Grado en Ingeniería Informática

Trabajo de Final de Grado

Desarrollo de una aplicación móvil de comunicación para pequeños municipios

Autor:
Miguel LIGERO NEBOT

Supervisor:
David Chiva Villalba
Tutor académico:
María José Aramburu Cabo

Fecha de lectura: 14 de julio de 2020
Curso académico 2019/2020

Resumen

A lo largo de este documento se reúne todo el trabajo realizado durante tres meses en la empresa *Wisclic Tech*, desarrollando un proyecto sobre una aplicación móvil para *Android* y *iOS* llamada "VueltAlPueblo". Utilizando una metodología predictiva en cascada se ha construido, mediante el kit de desarrollo de software *Flutter*, una plataforma que permite a los habitantes de pequeños municipios consultar los bandos y las noticias de su pueblo e inscribirse a eventos en su zona. En caso de ser administradores, esta plataforma permite gestionar las noticias y eventos que se publiquen en la aplicación así como las inscripciones de los asistentes a estos eventos. Esta memoria incluye la planificación del proyecto, diagramas y explicaciones sobre el análisis de los requisitos, los prototipos realizados en la fase de diseño, decisiones de implementación y conclusiones de la estancia en prácticas, entre otras cosas.

Palabras clave

Aplicación móvil, Flutter, Android, iOS, municipios, comunicación, eventos

Keywords

Mobile app, Flutter, Android, iOS, villages, communication, events

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

La estancia en prácticas ha tenido lugar en Wisclic Tech, una pequeña empresa situada en Espaitec formada por antiguos alumnos de la Universitat Jaume I. Los servicios que ofrecen se dividen en dos distintos. Por una parte, la empresa se encarga de desarrollar software a medida para sus clientes, estando especializados en aplicaciones web y móviles. Por otra parte, Wisclic Tech ofrece un sistema que da nombre a la propia empresa (Wisclic), y que se trata de una herramienta enfocada a facilitar el desarrollo de proyectos de aplicaciones web, haciendo que estos estén mejor organizados, proporcionando funcionalidades que reducen el tiempo de desarrollo, así como la gestión de tareas e incidencias.[1]

Uno de los proyectos de la empresa es una aplicación web dedicada a un municipio, Villanueva de Viver, en la cual se publican noticias y se gestionan eventos que vayan a tener lugar allí. Esto facilita a los organizadores toda la gestión de los eventos pero también a los asistentes, ya que pueden informarse de precios, fechas y condiciones especiales como por ejemplo, el tipo de menú de un evento con comida incluida, de una forma muy cómoda y fiable.

A partir de este proyecto, vieron que sería idóneo que esta aplicación web tuviera una versión para dispositivos móviles, de tal manera que facilitara aún más el acceso a los usuarios del municipio así como a los organizadores de eventos, teniendo un control más preciso de los asistentes. No obstante, decidieron que sería más óptimo crear una aplicación general, es decir, orientada a más de un municipio, ya que esto permitiría a los ciudadanos de un municipio poder enterarse de los sucesos de otro pueblo y no tener limitada la asistencia a eventos de su propio municipio.

De esta idea nace el proyecto desarrollado en la estancia en prácticas: una aplicación móvil, tanto para sistemas *Android* como para sistemas *iOS*, enfocada a facilitar la comunicación en pequeños municipios, destinada tanto para organizadores y administradores como para el público general.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es proporcionar una vía de información más cómoda a las personas pertenecientes a un pueblo en cuanto a bandos, noticias y eventos que se realicen allí.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Actualizar las formas de realizar un bando.
- Facilitar un acceso más rápido y cómodo a la información relacionada con un pueblo.
- Gestionar eventos de forma más sencilla.
- Monitorizar los asistentes a eventos de forma más efectiva.
- Unificar las cuentas de varias páginas web de ayuntamientos en una sola aplicación.
- Unificar aplicaciones de distintos pueblos para ahorrar espacio de memoria en los dispositivos de los usuarios pertenecientes o interesados en varios pueblos.

El alcance de esta aplicación incluye la redacción y publicación de los bandos y las noticias por parte de los administradores y la visualización de estos por parte de los usuarios. Además, la publicación de nuevos bandos se notificará a los usuarios para que no pasen desapercibidos si no entran en la aplicación a comprobar las novedades. En cuanto a lo relacionado con los eventos, los administradores podrán crearlos, modificarlos y eliminarlos. Los usuarios, en cambio, podrán visualizar los eventos disponibles e inscribirse en ellos.

Por otra parte, el alcance no incluye la creación de las comunidades ni la inserción de nuevos pueblos debido a que es una tarea externa a la aplicación móvil. Además, tampoco se incluye la implementación de los métodos relacionados con el borrado y la modificación de bandos en la base de datos.

En cuanto al alcance informático, para este proyecto no se necesitará ningún tipo de hardware específico. Los servidores usados serán de dos tipos: los servidores estándar que ya están en activo para almacenar los datos y los de *Firebase* (plataforma en línea que ofrece *Google* para el desarrollo de aplicaciones), para hacer labores de notificaciones en tiempo real y almacenado de bandos. Además, el equipo utilizado para desarrollar la aplicación será el propio del estudiante.

En lo que respecta al alcance organizativo, la empresa no intervendrá, aparte de la labor de supervisión, en el desarrollo de la aplicación móvil y por tanto, el proyecto lo realizará una única persona. No obstante, el proyecto de la web sigue en desarrollo, por lo que habrá que poner en común diferentes apartados y no será un proyecto totalmente aislado.

1.3. Descripción del proyecto

Aunque el concepto del proyecto se basa en la aplicación web creada para el municipio de Villanueva de Viver, la construcción de la aplicación empieza desde cero. Para implementar

la aplicación existen distintas opciones de entre las cuales la empresa ha escogido *Flutter*, un *framework* creado por *Google* que permite desarrollar a la vez aplicaciones móviles tanto para *Android* como para *iOS*.

La aplicación es íntegramente *front-end*, es decir, se basa en mostrar datos de una manera gráfica para los usuarios. Estos datos proceden del *back-end*, el cual en este proyecto está implementado en el sistema propio de la empresa llamado *Wisclic*. Además, se emplea *Firebase* para almacenar los bandos de los distintos municipios, y el sistema *Firebase Cloud Messaging* para notificar a los usuarios cuando se crea un nuevo bando en alguno de los pueblos a los que están suscritos. La aplicación se encargará de hacer las llamadas pertinentes a la *API* y así poder realizar las diferentes funciones propuestas. Por ello, la aplicación requiere conexión a Internet, ya que si no la funcionalidad es prácticamente nula.

Como editor de código fuente se emplea *Visual Studio Code*, un editor muy versátil y completo, capaz de soportar las tecnologías a utilizar y poder ver los resultados en un dispositivo móvil real.

Además de esto, otras herramientas adicionales usadas en el proyecto son: *Trello*, una aplicación web para la administración del proyecto; *Figma*, un software para diseñar las interfaces de usuario de la aplicación; *MS Project*, un programa creado por *Microsoft* para la gestión de proyectos; y *Lucidchart*, una web para crear en línea diagramas de diferentes tipos.

1.4. Estructura de la memoria

Para organizar esta memoria se ha seguido una estructura divida en capítulos, cada uno centrado en una o dos de las fases de las que consta el proyecto.

En el capítulo 2 veremos la metodología de planificación que se ha seguido así como una estimación de recursos y costes del proyecto.

En el capítulo 3 veremos tanto el análisis del proyecto como el diseño de bases de datos y de interfaces de usuario.

En el capitulo 4 nos centraremos en la fase de implementación así como de las pruebas llevadas a cabo para comprobar la robustez de la aplicación.

Finalmente, en el capitulo 5 se presentan las conclusiones de este proyecto en todos los ámbitos: profesional, formativo y personal.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Al tratarse de un proyecto con unos requisitos fijos y ser un trabajo llevado a cabo por solamente una persona, se ha decidido emplear una metodología tradicional guiada por el PMBOK (*Project Management Body of Knowledge*), cuyo ciclo de vida es en cascada. El modelo de ciclo de vida en cascada se basa en que las diferentes fases del proyecto deben empezar y finalizar de manera lineal y secuencial, así que cuando se termina una fase, empieza la siguiente y no se vuelve a ella. En la figura 2.1 podemos ver el esquema de la planificación en cascada llevada a cabo en el proyecto.

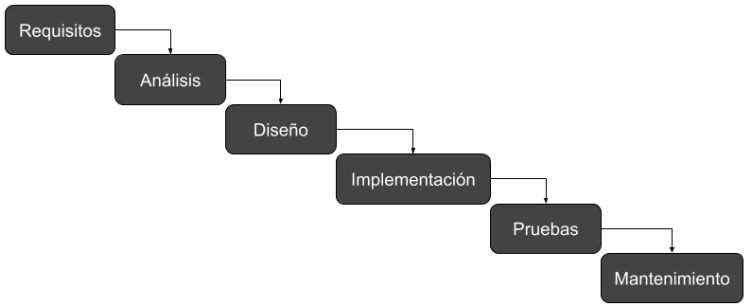


Figura 2.1: Ciclo de vida en cascada del proyecto.

La elección de esta metodología se debe a que, además de por recomendación de la propia empresa, el factor de ser una única persona hace que sea más recomendable tener toda la planificación hecha desde el principio y ceñirse a ella para ahorrar el mayor tiempo posible. Más adelante, en la sección 2.4, se explica la implicación del supervisor de la empresa en este proyecto.

2.2. Planificación

Para llevar a cabo la planificación del proyecto según la metodología predictiva es necesario, en primer lugar, realizar la definición de las actividades que se harán durante todo el proyecto. Para ello, se construyó en la página web *Trello* un tablero dónde se definían todas las actividades que se llevarían a término en este proyecto. El tablero tenía originalmente esta estructura formada por ocho paquetes:

1) **Inicio**
    a) Identificar usuarios
    b) Recopilar requisitos
    c) Definir alcance del sistema
    d) Aprendizaje básico de *Flutter* y *Dart*
    e) Firma de la propuesta técnica
2) **Planificación**
    a) Definir actividades
    b) Crear diagrama EDT (Estructura de Descomposición del Trabajo)
    c) Estimar costes
3) **Análisis**
    a) Definición de requisitos
    b) Diagrama de casos de uso
    c) Diagrama de clases
4) **Diseño**
    a) Diseño de la base de datos
        (I) Diseño conceptual
        (II) Diseño lógico
    b) Diseño de interfaces de usuario
        (I) Módulo 'Bandos'
        (II) Módulo 'Noticias'
        (III) Módulo 'Eventos'
5) **Implementación**
    a) Implementar interfaces

(I) Módulo 'Bandos'
(II) Módulo 'Noticias'
(III) Módulo 'Eventos'

b) Programación

(I) Módulo 'Registro'
(II) Integración con *API*

6) **Pruebas**

a) Pruebas de aceptación de usuarios
b) Pruebas de integración

7) **Cierre y puesta en marcha**

a) Cerrar el proyecto

8) **Seguimiento y control**

a) Reuniones semanales

Otra de las actividades para la planificación del proyecto es la creación de un EDT (Estructura de descomposición del trabajo), que se trata de un esquema en el cual se reflejan todas las fases de un proyecto y las actividades que contiene cada una de ellas. En la figura 2.2 se muestra el EDT realizado en este proyecto, el cual incluye las actividades definidas anteriormente. A continuación del diagrama se incluye el diccionario del EDT en las siguientes tablas, desde la 2.1 hasta la 2.8, una tabla para cada paquete del diagrama.

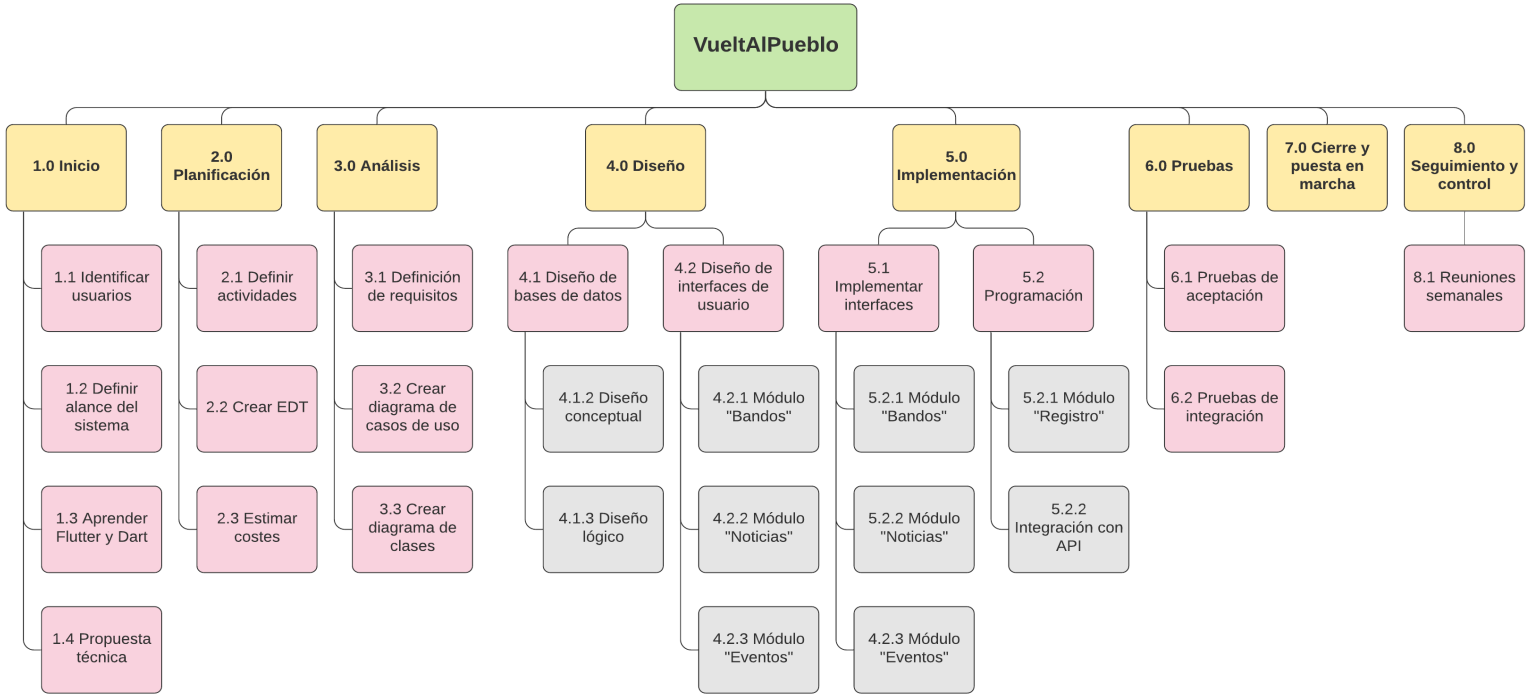


Figura 2.2: Diagrama de Estructura de Descomposición de Trabajo (EDT) del proyecto.

Código del paquete de trabajo	**Nombre del paquete de trabajo**
1.	Inicio
Objetivo:	Comenzar con el proyecto
Descripción:	En este paquete se realizan las actividades relacionadas con los requisitos del proyecto, acabando en la redacción y entrega de la propuesta técnica.
Actividades incluidas:	• Identificar usuarios • Definir alcance del sistema • Aprendizaje de Flutter y Dart • Propuesta técnica
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Tutora **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Tutora **Requisitos a cumplir:** En la propuesta técnica se debe incluir la descripción y objetivos del proyecto, la metodología, planificación y las herramientas a utilizar. **Forma de aceptación:** La propuesta se evaluará como "satisfactoria" o "no satisfactoria" .
Fechas:	**Inicio:** 11/02/2020 **Fin:** 19/02/2020

Tabla 2.1: Paquete Inicio.

Código del paquete de trabajo	**Nombre del paquete de trabajo**
2.	Planificación
Objetivo:	Realizar la planificación del proyecto.
Descripción:	Siguiendo la metodología predictiva, se realiza la planificación para todo el proyecto.
Actividades incluidas:	• Definir actividades • Crear EDT • Estimar costes
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Tutora y Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Tutora **Requisitos a cumplir:** Costes temporales no superan las 450 horas. **Forma de aceptación:** La propuesta se evaluará como "satisfactoria" o "no satisfactoria" .
Fechas:	**Inicio:** 19/02/2020 **Fin:** 21/02/2020

Tabla 2.2: Paquete Planificación.

Código del paquete de trabajo	Nombre del paquete de trabajo
3.	Análisis
Objetivo:	Obtener el análisis de los requisitos del proyecto.
Descripción:	En este paquete se definen los requisitos presentados en la fase de inicio y se realizan las actividades de análisis necesarias para ello.
Actividades incluidas:	• Definición de requisitos • Diagrama de casos de uso • Diagrama de clases
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Supervisor **Requisitos a cumplir:** Realizar todas las actividades incluidas en el paquete. **Forma de aceptación:** Sin concretar.
Fechas:	**Inicio:** 21/02/2020 **Fin:** 25/03/2020

Tabla 2.3: Paquete Análisis.

Código del paquete de trabajo	**Nombre del paquete de trabajo**
4.	Diseño
Objetivo:	Realizar los diseños necesarios para el proyecto.
Descripción:	En esta fase se realizan todos los diseños que se emplearán en el proyecto, tanto de bases de datos como de interfaces de usuario.
Actividades incluidas:	• Diseño conceptual de la base de datos. • Diseño lógico de la base de datos. • Diseño de interfaces de usuario: Módulo "Eventos" Módulo "Noticias" Módulo "Bandos"
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Supervisor **Requisitos a cumplir:** Realizar todas las actividades incluidas en el paquete. **Forma de aceptación:** Sin concretar.
Fechas:	**Inicio:** 25/02/2020 **Fin:** 05/03/2020

Tabla 2.4: Paquete Diseño.

Código del paquete de trabajo	Nombre del paquete de trabajo
5.	Implementación
Objetivo:	Implementar un código que cumpla los requisitos.
Descripción:	En esta fase se realiza toda la implementación del código de la plataforma.
Actividades incluidas:	• Implementar interfaces de usuario: Módulo "Eventos" Módulo "Noticias" Módulo "Bandos" • Implementar módulo "Registro" • Integración con API
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Supervisor **Requisitos a cumplir:** Implementar todos los requisitos de la plataforma. **Forma de aceptación:**
Fechas:	**Inicio:** 05/03/2020 **Fin:** 22/05/2020

Tabla 2.5: Paquete Implementación.

Código del paquete de trabajo	Nombre del paquete de trabajo
6.	Pruebas
Objetivo:	Comprobar que el código funciona correctamente.
Descripción:	En esta fase se comprueba que el código implementado en la fase anterior funciona correctamente y cumple los requisitos.
Actividades incluidas:	• Pruebas de integración • Pruebas de aceptación de usuarios
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Supervisor **Requisitos a cumplir:** La aplicación permite realizar todos los casos de uso sin problemas. **Forma de aceptación:**
Fechas:	**Inicio:** 22/05/2020 **Fin:** 25/05/2020

Tabla 2.6: Paquete Pruebas.

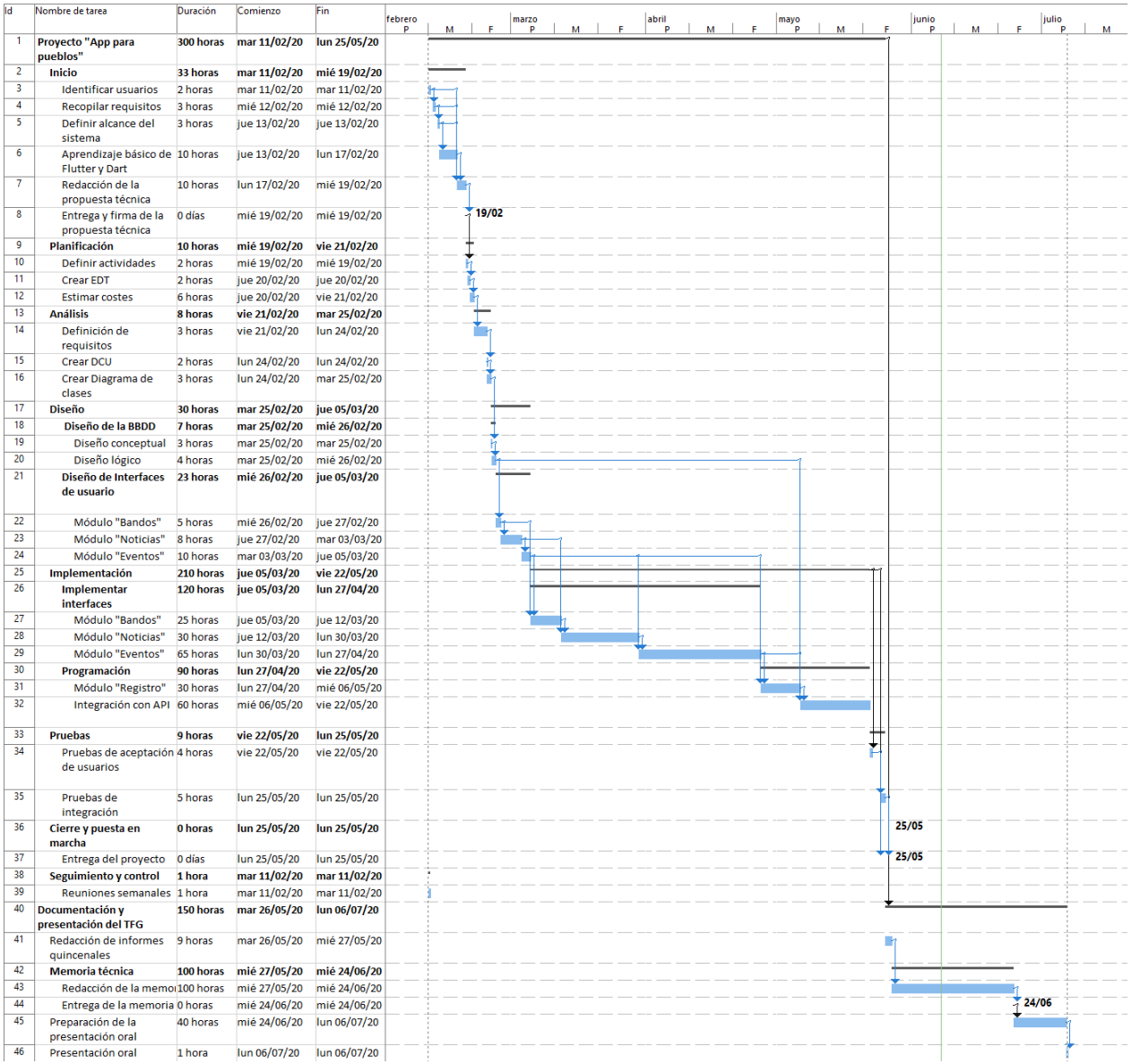
Código del paquete de trabajo	**Nombre del paquete de trabajo**
7.	Cierre y puesta en marcha
Objetivo:	Finalizar el proyecto
Descripción:	Se procede al cierre del proyecto y se evalúa el trabajo realizado.
Actividades incluidas:	• Cierre del proyecto
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Supervisor **Requisitos a cumplir:** Ninguno en concreto. **Forma de aceptación:**
Fecha:	25/05/2020

Tabla 2.7: Paquete Cierre y puesta en marcha.

Código del paquete de trabajo	Nombre del paquete de trabajo
8.	Seguimiento y control
Objetivo:	Llevar un seguimiento del trabajo realizado en la empresa.
Descripción:	Actividades que ayudan a llevar un control sobre el desarrollo del proyecto durante la estancia en prácticas.
Actividades incluidas:	• Reuniones semanales • Informes quincenales
Responsabilidades:	**Responsable:** Miguel **Aprueba:** Tutora y Supervisor **Consultado:** Supervisor **Informado:** Supervisor
Criterios de aceptación:	**Responsable de aceptación:** Tutora **Requisitos a cumplir:** Entregar un informe cada quince días sobre el trabajo realizado en ese tiempo. **Forma de aceptación:** La tutora corregirá los informes y los comentará con el alumno.
Fechas:	Tareas periódicas

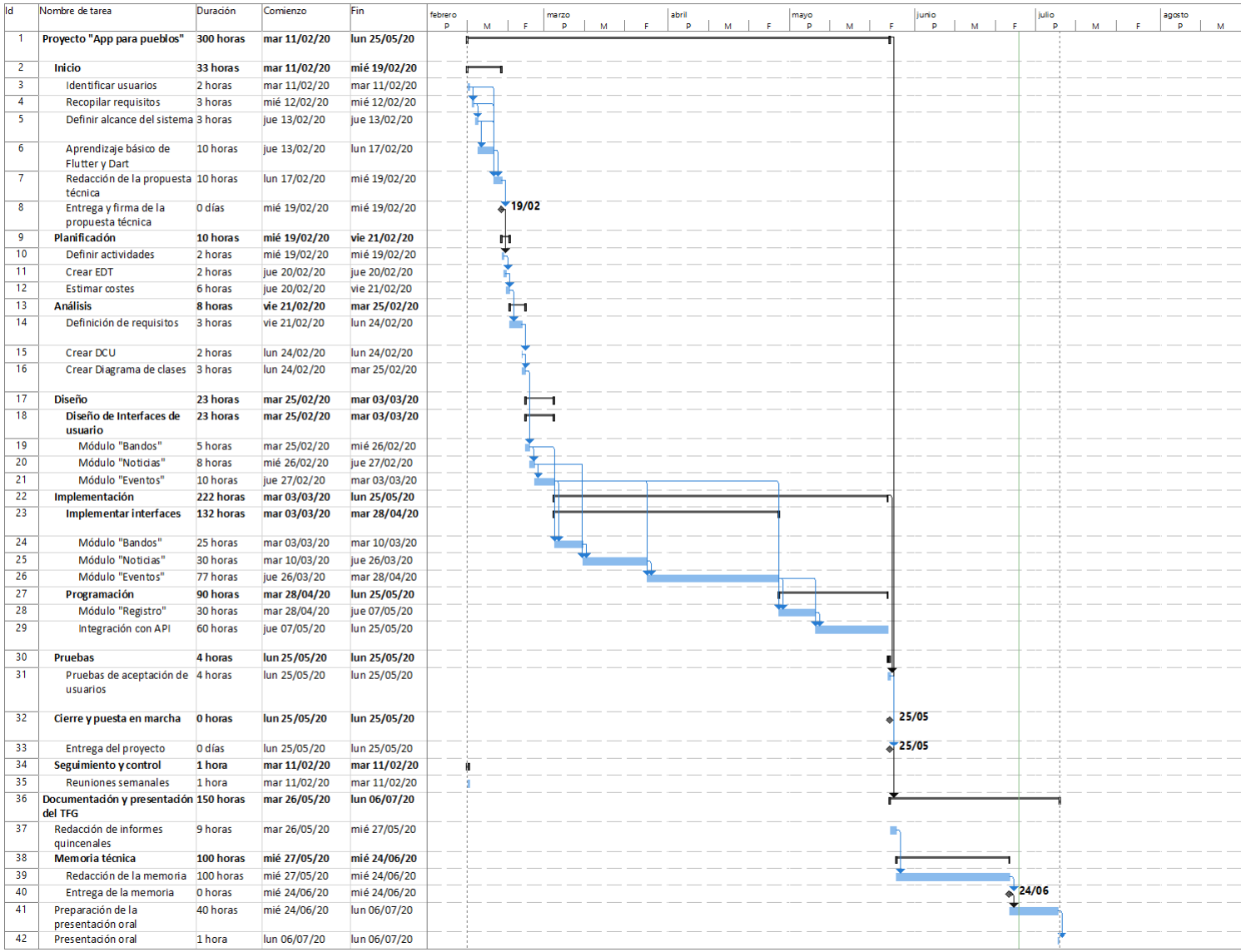
Tabla 2.8: Paquete Seguimiento y control.

Además de definir las actividades a realizar durante todo el proyecto también se ha realizado la planificación temporal de estas actividades. Para ello, como podemos ver en la figura 2.3, se ha realizado un diagrama de *Gantt* en el cual se muestran todas las actividades y su distribución en el calendario. No obstante, como se explica en secciones posteriores, este diagrama es una versión inicial ya que el diseño de la base de datos y las pruebas de integridad no se llegaron a realizar por diversas circunstancias. El diagrama de *Gantt* final se corresponde con la figura 2.4, en el cual el tiempo destinado a estas actividades se ha empleado en la implementación del módulo 'Eventos'.



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto "App para pueblos"**	**300 horas**	**mar 11/02/20**	**lun 25/05/20**
2	**Inicio**	**33 horas**	**mar 11/02/20**	**mié 19/02/20**
3	Identificar usuarios	2 horas	mar 11/02/20	mar 11/02/20
4	Recopilar requisitos	3 horas	mié 12/02/20	mié 12/02/20
5	Definir alcance del sistema	3 horas	jue 13/02/20	jue 13/02/20
6	Aprendizaje básico de Flutter y Dart	10 horas	jue 13/02/20	lun 17/02/20
7	Redacción de la propuesta técnica	10 horas	lun 17/02/20	mié 19/02/20
8	Entrega y firma de la propuesta técnica	0 días	mié 19/02/20	mié 19/02/20
9	**Planificación**	**10 horas**	**mié 19/02/20**	**vie 21/02/20**
10	Definir actividades	2 horas	mié 19/02/20	mié 19/02/20
11	Crear EDT	2 horas	jue 20/02/20	jue 20/02/20
12	Estimar costes	6 horas	jue 20/02/20	vie 21/02/20
13	**Análisis**	**8 horas**	**vie 21/02/20**	**mar 25/02/20**
14	Definición de requisitos	3 horas	vie 21/02/20	lun 24/02/20
15	Crear DCU	2 horas	lun 24/02/20	lun 24/02/20
16	Crear Diagrama de clases	3 horas	lun 24/02/20	mar 25/02/20
17	**Diseño**	**30 horas**	**mar 25/02/20**	**jue 05/03/20**
18	**Diseño de la BBDD**	**7 horas**	**mar 25/02/20**	**mié 26/02/20**
19	Diseño conceptual	3 horas	mar 25/02/20	mar 25/02/20
20	Diseño lógico	4 horas	mar 25/02/20	mié 26/02/20
21	**Diseño de Interfaces de usuario**	**23 horas**	**mié 26/02/20**	**jue 05/03/20**
22	Módulo "Bandos"	5 horas	mié 26/02/20	jue 27/02/20
23	Módulo "Noticias"	8 horas	jue 27/02/20	mar 03/03/20
24	Módulo "Eventos"	10 horas	mar 03/03/20	jue 05/03/20
25	**Implementación**	**210 horas**	**jue 05/03/20**	**vie 22/05/20**
26	**Implementar interfaces**	**120 horas**	**jue 05/03/20**	**lun 27/04/20**
27	Módulo "Bandos"	25 horas	jue 05/03/20	jue 12/03/20
28	Módulo "Noticias"	30 horas	jue 12/03/20	lun 30/03/20
29	Módulo "Eventos"	65 horas	lun 30/03/20	lun 27/04/20
30	**Programación**	**90 horas**	**lun 27/04/20**	**vie 22/05/20**
31	Módulo "Registro"	30 horas	lun 27/04/20	mié 06/05/20
32	Integración con API	60 horas	mié 06/05/20	vie 22/05/20
33	**Pruebas**	**9 horas**	**vie 22/05/20**	**lun 25/05/20**
34	Pruebas de aceptación de usuarios	4 horas	vie 22/05/20	vie 22/05/20
35	Pruebas de integración	5 horas	lun 25/05/20	lun 25/05/20
36	**Cierre y puesta en marcha**	**0 horas**	**lun 25/05/20**	**lun 25/05/20**
37	Entrega del proyecto	0 días	lun 25/05/20	lun 25/05/20
38	**Seguimiento y control**	**1 hora**	**mar 11/02/20**	**mar 11/02/20**
39	Reuniones semanales	1 hora	mar 11/02/20	mar 11/02/20
40	**Documentación y presentación del TFG**	**150 horas**	**mar 26/05/20**	**lun 06/07/20**
41	Redacción de informes quincenales	9 horas	mar 26/05/20	mié 27/05/20
42	**Memoria técnica**	**100 horas**	**mié 27/05/20**	**mié 24/06/20**
43	Redacción de la memoi	100 horas	mié 27/05/20	mié 24/06/20
44	Entrega de la memoria	0 horas	mié 24/06/20	mié 24/06/20
45	Preparación de la presentación oral	40 horas	mié 24/06/20	lun 06/07/20
46	Presentación oral	1 hora	lun 06/07/20	lun 06/07/20

Figura 2.3: Diagrama de *Gantt* inicial del proyecto.



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto "App para pueblos"**	**300 horas**	**mar 11/02/20**	**lun 25/05/20**
2	**Inicio**	**33 horas**	**mar 11/02/20**	**mié 19/02/20**
3	Identificar usuarios	2 horas	mar 11/02/20	mar 11/02/20
4	Recopilar requisitos	3 horas	mié 12/02/20	mié 12/02/20
5	Definir alcance del sistema	3 horas	jue 13/02/20	jue 13/02/20
6	Aprendizaje básico de Flutter y Dart	10 horas	jue 13/02/20	lun 17/02/20
7	Redacción de la propuesta técnica	10 horas	lun 17/02/20	mié 19/02/20
8	Entrega y firma de la propuesta técnica	0 días	mié 19/02/20	mié 19/02/20
9	**Planificación**	**10 horas**	**mié 19/02/20**	**vie 21/02/20**
10	Definir actividades	2 horas	mié 19/02/20	mié 19/02/20
11	Crear EDT	2 horas	jue 20/02/20	jue 20/02/20
12	Estimar costes	6 horas	jue 20/02/20	vie 21/02/20
13	**Análisis**	**8 horas**	**vie 21/02/20**	**mar 25/02/20**
14	Definición de requisitos	3 horas	vie 21/02/20	lun 24/02/20
15	Crear DCU	2 horas	lun 24/02/20	lun 24/02/20
16	Crear Diagrama de clases	3 horas	lun 24/02/20	mar 25/02/20
17	**Diseño**	**23 horas**	**mar 25/02/20**	**mar 03/03/20**
18	**Diseño de Interfaces de usuario**	**23 horas**	**mar 25/02/20**	**mar 03/03/20**
19	Módulo "Bandos"	5 horas	mar 25/02/20	mié 26/02/20
20	Módulo "Noticias"	8 horas	mié 26/02/20	jue 27/02/20
21	Módulo "Eventos"	10 horas	jue 27/02/20	mar 03/03/20
22	**Implementación**	**222 horas**	**mar 03/03/20**	**lun 25/05/20**
23	**Implementar interfaces**	**132 horas**	**mar 03/03/20**	**mar 28/04/20**
24	Módulo "Bandos"	25 horas	mar 03/03/20	mar 10/03/20
25	Módulo "Noticias"	30 horas	mar 10/03/20	jue 26/03/20
26	Módulo "Eventos"	77 horas	jue 26/03/20	mar 28/04/20
27	**Programación**	**90 horas**	**mar 28/04/20**	**lun 25/05/20**
28	Módulo "Registro"	30 horas	mar 28/04/20	jue 07/05/20
29	Integración con API	60 horas	jue 07/05/20	lun 25/05/20
30	**Pruebas**	**4 horas**	**lun 25/05/20**	**lun 25/05/20**
31	Pruebas de aceptación de usuarios	4 horas	lun 25/05/20	lun 25/05/20
32	**Cierre y puesta en marcha**	**0 horas**	**lun 25/05/20**	**lun 25/05/20**
33	Entrega del proyecto	0 días	lun 25/05/20	lun 25/05/20
34	**Seguimiento y control**	**1 hora**	**mar 11/02/20**	**mar 11/02/20**
35	Reuniones semanales	1 hora	mar 11/02/20	mar 11/02/20
36	**Documentación y presentación del TFG**	**150 horas**	**mar 26/05/20**	**lun 06/07/20**
37	Redacción de informes quincenales	9 horas	mar 26/05/20	mié 27/05/20
38	**Memoria técnica**	**100 horas**	**mié 27/05/20**	**mié 24/06/20**
39	Redacción de la memoria	100 horas	mié 27/05/20	mié 24/06/20
40	Entrega de la memoria	0 horas	mié 24/06/20	mié 24/06/20
41	Preparación de la presentación oral	40 horas	mié 24/06/20	lun 06/07/20
42	Presentación oral	1 hora	lun 06/07/20	lun 06/07/20

Figura 2.4: Diagrama de *Gantt* final del proyecto.

2.3. Estimación de recursos y costes del proyecto

En un principio la estancia en prácticas tuvo lugar en la oficina de la empresa *Wisclic Tech*, que se encuentra en la primera planta del edificio Espaitec 2, situado en el campus de la Universitat Jaume I. Al ser una pequeña *start-up* enfocada en el desarrollo de software, toda su actividad se concentra en esta oficina. No obstante, a causa de la situación provocada por el COVID-19, todo el proyecto restante se realizó mediante el teletrabajo, utilizando la herramienta *Hangouts* para comunicarse con el resto de compañeros.

En cuanto a los recursos que se han utilizado en este proyecto se pueden dividir en tres tipos: humanos, hardware y software. A continuación se detallan cada uno de estos tipos.

2.3.1. Recursos humanos

El proyecto ha sido llevado a cabo por una persona, que ha realizado tareas tanto de diseñador web como de desarrollador. El desglose del coste es:

Puesto: Programador Junior a 9€/h

Salario medio: 1560€/mes (jornada completa de 40h)

Salario jornada de 25h: 900€/mes

Duración proyecto: 300 horas = 3 meses

Coste: 900 * 3 = 2700€

2.3.2. Recursos hardware

Por lo que corresponde a recursos hardware se ha utilizado un ordenador portátil, propiedad del alumno, para programar la aplicación móvil. Sus características son:

HP Pavilion 15-BC303NS: 619€

- Sistema Operativo: *Windows 10 Home*
- Procesador: *Intel Core i5-7200U*
- Memoria RAM: 8GB

Además, para probar la aplicación no se usa ningún simulador, sino que se utiliza un dispositivo móvil real. En este caso, el terminal empleado es un *Oneplus 5T*, también propiedad del alumno.

2.3.3. Recursos software

Para este proyecto todos el software utilizado se trata de software libre y por lo tanto no supone ningún tipo de coste. A continuación se muestra todo el software utilizado:

- *Flutter*: *SDK* (*Software Development Kit*) de código fuente abierto escrito en el lenguaje de programación *Dart*.
- *Visual Studio Code*: editor de código fuente desarrollado por *Microsoft*.
- *Firebase*: plataforma para el desarrollo de aplicaciones web y aplicaciones móviles desarrollada por *Google*.

2.4. Seguimiento del proyecto

Durante toda la estancia en prácticas se han producido dos tipos de actividades para el seguimiento del progreso del proyecto: reuniones en la empresa e informes quincenales.

Respecto a las reuniones, se trataban de charlas breves con todos los compañeros de la empresa en las cuales se hablaba del trabajo hecho hasta ese momento, así como del trabajo más inmediato por hacer. Estas reuniones se producían con una frecuencia semanal, no obstante, siempre que se acababa alguna funcionalidad importante de la aplicación se mostraba a los compañeros para saber si el acabado era bueno y así poder pasar de funcionalidad. Además, también servían para tomar decisiones sobre los problemas que iban surgiendo, para encontrar entre todos la solución más óptima.

Por otra parte, cada quince días laborales se presentaba un informe con una estructura concreta. En él se explicaba en primer lugar el trabajo realizado en la quincena anterior, después el trabajo realizado en la quincena actual y finalmente el trabajo a realizar en la siguiente. De esta manera, la tutora ha podido observar el progreso del alumno y dar *feedback* sobre cualquier detalle necesario.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

La fase de análisis de un proyecto se caracteriza por ser la parte en la cual se especifican los requisitos a tener en cuenta. Aunque en la fase de inicio se haga una descripción del proyecto y se nombren todos o gran parte de los requisitos, no se suele entrar en detalle sobre estos. Es por eso que en esta fase se procede a realizar la definición de estos requisitos para entender mejor la plataforma a desarrollar en las siguientes actividades.

3.1.1. Diagrama de casos de uso

Para representar todos los casos de uso que tendrán lugar en la plataforma se ha creado un diagrama de casos de uso (figura 3.1) en el cual se distinguen cuatro actores:

- **Usuario no registrado**: Representa a aquellas personas que utilizan la plataforma sin haberse registrado previamente.
- **Usuario registrado**: Representa a todo individuo con cuenta en la plataforma.
- **Administrador de comunidad**: Representa a aquel individuo con permisos adicionales en la plataforma.
- ***Firebase***: Representa el servicio de notificaciones de *Google*.

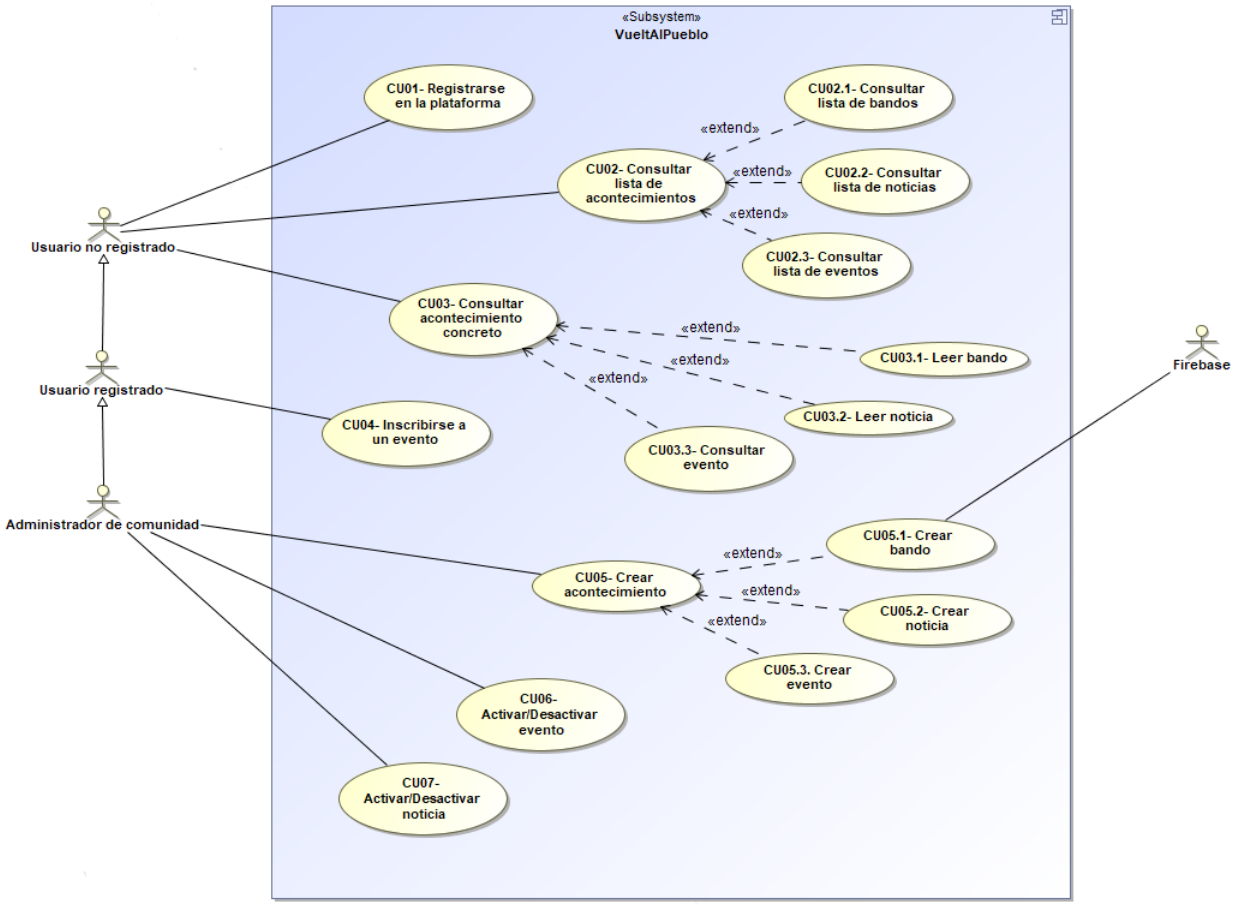


Figura 3.1: Diagrama de casos de uso del proyecto.

En las siguientes tablas (Tabla 3.1 a Tabla 3.7) se define cada uno de los casos de uso que aparecen en el diagrama anterior y se amplía la información sobre ellos.

Especificación del caso de uso	
Identificador	CU01
Nombre	Registrarse en la plataforma
Versión	1.0
Descripción	El sistema tiene que permitir que un usuario no registrado pueda registrarse en la plataforma.
Alcance	Permitir a alguien crear una cuenta en la plataforma.
Nivel	Tarea principal
Actores	Usuario no registrado
Relaciones	Ninguna
Precondición	El usuario no tiene que estar registrado en la plataforma.
Condición final con éxito	El usuario pasa a estar registrado en la plataforma.

Condición final con fracaso	El usuario no se registra en la plataforma.
Trigger	Querer acceder a las funcionalidades de la plataforma.
Secuencia normal	
1	El usuario inicia la aplicación.
2	El usuario elige la opción “registrarse” en la aplicación.
3	El sistema muestra los campos de datos a rellenar.
4	El usuario rellena los campos y confirma la acción.
5	El sistema registra los datos.
Excepciones <acción 4>	Excepción Usuario ya registrado
1	Si el usuario ya tiene una cuenta, el sistema no registra los datos.
Frecuencia esperada	Diaria
Importancia	Necesario
Prioridad	Corto plazo

Tabla 3.1: CU01 - Registrarse en la plataforma

Especificación del caso de uso	
Identificador	CU02
Nombre	Consultar lista de acontecimientos
Versión	1.0
Descripción	El sistema tiene que permitir a los usuarios consultar la lista de bandos, noticias y eventos activos.
Alcance	Mostrar sólo los eventos y noticias activas.
Nivel	Tarea principal
Actores	Usuario no registrado
Relaciones	CU02.1- Consultar lista de bandos; CU02.2- Consultar lista de noticias; CU02.3- Consultar lista de eventos
Precondición	Que haya bandos, noticias o eventos que mostrar.
Condición final con éxito	Se muestra una lista de todos los bandos, noticias y eventos activos.
Condición final con fracaso	No se muestra ningún bando, noticia o evento.
Trigger	Querer ver los bandos, noticias y eventos disponibles en mi pueblo.
Secuencia normal	
1	El usuario entra en el sistema con la sesión iniciada.
2	El sistema muestra la página principal de la plataforma, dividida en tres apartados.
3	El usuario elige uno de los tres apartados.
3	El sistema muestra la lista de bandos, noticias o eventos según el apartado escogido.
Frecuencia esperada	Diaria

Importancia	Necesario
Prioridad	Corto plazo
Comentarios	Por defecto mostrará uno de los apartados, pero el usuario puede cambiar fácilmente de apartado.

Tabla 3.2: CU02 - Consultar lista de acontecimientos

Especificación del caso de uso	
Identificador	CU03
Nombre	Consultar acontecimiento concreto
Versión	1.0
Descripción	El sistema ha de permitir al usuario consultar de forma más detallada un bando, noticia o evento seleccionado.
Alcance	Mostrar un solo acontecimiento con la información más detallada que en la lista a la que pertenece.
Nivel	Tarea principal
Actores	Usuario no registrado
Relaciones	CU03.1 - Leer bando; CU03.2 - Leer noticia; CU03.3- Consultar evento
Precondición	Tiene que existir el acontecimiento y estar visible para los usuarios.
Condición final con éxito	Se muestra información adicional del acontecimiento seleccionado.
Condición final con fracaso	No se permite seleccionar un acontecimiento para saber más sobre él.
Trigger	Un usuario quiere leer un bando, una noticia o información adicional de un evento en su pueblo.
Secuencia normal	Acción Leer bando
1	El usuario escoge el apartado “Bandos” .
2	El sistema muestra la lista de bandos del pueblo.
3	El usuario selecciona el bando que quiere leer.
4	El sistema muestra una pantalla con toda la información del bando.
Secuencia normal	Acción Leer noticia
1	El usuario escoge el apartado “Noticias” .
2	El sistema muestra la lista de noticias relacionadas con el pueblo.
3	El usuario selecciona la noticia que quiere leer.
4	El sistema muestra una pantalla con la noticia completa.
Secuencia normal	Acción Consultar evento
1	El usuario escoge el apartado “Eventos” .
2	El sistema muestra la lista de eventos activos del pueblo.
3	El usuario selecciona el evento del cual quiere informarse más.
4	El sistema muestra una pantalla con toda la información del evento.
Frecuencia esperada	Diaria

Importancia	Necesario
Prioridad	Corto plazo
Comentarios	

Tabla 3.3: CU03 - Consultar acontecimiento concreto.

Especificación del caso de uso	
Identificador	CU04
Nombre	Inscribirse a un evento
Versión	1.0
Descripción	El sistema ha de permitir a un usuario registrado poder inscribirse a un evento de la plataforma.
Alcance	Un usuario podrá inscribir a varias personas y conservar todos los tickets.
Nivel	Tarea principal
Actores	Usuario registrado
Relaciones	Ninguna
Precondición	El usuario tiene que estar registrado en la plataforma y tiene que existir el evento, además de estar visible para los usuarios.
Condición final con éxito	El usuario se inscribe en un evento y obtiene los tickets pertinentes.
Condición final con fracaso	El usuario no se inscribe en ningún evento.
Trigger	Un usuario quiere acudir a un evento disponible en la plataforma.
Secuencia normal	
1	El usuario está viendo un evento concreto y decide inscribirse.
2	El sistema muestra un formulario de inscripción.
3	El usuario rellena el formulario y confirma la acción.
4	El sistema registra la inscripción.
Frecuencia esperada	Diaria
Importancia	Necesario
Prioridad	Corto plazo
Comentarios	Ninguno.

Tabla 3.4: CU04 - Inscribirse a un evento.

Especificación del caso de uso	
Identificador	CU05
Nombre	Crear acontecimiento
Versión	1.0
Descripción	El sistema ha de permitir a un administrador crear un acontecimiento.
Alcance	El administrador tiene que tener permisos en el pueblo en el cual se crea el acontecimiento.
Nivel	Tarea principal
Actores	Administrador de comunidad; Firebase
Relaciones	CU05.1 - Crear bando; CU05.2 - Crear noticia; CU05.3- Crear evento
Precondición	El administrador tiene permisos en el pueblo activo.
Condición final con éxito	Se crea el acontecimiento.
Condición final con fracaso	No se crea ningún acontecimiento.
Trigger	Un administrador de comunidad debe informar sobre un bando, una noticia o publicar un nuevo evento en el pueblo.
Secuencia normal	Acción Crear Bando
1	El administrador se encuentra en la lista de bandos y selecciona la opción de “Añadir nuevo bando” .
2	El sistema muestra un formulario.
3	El administrador rellena el formulario y confirma la acción.
4	Firebase registra el nuevo bando y notifica a los usuarios suscritos al pueblo.
Secuencia normal	Acción Crear Noticia
1	El administrador se encuentra en la pantalla de “gestionar noticias” y selecciona la opción de “Crear noticia” .
2	El sistema le muestra un formulario.
3	El administrador rellena el formulario y confirma la acción.
4	El sistema registra la nueva noticia.
Secuencia normal	Acción Crear Evento
1	El administrador se encuentra en la pantalla de “gestionar eventos” y selecciona la opción de “Crear evento” .
2	El sistema le muestra un formulario.
3	El administrador rellena el formulario y confirma la acción.
4	El sistema registra el nuevo evento.
Frecuencia esperada	Diaria
Importancia	Tarea principal
Prioridad	Corto plazo
Comentarios	Para obtener los permisos de administrador de comunidad se deberá acudir a la web.

Tabla 3.5: CU05 - Crear acontecimiento.

Especificación del caso de uso	
Identificador	CU06
Nombre	Activar/Desactivar evento
Versión	1.0
Descripción	El sistema ha de permitir a un administrador poder ocultar o visibilizar un evento de la plataforma.
Alcance	El administrador debe tener permisos sobre el evento elegido.
Nivel	Tarea principal
Actores	Administrador de comunidad
Relaciones	Ninguna
Precondición	El administrador debe ser el que ha creado el evento o tener permisos especiales para gestionarlo.
Condición final con éxito	El evento se oculta/visibiliza en la lista de eventos.
Condición final con fracaso	No cambia el estado de visibilidad del evento.
Trigger	Un administrador quiere que un evento deje de/pase a estar visible para los usuarios.
Secuencia normal	
1	El administrador se encuentra en la pantalla de "gestionar eventos" y selecciona la opción de Ocultar/Visibilizar evento en uno concreto.
2	El sistema procesa la petición y se oculta/visbiliza el evento en la lista de eventos.
Frecuencia esperada	Diaria
Importancia	Necesario
Prioridad	Corto plazo
Comentarios	Ocultar el evento o visibilizarlo dependerá de su estado anterior.

Tabla 3.6: CU06 - Activar/desactivar evento.

Especificación del caso de uso	
Identificador	CU07
Nombre	Activar/Desactivar noticia
Versión	1.0
Descripción	El sistema ha de permitir a un administrador poder ocultar o visibilizar una noticia de la plataforma.
Alcance	El administrador debe tener permisos sobre la noticia elegida.
Nivel	Tarea principal
Actores	Administrador de comunidad
Relaciones	Ninguna
Precondición	El administrador debe ser el que ha creado la noticia o tener permisos especiales para gestionarla.
Condición final con éxito	La noticia se oculta/visibiliza en la lista de noticias.

Condición final con fracaso	No cambia el estado de visibilidad de la noticia.
Trigger	Un administrador quiere que una noticia deje de/pase a estar visible para los usuarios.
Secuencia normal	
1	El administrador se encuentra en la pantalla de "gestionar noticias" y selecciona la opción de Ocultar/Visibilizar noticia en una en particular.
2	El sistema procesa la petición y se oculta/visbiliza la noticia en la lista de noticias.
Frecuencia esperada	Diaria
Importancia	Necesario
Prioridad	Corto plazo
Comentarios	Ocultar la noticia o visibilizarla dependerá de su estado anterior.

Tabla 3.7: CU07 - Activar/desactivar noticia.

3.1.2. Requisitos funcionales de la aplicación

A partir de los casos de uso se obtienen los requisitos funcionales de la aplicación. A continuación, se clasifican los requisitos funcionales del proyecto divididos por módulos.

1) Módulo 'Bandos'
    a) Crear un bando.
    b) Mostrar la lista de bandos.
    c) Mostrar un bando concreto.
2) Módulo 'Noticias'
    a) Crear una noticia.
    b) Editar una noticia.
    c) Mostrar la lista de noticias.
    d) Mostrar una noticia concreta.
    e) Ocultar una noticia visible.
    f) Visibilizar una noticia oculta.
    g) Eliminar una noticia.
3) Módulo 'Eventos'
    a) Crear un evento.
    b) Editar un evento.
    c) Mostrar la lista de evento.
    d) Mostrar un evento concreto.

e) Ocultar un evento visible.

f) Visibilizar una evento oculto.

g) Eliminar un evento.

h) Inscribirse a un evento.

i) Consultar 'Mis Inscripciones'.

j) Confirmar asistencia de un participante.

4) Módulo 'Registro'

a) Iniciar sesión.

b) Registrar un nuevo usuario.

c) Cerrar sesión.

d) Cambiar de pueblo activo.

e) Activar/Desactivar notificaciones sobre un pueblo.

Además de los casos de uso especificados anteriormente, en los requisitos funcionales se añade la posibilidad de eliminar noticias y eventos, todo lo relacionado con la gestión de las inscripciones a los eventos y dos funcionalidades genéricas para hacer más cómodo el uso de la aplicación: cambiar el pueblo activo y activar o desactivar las notificaciones sobre un pueblo.

3.1.3. Diagrama de clases y requisitos de datos

Para profundizar en los datos que se necesitarán en la aplicación se realiza un diagrama de clases en el cual se muestran todas las entidades que se usarán para poder llevar a cabo los casos de uso explicados anteriormente. En la figura 3.2 se muestra el diagrama de clases del proyecto y en las tablas a continuación (tablas 3.8 a 3.14) se describen los requisitos de datos en cada una de las entidades del diagrama por separado.

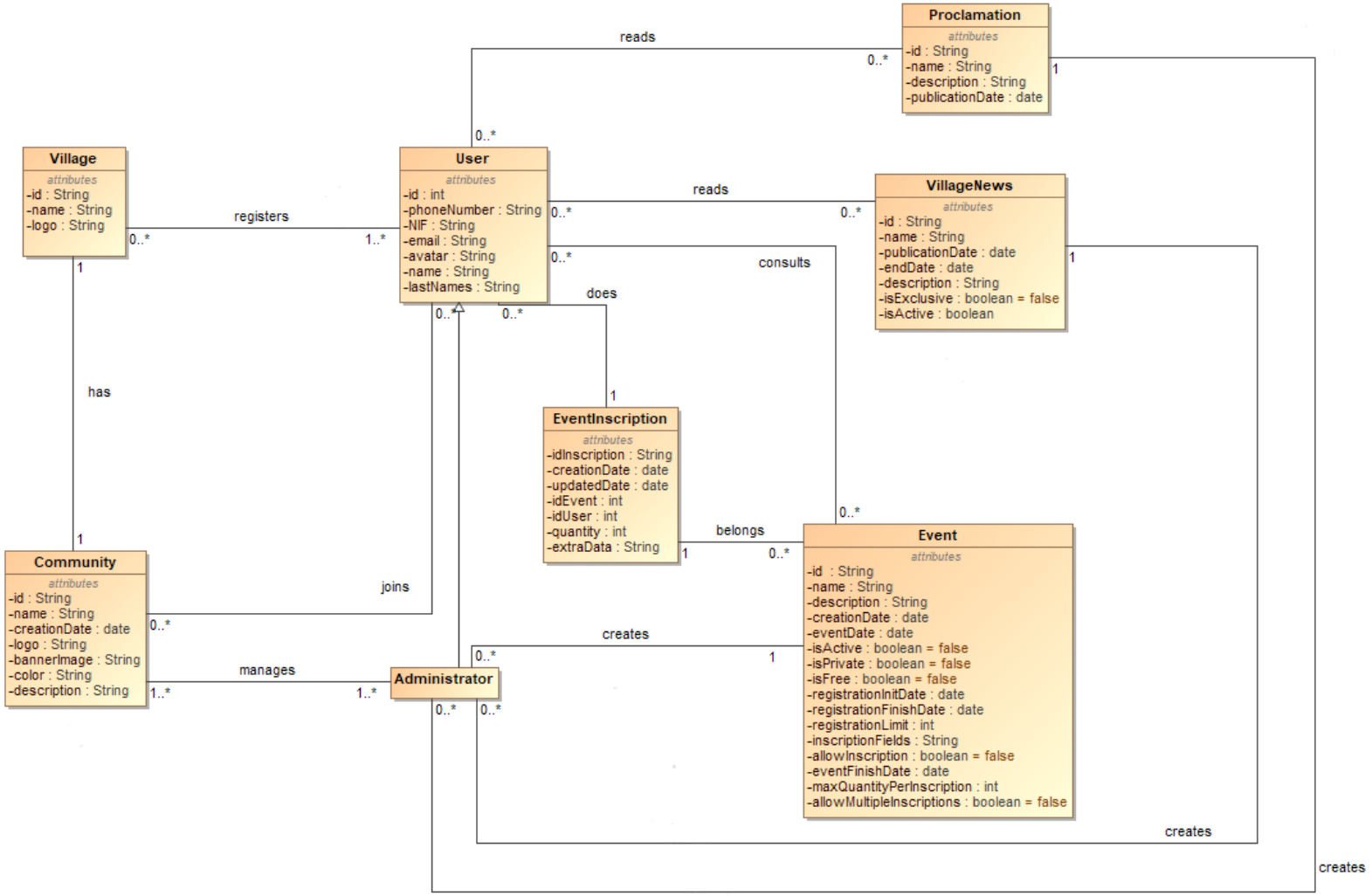


Figura 3.2: Diagrama de clases del proyecto.

Código	RD01
Nombre	Usuario
Atributos	Nombre de usuario, nombre, correo electrónico, contraseña cifrada, teléfono y día de nacimiento.
Comentarios	Además de los atributos mencionados arriba, también se guarda de los usuarios la lista de comunidades que pueden gestionar. La mayoría de usuarios tienen esta lista vacía.

Tabla 3.8: Requisitos de datos (Usuario).

Código	RD02
Nombre	Pueblo
Atributos	Id, nombre y logo
Comentarios	Ninguno.

Tabla 3.9: Requisitos de datos (Pueblo).

Código	RD03
Nombre	Comunidad
Atributos	Id, nombre, fecha de creación, logo, imagen, color y descripción.
Comentarios	Además de los atributos mencionados arriba, las comunidades tienen otros atributos para facilitar operaciones con ellas.

Tabla 3.10: Requisitos de datos (Comunidad).

Código	RD04
Nombre	Bando
Atributos	Id, nombre, descripción y fecha de publicación.
Comentarios	Ninguno.

Tabla 3.11: Requisitos de datos (Bando).

Código	RD05
Nombre	Noticia
Atributos	Id, nombre, fecha de publicación, fecha de fin, descripción, es exclusiva y está activa.
Comentarios	El atributo exclusiva hace referencia a si la noticia es exclusiva de una sola comunidad y activa equivale a oculta o visible.

Tabla 3.12: Requisitos de datos (Noticia).

Código	RD06
Nombre	Evento
Atributos	Id, nombre, descripción, fecha de creación, fecha de evento, está activo, fecha inicial y final para inscribirse, número de plazas y precio.
Comentarios	En el diagrama aparecen atributos adicionales que facilitan a los creadores de eventos hacer formularios a medida para los asistentes al evento.

Tabla 3.13: Requisitos de datos (Evento).

Código	RD07
Nombre	Inscripción
Atributos	Id, fecha de creación, id del evento, id del usuario y cantidad de participantes.
Comentarios	La información extra hace referencia a lo contestado en los formularios personalizados de cada evento.

Tabla 3.14: Requisitos de datos (Inscripción).

3.2. Diseño de la arquitectura del sistema

Aunque en un principio estaba planificado hacer el diseño de una nueva base de datos, la empresa decidió cambiar la plataforma en la que se encontraba el *backend*, creando ellos mismos la nueva base de datos. Para ello utilizaron un diseño muy similar al que ya tenían para la aplicación web del municipio de Villanueva de Viver y no era necesario hacer diseños nuevos.

No obstante, para hacerse una idea sobre cómo es el diseño de la base de datos, basta con mirar el diagrama de clases del apartado anterior (figura 3.2), ya que es muy similar pero con atributos adicionales en cada tabla que sirven para facilitar algunas operaciones en el *backend*.

3.3. Diseño de la interfaz

Al ser un proyecto sobre una aplicación móvil para el público general, las interfaces de usuario son muy importantes. Por ello, las interfaces deben ser intuitivas y la navegación en la aplicación sencilla. Teniendo en mente esto, se han diseñado mediante la plataforma *Figma* todas las interfaces por módulos, para que ninguna se pasase por alto. Además, estos prototipos están hechos de manera tan detallada que la diferencia entre ellos y el resultado final es mínima. A continuación se explican los distintos módulos y se muestran las diferentes interfaces, validadas e ideadas por el diseñador de la empresa.

3.3.1. Módulo Bandos

Este módulo engloba los prototipos relacionados con los bandos. En la figura 3.3 podemos observar cómo sería la pantalla en la cual se listan los bandos publicados en un pueblo. Por otra parte, la figura 3.4 muestra cómo se vería un bando concreto al pulsar en él en la lista de bandos de la figura anterior.

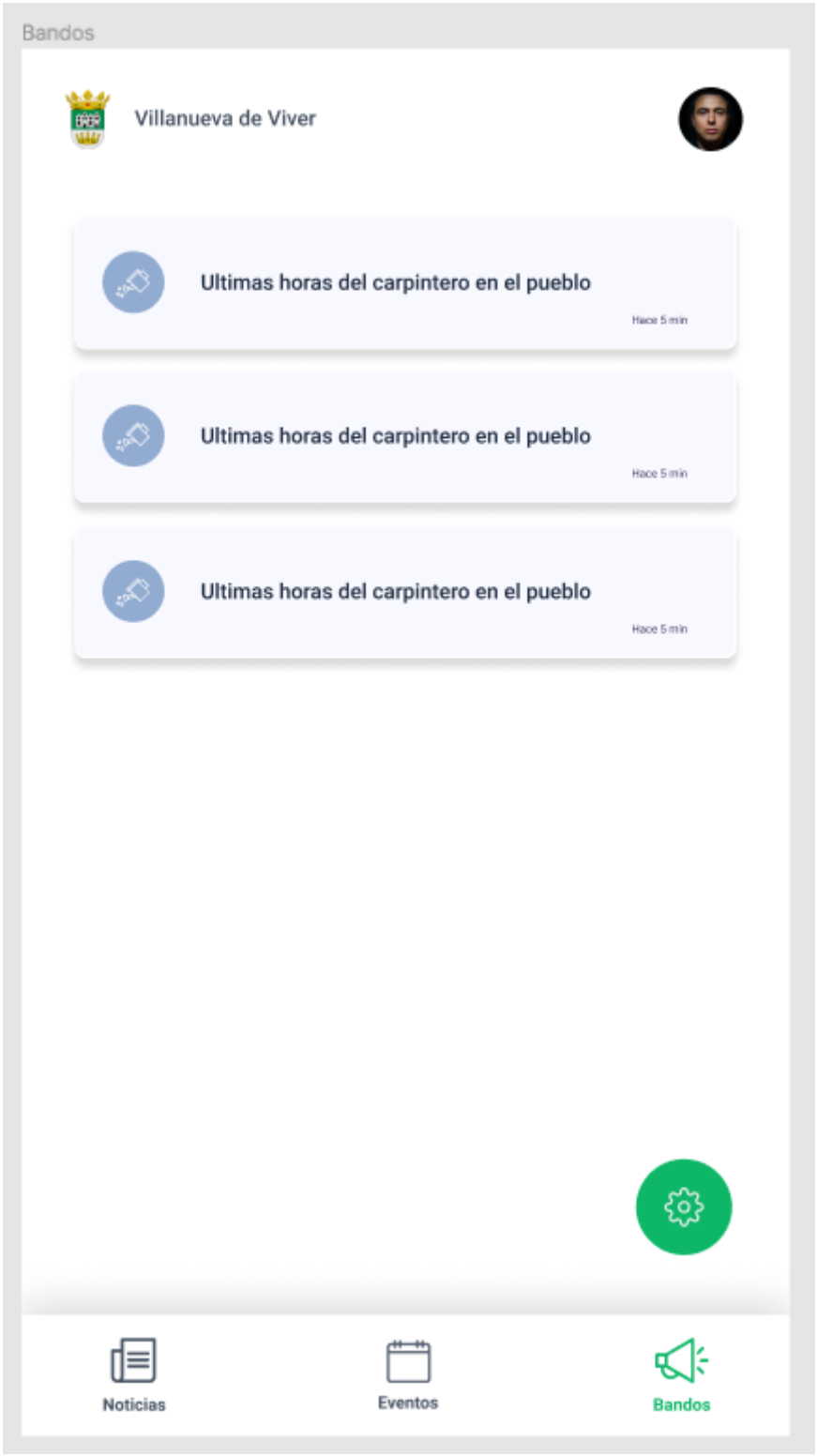


Figura 3.3: Lista de bandos.

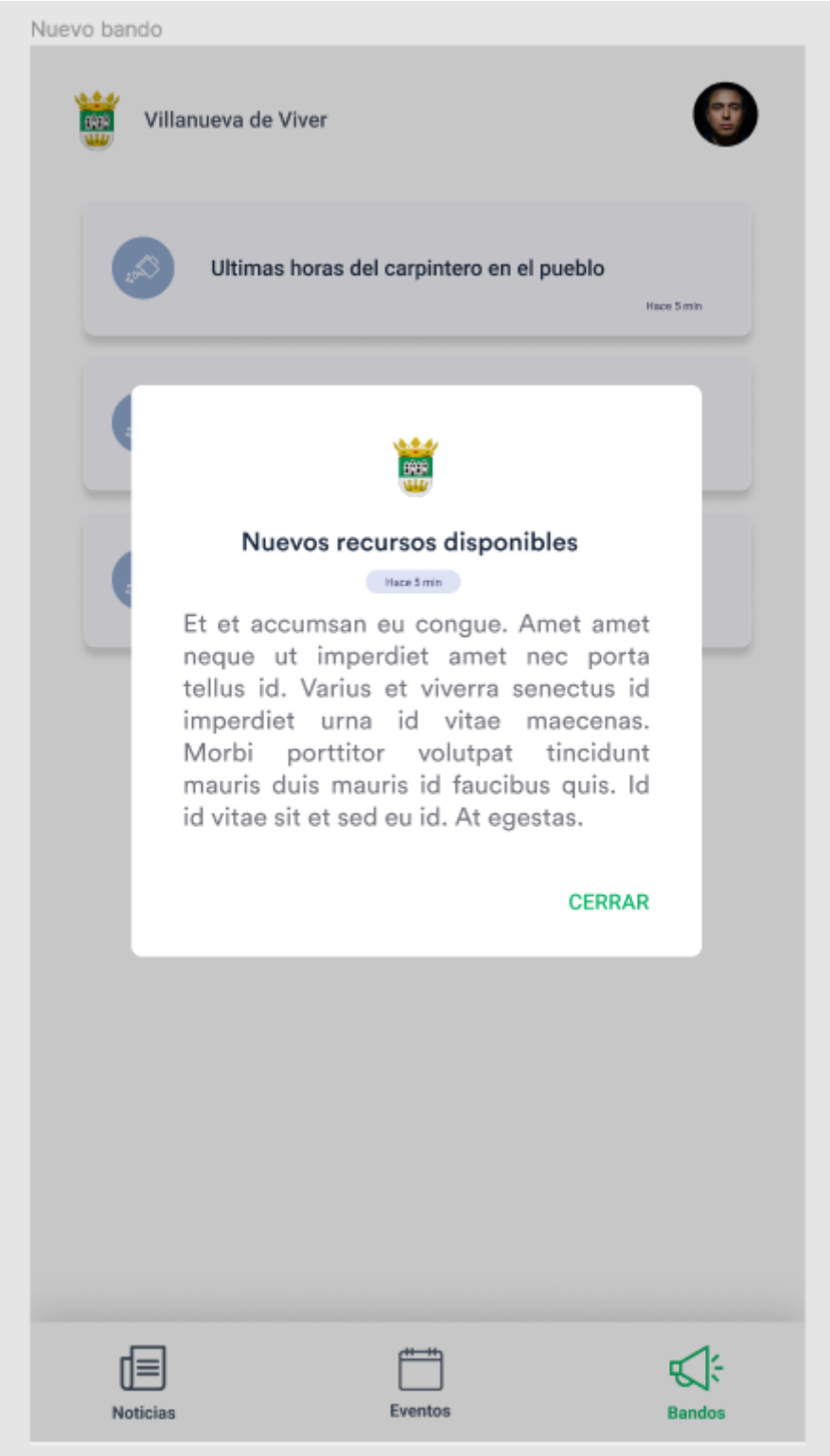


Figura 3.4: Bando específico.

3.3.2. Módulo Noticias

En este módulo se incluyen todos los prototipos sobre la parte de noticias. En las figuras 3.5 y 3.6 se muestra la lista de noticias y una noticia ampliada, respectivamente. Estas interfaces corresponden tanto a la vista de usuarios como de administradores. No obstante, las dos figuras siguientes, 3.7 y 3.8, son exclusivas de los administradores de comunidad o de pueblo.

Figura 3.5: Lista de noticias.

Figura 3.6: Noticia específica.

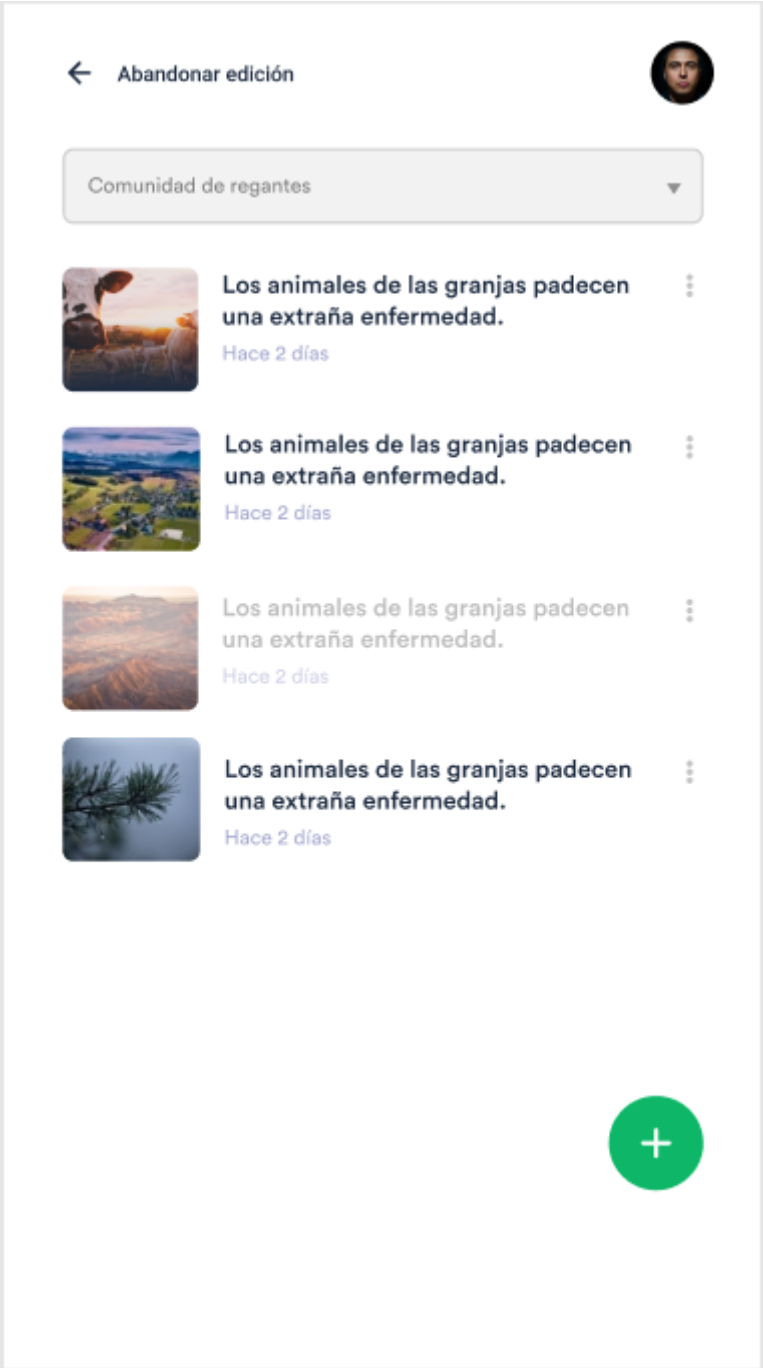


Figura 3.7: Lista de noticias de un administrador.

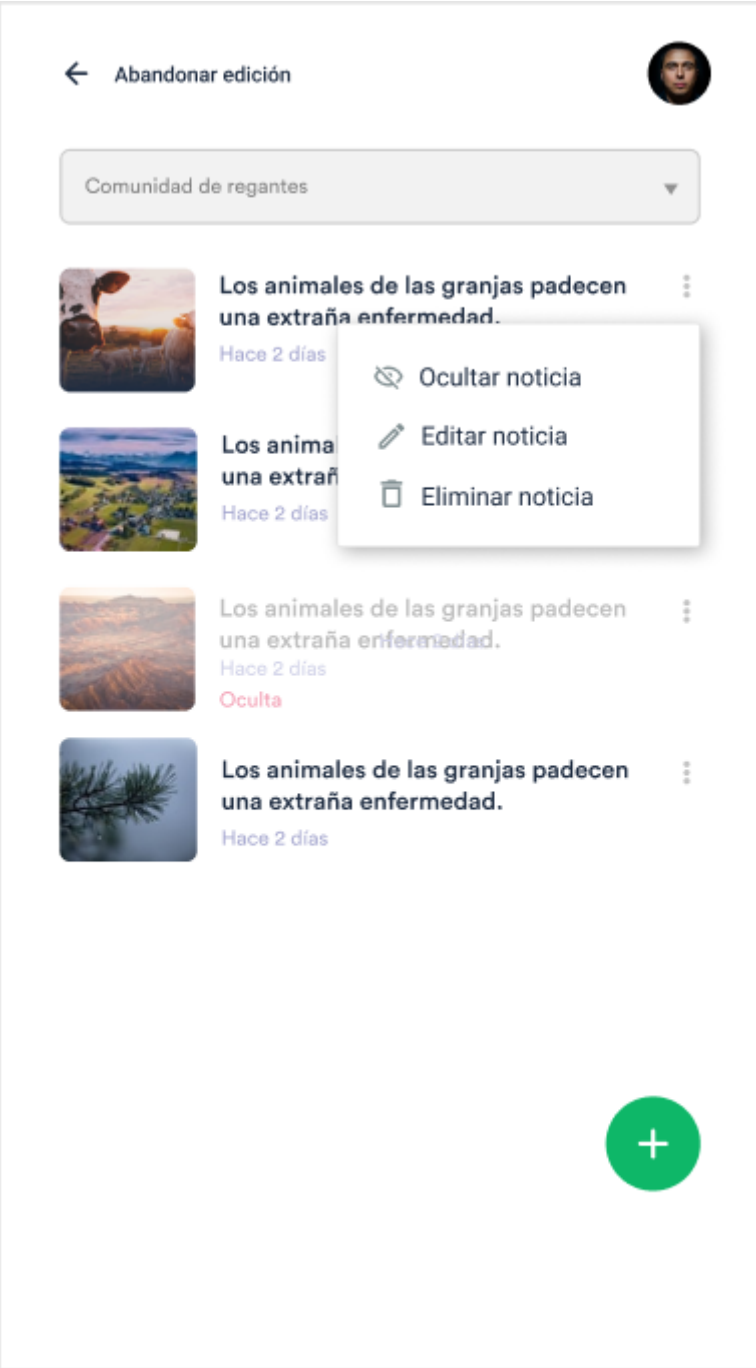


Figura 3.8: Menú de opciones de noticia (Administrador).

3.3.3. Módulo Eventos

Incluye las interfaces sobre eventos, tanto la parte que pueden visualizar todos los usuarios como la parte de los administradores, que incluye también la gestión de inscripciones (figuras 3.13, 3.14 y 3.15). El resto de interfaces se corresponden al equivalente de noticias del apartado anterior pero para eventos (figuras 3.9, 3.10, 3.11 y 3.12).

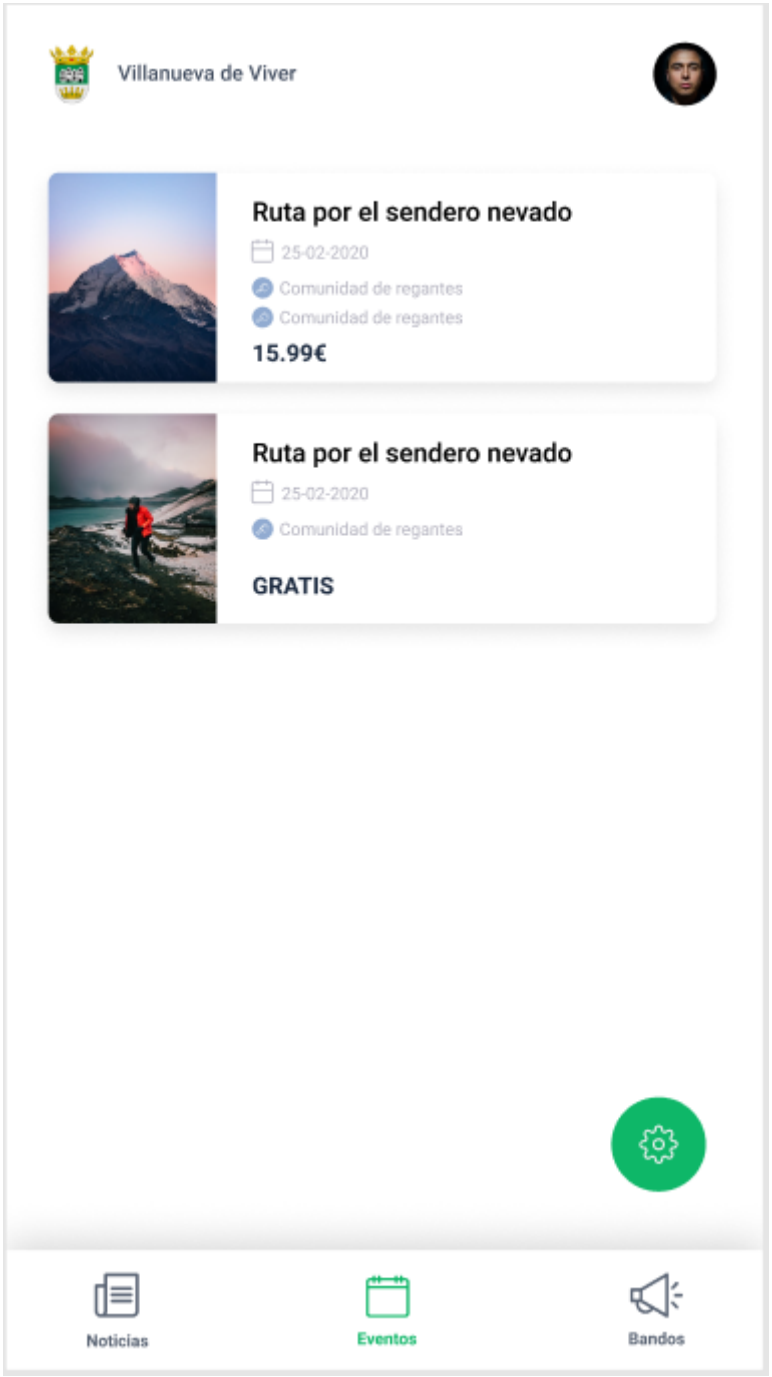


Figura 3.9: Lista de eventos.

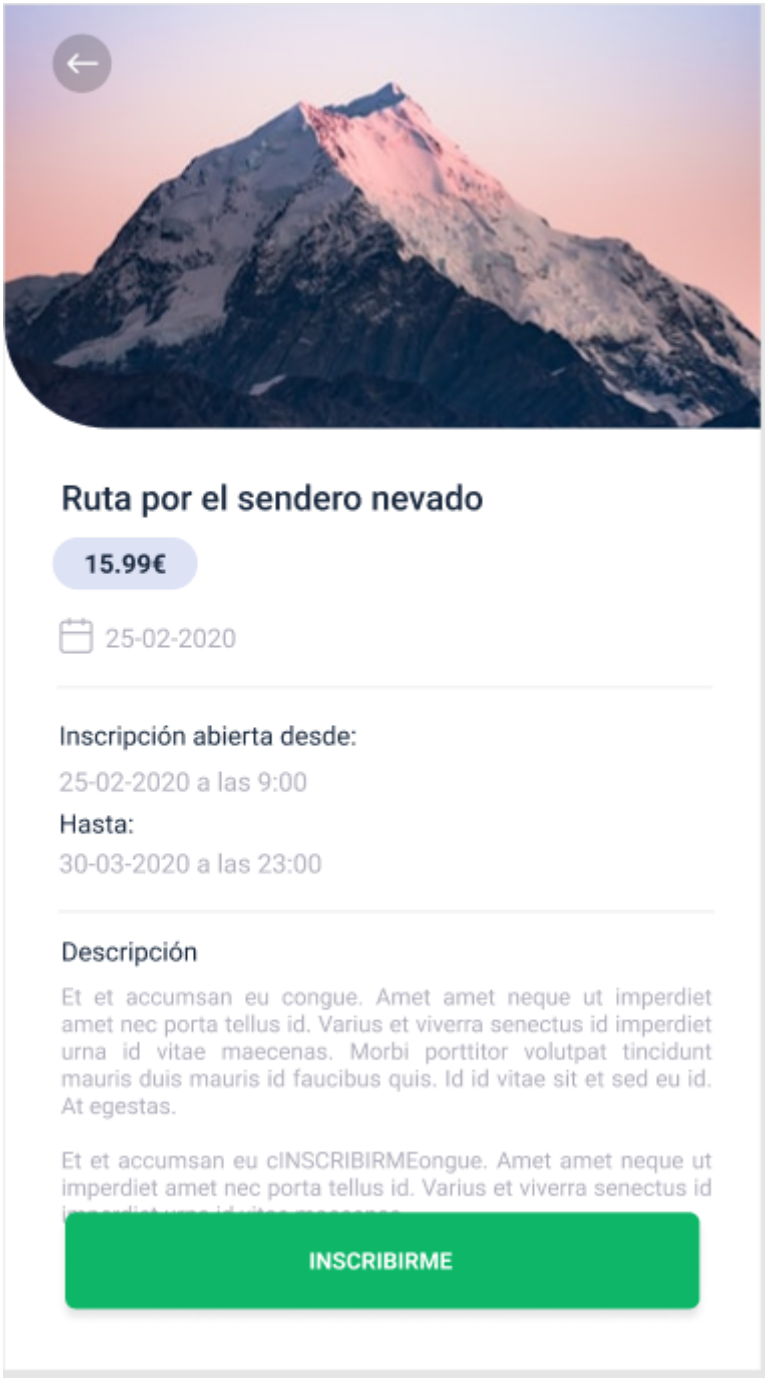


Figura 3.10: Evento específico.

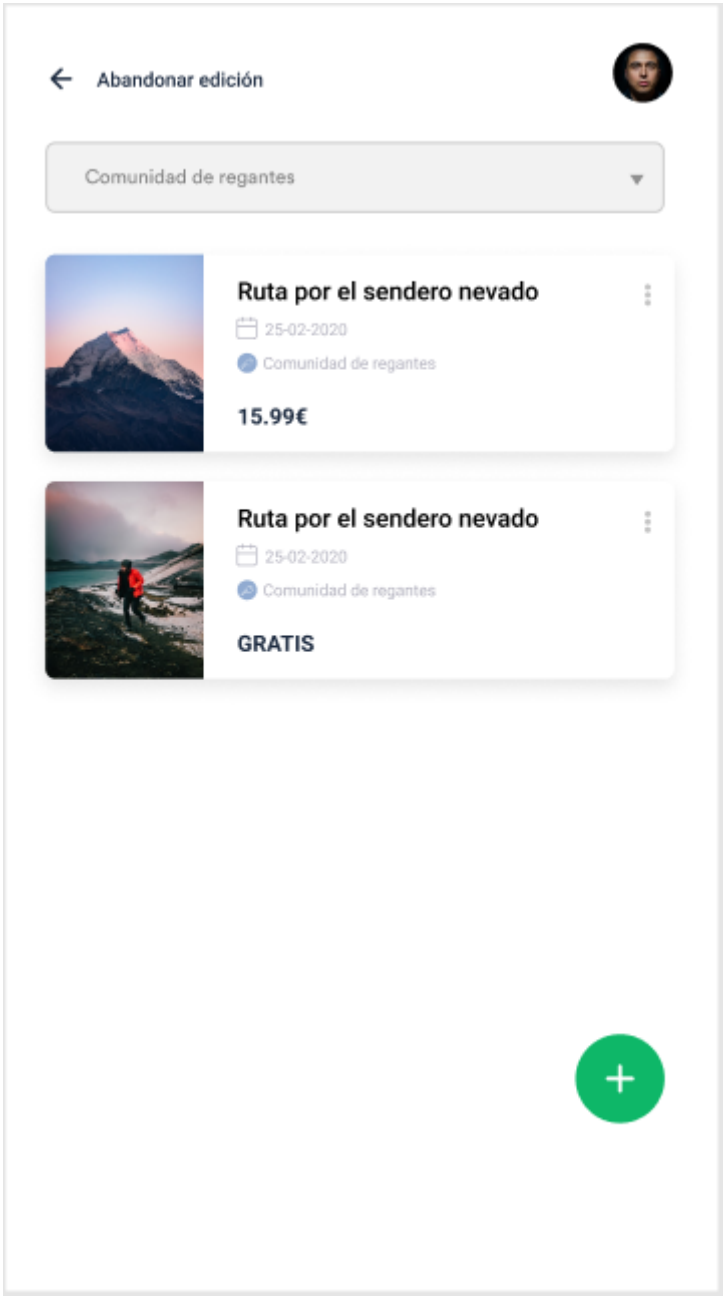


Figura 3.11: Lista de eventos de un administrador.

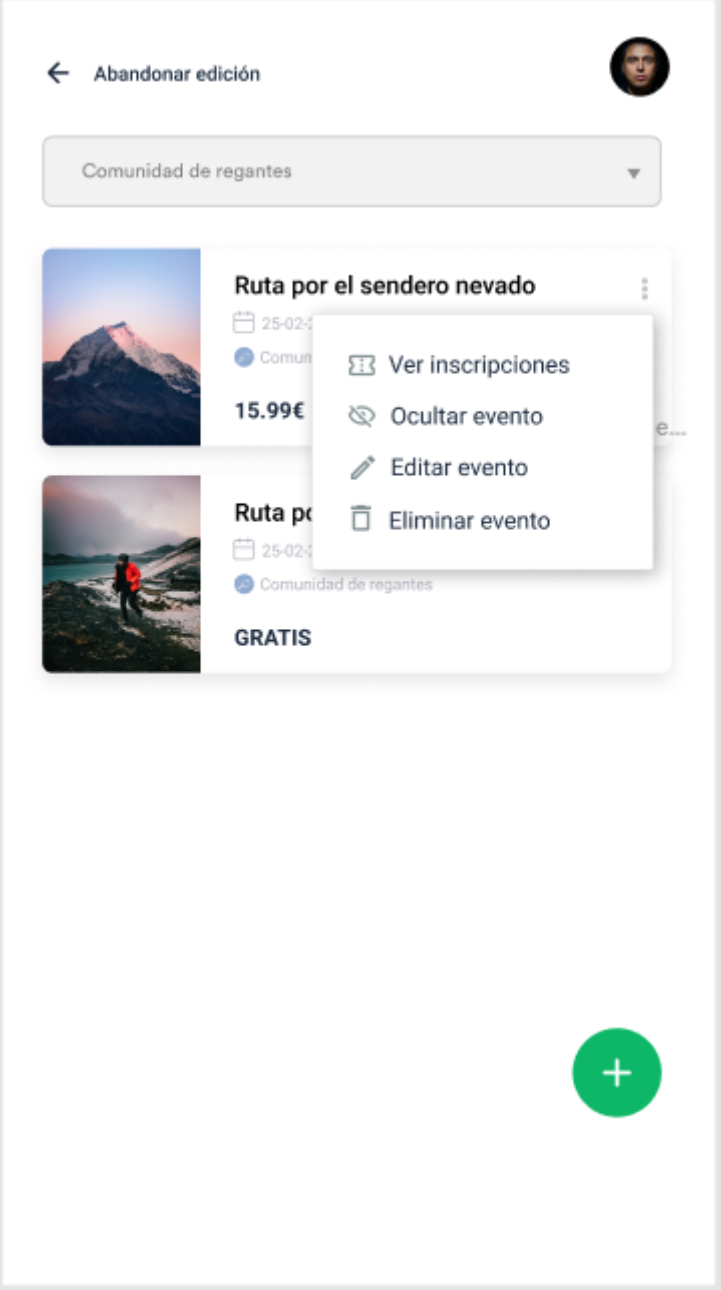


Figura 3.12: Menú de opciones de evento (Administrador).

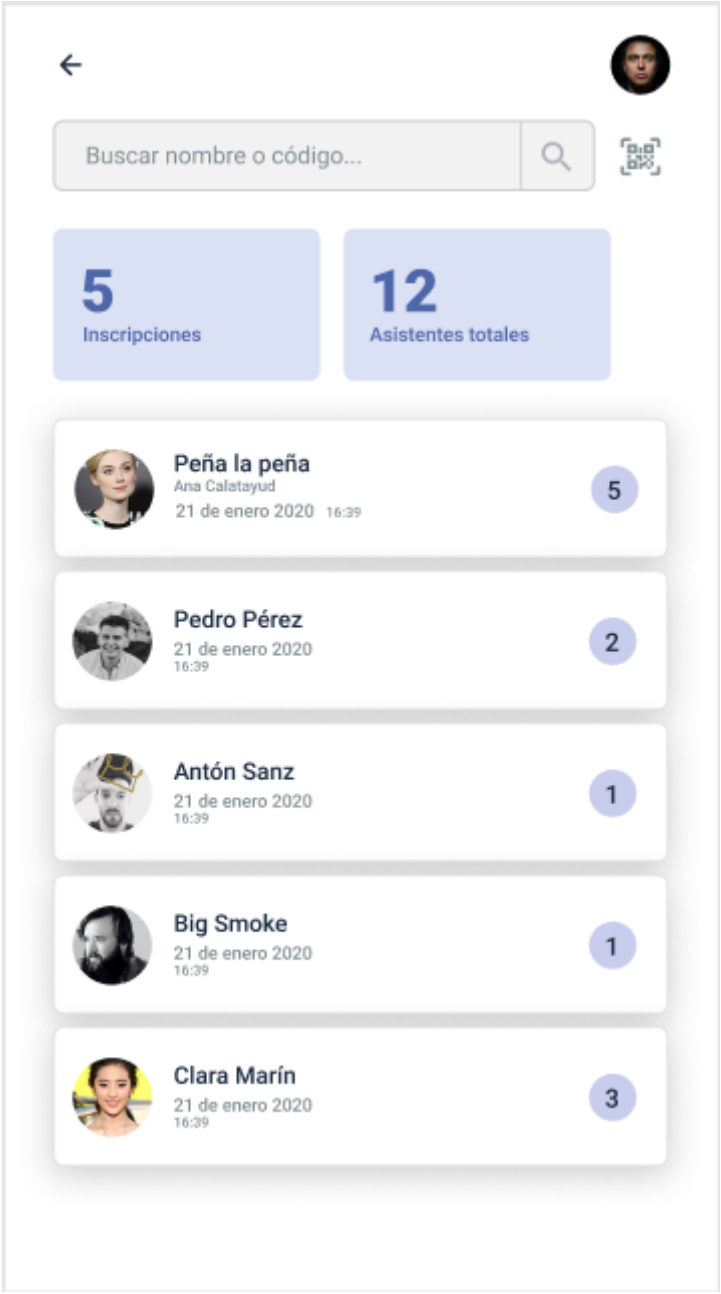


Figura 3.13: Lista de inscripciones a un evento.

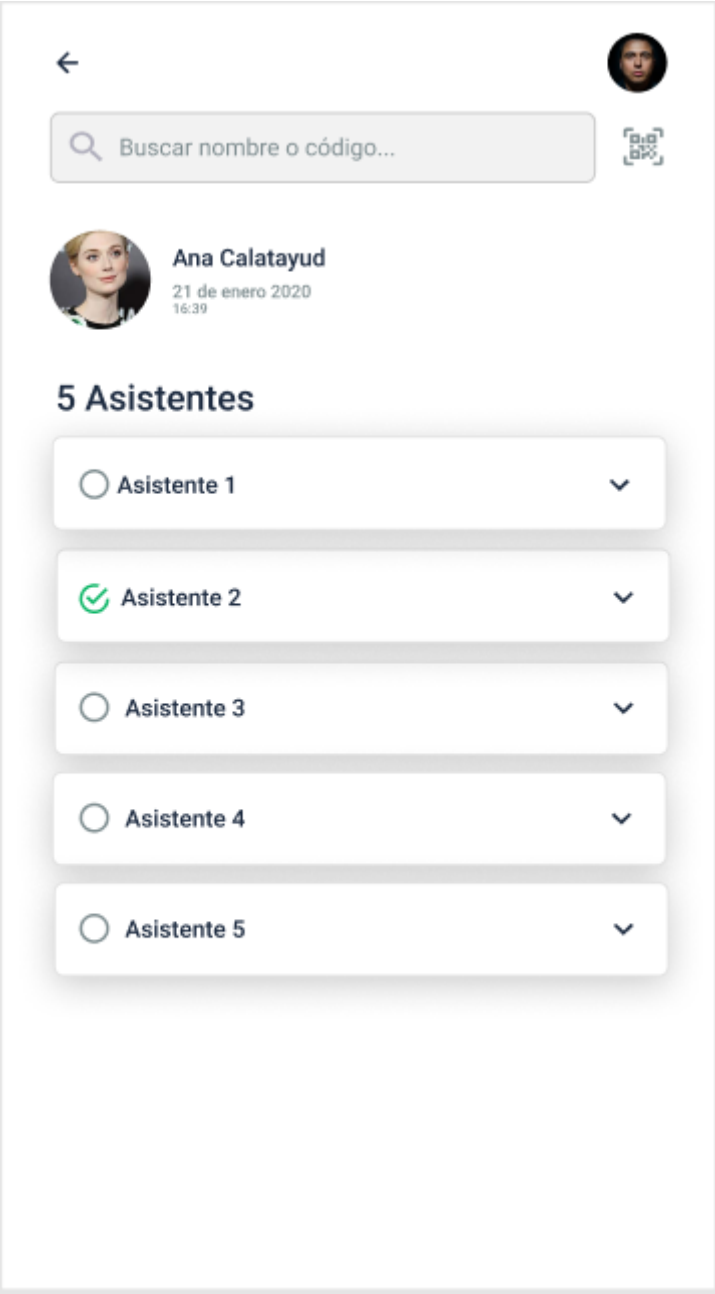


Figura 3.14: Inscripción específica.

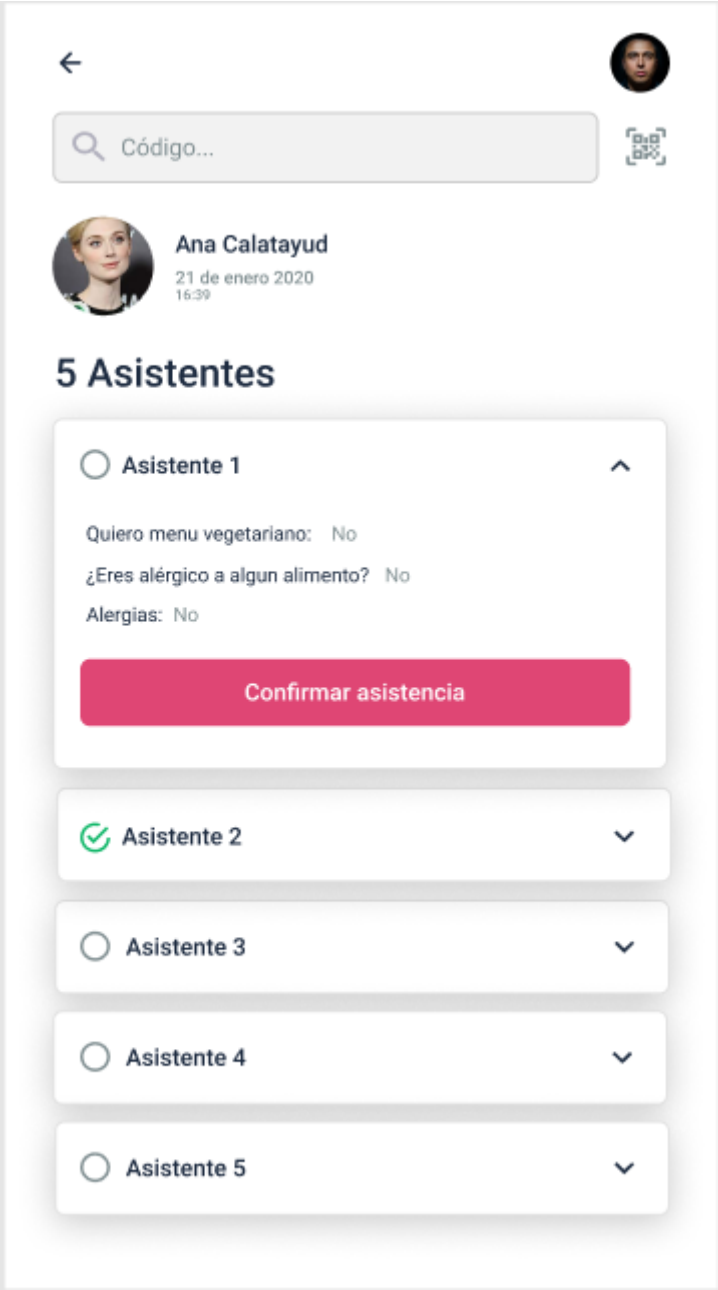


Figura 3.15: Inscripción específica (Asistente desplegado).

3.3.4. Módulo Registro

Para el inicio de sesión y el registro de usuarios se han diseñado dos interfaces, figuras 3.16 y 3.17 respectivamente. No obstante, la aplicación permite acceder a un pueblo sin tener cuenta. Por ello, la primera interfaz que aparece a un usuario que acaba de instalar la aplicación es la de acceder sin cuenta (figura 3.18).

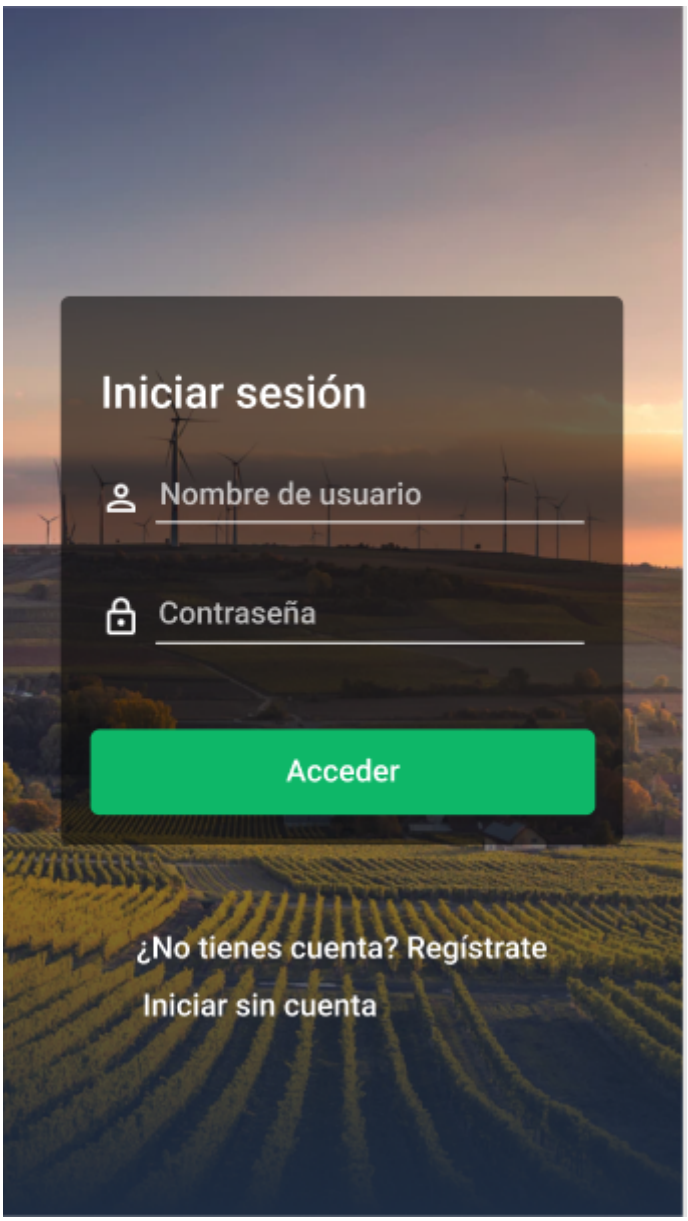


Figura 3.16: Inicio de sesión.

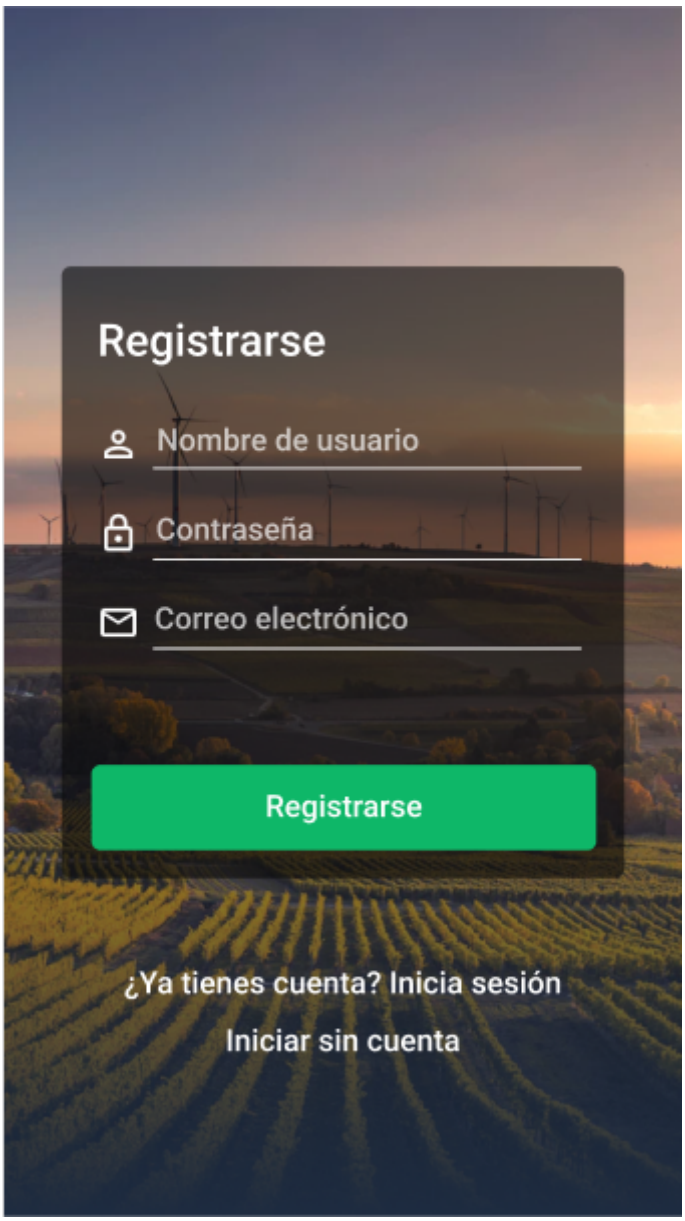


Figura 3.17: Registrarse en la aplicación.

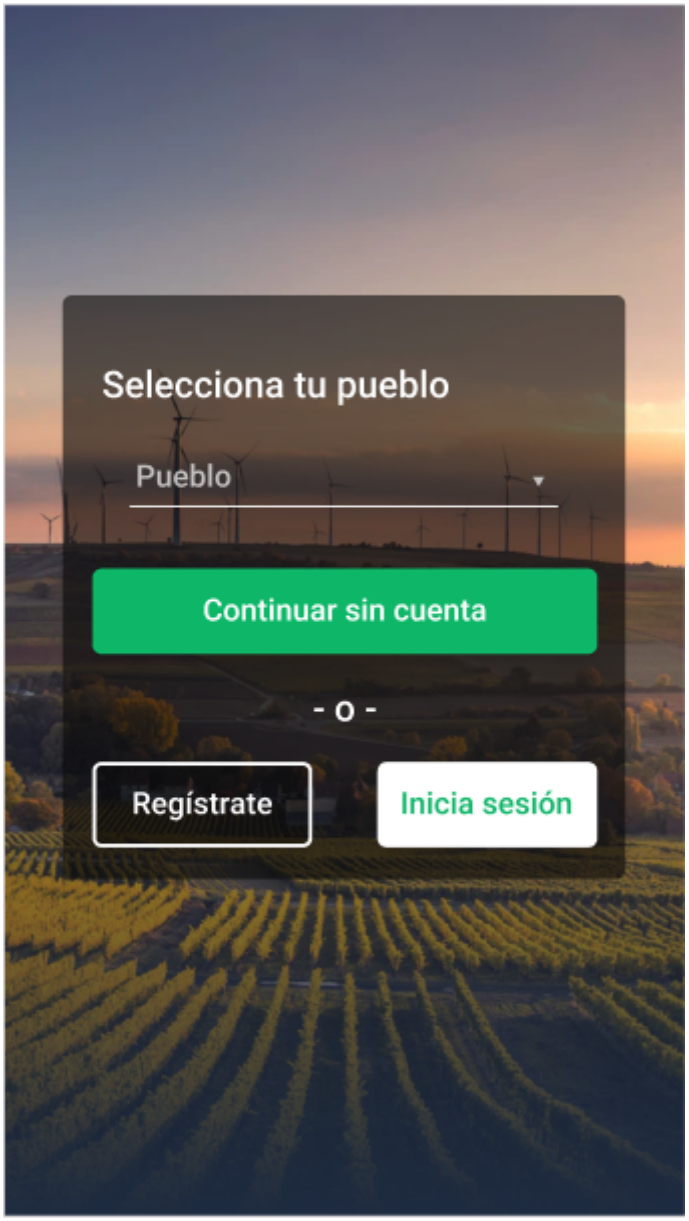


Figura 3.18: Acceder sin cuenta.

Capítulo 4

Implementación y pruebas

4.1. Detalles de implementación

El primer paso para llevar adelante el proyecto era familiarizarse con *Flutter*, el *SDK* creado por *Google* para desarrollar tanto aplicaciones web como móviles de forma sencilla. El código de una aplicación hecha en Flutter se escribe en el lenguaje de programación *Dart*, que comparte similitudes con *Java* y *Javascript*. Sin embargo, *Flutter* tiene la peculiaridad de que "todo es un *widget*"[2], haciendo que la forma de programar sea extraordinariamente sencilla, incluyendo *widgets* dentro de *widgets*, siguiendo una lógica. Un ejemplo del árbol de *widgets* de una interfaz hecha en *Flutter* sería el de la figura 4.2, donde podemos observar como se dividiría en estos componentes la figura 4.1. [3]

Figura 4.1: Ejemplo de interfaz en *Flutter*.

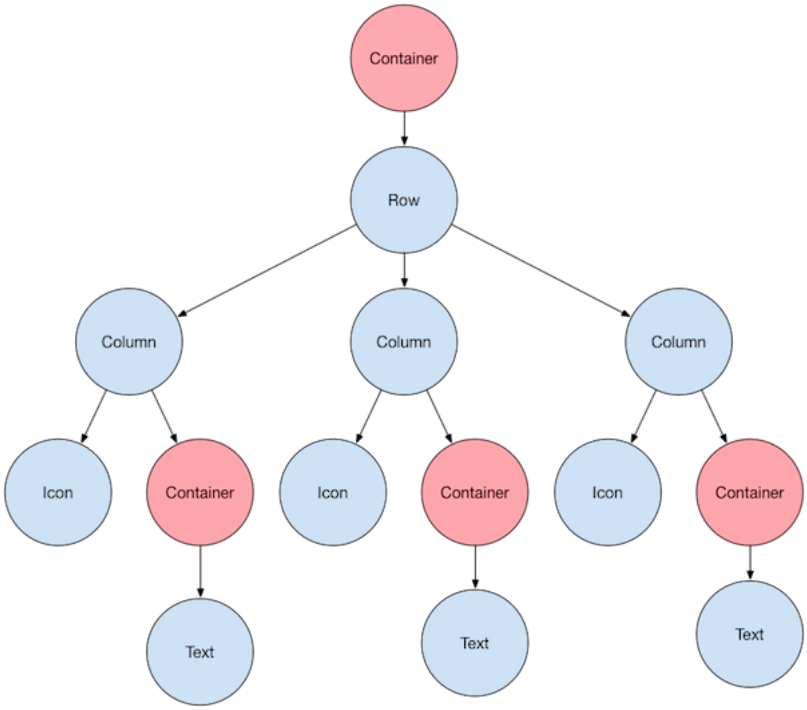


Figura 4.2: Árbol de *widgets* de la figura 4.1.

Una vez introducido esto, hay que distinguir entre dos tipos de *widgets*: *Stateful* (Con estado) y *Stateless* (Sin estado)[4]. Si el usuario interactúa con él haciendo que cambie, se trata de un *widget* con estado, mientras que si solo muestra información y no se puede interactuar con él, ese componente es un *widget* sin estado. Además, a diferencia de la mayoría de *frameworks* utilizados en el grado, *Flutter* es declarativo[5] en vez de imperativo. Es decir, construye la interfaz a partir del estado de la aplicación, de tal forma que para cambiar una interfaz cambiamos su estado y *Flutter* la vuelve a dibujar. A continuación, se explica la estructura del proyecto y se entra con detalle en algunas clases, para las cuales se muestra el uso tanto de los *widgets* con estado como de los *widgets* sin estado.

4.1.1. Estructura del proyecto

El proyecto, desarrollado en *Visual Studio Code*, tiene como estructura general la que se muestra en la figura 4.3, en la cual se distinguen las siguientes carpetas:

- *android*: En esta carpeta se encuentra todo lo necesario para que el código funcione correctamente en dispositivos *Android*. Por ejemplo, si queremos pedir permisos para poder utilizar la cámara del dispositivo hay que modificar un fichero de esta carpeta.
- *assets*: En ella se encuentran todas las imágenes e iconos personalizados utilizados en el proyecto.
- *build*: Contiene todo lo necesario para poder construir el *APK* (*Android Application Package*) de la aplicación.

- *functions*: Incluye el código para implementar las notificaciones automatizadas en *Firebase* (no ha sido necesario implementarlo por el alumno).
- *ios*: Como la carpeta *android* pero enfocada en dispositivos de *Apple*.
- *lib*: Contiene todo el código implementado en el proyecto.
- *test*: En esta carpeta se implementan los tests del código de la aplicación.

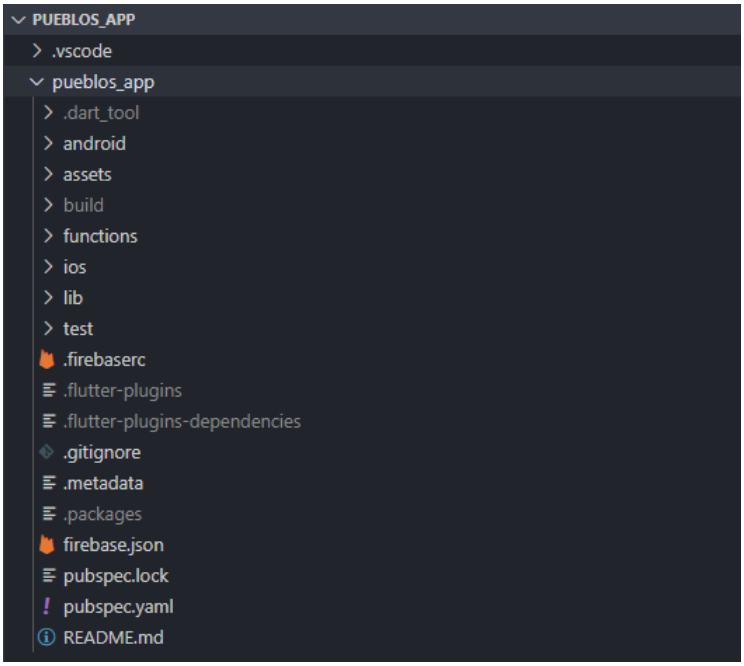


Figura 4.3: Estructura general del proyecto.

Además de estas carpetas destaca el fichero *pubspec.yaml*, en el cual se incluyen todas las dependencias que tiene el proyecto (figura 4.4). Las tres primeras, *firebase_auth*, *cloud_firestore* y *firebase_messaging*, son necesarias para utilizar *Firebase* y sus diferentes productos, como la base de datos *Cloud Firestore* y el sistema de notificaciones *Firebase Cloud Messaging*. Por otra parte, la dependencia *http* es imprescindible para el proyecto ya que nos permite realizar llamadas *HTTP*, usadas para comunicarnos con el *backend*. Las dos siguientes dependencias, *shared_preferences* y *flutter_secure_storage*, se utilizan para poder guardar datos en el dispositivo móvil. Por último, las tres últimas dependencias (*flutter_html*, *barcode_scan* y *webview_flutter*) permiten tratar textos en *HTML*, leer códigos *QR* y poder conectarse con una web respectivamente.

Figura 4.4: Dependencias del código del proyecto.

Una vez explicada la estructura de un proyecto en *Flutter*, hay que fijarse en la carpeta *lib*, que es la que contiene el código de la aplicación (figura 4.5). La carpeta contiene un total de tres carpetas en su interior(*components*, *model* y *screens*), además de clases auxiliares que se explicarán más adelante.

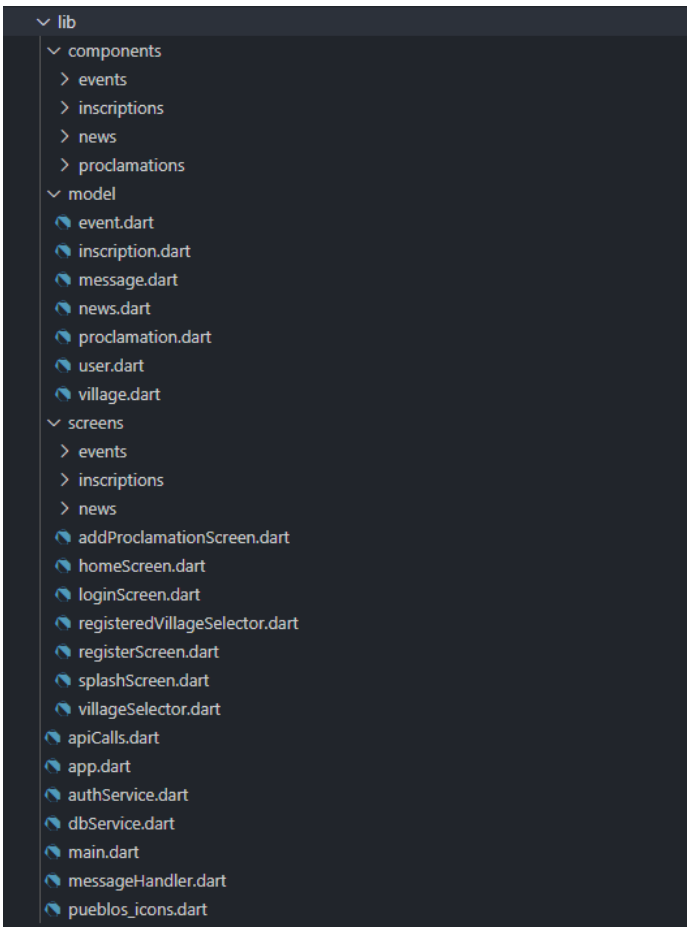


Figura 4.5: Clases del proyecto.

En la carpeta *components* se encuentran aquellos componentes que se añaden a las interfaces, como por ejemplo *cards* o *containers*. Estos componentes se clasifican según el apartado de la aplicación al que pertenecen, ya sean eventos, noticias, bandos o inscripciones. Para ilustrar el contenido de estas carpetas se utiliza de ejemplo el apartado de noticias.

Para crear la lista de noticias, similar al prototipo mostrado en el apartado del diseño de interfaces, se utiliza una clase llamada *newsContainer* (figuras 4.6 y 4.7), que se trata de un *StatefulWidget*. Su método *initState()* llama a la *API* para conseguir la lista de noticias del pueblo en el que estemos y, usando un método que se mostrará más adelante, se convierte la respuesta en una lista de objetos del tipo *News*. A partir de esta lista se contruye un *widget* de tipo *ListView* que creará tantos *NewsElement*, otra de las clases del paquete, como noticias haya.

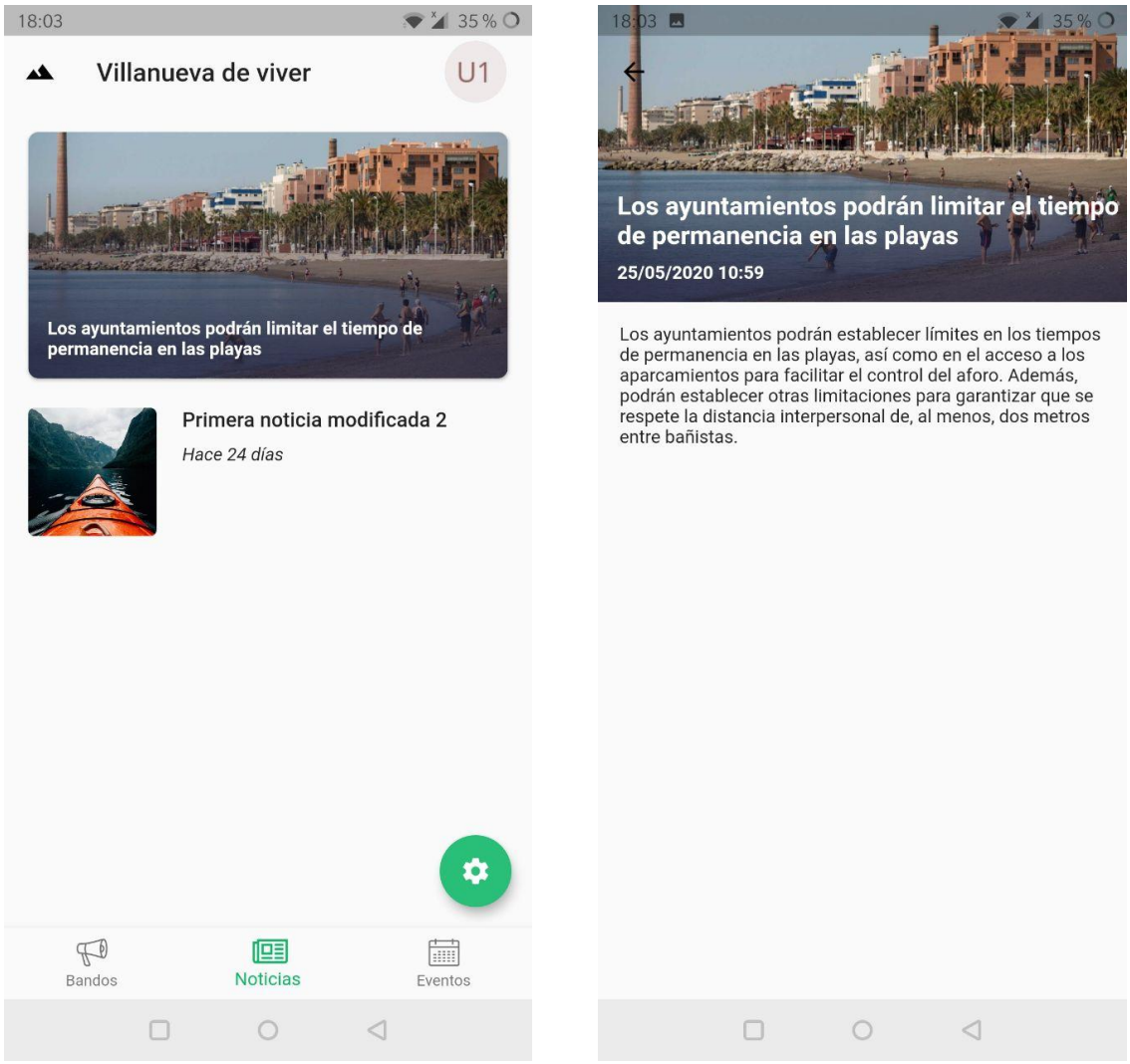
```dart
class NewsContainer extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => _NewsContainerState();
}

class _NewsContainerState extends State<NewsContainer> {
  var news = List<News>();
  bool isLoading = true;
  String _domain = "";
  String _activeVillageId = "";
  String token;

  @override
  initState() {
    super.initState();
    _getNews();
    AuthService().refreshToken();
  }
```

Figura 4.6: Clase *newsContainer*.

```
@override
Widget build(BuildContext context) {
  return isLoading
      ? Center(child: CircularProgressIndicator())
      : Container(
          padding: EdgeInsets.all(15),
          child: Column(
            children: <Widget>[
              Expanded(
                child: ListView.builder(
                    itemCount: news.length,
                    itemBuilder: (context, index) {
                      if (index == 0) {
                        return ImageNewsCard(
                            news[0].id,
                            news[0].image,
                            news[0].name,
                            news[0].description,
                            news[0].publishDate,
                            _domain); // ImageNewsCard
                      } else {
                        return NewsElement(
                            news[index].id.toString(),
                            news[index].image.toString(),
                            news[index].name,
                            news[index].description,
                            news[index].publishDate,
                            _domain); // NewsElement
                      }
                    }), // ListView.builder
              ), // Expanded
            ], // <Widget>[]
          ), // Column
        ); // Container
}
```

Figura 4.7: Clase *newsContainer* (cont.).

La clase *NewsElement* se trata también de un *StatefulWidget* que, al pulsarlo, cambiará a un nuevo *widget* llamado *DetailedNewsItem*, un *StatelessWidget* que permitirá visualizar la noticia pulsada al completo. (figura 4.8)



(a) Lista de noticias.

(b) Noticia ampliada.

Figura 4.8: Leer una noticia.

Por otra parte, en *model* están las clases que definen los objetos que se utilizan en la aplicación: *event*, *inscription*, *message*, *news*, *proclamation*, *user* y *village*. Cada una de estas clases tiene los atributos necesarios para poder mostrar la información pertinente en la aplicación, y además, un método que transforma un objeto *JSON* (*JavaScript Object Notation*) pasado como parámetro en uno de estos objetos (figura 4.9).

```
News.fromJson(Map<String, dynamic> json)
    : id = json['wid'],
      image = json['image'],
      name = json['name'],
      description = json['description'],
      active = json['active'],
      publishDate = json['publishDate'];
```

Figura 4.9: Método que convierte un objeto *JSON* a un objeto *News*.

Continuando por la estructura de la carpeta *lib* tenemos el paquete *screens*, en el cual se encuentran las diferentes pantallas que tiene la aplicación como, por ejemplo, la pantalla de inicio de sesión o la de gestionar inscripciones de un evento. Muchas de estas clases utilizan los componentes explicados anteriormente para construir la interfaz ya que, por lo general, estas pantallas simplemente contienen un componente que es un *widget* de tipo *container* en el cual se incluye todo lo demás. Un ejemplo de esto es la figura 4.10, en la que se puede observar el contenido de la clase relativa a la gestión de noticias. No obstante, el resto de pantallas son más complejas a la hora de construir la interfaz, como por ejemplo la pantalla de registrarse en la plataforma.

```dart
class ConfigNewsScreen extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => _ConfigNewsScreenState();
}

class _ConfigNewsScreenState extends State<ConfigNewsScreen> {

  @override
  void initState() {
    super.initState();
    AuthService().refreshToken();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
          backgroundColor: Colors.transparent,
          elevation: 0,
          title: Text(
            "Abandonar edición",
            //style: TextStyle(color: Colors.white),
          )), // Text // AppBar
          floatingActionButton: FloatingActionButton(
      onPressed: () {
        Navigator.push(context,
            MaterialPageRoute(builder: (context) => AddNewsScreen()));
      },
      child: Icon(Icons.add),
      backgroundColor: Color(0xFF29BF79),
    ), // FloatingActionButton
      body: Container(
        padding: EdgeInsets.all(20),
        child: EditableNewsContainer(),
      ), // Container
    ); // Scaffold
  }
}
```


Figura 4.10: Clase *ConfigNewsScreen*.

Para finalizar, existen un grupo de archivos que no pertenecen a ninguna de las categorías anteriormente descritas. Cada uno de estos ficheros realiza una función distinta pero necesaria para el correcto funcionamiento de la aplicación. Estos archivos son:

- *apiCalls*. Se trata de un fichero en el que se encuentran todas las llamadas a la *API*.
- *app.dart*. En él se encuentra toda la aplicación. En el apartado 4.1.2 se detallará su funcionamiento.
- *authService*. Engloba los métodos relativos a la autenticación de usuarios, como el registro, el inició de sesión o el cierre de sesión.
- *dbService*. Este archivo contiene lo conexión con *Firebase* y los métodos a la base de datos *Cloud Firestore*.
- *main.dart*. Este fichero se ejecuta en primer lugar. En el apartado 4.1.2 se detallará su funcionamiento.
- *messageHandler*. Sirve para notificar a los usuarios mediante *Firebase Cloud Messaging*.
- *pueblos_icons*. En él se definen los iconos personalizados importados en el proyecto.

4.1.2. Decisiones de desarrollo

Mostrada la estructura de ficheros del proyecto y otros detalles sobre *Flutter*, como los tipos de *widget*, queda por último explicar diferentes decisiones tomadas a lo largo del desarrollo, así como detalles que no se tuvieron en cuenta en la fase de análisis ni en la de diseño.

En primer lugar, convendría explicar qué sucede cuando iniciamos la aplicación. El primer fichero en ejecutarse es el *main.dart*, el cual tiene la función de definir características de la aplicación y ejecutar después el archivo *app.dart*. Como vemos en la figura 4.11, esta clase presenta tres detalles a tener en cuenta: es un *widget* sin estado (1), define cual es la pantalla principal (2) y también las rutas de la aplicación (3).

```dart
class App extends StatelessWidget {
  AuthService appAuth = AuthService();

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        FocusScopeNode currentFocus = FocusScope.of(context);

        if (!currentFocus.hasPrimaryFocus) {
          currentFocus.unfocus();
        }

        if (!currentFocus.hasPrimaryFocus) {
          currentFocus.unfocus();
        }
      },
      child: MaterialApp(
        theme: ThemeData(
          primaryColor: Color(0xFF0EB768),
          accentColor: Color(0xCC272741),
        ), // ThemeData
        home: SplashScreen(),
        routes: <String, WidgetBuilder>{
          '/LoginScreen': (BuildContext context) => LoginScreen(),
          '/HomeScreen': (BuildContext context) => HomeScreen(),
          '/VillageSelector': (BuildContext context) => VillageSelector(),
          '/ConfigNews': (BuildContext context) => ConfigNewsScreen(),
          '/ConfigEvents': (BuildContext context) => ConfigEventsScreen(),
          '/RegisteredVillageSelector': (BuildContext context) => RegisteredVillageSelector()
        },
      ), // MaterialApp
    ); // GestureDetector
  }
}
```

(1) (2) (3)

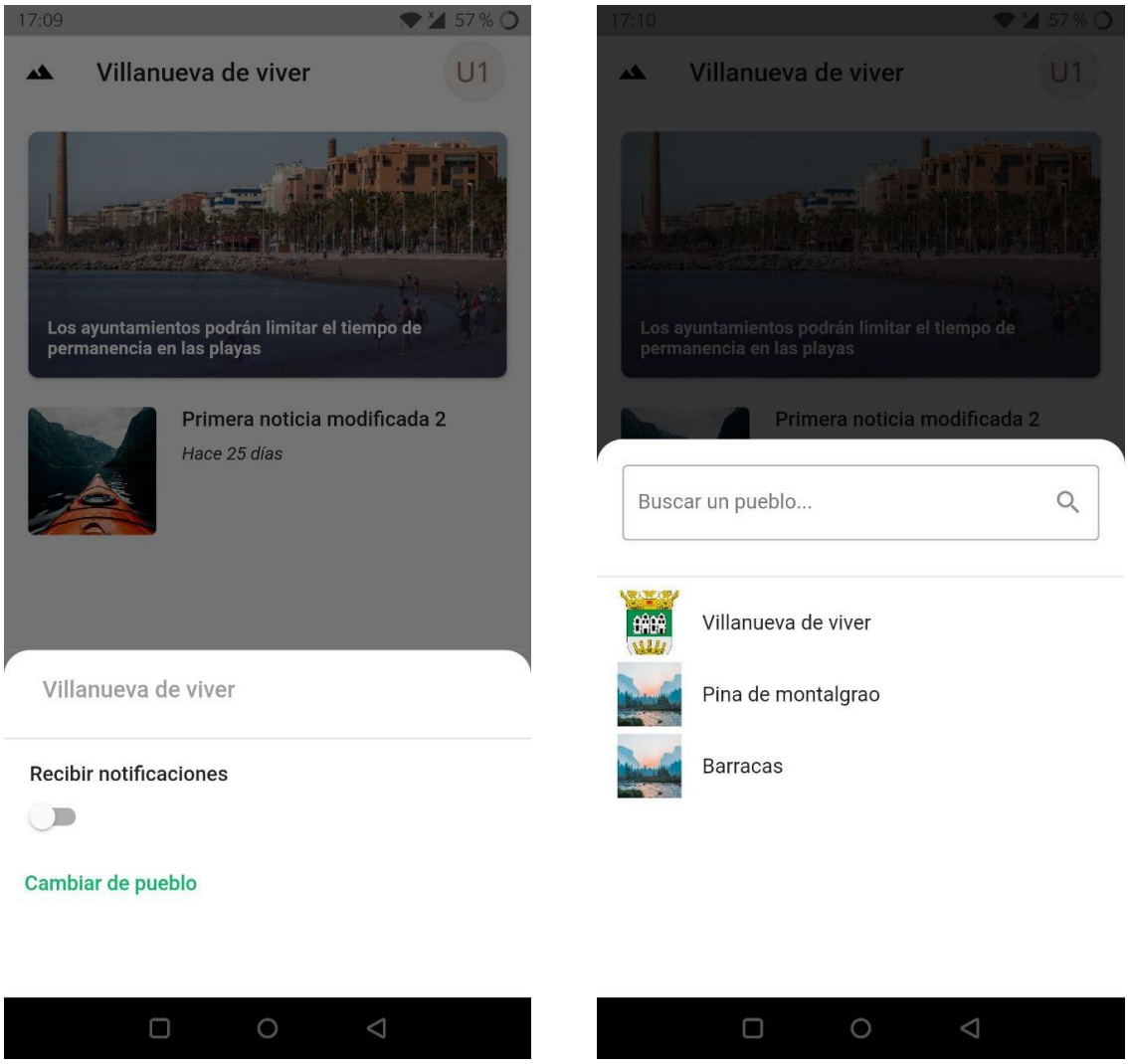
Figura 4.11: Clase *app.dart*.

La pantalla principal es una pantalla de bienvenida que consta únicamente del logotipo de la aplicación, pero tiene la función de comprobar si hay una sesión iniciada, con lo que nos llevaría a la página principal. Si no la hay, llevaría a la pantalla de selección de pueblo o autenticación. Para este último caso habría tres opciones: acceder sin autenticarse, registrarse como nuevo usuario en la plataforma o iniciar sesión si ya se disponía de una cuenta. Al llevar a cabo cualquiera de estas tres opciones se accedería a la página principal.

La clase *HomeScreen* es la más extensa y compleja del proyecto ya que es el centro de la aplicación que permite acceder al resto de pantallas. En la figura 4.12 se destacan tres elementos de la pantalla principal que no se han comentado hasta ahora:

Figura 4.12: Pantalla principal de la aplicación.

1) **Menú de pueblo**. Al pulsar este icono se abre un menú que permite realizar dos acciones. La primera se trata de un selector que permite al usuario decidir si quiere recibir notificaciones sobre ese pueblo o no. La otra acción permite al usuario seleccionar otro pueblo de los disponibles en la plataforma (figura 4.13). Además, contiene una barra de búsqueda que facilita la elección de pueblo en caso de haber un gran número de ellos.

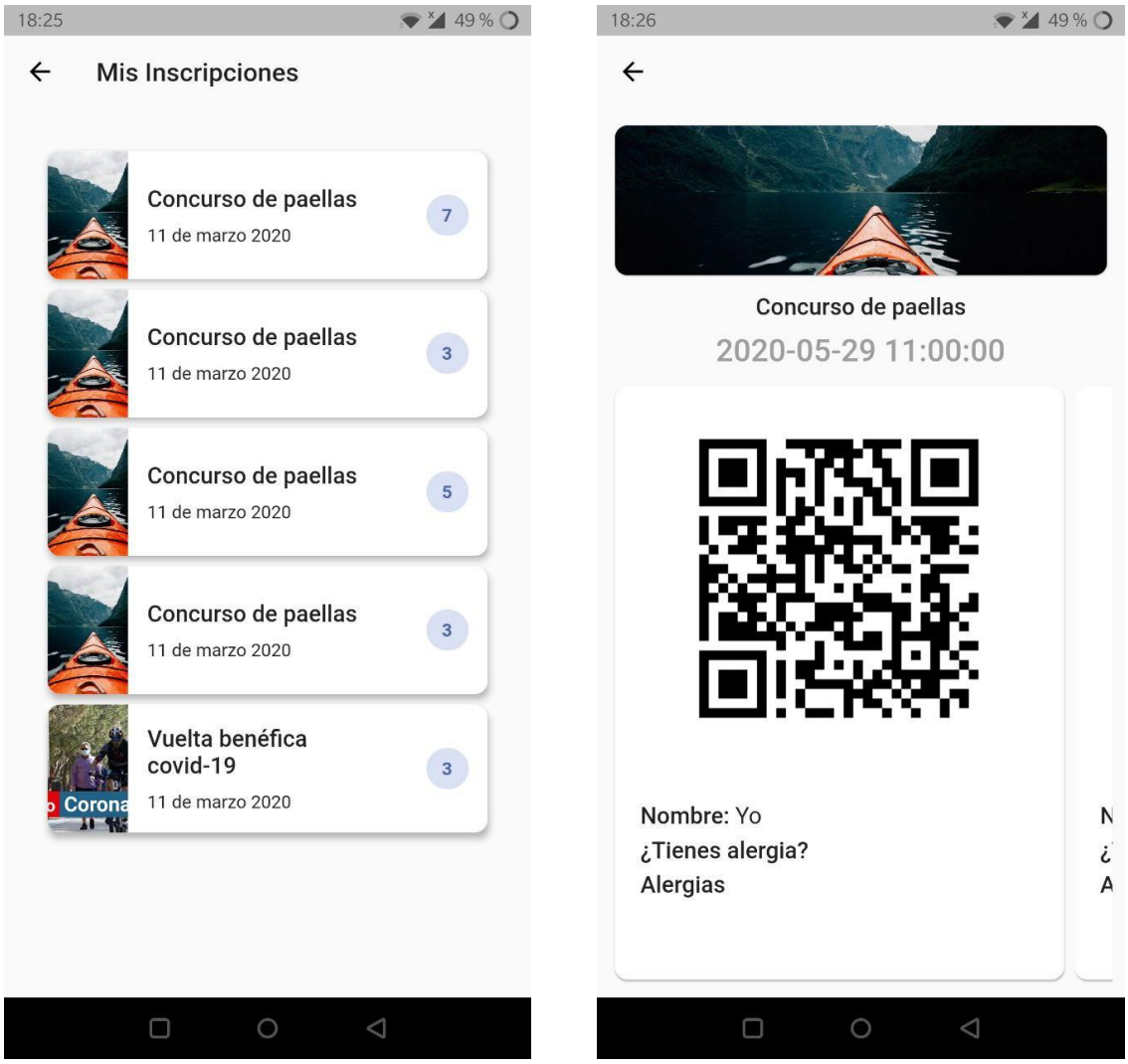


(a) Primer *modal* del menú.

(b) Opción *Cambiar de pueblo*.

Figura 4.13: Menú de pueblo.

2) **Menú de usuario**. Si pulsamos en el avatar del usuario aparecerá un menú similar al anterior con dos opciones. La primera lleva a una pantalla con todas las inscripciones a eventos realizadas en la plataforma, pudiendo desde aquí acceder al código *QR* del ticket además de a información adicional sobre la inscripción. Este código *QR* podrá ser escaneado por el administrador del evento para confirmar la asistencia del participante (figura 4.14). La otra opción permite cerrar la sesión, llevando de vuelta a la pantalla de inicio de sesión.



(a) Lista de *Mis Inscripciones*.

(b) Inscripción detallada.

Figura 4.14: Opción del menú de usuario *Mis Inscripciones*.

3) **Menú de gestión**. Este botón flotante, que aparece solamente a los administradores del pueblo, permite acceder a la pantalla de configuración de la pestaña actual. Por ejemplo, desde la pestaña de eventos se accederá a la pantalla de configuración de eventos.

4.1.3. Problemas del desarrollo

La mayoría de problemas que han surgido a lo largo del proyecto han sido causados por la inexperiencia. Tratar correctamente la respuesta de una petición *HTTP* o mantener la sesión activa del usuario son algunos de los detalles que, en un primer momento, requirieron algo de tiempo, pero que una vez aprendidos ya no supusieron ningún problema. De entre todos, destaco principalmente tres:

- **Tratar objetos JSON**. Todas las llamadas a la *API* devuelven un objeto *JSON* del cual se extrae la información necesaria. No obstante, algunas de las respuestas tienen una estructura compleja de la cual hay que extraer los datos y realizar operaciones con ellos. En concreto, el mayor problema fue con las inscripciones, que tienen campos específicos que no son comunes en todas, como por ejemplo las alergias.
- **Notificaciones automatizadas de *Firebase***. En un principio estaba planeado que, al crear un bando, se notificara a todos los usuarios suscritos a ese pueblo. Sin embargo, el reducido tiempo del proyecto no permitió que se pudiera automatizar las notificaciones, aunque sí que fue posible hacer que funcionaran mediante la consola de *Firebase*.
- **Actualización de interfaces y redirección**. En algunas de las acciones que puede realizar un administrador, por ejemplo la creación de una noticia o de un evento, no se redirige correctamente a la pantalla pertinente. Además, es necesario cambiar de una pestaña a otra para ver el cambio realizado. Este problema se detectó cuando la estancia en prácticas estaba llegando a su fin por lo que no hubo tiempo de solucionarlo. Para ello, se debía revisar la comunicación de la aplicación con el *WebView* en el cual se realizan estas acciones.

4.2. Verificación y validación

Para comprobar el correcto funcionamiento de la aplicación móvil se han realizado pruebas de aceptación de usuario por parte de compañeros de la empresa. Estas pruebas se realizaron a lo largo de la última semana y en ellas se iban nombrando los fallos y la forma de solucionarlos, así como posibles mejoras que añadir en un futuro.

Una gran ventaja de *Flutter* es que permite ver en tiempo real, en un dispositivo real, los cambios que se van realizando en el código. Por eso, el número de fallos relativos a las interfaces es muy bajo, ya que en cuanto se veía uno de estos errores, se corregía inmediatamente. De este modo, la consecución de objetivos respecto a los requisitos funcionales definidos ha sido casi completa. La parte de notificaciones no ha sido totalmente implementada y hay detalles que necesitan ser pulidos a la hora de lanzar la aplicación al mercado, como las redirecciones o pequeños detalles de interfaz. No obstante, el resto de requisitos se cumplen correctamente en su totalidad.

Otro tipo de pruebas, como las de integración o de unidad, no se han podido llevar a cabo debido al tiempo dedicado a la implementación del código. Aunque este tipo de pruebas son importantes, se priorizaron las pruebas de aceptación porque era necesario obtener las sensaciones que tenían los usuarios al usar la aplicación, ya que hay errores que son de concepto y no de codificación, como por ejemplo una navegación complicada o una redirección sin sentido al ejecutar una tarea.

Capítulo 5

Conclusiones

En estos tres meses de estancia en prácticas se han conseguido alcanzar todos los objetivos del proyecto. Exceptuando la automatización de las notificaciones, todos los requisitos funcionales se cumplen sin problemas. Sin embargo, para lanzar una aplicación de estas características al mercado es necesario emplear más tiempo en pequeños detalles que mejoren la experiencia de usuario. Además, las aplicaciones móviles de este tipo necesitan un mantenimiento constante posterior al lanzamiento, escuchando el *feedback* de usuarios corrientes y administradores.

En el ámbito formativo, este proyecto se ha realizado íntegramente utilizando *Flutter*, una herramienta muy potente que desconocía hasta entonces. Con *Flutter*, el desarrollo de aplicaciones móvil se vuelve más sencillo gracias a su funcionamiento mediante *widgets*, que simplifican la construcción de interfaces de usuario. Además, el código está escrito en Dart, un lenguaje de programación tan similar a Java que, prácticamente con lo aprendido en la carrera sobre este, ha bastado para construir la aplicación.

Finalmente, en cuanto a mi experiencia personal, creo que la estancia en prácticas ha sido muy productiva y gratificante, tanto en el ámbito formativo como en el profesional. La experiencia con la empresa ha sido positiva, ya que, el hecho de ser tan pequeña, hace que sea mucho más acogedora. El ambiente con los compañeros era muy agradable y siempre que me surgía algún tipo de duda me la solucionaban de inmediato y amablemente. A pesar de la situación excepcional surgida a causa del COVID-19, he trabajado cómodamente y sin ningún problema derivado de ello.

Bibliografía

[1] Wisclic Tech: *Ficha de empresa, Espaitec*. [Última visita: 01/06/2020]. URL: https://espaitec.uji.es/el-parque/empresas-del-parque/wisclic-tech-/.

[2] Flutter: *Technical overview*. [Última visita: 10/06/2020]. URL: https://flutter.dev/docs/resources/technical-overview.

[3] Flutter: *Layouts en Flutter*. [Última visita: 10/06/2020]. URL: https://flutter-es.io/docs/development/ui/layout.

[4] Flutter: *Adding interactivity to your Flutter app*. [Última visita: 10/06/2020]. URL: https://flutter.dev/docs/development/ui/interactive.

[5] Flutter: *Start thinking declaratively*. [Última visita: 10/06/2020]. URL: https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative.