
| 6/12/2023 16:43:49 | 5 | 5 | 5 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| 14/06/2023 12:38 | 4 | 4 | 5 | 4 | 4 | 5 | 5 | 4 | 3 | 5 | 5 |
| 14/06/2023 14:02 | 4 | 4 | 5 | 4 | 3 | 5 | 4 | 4 | 4 | 4 | 5 |
| 14/06/2023 15:02 | 5 | 5 | 4 | 5 | 5 | 5 | 4 | 5 | 5 | 5 | 4 |
| 14/06/2023 19:00 | 4 | 4 | 4 | 4 | 4 | 5 | 4 | 5 | 5 | 5 | 4 |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| Enunciados | satisfacció | posibilidad | a que fue | que la | a que | a que la | navegación | a que la | velocidad | parecido el | parecido la |
| sobre 10 | 9,09 | 9,09 | 9,45 | 8,36 | 8,73 | 10,00 | 9,27 | 9,09 | 9,09 | 9,82 | 9,27 |
| sobre 5 | 4,55 | 4,55 | 4,73 | 4,18 | 4,36 | 5,00 | 4,64 | 4,55 | 4,55 | 4,91 | 4,64 |
| numEncuestados | 11 | | | | | | | | | | |

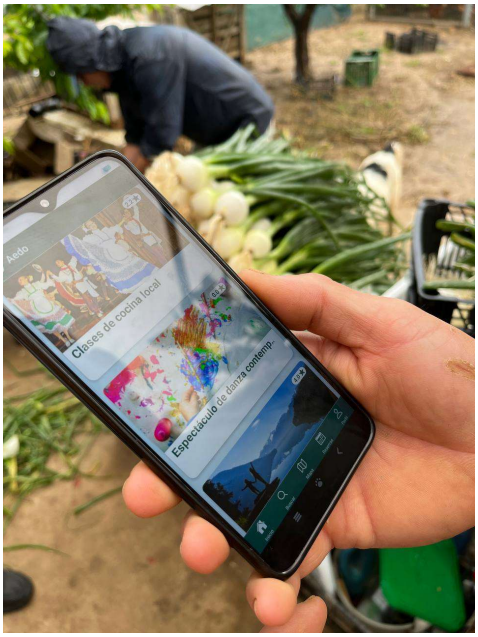
La figura anterior muestra las respuestas respondidas por los once encuestados y las puntuaciones medias. A través del presente enlace, se puede consultar los datos de la encuesta y los enunciados asociados a cada pregunta.

# Anexo III: Experimentos realizados

Se realizaron tres experimentos con la aplicación. El primer experimento se realizó durante la Feria de Proyectos de la ETSINF Se expuso el primer *MVP* de la aplicación. Durante la experiencia en la feria se extrajo mucha información útil y feedback por parte de los asistentes; ya fueran alumnos, profesores o empresas.


El segundo experimento se realizó fuera del recinto universitario, directamente en Játiva con un agricultor que mostró interés en la aplicación. Se visitó el campo de trabajo y nos explicó que la aplicación le parecería útil para enseñar a los usuarios/as de qué manera se trabajan las hortalizas.

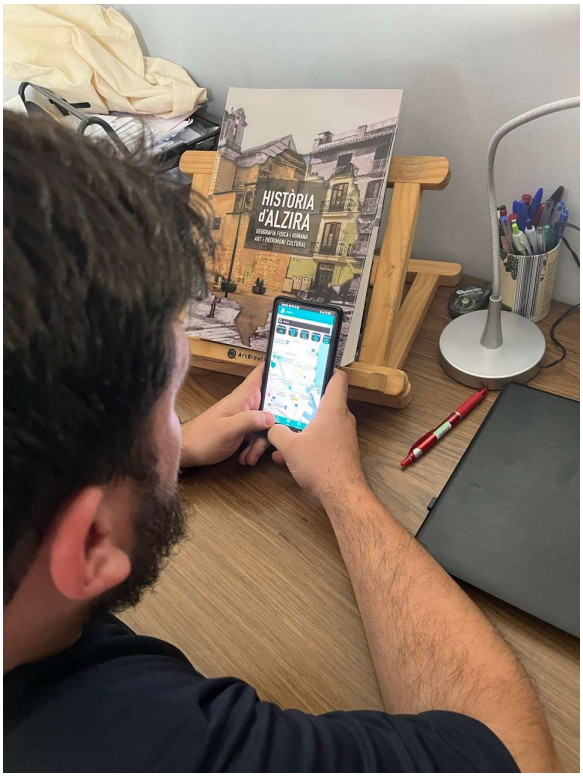





Tras la visita del campo, se nos acompañó a la tienda del barrio donde el agricultor comercializa con el producto kilómetro cero directamente desde el campo.



Para el tercer, y último, experimento se contactó con un profesor de historia y se le planteó la posibilidad de utilizar la aplicación para poder inscribir a su clase de alumnos a las experiencias ofrecidas en la plataforma.



# Anexo IV: Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

| Objetivos de Desarrollo Sostenibles | Alto | Medio | Bajo | No Procede |
|---|---|---|---|---|
| ODS 1. **Fin de la pobreza.** | | | | X |
| ODS 2. **Hambre cero.** | | | | X |
| ODS 3. **Salud y bienestar.** | X | | | |
| ODS 4. **Educación de calidad.** | | X | | |
| ODS 5. **Igualdad de género.** | X | | | |
| ODS 6. **Agua limpia y saneamiento.** | | | | X |
| ODS 7. **Energía asequible y no contaminante.** | | | | X |
| ODS 8. **Trabajo decente y crecimiento económico.** | | | | X |
| ODS 9. **Industria, innovación e infraestructuras.** | | | | X |
| ODS 10. **Reducción de las desigualdades**. | | | X | |
| ODS 11. **Ciudades y comunidades sostenibles**. | X | | | |
| ODS 12. **Producción y consumo responsables.** | | | | X |
| ODS 13. **Acción por el clima.** | | | X | |
| ODS 14. **Vida submarina.** | | | X | |
| ODS 15. **Vida de ecosistemas terrestres.** | | | X | |
| ODS 16. **Paz, justicia e instituciones sólidas.** | | | | X |
| ODS 17. **Alianzas para lograr objetivos.** | | | | X |

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

La aplicación Aedo está ampliamente alineada con tres importantes Objetivos de Desarrollo Sostenible de las Naciones Unidas: Salud y bienestar (Objetivo 3), Igualdad de género (Objetivo 5) y Ciudades y comunidades sostenibles (Objetivo 11).

- Objetivo 3: Salud y bienestar: Aedo fomenta el compartir experiencias entre usuarios/as, muchas de las experiencias son al aire libre y repercuten en un incremento de la salud, debido a la motivación por parte del equipo de Aedo en incentivar el movimiento de personas a sitios completamente distintos a los que uno puede estar acostumbrado.

- Objetivo 5: Igualdad de género: Aedo no solo permite que cualquier usuario/as, sin importar su género, se convierta en mentor de experiencias, sino que también promueve activamente la

igualdad de género en todas las áreas. La aplicación destaca y promociona experiencias y logros realizados tanto por hombres como por mujeres, brindando visibilidad a personas de todos los géneros. Es por lo que Aedo favorece la igualdad de género. Gracias a la aplicación se pueden compartir, por ejemplo, trabajos realizados tanto por hombres como por mujeres. Además, asociaciones de mujeres o relacionados con la igualdad de género de los diferentes pueblos pueden hacer uso de Aedo para potenciar y publicitar sus actividades culturales.

- Objetivo 11: Ciudades y comunidades sostenibles: La aplicación Aedo tiene como objetivo primordial fomentar la descentralización del turismo en las grandes ciudades y trasladarlo a zonas menos conocidas, pero con la misma capacidad de generar interés en los turistas. Teniendo en cuenta esa premisa, Aedo "obliga" a los usuarios/as consumidores de experiencias a desplazarse. Muchas veces la ubicación de las experiencias serán localidades de la España vacía, ayudando a la economía y las personas locales de dichas poblaciones. Aedo se esfuerza por no solo descentralizar el turismo, sino también por promover la sostenibilidad y el desarrollo equilibrado de las comunidades locales. Además, Aedo fomenta la colaboración con organizaciones locales y pequeñas empresas para ofrecer experiencias auténticas y sostenibles a los usuarios/as. De esta manera, Aedo ayuda a evitar la sobrecarga turística en áreas urbanas y fomenta un turismo más responsable y consciente, que a su vez contribuye a la preservación del patrimonio cultural y natural de las comunidades visitadas.

La aplicación puede estar en menor medida relacionada con los objetivos 4 (Educación de calidad), 10 (reducción de las desigualdades), 13 (Acción por el clima), 14 (Vida submarina) y 15 (Vida de ecosistemas terrestres). La razón de este resquicio reside en que se cumplan o no los objetivos recae en el uso que dan los usuarios/as a la aplicación; por ejemplo: si un usuario/a promueve acciones por el clima, entonces la aplicación se relaciona con el ODS número 13.

En conclusión, la aplicación Aedo está alineada con los Objetivos de Desarrollo Sostenible de las Naciones Unidas al promover la salud y el bienestar a través del intercambio de experiencias al aire libre, fomentar la igualdad de género al permitir que todos los usuarios/as compartan sus trabajos y logros, y contribuir a ciudades y comunidades sostenibles al descentralizar el turismo y apoyar el desarrollo equilibrado y sostenible en áreas menos conocidas. Además, aunque en menor medida, la aplicación Aedo puede tener relación con otros Objetivos de Desarrollo Sostenible, como la educación de calidad, la reducción de las desigualdades, la acción por el clima, la protección de la vida submarina y la conservación de los ecosistemas terrestres, dependiendo del uso que los usuarios/as hagan de la plataforma y de las acciones que promuevan a través de ella.