Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

**UNIVERSIDAD DE JAÉN**

*Escuela Politécnica Superior de Jaén*

Trabajo Fin de Grado

# Aplicación deportiva multiplataforma y serverless para la gestión de un negocio real

**Alumno: Ignacio Casale Linde**

Tutor: José Ramón Balsas Almagro
Dpto: Informática

**Septiembre, 2022**

# Índice

# Índice de Ilustraciones

# Índice de Tablas

# AGRADECIMIENTOS

A mi familia por su apoyo constante, sus consejos y sugerencias.

A mi tutor, José Ramón Balsas Almagro, por todas sus correcciones y ayuda.

A todos los profesores que me han impartido clase, por abrirme las puertas del aprendizaje.

# 1. INTRODUCCIÓN

Como primera toma de contacto con el proyecto se busca que se comprenda el contexto en el que se encuentra. Para ello, en el siguiente apartado veremos la motivación detrás de este trabajo, y a continuación los objetivos y metodología que se ha decidido seguir. Para finalizar esta sección se comentará la estructura del documento, donde se podrá entender de forma general cada apartado del documento y su utilidad.

La intención de este trabajo es realizar el estudio y desarrollo de una aplicación basada en las tecnologías web. Se pretende desarrollar una aplicación multiplataforma para la digitalización de un negocio real de clases deportivas.

## 1.1. Motivación

Como se ha comentado anteriormente, la idea de este proyecto es tener una utilidad para un negocio real que desde el confinamiento por COVID-19 empezó a realizar las clases deportivas de forma online. El negocio de esta persona, que a partir de ahora nos referiremos a ella como la cliente, ha ido cambiando y adaptándose con el tiempo. Su funcionamiento actual consiste en la impartición de clases deportivas por parte de la cliente a las alumnas que deciden matricularse. En sus clases sólo admite chicas e intenta generar un ambiente de colaboración y comodidad. En siguientes apartados se analizará su negocio con más detalle.

Cabe destacar que su negocio ha ido creciendo más cada año desde que imparte las clases online. Gracias a esto también ha podido subcontratar servicios y consejería de marketing, que han hecho que la forma de mostrar e impartir las clases vaya cambiando, para que ella tenga menos carga de trabajo sin perder la calidad inicial. Esto es importante mencionarlo, ya que desde ese momento tiene un objetivo muy claro: alcanzar un mayor número de usuarios. Necesita más alumnas matriculadas para así generar más ingresos y poder permitirse ampliar más su negocio, contratar a

personal administrativo, o incluso otros profesionales que impartan las clases también. Y de esta forma, continuar disminuyendo su carga de trabajo y aumentando sus ingresos.

Por estos motivos, ahora necesita una aplicación que le facilite el control de sus alumnas y clases. Además a sus alumnas les puede venir bien un lugar unificado para acceder a los diferentes enlaces de las clases, ya que actualmente se está haciendo por correo electrónico o Whatsapp. En resumen, tener una página web propia puede mejorar la fuerza de su marca, dar más calidad a su negocio, conseguir aumentar el número de alumnas matriculadas y quitarle o faciliarle cierta carga de trabajo gracias a la digitalización.

## 1.2. Objetivos

Para el desarrollo de este proyecto se han establecido los siguiente objetivos:

1. **Realizar el análisis del sistema para que cumpla con todos los requisitos marcados por el cliente de la forma más eficiente:** El negocio del cliente está en continuo crecimiento y cambio, por lo que será necesario adaptarse a ello. De forma general los puntos más importante son:
    - Sistema de control de acceso.
    - Gestión de las alumnas y de las clases por parte de los administradores de la aplicación.
    - Sistema de clases para las alumnas, donde puedan ver la información de estas.
    -
2. **Realizar una aplicación real que pueda ser utilizada por varios usuarios:** No se trata de un prototipo, debe crearse una aplicación completa conteniendo el mínimo número de fallos. Debe de poder utilizarse por numerosos usuarios de forma simultánea y tener una experiencia de uso satisfactoria para que no afecte negativamente al negocio de la cliente.

3. **Desarrollar el Front-end basado en tecnologías web teniendo en cuenta la marca personal del cliente. La aplicación debe ser fácil de utilizar para todo tipo de perfiles de usuario:** El cliente tiene su propia marca, con su correspondiente tipografía y paleta de colores, la página debe de seguir su identidad. Además, debido a que sus alumnas son personas entre 30 y 60 años no todas tienen un gran conocimiento sobre el manejo del teléfono móvil y algunas tienen problemas de vista. Por ello, la aplicación debe de ser fácil de entender, leer y utilizar.

4. **Preparar y gestionar un servidor para el despliegue de la aplicación:** La aplicación debe de ser accesible desde el navegador, también debe de poder descargarse para el móvil independientemente de su sistema operativo. Preferiblemente que sea posible descargarla desde la Play Store o la App Store. De esta manera se está aportando mucha comodidad a las personas que quieran probar el servicio de la cliente. Favoreciendo así la captación de potenciales alumnas para la cliente.

## 1.3 Metodología

El tipo de metodología que se va a aplicar en este proyecto es una metodología ágil **[1]**, la cuál tiene un enfoque iterativo donde se prioriza la entrega de pequeñas porciones de trabajo en vez de lanzamientos de gran envergadura. En este apartado se va a discutir dos de las metodologías ágiles más utilizadas actualmente: *scrum y kanban.* Posteriormente se va a explicar la forma en la que este proyecto se va a organizar.

Cuando se habla de metodología ágil, lo que primero se nos viene a la cabeza es *scrum*, ya que es la metodología ágil más utilizada por las empresas. Sin embargo, no hay que tomar *scrum* como sinónimo de metodología ágil. *Scrum* **[2]** es solo un marco de trabajo que propone unas directrices a seguir que siguen los principios de la metodología ágil. También es común comparar *scrum* y *kanban* para determinar cuál es mejor y cuándo usar una u otra. Lo cierto es que ambas metodologías son compatibles entre sí. Aplicando una metodología normal de *scrum* se puede aplicar *kanban* para dar

mayor flexibilidad o aumentar la calidad, colaboración, y compañerismo. Aunque también existe una metodología llamada *scrumban* que combina características de ambas.

Para comprender mejor qué marco de trabajo se adapta a este proyecto y es necesario conocer su funcionamiento, como se organizan y cómo trabajan con el flujo de trabajo.

### 1.3.1 Scrum

Es la metodología más popular actualmente y toma más importancia en la gestión de grandes equipos con roles diferenciados. También por ello, es la más rígida de todas. Produce muy buenos resultados gracias a la completa organización que emplea. Es más eficaz cuando existe un propietario de producto que tiene sus objetivos claros o que conoce lo que esperan recibir sus clientes en el tiempo.

El elemento clave son los *sprints* **[3]**, interacciones de 1 o 2 semanas donde se trabaja en un objetivo específico previamente definido en la planificación de *sprints*. Como se puede observar en la **Ilustración 1**, el ciclo de vida de un *sprint* está compuesto de varias partes. En ellas deben de intervenir el propietario del producto, el equipo de desarrollo y el maestro de Scrum **[4]**, quien se asegura de que se sigue el marco de *scrum*. Realizar un buen *sprint* requiere de flexibilidad, de una adecuada colaboración del equipo para estimar qué tareas se van a realizar y de revisiones periódicas.

Del párrafo anterior, aunque solo describe brevemente una parte de *Scrum*, se puede ver que es un marco de trabajo completo y muy organizado, donde el descontrol o fallos no tienen cabida. Sin embargo, sí tiene algunos aspectos negativos:

- Son necesarios varios roles para su funcionamiento, la ausencia de algunos de estos puede frenar el avance.
- Una mala estimación de las tareas a realizar en un *sprint* puede desmotivar al equipo al no cumplir el objetivo propuesto en el tiempo establecido.

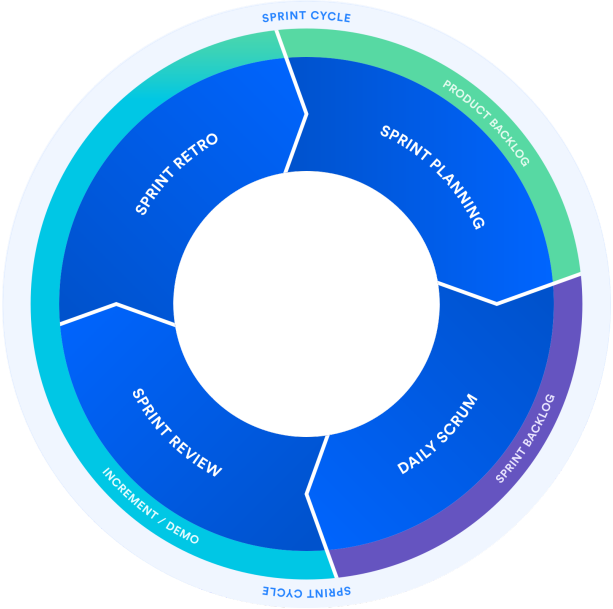



***Ilustración 1: Ciclo de vida de un spring.***[1]

## 1.3.2 Kanban

Según nos explica David J. Anderson **[5]** Se trata de un sistema de arrastre donde a cada trabajo que va a empezar se le asigna una tarjeta. Esta tarjeta se desvincula cuando ese trabajo termina. Hay tarjetas limitadas y un nuevo trabajo solo puede iniciarse cuando una tarjeta está disponible.

Este principio es aplicado en el desarrollo de software para limitar el trabajo en progreso. En *Kanban* inicialmente se crea el *backlog* **[6]**, posteriormente en una rápida reunión o de forma autosuficiente se eligen cuáles de esas tareas se van a realizar. El número de tareas en progreso debe de

[1] Fuente: https://www.atlassian.com/es/agile

estar limitado para que se pueda trabajar con holgura y exista la posibilidad de mejora. Si se ha alcanzado el límite, solo se podrá empezar una nueva tarea cuando se haya terminado una.

Para organizar el trabajo se emplea un tablero, **Ilustración 2**, que puede ser virtual o físico. En este tablero se establece el flujo de trabajo y se divide en diferentes columnas según se necesite. Además para hacerlo más visual cada tarea puede tener un color diferente para ver su prioridad.

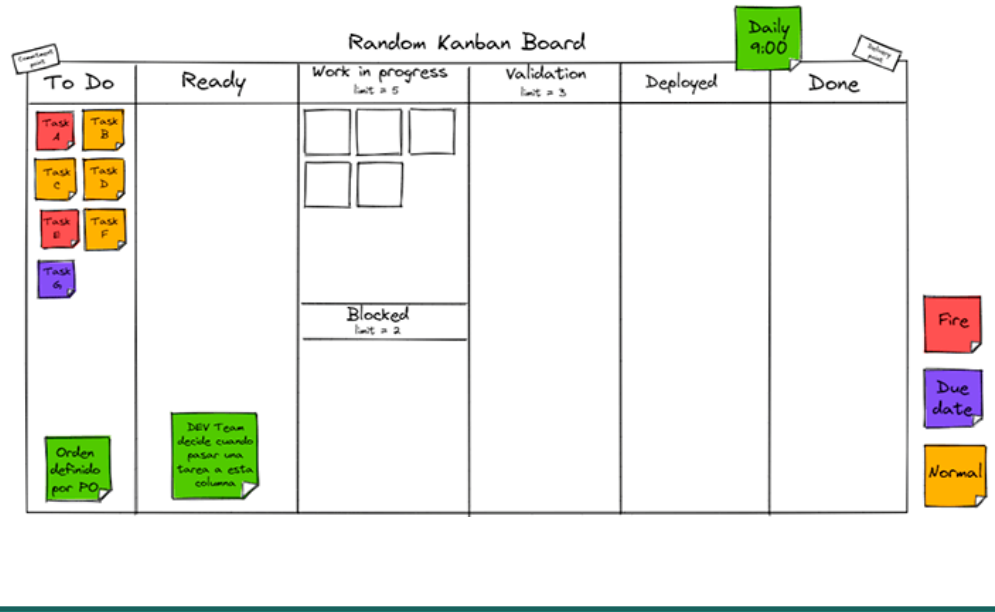



***Ilustración 2: tablero kanban.***[2]

**¿Por qué usar un sistema kanban?**

- Sigue la cultura *Kaizen,* donde los individuos se sienten libres para tomar acción sin temor. Tienen mucha libertad para auto-gestionarse, y se crea un clima de respeto y colaboración.
- Se puede aplicar a cualquier sistema sin importar su tamaño y sin cambiar la estructura jerárquica y los roles. Aunque también fomenta una jerarquía más plana, eliminando roles que no son necesarios y ganando así mayor comunicación y rapidez.

[2] Fuente: https://profile.es/blog/como-hacer-tablero-kanban-ejemplo/

- Iteraciones continuas de entrega, donde se evita la presión, los cuellos de botella y se pierde menos tiempo en las estimaciones para emplearse en el desarrollo.
- Tiene la receta para el éxito:
    - Enfocarse en la calidad.
    - Reducir el trabajo en progreso.
    - Entregar a menudo.
    - Equilibrar la demanda contra la entrega.
    - Priorizar.
    - Atacar las fuentes de variabilidad para mejorar la previsibilidad.

### 1.3.3 Conclusión

Tras esta comparación es fácil ver porqué este proyecto usará el modelo *kanban*. Dos aspectos muy positivos es que es muy flexible (necesario para adaptarse a los posibles cambios en el negocio de la cliente) y que es posible aplicarlo siendo una sola persona (*scrum* necesita varios roles para su completa aplicación).

En este proyecto, para el análisis de requisitos se va a utilizar historias de usuario y su correspondiente descomposición en tareas. Como es conveniente priorizar las tareas según su importancia, se van a dividir en las diferentes categorías que propone el método *MosCoW*.

Se va a buscar implementar todas las tareas con mayor prioridad y aquellas no tan críticas pero que sí deberían incluirse, ya que la intención es entregar una aplicación completa y funcional.

Se hará uso de un modelo incremental y se establecerán solo 4 columnas iniciales. Debido a la sencillez de ser una sola persona no veo necesario hacer más divisiones. Estas columnas serán:

- Backlog: Donde se colocarán las tareas pendientes de realizar.
- En curso: Aquí se encontrarán las tareas que se estén realizando.
- En revisión: Después de realizar una tarea, puede pasar a esta columna si se piensa que es necesario revisarla para buscar errores o mejorar la calidad del código.

- Terminadas: Aquí estarán todas las tareas que se hayan terminado. Una tarea de esta columna puede pasar a la columna "En revisión" si posteriormente se detecta algún fallo.

Uno de los aspectos fundamentales de *kanban* es limitar el trabajo en progreso, en este caso se van a definir las columnas "En curso" y "En revisión" como el trabajo en progreso, con un límite de dos tareas a la vez. De esta forma, podría haber una tarea en curso y otra en revisión, pero nunca dos en curso y una en revisión, por ejemplo. Pienso que este límite me permitirá trabajar sin saturación pero pudiendo trabajar simultáneamente si fuesen necesarios pequeños cambios en alguna tarea diferente. Además, esto también permitiría tener dos tareas a la vez en curso. Puede parecer contra-intuitivo que una persona realice dos tareas a la vez. Sin embargo, si una tarea está relacionada con el *back-end* y la otra con el *front-end* y pertenecen a la misma página, sería posible o incluso necesario, avanzar en ambas. En la siguiente tabla, **Tabla 1**, se puede ver una idea de esta organización.

| **Backlog** | **Trabajo en progreso (2/2)** | | **Terminado** |
|---|---|---|---|
| | **En curso** | **En revisión** | |
| Tarea 5<br>Tarea 6<br>Tarea 7<br>… | Tarea 4 | Tarea 2 | Tarea 1<br>Tarea 3 |

***Tabla 1: Ejemplo del tablero kanban*** [3]

[3] Fuente: elaboración propia.

Para terminar este apartado, mencionar que existen numerosos tableros virtuales como *Trello*[4] o *Jira*[5]. Este último es mucho más completo y tiene más funciones. Sin embargo, se usará el primero mencionado debido a que es más simple y suficiente para este proyecto.

## 1.4 Estructura del documento

En este apartado comentaremos la estructura de este documento. Cabe mencionar, que para facilitar su lectura, todos los índices, referencias bibliográficas, menciones a las ilustraciones y a las tablas, se han preparado para que se pueda hacer clic sobre ellas y avances al lugar correspondiente. Además, se han marcado con este **color**, para que se identifiquen fácilmente.

La estructura del documento es la siguiente:

**Apartado 2: análisis.**

Se empezará realizando un análisis preliminar donde se entenderá el contexto del negocio de la cliente y la necesidad de este proyecto. A continuación, se realizará un análisis de las tecnologías actuales y se seleccionará la que mejor para este caso. También se hará un estudio de aplicaciones similares en el mercado para tomar ideas. Por último, se verán las historias de usuario y su división de tareas, un estudio de viabilidad y el modelo de dominio.

**Apartado 3: diseño.**

En este apartado se podrá ver el diseño del sistema. Se explicarán varios diagramas como el diagrama de clases, de secuencia, etc. También se verá el diseño de la interfaz, un aspecto muy importante para una aplicación que espera tener un uso real. Por último se mostrará el plan de pruebas.

[4] Enlace: https://trello.com/

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

**Apartado 4: implementación.**

Este apartado inicia con una explicación de cómo instalar el framework elegido. Después, se describirán los aspectos más importantes del framework y su estructura. Por último, se mostrará también cómo se han realizado algunas de las características más relevantes, y cómo se ha realizado el despliegue de la aplicación.

**Apartado 5: resultados de las pruebas**

Se comentarán los resultados de las pruebas obtenidas según el plan de pruebas inicial.

**Apartado 6: conclusión**

Se comentará el estado final del proyecto, y los objetivos que no se han podido cumplir. También se dará una valoración personal de las dificultades encontradas y de lo aprendido.

**Para finalizar se podrá ver tres anexos:** vídeo general, material suministrado y notas adicionales. En el primero, simplemente se menciona que materiales se van a enviar junto a este documento. En el segundo, se mencionará como acceder como administrador o como usuario, para ver todas las secciones de la aplicación.

# 2. ANÁLISIS

## 2.1 Analisis preliminar

Para comenzar con este apartado se va a describir como es el negocio de la cliente. Actualmente cuenta con varias modalidades de clases y recientemente cambió su forma de impartirlas. Cada semana graba varias clases para que las alumnas accedan a ellas semanalmente, y solo imparte clases en directo dos días a la semana. Las alumnas ven las clases grabadas a través de enlaces privados de youtube. Estos enlaces los comparte previamente la cliente por Whatsapp y por correo electrónico.

En este momento, las alumnas pagan todas la misma mensualidad, independientemente de a cuántas clases asista. Esto les permite poder acceder a cualquier clase grabada o en directo que quieran. En sus directos actualmente no se superan las 20 alumnas, por lo que ahora mismo funciona bien así a pesar de no contar con un sistema de reserva.

Las alumnas no tienen ningún sitio donde poder ver las clases ofertadas de forma organizada. Es la cliente quien se encarga de comunicarlo por escrito y les explica y recomienda un tipo de clase u otro. Por ello es necesario un lugar donde las alumnas puedan ver fácilmente las clases que existen y su descripción para que así la clienta pueda dedicar más tiempo a otros asuntos. Además, así también las alumnas podrían acceder a los enlaces de las clases más fácilmente.

Por último mencionar que la cliente gestiona los pagos con las alumnas cada mes sin hacer uso de ningún software, por ello sería interesante que la aplicación se pudiese encargar de ello, así como tener un sistema de chat para eliminar completamente la comunicación por Whatsapp o para comunicar mensajes importantes.

## 2.2 Aplicaciones similares

En este apartado se va a poder ver el estudio de varias aplicaciones similares o que podrían servir como alternativas a la cliente. La finalidad de este apartado es comprender qué aplicaciones hay en el mercado, ver si alguna podría cumplir todos los requisitos de la cliente y si no, utilizar este análisis para extraer ideas de funcionalidad o de la interfaz.

### 2.2.1 Clupik

Se trata de una aplicación que permite tener tu propio espacio para tu club. Te da la posibilidad de tener una página y una aplicación web personalizada. Además de poder gestionar los pagos y ver

estadísticas. Como se puede ver en su página de Inicio, **Ilustración 3**, cuenta con un chat para ayudar a las personas que lo necesiten.

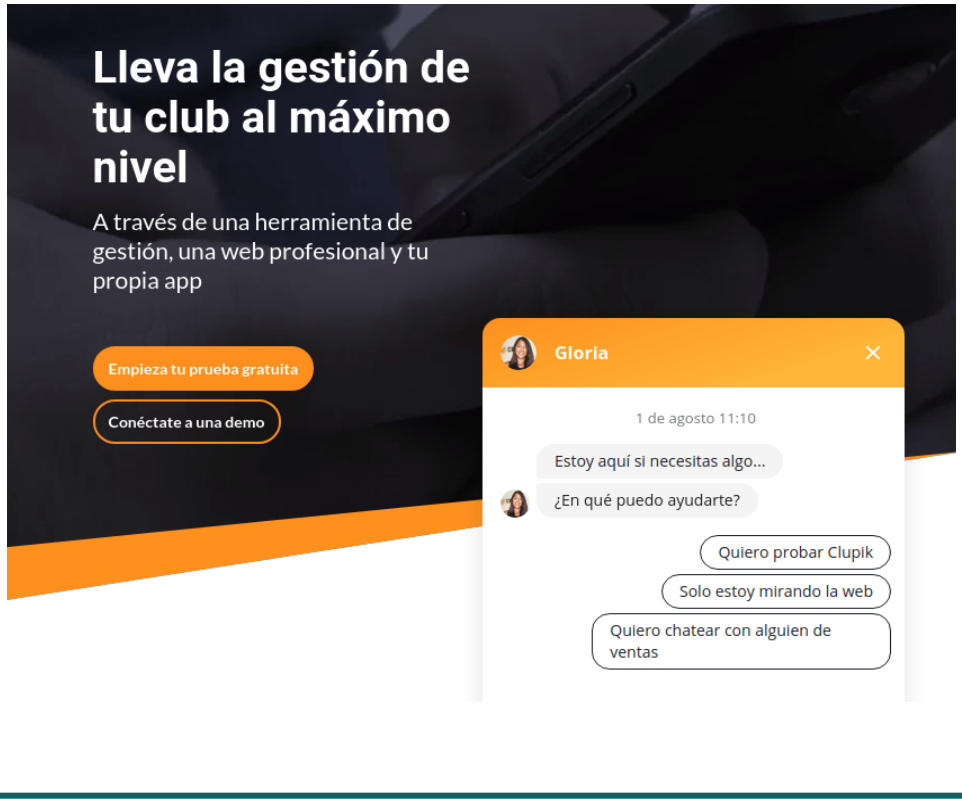



***Ilustración 3: página de Inicio en CLUPIK.[6]***

Es una página muy profesional, y completa. Más de 600 clubs la están usando actualmente. Cuenta con una interfaz muy bien diseñada que transmite profesionalidad, los colores elegidos también resaltan los elementos más importantes y tienen buena legibilidad. Aunque tiene algunos fallos menores, **Ilustración 4**, en la adaptación a dispositivos móviles, lo que resta confianza a crear tu "propia" web con ellos. Por ejemplo, el menú no tiene márgenes a la derecha, lo que esconde un poco los iconos que indican que hay elementos desplegables.

[6] Fuente: https://clupik.com/

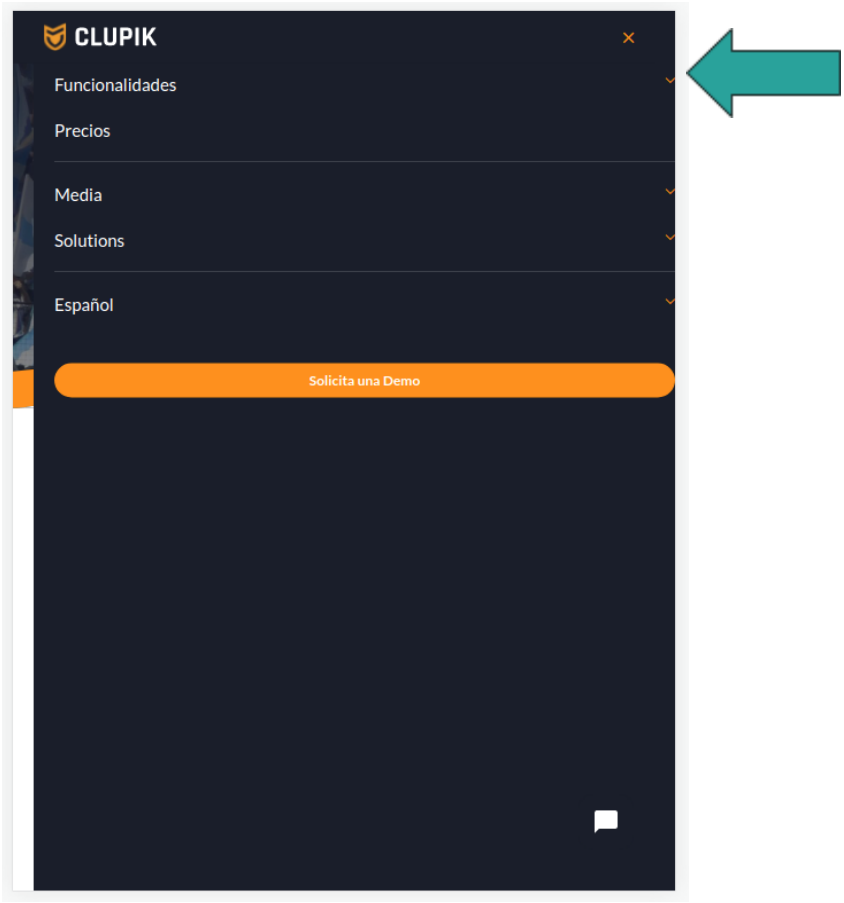



***Ilustración 4: fallo de interfaz en CLUPIK.***[7]

La forma de crear la aplicación web es a través de plantillas. Por lo que un usuario sin experiencia podría hacerlo pero no ofrece una completa personalización.

[7] Fuente: https://clupik.com/

### 2.2.1 Fitness Park

Esta vez no se trata de una aplicación para gestionar tu club (no hay muchas alternativas a la primera mencionada), si no de la página de un gimnasio. Se trata de una franquicia de gimnasios situados en varias ubicaciones. El ejemplo que se va a comentar aquí, está situado en Granada, en el centro comercial Nevada.

Cuenta con una interfaz impactante, **Ilustración 5**, donde el color amarillo predomina. Un color muy común en los gimnasios para enfatizar energía, vitalidad, y fuerza.

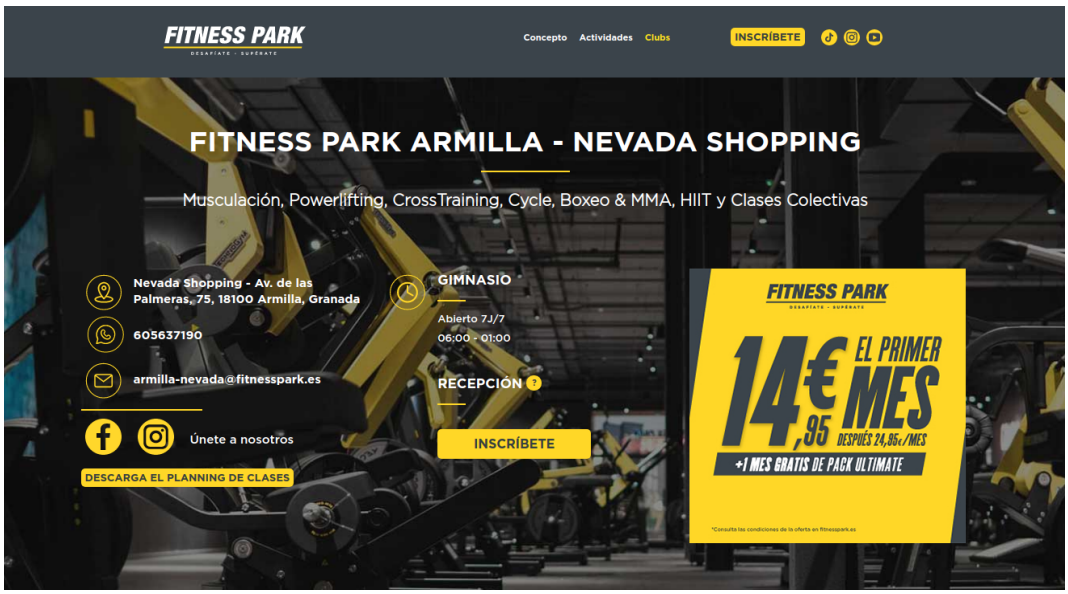



*Ilustración 5: página de Inicio Fitness Park*[8]

Aunque se pueden extraer algunas ideas del diseño de la interfaz, como la enorme cantidad de información que se muestra en la primera sección, lo cuál es útil para el usuario. No me parece la mejor

[8] Fuente: https://www.fitnesspark.es/

manera de estrucutrarla, ya que produce un efecto de mucha saturación. Además ya se pueden apreciar algunos fallos, como el *padding*[9] en algunos de los botones, que es muy escaso.

## 2.3 Justificación del desarrollo

La cliente no busca un software de terceros que pueda resolver sus necesidades. Está interesada en tener su propia aplicación por varios motivos:

- Reforzamiento de la marca personal.
- Libertad y control sobre todas las funcionalidades.
- Hacer cambios o añadir nuevas características en cualquier momento.

## 2.4 Selección de tecnologías

Hoy en día cada vez más personas realizan sus compras por internet, incluso si el negocio no vende ningún producto físico, es importante tener presencia en internet ya que da mayor seguridad a los usuarios, sirve como una carta de presentación actualizada, permite ser más competitivo y eficiente.

En este proyecto es importante la solución web para poder gestionar, controlar y analizar los datos de los usuarios y de las clases con mayor facilidad. También es necesario para que las alumnas tengan toda la información en un único lugar y puedan acceder fácilmente a ella. Además, tener una web y aplicación propia fortalece la marca personal del negocio.

### 2.4.1 Tipo de aplicaciones

Podemos diferenciar 3 tipos de aplicaciones **[7]** según su funcionamiento y desarrollo:

[9] ¿Qué es el padding? https://www.w3schools.com/css/css_padding.asp

- **Aplicaciones Nativas**

Las aplicaciones nativas se desarrollan específicamente para un sistema operativo. Se utilizan lenguajes como *Objective-C* o *Swift* para *iOS*, *Kotlin* para *Android* o *.Net* para *Windows Phone*. Se desarrollan tantas aplicaciones como sistemas operativos sean en los que se van a instalar. Lo más habitual es crear dos aplicaciones, una para *Android* y otra para *iOS*. De esta forma se consiguen crear aplicaciones adaptadas a cada sistema operativo, ofreciendo así una experiencia completa y con mejor rendimiento para los usuarios.

Sin embargo, también suelen ser las aplicaciones a las que más presupuesto **[8]** se tiene que dedicar debido al mayor trabajo que conlleva desarrollar una aplicación para cada sistema operativo.

- **Aplicaciones Web**

Se desarrollan normalmente con lenguaje *JavaScript*, junto con *HTML* y *CSS*. Son páginas web con apariencia de aplicación nativa. Cómo se accede desde el navegador su ejecución es posible en cualquier sistema operativo.

Son más fáciles y rápidas de desarrollar que las aplicaciones nativas, además un solo código fuente sirve para todos los sistemas operativos, por lo que se ahorra tiempo y costes. Sin embargo su rendimiento es menor **[9]**, necesitan conexión a internet para funcionar y no tienen acceso a las características hardware del dispositivo.

- **Aplicaciones Híbridas**

Combina los elementos de aplicaciones nativas y web **[10]**. Son aplicaciones web que se han colocado sobre un shell de una aplicación nativa. Debido a esta capa adicional pueden ser un poco más lentas que las aplicaciones nativas, sin embargo permite ejecutarse en cualquier dispositivo y acceder a las características hardware de este. Además, pueden no necesitar conexión a internet para funcionar.

Son rápidas de desarrollar, son más económicas que las apps nativas y su utilización reporta una mejor experiencia que una aplicación web. Algunos ejemplos actuales de aplicaciones híbridas son las versiones para dispositivos móviles de: *Facebook, Instagram, Whatsapp, Twitter, Uber*, etc.

- **Conclusión**

Si la aplicación a desarrollar requiere gran uso de los recursos hardware y necesita que sea lo más rápida posible, lo mejor es elegir una aplicación nativa. Aunque hay que tener en cuenta que hará falta tiempo y personal cualificado en varios sistemas operativos si se desea hacer la aplicación en varias plataformas.

Las aplicaciones web se suelen usar para optimizar y adaptar una página web a un dispositivo móvil, útiles para aplicaciones sencillas que no requieran funcionalidad del dispositivo.

Si no se tiene un presupuesto elevado o tiempo suficiente pero se necesita una aplicación con características nativas lo mejor sería optar por una aplicación híbrida. Actualmente el rendimiento de este tipo de aplicaciones puede ser parecido a las aplicaciones nativas y pueden acceder prácticamente a los mismos recursos hardware.

Puesto que en este Trabajo de Fin de Grado es desarrollado únicamente por una persona con un tiempo limitado, la opción más adecuada es usar una aplicación híbrida ya que de esta forma se puede tener rápidamente la aplicación en dispositivos *Android* e *iOS* con las ventajas comentadas anteriormente. Pero, para desarrollar este tipo de aplicaciones hay multitud de frameworks, a continuación veremos algunos de ellos.

### 2.4.2 Frameworks Front-end

Un framework para el front-end te ayuda a realizar el front-end de la aplicación **[11]**. Te facilita el uso de componentes ya creados, el uso de parámetros para cambiar rápidamente los estilos, el uso de preprocesadores de css, virtual DOM, modularidad de componentes, reactividad, aplicaciones SPA, etc.

Hay varios frameworks para el front-end, últimamente los más populares son:

- **React**

React, **Ilustración 6**, fue desarrollado por Facebook[10]. Ofrece interfaces interactivas de modo declarativo y simple, el uso de componentes, DOM virtual, compatibilidad con SEO, no impone patrones o plantillas, etc.

Es una librería para construir interfaces de usuario y tiene una arquitectura **[12]** MVC o MVVM según como se configure.

[10] Enlace: https://reactjs.org/

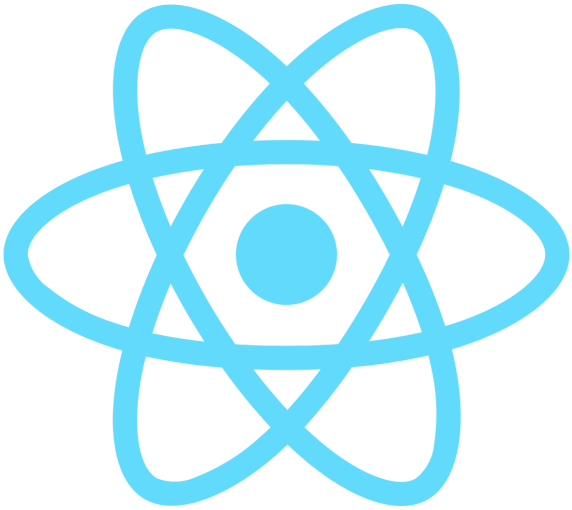


*Ilustración 6: logo React*[11]

- **Vue.js**

Vue, **Ilustración 7**, De código abierto con licencia MIT. Es un framework[12] modularizado. Cuenta con una curva de aprendizaje leve. Ofrece funcionalidades intuitivas, modernas y fáciles de usar. No impone qué tecnologías debes de usar. Comunicación entre componentes mediante eventos. DOM virtual, etc.

[11] Fuente: https://es.wikipedia.org/wiki/React

[12] Enlace: https://vuejs.org/

*Ilustración 7: logo Vue*[13]

## 2.4.3 Frameworks generales

- **Angular**

Angular[14], **Ilustración 9**, es de código abierto y desarrollado por *Google* para facilitar la creación y programación de aplicaciones web de una sola página.

Separa completamente el frontend y el backend en la aplicación, evita escribir código repetitivo y mantiene todo más ordenado gracias a su patrón MVC (Modelo-Vista-Controlador) asegurando los desarrollos con rapidez, a la vez que posibilita modificaciones y actualizaciones. El lenguaje principal de programación de Angular es Typescript.

[13] Fuente: https://es.m.wikipedia.org/wiki/Archivo:Vue.js_Logo_2.svg

[14] Enlace: https://angular.io

*Ilustración 9: logo Angular*[15]

- **React Native**

React Native, **Ilustración 10**, es un framework[16] de javascript para crear aplicaciones para *iOS* y *Android*. En lugar de desarrollar una aplicación web híbrida o en *HTML5*, lo que obtienes al final como resultado es una aplicación real nativa. Es uno de los frameworks más utilizados, y cuenta con una extensa comunidad.

[15] Fuente: https://es.m.wikipedia.org/wiki/Archivo:Angular_full_color_logo.svg

[16] Enlace: https://reactnative.dev

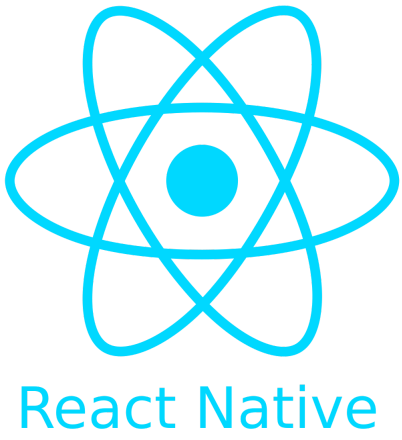



***Ilustración 10: logo React Native***[17]

- **Flutter**

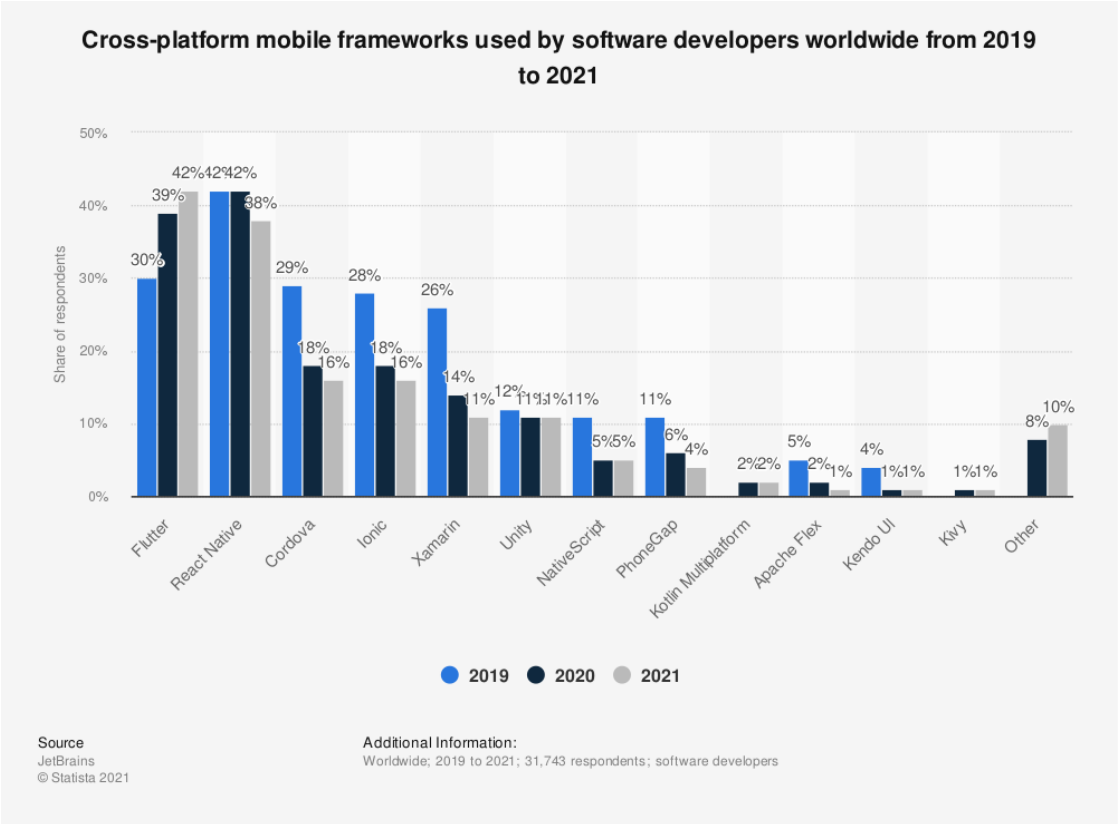
Flutter, **Ilustración 11**, es un framework[18] creado por *Google* que nos proporciona un conjunto de herramientas que tienen como finalidad el crear interfaces de software. Crea aplicaciones *iOS* y *Android*. Usa *Dart* como lenguaje de programación, tiene un rendimiento nativo gracias a este lenguaje. Es *OpenSource* y tiene una gran comunidad. En 2021 fue el framework más utilizado, **Ilustración 12**, por los desarrolladores.

[17] Fuente: https://github.com/aomine1745/react-native-udemy

[18] Enlace: https://flutter.dev


***Ilustración 11: logo Flutter*[19]**

Aunque tiene muy buenas características, tener que aprender su propio idioma de programación para usarlo, echa para atrás a muchos desarrolladores.




[19] Fuente: https://commons.wikimedia.org/wiki/File:Google-flutter-logo.svg

***Ilustración 12: frameworks más usados 2019-2021[20]***

- **Ionic**

Ionic, **Ilustración 13**, está basado en angular y de código abierto[21]. Nos permite desarrollar aplicaciones para *iOS* nativo, *Android* y la web, desde una única base de código. Su compatibilidad y, gracias a la implementación de *Cordova* e *Ionic Native,* hacen posible trabajar con componentes híbridos.


***Ilustración 13: logo Ionic[22]***

Tiene más de 120 características nativas como Bluetooth, huella dactilar, etc. Cuenta con un pack de iconos propios, componentes, live reload, AoT Compiling, una API de animaciones personalizadas, etc.

[20] Fuente:https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[21] Enlace: https://ionicframework.com

[22] Fuente: https://commons.wikimedia.org/wiki/File:Ionic_Logo.svg

- **Quasar**

Quasar, **Ilustración 14**, es un framework[23] todoterreno que usa *Vue.js* y permite crear aplicaciones de una sola página, proyectos de renderizado del lado del servidor, aplicaciones web progresivas, así como aplicaciones móviles y de escritorio. A pesar de ser un framework muy reciente **[15]**, ya que fue lanzado en 2016, se está volviendo muy popular porque:

- Está basado en *Vue.js*.
- Interfaz de usuario de última generación.
- El mejor soporte para navegadores de escritorio y móviles.
- Propia CLI.
- Fácil personalizable (css) y extensible (js).
- Gran comunidad, cada vez mayor.
- Fácil de aprender.

***Ilustración 14: logo Quasar[24]***

[23] Enlace: https://quasar.dev

[24] Fuente: https://quasar.dev/

- **Conclusión**

React Native y Flutter pueden ser muy buenas aplicaciones para *iOS* y *Android*, aunque este último tiene el inconveniente de tener que aprender su propio lenguaje, Dart.

Sin embargo, Quasar, es sencillo de aprender y puede usarse para muchas más plataformas. También cuento con algo de experiencia previa con este framework, por lo que esto acelerará el proceso de desarrollo evitando perder tiempo en aprender a utilizarlo. Además es muy posible que cada vez sea más usado y demandado por las empresas, por lo que ganar más experiencia me puede ser muy útil.

### 2.4.4 Tecnologías para el despliegue y el back-end

En este apartado se van a discutir las tecnologías disponibles para el despliegue y el back-end de la aplicación.

- **¿Back-end en el lado del cliente o del servidor?**

Lo primero es decidir si se va a realizar el back-end en el lado del cliente o del servidor **[16]**. Recientemente cada vez más aplicaciones realizan la lógica del negocio en el lado del cliente, lo que se suele llamar *serverless*, pero esto no significa que no usen un servidor. Lo que sucede es que se usa un servidor externo como un servicio al que te conectas desde el lado del cliente. Este servidor en la nube puede darte servicios para conectarte a una base de datos, guardar archivos, o incluso ejecutar código.

Esto tiene numerosas ventajas: un rápido desarrollo; una alta escalabilidad solo de los recursos necesarios; se elimina la necesidad del mantenimiento del servidor; puede reducir la latencia, funciones en tiempo real son más fáciles de implementar, etc.

Aunque también tiene desventajas: al estar la lógica en el lado del cliente, hay que tener cuidado con la seguridad; aunque cada día hay más mejoras, el rendimiento puede verse afectado si el cliente tiene que cargar muchos archivos JavaScript; los datos que guardes en el servidor externo no te pertenecen del todo y puedes no tener control total sobre ellos.

Elegir una arquitectura u otra dependerá de las necesidades de la aplicación y de la decisión de los desarrolladores, en general, una arquitectura serverless se recomienda más para aplicaciones pequeñas o medianas. La arquitectura tradicional funciona mejor cuando la aplicación es muy grande, y tiene una carga de trabajo predecible, pudiendo mantener y escalar tu propio servidor más fácilmente.

Como esta aplicación será pequeña, con incertidumbre de hasta dónde puede escalar, tiene más sentido usar una arquitectura serverless. También esto permitirá una mayor velocidad de implementación, lo que viene bien al ser solo un desarrollador.

Hay varias empresas que ofrecen estos servicios, como *Firebase*[25], **Ilustración 15**, *o Backendless*[26]**Ilustración 16**. Ambas tienen varios aspectos en común, pero la primera se paga por el uso, mientras que *Backendless* se paga mensualmente.


***Ilustración 15: logo Firebase***

[25] Enlace: https://firebase.google.com/
[26] Enlace: https://backendless.com

*Ilustración 16: logo Backendless*

- **Tecnologías para el despliegue**

Para el despliegue de la aplicación para poder usarse como página web, hay varias opciones. Se podría contratar un VPS y configurar un servidor web. También *Firebase* tiene un sencillo sistema de *hosting*. Pero existen más alternativas que actualmente se están usando mucho. Como por ejemplo *AWS* [27], el cuál el precio a pagar es muy bajo ya que depende del uso que tu aplicación esté haciendo en cada momento. Escala automáticamente según las necesidades.

También es muy común usar *Terraform*[28] junto a *AWS*. Esto permite crear la infraestructura con código, para que sea más rápido de modificar ciertas políticas o características o para crear varios servidores a la vez, lo que hacerlo manualmente sería más lento.

## 2.5 Propuesta de solución

Se pretende crear una aplicación multiplataforma y *serverless* gracias al uso de *Quasa*r y *Firebase*. Para desplegar la aplicación en internet podría usarse el sistema de *hosting* de *Firebase, pero* se va tratar de usar *AWS* con *Terraform,* principalmente por motivos didácticos.

Esta aplicación deberá de servir a la cliente para gestionar más fácilmente a todas sus alumnas y sus clases. Lo ideal sería que le ahorrase tiempo de trabajo a ella pero también a sus alumnas, de forma que estás puedan acceder a la información de las clases más rápido y de manera más organizada. Una

[27] Enlace: https://aws.amazon.com/es/
[28] Enlace: https://www.terraform.io

visión más avanzada de la aplicación sería que actuase como red social entre todas las alumnas y la cliente, permitiendo una comunicación más única y cercana.

## 2.6 Especificación de las funcionalidades

Antes de ver las historias de usuario veo necesario ver las funcionalidades que necesitará la aplicación. De esta forma se entenderá mejor cómo será su funcionamiento y se podrán realizar las historias de usuario más fácilmente. Se ha hecho una división en dos, para mostrar aquellas funcionalidades necesarias, **Tabla 2**, y aquellas que aportan valor añadido pero no son vitales para el funcionamiento básico, **Tabla 3**. Esto luego también ayudará a determinar la prioridad de las historias de usuario.

| Utilidad mínima necesaria |
|---|
| Sistema de Inicio de sesión |
| Sistema de Registro protegido con contraseña |
| Página de Inicio |
| Página de administración para poder visualizar a todas las alumnas, ver sus datos y desactivar su cuenta si fuese necesario |
| Página de administración para poder crear y editar las clases: cambiar su nombre, enlaces, etc |

| |
|---|
| Página de administración para actualizar un calendario mensual donde se indica el horario de todas las clases |
| Las alumnas deben de poder apuntarse a diferentes clases para poder acceder a sus enlaces |
| Las alumnas deben de poder desapuntarse de una clase |
| Las alumnas deben de poder ver el calendario mensual |
| Una alumna con la cuenta desactivada no debe de poder acceder a las clases |

***Tabla 2: funcionalidades mínimas***

| **Utilidad secundaria** |
|---|
| Página de administración para cambiar la contraseña de Registro |
| Las alumnas deben de poder cambiar su imagen de perfil |
| La administradora debe poder cambiar la imagen de las clases |
| Sistema de reserva para las clases |
| Sección de ajustes para poder activar o desactivar notificaciones, activar modo oscuro, etc |
| Sistema de pagos, las alumnas deben de poder pagar la mensualidad desde la propia aplicación |

| |
|---|
| Sistema de chat |

***Tabla 3: Funcionalidades secundarias***

## 2.7 Historias de usuario

Como se mencionó anteriormente, se va utilizar el modelo *MoSCoW* **[17]**, el cuál propone 4 categorías de prioridades, **Tabla 4**, para priorizar las historias de usuario.

| **Prioridades** | |
|---|---|
| Debe tener | Característica crítica para el sistema. Sin ella el proyecto será un fracaso. |
| Debería incluir | Características críticas pero no imprescindibles. |
| Podría incluir | Añaden valor pero no son críticas |
| No se va a hacer | No aportan un valor en este momento o no merecen la inversión |

***Tabla 4: Prioridades MoSCoW***

A continuación, **Tablas 5-24**, se muestran las historias de usuario, con su prioridad y los criterios de aceptación.

| **ID: 1 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero ver una lista con todas mis alumnas, para tener mayor gestión sobre ellas |
| Prioridad | Debe tener |
| Criterios de aceptación | - Solo la administradora puede tener acceso<br>- Debe de poder acceder a la información de cada alumna<br>- Debe poder filtrar y buscar a las alumnas |

***Tabla 5: historia de usuario 1***

| **ID: 2 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero poder activar/desactivar a una alumna, para controlar su acceso a las clases |
| Prioridad | Debe tener |
| Criterios de aceptación | - Solo la administradora puede tener acceso<br>- Debe de poder activar o desactivar a una alumna en cualquier momento<br>- La alumna bloqueada no debe poder acceder a ninguna parte de la aplicación hasta que vuelva a ser activada (excepto zonas donde no haga falta estar *logueado)* |

***Tabla 6: historia de usuario 2***

| **ID: 3 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero poder crear/editar/borrar una clase, para que mis alumnas tengan la información actualizada al momento |
| Prioridad | Debe tener |

| | |
|---|---|
| Criterios de aceptación | - Solo la administradora puede tener acceso<br>- Debe de poder crear, modificar o borrar una clase en cualquier momento<br>- Una clase eliminada no será visible nunca más.<br>- La información nueva debe ser reflejada en la sección de las alumnas |

***Tabla 7: historia de usuario 3***

| **ID: 4 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero poder generar un código de registro, para que no cualquiera pueda registrarse |
| Prioridad | Debería incluir |
| Criterios de aceptación | - Solo la administradora puede tener acceso<br>- Debe de poder cambiar el código en cualquier momento.<br>- Cuando se cree un código nuevo, el anterior debe de dejar de funcionar |

***Tabla 8: historia de usuario 4***

| **ID: 5 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero poder actualizar un calendario de clases, para que las alumnas puedan ver los horarios de forma organizada y actualizada. |
| Prioridad | Debería incluir |
| Criterios de aceptación | - Solo la administradora puede tener acceso para modificar el calendario<br>- Debe de poder cambiar el calendario en cualquier momento y reflejar el cambio en la sección de alumnas<br>- Solo las alumnas registradas deben de poder ver el calendario |

*Tabla 9: historia de usuario 5*

| ID: 6 (ver tareas ⬇️) | |
|---|---|
| Historia de usuario | Como administradora, quiero poder comunicarme con las alumnas desde la aplicación, para no depender de otras aplicaciones. |
| Prioridad | Podría incluir |
| Criterios de aceptación | - Solo las alumnas registradas podrán comunicarse con la administradora<br>- La administradora debe de conocer la identidad de la alumna con la que está hablando<br>- La administradora debería poder enviar un mensaje a la vez a todas las alumnas |

*Tabla 10: historia de usuario 6*

| ID: 7 (ver tareas ⬇️) | |
|---|---|
| Historia de usuario | Como administradora, quiero que las alumnas puedan reservar las clases, para evitar que acudan demasiadas personas a la misma clase. |
| Prioridad | Podría incluir |
| Criterios de aceptación | - Solo las alumnas registradas podrán reservar la clase<br>- Cuando una clase alcance su capacidad máxima de reserva, no puede dejar reservar, se debe de indicar y permitir a la alumna ponerse en cola<br>- No se podrá acceder a la clase en directo si no se ha reservado<br>- La administradora debe de saber que alumnas han reservado<br>- La alumna debe de poder cancelar su reserva |

*Tabla 11: historia de usuario 7*

| **ID: 8 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero un sistema de penalización cuando una alumna reserve una clase y no asista, para asegurar la asistencia de las alumnas. |
| Prioridad | Podría incluir |
| Criterios de aceptación | - El sistema debe de ser automático<br>- Se le debe de notificar a la alumna |

***Tabla 12: historia de usuario 8***

| **ID: 9 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero un sistema de pagos, para que las alumnas puedan pagar desde la aplicación |
| Prioridad | Podría incluir |
| Criterios de aceptación | - El sistema debe de ser automático<br>- Una alumna que no haya pagado la mensualidad será bloqueada automáticamente<br>- Se debe de avisar a la alumna de que será bloqueada |

***Tabla 13: historia de usuario 9***

| **ID: 10 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como administradora, quiero controlar a los monitores contratados, para asignarlos a diferentes clases. |
| Prioridad | No se va a hacer |
| Criterios de aceptación | - La administradora debe de poder cambiar el cualquier momento el |

| | |
|---|---|
| | monitor asignado a cada clase<br>- El monitor debe de ser notificado del cambio |

***Tabla 14: historia de usuario 10***

| **ID: 11 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como monitor no administrador, quiero poder ver a qué clases estoy asignado, para ver su información. |
| Prioridad | No se va a hacer |
| Criterios de aceptación | - Un monitor no administrador debe de poder ver las alumnas apuntadas a la clase asignada<br>- No debe de poder cambiar ninguna información de la clase, excepto el enlace en directo |

***Tabla 15: historia de usuario 11***

| **ID: 12 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero poder registrarme, para tener una cuenta |
| Prioridad | Debe tener |
| Criterios de aceptación | - El formulario de registro debe de ser válido, si no, se debe de indicar un error<br>- No se podrá registrar si el email ya existe<br>- Las contraseña deberá de introducir dos veces y coincidir, además de tener una longitud mínima necesaria.<br>- Una alumna solo se podrá registrar si conoce el código de registro |

***Tabla 16: historia de usuario 12***

| **ID: 13 (ver tareas ⬇️)** |
|---|

| | |
|---|---|
| Historia de usuario | Como alumna, quiero poder iniciar sesión, para acceder a mi cuenta |
| Prioridad | Debe tener |
| Criterios de aceptación | - El formulario de inicio de sesión debe de ser válido, si no, se debe de indicar un error |

***Tabla 17: historia de usuario 13***

| **ID: 14 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como usuario, quiero cerrar sesión, para no mantener mi sesión abierta. |
| Prioridad | Debe tener |
| Criterios de aceptación | - La sesión debe cerrarse inmediatamente y no debe ser posible navegar por una sección que se requiera autentificación. |

***Tabla 18: historia de usuario 14***

| **ID: 15 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero poder cambiar mi foto de perfil, para personalizar mi cuenta |
| Prioridad | Podría tener |
| Criterios de aceptación | - Una alumna debe de estar registrada y haber iniciado sesión para cambiar su foto.<br>- Una alumna solo podrá cambiar su foto, y no la de otra alumna.<br>- El cambio debe de reflejarse para que lo vea la alumna. |

***Tabla 19: historia de usuario 15***

| **ID: 16 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero poder apuntarme a clases, para tener acceso a su información. |
| Prioridad | Debe tener |
| Criterios de aceptación | - Una alumna debe de estar registrada y haber iniciado sesión para apuntarse a una clase<br>- La clase a la que se haya apuntado debe de aparecer ahora en otra sección.<br>- No debe de poder apuntarse a una clase más de una vez.<br>- Las acciones de las alumnas no deben afectar a las otras alumnas. |

***Tabla 20: historia de usuario 16***

| **ID: 17 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero poder desapuntarme de una clase que no me interesa, para no saturarme con su información. |
| Prioridad | Debe tener |
| Criterios de aceptación | - Una alumna debe de estar registrada y haber iniciado sesión para apuntarse a una clase<br>- La clase a la que se haya desapuntado debe de aparecer ahora en otra sección.<br>- Las acciones de las alumnas no deben afectar a las otras alumnas. |

***Tabla 21: historia de usuario 17***

| **ID: 18 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero recibir notificaciones, para que no se me olvide cuando comienza una clase. |

| | |
|---|---|
| Prioridad | No se va a hacer |
| Criterios de aceptación | - Una alumna debe de estar registrada y haber iniciado sesión para recibir notificaciones<br>- La alumna debe de tener las notificaciones activas para recibir notificaciones |

***Tabla 22: historia de usuario 18***

| **ID: 19 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero recibir notificaciones, para que no se me olvide renovar mi suscripción. |
| Prioridad | No se va a hacer |
| Criterios de aceptación | - Una alumna debe de estar registrada y haber iniciado sesión para recibir notificaciones<br>- La alumna debe de tener las notificaciones activas para recibir notificaciones |

***Tabla 23: historia de usuario 19***

| **ID: 20 (ver tareas ⬇️)** | |
|---|---|
| Historia de usuario | Como alumna, quiero acceder a la aplicación desde el navegador o cualquier dispositivo, para tener facilidades de acceso. |
| Prioridad | Debería tener |
| Criterios de aceptación | - Una alumna debe de estar registrada y haber iniciado sesión para acceder a la aplicación<br>- Cualquier usuario sin registrar debe de poder acceder a la aplicación para poder registrarse |

***Tabla 24: historia de usuario 20***

## 2.7 Planificación inicial de tareas

Para la planificación inicial de tareas, que se corresponderá con el *backlog*, se van a ir creando las tareas necesarias para cada historia de usuario. Además la estimación de tiempo se va a realizar para cada tarea. Se podría haber realizado la estimación de forma más general para cada historia de usuario, pero pienso que saldría una estimación menos real.

Se va utilizar un sistema muy simple pero práctico para estimar, el sistema de tallaje de camisetas **[18]**. En realidad este sistema propone asignar a cada tarea una talla de una camiseta para conocer el tamaño o dificultad de la tarea. En general, para estimar si se va a tardar más en hacer o menos. Así, a la hora de trabajar con el *backlog*, los desarrolladores pueden elegir la siguiente tarea a realizar según la talla, teniendo una estimación de sí tardará más en hacerla o menos. ¿Pero cómo estimar las horas totales con este sistema? He pensado en asignar a cada talla, **Tabla 25**, un número de horas en concreto. Así se consigue una estimación general y rápida. Luego basta con "sumar" las tallas para conocer la estimación en horas.

Creo que este sistema también permite identificar si una tarea del *backlog* podría descomponerse en tareas más pequeñas. Si se le asigna sin duda la talla XXL a una tarea, tal vez esa tarea se pueda dividir en dos, tres o más tareas. Y así estimar con más precisión. Para seguir el sistema *kanban* de entrega continua y donde las tareas deben de ser lo más pequeñas posible, lo ideal sería utilizar solo tallas desde la XXS hasta la M o L.

| Tallaje | Horas estimadas |
|---|---|
| (-) sin talla | 0 horas |
| XXS | 2 horas |
| XS | 5 horas |
| S | 10 horas |

| | |
|---|---|
| M | 20 horas |
| L | 40 horas |
| XL | 60 horas |
| XXL | 100 horas |

***Tabla 25: estimación de horas según la talla***

Además he añadido una categoría sin talla. Generalmente no vas a asignar esta categoría a ninguna tarea, al menos no a las iniciales, pero con el transcurso del tiempo pueden surgir nuevas tareas y algunas de ellas pueden ser solo pequeños recordatorios para cambiar algo en unos momentos. Así, si por ejemplo un software contase las tallas automáticamente y constantemente para obtener la horas estimadas, las horas de las tareas completadas y de aquellas del *backlog* para luego sacar estadísticas, estas pequeñas tareas no afectarían al cómputo total para no empeorar la estimación.

A continuación pasaremos a ver las tareas iniciales, **Tablas 26-45**, donde se podrá ver la descripción de la tarea, la talla asignada y el área al que pertenece. Pienso que añadir este último dato ayuda a dividir aún más las tareas. ¿Esta tarea puede pertenecer a dos áreas? Entonces, dividela.

| **Historia de usuario con ID 1 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Diseño de la aplicación: wireframes, prototipos, etc. | M | Diseño UI/UX |
| Crear la BBDD en Firebase y crear su estructura | S | Back-end |
| Creación de un layout para la página de administración | XS | Front-end |
| Creación de los estilos de la página de administración para | XS | Front-end |

| | | |
|---|---|---|
| gestionar las alumnas | | |
| Creación de una tabla con filtros para mostrar y buscar a las alumnas usando la información de un objeto | S | Front-end |
| Pensar y crear el controlador necesario para conectar la tabla de alumnas con la BBDD | XS | Back-end |
| Conectar Firebase con Quasar | XS | Back-end |
| Crear los métodos necesarios en el controlador para el control de la tabla | XS | Back-end |
| Mostrar las alumnas en la tabla de las alumnas a través del controlador que obtiene la información de la BBDD | XXS | Back-end |
| **Total de horas estimadas:** | **67 horas** | |

***Tabla 26: tareas de la historia de usuario 1***

| **Historia de usuario con ID 2 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de una página para ver la información de una alumna | XXS | Front-end |
| Creación y uso de los métodos en el controlador para poder cambiar el estado de una alumna a bloqueado y activo | XS | Back-end |
| Una alumna bloqueada no debe | XXS | Back-end |

| | | |
|---|---|---|
| de tener acceso a la aplicación | | |
| Mostrar que una alumna no tiene acceso a la aplicación | XXS | Front-end |
| Poder acceder a cada alumna según su ID para activarla o desactivarla | XXS | Front-end |
| **Total de horas estimadas:** | **13 horas** | |

*Tabla 27: tareas de la historia de usuario 2*

| **Historia de usuario con ID 3 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de los estilos de una página para ver y crear las clases | XXS | Front-end |
| Creación de un componente clase para ser usado en otras páginas | XXS | Front-end |
| Creación de los estilos de una página para ver la información de una clase | XXS | Front-end |
| Formulario para crear, editar y borrar una clase | XXS | Front-end |
| Acceder a la información de cada clase según su ID | XXS | Front-end |
| Crear en el controlador los métodos necesarios para modificar las clases | XS | Back-end |
| Conectar los formularios para modificar clases con el controlador | XSS | Back-end |

| **Total de horas estimadas:** | **17 horas** |
|---|---|

*Tabla 28: tareas de la historia de usuario 3*

| **Historia de usuario con ID 4 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de una página para generar un código de registro | XXS | Front-end |
| Generar los métodos en el controlador para editar el código de registro en la BBDD y conectarlo con la página | XXS | Back-end |
| **Total de horas estimadas:** | **4 horas** | |

*Tabla 29: tareas de la historia de usuario 4*

| **Historia de usuario con ID 5 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de una página para mostrar el calendario y su formulario para editarlo | XXS | Front-end |
| Crear los métodos en el controlador para editar el calendario en la BBDD | XS | Back-end |
| Almacenar imágenes en Firebase y conectar con la página del calendario | S | Back-end |
| **Total de horas estimadas:** | **17 horas** | |

*Tabla 30: tareas de la historia de usuario 5*

| Historia de usuario con ID 6 (ver historia ⬆️) | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Investigación sobre sistemas de chat | XS | General |
| Crear lógica para el chat | L | Back-end |
| Crear los estilos y estructura necesarios para el chat | S | Front-end |
| **Total de horas estimadas:** | **55 horas** | |

*Tabla 31: tareas de la historia de usuario 6*

| Historia de usuario con ID 7 (ver historia ⬆️) | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Crear la lógica para un sistema de reservas | M | Back-end |
| Estilos y estructura para mostrar el sistema de reservas | XS | Front-end |
| **Total de horas estimadas:** | **25 horas** | |

*Tabla 32: tareas de la historia de usuario 7*

| Historia de usuario con ID 8 (ver historia ⬆️) | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |

| | | |
|---|---|---|
| Crear la lógica para un sistema de penalización | S | Back-end |
| Estilos y estructura para mostrar el sistema de penalización | XS | Front-end |
| **Total de horas estimadas:** | **15 horas** | |

*Tabla 33: tareas de la historia de usuario 8*

| **Historia de usuario con ID 9 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Pensar e investigar cómo introducir un sistema de pagos | XXS | General |
| Crear la lógica para un sistema de pagos | M | Back-end |
| Página para mostrar el sistema de pagos | S | Front-end |
| **Total de horas estimadas:** | **32 horas** | |

*Tabla 34: tareas de la historia de usuario 9*

| **Historia de usuario con ID 10 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Lógica para asignar distintos monitores a las clases | XXS | Back-end |
| **Total de horas estimadas:** | **2 horas** | |

*Tabla 35: tareas de la historia de usuario 10*

| Historia de usuario con ID 11 (ver historia ⬆️) | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Paǵina para los monitores | XS | Front-end |
| Lógica de la página de los monitores | S | Back-end |
| **Total de horas estimadas:** | **15 horas** | |

*Tabla 36: tareas de la historia de usuario 11*

| Historia de usuario con ID 12 (ver historia ⬆️) | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de una página de registro | XXS | Front-end |
| Componente formulario control de acceso | XS | Front-end |
| Página de Inicio | M | Front-end |
| Controlador para el control de acceso | XS | Back-end |
| Métodos y lógica para registrar usuarios | S | Back-end |
| **Total de horas estimadas:** | **42 horas** | |

*Tabla 37: tareas de la historia de usuario 12*

| **Historia de usuario con ID 13 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de una página de Inicio de Sesión | XXS | Front-end |
| Métodos y lógica para Inicio de sesión | S | Back-end |
| **Total de horas estimadas:** | **12 horas** | |

*Tabla 38: tareas de la historia de usuario 13*

| **Historia de usuario con ID 14 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Añadir un botón para cerrar sesión | XXS | Front-end |
| Protección de rutas | XXS | Back-end |
| **Total de horas estimadas:** | **4 horas** | |

*Tabla 39: tareas de la historia de usuario 14*

| **Historia de usuario con ID 15 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Mostrar imagen de perfil y formulario para cambiarla | XXS | Front-end |
| Lógica y métodos para cambiar foto de perfil | XS | Back-end |

| | |
|---|---|
| **Total de horas estimadas:** | **7 horas** |

*Tabla 40: tareas de la historia de usuario 15*

| **Historia de usuario con ID 16 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Página con todas las clases disponibles | XXS | Front-end |
| Página donde se muestra la información completa de la clase | XS | Front-end |
| Lógica y métodos para acceder a la información de cada clase según su id | XS | Back-end |
| Métodos para cada alumna poder apuntarse a una clase | S | Back-end |
| **Total de horas estimadas:** | **22 horas** | |

*Tabla 41: tareas de la historia de usuario 16*

| **Historia de usuario con ID 17 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Métodos para cada alumna poder desapuntarse a una clase | XXS | Back-end |
| **Total de horas estimadas:** | **2 horas** | |

*Tabla 42: tareas de la historia de usuario 17*

| **Historia de usuario con ID 18 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Creación de un sistema de notificaciones multiplataforma | M | Back-end |
| Lógica para mostrar notificaciones cuando una clase vaya a comenzar | XS | Back-end |
| **Total de horas estimadas:** | **25 horas** | |

*Tabla 43: tareas de la historia de usuario 18*

| **Historia de usuario con ID 19 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Lógica para mostrar notificaciones cuando la suscripción de pagos vaya a terminar | XS | Back-end |
| **Total de horas estimadas:** | **5 horas** | |

*Tabla 44: tareas de la historia de usuario 19*

| **Historia de usuario con ID 20 (ver historia ⬆️)** | | |
|---|---|---|
| **Tareas** | **Talla** | **Área** |
| Desplegar la aplicación a Internet | S | Back-end |
| Build para Android e iOs | S | Back-end |

| **Total de horas estimadas:** | **20 horas** |
|---|---|

*Tabla 45: tareas de la historia de usuario 20*

Finalmente obtenemos una estimación de 401 horas, **Tabla 46**, para la parte del desarrollo, sin tener en cuenta las horas necesarias para desarrollar este documento. Como el TFG tiene una duración de 300 horas, se irán seleccionando aquellas historias más prioritarias hasta alcanzar esa cifra.

| Total horas estimadas: | **401 horas** |
|---|---|

*Tabla 46: total horas estimadas*

## 2.8 Estudio de viabilidad

### 2.8.1 Costes de desarrollo y producción

En este apartado se va a ver los costes necesarios para realizar este proyecto. Se van a dividir en costes software, costes de servicios externos, costes hardware y costes humanos. También se comentará la forma legal necesaria que habrá que adoptar para poder vender el producto y por último se verá un diagrama de Gantt, donde se podrá apreciar la organización y división del trabajo.

#### Costes Software

A continuación se muestran los costes de software, **Tabla 47**, por mes, en 5 meses y en un año. Esta aplicación no requiere inversión inicial en software, ya que todo se puede realizar con las versiones de software gratuito.

| Software | 1 mes | 5 meses | 1 año |
|---|---|---|---|
| Figma (versión gratuita) | 0 euros | 0 euros | 0 euros |
| GitKraken (versión estudiante) | 0 euros | 0 euros | 0 euros |
| Visual Studio Code | 0 euros | 0 euros | 0 euros |
| Google Chrome | 0 euros | 0 euros | 0 euros |
| Firefox | 0 euros | 0 euros | 0 euros |
| Google Docs | 0 euros | 0 euros | 0 euros |
| BitBucket | 0 euros | 0 euros | 0 euros |
| Trello | 0 euros | 0 euros | 0 euros |
| **Total** | 0 euros | 0 euros | 0 euros |

***Tabla 47: costes software***

## Costes de Servicios

Ahora pasaremos a ver los costes de los servicios externos. En este apartado no se han incluido costes como el internet o facturas de alquiler, luz, etc. Como esos gastos se van a pagar independientemente del desarrollo del proyecto, y no supone un mayor gasto considerable, he decidido no tenerlos en cuenta. A continuación se muestran los costes de servicios, **Tabla 48**, por mes, en 5 meses y en un año.

| Servicio | 1 mes | 5 meses | 1 año |
|---|---|---|---|
| AWS | 1-4 euros | 5 - 20 euros | 12 - 48 euros |

| | | | |
|---|---|---|---|
| Firebase | 0 euros | 0 euros | 0 euros |
| **Total** | 1-4 euros | 5 - 20 euros | 12 - 48 euros |

***Tabla 48: costes de servicios***

Como estos servicios se pagan por el uso, es complicado determinar cuánto va a costar. Más teniendo en cuenta que no hay un historial de la carga de trabajo que la aplicación requiere. En AWS para el servicio S3 bucket **[19]**, los precios varían según la transferencia de datos diarias, el almacenamiento utilizado, etc. En general, rondan los 0.02-0.09 euros/GB. Para Firebase **[20]**, ocurre lo mismo, aunque el plan gratuito es bastante holgado, la parte más limitante es que solo permite 100 accesos simultáneos a la base de datos. Tal vez no sea suficiente con el tiempo, pero al comienzo si. Con el plan de pago se pueden realizar hasta 200.000 accesos simultáneos, pero la página no deja claro el precio de esta característica.

Como se puede observar, los gastos de estos servicios para una aplicación pequeña como esta, son muy pocos. Por lo que viene muy bien para un pequeño negocio, ya que la cliente será quien se haga cargo de ellos.

### Costes Hardware

En cuanto a los costes hardware, se van a mostrar los dispositivos necesarios y recomendados. Para el desarrollo solo hace falta 1 portatil con cualquier sistema operativo. Pero para testear bien la aplicación hacen falta más dispositivos con diferentes sistemas operativos. Aunque se pueden usar simuladores, es más fiable testear en un dispositivo real. Además para realizar la exportación de la aplicación para iOS es necesario un Mac. A continuación se muestran los costes hardware, **Tabla 49**, por unidad, en un mes, en 5 meses y en un año.

| Hardware | Coste por | Tiempo de | 1 mes | 5 meses | 1 año |
|---|---|---|---|---|---|

| | unidad | vida estimado | | | |
|---|---|---|---|---|---|
| Portatil Linux + windows | 1200 euros | 4 años | 26 euros | 130 euros | 312 euros |
| Portatil Mac | 1500 euros | 4 años | 32.5 euros | 162.5 euros | 390 euros |
| Telefono Android | 300 euros | 4 años | 6,25 euros | 31,25 euros | 75 euros |
| Telefono iOS | 800 euros | 4 años | 17.3 euros | 86,6 euros | 208 euros |
| **Total** | 3800 euros | - | 82,05 euros | 374,4 euros | 985 euros |

***Tabla 49: costes hardware***

Para determinar el tiempo de vida útil, se ha consultado la tabla de amortizaciones de la Agencia Tributaria para equipos tecnológicos **[21]**, donde la forma más común de calcularlo es usar el coeficiente máximo anual del 26% respecto al precio del producto.

Si no se dispone de estos dispositivos o se puede acceder a ellos con frecuencia, habrá que realizar una inversión de 3800 euros. Aunque, para comenzar el desarrollo solo es estrictamente necesario el portátil, lo que sería 1200 euros.

### Costes Humanos

Para el cálculo de los costes humanos hay que tener en cuenta los diferentes puestos involucrados, su sueldo **[22]** y las horas trabajadas. A continuación se muestran los costes humanos, **Tabla 50**, según el puesto de trabajo, y las horas trabajadas.

| Puesto | Horas trabajadas | Sueldo por hora | Total |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Analista programador | ~90 horas | 14,10 euros | 1269 euros |
| Diseñador UI/UX | 40 horas | 12,5 euros | 500 euros |
| Programador Full-Stack | 300 horas | 19 euros | 5.700 euros |
| **Total** | - | - | **7469 euros** |

***Tabla 50: costes humanos***

Estos gastos de personal, en realidad determinan el precio de venta de la aplicación para la cliente. Pero para poder vender la aplicación como un producto hay que elegir la forma legal adecuada.

## 2.8.2 Forma legal elegida y costes asociados

Como Luis Martín Mesa comenta en su libro Contabilidad y Fiscalidad (Universidad de Jaén) **[23]**, hay multitud de clasificaciones de empresas, ya sea por su tamaño, o por las características de su titular. Y es necesario constituir una empresa para ser una unidad económica de producción, ya sea para ofrecer un servicio o vender un producto. Está claro que en este caso, al ser una sola persona, la empresa se clasificaría como microempresa por su tamaño, pero por las características de su titular, hay varias que se adaptan a un solo socio. Por ejemplo, sería posible fundar una Sociedad Limitada Nueva empresa, ya que solo requiere un socio como mínimo, pero siendo la aplicación una única fuente de ingresos puntual no tiene mucho sentido. Además de tener que añadir a las reservas un capital mínimo. Para este caso tiene más sentido darse de alta como autónomo. Donde el mayor gasto a tener en cuenta es una tarifa de 293 euros al mes. Sin embargo, para nuevos autónomos y durante el primer año, solo es 60 al mes.

Cuando te das de alta de autónomo puedes darte de baja cuando quieras. Si tenemos en cuenta el descuento en la tasa de autónomos podría ser interesante cobrar un mantenimiento y mejora de la aplicación durante el primer año. En la siguiente tabla, **Tabla 51**, se puede observar una cuenta

previsional de resultados donde se propone trabajar 10 horas a la semana en el transcurso del año para continuar mejorando la aplicación, corregir errores, o realizar el mantenimiento que sea necesario.

| CONCEPTO | PRIMER MES | PRIMER AÑO |
|---|---|---|
| **Ingresos previstos** | | |
| Ventas netas | 7.469 € | 9.120 € |
| **Gastos previstos** | | |
| Gastos de subcontratación | 0 € | 0 € |
| Gasto de personal | 0 € | 0€ |
| Bienes y materiales | 300 € | 0 € |
| Otros gastos (software, cuota de autónomos) | 60 € | 720 € |
| Amortizaciones | 75,8 € | 910 € |
| **Resultado de Explotación (BAIT)** | **7.033,2 €** | **7.490 €** |
| Gastos financieros | 0 € | 0 € |
| **Resultado Antes de Impuestos (BAT)** | **7.033,2 €** | **7.490 €** |
| Impuesto sobre beneficios (IRPF) | 492,3 € (7%) | 524,3 € (7%) |
| **Resultado del ejercicio** | **6540,9 €** | **6965,7 €** |
| Dividendos | 6540,9 € | 6965,7 € |

*Tabla 51: cuenta previsional de resultados*

Como se ha podido ver, para el primer mes, en bienes y materiales solo se ha tenido en cuenta el teléfono Android de 300 euros. Esto es así porque a partir de ese precio, el gasto se debe amortizar. Los 75,8 euros de la fila de la amortización corresponden a los demás equipos. También se ha añadido la cuota de autónomos para el primer mes. En cuanto a los impuestos, los autónomos tienen que pagar un 7% de IRPF el primer año.

Luego, durante el año siguiente se continua amortizando. Las ventas netas de ese año vienen de calcular 10 horas semanales de trabajo por 19 euros la hora (desarrollador Full-Stack).

## 2.8.3 Diagrama de Gantt

Este diagrama de Gantt, **Ilustración 17**, muestra la organización inicial del proyecto. Como columnas se encuentran las semanas y como filas los diferentes apartados del proyecto. Como se puede ver, hay 20 semanas, que se corresponde con 5 meses. Se piensa dividir 300 horas de la implementación más 100 horas del desarrollo de la memoria en estos 5 meses. El resultado son 4 horas diarias de trabajo en días laborales.

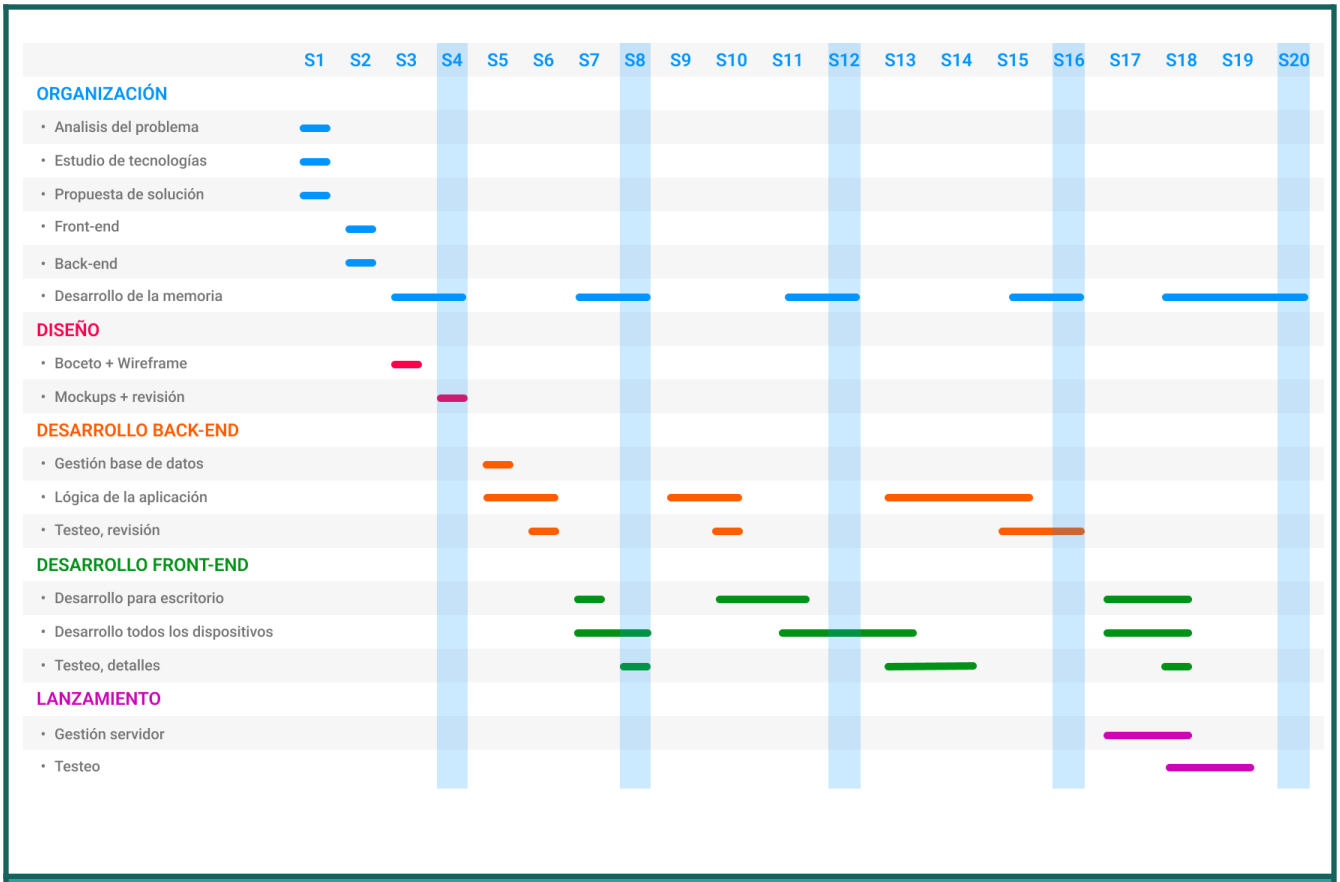



**Ilustración 17: Diagrama de Gantt**

- **Primer mes:**
    - Primera semana: En esta primera semana únicamente se trabajará simultáneamente con el análisis del problema, el estudio de tecnologías y con la propuesta de solución.
    - Segunda semana: Se organizarán diferentes aspectos relacionados con el front-end y el back-end. Además de la organización de la estructura de carpetas y archivos del framework.
    - Tercera y cuarta semana:: Se empezará a escribir este documento. Se escribirá la planificación inicial, historias de usuario y todo lo necesario para empezar a trabajar de forma organizada. También se realizarán los diseños de la aplicación.

- **Segundo mes:**
    - Primera y segunda semana: Aquí se empieza con la parte del back-end. En concreto con la gestión de la base de datos. Se establecerá su estructura, se estudiará su funcionamiento y reglas de seguridad. Se realizará la conexión de la base de datos con la aplicación. Se empezarán a añadir las funcionalidades para el control de acceso y de las clases. Así como el control y protección de rutas de la aplicación. En la segunda semana se irá revisando lo añadido en busca de errores.
    - Tercera semana: Se empezará con el desarrollo Front-end, principalmente para la versión de escritorio. Se darán estilos y estructura a las páginas relacionadas con el control de acceso y las clases. También se continuará con el desarrollo de la memoria.
    - Cuarta semana: Se continuará con el desarrollo Front-end, con más énfasis en la adaptación para dispositivos móviles. Se testeará también lo añadido hasta este momento y se continuará con la memoria.

- **Tercer mes:**
    - Primera semana: Se continuará con el desarrollo Back-end, terminando la lógica de la parte de las clases.

- Segunda semana: Se empezará con el desarrollo de Back-end de la parte de administración, para el control de las alumnas y las clases. Se revisará lo añadido en busca de fallos. A la vez se continuará con el desarrollo Front-end.
- Tercera y cuarta semana: Se continuará con el desarrollo Front-end, dando estilos y estructura a las páginas de las clases y de la parte de la administración. Se continuará con la memoria.

- **Cuarto mes:**
  - Primera y segunda semana: Se trabajará simultáneamente con el Back-end y el Front-end. Terminando y puliendo los aspectos generales de la aplicación. Y si el tiempo lo permite añadiendo funcionalidades secundarias, como el sistema de chat.
  - Tercera semana y cuarta semana: Se continuará con el desarrollo de la memoria y se terminará el Back-end de la aplicación.

- **Quinto mes:**
  - Primera semana: Se trabajará en el Front-end de la aplicación, se realizará la página de Inicio correctamente y algunas páginas secundarias. Se empezará a trabajar con AWS y Terraform para el primer despliegue de la aplicación.
  - Segunda semana y tercera semana: Se terminará de trabajar con el Front-end , testear el despliegue de la aplicación y con la memoria.
  - Cuarta semana: Se terminará de completar y revisar este documento.

## 2.9 Comparativa de estimación y tiempo dedicado

En este apartado, desarrollado después de finalizar los 5 meses comentados en el Diagrama de Gantt, se va a ver el tiempo estimado y el tiempo dedicado final. En tiempo estimado total son 400 horas: 300 horas para el desarrollo y 100 horas del análisis y escribir este documento.

A cada mes se le ha asignado un tiempo estimado diferente, **Ilustración 18**, que en total suma las 400 horas. Esto simplemente ha sido mi forma de organizarme. He pensado que los primeros meses dedicaría menos tiempo, y que al final iría dedicando más. El tiempo estimado que se muestra contiene la estimación del desarrollo y de escribir este documento.

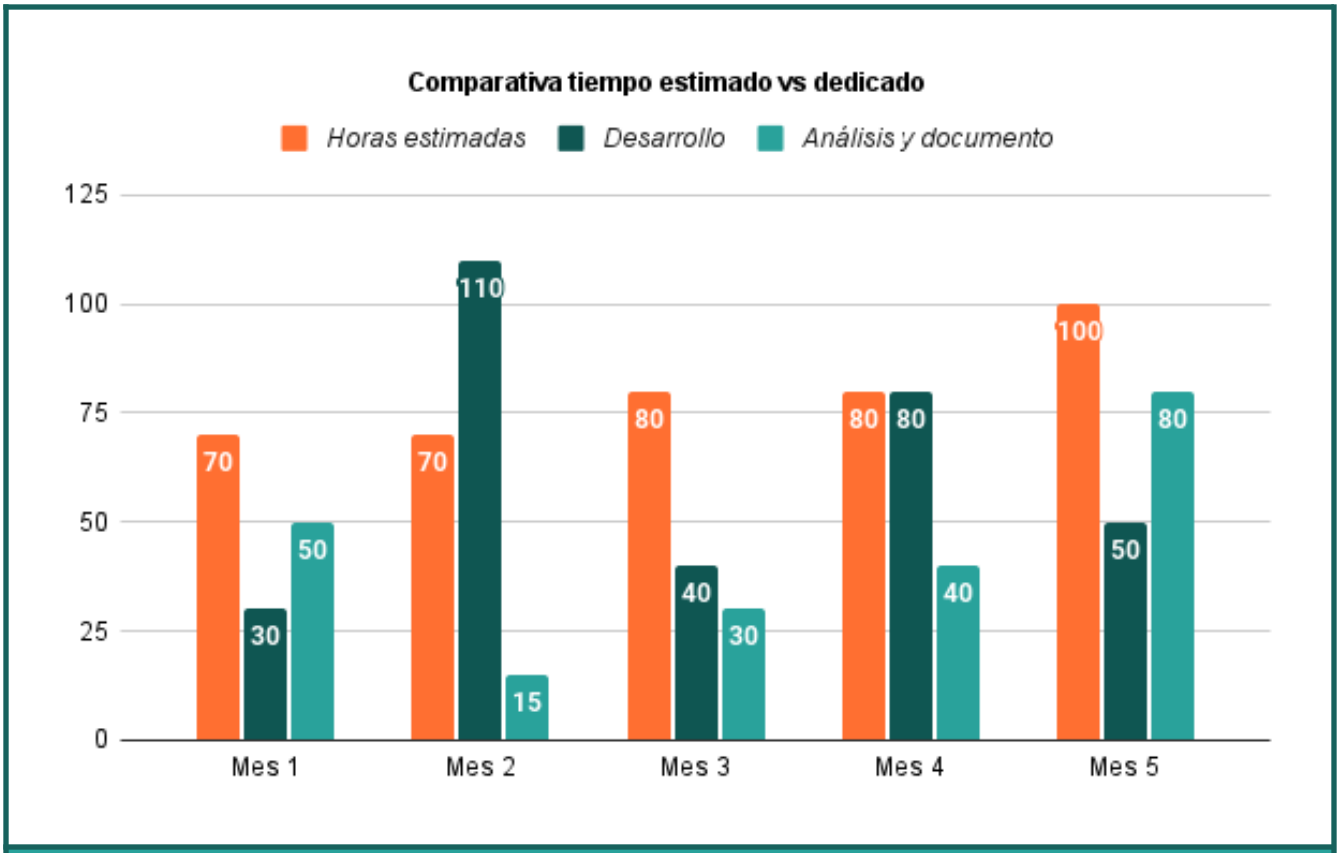



**Ilustración 18: Comparativa tiempo estimado y tiempo dedicado**

- **Primer mes:**

Como *kanban* trata de entregas continuas, el primer mes trabajé lo que me establecí y fui entregando conforme terminaba, por eso el tiempo real total (50+30) casi coincide con el estimado (70). En general me dió tiempo para realizar los primeros análisis establecidos en el primer mes del Diagrama de Gantt y terminar los prototipos más importantes de la aplicación.

- **Segundo mes:**

El segundo mes, tuve un gran bloqueo con el diseño de los diferentes diagramas, por lo que apenas avancé con el documento. Sin embargo, aproveché para avanzar todo lo que pude con el desarrollo. Como este mes estuve sin asistir a clases por enfermedad, pude dedicar muchas más horas y prácticamente terminar la mayoría de funcionalidades prioritarias.

- **Tercer mes:**

Este tercer mes pude avanzar más con el documento y los diagramas. En tiempo de desarrollo lo invertí en terminar funciones prioritarias y corregir aspectos de la interfaz. Sin embargo, este mes invertí menos tiempo del estimado porque coincidió con los exámenes.

- **Cuarto mes:**

Tras eliminar el bloqueo que tenía con los diagramas empecé a dedicar más tiempo al documento. Aunque desde el principio del desarrollo intenté seguir la receta para el éxito de *kanban* y enfocarme en la calidad de la entrega, el retraso generado en la creación de los diagramas generó una mala estructuración en algunos elementos. Por lo que en la parte de desarrollo tuve que dedicar tiempo a refactorizar y estructurar el código. También implementé algunas funcionalidades secundarias.

- **Quinto mes:**

Este último mes dediqué gran tiempo al documento para terminar de completarlo y conseguir una buena presentación. Dediqué menos horas a la implementación, para corregir algunos errores, pequeños retoques, correcciones de la interfaz y terminar alguna funcionalidad secundaria.

- **Conclusión:**

Al final dediqué un total de **525 horas**. Trabajé un poco más de lo que pensé inicialmente. Sin embargo, **310 horas** fueron para la implementación, cerca de las 300 horas estimadas. Aunque la estimación con el tiempo real están muy próximos, en realidad no añadí todas las funcionalidades que estimé (pero si las prioritarias). Como tuve que dedicar más horas de las que pensé al análisis y a escribir este documento, en total **215 horas**, no pude añadirlas todas. A pesar de eso, si era algo que tenía muy previsto desde el principio, no estaba seguro a que me daría tiempo a realizar y sabía que seguramente me retrasaría, pero por eso establecí y me centré en las funcionalidades más prioritarias.

## 2.10 Modelo de dominio

En el siguiente diagrama, **Ilustración 19**, se pueden observar los principales elementos y relaciones entre ellos, teniendo en cuenta una visión global y avanzada del sistema.

Observamos dos entidades, *Alumna* y *Monitora*, ambas heredan de la entidad *Usuari*o. Esta entidad representa a un usuario en la aplicación, el cuál debe de tener un nombre, contraseña, email y un ID asignado. Estas características también las tendrán las *Alumnas* y las *Monitoras*, pues son *Usuarios*. Sin embargo, tienen un rol muy diferente. Por un lado, las *Alumnas* tienen el atributo estado, para representar si es una *Alumna* activa o bloqueada. Además, asisten a las clases de dos formas:

realizando una *Reserva* de la *Clase* en directo o accediendo a la información de la *Clase* para ver una grabación.

Por otro lado, las *Monitoras* imparten las *Clases*. Una *Monitora* puede ser administradora o no, ambas imparten las *Clases* pero solo las administradoras se encargan de gestionar a las *Alumnas* y la información de las *Clases*. Cabe destacar que una *Monitora* puede impartir 1 o todas las *Clases* pero una *Clase* tiene asignada una única *Monitor*a.

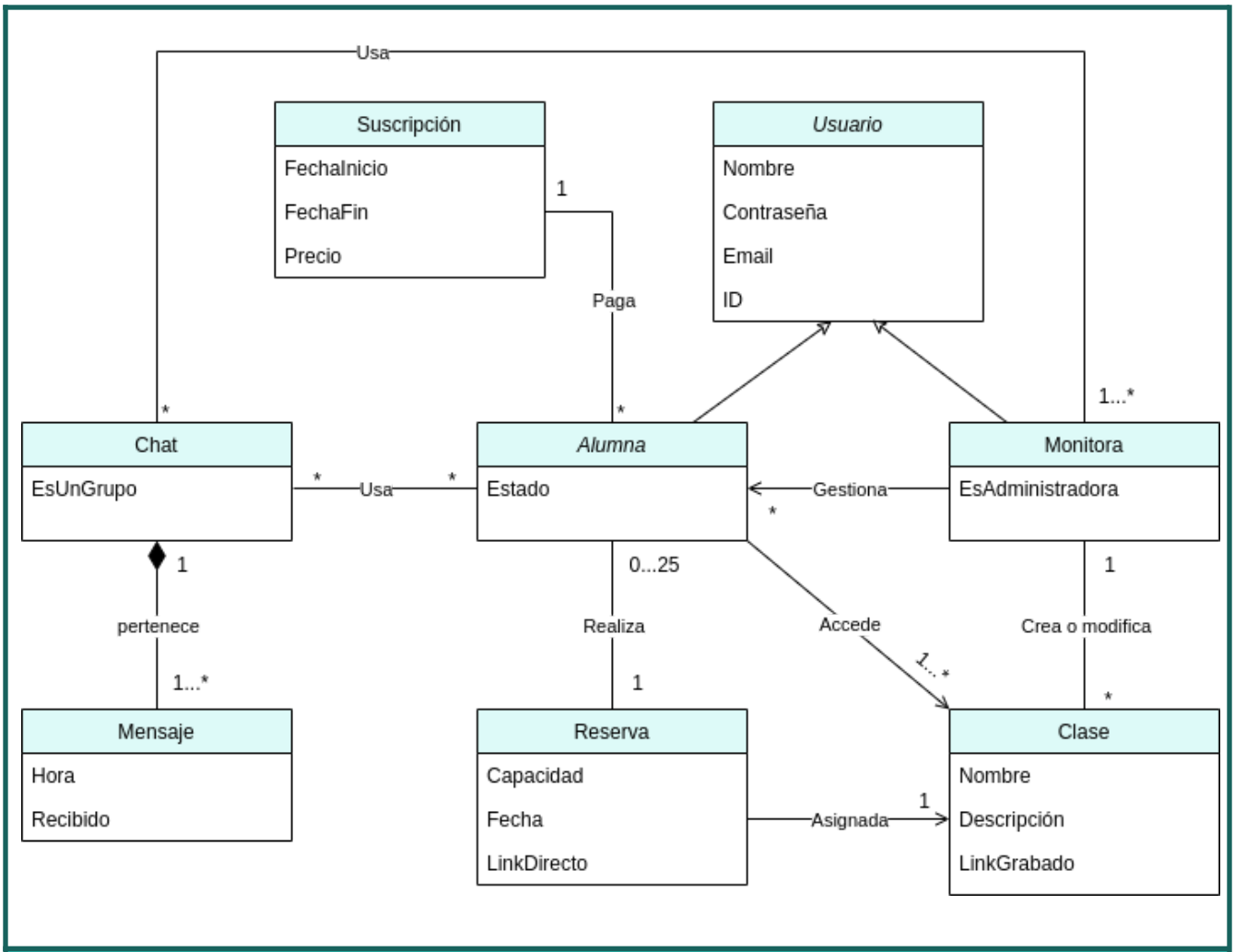



**Ilustración 19: Diagrama modelo de Dominio**

Por último, una *Alumna* paga una *Suscripción* para continuar usando la aplicación (si no, será bloqueada). Con esta suscripción se puede acceder a todas las *Clases* en directo o grabadas. Para una

atención más personalizada, las *Alumnas* y las *Monitoras* pueden usar un *Chat*, este puede ser un grupo o un chat individual. El *Chat* está compuesto por *Mensajes*.

# 3. DISEÑO

En este apartado se verán diferentes diagramas que ayudarán a entender el diseño de la aplicación. Pero antes de eso se va a explicar como funciona internamente Vue.

Sigue el patrón de MVVM (Model-View-ViewModel) **[13]**, **Ilustración 8.** El cúal ayuda a separar la lógica de la aplicación de la interfaz de usuario. Esto permite mayores oportunidades de reutilización de código, y facilita la prueba, mantenimiento y evolución de la aplicación.

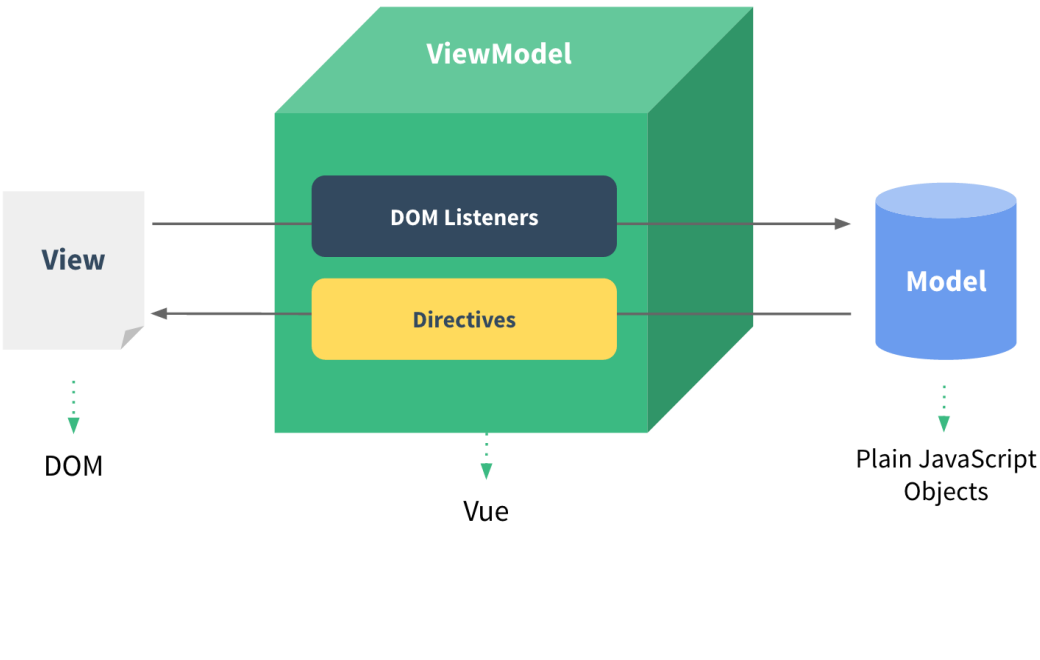



*Ilustración 8: arquitectura Vue*[29]

[29] Fuente: https://012.vuejs.org/guide/

- **View:** Se trata del DOM **[14]** que es controlado por instancias de Vue. Cada instancia de Vue está relacionada con un elemento del DOM. Cuando una vista está compilada, es reactiva a los cambios.
- **View-Model:** Es el objeto que sincroniza la vista (view) y el modelo (model). Cada instancia de Vue es un view-model. Es el objeto con el que los desarrolladores interactúan más a menudo.
- **Model:** Un objeto Javascript. Una vez que es usado dentro de una instancia de Vue, se vuelve reactivo. Cuando cambias su propiedades, la instancia de Vue que esté observando se dará cuenta de esos cambios.

El *view-model* aísla la vista (*view)* del modelo (*model),* permitiendo que evolucionen independientemente. Esto permite que los diseñadores y desarrolladores trabajen de forma independiente y simultáneamente. Los primeros en la vista y los desarrolladores en el modelo y en el modelo de vista.

## 3.1 Diagrama Entidad-Relación

Un diagrama entidad-relación refleja como entidades o conceptos se relacionan entre sí dentro de un sistema. Generalmente se usan para diseñar bases de datos relacionales, pero también puede ilustrarse una base de datos no relacional. Suele ser el primer paso para determinar el alcance y requisitos del sistema.

En este diagrama, **Ilustración 20**, al tratarse de un sistema NoSQL se reflejan los documentos en los que finalmente se materializan las entidades del modelo de dominio. Este diagrama se ha realizado de forma manual, ya que Quasar no cuenta con ODM (Object Document Mapper) **[24]**.

En *allClasses* se guarda la información de todas las clases, incluyendo las clases en directo, su fecha y capacidad. Ya que como cada reserva está asignada a una clase, es más sencillo que todos los datos estén en el mismo documento. En *usersData* se encuentran los datos de todas las alumnas. Luego,

encontramos un documento relacionado con los anteriores, *idClassesXuser*. Este documento guarda los ID de las alumnas y en cada uno de ellos los ID de las clases en las que las alumnas se han apuntado. Esta información se podría haber guardado dentro de *usersData,* sin embargo de esta forma se consigue una velocidad de acceso más directa y rápida. Relacionado también con *usersData* se encuentra la entidad subscription*,* un documento que guarda los ID de las alumnas y dentro de cada ID información de la suscripción. De esta forma se accede a la información de forma más organizada y rápida.

Por último, en *adminData* se guardan los datos de las monitoras, las cuales pueden estar asignadas a varias clases. En el documento *Mensajes* se guardan sus datos y el ID del chat al que pertenece el mensaje. En el *Chat* se guarda su ID, los IDs de las alumnas y de las monitoras que están en ese chat.

En una base de datos relacional se busca la normalización para evitar anomalías. Normalizar a la tercera o cuarta forma normal puede aumentar el coste de acceso para ciertas consultas, a cambio de una mejor estructuración y evitar problemas.

Sin embargo, en una base de datos no relacional esto no es así **[25]**. No hay reglas de normalización que seguir. Los datos debes estructurarlos según las necesidades de la aplicación, para que los accesos más comunes sean los de más rápido acceso. Por ello, una misma base de datos NoSQL para aplicación se puede estructurar de muchas formas distintas. Aunque puede haber algunos consejos para normalizar una base de datos no relacional, puede no ser lo más recomendable para este tipo de sistema. Ya que aquí no importa que los datos estén repetidos, si eso favorece la velocidad de acceso. De hecho, hace varios años el coste de almacenamiento podría ser más importante que el del CPU, pero hoy en día ya no es así. Ahora es más limitado la CPU que la cantidad de datos a almacenar.

Por esto, se ha decidido crear la entidad idClassesXuser. A pesar de que almacena los identificadores de los usuarios y luego los de clases, datos que ya están almacenados en otras entidades. Las clases de cada usuario podrían guardarse perfectamente en usersData, pero se ha pensado que guardarlo de esta otra forma favorece la velocidad de acceso.

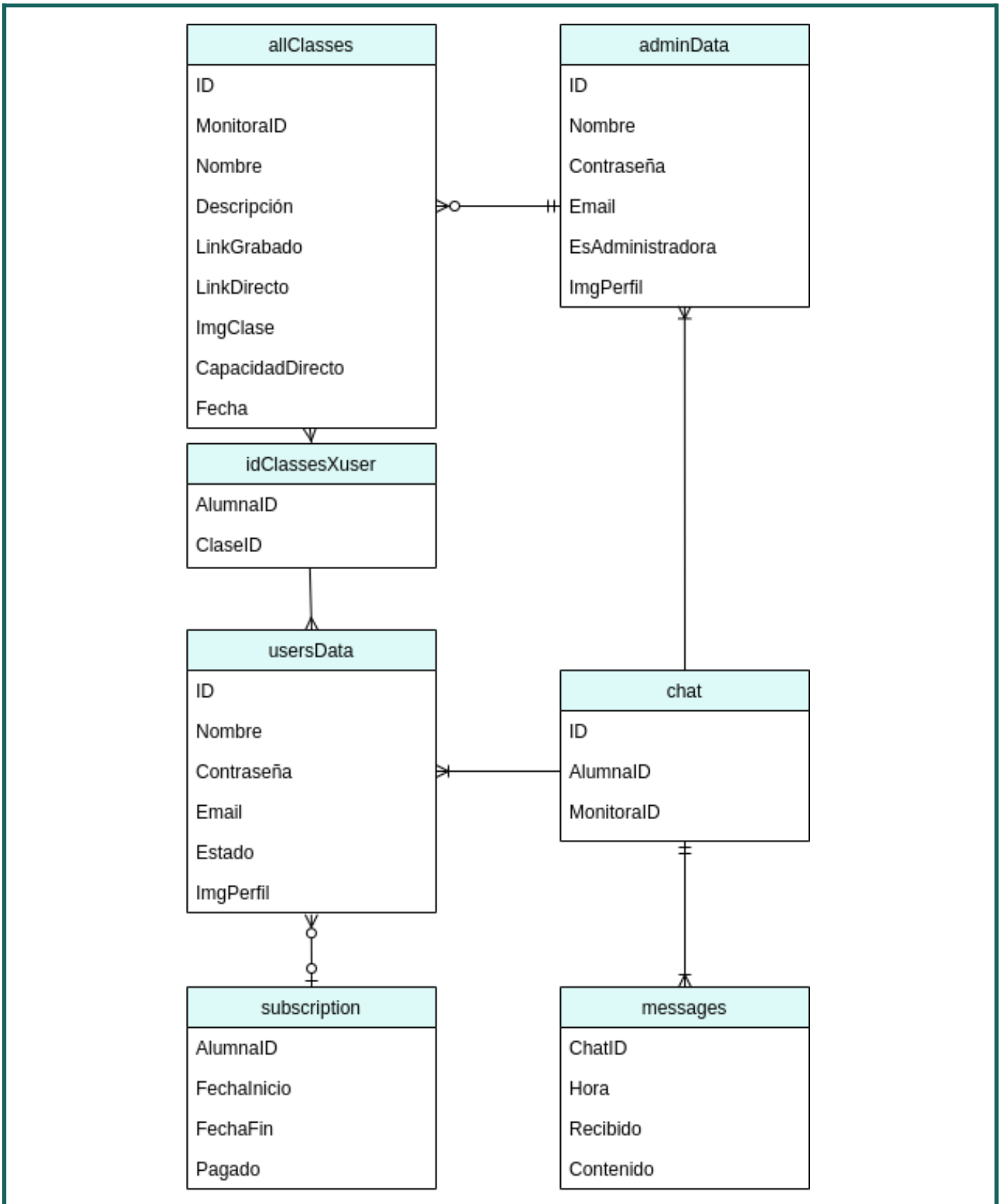



*Ilustración 20: diagrama Entidad-Relación*

## 3.2 Diagrama de clases

Antes de ver los diagramas de clases es necesario saber que una aplicación *Vue* está orientada a componentes. Los componentes son elementos donde se encapsula código reutilizable. Contienen código *html, javascript y css*. Lo importante es que un componente se puede reemplazar fácilmente por otro, además de insertar unos componentes dentro de otros. Esto permite crear una aplicación modularizada y escalable.

He dividido el diagrama de clases en 3 diagramas diferentes para facilitar su visualización. En cada diagrama se puede ver una funcionalidad distinta. Podremos ver clases con el mismo nombre en varios diagramas, esto significa que están usando la misma clase (cuya implementación se traduce con el uso del mismo componente). Esto es posible gracias a poder utilizar un componente en otro y la renderización condicional de *Vue* **[26]**. Aquellas clases en las que aparecen puntos suspensivos es simplemente para simplificar los atributos que aparecen en cada clase, apareciendo solo los que se usen en ese diagrama. Además hay algunas relaciones que he omitido en algunos diagramas pero sí aparecen en otros donde esa relación es importante.

### 3.2.1 Diagrama 1: registro y control de acceso

Este diagrama, **Ilustración 21**, representa las clases que dan la funcionalidad al control de acceso en la aplicación.

En *Quasar* hay un componente principal que se corresponde con la clase *App*, que está compuesta por *RouterView* (el sistema de enrutamiento) y de este "cuelgan" todos las demás clases. En este diagrama puede verse como la primera clase que compone a *RouterView* es *HomeLayout*. Esta clase, y las demás clases que se verán en los otros dos diagramas, y que contienen la palabra *layout,* hacen referencia al componente encargado de mostrar los elementos de navegación principales, que serán diferentes según donde nos encontremos.

Después nos encontramos con las demás clases:

- Clases que hacen referencia a las páginas de la aplicación: *SignUp, LogIn, Home.*
    - *SignUp:* Esta clase hace referencia a la página de registro, donde el usuario puede registrarse si posee el código de registro.
    - *LogIn:* Esta clase hace referencia a la página de inicio de sesión, donde el usuario puede iniciar sesión en su cuenta.
    - *Home:* Esta clase hace referencia a la página de inicio. Es la primera página que se muestra la primera vez que se entra a la aplicación.

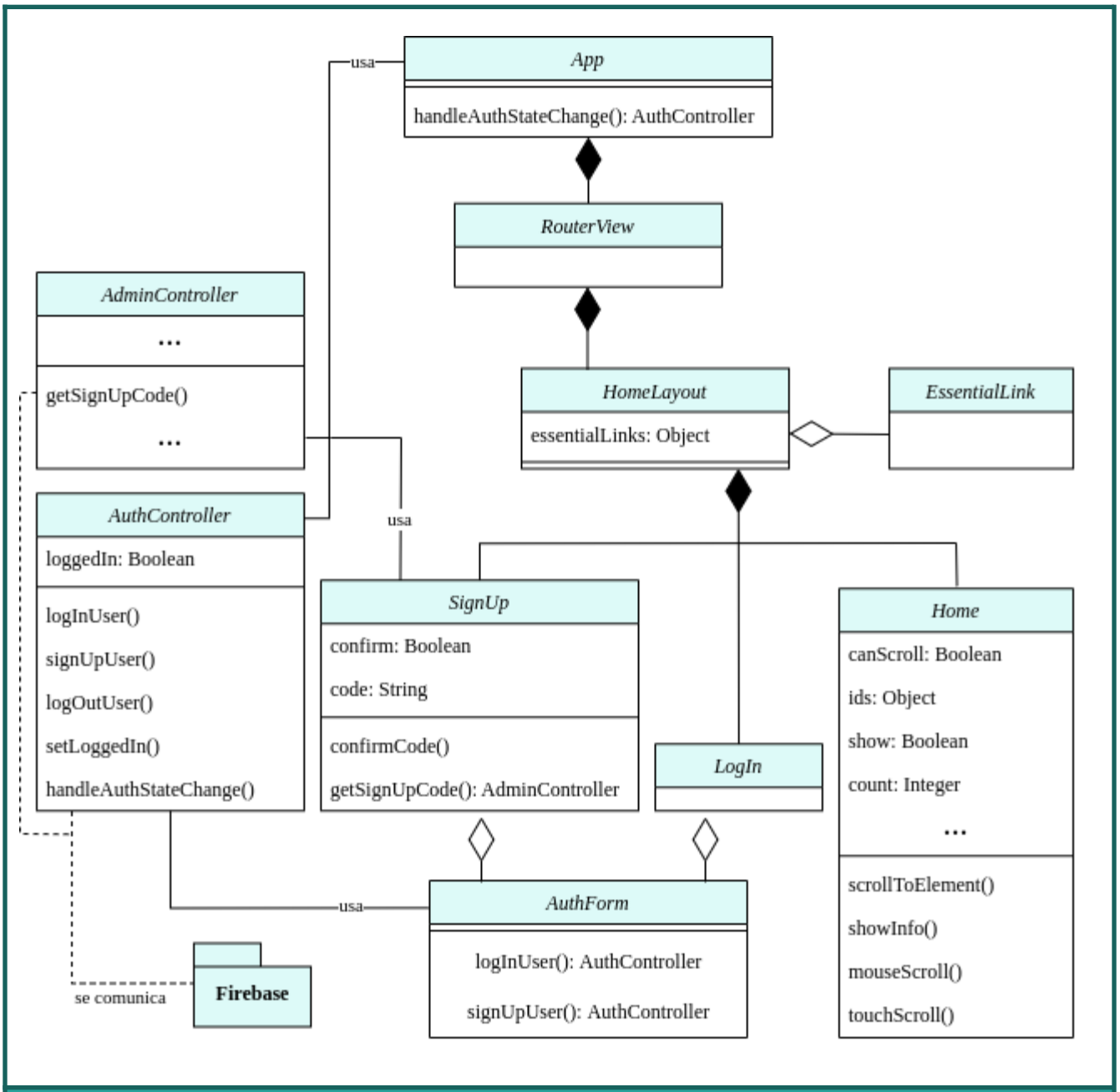



***Ilustración 21: diagrama de clases 1***

- Clases que hacen referencia a componentes reutilizables que se usan en las páginas: *AuthForm* y *EssentialLink.*

- *AuthForm:* Esta clase hace referencia al componente que es utilizado por la página de registro y de inicio de sesión. Contiene el formulario para ambas funcionalidades, gracias a la renderización condicional.
- *EssentialLink:* Esta clase hace referencia al componente usado por las *Layout* y se encarga de establecer los enlaces de navegación.

- Clases que hacen referencia al *Store* de *Vuex*: controlan diferentes funcionalidades y se comunican directamente con *Firebase.* Son AuthController y AdminController.
  - AuthController: Guarda los datos y contiene las funciones necesarias para el control de acceso. Se comunica con Firebase para autentificar o crear nuevos usuarios.
  - AdminController: Guarda los datos y contiene las funciones necesarias para la gestión de los usuarios, edición de clases y de aspectos de control generales de la aplicación. Actúa como controlador general, usado principalmente en las páginas de la administradora.

El *Store* de *Vuex* se verá detenidamente más adelante. Pero se trata de un contenedor reactivo que pueden utilizar todos los componentes de *Vue*. Esto permite almacenar y manipular datos de forma reactiva y controlada.

### 3.2.2 Diagrama 2: sección de alumnas

En este diagrama, **Ilustración 22**, podemos observar todas las clases que forman la sección de las alumnas, es decir, cuando una alumna inicia sesión. La clase *UserLayout* aquí se traduce como el componente de navegación que usarán los usuarios. Donde también podrán cerrar sesión, ver su información y cambiarla.

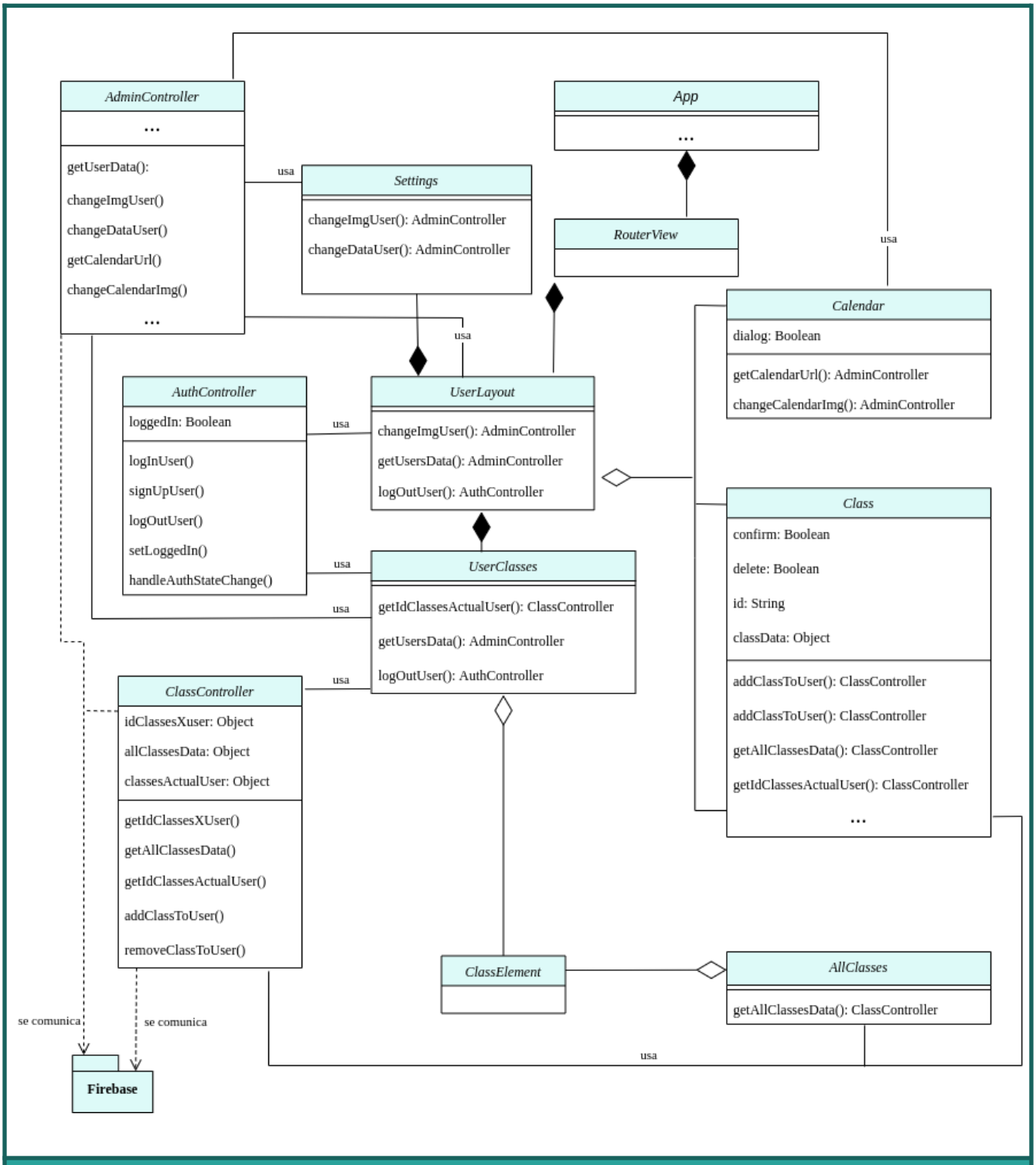



*Ilustración 22: diagrama de clases 2*

Después nos encontramos con las demás clases, solo se mencionan aquellas que no se han visto previamente:

- Clases que hacen referencia a las páginas de la aplicación: *Calendar, Class, AllClasses, UserClasses y Settings.*
    - *Calendar:* Esta clase hace referencia a la página del Calendario, donde el usuario puede consultar el calendario semanal para ver los horarios de forma global.
    - *Class:* Esta clase hace referencia a la página de las clases, donde el usuario puede ver la información de la clase. Apuntarse o desapuntarse de ella y acceder a la clase en directo o grabada.
    - *AllClasses:* Esta clase hace referencia a la página de todas las clases. Aquí el usuario puede ver todas las clases ofertadas.
    - *UserClasses:* Esta clase hace referencia a la página de las clases del usuario. Aquí aparecerán solo las clases en las que el usuario se haya apuntado.
    - *Settings:* Esta clase hace referencia a la página de ajustes. Desde donde el usuario podrá cambiar ciertos aspectos de la aplicación y su información.

- Clases que hacen referencia a componentes reutilizables que se usan en las páginas: *ClassElement.*
    - *ClassElement:* Esta clase hace referencia al componente que es utilizado por varias páginas que muestran las clases disponibles, por ejemplo para ver las clases del usuario. Se trata de un elemento tarjeta que muestra cierta información de la clase y desde el cuál se puede acceder a la información completa de la clase.

- Clases que hacen referencia al *Store* de *Vuex*: controlan diferentes funcionalidades y se comunican directamente con *Firebase.* En este caso, *ClassController*.

- ClassController: Guarda los datos y contiene las funciones necesarias para la gestión de las clases. Se comunica con Firebase para controlar las clases en relación al usuario.

### 3.2.3 Diagrama 3: sección de administración

En este tercer diagrama, **Ilustración 23**, podemos ver todas las clases que forman la sección de administración, es decir, cuando la administradora inicia sesión. La clase *AdminLayout* aquí se traduce como el componente de navegación que usarán los administradores.

Después nos encontramos con las demás clases, solo se mencionan aquellas relevantes:

- Clases que hacen referencia a las páginas de la aplicación: *Student, AdminStudents, SignUpCode, AdminClasses, Class, Calendar.*
  - *Student:* Esta clase hace referencia a la página que muestra la información de una alumna. Desde aquí un administrador puede bloquear o desbloquear su cuenta.
  - *AdminStudents:* Esta clase hace referencia a la página que muestra todas las alumnas registradas.
  - *SignUpCode:* Esta clase hace referencia a la página donde se puede cambiar el código de registro.
  - *AdminClasses:* Esta clase hace referencia a la página donde un administrador puede ver todas las clases y crear nuevas clases.
  - *Calendar:* Esta clase hace referencia a la página del Calendario, donde un administrador puede ver y cambiar el calendario semanal.
  - *Class:* Esta clase hace referencia a la página de las clases, donde un administrador puede ver la información de la clase, borrarla y editarla.

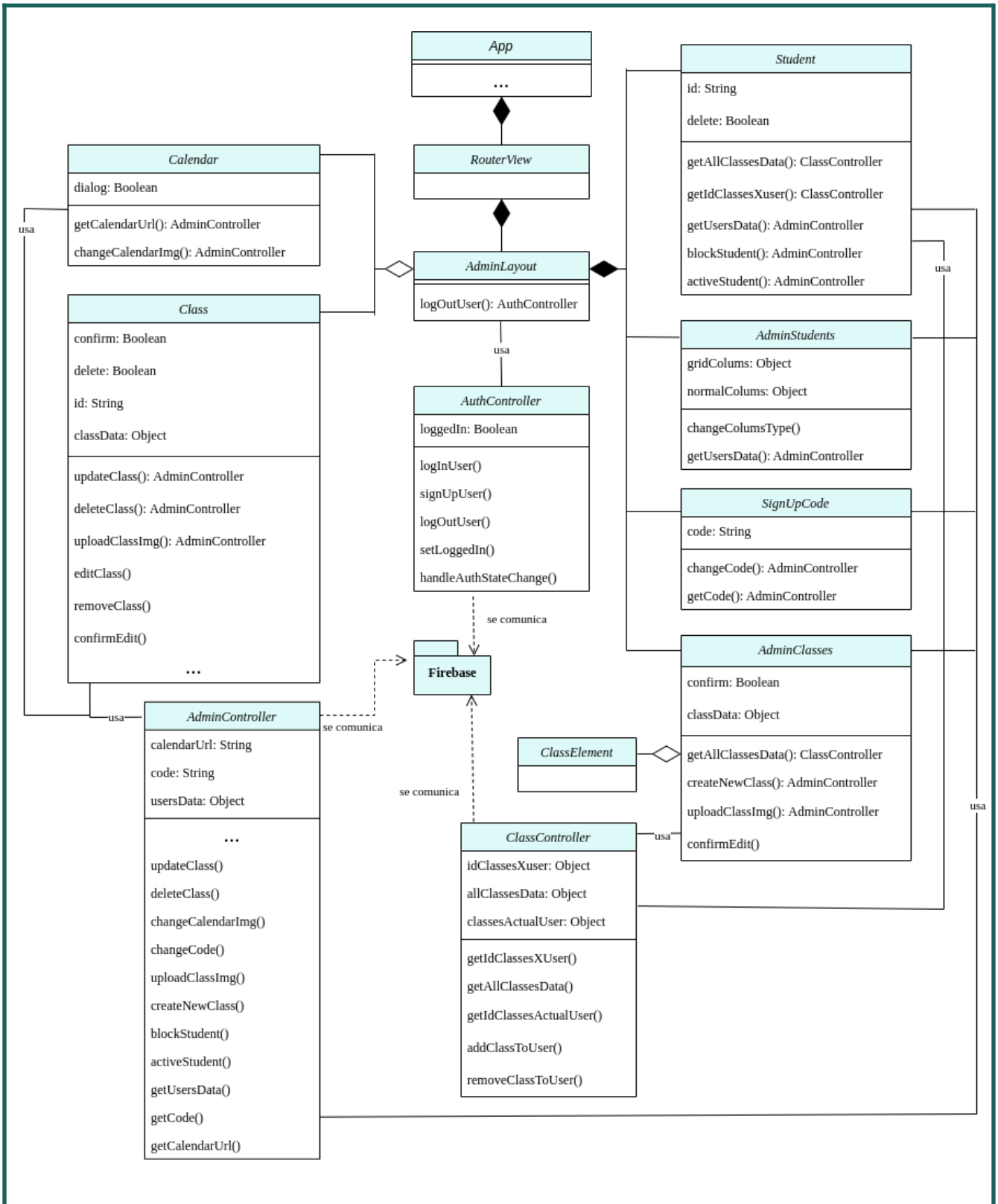



*Ilustración 23: diagrama de clases 3*

### 3.2.1 Diagrama 4: sistema de chat

En este cuarto diagrama, **Ilustración 24**, podemos ver todas las clases que forman el sistema de chat. Como el sistema de chat es común para las alumnas y la administración, se usará el componente *UserLayout* o *AdminLayout* según la sección.

Como se puede observar, hay un nuevo controlador, *ChatController*, que se comunica con *Firebase*, y es usado por *ChatElement* y *Chat*s. *ChatElement* hace referencia al componente que muestra la conversación de un chat en concreto. *Chats* se encarga de mostrar todas las conversaciones (grupos) disponibles según el usuario.

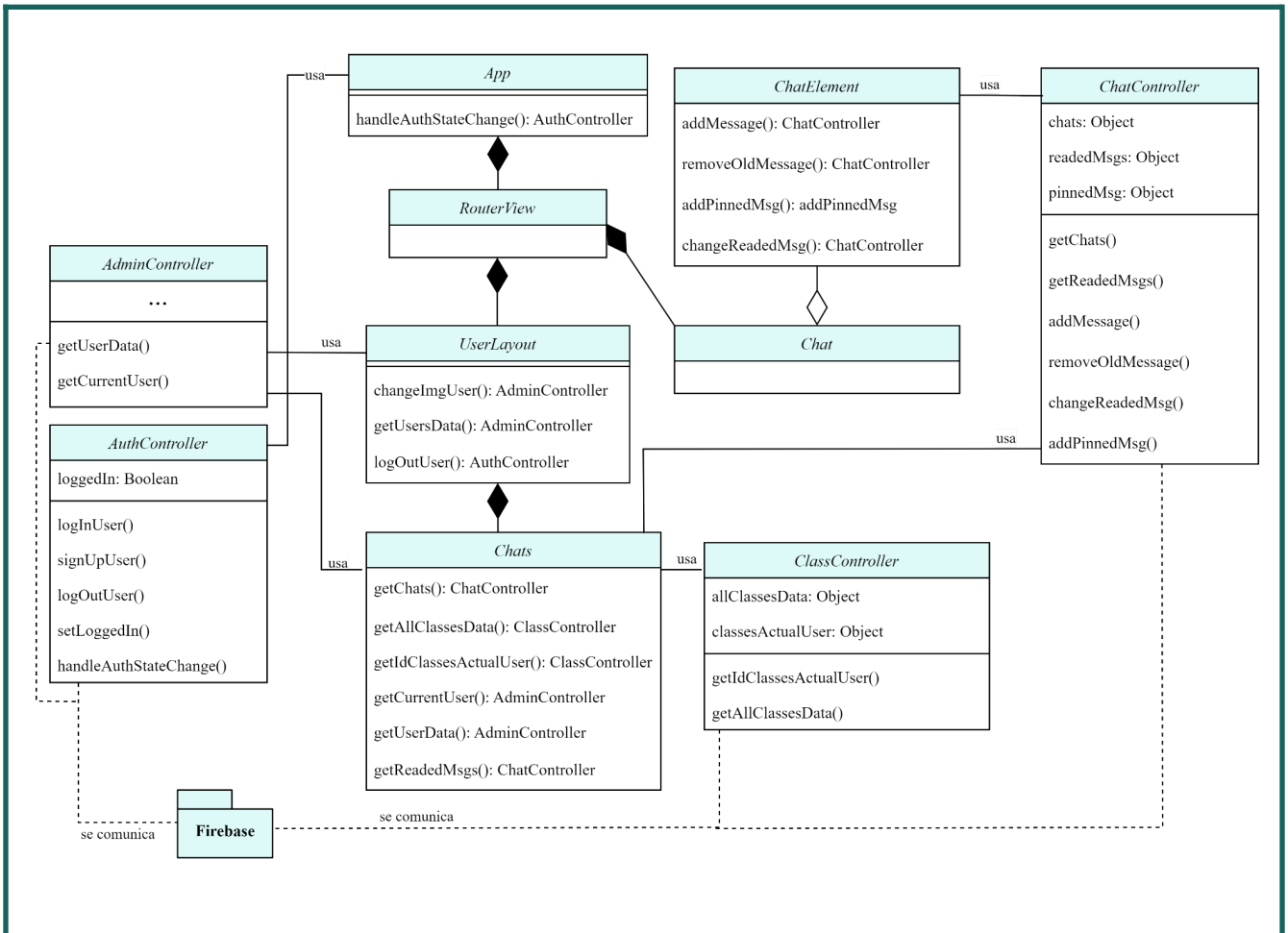



***Ilustración 24: diagrama de clases 4***

## 3.3 Diagrama de secuencia

En este apartado se van a comentar los diagramas de secuencia más relevantes, para así entender mejor el flujo de trabajo. Gracias a los diagramas de secuencia podremos entender mejor los procesos y objetos que coexisten simultáneamente, así como los mensajes intercambiados entre ellos. También es más fácil encontrar errores y entender la lógica de una operación o función.

Como se ha visto en anteriores diagramas, el primer objeto con el que interactúa el usuario es el sistema de enrutamiento, como es algo general en todos, no se va a tener en cuenta. También gracias a que se trata de una aplicación *serverless* cierta lógica del *backend* queda delegada a Firebase, lo que hace al diagrama más simple.

El usuario es representado de color azul, los objetos que representan los servicios y estrucutra de la aplicación se han puesto de color verde oscuro. Mientras que los servicios externos de un verde más claro.

### 3.3.1 Diagrama 1: control de acceso

En este primer diagrama, **Ilustración 25,** se muestra la lógica para iniciar sesión. El usuario interactúa en primer lugar con la vista *SignUp,* y a través de esta utiliza el componente *AuthForm* que se encarga de gestionar y comprobar que los datos sean válidos. Cuando no hay errores, se llama a *AuthControl* el cuál actua como intermediario y se comunica con *Firebase.* Este comprueba si los datos que AuthControl le proporciona son correctos (los que introdució el usuario). Si lo son, manda una respuesta afirmativa a *AuthForm* y este redirige al usuario. Si no lo son, sucede lo mismo pero en vez de redirigir al usuario, se muestra un mensaje de error.

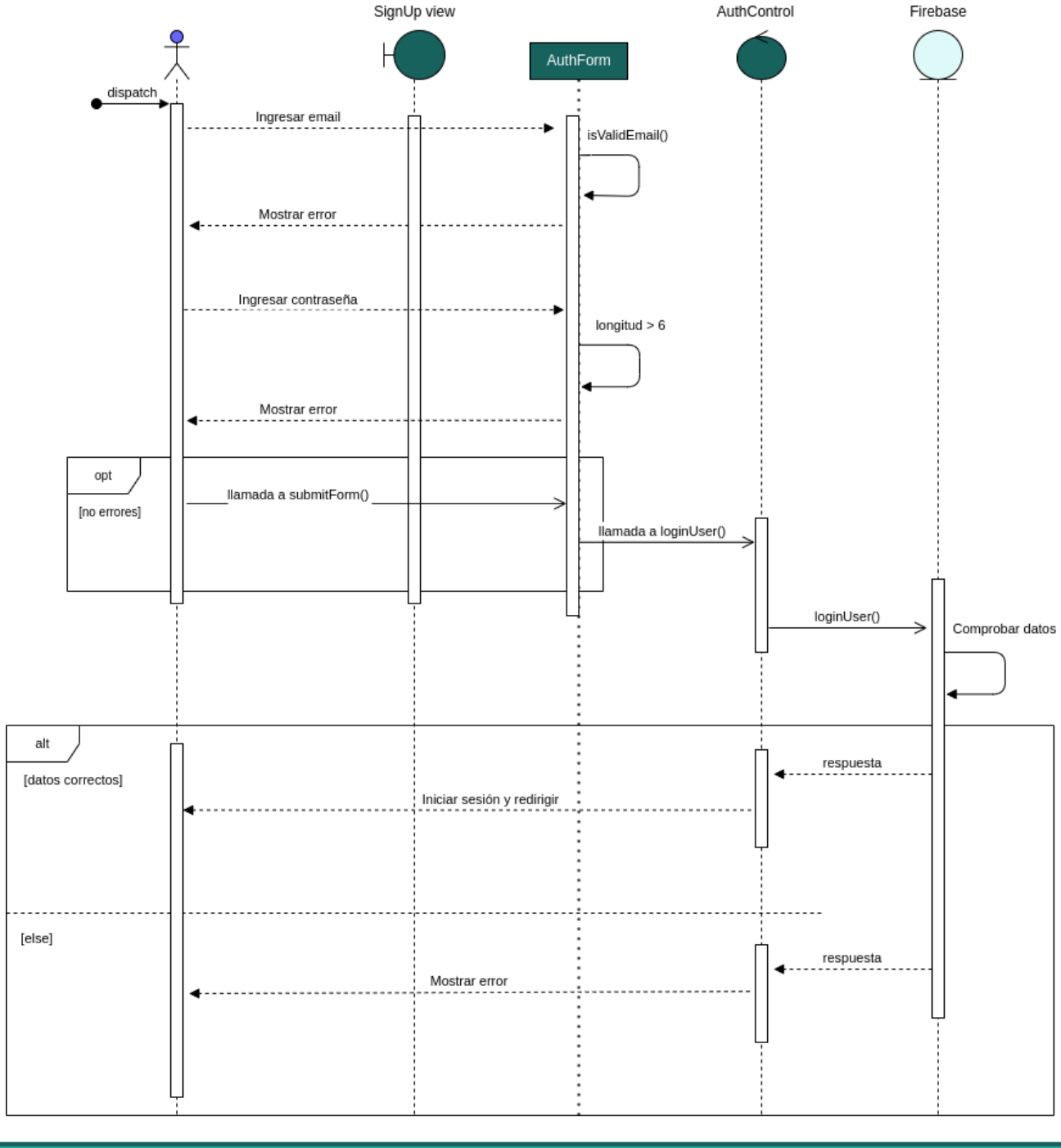



***Ilustración 25: diagrama de secuencia 1***

### 3.3.2 Diagrama 2: sección de alumnas

En este diagrama, **Ilustración 26**, se puede observar el proceso de apuntarse a una clase, así como el funcionamiento del sistema después de que un usuario se conecte.

Antes que nada, el *ClassController* y *Firebase* intercambian información. Básicamente se leen las clases que la base de datos tiene almacenadas y se guardan en el *state* de *ClassController.* Posteriormente, cuando el usuario accede a la página *allClasses*, se obtienen y se muestran todas las clases, mediante la instancia del componente *elementClass*. Seleccionando este elemento, el usuario puede acceder a una clase en concreto. lo que lo redirigirá a la página *infoClass* donde se muestra toda la información al completo de la clase. Una vez aquí puede apuntarse a la clase mediante un botón, el cuál se comunica con el *ClassController* y este con Firebase para reflejar el cambio en la base de datos. Cuando termina esta operación el usuario es redirigido a *allClasses*.

Por otro lado, si accede a *userClasses*, puede ver todas las clases a las que está apuntado gracias al componente *elementClass*. Esta información se obtiene de *ClassController*, gracias a la primera lectura inicial que se realizó. Si hace clic en *elementClass* en está vista, es llevado de nuevo a *infoClass* pero esta vez, en vez de apuntarse, tendrá la opción de desapuntarse. Si lo hace, es redirigido a *userClasses.*

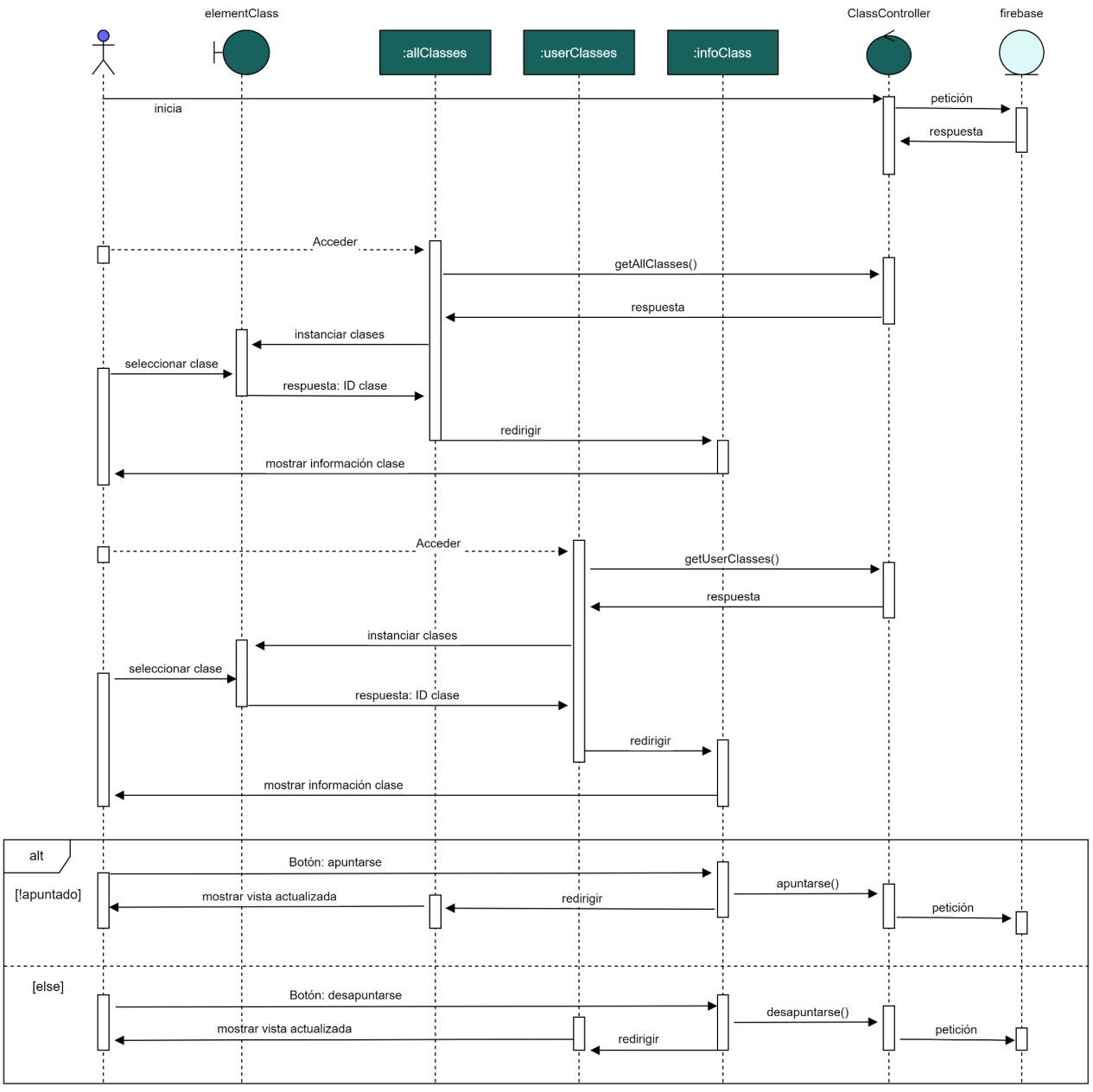



***Ilustración 26: diagrama de secuencia 2***

### 3.3.3 Diagrama 3: sección de administración

En el siguiente diagrama se ve la funcionalidad de la parte de administración. Desde aquí el administrador puede crear nuevas clases, o modificarlas. Al igual que controlar y ver a sus alumnas.

Debido al tamaño de este diagrama se ha dividido en dos partes. En la primera, **Ilustración 27**, la administradora accede a la sección de todas las clases donde puede ver todas las clases creadas. Si decide entrar en una clase, además de ver su información de la misma forma que la vería un usuario corriente, puede editarla tras rellenar un formulario (proceso simplificado en el diagrama) o borrarla.

En la segunda parte, **Ilustración 28**, si no decide entrar a ver la información de una clase puede crear una tras rellenar un formulario. Además también puede acceder a la sección de sus alumnas, donde verá a todas ellas en una tabla. Aquí se mostrará cierta información como su email, nombre, edad, estado. Pero también puede acceder a una de ellas para ver más información y decidir si quiere bloquearla o activarla.

Por último, en cualquier momento también puede decidir por buscar a una alumna a través de su nombre, edad, email, etc. O puede ordenarlas por alguna de estas características. En todos los casos en los que se realiza alguna modificación, interviene *AdminContoller* el cuál se comunica con *Firebase* para que realice el cambio en la base de datos o en el *storage*. Este a continuación le devuelve el estado actualizado a *AdminContoller* para que las vistas se actualicen con la nueva información.

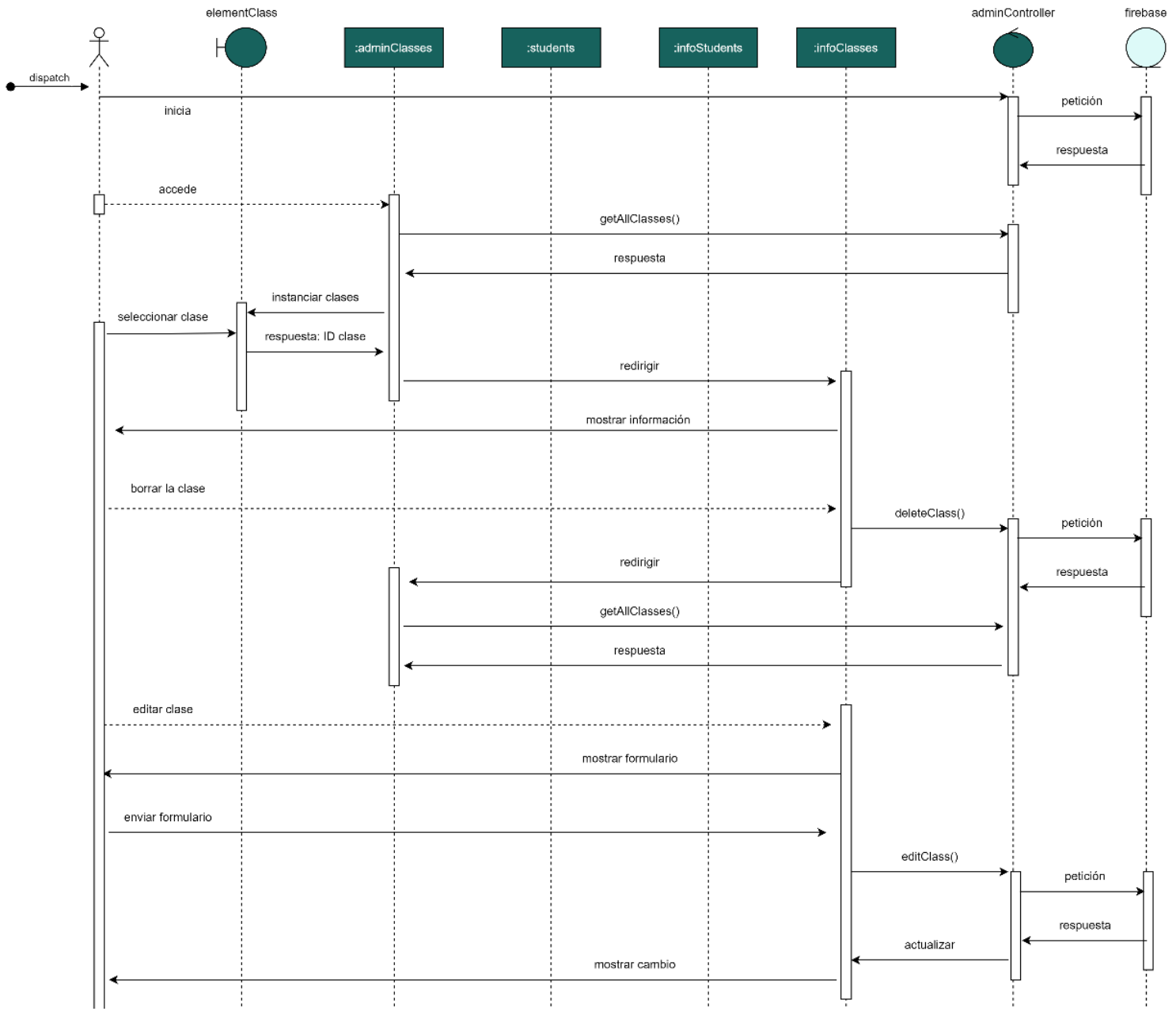



***Ilustración 27: diagrama de secuencia 3-1***

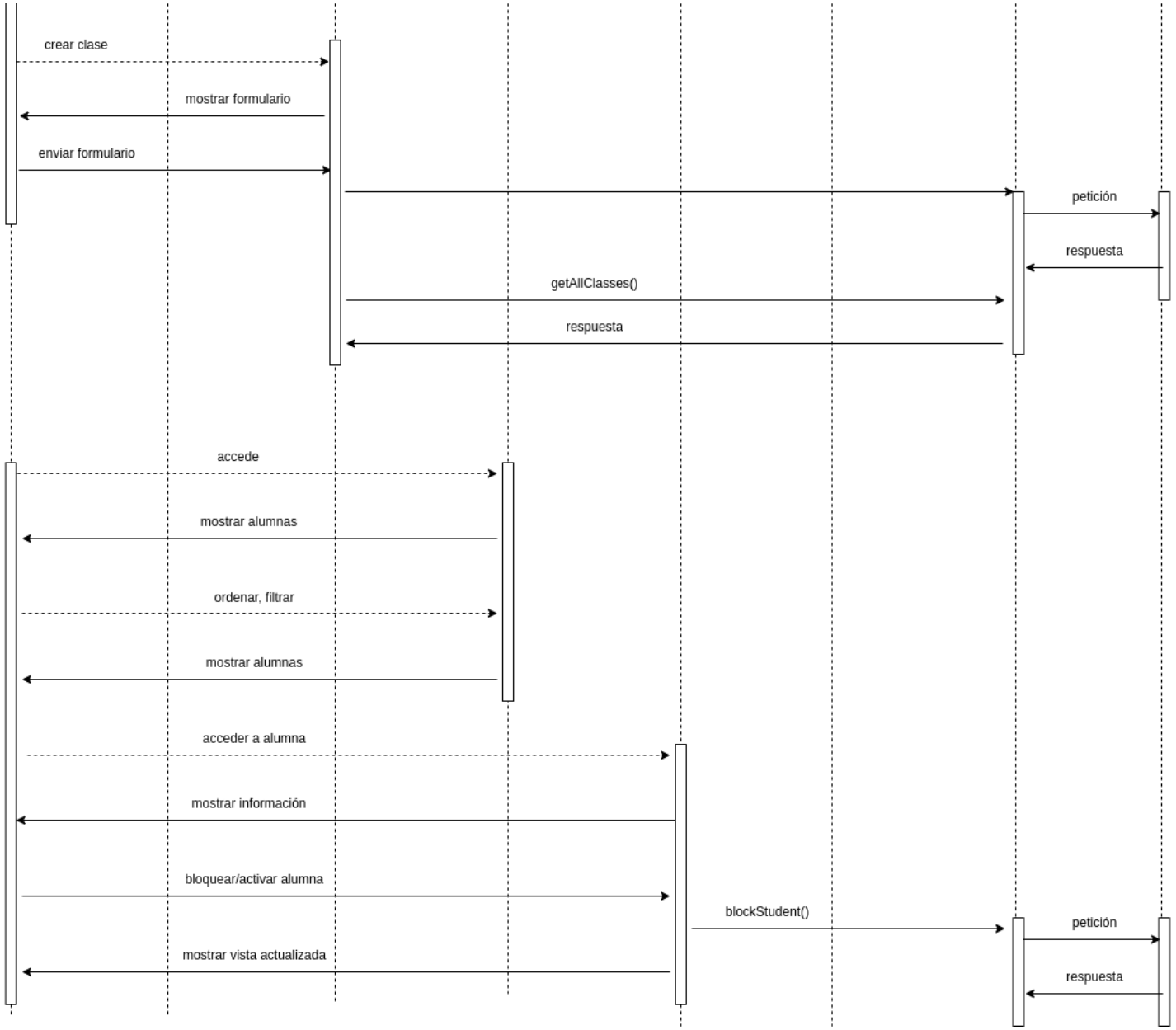
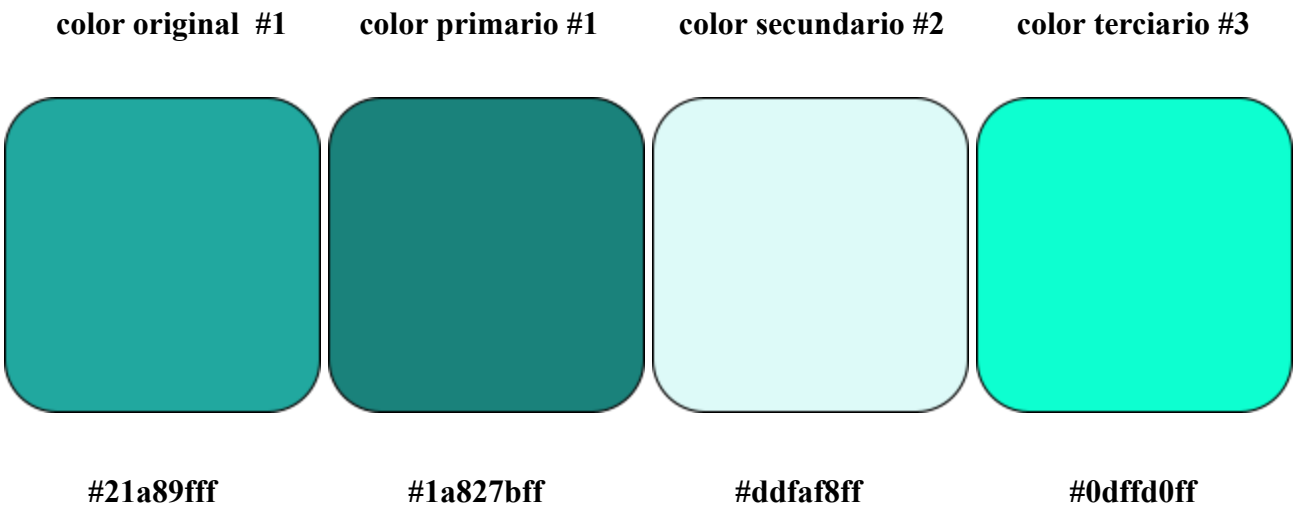



***Ilustración 28: diagrama de secuencia 3-2***

## 3.4 Diseño de la interfaz

### 3.4.1 Colores

Para el diseño de la interfaz se ha usado el programa Figma[30]. Se ha intentado conseguir un diseño UI y UX adecuado, teniendo en cuenta el tipo de usuario que usará la aplicación. Para los colores, se han utilizado los que la cliente ya usaba anteriormente para sus publicaciones en las redes sociales. Sin embargo, tras analizar los colores, el color que se pensó usar como primario no poseía un buen contraste con el blanco. Esto impedía poner texto blanco sobre un fondo de este color, lo que visualmente se ha visto que queda más atractivo y a la cliente le gustó. Por ello, como color primario se ha elegido una versión más oscura del color original.




[30] Enlace: https://www.figma.com

Además de estos colores, se ha usado el blanco y negro no puros. Esto es debido a que el blanco y el negro puro posee un contraste muy alto y el ojo no está acostumbrado a verlo en la realidad, por ello puede cansar la vista más rápidamente.

Otro motivo que puede cansar la vista es la abundancia de colores claros, por ello, se ha seleccionado un color más oscuro para el fondo en el tema oscuro por si se logra añadir la opción de cambiar entre este y el tema claro. También como color de advertencia, para recalcar aquellas cosas importantes se ha pensado utilizar el amarillo. Este color está muy relacionado con el deporte, ya que transmite energía y fuerza. Se ha decidido utilizar un tono no saturado para que sea fácil de ver sin que sobrecargue la vista, además estar muy próximo al verde (color que predomina en la aplicación) para que así armonice mejor con la aplicación.

| **color oscuro #1** | **color alerta #2** | **color negro #3** | **color blanco #3** |
|---|---|---|---|
| **#0F4D48** | **#F9FF72** | **#0F0F0F** | **#FCFCFC** |

La herramienta utilizada para comprobar el contraste y correcta legibilidad del texto e iconos en la aplicación ha sido Adobe Color[31]. Tras revisar todos los colores, el primario se ha usado como fondo para los elementos principales como la barra de navegación o aquellos interactuables como los botones.

[31] Enlace: https://color.adobe.com/es/create/color-contrast-analyzer

Como fondo general se ha usado el color secundario y el color terciario se ha usado como pequeño detalle en alguna interacción para resaltar el elemento.

El contraste del blanco con el color primario tiene una proporción de 4,53 / 1, **Ilustración 29**. Con este contraste el texto podrá verse bien en iconos, títulos y texto pequeño. Aunque está muy cerca de no estar recomendado para texto, ya que el mínimo es un contraste de 4.5 / 1. Sin embargo, esto es para un tamaño de texto de 17px o inferior. Obviamente cuanto más pequeña la fuente peor. Por ello sólo se utilizará un tamaño de fuente mínimo 16 pixeles. Cuando se utilice un tamaño menor (14 píxeles) para elementos no importantes, se pondrá la letra en negrita, lo cual aumentará su contraste (entrando dentro de la sección de texto grande: para 18px y superior o 14px en negrita y superior).


*Ilustración 29: contraste blanco y color primario*

En cuanto al fondo oscuro para el tema oscuro, el contraste es mucho mayor, **Ilustración 30**. Aunque este color se usará como fondo general, y para los elementos interactuables con texto, se seguirá usando el color primario.


***Ilustración 30: contraste blanco y fondo oscuro***

También el color de alerta con el fondo oscuro, **Ilustración 31**, obtiene un alto contraste, de 9,41 / 1. Sin embargo, el color de alerta con el color blanco no podrán utilizarse juntos, ya que obtienen un mal contraste, **Ilustración 32**. Por ello como color de alerta en el tema claro se usará el color oscuro, **Ilustración 33**.

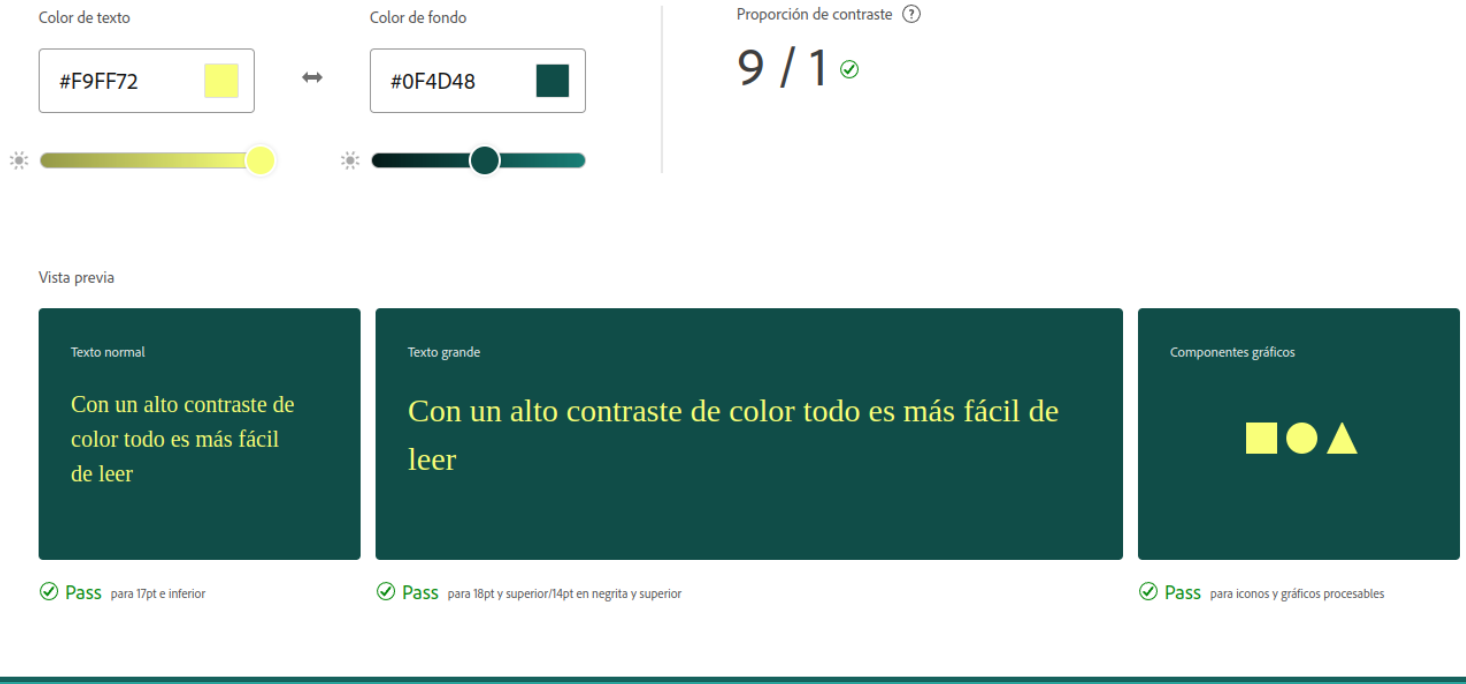



***Ilustración 31: contraste color alerta y fondo oscuro***


***Ilustración 32: contraste color alerta y color blanco***

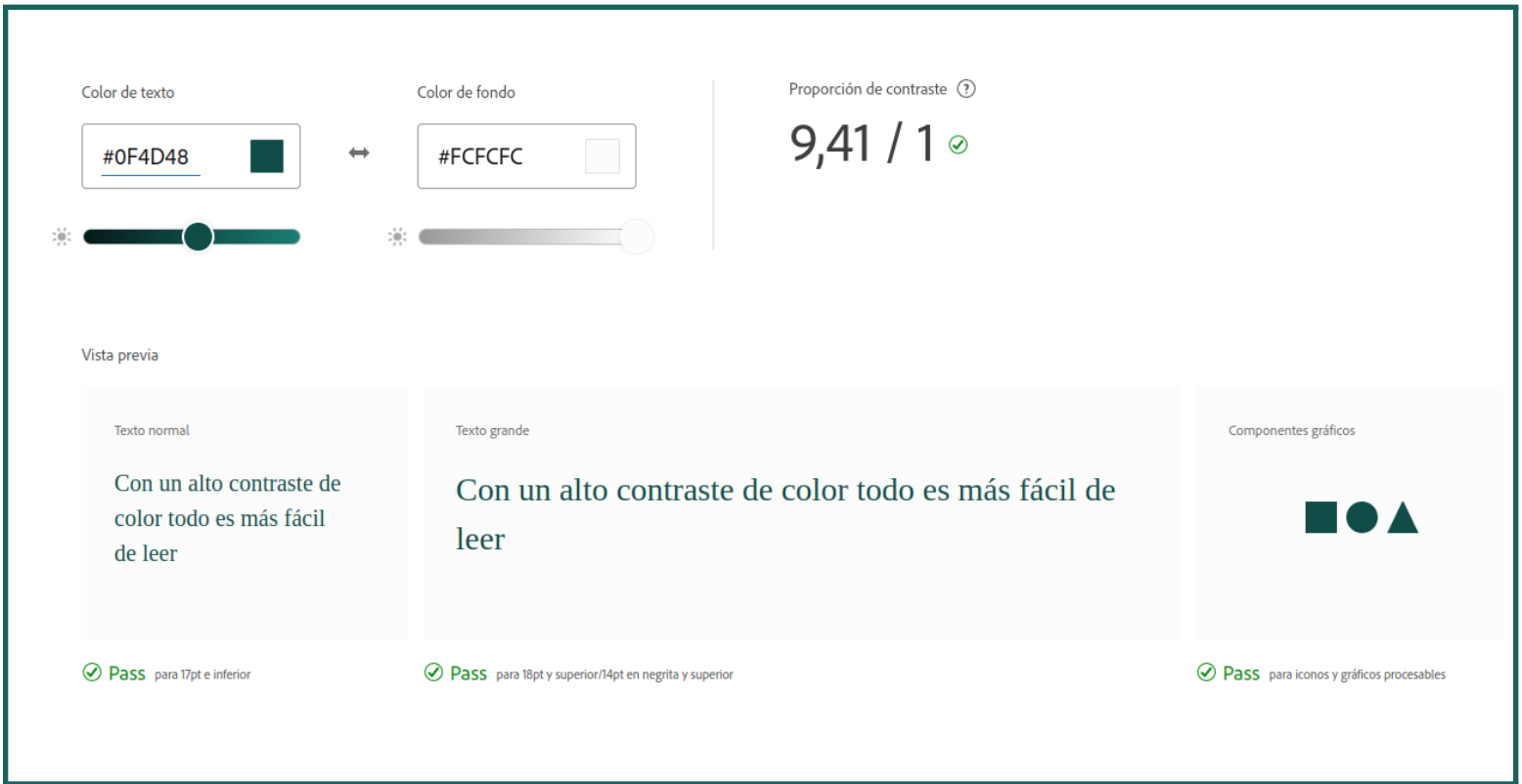



***Ilustración 33: contraste color oscuro y primario***

## 3.4.2 Fuente para el texto

Para el texto se ha utilizado la fuente que la cliente ya tenía elegida. *Glacial* para el texto normal y *Bebas Neue* para los títulos importantes. Se han establecido diferentes niveles de letra según la función del texto:

- 14 píxeles (prioridad en negrita): para textos no importantes, que no se leen a menudo y no son muy largos.
- 16 píxeles: tamaño mínimo y estándar para el texto que se leerá con frecuencia. Este es el tamaño mínimo recomendado para que el texto se pueda leer sin forzar la vista desde una distancia cómoda.
- 18 píxeles: principalmente para elementos interactuables.

- 20 píxeles: para subtítulos.
- 32 píxeles: para títulos.

Estos son los tamaños de letra que se utilizarán principalmente en la aplicación móvil. Pero es posible que dependiendo del texto y la ubicación de este, pueda variar para mejor el diseño y/o legibilidad. Para la aplicación de escritorio son algo mayores.

### 3.4.3 Prototipos

En este apartado se van a ver los prototipos más relevantes de la aplicación. No se muestran los prototipos de todas las vistas ya que algunos son muy similares entre sí o son muy simples.

La **Ilustración 34** corresponde con lo que vería la administradora para gestionar a las alumnas. Cada tarjeta representa a una alumna, contiene su foto, e información de la alumna. Aquellas alumnas bloqueadas aparecerán en amarillo. Al acceder a algunas de estas tarjetas se podrá ver más información de la alumna y realizar algunas acciones.

La **Ilustración 35** corresponde con lo que vería la administradora para controlar las clases actuales. Desde aquí se puede crear una nueva clase. Para editar o borrar la clase es necesario entrar dentro de la clase deseada.

La **Ilustración 36** se corresponde con el chat de la aplicación desde el punto de vista de una alumna. Los mensajes propios aparecen en gris. Los demás mensajes en el color primario. Y los mensajes prioritarios de la adminitradora aparecerían en amarillo.


*Ilustración 34: prototipo gestión de alumnas*

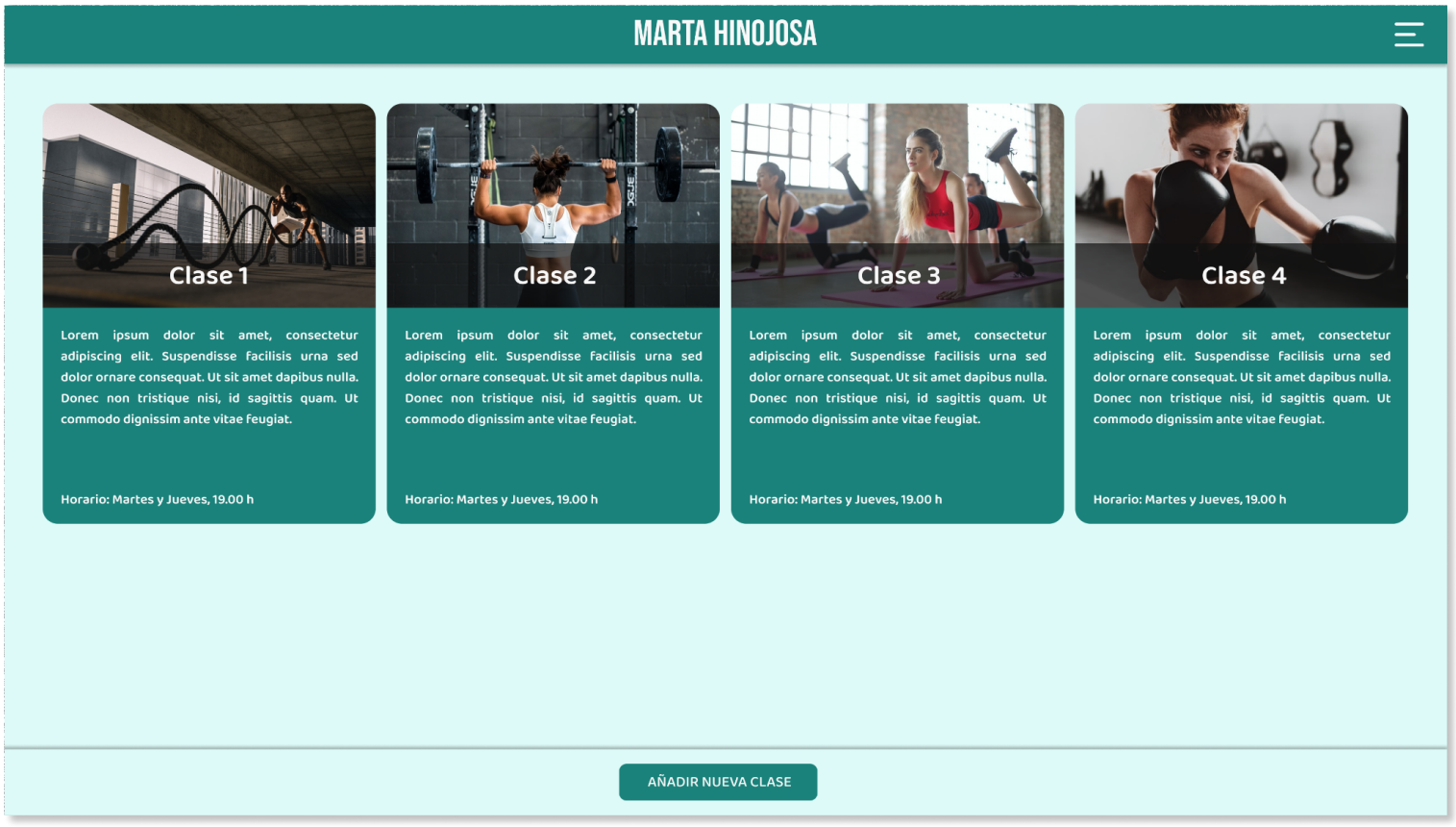



*Ilustración 35: prototipo clases*

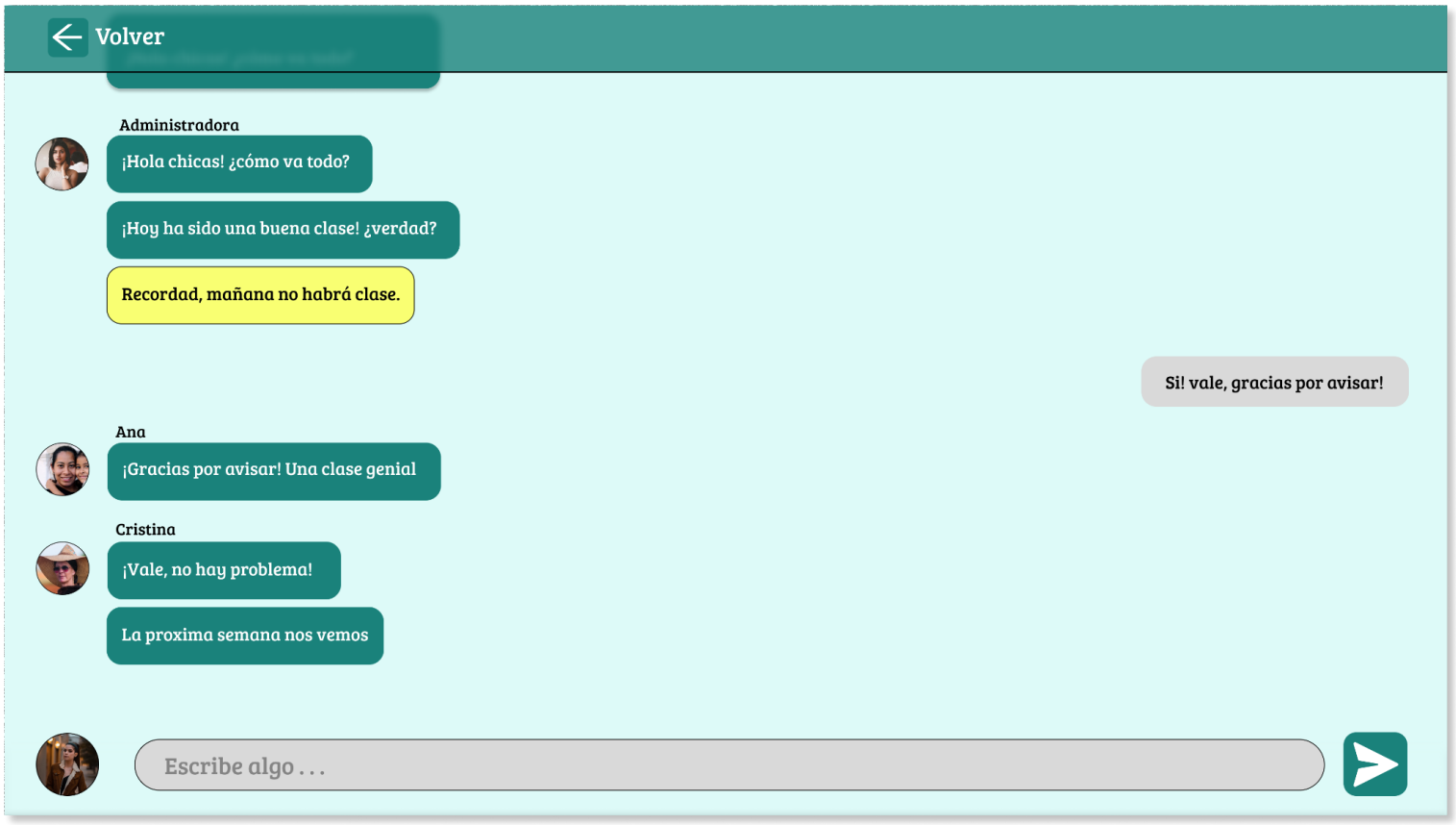



*Ilustración 36: prototipo chat*

Por último, en la **Ilustración 37** se pueden ver los diseños adaptados a dispositivos móviles. En la **Ilustración 38** se pueden apreciar estos diseños en el tema oscuro.

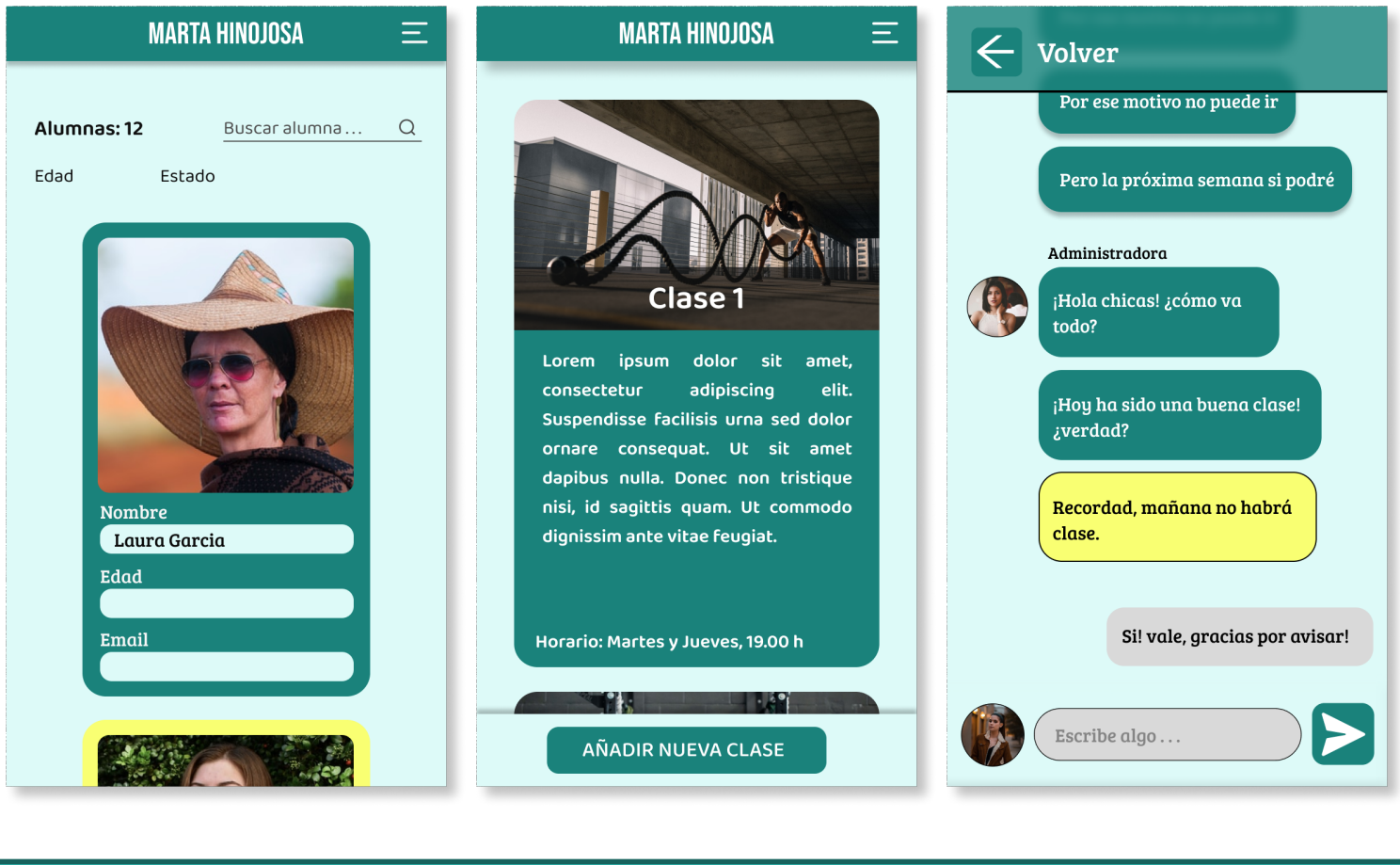



*Ilustración 37: prototipos móvil*

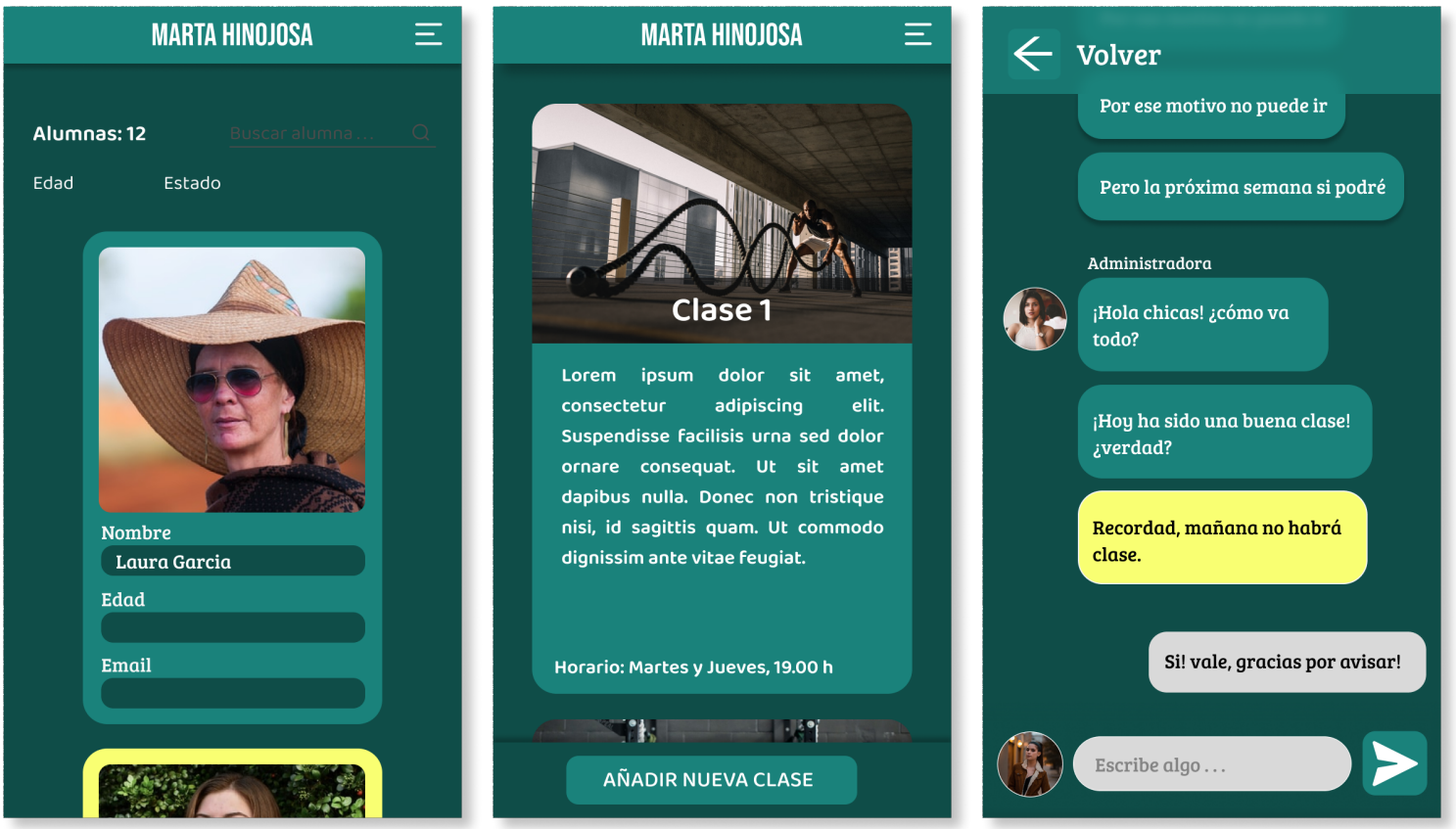



*Ilustración 38: prototipos móvil modo oscuro*

También se ha realizado una prueba para daltonismo. Como se puede ver, **Ilustración 39,** todos los colores se diferencian bien excepto el secundario y el terciario. Por ello el color terciario no será un color que se usará en la parte del usuario.

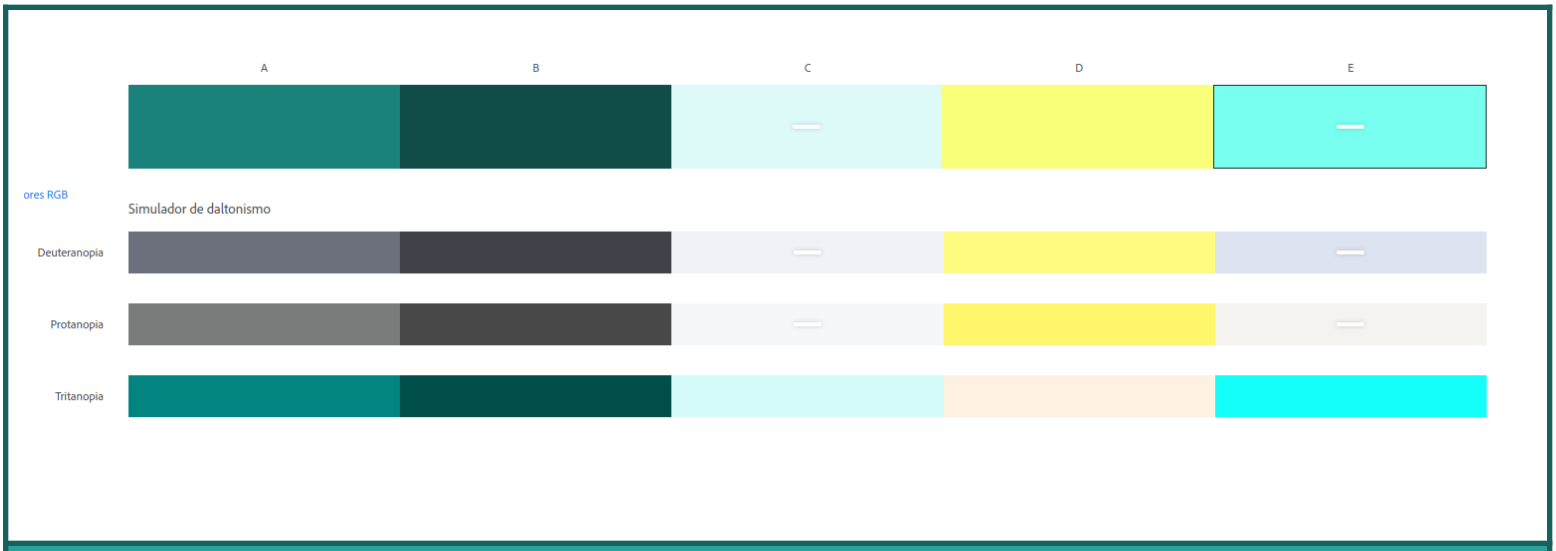



***Ilustración 39: prueba daltonismo***

## 3.5 Plan de Pruebas

Las realización de las pruebas se ejecutarán concurrentemente con el desarrollo del sistema. Se irán probando los criterios de aceptación de las historias de usuario mediante el sistema de caja negra **[27]**, donde se prueba el funcionamiento del software desde el punto de vista de un observador externo. Como la parte del servidor recae en Firebase, hay menos aspectos que son necesarios probar. Sin embargo, un aspecto fundamental es la seguridad. Cuando se escriban las reglas de seguridad, Firebase cuenta con una sección, **Ilustración 40**, donde se pueden probar estas reglas con peticiones de diferente ámbito.

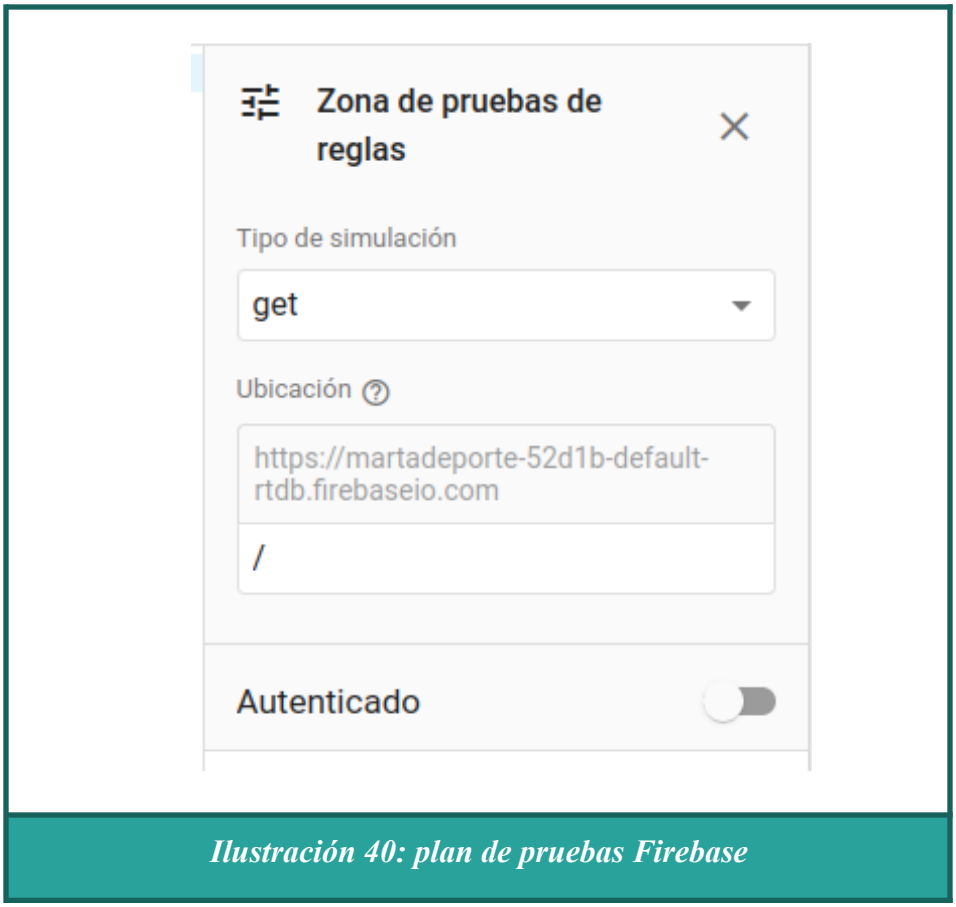



*Ilustración 40: plan de pruebas Firebase*

# 4. IMPLEMENTACIÓN

## 4.1 Arquitectura

En este apartado se va a comentar la arquitectura de la aplicación. En la **Ilustración 4**1 se muestran los diferentes servicios que la aplicación utiliza.

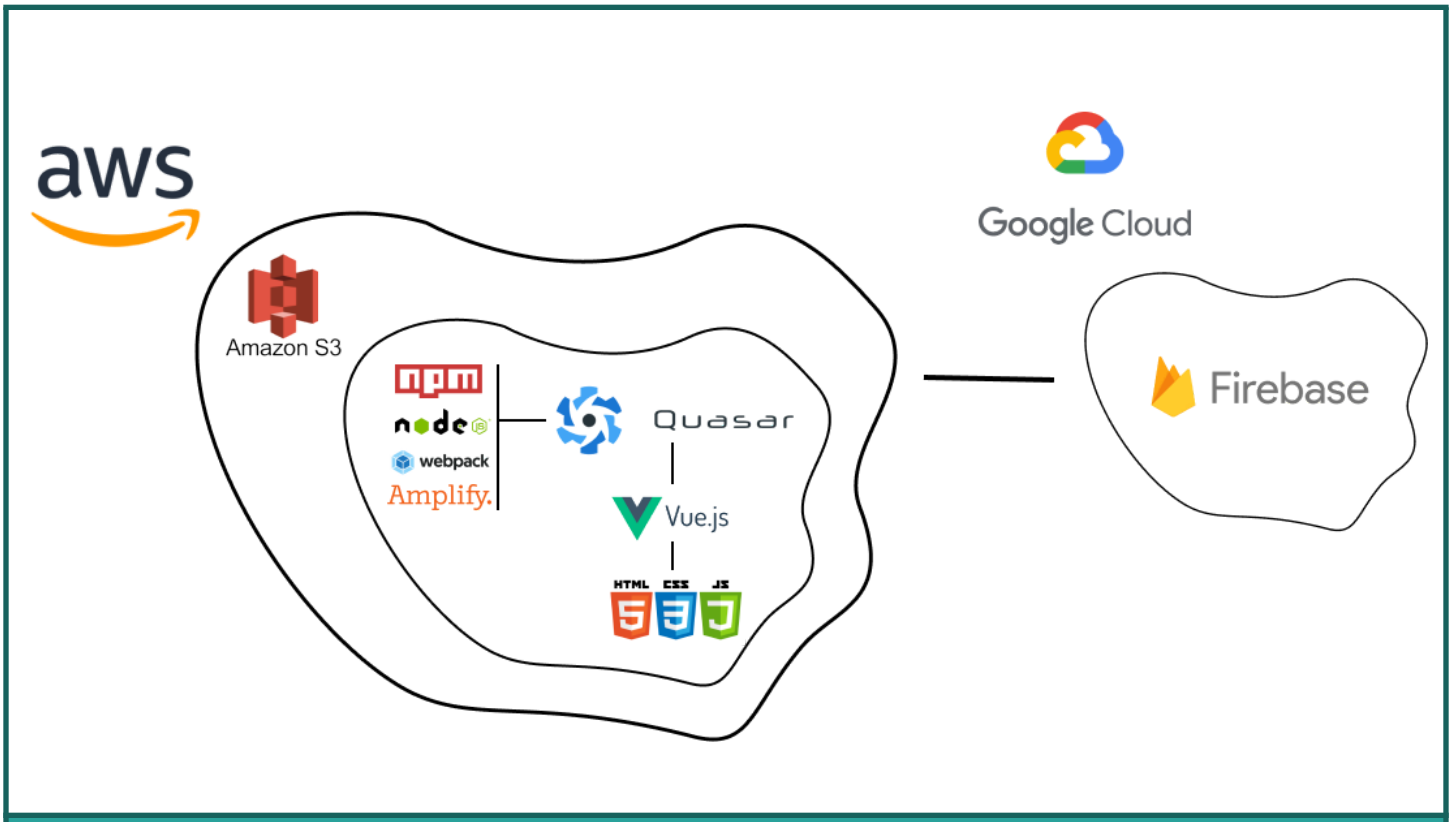



***Ilustración 41: arquitectura app***

Imagen X: Arquitectura App.

En primer lugar, Quasar depende de *node.js* y *webpack*, también es necesario *npm* para la instalación de paquetes adicionales. En segundo lugar, *Quasar* utiliza *Vue.js* como framework para el front-end. Este hace uso de *html, css y js.* Por último, *Firebase* se conecta con *Quasar* a través de *Javascript*.

Como se está usando la base de datos de *Firebase*, la conexión con ella y la lógica de la aplicación se realiza con *Javascript* y recae en la parte del cliente.

## 4.2 Detalles sobre la implementación

En este apartado se van a ver diferentes detalles de implementación, primero se comentará cómo empezar el proyecto, explicando como instalar todo lo necesario. Luego se explicará de forma general cómo funciona los elementos más importantes de quasar. Por último se hablará en más detalle de ciertas características.

### 4.2.1 Instalación

Para empezar el proyecto hay que instalar Quasar **[28]** y después hay que conectarlo con Firebase y su base de datos **[29]**.

- **Quasar CLI**

  Para instalar quasar hay que ejecutar el siguiente comando:

```
$ npm init quasar
```

ó

```
$ npm install -g @quasar/cli (global)
```

Es necesario tener node.js instalado y npm. A continuación para crear una carpeta del proyecto hay que usar el siguiente comando:

```
$ quasar create <folder_name>
```

Habrá que responder a varias preguntas para la configuración inicial del proyecto. Por último, para poner en funcionamiento la aplicación hay que usar este comando:

```
$ quasar dev
```

- **Firebase**

Primero hay crear el proyecto en la página web de Firebase[32] y luego instalar el módulo para la conexión con Firebase, con el siguiente comando:

```
$ npm install firebase
```

Después hay que crear una carpeta llamada boot en *src*/, dentro hay que crear un archivo para la conexión con Firebase. Antes de esto, en *quasar.conf.js* hay que indicar que ese archivo es un archivo boot para que se inicie al comienzo de la aplicación:

```
boot: [
   'firebase'
 ],
```

[32] Enlace: https://firebase.google.com/

Por último, en *src/boot/firebase.js* añadimos el siguiente código para realizar la conexión:

```
// Import the functions you need from the SDKs you need

import { initializeApp } from "firebase/app";



// Your web app's Firebase configuration

// For Firebase JS SDK v7.20.0 and later, measurementId is optional

const firebaseConfig = {

   //

};



// Initialize Firebase

const appFirebase = initializeApp(firebaseConfig);



export { appFirebase }
```

Los datos que hay que poner en el objeto *firebaseConfig* son proporcionados por Firebase cuando se crear el proyecto en la página web. También hay que realizar la exportación de la constante *appFirebase* para poder acceder a ella en los demás archivos.

### 4.2.2 Estructura inicial

- **Directorios**

Al crear el proyecto aparecen diferentes carpetas, creando una estructura de directorios **[30]**:

-*quasar/*: podemos encontrar archivos Javascript del framework Quasar.

-*node_modules/*: diferentes plugins y librerías instalados, ya sea por Quasar o por nosotros mismos.

-*public/*: assets públicos y estáticos.

-*src/*: directorio principal de trabajo.

-*assets/*: assets dinámicos, procesados por webpack[33].

-*components/*: archivos Vue para ser usados en páginas y layouts.

-*css/*: css y variables globales.

-*layouts/*: archivos Vue que actúan como layouts para ser usados en diferentes páginas.

-*pages/*: archivos Vue donde se escribirán las diferentes vistas de la aplicación.

-*boot/*: archivos boot, código de inicialización de la aplicación.

-*router/*: Sistema de rutas de Vue. Permite cargar todas las páginas de una sola vez cuando el usuario entra a la aplicación, evitando así recargas de la aplicación y permitiendo una navegación fluida.

-*store/*: sistema de almacenamiento de vuex[34]. Es un almacenamiento centralizado para todos los componentes de la aplicación con reglas que aseguran que la información solo cambie de forma adecuada.

---

[33] Enlace: https://webpack.js.org/

[34] Enlace: https://vuex.vuejs.org/

-*quasar.conf.js/*: archivo Javascript de configuración de Quasar.

Ahora que se ha visto la estructura de los directorios, se va a hablar más en detalle de la estructura de ciertos archivos.

- **Componentes y páginas**

Los componentes y páginas tienen la misma estructura para poder crear archivos html+css+js. Pero se diferencia en varias cosas: las páginas serán los archivos por los que el usuario navegará, por lo tanto estarán incluidos en el sistema de rutas de Vue. Los componentes estarán insertados dentro de estas páginas para añadir ciertos elementos.

Muchas veces es común tener varios elementos iguales dentro de la aplicación, ya sean botones, tablas, tarjetas, etc. Vue usa estos componentes como "funciones" para evitar tener que repetir el código. Además esto no solo permite evitar la redundancia de código, sino que también estos componentes son personalizables a través de parámetros que nosotros mismos podemos crear, de esta manera podemos cambiar los textos, valores o estilos cuando sea necesario. Además de, claro está, pasar valores obtenidos de la base de datos. La comunicación entre componentes se verá en detalle más adelante.

Lo interesante ahora es conocer como son la estructura de estas páginas y componentes. En un solo archivo permiten tener código HTML, CSS y Javascript:

```
<template>
 <div>
   // código HTML
 </div>
</template>
```

```
<script>
  // código JS
</script>

<style lang="scss" scoped>
  // código CSS
</style>
```

Como se puede ver en el código de arriba, la página está dividida en 3 grandes etiquetas: template, script y style.

El primero de ellos corresponde al código HTML, dentro de la etiqueta <*template*> debe de haber solo una etiqueta padre. Esto quiere decir que si se escribe varias etiquetas dentro de la etiqueta <*template*> el código no funcionará. Por ello es recomendado escribir una etiqueta <*div*> y dentro de ella ya escribir las demás etiquetas. En esta sección es donde se insertan los componentes de Vue.

En la segunda etiqueta se escribe el código Javascript. Aquí también es donde se importan diferentes componentes, se realiza la comunicación con el store de vuex, se escriben diferentes funciones o se hace uso del ciclo de vida de Vue **[31]**.

Por último, en la etiqueta <*style*> se escribe el código css. Aquí mediante parámetros se puede especificar que preprocesador usar, por ejemplo sass o scss. También se puede especificar si los estilos son aplicables solo a esta página (recomendado) o son globales, gracias al parámetro *scoped*.

Los componentes tienen la misma estructura que las páginas como ya se ha mencionado anteriormente, esto tiene sentido ya que al final son lo mismo, código HTML, JS y CSS. Sin embargo, es necesario definir ese componente cuando vaya a ser usado. Esto se hace en la etiqueta <script> de la página donde se desee usar:

```
<script>

export default {

   components: {

       'test' : require('components/TestComponent.vue').default

   }

}

</script>
```

De esta forma, para usarlo solo será necesario escribir el siguiente código HTML:

```
<test/>
```

- **Layouts**

El layout es un componente de Quasar que permite encerrar el código de las páginas dentro de una estructura navegable como puede ser una barra lateral o un header. Esto se puede configurar al gusto del usuario pero existe un constructor de layouts[35] que permite rápidamente diseñarlas de forma gráfica y obtener el código necesario para su implementación.

Como se trata de un archivo vue, su estructura es igual a las páginas o componentes, pero al crear una ya viene con el código para cambiar fácilmente sus estilos, y la funcionalidad de navegación. Además está creada con un diseño responsable.

Se pueden crear tantas layouts como se estime conveniente, pero todas las páginas pueden ser encerradas en una sola layout. Por lo tanto solo será necesario crear más de una layout si el código

[35] Enlace: https://quasar.dev/layout-builder

entre ellas difiere demasiado. Por ejemplo, para diferenciar el header del inicio de una página web del que podría tener un administrador.

- **Route**

Todas las páginas que sean accesibles deben estar declaradas en el sistema de rutas. Se trata de un array de objetos con una nomenclatura específica:

```
const routes = [
 {
   path: "/",
   component: () => import("layouts/MainLayout.vue"),
   children: [
     { path: "", component: () => import("pages/Index.vue") },
     {
       path: "/page1",
       component: () => import("pages/Page1.vue"),
     },
   ],
 },
 {
   path: "/admin",
   component: () => import("layouts/AdminLayout.vue"),
   children: [
     {
       path: "/page1",
       component: () => import("pages/PageAdmin1.vue"),
```

```
    },
    {
      path: "/page2",
      component: () => import("pages/PageAdmin2.vue"),
    },
  ],
},
```

Como se puede observar, dentro del array podemos encontrar dos objetos diferentes. Esto es así para indicar que rutas usan cada layout, ya que en este ejemplo hay un layout principal y otro para el administrador. Con el atributo *path* se indica la ruta principal, luego se importa el layout a usar y por último se crea un array llamado *children* para especificar las diferentes rutas que habrá. Aquí cada nuevo objeto representa una ruta, por ello es necesario definir el *path* y que página usará.

- **Store**

Vue Store nos permite almacenar datos, obtenerlos, modificarlos y realizar acciones. Estos datos se guardan de forma local pero son accesibles desde cualquier parte del código. Para empezar a usar Vue Store solo necesitamos un archivo Javascript. Sin embargo, hay una forma más organizada de hacerlo, que es dividiendo por carpetas los diferentes módulos. Por ejemplo, en una carpeta solo se guardan los datos necesarios para el inicio de sesión, y en otra carpeta los necesarios para otro tipo de utilidad, y así sucesivamente.

Dentro de cada carpeta hay varios archivos Javascript. En total 5: acciones, getters, index, mutaciones y el state.

**-State:** Aquí se escriben las variables que se desean almacenar.

**-Getters:** Devuelve el valor de una variable del state.

**-Mutations:** Función para modificar el valor de una variable del state.

**-Actions:** Realizan alguna acción.

**-Index:** Para reunir todos los archivos en uno solo y ser exportados como un módulo.

De esta forma se crean los diferentes módulos que deberán ser declarados en un archivo Javascript principal que irá situado en *store/index.js*.

```
export default store(function (/* { ssrContext } */) {
 const Store = createStore({
   modules: {
     module1,
     module2,
   },
 })
 return Store
})
```

En el objeto modules se declaran los módulos que vayan a usarse, pero antes deben ser declarados de la siguiente manera:

```
import module1 from './module1Store'
import module2 from './module2Store'
```

## 4.2.3 Desarrollo del proyecto

- **Inicio de sesión**

Para realizar el inicio de sesión primero se ha creado una carpeta llamada *Auth* en *src/components/*. Dentro de la carpeta *Auth* se crea el archivo *LoginForm.vue*, en él se encuentra el HTML del formulario de inicio de sesión y los métodos necesarios.

Cada div tiene su propia clase indicando si es una fila o columna, así como su margen[36]. Esto es parecido a Bootstrap y sirve para establecer una correcta posición de los elementos. Así, sin tocar el css, se pueden colocar los elementos y adaptarse para diferentes dispositivos. Aunque algunas adaptaciones serán necesarias más adelante.

Si continuamos leyendo el código podemos ver el input del nombre. Aquí hay varias cosas que comentar, la propiedades v-model, y lazy-rules.

```
<form @submit.prevent="submitForm" class="formAuth">
  <div class="row flex justify-center text-center">
    <h3 class="title col">{{ title }}</h3>
  </div>
  <div class="row q-mb-lg" v-if="signUp">
    <q-input
      v-model="formData.name"
      class="col"
      placeholder="Nombre Completo"
      type="text"
      lazy-rules
      ref="name"
      :rules="[(val) => val.length > 0 || 'Introduce un nombre']"
    >
      <template v-slot:prepend>
        <q-icon name="person" />
```

[36] Enlace: https://quasar.dev/style/spacing

```
      </template>
    </q-input>
  </div>
```

Con lazy-rules podemos usar reglas para comprobar un valor y hacer algo en consecuencia. En este caso, se comprueba si el email es válido, y si no lo es se muestra el mensaje: "Introduce un email válido" | `:rules="[ val => isValidEmail(val)|| 'Introduce un email válido.']"`.

Con v-model="formData.email" es una puerta de doble sentido. Hace referencia a la variable email del objeto formData. Este objeto está creado en la etiqueta <*script*> del componente. Es necesario encerrar el objeto dentro una estructura llamada *data()* y realizar un *return* para que el objeto funcione correctamente.

El v-model está ligado al input del email, cuando el usuario cambia su valor, la variable email se actualiza a la vez. También cuando esta variable cambia por algún otro motivo, por ejemplo por un método, el valor que el usuario ve en el input también cambiaría al instante. A continuación se puede observar el objeto *formData*:

```
data() {
  return {
    formData: {
      email: "",
      password: "",
      password2: "",
      age: "",
      name: "",
    },
  };
},
```

En este archivo también está la función *submitForm()*. Esta función valida el formulario y llama a la acción loginUser en el módulo Auth del store de Vuex para iniciar sesión. La función es llamada en

el parámetro del form con `@submit.prevent="submitForm"`, gracias al modificador prevent se consigue que la página no se recargue cuando se pulsa el botón de iniciar sesión. Por último en la página *Login.vue* se llama al componente *LoginForm.vue*.

Ahora la página que se llamará para el inicio de sesión será la de *Login.vue*, que cargará el componente *LoginForm.vue*. La utilidad de hacerlo así es que este componente se puede reutilizar en un futuro, para por ejemplo la página de registro.

Por último, a continuación se pueden ver algunas de las acciones creadas para el inicio de sesión:

```
//Log in to the application with an email and password. If the data does not
//exist in firebase, a dialog is displayed informing about the error.
export function logInUser({ }, payload) {
   signInWithEmailAndPassword(firebaseAuth, payload.email, payload.password)
       .then(response => {
           //execute code after login
       })
       .catch(error => {
           showDialog("Usuario o Contraseña incorrecta.");
       })
}

//Register a user in firebase.
export function signUpUser({ }, payload) {
   createUserWithEmailAndPassword(firebaseAuth, payload.email,
payload.password)
       .then((userCredential) => {
           // Signed in
           const user = userCredential.user;

           const usersData = ref(firebaseDB, "usersData/" + user.uid);
           onValue(usersData, (snapshot) => {
               let profileImages = {
                   '0': "urlexample",
```

```
                    '1': "urlexample",
                    '2': "urlexample",
                }

                // A post entry.
                const postData = {
                    age: payload.age,
                    birthday: payload.day+"/"+payload.month+"/"+payload.year,
                    email: payload.email,
                    id: user.uid,
                    name: payload.name,
                    state: true,
                    url: profileImages[Math.floor(Math.random() * 3)],
                };

                const updates = {}
                updates['usersData/' + user.uid] = postData;
                update(ref(firebaseDB), updates);
            })
        })
        .catch((error) => {
            showDialog(error.message);
        });

}

//It is triggered each time the user logs in or logs out (or reloads the page).
//If logged in,the profile page is loaded and the loggedIn variable is updated
//to true.
export function handleAuthStateChange({ commit, dispatch }) {
   onAuthStateChanged(firebaseAuth, user => {

        if (user) {
            let adminID = rootGetters['adminController/getAdminID'];
            commit('setLoggedIn', true);
            LocalStorage.set('loggedIn', true); //Variable stored locally in the
```

```
client. It is used to guarantee the privacy of the routes.
          if (user.id == adminID) {
              this.$router.push('/admin/students');
          } else {
              this.$router.push('/userClasses');
          }
          //The action fbReadData is called
          dispatch('classController/fbReadData', null, { root: true })
          dispatch('adminController/fbReadDataUsers', null, { root: true })
          dispatch('chatController/fbReadDataChats', null, { root: true })
          dispatch('chatController/fbReadlocalMsgs', null, { root: true })

      } else {
          commit('setLoggedIn', false)
          LocalStorage.set('loggedIn', false);
      }
  })
  dispatch('adminController/fbReadCode', null, { root: true })
}
```

Se pueden observar 3 funciones. La primera permite el inicio de sesión gracias a la función *signInWithEmailAndPassword()* que firebase nos proporciona. Los datos necesarios para el inicio de sesión son pasados como payload a la función.

La segunda función registra a un usuario en firebase. Se usa la función createUserWithEmailAndPassword() que Firebase proporciona, esto crea un usuario en el sistema de autentificación de Firebase. Además se crea un nuevo usuario en la base de datos con el payload pasado por parámetro. También se añade una foto aleatoria de perfil de 3 disponibles.

La tercera, se activa cuando el usuario inicia, cierra sesión o recarga la página. Si se ha iniciado sesión se manda al sistema de rutas la página profile. También se actualiza la variable *loggedIn* creada anteriormente en el *State* y se llama a una acción de otro módulo para leer las clases deportivas. Como se puede ver, se realiza un *commit* llamando a la mutación previamente creada para modificar el valor

del *State*. Luego se accede a *LocalStorage* poniendo la variable *loggedIn* a verdadero. Esta variable no es la misma que la del *State* aunque se le ha dado el mismo nombre. *LocalStorage* permite almacenar datos de forma local en el cliente aún cuando este ha abandonado la página. De esta forma cuando la página se pone en marcha, comprueba este valor para ver si el usuario está logueado o no y proteger así las rutas.

Tras crear un usuario en la base de datos de Firebase, **Ilustración 42**, se guarda usando como llave su identificador. Pero no se guarda ninguna información sobre su contraseña, ya que esto lo gestiona el sistema de autenticación de Firebase.


***Ilustración 42: users Firebase Realtime***

Para acabar, mencionar que para proteger las rutas, de un usuario autenticado o no, se ha creado un archivo boot con el siguiente código:

```
export default boot(({router}) => {
  router.beforeEach((to, from, next) => {
   let loggedIn = LocalStorage.getItem('loggedIn');
```

```
  if(!loggedIn && (to.path == '/home' || to.path == '/signUp'  || to.path ==
'/error')) {
    next();
  }else if(!loggedIn && to.path !== '/login'){
    next('/home');
  }else{
    next();
  }

  if(to.path == '/') next('/home')

})
```

Donde se obtiene el valor del LocalStorage de la variable *loggedIn* y se realizan diferentes comprobaciones para permitir, o redirigir el paso.

- **Clases deportivas**

En esta sección se comentará como los alumnas se registran en las clases y acceden a su información, también se verá como la administradora puede crear o editar estas clases. Para todo esto solo han hecho falta 4 archivos.

El primero es un componente, *ClassElement.vue*. Se trata de la "portada" de la clase, es una tarjeta que muestra una imagen, el título, descripción de la clase y si se hace click se redirige a la página de la clase correspondiente. En la **Ilustración 43** se pueden apreciar dos instancias de este componente, cada una con datos diferentes.

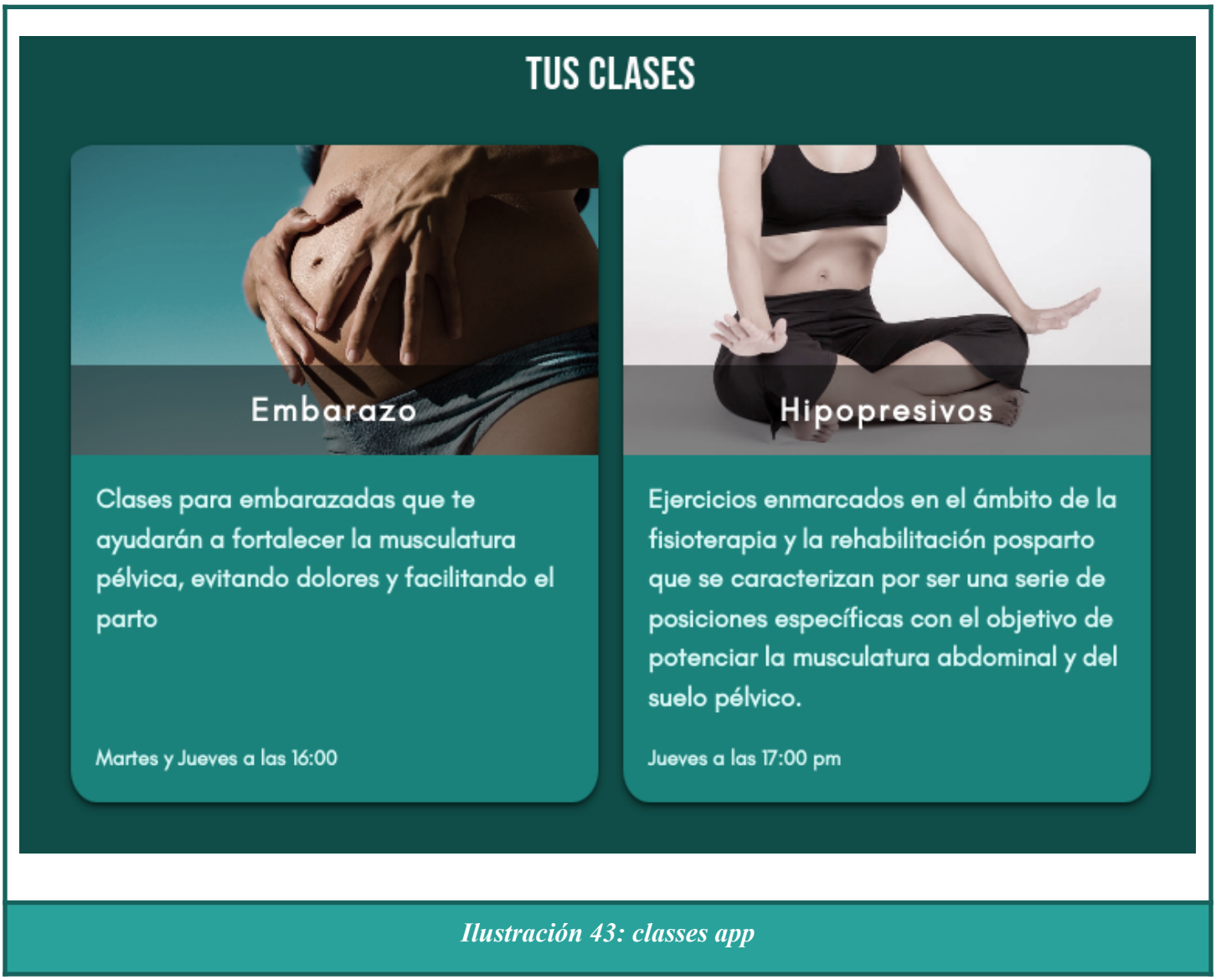



***Ilustración 43: classes app***

Que se muestren datos diferentes es posible gracias a la comunicación entre archivos Vue. Esto se hace con la propiedad *props*. Dentro del componente se indica esta característica y se establece un nombre a los datos que se pasarán:

```
props: ["data", "ID"],
```

En este caso un objeto data que contendrá los datos de la clase, y el ID de la clase. De esta forma, para luego mostrar la información de la clase en el componente, simplemente se accede a este objeto:

```
<p class="description">
  {{ data.text }}
</p>
```

Ahora pasaremos a ver el segundo y tercer archivo, *Admin-classes.vue* y *allClasses.vue.* El primer archivo corresponde a la página que el administrador ve, donde se instancian el componente de las clases, y además desde donde puede añadir nuevas clases. El segundo es donde las alumnas pueden ver todas las clases disponibles, con instancias del componente de las clases. Como se puede ver, en común comparten la instancia del primer componente mencionado:

```
<classElement
  v-for="(data, key) in allClasses"
  :key="key"
  :ID="key"
  :data="data"
></classElement>
```

Este componente se muestra tantas veces como el tamaño del objeto *allClasses*, el cuál es obtenido de la base de datos de Firebase y contiene todas las clases creadas en tiempo real. Por último con los identificadores *:ID* y *:data* se hace referencia a los *props* creados anteriormente en el componente para pasar los valores deseados.

*Class.vue* es el cuarto archivo y contiene la información de clase al completo, se trata de otra página que se accede a ella cuando se hace click en el componente *ClassElement.vue*. Su diseño se puede apreciar en la **Ilustración 44**.

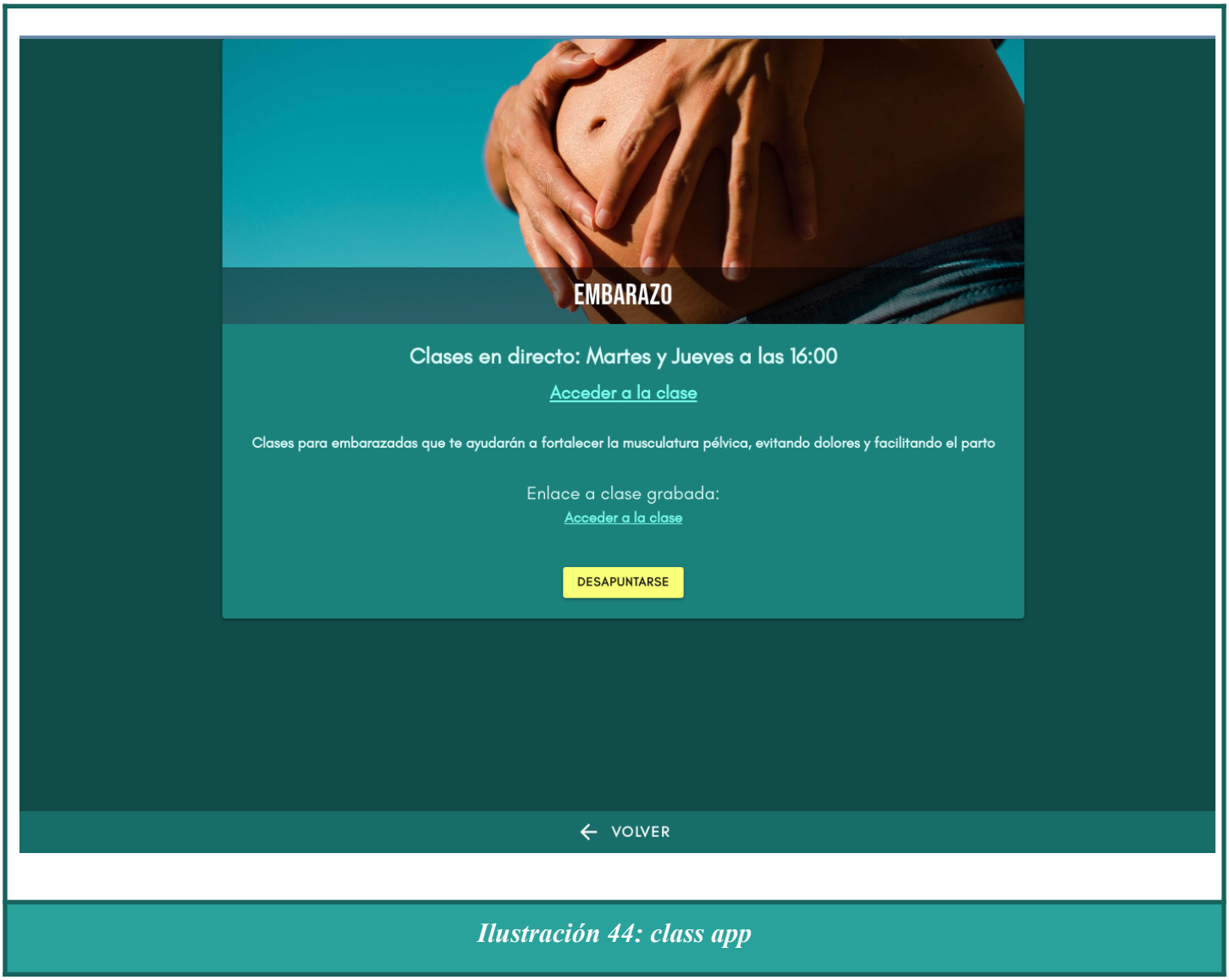



***Ilustración 44: class app***

Esta página tiene la peculiaridad de que aunque comparte elementos comunes, otros elementos cambian dependiendo de si el que accede es una alumna o la administradora. Como por ejemplo los

botones de editar y borrar la clase no se renderizan para las alumnas. Esto se hace con un pequeño condicional, el cuál es una directiva proporcionada por Vue:

```
<div class="row" v-if="getRoute.path == '/admin/class'">
```

Por último, para la gestión de toda la información de estas clases se ha creado un nuevo módulo en el *store*. En este módulo se establecen ciertas acciones para leer o borrar la información de la base de datos y añadirla o eliminarla del state[37].

La primera función lee los datos de la base de datos de Firebase. Primero se obtiene el ID del usuario actual, y luego las referencias a la base de datos. Posteriormente, con la función *onValue()* se detecta si hay algún cambio en la base de datos y con *commit* se llama a la mutación *addClassesID* para guardar en el *state* los IDs de las clases que ese usuario tiene. Luego la última función *updateClassesUser()* añade toda la información de las clases que ese usuario tiene al *state*.

```
export function fbReadData({ commit, state }) {

 const userID = firebaseAuth.currentUser.uid;

 const userClass = ref(firebaseDB, "classData/" + userID);

 const allUserClass = ref(firebaseDB, "classData/");



onValue(userClass, (snapshot) => {

   const classes = snapshot.val();

   commit("addClassesID", classes);

   updateClassesUser(state, commit)

 })
```

[37] Enlace: https://vuex.vuejs.org/guide/state.html

Hay dos funciones más que controlan cuando un usuario se apunta a una clase o se borra de una. Acciones que cambian la base de datos. Por ejemplo, el código de abajo corresponde con la función para desapuntar a un usuario de una clase:

```
export function removeNewClassToUser({ commit, state }, id) {

 const userID = firebaseAuth.currentUser.uid;



 remove(ref(firebaseDB, 'classData/' + userID + '/' + id), null);



 commit("removeNewClassesID", id)

 commit("removeNewUserClasses", id)

}
```

- **Administración de alumnas**

La administradora debe de poder ver a todas sus alumnas, buscar con facilidad a algunas de ellas, poder ver su información y desactivar o activar su cuenta. Esto se realiza en la página *Admin-studens.vue.* Como se aprecia en la **Ilustración 45** y en la **Ilustración 46**, se trata de una tabla que puede cambiar su modo de visualización para ver la información más comprimida o más visual. Además permite buscar y ordenar a las alumnas por su nombre, edad, correo o estado.

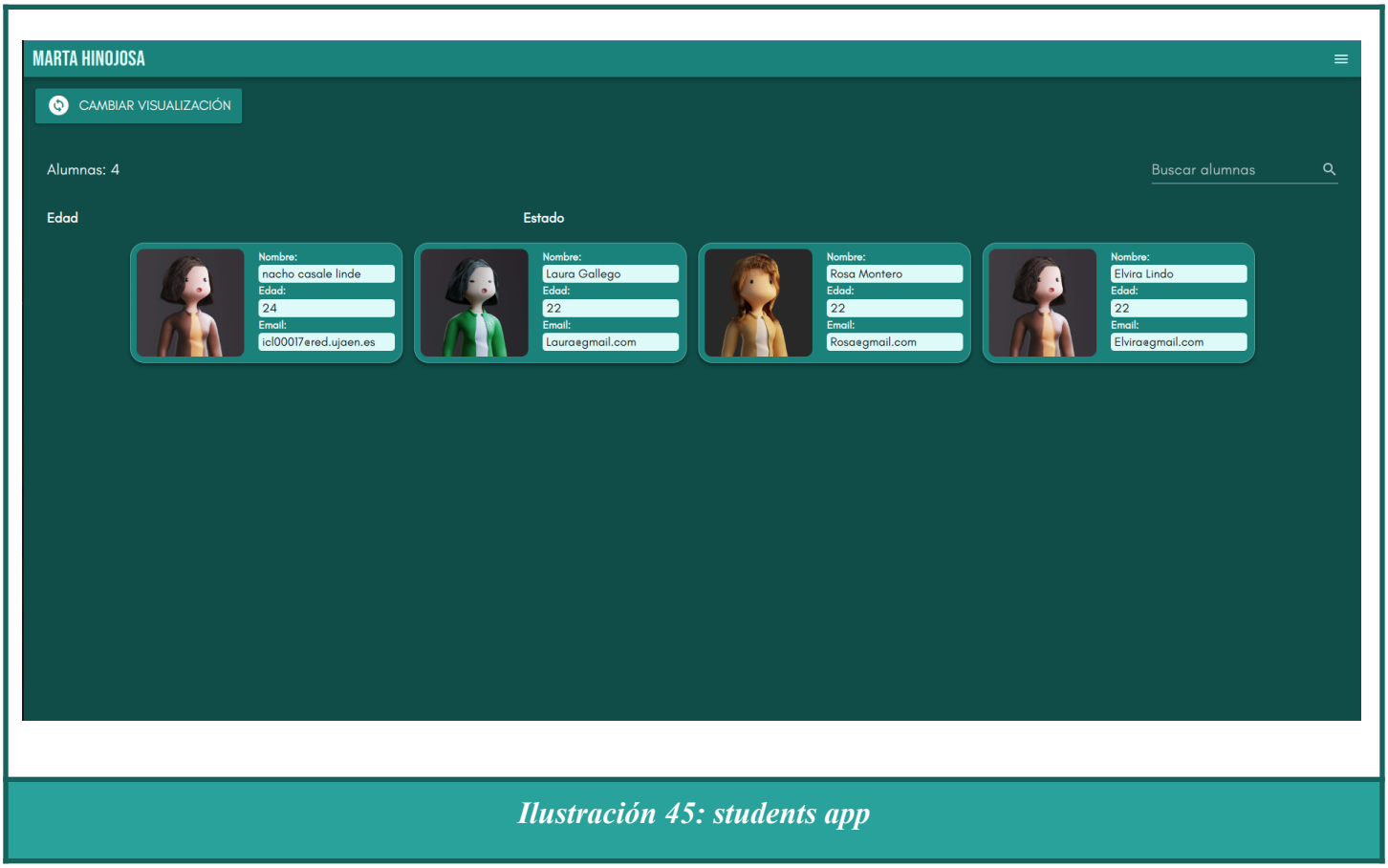



***Ilustración 45: students app***


***Ilustración 46: students 2 app***

Estas tablas están creadas gracias a la personalización de una tabla[38] de Quasar, la cuál ya viene con la funcionalidad para la búsqueda y el filtrado. Sin embargo, esta personalización no se limita únicamente a la modificación de los estilos, sino que también al uso del elemento *slot* de Vue. El cual es usado internamente por las tablas que Quasar nos proporciona, pero conociendo su funcionamiento podemos personalizar estas tablas tanto como se quiera. Su funcionamiento es sencillo **[32]**, usando la etiqueta *slot* se puede definir un valor por defecto para algún componente, pero luego este valor puede ser sustituido por otro valor cuando se usa el componente. También es posible añadir nombres a los *slots* para diferenciarlos de otros.

```
        <template v-slot:item="props">
```

[38] Enlace: https://quasar.dev/vue-components/table

```
        <q-card
          bordered
          class="q-ma-sm cardUser"
          v-bind:class="{
            cardUser: props.row.state,
            cardUserDelete: !props.row.state,
          }"
          @click="$router.push({ path: '/admin/student/' + props.row.id })"
        >
          <div class="q-ma-sm row">
           // Contenido
          </div>
        </q-card>
      </template>
```

La sección de código de arriba corresponde con las tarjetas visuales de la tabla. Esta etiqueta *template* se inserta dentro de la tabla de Quasar y con *v-slot:item* se hace referencia a que se va a cambiar el *slot* con nombre *item*. Luego también se ha establecido el nombre *props* donde se van a ligar las propiedades del slot item. Gracias a esto, se puede acceder al objeto que corresponde con los datos de nuestras filas. Por ejemplo, con *props.row.name*.

- **Subir imagenes**

Para subir imágenes hay que hacer uso del *storage*[39] de Firebase. Se trata de un almacenamiento en la nube para guardar archivos, como por ejemplo, las imágenes de perfil de las alumnas o las imágenes de las clases. Para el control de las imágenes se ha creado un nuevo módulo del *store*. En las acciones de este módulo se usan 2 funciones cuando el usuario quiere actualizar su imagen de perfil.

[39] Enlace: https://firebase.google.com/docs/storage

La primera función es para subir y actualizar la imagen antigua de perfil por la nueva imagen elegida por el usuario:

```
export function uploadProfileImg({ }, payload) {

 const userID = firebaseAuth.currentUser.uid;

 const storageRef = sRef(storage, 'profileImages/' + userID);



 uploadBytes(storageRef, payload).then((snapshot) => {

   getDownloadURL(sRef(storage, 'profileImages/' + userID))

     .then((url) => {

       changeUrlProfileImg(userID, url);

     })

     .catch((error) => {

       console.log(error)

     });

 });

}
```

Su funcionamiento es sencillo, obtiene la referencia del storage de Firebase y del usuario actual. A continuación llama a la función *uploadBytes()* que es proporcionada por Firebase y es la encargada de subir el archivo a la nube. Después se obtiene la url que representa esa imagen en Firebase, para así posteriormente llamar a la segunda función, *changeUrlProfileImg()*.

```
function changeUrlProfileImg(userID, payload) {

 const updates = {}

 updates['usersData/' + userID + '/url'] = payload;

 update(ref(firebaseDB), updates);

}
```

Esta función actualiza en la BBDD la url de la imagen de perfil del usuario, para que así la aplicación cargue correctamente la imagen.

- **Sistema de chat**

El sistema de chat utiliza la base de datos en tiempo real de Firebase, **Ilustración 47**, donde almacena los mensajes en orden. Para hacer esto, se ordenan según su identificador. Firebase ordena automáticamente por orden alfabético y numérico. La clave de los mensajes se crea en el cliente cogiendo el tiempo UTC (tiempo universal), para evitar que se desordenen los mensajes si hay diferentes horas locales, y formateando adecuadamente para que se añadan los 0 necesarios en los datos necesarios. Por ejemplo, la hora 9 no se admite, en su lugar se añade un 0 delante, quedando 09. De esta forma se tiene siempre el mismo número de cifras y no se desordenan los mensajes. Como en el tiempo que se usa en la clave se añaden también los milisegundos sería muy extraño tener dos mensajes con el mismo ID en una aplicación tan pequeña como esta. Sin embargo, por si acaso, y ya que no se pueden tener dos ID idénticos en Firebase, se ha añadido un código aleatorio como parte del ID de los mensajes. Así se asegura que están ordenados y con ID único. Una vez realizado esto, en el lado del cliente se obtienen todos los mensajes y se muestran correctamente gracias a que ya están ordenados.

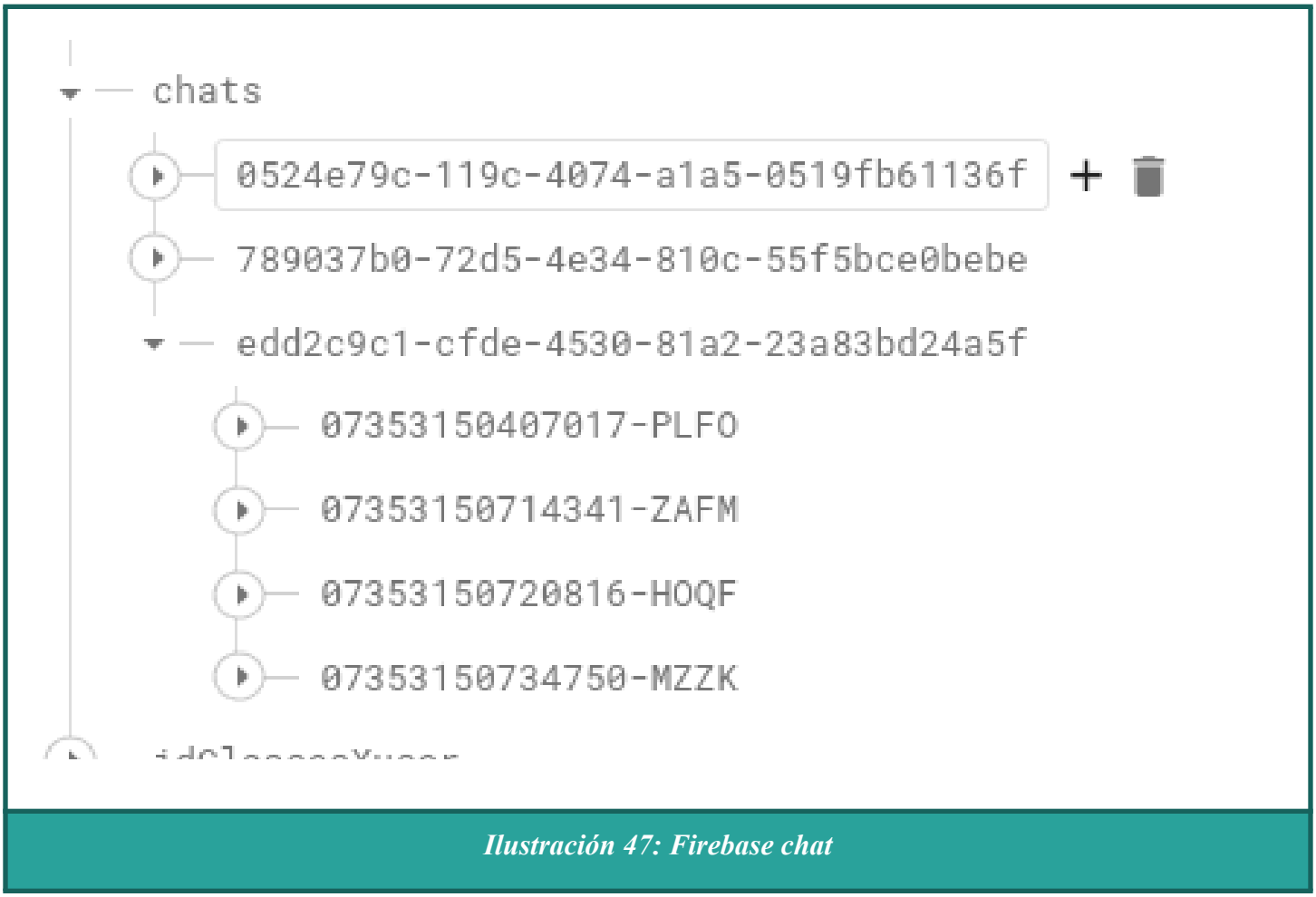



*Ilustración 47: Firebase chat*

Si analizamos la clave de cerca, por ejemplo: “07353150734750-MZZK” , vemos que:

- Los dos primeros números: “07” hacen referencia al mes (enero es el 0). Como 7 es sólo un dígito se le añade un “0” delante para que no se desordene cuando llegue el mes “10”, ya que “1” < “7” pero “0” < “1”. Así “07” < “10”.
- Los siguientes dos números: “35” hacen referencia a la semana.
- El siguiente: “3” hace referencia al día, también empezando desde el 0.
- Los siguientes cuatro números: “1507” hacen referencia a la hora y minutos. En este caso serían las 15:07. Aquí también se añaden ceros si es necesario.
- Los siguientes dos números: “34” son los segundos. También se añade un 0 si es necesario.

- Los últimos 3 números son los milisegundos: “750”, también se añaden ceros si es necesario.
- Por último, las letras del final son aleatorias para garantizar que no se repiten los IDs.

Con esto se garantiza que los mensajes no se desordenen en un año, pero como se puede ver, no se ha incluido el año en la fecha, por lo que los mensajes podrían desordenarse entre diferentes años. Aunque sería rápido de añadir, es algo que no va a ocurrir en esta aplicación, ya que se ha creado un sistema para ir borrando los mensajes más antiguos. Esto se ha pensado para que la base de datos no se llene de excesiva información, ya que almacenarla también tiene un coste. El límite de los mensajes se puede establecer al que la cliente desee. Pero por ejemplo, con un límite de 1.000-5.000 mensajes te aseguras que no se repitan en un año y que se mantenga baja la ocupación de la base de datos. Pero, ¿cómo se controla los mensajes que se borran? En el código que viene a continuación se puede ver.

*Esta función, createMessageToSend(),* se ejecuta cada vez que se envía un mensaje. Como se ve en el código de abajo, se obtiene la fecha actual en UTC y se genera un código aleatorio. Todo esto servirá para crear la clave del mensaje que se va a mandar.

```
createMessageToSend() {
    var date = new Date();
    var code = this.generateRandomCode();
    var msUTC = date.getUTCMilliseconds();
    var secUTC = date.getUTCSeconds();
    var minUTC = date.getUTCMinutes();
    var hourUTC = date.getUTCHours();
    var dayUTC = date.getUTCDay();
    var monthUTC = date.getUTCMonth();
```

En esta segunda parte simplemente se formatea las fechas para añadir los ceros cuando sean necesarios:

```
    if (secUTC.toString().length < 2) {
```

```
    secUTC = "0" + secUTC.toString();
  }

  if (msUTC.toString().length < 3) {
    if (msUTC.toString().length == 2) {
      msUTC = "0" + msUTC.toString();
    } else if (msUTC.toString().length == 1) {
      msUTC = "00" + msUTC.toString();
    }
  }

  if (minUTC.toString().length < 2) {
    minUTC = "0" + minUTC.toString();
  }

  if (hourUTC.toString().length < 2) {
    hourUTC = "0" + hourUTC.toString();
  }

  if (monthUTC.toString().length < 2) {
    monthUTC = "0" + monthUTC.toString();
  }
```

Aquí se crea la variable que guardará el ID del mensaje. Como se puede ver se llama a una función para obtener el también el número de la semana actual:

```
  var messageID =
    monthUTC.toString() +
    this.weekNumber() +
    dayUTC.toString() +
    hourUTC.toString() +
    minUTC.toString() +
    secUTC.toString() +
```

```
      msUTC.toString() +
      "-" +
      code;
```

En esta parte se formatea la hora local, para mostrarla correctamente en la interfaz. También se crea un objeto auxiliar, el cuál se enviará más tarde como el objeto que se añadirá a la base de datos. Este objeto es la información del mensaje. La variable *text* obtiene el valor de *this.text*, la cuál contiene el valor del input donde el usuario ha escrito el mensaje.

```
    var localHours = date.getHours();
    if (localHours.toString().length < 2) {
      localHours = "0" + localHours.toString();
    }

    var localmin = date.getHours();
    if (localmin.toString().length < 2) {
      localmin = "0" + localmin.toString();
    }

    var objectAux = {
      id: this.getUsersData[0].id,
      name: this.getUsersData[0].name,
      urlProfileImg: this.getUsersData[0].url,
      text: this.text,
      hour: localHours,
      min: localmin,
      first: true,
      admin: false,
      classID: this.id,
      msgID: messageID,
    };
```

En esta última parte, se comprueba si quien ha escrito el mensaje es el administrador o no. Si lo es, se cambian algunos atributos, por ejemplo, se establece que el mensaje proviene del administrador con *object.admin = true.* Después se llama a otra función que comprueba cuál ha sido el último mensaje. Si el último mensaje pertenece al mismo usuario que va a enviar el mensaje no se muestra su foto de perfil, **Ilustración 47** (primera y segunda imagen), de lo contrario, si se muestra. A continuación se envía el mensaje por una función que lo añadirá a la base de datos. Se establece el input a vacío y se comprueba la longitud del chat. Si la longitud ha superado la máxima permitida, se coge el primer mensaje guardado (el más alto en la base de datos) y se borra, ya que este será el más antiguo. También para finalizar se hace un scroll hacia abajo, para ver mejor el mensaje que se acaba de escribir quitando la carga de trabajo al usuario de hacer scroll.

```
    if (this.getCurrentUser == this.getAdminID) {
      if (this.prioritaryMsg) objectAux.admin = true;
      objectAux.id = this.getAdminID;
      objectAux.name = "Marta Hinojosa";
      objectAux.urlProfileImg = this.imgAdminUrl;
    }

    objectAux = this.checkLastMessage(objectAux);

    this.addMessage(objectAux);

    this.text = "";

    this.checkLengthChat();

    this.scrollToBottom();
  },
```

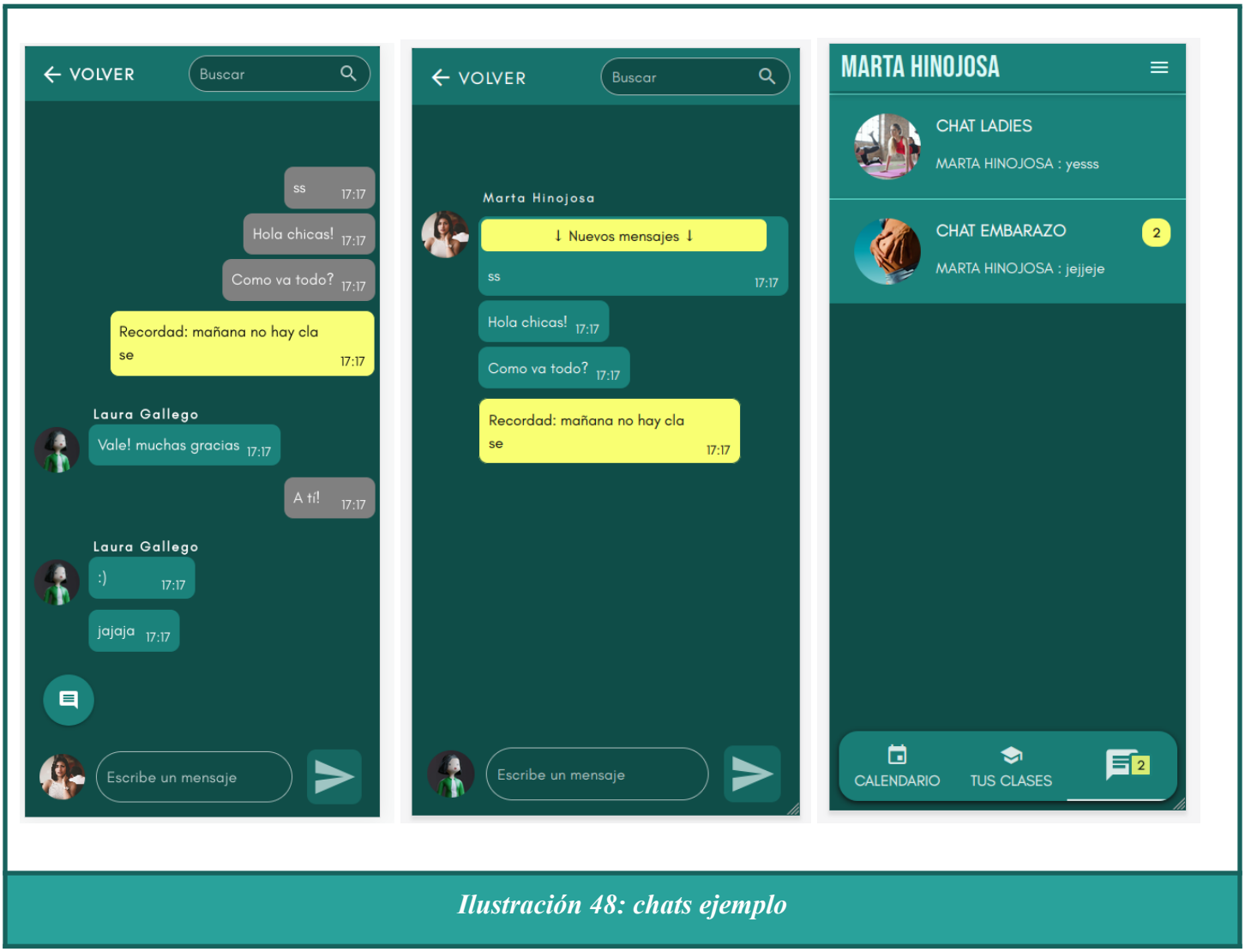



***Ilustración 48: chats ejemplo***

Hasta ahora hemos visto un sistema de chat funcional que auto-elimina los mensajes más antiguos. También se ha visto como se muestra la imagen de perfil según sea el primer mensaje o no, lo cuál ayuda a diferenciar a los diferentes usuarios. Pero hay más funcionalidades para ayudar a leer mejor el chat, diferenciarte entre tú y los demás usuarios o ver mensajes con más importancia. Por ejemplo, los mensajes propios aparecen en gris y a la derecha, y los mensajes de los demás aparecen en el color primario y a la izquierda. El nombre también aparece solo cuando es el primer mensaje y los márgenes entre estos mensajes son menores. Cuando se entra en la aplicación también se pueden ver

los mensajes recibidos hasta el momento, **Ilustración 48** (imagen 3). Se ven de forma global en la pestaña inferior de chat o en cada chat de forma individual. Cuando se entra un chat, se hace scroll instantáneo hasta el primer mensaje nuevo recibido, **Ilustración 48** (imagen 2).

Por último, hay 3 funcionalidades más que comentar: la barra de búsqueda, los mensajes prioritarios y los mensajes anclados.

- La barra de búsqueda, **Ilustración 49** (Imagen 1) permite buscar un mensaje por su texto o por el nombre de su usuario.
- Los mensajes prioritarios, **Ilustración 49** (Imagen 1 y 2), se muestran en amarillo y solo los puede mandar la administradora. Su función es destacar más en el chat para que a las alumnas no se les pase leerlos. La administradora puede cambiar rápidamente entre mensajes normales o prioritarios pulsando un botón flotante que le aparece encima de su foto de perfil.
- Los mensajes anclados, **Ilustración 50** (Imagen 2 y 3), son mensajes más importantes que los prioritarios o que necesitan ser recordados durante un tiempo. La administradora puede anclar cualquier mensaje y las alumnas les aparecerá en amarillo (hasta que sea leído), **Ilustración 49**, cuando un nuevo mensaje se haya fijado. Después de leer el mensaje pueden volver a leerlo hasta que sea sustituido por otro.

Como nota final, para anclar los mensajes y mostrar el número de mensajes pendientes, también ha sido necesario guardar esa información en la base de datos. También a cada alumna le aparecerá un chat u otro en función de a qué clase esté apuntada. Cada vez que la administradora cree una nueva clase, se creará un nuevo grupo.

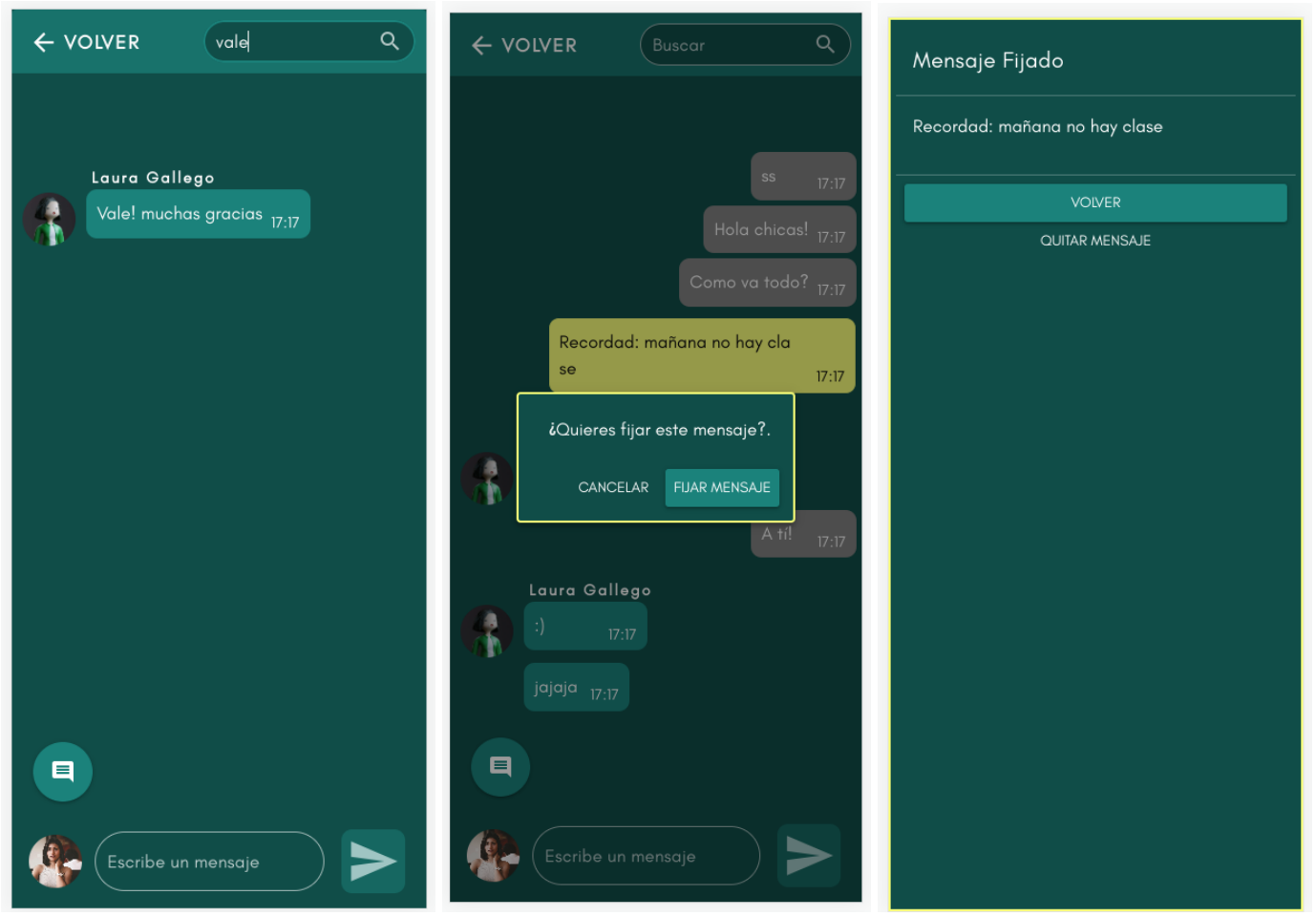



*Ilustración 49: chats ejemplo 2*

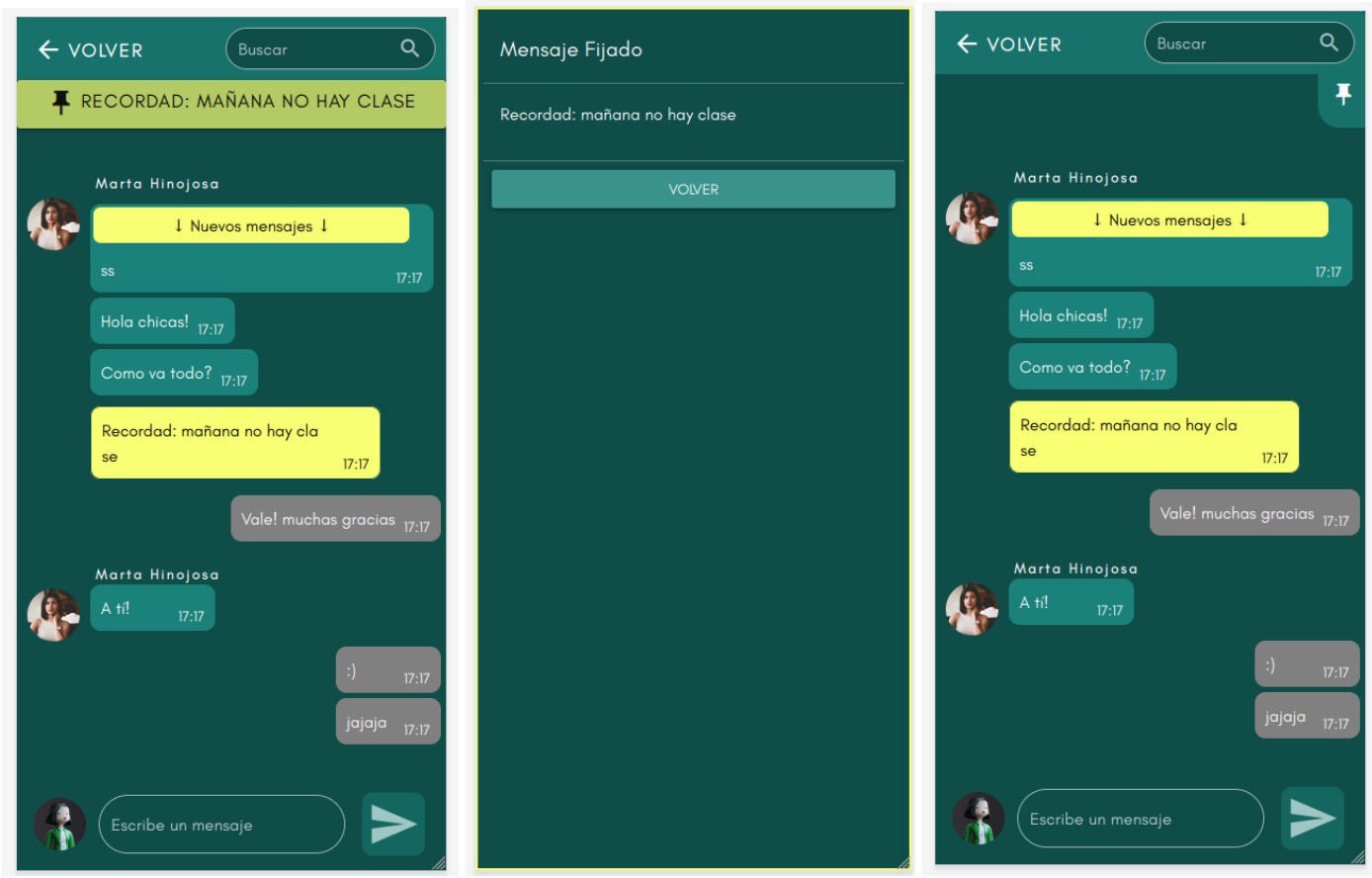



***Ilustración 50: chats mensaje anclado***

- **Aplicación móvil.**

Para poder exportar a una aplicación móvil hace falta tener Android Studio[40] (para android) y Xcode[41] (para Ios) instalado. En este apartado se va a hablar de la instalación y los pasos a seguir para exportar la aplicación a Android.

Generalmente, cuando se piensa en desarrollar una aplicación híbrida y tener su versión para móvil, se suele utilizar Cordova[42]. Sin embargo, es un framework que contiene varios problemas **[33]** y ralentiza el desarrollo. Actualmente existe un framework más reciente **[34]**, desarrollado por Ionic, que viene a sustituir a cordova. Se trata de Capacitor.

Este framework soluciona los problemas de cordova y tiene ciertas mejoras:

- Tiene una excelente integración con frameworks Javascript, como Vue.js.
- Crea fácilmente APPs, PWA o de escritorio (con Electron).
- Se puede acceder a características nativas sin plugins.
- Extensiones propias y compatibles con las de cordova.
- Rápido de instalar y ejecutar sin incompatibilidades.

Quasar tiene soporte para estos dos frameworks, pero por sus ventajas, se decidió usar Capacitor. Después de instalar Android Studio, hay que configurar el sistema para usarlo:

```
export ANDROID_HOME="$HOME/Android/Sdk"

export ANDROID_SDK_ROOT="$HOME/Android/Sdk"

PATH=$PATH:$ANDROID_SDK_ROOT/tools;

PATH=$PATH:$ANDROID_SDK_ROOT/platform-tools
```

Con estos comandos ya estaría. Sin embargo, actualmente puede haber problemas al correr el emulador, por lo que en este caso ha sido necesario cambiar el tercer comando por:

[40] Enlace: https://developer.android.com
[41] Enlace: https://apps.apple.com/es/app/xcode/id497799835?mt=12
[42] Enlace: https://cordova.apache.org/

```
PATH=$PATH:$ANDROID_SDK_ROOT/emulator;
```

Después de esto, hay que instalar Capacitor:

```
$ quasar mode add capacitor
```

Y después simplemente hay que correr el siguiente comando:

$ quasar [dev|build] -m capacitor -T [android|ios]

Con esto, se creará una apk para la plataforma elegida. Si se cambia la palabra “dev” por “build” se creará la exportación para producción. Para ello es necesario firmar la aplicación,y finalmente se obtendrá un archivo .abb en vez de .apk. Actualmente las tiendas como Play store utilizan esta extensión por diversas razones. Tras esto, ya se tendrá la aplicación funcionando, **Ilustración 51**, pero con el logo y la *splash screen* de Capacitor. Para cambiar esto hay que ejecutar el siguiente comando:

```
$ npm install -g capacitor-resources
```

Después hay que crear una carpeta llamada resources en la raíz del proyecto Capacitor (src-capacitor/) y añadir ahí el icono (1024x1024) y la *splash screen* (2732x2732). Por último hay que ejecutar este comando:

```
$ capacitor-resources
```

Tras esto se generan automáticamente todos los tamaños necesarios para todas las plataformas y se copiaran en Android Studio. Así se podrá tener un logo y una *splash screen*, **Ilustración 52**, personalizados.

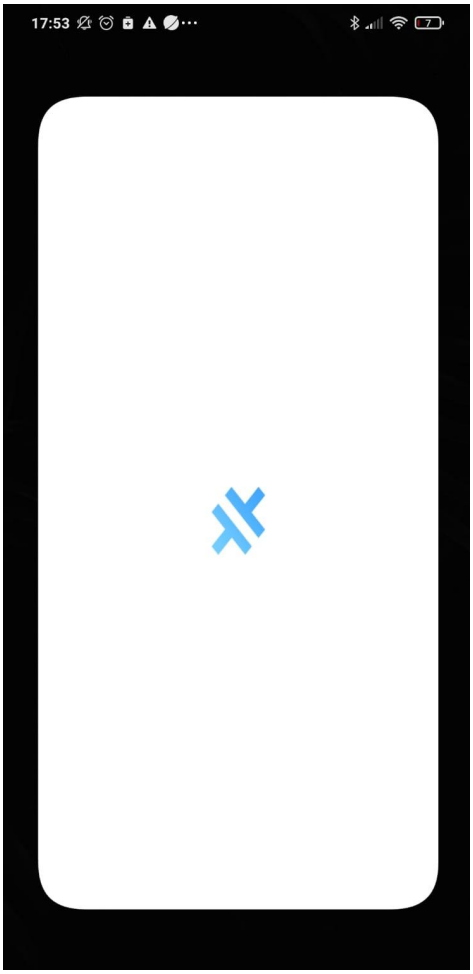



*Ilustración 51: app Android Capacitor logo*

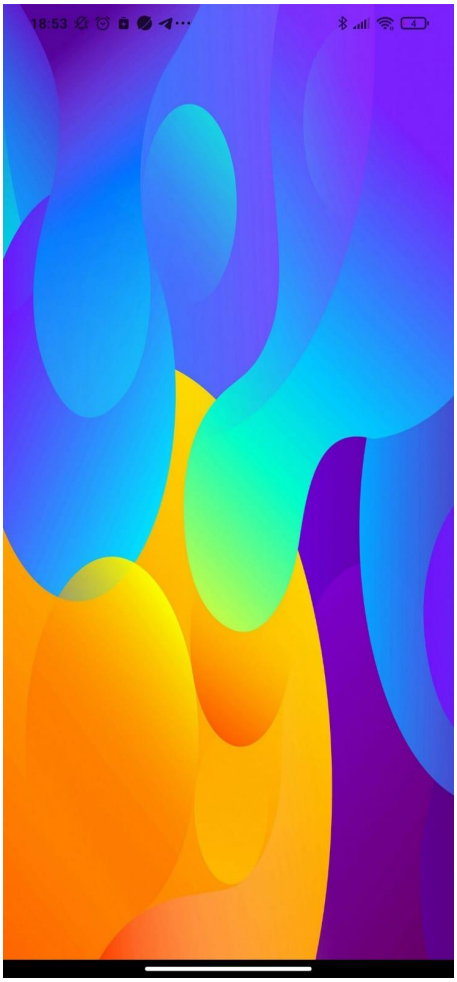



*Ilustración 52: app Android custom splash screen*

Sin embargo, esto último no funcionó (debido seguramente a que no encontró correctamente la ubicación de Android Studio), así que tuve que hacerlo manualmente en Android Studio. Para añadir los iconos, la forma más rápida, es crear un nuevo asset de imágenes, **Ilustración 53**, y sobreescribir el actual. Esto te permite visualizar y ajustar cómo se verán las imágenes.

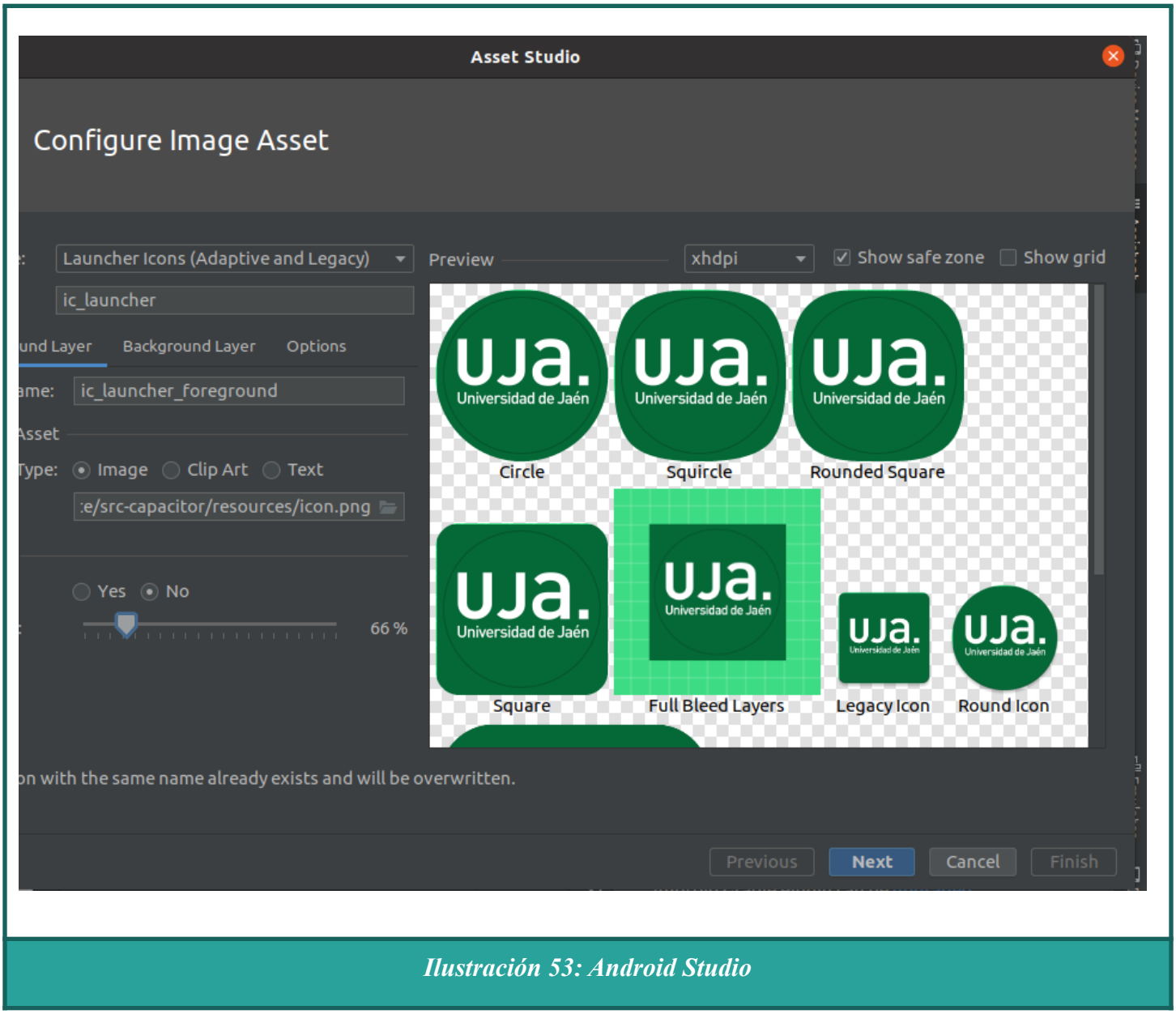



*Ilustración 53: Android Studio*

Para cambiar la *splash screen* hay que sustituir manualmente los archivos. Así finalmente se obtiene la aplicación con un logo, **Ilustración 54**, y *splash screen* personalizados.


***Ilustración 54: logo android app***

- **Desplegar la aplicación en AWS.**

En este apartado se va a explicar como se ha desplegado la aplicación en Amazon Web Services. Al final se decidió utilizar AWS Amplify, **Ilustración 55**. Este es un conjunto de herramientas **[35]** y servicios que permite acelerar el desarrollo de aplicaciones web y móviles.


***Ilustración 55: logo amplify***

Para empezar a usarlo hay que instalarlo en el proyecto actual con el siguiente comando:

```
$ npm install -g @aws-amplify/cli
```

Y con *$ amplify configure* se enlaza con la cuenta de AWS. Después hay que iniciarlo con:

```
$ amplify init
```

Esto hará que tengas que responder varias preguntas sobre el proyecto, como el nombre, tipo de proyecto, el comando para hacer una build y para iniciar tu framework actual. En el caso de Quasar son:

```
Build Command: quasar build
```

```
Start Command: quasar serve
```

Por último, para configurar el ambiente donde correrá la aplicación y publicarla hay que correr los siguientes comandos:

```
$ amplify add hosting
```

```
$ amplify publish
```

Esto construirá la aplicación subirá la aplicación, **Ilustración 56**, y la subirá a AWS, **Ilustración 57**. De esta forma, cada vez que se quiera actualizar la aplicación, bastará con escribir el último comando. Al hacerlo, incluso modificando o añadiendo páginas, mostrará que no hay cambios, y si quieres subir la aplicación de todas formas. Esto se debe a que no se ha modificado ninguna opción del ambiente donde se hostea y tampoco hay cambios en el back-end (por la naturaleza de esta aplicación). Aún así, al marcar que sí, se actualizará el front-end y todos los archivos Javascript.

También mencionar, que como esta aplicación es estática y "no tiene" backend es posible desplegarla en un bucket S3 (que es un contenedor de archivos) sin tener que configurar un servidor web. En este caso se usa *CloudFront*[43] para entregar el contenido mediante https y con baja latencia.

[43] Enlace: https://aws.amazon.com/es/cloudfront/

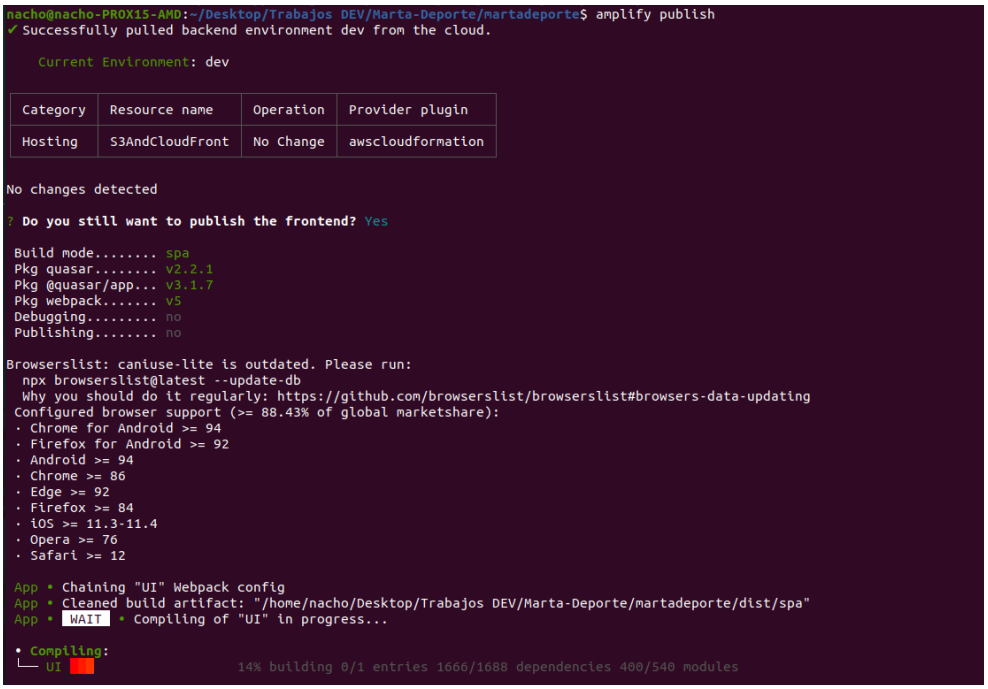



*Ilustración 56: amplify publish*

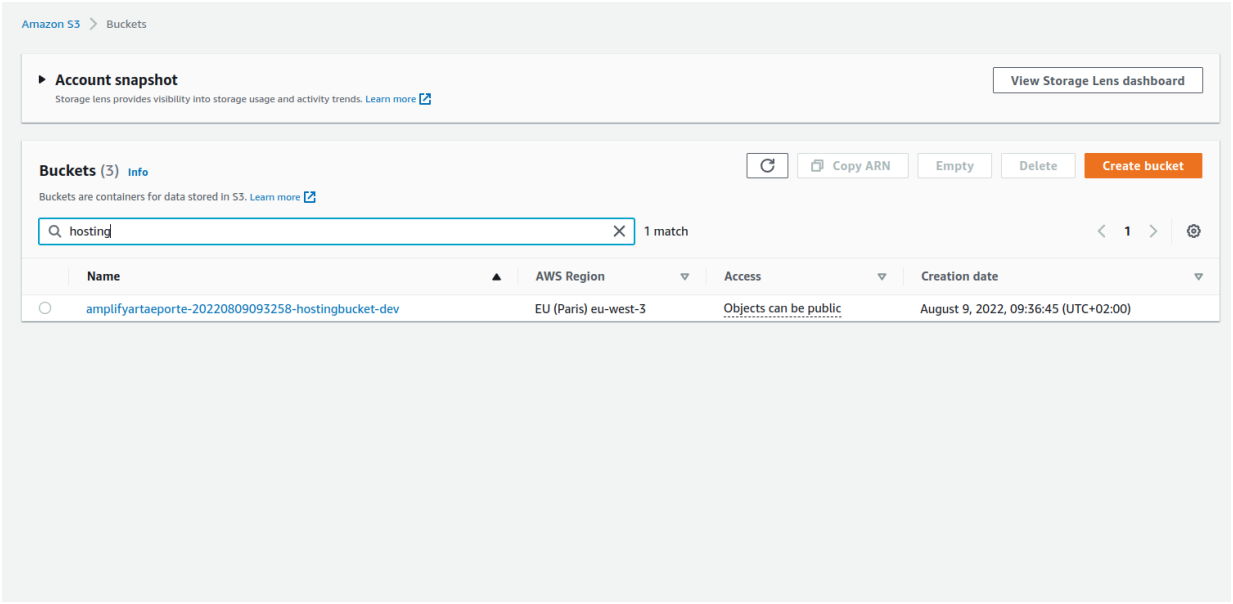



*Ilustración 57: AWS S3 bucket*

# 5. RESULTADOS DE LAS PRUEBAS

En este apartado se van a ver los resultados obtenidos de las pruebas realizadas.

## 5.1 Control de acceso

Inicialmente, cuando queremos registrarnos, **Ilustración 58**, debemos introducir un email obligatoriamente, y no se podrá continuar al siguiente paso hasta que el email tenga una forma válida.


***Ilustración 58: email error***

Continuando con el registro, **Ilustración 59**, encontramos la fecha de cumpleaños. Aquí podemos introducir el día, el mes y el año. Todos los campos son obligatorios y se comprueba si son válidos:

- El día debe estar comprendido entre 1 y 31. Solo se admite una longitud máxima de dos caracteres.
- El mes debe de estar comprendido entre 1 y 12. Solo se admite una longitud máxima de dos caracteres.
- El año debe de estar comprendido entre 1930 y 2006. Solo se admite una longitud de cuatro caracteres.

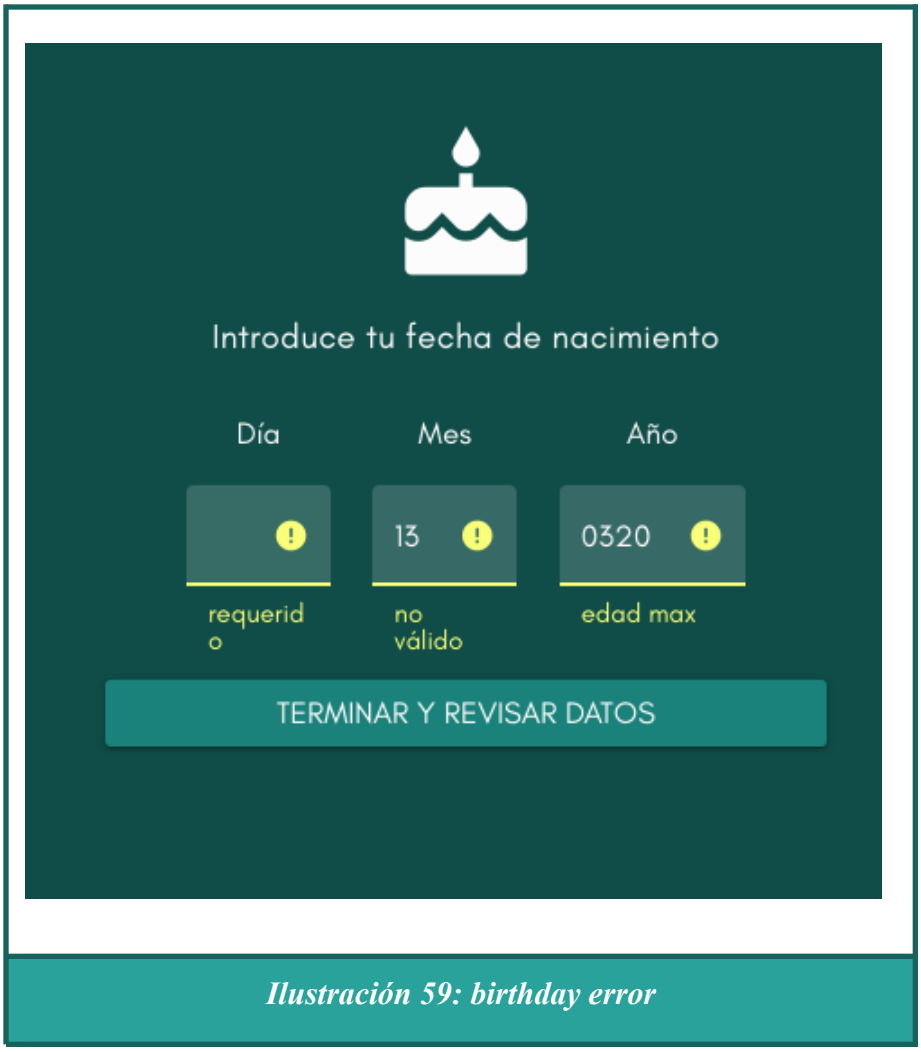



***Ilustración 59: birthday error***

En el formulario del final, **Ilustración 60**, donde se pueden revisar los datos y añadir una contraseña se vuelve a comprobar todos datos. Si hay algo incorrecto se marcará con un error al intentar registrarse. Además si el correo registrado ya existe, se muestra un error comunicándolo.

- Se debe de introducir un nombre
- Una fecha de cumpleaños válida
- Un email válido
- Una contraseña con un mínimo de 6 caracteres. Se debe repetir la contraseña y que coincidan

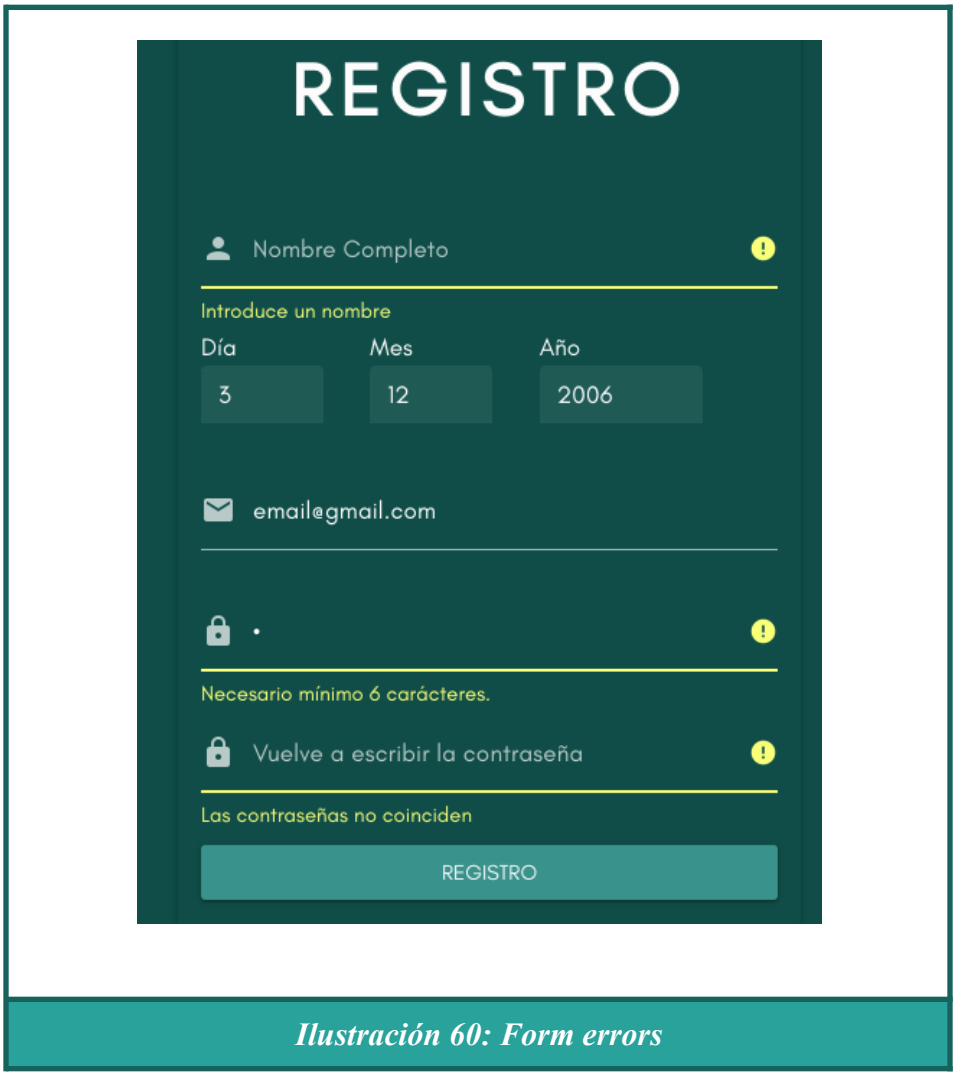



***Ilustración 60: Form errors***

En cuanto al formulario de inicio de sesión, se comprueba también que el email sea válido y se haya introducido una contraseña. Al intentar iniciar sesión, se comprueban las credenciales en Firebase, si son incorrectas, **Ilustración 61**, se muestra un error.

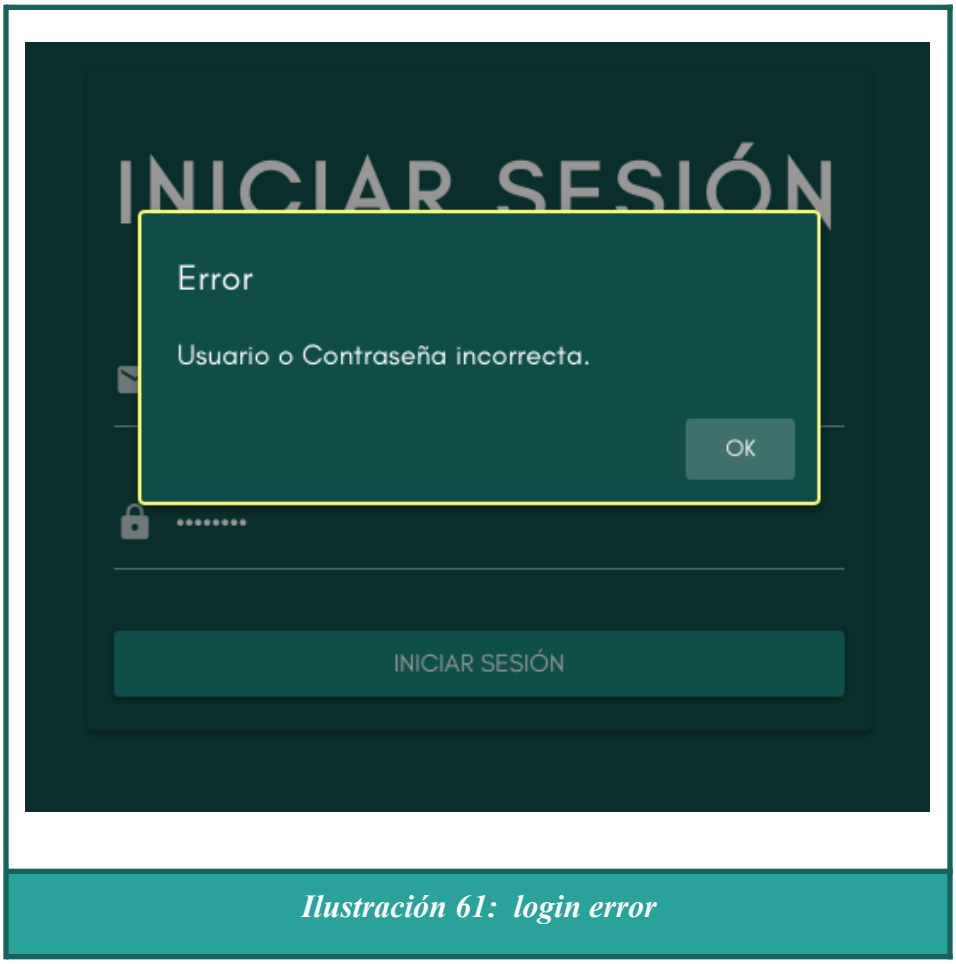



***Ilustración 61: login error***

Para el registro hay dos formularios (uno paso a paso y otro para revisar). Cada uno tiene comprobaciones similares pero también algunas diferentes, en total unas 25 comprobaciones para evitar errores y que muestran una alerta. Hay varios formularios más en la aplicación, los cuales varios de ellos comparten código y comprobaciones. Tomando en cuenta esto, hay 12 comprobaciones más que realizar.En total se realizaron alrededor de 37 pruebas únicas para probar las comprobaciones de formularios que podrían provocar error o un comportamiento no deseado. Sin embargo, varias de estas

pruebas se repitieron sobre los distintos formularios que compartían código, para asegurarse que también funcionan correctamente. También se realizaron más pruebas para asegurarse de que datos que deben ser aceptados, no son detectados como error, y efectivamente, pueden pasar el formulario. Además también hay otro tipo de pruebas que se realizaron que no involucran formularios, como la protección de las rutas, aquí se realizaron alrededor de 12 pruebas. En el sistema de chat se fueron realizando diferentes pruebas para ver que los mensajes, y sus notificaciones se entregan correctamente a todos los usuarios que se deben de entregar. Para esto, se fue comprobando la base de datos y varias instancias de los clientes. Se realizaron alrededor de 8 pruebas (comprobar ID del mensaje, ordenamiento de los mensajes, mensajes entregados, notificaciones, etc) que se repitieron en diferentes etapas del desarrollo.

Gracias a estas pruebas pude detectar algunos errores en la protección de rutas. También fueron esenciales para el sistema de chat, ya que detecté varios errores con la generación del ID único que provocaba que se ordenarán mal los mensajes. Por último, puede detectar y solucionar varios problemas con las notificaciones de los mensajes.

## 5.2 Reglas de seguridad

Para comprobar que solo se accede a los recursos que un usuario tiene acceso, Firebase cuenta con una zona de pruebas. En las reglas de seguridad de Firebase **[36]**, **Ilustración 62-63**, se ha establecido de forma general que las escrituras y lecturas sean rechazadas. De esta forma, si se olvida de escribir una regla, la petición será rechazada. Después se han escrito diferentes reglas según para qué documento. Ya sea que solo tenga acceso el administrador, o un usuario autenticado o solo el usuario autentificado con el mismo identificador que el documento al que se accede.

```
{
  "rules": {
    ".read": "false",
    ".write": "false",

    "chats": {
      ".read": "auth.uid != null",
      ".write": "auth.uid != null",
    },

    "readedMsg":{
      ".read": "auth.uid != null",
      ".write": "auth.uid != null",
    },

    "pinnedMsg":{
      ".read": "auth.uid != null",
      ".write": "root.child('adminData').child(auth.uid).child('role').val() == 'admin'",
    },

    "usersData": {
      ".read": "auth.uid != null",
      ".write": "true",
      "$uid": {
        ".write": "auth.uid == $uid"
       }
     },

```

***Ilustración 62: bbdd realtime 1***

```
28
29      "idClassesXuser": {
30        ".read": "root.child('adminData').child(auth.uid).child('role').val() == 'admin'",
31        ".write": "root.child('adminData').child(auth.uid).child('role').val() == 'admin'",
32        "$uid": {
33          ".read": "auth.uid == $uid",
34          ".write": "auth.uid == $uid"
35        }
36      },
37
38      "allClasses": {
39          ".read": "auth.uid != null",
40          ".write": "root.child('adminData').child(auth.uid).child('role').val() == 'admin'"
41       },
42
43      "adminData": {
44          "code": {
45             ".read": "true",
46          },
47          "$uid": {
48             ".read": "auth.uid == $uid",
49             ".write": "auth.uid == $uid"
50          },
51          ".read": "auth.uid != null",
52          ".write": "root.child('adminData').child(auth.uid).child('role').val() == 'admin'"
53       },
54
55
56    }
57  }
```

***Ilustración 63: bbdd realtime 2***

Si hacemos uso de la zona de prueba de las reglas, podemos por ejemplo hacer una petición de tipo get, sin autentificar, a */usersData.* Como se puede ver en la **Ilustración 64**, la petición es rechazada. Primero en la línea 3 (ya que todas las peticiones son rechazadas inicialmente) y luego en la línea 22, ya que el usuario necesita estar autentificado.

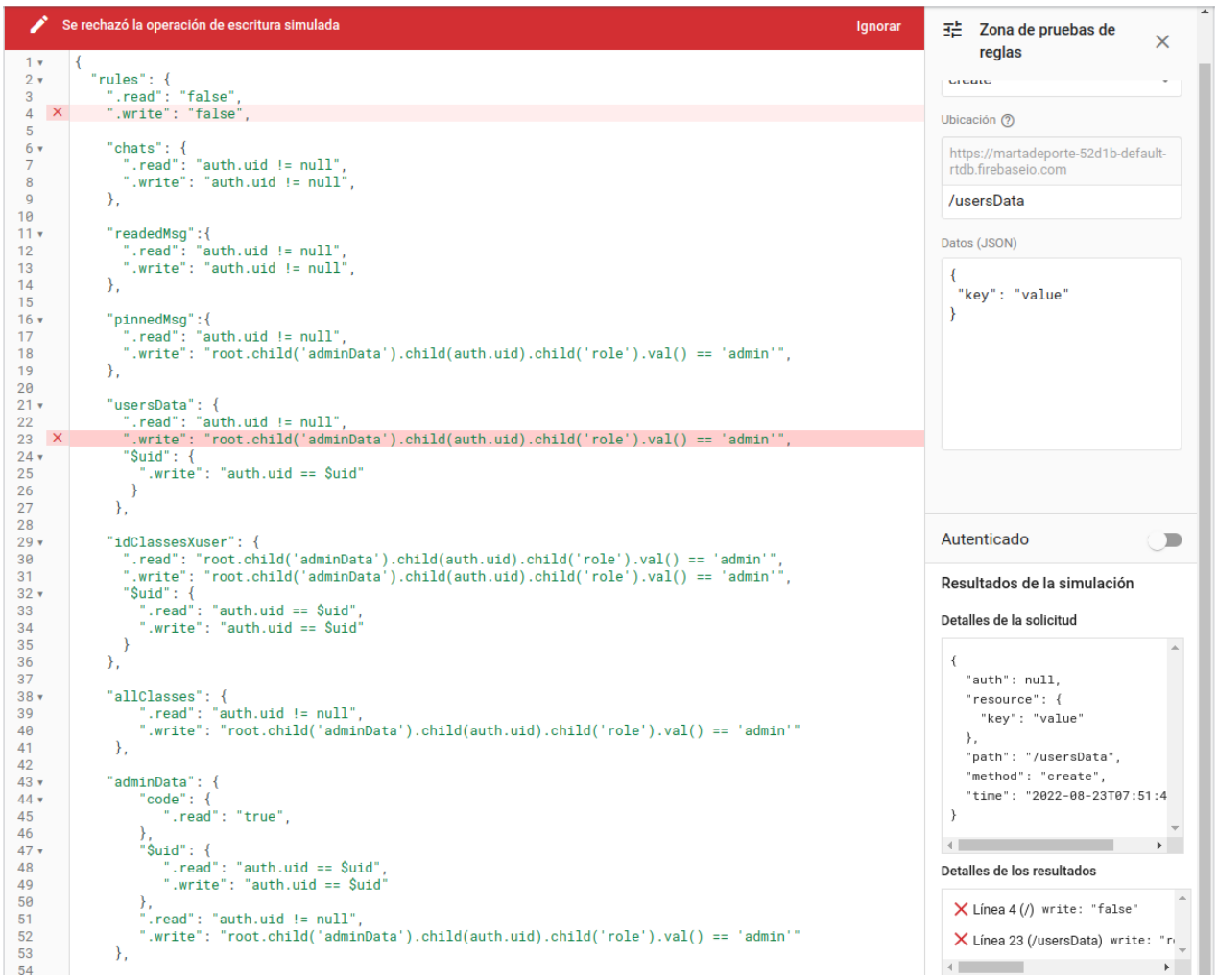



*Ilustración 64: permiso denegado bbdd*

Si establecemos que el usuario esté autenticado, **Ilustración 65**, ahora sí tendremos acceso a ese documento. La primera regla de la línea 3 no se tendrá en cuenta al ser aceptada la de la línea 22.

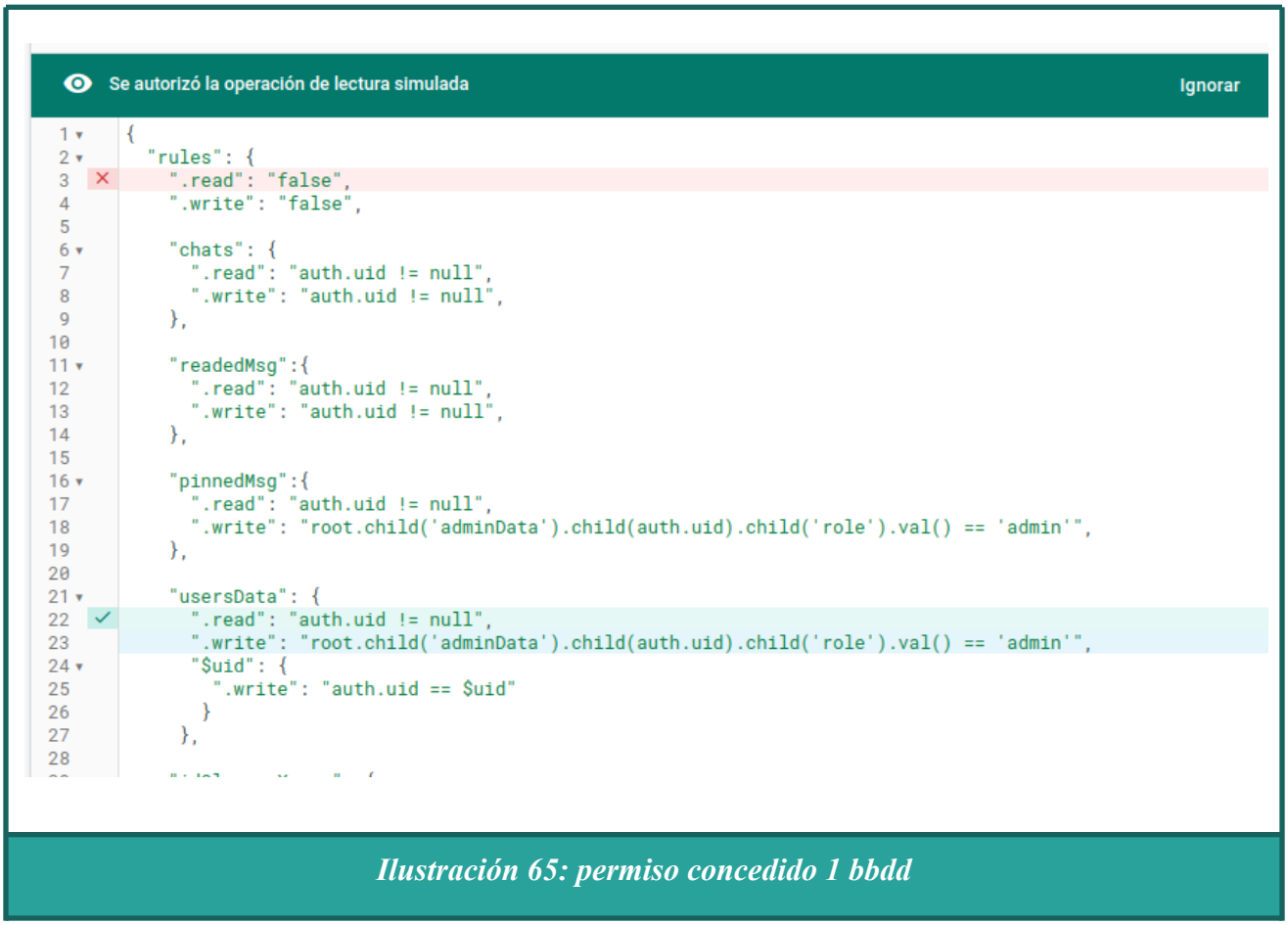



***Ilustración 65: permiso concedido 1 bbdd***

Si accedemos dentro de un id de usuario en concreto, **Ilustración 66**, con una operación de escritura, solo se permitirá si el identificador autentificado es igual al del documento que se quiere acceder.

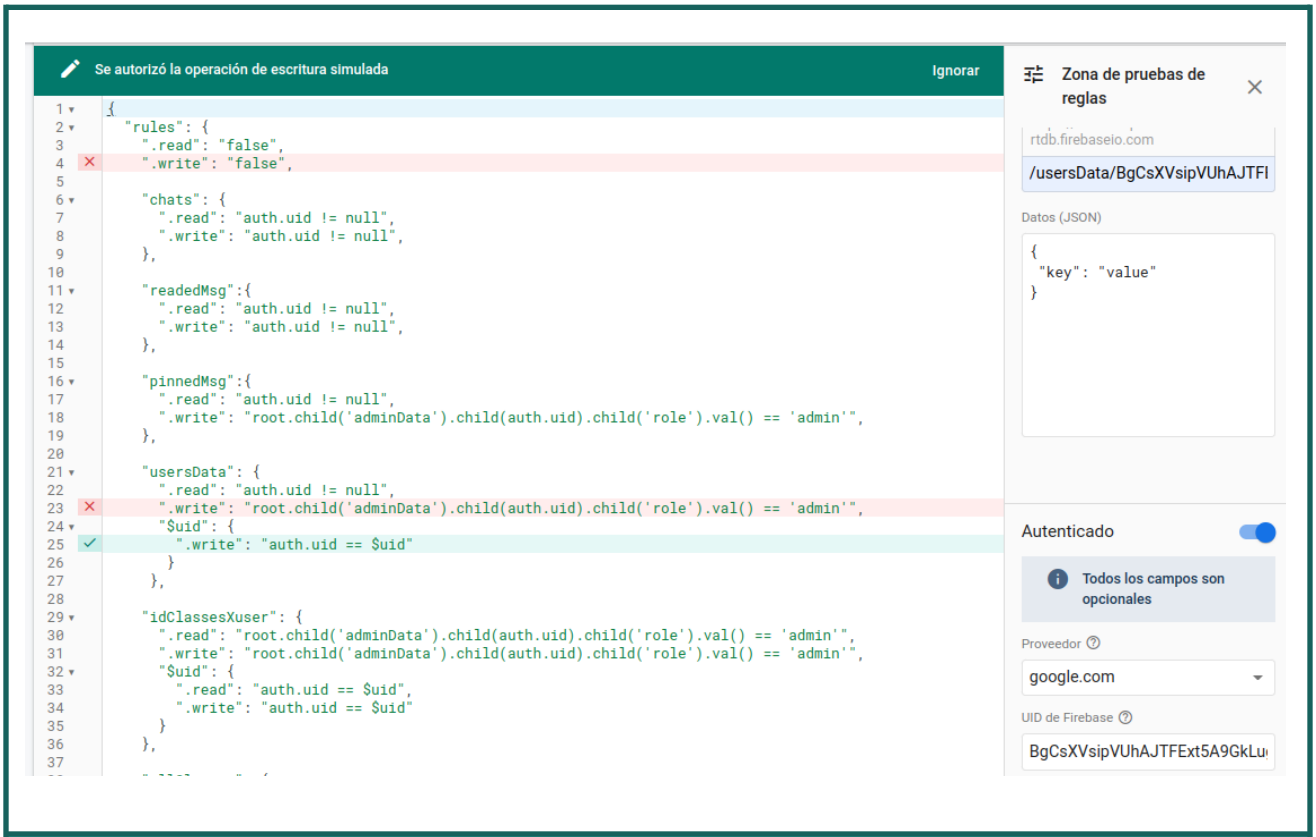



***Ilustración 66: permiso concedido 2 bbdd***

De esta forma se han ido probando todas las peticiones posibles para garantizar la seguridad de la base de datos. En promedio se realizaron 16 pruebas (8 de escritura, 8 lectura) para comprobar que los accesos permitidos realmente lo eran. Luego se realizaron otras 16 pruebas para comprobar que los accesos denegados realmente lo eran. Gracias a ellas puede detectar algunas zonas que no debían ser escritas por nadie excepto el administrador.

# 6. CONCLUSIONES

En este apartado hablaré del cumplimiento de los objetivos y de las dificultades encontradas.

Inicialmente pensé los objetivos acorde con las necesidades mínimas e intereses de la cliente, intentando buscar lo mínimo posible para que la aplicación pueda ser desarrollada y utilizada. Esto lo hice así por dos motivos: primero para asegurar que a pesar de los contratiempos que pudiera haber, cumplir con la meta de desarrollar la aplicación; y segundo, conseguir dedicar el suficiente tiempo a la aplicación para cuidar su interfaz y usabilidad. Al final pude cumplir tantos los objetivos principales como algunos secundarios. Gracias a que contaba con algo de experiencia usando Quasar, ciertas tareas se me agilizaron y pude indagar más en otros aspectos. Sin embargo, realmente no contaba con experiencia realizando un proyecto de esta magnitud yo solo. Tampoco tenía experiencia en bases de datos NoSQL y usando Firebase. Por ello, hay varias cosas que se me complicaron y no sabía cómo abarcar.

Tras conseguir realizar las tareas con las que tenía nula experiencia, fuí informándome y aprendiendo más. Con ello me pude dar cuenta de ciertos aspectos que no desarrollé con las mejores prácticas. Por lo que también invertí un tiempo en corregirlas. De hecho, todavía hay ciertos aspectos que no he podido cambiar, ya que tendría que haberlo tenido en cuenta desde el principio, y ahora sé que podrían estar mejor programadas/diseñadas. Como por ejemplo la base de datos y sus reglas de seguridad. Actualmente hay varias vulnerabilidades, que podrían mejorarse. Por ejemplo, en el sistema de chat, los mensajes se ordenan según la hora universal que el usuario envía desde su móvil, pero si este comando falla (o el usuario cambia la hora manualmente) el mensaje podría desordenarse. Por esto, la hora debería establecerse desde un servidor (Cloud Functions de Firebase, por ejemplo).

También la elección de la base de datos ha sido la correcta. Firebase cuenta con la base de datos Realtime y con la base de datos Firestore. Ambas son en tiempo real pero tienen varias diferencias. Una de ellas es la forma de facturación. Realtime cobra por GB transferido y Firestore por peticiones de lectura y escritura. Por ello, en el chat grupal, donde puede haber muchas lecturas, Realtime se adapta

mejor. Aunque otra información, como la de los usuarios, la cuál no se va a leer ni escribir tan a menudo, podría ir mejor en Firestore.

Algo que se me complicó fue el desarrollo de este documento, ya que era la primera vez que hacía un documento de esta magnitud y que incluye conocimientos en muchas áreas. La parte del análisis y diseño de la arquitectura frenó bastante el desarrollo general de la aplicación. A pesar de ello, al final pude salir del bloqueo y terminarlo..

Por último comentar que tras este proyecto he aprendido mucho sobre el desarrollo completo de una aplicación. Empezando por el diseño de su arquitectura, su organización (pude indagar más sobre kanban y scrum), su seguridad, el desarrollo back-end y front-end, y su despliegue. Por lo que, aunque me hubiera gustado incluir todos los objetivos secundarios (como el sistema de pagos o de reserva), me siento satisfecho con todo lo que he aprendido. Ya que ahora me siento con más confianza de empezar un proyecto desde el principio, poder organizarlo y desarrollarlo mucho mejor. Por lo que creo que el desarrollo de este Trabajo de Fin de Grado ha cumplido su función.

# BIBLIOGRAFÍA

**[1]** Atlassian. (2022). *¿Qué es ágil?* https://www.atlassian.com/es/agile // **(ver en el documento)**

**[2]** Atlassian. (2022). *Scrum: qué es, cómo funciona y por qué es excelente*. https://www.atlassian.com/es/agile/scrum // **(ver en el documento)**

**[3]** Atlassian. (2022). *Todo lo que necesitas saber sobre los sprints de scrum*. https://www.atlassian.com/es/agile/scrum/sprints // **(ver en el documento)**

**[4]** Atlassian. (2022). ¿Qué es un experto en scrum? https://www.atlassian.com/es/agile/scrum/scrum-master // **(ver en el documento)**

**[5]** Anderson, D. J. (2011). *Kanban: Cambio Evolutivo Exitoso Para su Negocio de Tecnología (Spanish Edition)*. Blue Hole Press. // **(ver en el documento)**

**[6]** Tamarit, R. G. (2021, 30 enero). *¿Qué es el Backlog?* Muy Agile. https://muyagile.com/que-es-el-backlog/ // **(ver en el documento)**

**[7]** Martinez, A. (2021, 17 septiembre). *Tipos de aplicaciones móviles: Nativas, web e híbridas*. Future Space S.A. https://www.futurespace.es/tipos-de-aplicaciones-moviles/ // **(ver en el documento)**

**[8]** R. (2022, 27 julio). *Why is native apps often more costly than hybrid app development?* RG Infotech. https://www.rginfotech.com/native-vs-hybrid-app-development-cost/ //**(ver en el documento)**

**[9]** *Cloudflare*. (2010). ResearchGate. https://www.researchgate.net // **(ver en el documento)**

**[10]** Sidana, M. (2018, 27 abril). *The Rise of Hybrid Apps - Mandy Sidana*. Medium. https://medium.com/@Mandysidana/the-rise-of-hybrid-apps-f101a825991b // **(ver en el documento)**

**[11]** *Introducción a Vue JS ≫ Qué es y sus características*. (2019, 3 noviembre). Coding Potions - Blog de programación y desarrollo web. https://codingpotions.com/que-es-vue // **(ver en el documento)**

**[12]** Lee, G. P. (2021, 13 diciembre). JavaScript Technical Interview Question: is React a MVC or MVVM? Medium. https://medium.com/developers-tomorrow/javascript // **(ver en el documento)**

**[13]** Getting Started - vue.js. (2022). Vue. https://012.vuejs.org/guide/ // **(ver en el documento)**

**[14]** DOM - Glosario | MDN. (2020, 8 diciembre). DOM. https://developer.mozilla.org/es/docs/Glossary/DOM // **(ver en el documento)**

**[15]** Reyes, R. S. (2021, 7 noviembre). Quasar - El Framework todo terreno de VueJS. Roylans.Dev - Blog Sobre Desarrollo Web. https://roylans.dev/quasar-framework // **(ver en el documento)**

**[16]** Backendless Corp. (2022, 28 marzo). Backendless | Visual App Development Platform UI Builder and Backend. Backendless. https://backendless.com // **(ver en el documento)**

**[17]** Busquets, C. (2021, 24 diciembre). MoSCoW: Qué es y cómo utilizarlo para priorizar. uiFromMars. https://www.uifrommars.com/priorizacion-metodo-moscow/ // **(ver en el documento)**

**[18]** Agile, E. (2021, 21 febrero). Agile Estimation Techniques: A Deep Dive Into T-Shirt Sizing. Easy Agile. https://www.easyagile.com/blog/agile-estimation-techniques/ // **(ver en el documento)**

**[19]** Precios Amazon Web Service S3 | Amazon Simple Storage Service. (2022). Amazon Web Services, Inc. https://aws.amazon.com/es/s3/pricing/ // **(ver en el documento)**

**[20]** Firebase Pricing. (2022). Firebase. https://firebase.google.com/pricing // **(ver en el documento)**

**[21]** Agencia Tributaria: 3.5.4 Tabla de amortización simplificada. (2022, 29 abril). Agencia. https://sede.agenciatributaria.gob.es/Sede/amortizacion-simplificada.html // **(ver en el documento)**

**[22]** Salario de Analista programador/a en España. (2022). Salarios. https://es.indeed.com/career/analista-programador/salaries // **(ver en el documento)**

**[23]** Mesa, M. L. (2022). Contabilidad y fiscalidad: Adaptación al PGC de las PYMES 2008 (Monografias Juridicas, Economicas y Sociales). Universidad de Jaén. Servicio de Publicaciones. // **(ver en el documento)**

**[24]** Phalcon - ODM (Object-Document Mapper) - Además de su capacidad para mapear tablas en bases de datos relacionales, Phalco - Español. (2022). ODM. https://runebook.dev/es/docs/phalcon/reference/odm // **(ver en el documento)**

**[25]** *Diseño de modelos de datos NoSQL*. (2020, 18 octubre). Blog de Andrés Hevia. https://andreshevia.com/2020/10/18/diseno-de-modelos-de-datos-nosql/ // **(ver en el documento)**

**[26]** *Renderización Condicional. Vue.js*. (2022). https://es.vuejs.org.html // **(ver en el documento)**

**[27]** pmoinformatica.com. (2022). *Pruebas de caja negra ISTQB*. La Oficina de Proyectos de Informática. http://www.pmoinformatica.com/pruebas-caja-negra // **(ver en el documento)**

**[28]** *Quasar CLI*. (2022). Quasar Framework. https://quasar.dev // **(ver en el documento)**

**[29]** *Add Firebase to your JavaScript project*. (2022). Firebase. https://firebase.google.com/docs/web/setup // **(ver en el documento)**

**[30]** *Directory Structure*. (2022). Quasar Framework. https://quasar.dev/quasar-cli-webpack/directory-structure // **(ver en el documento)**

**[31]** *Lifecycle Hooks | Vue.js*. (2022). Vue. https://vuejs.org/guide/ // **(ver en el documento)**

**[32]** *Slots â Vue.js*. (2022). Vue. https://es.vuejs.org/v2/guide/components // **(ver en el documento)**

**[33]** Fenollosa, A. (2020, 24 julio). ¡Deja de usar Cordova! Actualizate a Capacitor. Programador Web Valencia. https://programadorwebvalencia.com/con-capacitor/ // **(ver en el documento)**

**[34]** *What is Capacitor*. (2022). Quasar Framework. https://quasar.dev/quasar-cli-vite/developing-capacitor-apps/introduction // **(ver en el documento)**

**[35]** *Preguntas frecuentes de AWS Amplify | Servicios de front-end web y móviles | Amazon Web Services*. (2022). Amazon Web Services, Inc. https://aws.amazon.com // **(ver en el documento)**

**[36]** *Reglas de seguridad básicas | Reglas de seguridad de*. (2022). Firebase. https://firebase.google.com/docs/rules/basics?hl=es-419 // **(ver en el documento)**

# ANEXOS

## Vídeo general

En este apartado se adjunta un vídeo donde se puede ver rápidamente la mayor parte de la aplicación. El vídeo comienza enseñando las principales secciones, y señala los puntos clave de cada uno.

La realización de este vídeo tiene como objetivo facilitar la corrección y comprensión de la aplicación. El vídeo se puede ver en Google Drive a través del siguiente enlace:

**ACCEDER AL VÍDEO**

*Poner en calidad HD.

Hay algunas características que se ven y no se han explicado en el apartado "detalles de implementación" Como por ejemplo el tema oscuro o cambio de letra. El tema oscuro se consigue teniendo estilos diferentes para cada tema, Quasar permite conmutar entre ellos. El cambio del tamaño de letra se consigue al usar las unidades *rem* las cuáles son relativas al elemento *html*. Cambiando el porcentaje del tamaño de este elemento todas las letras cambian automáticamente. Ambas características hacen uso de *local storage* para guardar la configuración del usuario.

Durante todo el vídeo se muestra la versión online de la aplicación web. Pero la versión móvil funciona exactamente igual.

## Material suministrado

Junto a este documento se va a incluir el código de la aplicación, enlace a la aplicación web y el ejecutable (.apk) para Android. Todo se encuentra comprimido en la carpeta IgnacioCasaleLindeTFG-material. Dentro de ella está:

- IgnacioCasaleLindeMemoria.pdf: se trata de este documento.
- Ejecutable Android: carpeta que contiene el ejecutable para Android.
- EnlaceWeb.txt: contiene el enlace a la aplicación web (2022).
- Código: carpeta que contiene todo el código del proyecto.

## Notas adicionales

Como la aplicación tiene varias secciones se va a explicar como entrar a cada una de ellas. Para entrar a la sección de administración hay que usar los siguientes datos:

- Email: admin@gmail.com
- Contraseña: admin123.

Para entrar a la sección de alumnas los siguientes:

- Email: student@gmail.com
- Contraseña: student123.

Al chat se puede acceder desde ambas secciones. El tamaño máximo de mensajes de cada chat se ha limitado a 20.