Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Desarrollo de una aplicación para la evaluación del estado socioemocional de alumnos refugiados

Alumno:
Juan Torres Viloria

Tutor:
Diego García Álvarez

A mi padre, a mi madre y a mi hermana, por el apoyo incondicional e incesante que siempre me ofrecen.

Agradecimientos

Gracias a toda mi familia, en especial a mis padres, mi hermana y mi tía, por su motivación constante y apoyo que siempre me han regalado y que me impulsa a avanzar hacia delante continuamente.

Gracias a *Choscar*, mi motivación inicial para aventurarme en esta carrera y quien ha seguido ayudandome desinteresadamente en todo momento.

Gracias a Sergio, amigo con el que he realizado cada trabajo, práctica o problema en esta carrera y que ha sido mi fiel compañero.

Gracias a todo el grupo *Fainac*, por las incontables charlas y buenos ratos en *Discord* y a mis amigos de *Amaranto 370*. De manera pasiva y sin saberlo, han sido un apoyo moral durante todo este proyecto.

Gracias a Diego, mi tutor, quien nunca ha perdido la esperanza en mí y ha estado disponible en cualquier momento para ayudarme a avanzar con el proyecto, motivándome de forma continua.

Gracias en general a todos mis amigos por preocuparse por mí estos meses y alegrarme los días.

Gracias.

Resumen

El objetivo de este proyecto es el desarrollo de una aplicación móvil que mejore la adaptación al sistema educativo y social de los refugiados en las aulas.

La aplicación girará en torno a las actividades. Una actividad detalla una pequeña tarea o ejercicio especificado por un profesor que el alumno deberá completar realizando la tarea propuesta y completando un cuestionario relacionado con la misma. Las actividades serán asignadas por los profesores y para ello podrán elegir una asignatura (si quieren diferenciar una actividad asociada a las matemáticas únicamente, por ejemplo) y un estado de ánimo. Esto último es referido a si una actividad es de ayuda cuando un alumno se encuentre desmotivado, triste o desconcertado; o si por el contrario es más pertinente realizarla estando alegre y con energías (como podría ser una actividad asociada a la Educación Física).

Los usuarios alumnos podrán especificar su estado de ánimo en un apartado de la aplicación, eligiendo una entre seis emociones básicas.

Esta aplicación permitirá a los alumnos ver actividades asignadas y completar cuestionarios asociados a las mismas; mientras que los profesores podrán crear y asignar dichas actividades, valorando siempre el estado en el que se encuentran sus alumnos. Ambos cuentan con un perfil donde podrán ver sus datos, y en el caso de los profesores, contarán con un listado de sus alumnos asignados donde podrán ver el estado de ánimo de cada uno, sirviendo de guía para crear/asignar actividades según qué caso.

Esta aplicación pretende servir de ayuda en las aulas, haciendo la vida más fácil al sector de alumnos con dificultades en la comunicación, así como a sus respectivos profesores.

Palabras clave: Actividad, alumno, estudiante, profesor, *Kotlin*, *Android*, *Jetpack Compose*, cuestionario.

Abstract

The goal of this project is the development of a mobile application that improves the adaptation to the educational and social system of refugees in the classroom.

The application will revolve around the activities. An activity details a small task or exercise specified by a teacher that the student must complete by performing the proposed task and completing a questionnaire related to it. The activities will be assigned by the teachers and for this they will be able to choose a subject (if they want to differentiate an activity associated only with mathematics, for example) and a state of mind. The latter refers to whether an activity is helpful when a student is unmotivated, sad or confused; or if, on the contrary, it is more pertinent to do it while being happy and energetic (such as an activity associated with Physical Education).

Student users will be able to specify their state of mind in a section of the application, choosing one of six basic emotions.

This application will allow students to see assigned activities and complete questionnaires associated with them; while teachers will be able to create and assign these activities, always seeing the status in which they are. Both have a profile where they can see their data, and in the case of teachers, they will have a list of their assigned students where they can see the state of mind of each one, serving as a guide to create/assign activities depending on the case.

This application aims to help in the classroom, making life easier for students with communication difficulties, as well as their respective teachers.

Keywords: Activity, student, teacher, *Kotlin*, *Android*, *Jetpack Compose*, questionnaire.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El presente trabajo corresponde con la memoria del Trabajo de Fin de Grado de Ingeniería Informática, documento que recoge todos los pasos a seguir a la hora de desarrollar un proyecto software, desde los objetivos iniciales hasta mejoras futuras que puedan surgir.

El proyecto consiste en realizar una aplicación móvil destinada a ayudar a alumnos refugiados en las aulas, proporcionando una gestión de actividades entre alumno-profesor propuesto por la Junta de Castilla y León.

1.1. Contexto

La presencia del alumando refugiado en las aulas requiere de una respuesta por parte del equipo docente a la altura de las circunstancias. Además, debido a las condiciones particulares de este colectivo, potenciar su desarrollo socioemocional es clave para su bienestar individual y social.

Este Trabajo de Fin de Grado se engloba dentro del proyecto IntegraCyL desarrollado por profesores de secundaria de Castilla y León y profesores de la Universidad de Valladolid.

En los últimos años las tecnologías han tenido un auge exponencial en todos los sectores, incluyendo el sector educativo. No son pocos los centros e instituciones que aplican las tecnologías para su uso diario en el material docente, un claro ejemplo son herramientas como el Campus Virtual, la plataforma Sigma para la gestión académica o la página de la Escuela de Ingeniería Informática de Valladolid, las cuales usamos prácticamente a diario.

Con un pretexto similar, se busca la integración de los alumnos refugiados en el entorno tecnológico, en este caso mediante una aplicación móvil. Sin embargo, en numerosas ocasiones las aplicaciones centradas a estos colectivos se centran en una finalidad meramente lingüística, siendo en su mayoría programas de aprendizaje del idioma del país de destino y

con escasos recursos para su continuidad para su desarrollo personal y social.

1.2. Objetivos

El principal objetivo de este proyecto es el de desarrollar una aplicación que permita la mejora del bienestar emocional del alumnado refugiado y su autorregulación para favorecer los procesos de inclusión social. La realización de pequeñas actividades asignadas por el equipo docente, y la oportunidad de mostrar el estado de ánimo de manera intuitiva serán claves en el desarollo del proyecto. Todo esto estará sujeto a los siguientes objetivos específicos:

1. Desarrollar una aplicación móvil.
2. Seguir todas las etapas del desarrollo software.
3. Desarrollar una documentación fidedigna.

Finalmente, existen también objetivos impuestos por el desarrollador que complementan a los objetivos académicos:

1. Conocer una tecnología actual y nueva para el alumno.
2. Crear una interfaz agradable e intuitiva al usuario.
3. Seguir una estructura de aplicación de acuerdo con los principios de *Clean Architecture* [4].

1.3. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 2 Estado del tema: Recoge aplicaciones similares al sistema propuesto.

Capítulo 3 Planificación: Describe la metodología seguida para la realización del proyecto, asi como el análisis de riesgos y la planificación.

Capítulo 4 Requisitos: En este capítulo se identificarán los requisitos, tanto funcionales, como no funcionales y de información, así como el Modelo de Casos de Uso y sus descripciones.

Capítulo 5 Análisis: En él se incluye el modelo de dominio, el modelo de análisis y realización de casos de uso.

Capítulo 6 Tecnologías y herramientas utilizadas: Descripción de las tecnologías y herramientas utilizadas durante el transcurso de este proyecto.

Capítulo 7 Diseño: Se muestra la arquitectura de la aplicación, así como el diseño de la base de datos y bocetos de la interfaz de usuario.

Capítulo 8 Implementación: Se explica cómo se han implementado las partes más importantes de la aplicación y qué estrategias se han tomado.

Capítulo 9 Pruebas: Diferentes pruebas realizadas para comprobar el sistema.

Capítulo 10 Seguimiento del proyecto: Muestra la comparativa entre la fecha planificada de cada fase y la fecha real tanto de comienzo como de final.

Capítulo 11 Conclusiones y trabajo futuro: Se realiza una valoración general de lo que ha sido el proyecto y se reflexiona sobre algunas posibles mejoras futuras.

Anexo A Manuales:

Anexo B Resumen de enlaces adicionales:

Bibliografía Referencias bibliográficas consultadas durante la realización del proyecto.

Capítulo 2

Estado del tema

En este capítulo se describen algunas aplicaciones que guardan cierta similitud o que pueden llegar a estar relacionadas con la aplicación desarrollada.

Como se ha comentado en el Capítulo 1, las alternativas existentes hoy en día que ofrecen servicios docentes para refugiados se pueden dividir en dos grupos: enfocadas a la lingüística o un juego/historia en su totalidad.

Con la finalidad de ofrecer una aplicación innovadora en el ámbito educativo para los refugiados en la aulas, se ha hecho una extensa búsqueda entre aplicaciones relacionadas disponibles en la Google Play Store, teniendo en cuenta sus funcionalidades y viendo las diferencias.

2.1. Aplicaciones relacionadas con el tema de estudio

Los siguientes apartados muestran un conjunto de aplicaciones, comparando sus puntos fuertes y sus debilidades, demostrando la originalidad de la aplicación que ofrece este proyecto.

2.1.1. aBlapp

La aplicación aBlapp [5] es un proyecto de acceso ilimitado orientada a personas refugiadas y migrantes en Argentina. Está orientada en los intercambios iniciales de interacción social en situaciones formales e informales: saludar, presentarse, dar información personal, completar un formulario y usar recursos para facilitar el intercambio. Constituye el entorno de entrada de la aplicación, con contenidos mínimos que se retoman en los entornos temáticos.

Ofrece seis entornos diferentes de comunicación e interacción social de la vida cotidiana, siendo: trámites, salud, trabajo, vivienda, transporte, compras.

Figura 2.1: Capturas aBlapp.

Puntos fuertes:

- Gran variedad de situaciones sociales a las que adaptarse.
- Ayuda a la inclusión en el sistema laboral y de tramitación.
- Disponible en varios idiomas.

Puntos débiles:

- Enfocada a un sector adulto y no a un entorno docente. Esta es la principal desventaja.
- Pequeños *bugs* que hacen salirte de la aplicación.
- Barra de navegación superior innecesaria ya que sólo existe una pantalla.
- Errores de interfaz al pulsar ciertos elementos.

2.1.2. Kids Story Creator

Kids Story Creator [6] es una aplicación algo más parecida al objetivo de este trabajo. Se trata de una aplicación para profesores y alumnos donde al escribir un tema, genera un pequeño cuento gracias al uso de técnicas de Inteligencia Artificial.

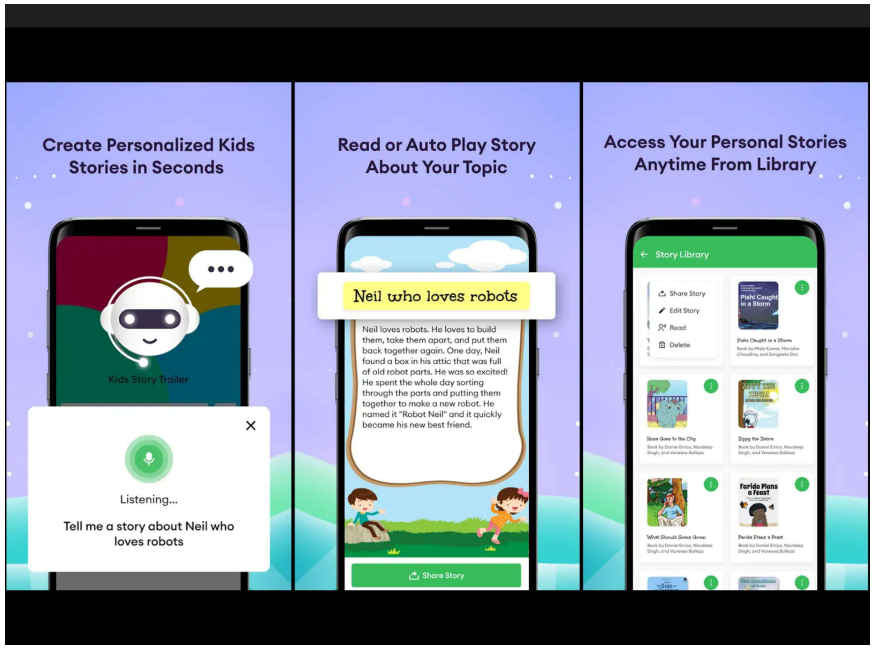

Figura 2.2: Capturas Kids Story Creator.

Puntos fuertes:

- Enfocada a menores y jóvenes.
- Gran variedad de personalización y ajustes.
- Crear cuentos cortos de manera gratuita y compartirlos.

Puntos débiles:

- No estimula el aprendizaje o independencia de los niños.
- Pequeños errores de interfaz.
- Límite para crear cuentos, al décimo debes hacer un ingreso para conseguir más tokens canjeables y generar nuevas historias.

Capítulo 3

Planificación

Este capítulo presenta la planificación propuesta para la realización del proyecto, la cual incluye metodología a aplicar, análisis de riesgos y el plan de trabajo seguido durante el periodo de desarrollo del proyecto, comenzando el 13 de febrero.

3.1. Metodología

Para el proceso de desarrollo de este proyecto se ha aplicado la Metodología en Cascada [7], que es un desarrollo secuencial en el cual al final de cada etapa se realiza una revisión para poder avanzar a la siguiente. En este proyecto se han definido 5, mostradas en la Figura 3.1. Estas etapas son:

- **Requisitos:** En esta etapa se identifican la viabilidad del proyecto, los requisitos de usuario y funcionalidades del sistema sin entrar en detalles técnicos de implementación. Se establecen la Descripción detalla del sistema, el Análisis de Requisitos de la aplicación y el Modelo de Casos de Uso.
- **Análisis:** Fase donde se identificarán algunas de las funcionalidades que tendrá la aplicación, es decir, sus casos de uso, y la presentación del modelo del sistema. Se establecen el Modelo de Dominio utilizado y el Modelo de Análisis junto con sus clases.
- **Diseño:** Durante esta fase se realizan procesos para obtener la estructura interna de la aplicación que servirá como base para su construcción. Se establece el Modelo de Diseño.
- **Codificación:** En esta fase se elige la tecnología para desarrollar la programación de la aplicación y se implementa el código fuente.
- **Pruebas:** Fase para comprobar el correcto funcionamiento de la aplicación mediante el uso de pruebas. Haciendo uso de prubas de aceptación, se analizan los Casos de Uso definidos.

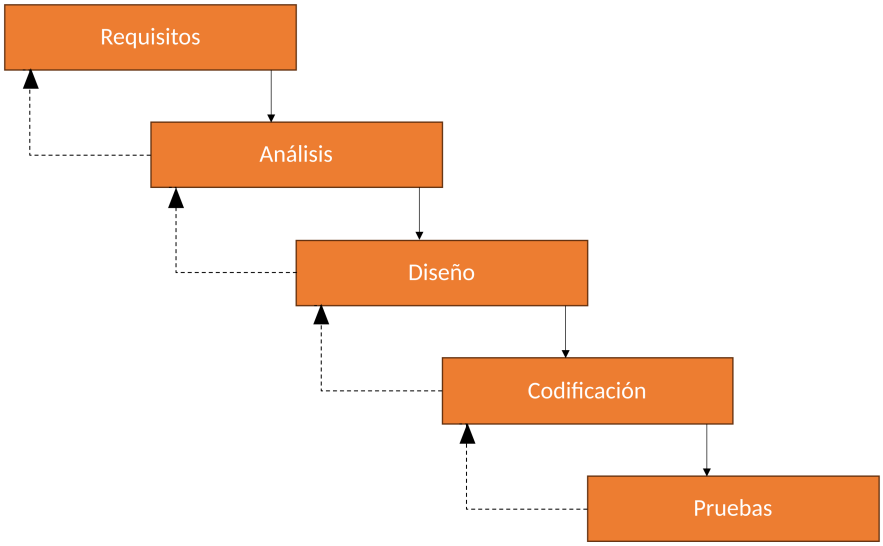

Figura 3.1: Diagrama de Cascada.

3.2. Análisis de riesgos

El Análisis de Riesgos es una herramienta de prevención con la que se pueden pronosticar las amenazas con potencial de afectar el desempeño de algún proceso. Se puede crear un plan de prevención y según los riesgos detectados tomar diferentes acciones: aceptarlo, mitigarlo o evitarlo.
La finalidad del Análisis de Riesgos es identificar, analizar y tratar esos riesgos antes de que ocurran de modo que se pueda actuar según convenga en cada caso. No todos los riesgos van a afectar al proyecto de la misma manera, por lo que es necesario medir y estipular los diferentes niveles en los que puede interferir un riesgo.

Para cada **riesgo** se establecen una Probabilidad y un Impacto.

- **Probabilidad:** La probabilidad de materialización del riesgo, este valor será aproximado ya que es imposible saber con exactitud la probabilidad de ocurrencia.

Probabilidad	Rango
Alta	Mayor de 50 % de probabilidad de ocurrencia.
Significante	30-50 % de probabilidad de ocurrencia.
Moderada	10-29 % de probabilidad de ocurrencia.
Baja	Menos de 10 % de probabilidad de ocurrencia.

Tabla 3.1: Descriptores cualitativos de la probabilidad de un riesgo y sus valores de rango asociados.

- **Impacto**: El impacto es la consecuencia efectiva que supondrá la ocurrencia del riesgo.

Impacto	Rango
Alto	Más de 30 % por encima del presupuesto.
Significante	20-29 % por encima del presupuesto.
Moderado	10-19 % por encima del presupuesto.
Bajo	Menos de 10 % por encima del presupuesto.

Tabla 3.2: Descriptores cualitativos del impacto de un riesgo y sus valores de rango asociados.

3.2.1. Riesgos

Los riesgos observados para este proyecto son:

R001	Fallo en la estimación
Descripción	Debido a que la estimación es una aproximación, es posible que no se sigan las fechas señaladas, si no se sigue la estimación afectará a la duración final del proyecto.
Probabilidad	Moderada.
Impacto	Significante.
Acciones de mitigación	▪ Organizar un horario de trabajo en un entorno agradable.
Acciones correctivas	▪ Reorganizar el horario establecido.

Tabla 3.3: R001 Fallo en la estimación.

R002	Sin orientación
Descripción	A la hora de plantear el TFG, no tener motivación o idea fija sobre qué hacerlo.
Probabilidad	Moderada.
Impacto	Moderado.
Acciones de mitigación	▪ Leer cuidadosamente los TFG disponibles y encontrar uno que más se ajuste a los gustos. ▪ Concretar citas con profesores para poder encontrar algo al gusto.
Acciones correctivas	▪ De los TFG restantes escoger el más parecido a lo que interesa. ▪ Plantear un TFG fuera de los disponibles, que sea viable y cubra los gustos.

Tabla 3.4: R002 Sin orientación.

R003	Problemas con tutor
Descripción	Tener de tutor de TFG a un profesor con el que no se congenia.
Probabilidad	Baja.
Impacto	Significante.
Acciones de mitigación	▪ Hablar con los diferentes tutores disponibles para poder conocerles.
Acciones correctivas	▪ Tener una charla para intentar solucionar problemas. ▪ Contrarer una buena relación con el nuevo profesor para poder continuar con el rumbo del TFG lo antes posible.

Tabla 3.5: R003 Problemas con tutor.

R004	Baja por enfermedad
Descripción	En mitad del desarrollo del TFG, contraer una enfermedad y necesitar reposo.
Probabilidad	Moderada.
Impacto	Moderado.
Acciones de mitigación	▪ Crear una planificación del TFG donde se cuente con espacio suficente en caso de coger enfermedad. ▪ Empezar a realizar el TFG en el primer cuatrimestre para contar con más tiempo.
Acciones correctivas	▪ Acudir al médico para poder tener una rápida recuperación. ▪ Solicitar un aplazo de entrega de TFG debido a la situación.

Tabla 3.6: R004 Baja por enfermedad.

R005	Pérdida dispositivo donde se esté realizando.
Descripción	Realizando la memoria de TFG o programando, no tenerlo en la nube guardado y que el dispositivo colapse, generando pérdida de información.
Probabilidad	Baja.
Impacto	Alto.
Acciones de mitigación	▪ Crear copias de seguridad frecuentemente y guardarlas en un dispositivo fiable. ▪ Guardar una copia en la nube. ▪ Mandar cada cierto tiempo una version del trabajo estable al tutor, para que él/ella también lo tenga.
Acciones correctivas	▪ Intentar recuperar el trabajo pérdido iniciando el dispositivo con arranque seguro. ▪ Si se ha enviado alguna versión del trabajo al tutor, pedir que facilite la última versión y continuar desde ahí.

Tabla 3.7: R005 Pérdida dispositivo donde se esté realizando.

R006	Falta de conocimiento.
Descripción	No saber cómo avanzar en ciertas tareas o partes del TFG por falta de conocimientos necesarios, como puede ser un lenguaje de programación a usar, un patrón de diseño o una tecnología específica.
Probabilidad	Significante.
Impacto	Significante.
Acciones de mitigación	▪ Escoger un TFG donde se tengan los conocimientos requeridos. ▪ Estudiar y comprender la materia necesaria para el TFG antes de empezar a hacer nada.
Acciones correctivas	▪ Realizar algún curso de la materia que no se conozca. ▪ Solicitar ayuda a gente con experiencia para comprender ciertas nociones.

Tabla 3.8: R006 Falta de conocimiento.

3.3. Plan de Trabajo

La realización de este proyecto está comprendida entre el día 13 de febrero de 2023 y el día 23 de junio del mismo año, fecha tope para la entrega del trabajo en convocatoria ordinaria. El periodo abarca aproximadamente cinco meses, correspondientes al periodo lectivo del segundo cuatrimestre.

El siguiente plan de trabajo mostrado en la Figura 3.2 describe las tareas y su estimación de tiempo a seguir para completar con éxito el proyecto. Los tiempos son una estimación, esto quiere decir que los tiempos reales pueden ser sometidos a cambios debido a algún riesgo, a pesar de los planes de mitigación escogidos.

El plan de trabajo se divide en las cinco fases comentadas previamente.

Nombre de tarea	Duración	Comienzo	Fin
IntegraCyL App	**95 días**	**lun 13/02/23**	**vie 23/06/23**
Análisis de requisitos	**7 días**	**lun 13/02/23**	**mar 21/02/23**
Análisis de sistema	**3 días**	**mié 22/02/23**	**vie 24/02/23**
Diseño de sistema	**13 días**	**lun 27/02/23**	**mié 15/03/23**
Codificación	**62 días**	**jue 16/03/23**	**vie 09/06/23**
Pruebas	**10 días**	**lun 12/06/23**	**vie 23/06/23**

Figura 3.2: Plan de Trabajo.

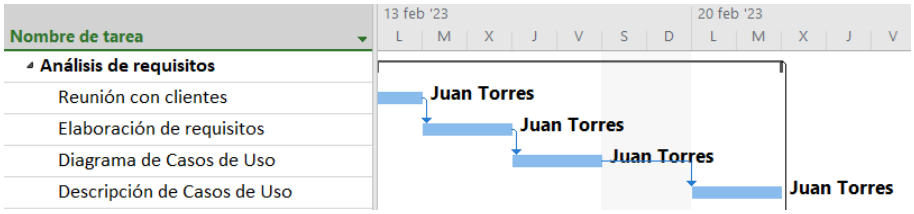

Figura 3.3: Diagrama de Gantt. Fase Análisis de Requisitos.

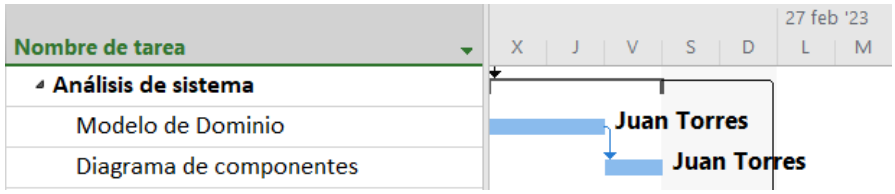

Figura 3.4: Diagrama de Gantt. Fase Análisis de Sistema.

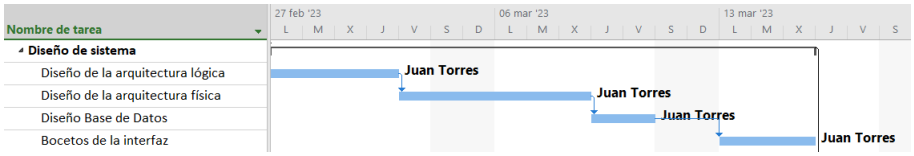

Figura 3.5: Diagrama de Gantt. Fase Diseño de Sistema.

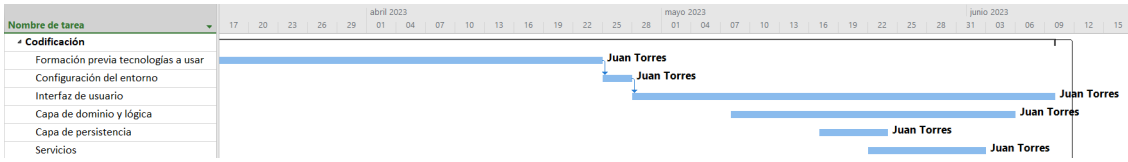

Figura 3.6: Diagrama de Gantt. Fase Codificación.

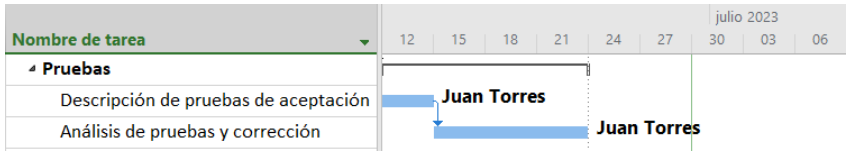

Figura 3.7: Diagrama de Gantt. Fase Pruebas.

Capítulo 4

Requisitos

4.1. Introducción

Este capítulo recoge la Descripción detalla del sistema, el Análisis de Requisitos de la aplicación, así como el Modelo de Casos de Uso.

Los requisitos son las características, las expectativas, los aspectos esperados o las capacidades que debe cumplir la aplicación, es decir es una descripción completa de lo que se espera del sistema. Los casos de uso sirven para describir los escenarios de uso que definen la funcionalidad del sistema.

4.2. Descripción detallada del sistema

Los usuarios definidos pueden ser Alumnos o Profesores. La aplicación se verá de diferente manera según que tipo de usuario haya iniciado sesión, por lo que es en el propio inicio de sesión cuando se hace esta diferenciación. El usuario tiene la posibilidad de mantener la sesión iniciada.

Las actividades son descripciones de pequeñas tareas o ejercicios que un alumno debe hacer, siempre cuentan un nombre de actividad, un cuestionario para que el alumno pueda responder preguntas asociadas a la misma y una descripción donde se especifique al alumno qué debe hacer o qué cosas debe tener en cuenta. De la misma manera, una actividad puede estar asignada a una asignatura específica o a un estado de ánimo. Todas las actividades cuentan con un estado que las define, siendo las opciones: En Curso, Pendiente o Finalizada.

Una vez iniciada la sesión, un usuario cuenta con tres vistas bien diferenciadas: Perfil, Actividades y Emociones en caso de iniciar como Alumno, o Perfil, Actividades, Alumnos y CrearActividad en caso de iniciar como Profesor. Puede alternar de sección gracias a una

barra de navegación situada en la parte inferior de la pantalla. Desde la barra de navegación, el usuario puede decidir a qué vista/sección acceder. El sistema muestra la vista asociada y se notifica de este cambio en la barra de navegación resaltando el nombre de la vista actual.

Independientemente del usuario, la sección Perfil es siempre igual, cambiando únicamente la información en pantalla. Para todos los usuarios se muestra el nombre completo del mismo encabezando la sección, un icono de perfil acompañado por el nombre de usuario. Un ejemplo para diferenciar y entender mejor el término nombre completo y nombre de usuario es Juan Torres Viloria y jtv01 respectivamente. Debajo del icono de perfil se indica que tipo de usuario y finalmente posterior a esta información se muestra, en caso de ser un profesor, el centro educativo al que pertenece; y en caso de ser un alumno, el centro junto con la etapa educativa, ciclo, curso y clase.

Cuando un usuario navega a la sección de Actividades, para el usuario alumno se muestran las actividades que tiene asignadas. Las opciones que el alumno tendrá aquí son: ordenar o filtrar las actividades, leer la información de cada una de ella y completarlas rellenando un cuestionario.

Las actividades se muestran en formado de listado y por encima de ellas existen tres opciones con los que el alumno puede filtrarlas según sus necesidades, las variantes son: En curso, Pendientes y Realizadas, es decir los estados de las propias actividades. Como la aplicación debe ser intuitiva y adaptada para menores, los propios nombres indican la función, es decir, si un alumno selecciona la opción En curso se mostrarán las actividades que esté en ese momento realizando. Si el alumno selecciona las tres opciones, se mostrarán todas las actividades asignadas para él.

En la vista previa de las actividades, el alumno puede ver una descripción de la actividad y los pasos a seguir para completarla. Si decide realizarla, podrá leer la descripción de la actividad y a continuación completar un cuestionario relacionado, formado por preguntas que complementan a la actividad. Cuando un usuario realiza una actividad completando el cuestionario, se considera a esa actividad como Finalizada.

Cuando es un profesor el que selecciona la opción de Actividades, la vista mostrada por pantalla es similar a la mostrada para un Alumno. La lista de actividades sigue el mismo esquema que para un alumno, pudiendo filtrarlas por En Curso, Pendientes o Realizadas. La diferencia radica en que en el caso de un alumno todas las actividades son las asociadas a él, pero en caso de un profesor son las actividades asignadas a sus alumnos. Esto quiere decir que cuando el profesor selecciona Realizadas, se muestran las actividades ya completadas por sus alumnos y que él les ha asignado. Además un Profesor puede ver el Cuestionario asociado a una Actividad que haya sido completado por un alumno.

Cuando selecciona CrearActividad aparece un listado de todas las Actividades disponibles. Entre la lista de actividades y el titulo de la sección, ahora se tendrá la opción para crear actividades desde cero. Cuando el profesor decide empezar una nueva actividad, aparece un cuestionario con diferentes campos a rellenar para poder crearla: nombre, descripción, emoción o curso/ciclo/etapa educativa asociada y preguntas que conjuntan el formulario a rellenar por el alumno una vez realizada. Cuando termina de completar los campos, la actividad se publica de forma remota y se puede utilizar para asignar a los alumnos.

La última sección es totalmente diferente según el usuario. En el caso de ser un alumno la sección se llama Emociones, y cuando la selecciona se muestran cuatro iconos asociados a cuatro emociones diferentes. El usuario alumno actualiza su estado de ánimo seleccionando uno de ellos, la funcionalidad de esto es que un profesor puede asignar actividades también en función del estado de un alumno.

Si el alumno selecciona el icono con una cara sonriente, indica que se encuentra alegre, contento o feliz, y a partir de esto el profesor se encarga de asignar una actividad acorde.

Si el usuario es un profesor, la última sección se llama Alumnos y es muy parecida a la de Actividades. En ella se muestra un listado de los alumnos asociados al profesor: si es profesor de matemáticas de 2º de la ESO, se mostrarán todos los alumnos que tengan esa asignatura.

Desde esta ventana un profesor puede consultar el estado de ánimo de sus alumnos y tomar las decisiones oportunas asignado actividades.

4.3. Requisitos

4.3.1. Requisitos funcionales

Los requisitos funcionales son aquéllos que describen qué debe hacer la aplicación

Código	Requisito	Descripción
RF-001	Inicio de sesión	La aplicación permitirá iniciar sesión con las credenciales propias del usuario, rellenando los campos de usuario y contraseña.
RF-002	Actividad alumno	La aplicación deberá ser capaz de mostrar las actividades de un alumno en la pantalla principal.
RF-003	Ordenar actividades	La aplicación deberá permitir al alumno ordenar sus actividades en función de unos parámetros.
RF-004	Información de actividad	La aplicación deberá mostrar información de una actividad específica cuando un usuario pulse sobre ella.
RF-005	Crear actividad	La aplicación deberá permitir a un profesor crear una actividad desde cero.
RF-006	Buscar actividad	La aplicación deberá permitir a un profesor buscar actividades ya existentes.
RF-007	Asignar actividad	La aplicación deberá permitir a un profesor asignar una actividad a un alumno.
RF-008	Analizar actividad	La aplicación deberá permitir a un profesor analizar una actividad de un alumno, mediante el cuestionario recibido asociado a la actividad.
RF-009	Mostrar estado ánimo	La aplicación permitirá a un alumno establecer su estado de ánimo.
RF-010	Información de alumnos	La aplicación deberá mostrar información de cada alumno asociado a un profesor en la vista del listado de alumnos.
RF-011	Cerrar Sesión	La aplicación deberá ser capaz de cerrar sesión.
RF-012	Mantener Sesión iniciada	La aplicación deberá permitir mantener la sesión iniciada una vez completado el inicio de sesión.

Tabla 4.1: Requisitos funcionales

4.3.2. Requisitos no funcionales

Los requisitos no funcionales definen propiedades del sistema, es decir, son aquéllos que describen cómo se debe a hacer el sistema.

Código	Requisito	Descripción
RNF-01	Sistema Operativo	La aplicación se implementará en Android (API 23 mínimo).
RNF-02	Versión	La aplicación funcionará para versiones de Android 6.0 o superior.
RNF-03	Entorno de desarrollo	La aplicación se desarrollará con Android Studio.
RNF-04	Almacenamiento de datos	El sistema almacenará los datos de las actividades, usuarios, cursos y cuestionarios en una base de datos (no relacional) Firebase.
RNF-05	Tiempo de respuesta	La aplicación deberá garantizar que el tiempo de respuesta en el 95 % de los casos será inferior a 5 segundos.
RNF-06	Modo vertical	La aplicación solo se podrá utilizar en modo vertical.
RNF-07	Intuitivo	La aplicación deberá tener una interfaz intuitiva para el usuario. Para ello se usará la libreria Jetpack Compose de Android.
RNF-08	Adaptación	La aplicación estará adaptada para los usuarios menores de edad o con capacidad de lectura limitada, se mantendrá un diseño simple y vocabulario ajustado.

Tabla 4.2: Requisitos no funcionales

4.3.3. Requisitos de información

Los requisitos de información son aquellos que describen qué debe almacenar la aplicación.

Código	Requisito	Descripción
RI-001	Usuarios	La aplicación deberá almacenar el nombre, contraseña y curso de los usuarios.

RI-002	Actividades	La aplicación deberá almacenar las actividades: nombre, descripción, imagen asociada, cuestionario.
RI-003	Cuestionarios	La aplicación deberá almacenar los cuestionarios: preguntas, respuestas y estado de la actividad.

Tabla 4.3: Requisitos de información

4.4. Casos de uso

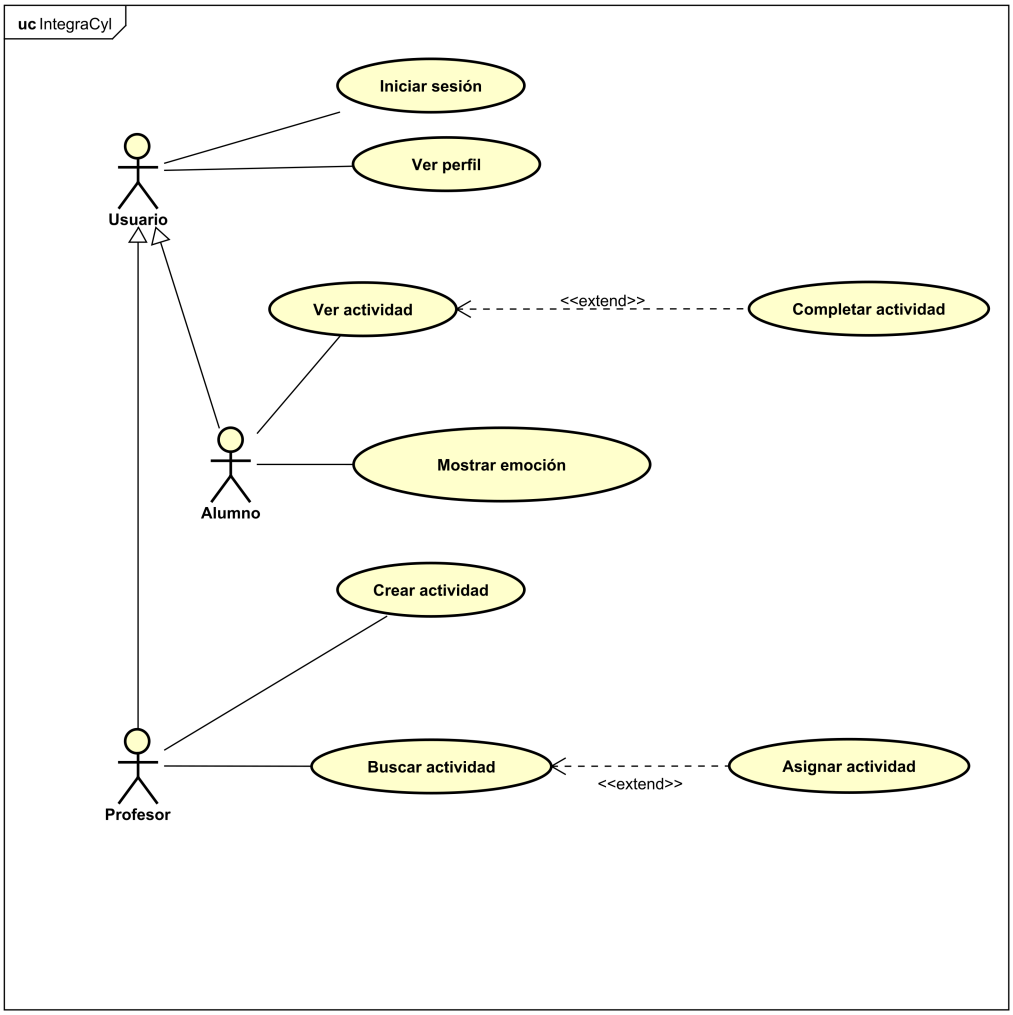

Figura 4.1: Casos de uso

4.4.1. Descripción casos de uso

A continuación se describirán los casos de uso mostrados en la Figura 4.1

Identificador	CU-01 Iniciar sesión
Actor principal	Usuario.

Descripción	El usuario se autentifica en el sistema a través de su usuario y contraseña.
Precondiciones	1. El usuario debe estar registrado en el sistema. 2. El usuario debe disponer de conexión a Internet.
Secuencia Normal	1. El usuario abre la aplicación. 2. El sistema muestra los datos que debe rellenar el usuario. 3. El usuario rellena los datos y envía el formulario. 4. El servicio de autenticación comprueba que los datos son de un usuario registrado.
Secuencia Alternativa	5.1 Los datos de acceso son incorrectos, el sistema muestra un mensaje de información al usuario y acto seguido el caso de uso vuelve al paso 2. 5.2 El servicio no está operativo, el sistema muestra un mensaje de información y el caso de uso vuelve al paso 2.
Postcondiciones	1. El usuario se autentifica en la aplicación.

Tabla 4.4: CU-01. Iniciar Sesión.

Identificador	CU-02 Ver perfil
Actor principal	Usuario.
Descripción	El usuario accede a la información de la sección Perfil.
Precondiciones	1. El usuario debe estar autenticado.
Secuencia Normal	1. El usuario selecciona la sección Perfil en la barra de navegación. 2. El sistema navega hasta la pantalla de Perfil y muestra la información.
Postcondiciones	1. El sistema se encuentra en la vista de perfil.

Tabla 4.5: CU-02. Ver perfil.

Identificador	CU-03 Ver actividad
Actor principal	Alumno
Descripción	El alumno accede a la información de la sección actividades y revisa la información de una actividad
Precondiciones	1. El usuario debe estar autenticado.
Secuencia Normal	1. El alumno selecciona la sección Actividades en la barra de navegación. 2. El sistema navega hasta la pantalla de Actividades y muestra la información. 3. El alumno selecciona una actividad para acceder a su descripción. 4. El sistema despliega la información de la actividad seleccionada.
Secuencia Alternativa	4.1. Si el alumno decide completar la actividad se realiza 4.7 CU-04 Completar actividad.
Postcondiciones	1. El sistema se encuentra en la vista de actividades. 2. La actividad seleccionada está desplegada.

Tabla 4.6: CU-03. Ver actividad.

Identificador	CU-04 Completar actividad
Actor principal	Alumno
Descripción	El alumno completa un cuestionario asociado a una actividad.
Precondiciones	1. El alumno debe haber seleccionado una actividad.
Secuencia Normal	1. El alumno selecciona Completar en la actividad. 2. El sistema muestra el cuestionario asociado a la actividad. 3. El alumno rellena los campos y selecciona Finalizar. 4. El sistema guarda la información.
Postcondiciones	1. La actividad pasa a estado Completada.

Tabla 4.7: CU-04. Completar actividad.

Identificador	CU-05 Mostrar emoción
Actor principal	Alumno
Descripción	El alumno actualiza su estado de ánimo/emoción.
Precondiciones	1. El usuario debe estar autenticado.
Secuencia Normal	1. El alumno selecciona la sección Emociones en la barra de navegación. 2. El sistema navega hasta la pantalla de Emociones y muestra las opciones. 3. El alumno selecciona una emoción para establacer su estado de ánimo actual. 4. El sistema actualiza el estado de ánimo del alumno y lo guarda.
Postcondiciones	1. La emoción del alumno se actualiza.

Tabla 4.8: CU-05. Mostrar emoción.

Identificador	CU-06 Crear actividad
Actor principal	Profesor
Descripción	El profesor accede a la información de la sección actividades y crea a partir de cero una nueva actividad.
Precondiciones	1. El usuario debe estar autenticado.
Secuencia Normal	1. El profesor selecciona la sección Actividades en la barra de navegación. 2. El sistema navega hasta la pantalla de Actividades y muestra la información. 3. El profesor pulsa el botón para crear una nueva actividad. 4. El sistema despliega una ventana con un cuestionario para rellenar con la información de la nueva actividad. 5. El profesor rellena cada uno de los campos: nombre, asignaturas asociadas, emociones asociadas, descripción. 6. El profesor selecciona el botón Siguiente. 7. El sistema despliega una nueva ventana para crear el cuestionario asociado a la actividad. 8. El profesor completa los campos y selecciona Finalizar.
Secuencia Alternativa	5.1. Si alguno de los campos obligatorios está vacío, el sistema muestra un mensaje por pantalla. 6.1. Si el profesor selecciona Atrás, el caso de uso vuelve al paso 2.
Postcondiciones	1. El sistema se encuentra en la vista de actividades. 2. Se crea una nueva actividad.

Tabla 4.9: CU-06. Crear actividad.

Identificador	CU-07 Buscar actividad
Actor principal	Profesor
Descripción	El usuario profesor busca una actividad entre todas las posibles.
Precondiciones	1. El usuario profesor se encuentra en la vista de actividades.
Secuencia Normal	1. El usuario profesor selecciona una actividad. 2. El sistema despliega la información.
Secuencia Alternativa	2.1. Si no hay ninguna actividad que coincida con ese nombre, el sistema lo avisa por pantalla. 2.2. Si se encuentra una actividad y el profesor decide asignarla, se ejecuta 4.11 CU-08 Asignar actividad.
Postcondiciones	1. El sistema se encuentra en la vista de actividades.

Tabla 4.10: CU-07. Buscar actividad.

Identificador	CU-08 Asignar actividad
Actor principal	Profesor
Descripción	El usuario profesor asigna una actividad a un alumno o grupos de alumnos.
Precondiciones	1. El usuario profesor se encuentra en la vista de actividades.
Secuencia Normal	1. El usuario profesor selecciona una actividad. 2. El sistema despliega la información de la actividad. 3. El usuario profesor selecciona Asignar. 4. El sistema muestra un recuadro de texto. 5. El usuario profesor escribe el nombre de un alumno y selecciona Aceptar. 6. El sistema guarda la selección y asigna la actividad al usuario alumno.
Secuencia Alternativa	3.1. Si el profesor selecciona Atrás, el caso de uso vuelve al paso 2. 5.1. Si no existe un usuario alumno que coincida, el sistema no mostrará ningún resultado y el caso de uso vuelve al paso 4. 5.2. Si el profesor selecciona Atrás, el caso de uso vuelve al paso 2.
Postcondiciones	1. El sistema se encuentra en la vista de actividades. 2. Se asigna una actividad a un usuario alumno.

Tabla 4.11: CU-08. Asignar actividad.

Identificador	CU-09 Ver alumnos
Actor principal	Profesor
Descripción	El usuario profesor consulta los alumnos asignados.
Precondiciones	1. El usuario debe estar autenticado.
Secuencia Normal	1. El profesor selecciona la sección Alumnos en la barra de navegación. 2. El sistema navega hasta la pantalla de Alumnos y muestra la información.
Postcondiciones	1. El sistema se encuentra en la vista de Alumnos.

Tabla 4.12: CU-09. Ver alumnos.

Capítulo 5

Análisis

5.1. Introducción

En este capítulo se presentan el Modelo de Dominio y el Modelo de Análisis.

5.2. Modelo de Dominio

El modelo de dominio es un modelo conceptual en el que se describen las entidades, atributos y relaciones que del dominio del problema. El modelo propuesto cuenta con:

- **Actividades**, reflejan todas las actividades que habrá en el sistema. Podrán estar en diferentes estado según si han sido completadas o evaluadas y se les podrá asociar diferentes atributos; entre ellos curso, estado de ánimo o alumno. Al finalizarse, un alumno cuenta con la opción de realizar un cuestionario acorde a la misma.
- **Alumnos**, representan los usuarios capaces de visualizar actividades y completarlas. Pertenecen a un centro educativo y a su vez son parte de un curso, clase y grupo. También pueden ser nombrados como Estudiantes.
- **Profesores**, usuarios que pueden crear y asignar actividades a los alumnos. Pueden consultar el estado actual de sus alumnos asignados. Los alumnos asignados son aquellos que coinciden en curso, centro educativo y al menos una asignatura.
- **Cuestionarios**, asociado a una actividad, está formado por una serie de preguntas a las cuales un alumno debe responder y el profesor que asignó la actividad debe analizar.
- **Centros Educativos**, representan los centros que contarán con la aplicación y cuyos docentes y alumnos estarán registrados.

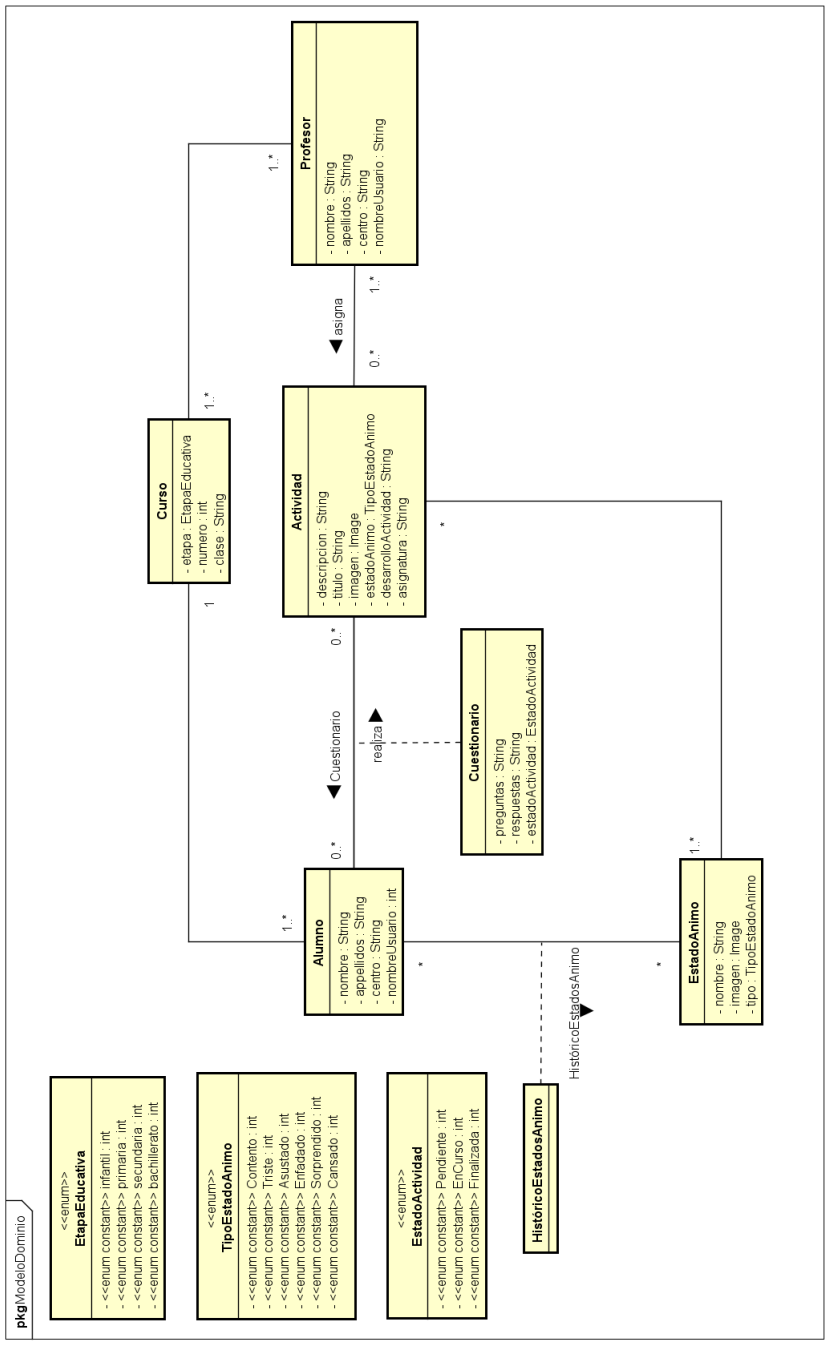

Figura 5.1: Modelo de Dominio.

5.3. Modelo de Análisis

5.3.1. Clases de Análisis

Una vez se ha comprendido el Modelo de Dominio del proyecto, se dotan a las clases constituyentes del mismo de las operaciones necesarias para llevar a cabo la funcionalidad descrita previamente en los Casos de Uso, Figura 4.1. El resultado se muestra en la Figura 5.2:

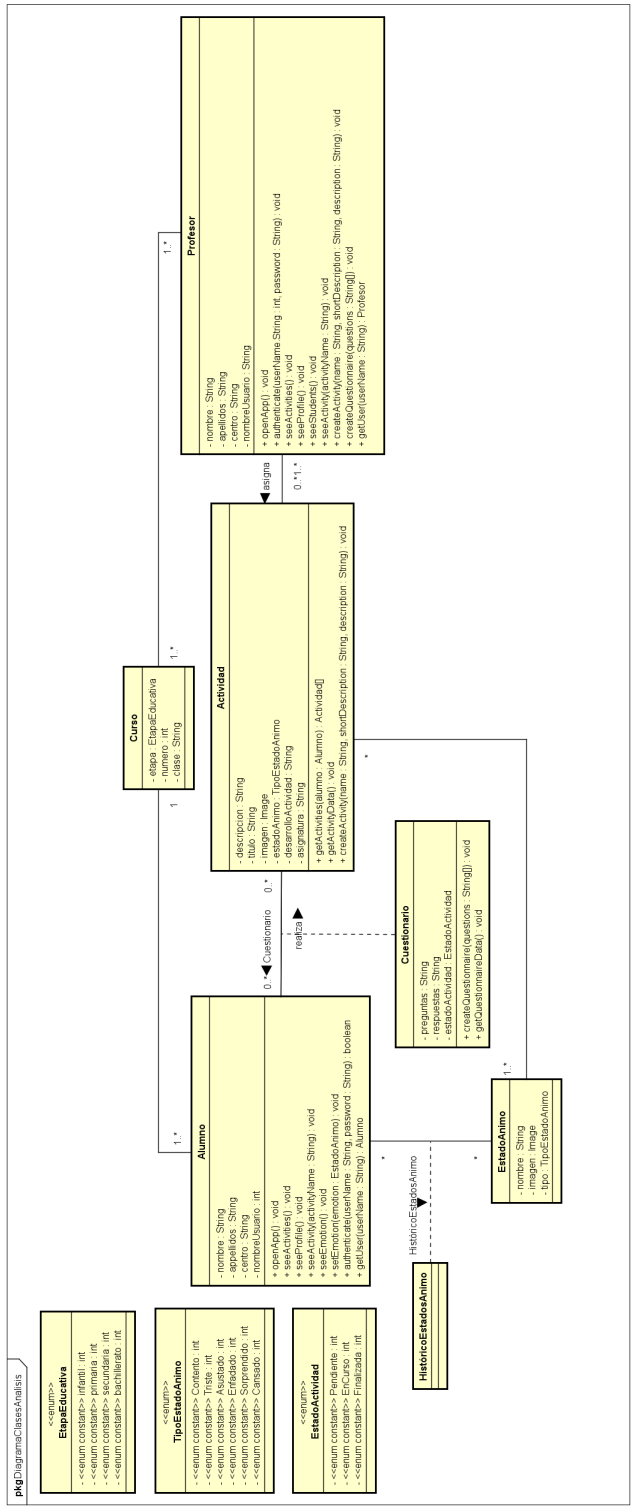

Figura 5.2: Diagrama de Clases de Análisis.

5.3.2. Realización de los Casos de Uso de Análisis

A continuación se muestran gráficamente los casos de uso de análisis más importantes para el proyecto, o más representativos, descritos con anterioridad.

CU-01 Iniciar Sesión 4.4

La Figura 5.3 muestra el diagrama de secuencia del Caso de Uso CU-01 Iniciar Sesión descrito en la Tabla 4.4.

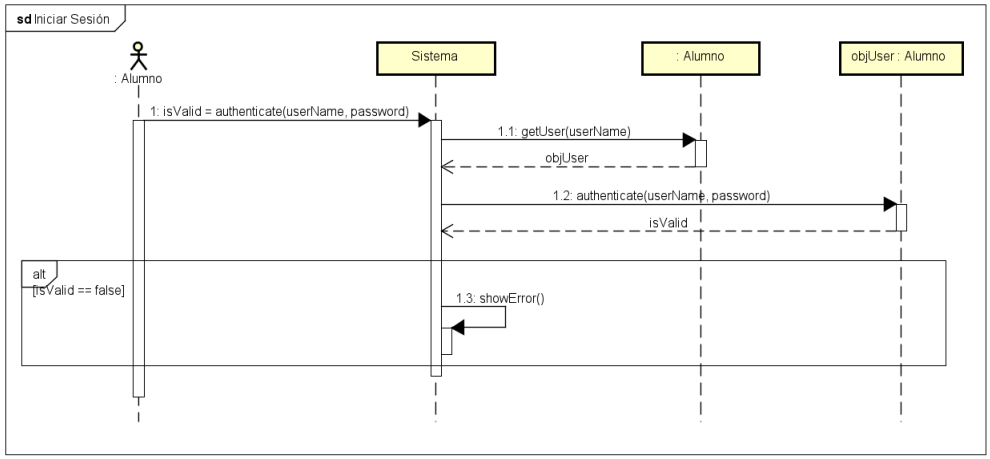

Figura 5.3: Realización CU-01 Iniciar Sesión.

CU-03 Ver actividad 4.6

La Figura 5.4 muestra el diagrama de secuencia del Caso de Uso CU-01 Iniciar Sesión descrito en la Tabla 4.6.

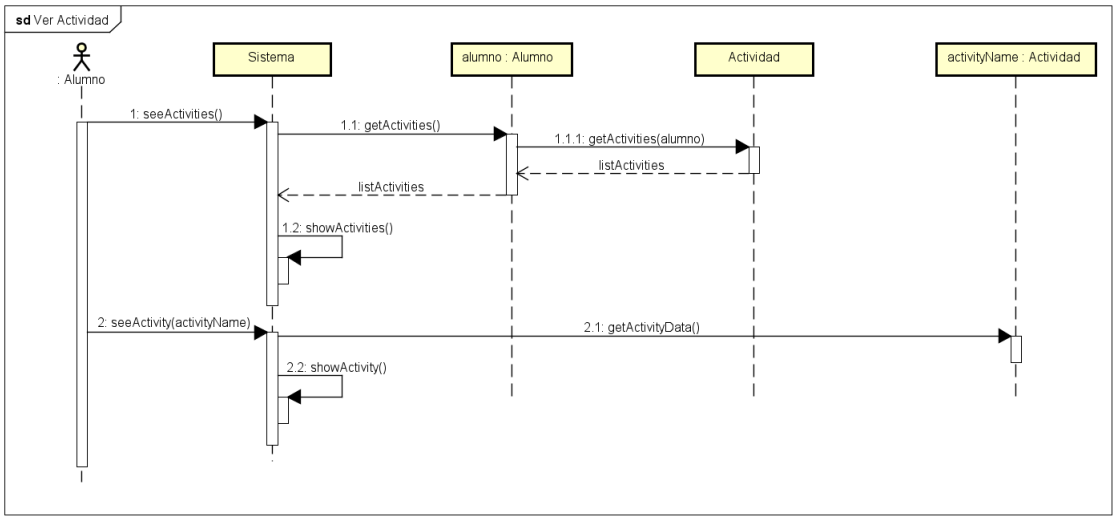

Figura 5.4: Realización CU-03 Ver Actividad.

CU-06 Crear actividad 4.9

La Figura 5.5 muestra el diagrama de secuencia del Caso de Uso CU-01 Iniciar Sesión descrito en la Tabla 4.9.

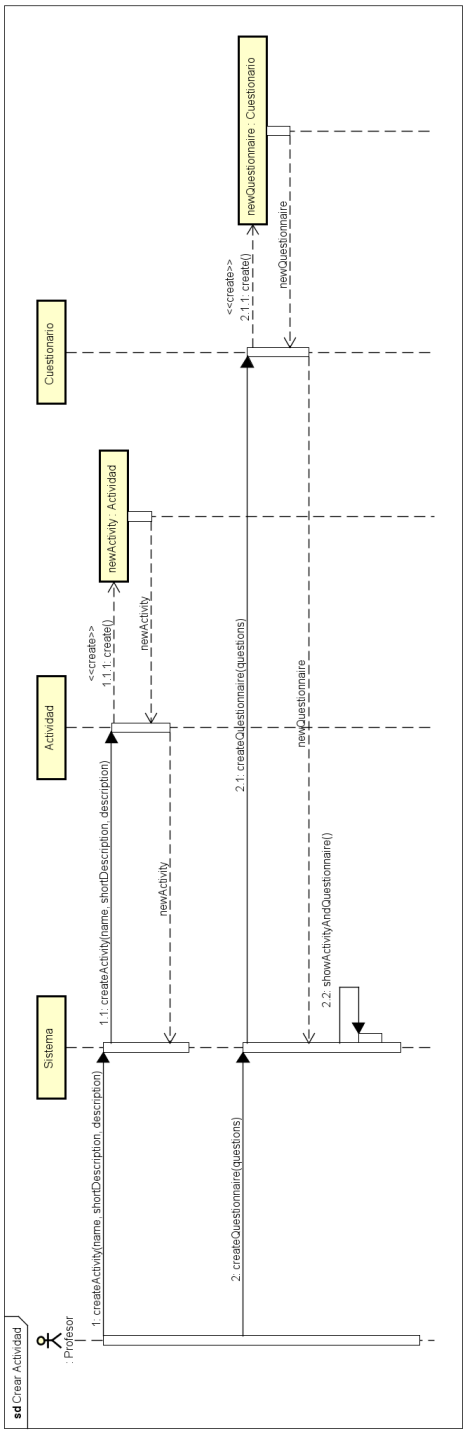

Figura 5.5: Realización CU-06 Crear Actividad.

Capítulo 6

Diseño

Este capítulo se centra en el *Workflow* de Diseño y en él se detallan las decisiones tomadas en la parte de diseño y la arquitectura de la aplicación propuesta. Recoge los diseños de la arquitectura lógica del sistema y de la arquitectura física, junto con bocetos iniciales de la interfaz gráfica, así como otros patrones de diseño usados.

6.1. Arquitectura lógica del sistema

En esta sección se especifica y describe la arquitectura lógica del sistema basada en el patrón arquitectónico Modelo-Vista-Modelo de Vista.

6.1.1. Patrón MVVM

El patrón Modelo-Vista-Modelo de Vista [8] es un patrón muy conocido e implementado en la mayoría de proyectos actualmente. Esto se debe a la gran ayuda que ofrece para separar de una manera limpia la lógica de presentación y de negocio de una aplicación (Modelo) de su interfaz de usuario (Vista), usando como nexo el Modelo de Vista.

Se ha decidido optar por esta alternativa porque al mantener una separación limpia entre la lógica de la aplicación y la interfaz de usuario, se logra evitar y abordar numerosos problemas de desarrollo, además de facilitar la prueba, mantenimiento y evolución de la aplicación.

Este patrón es además uno de los más utilizados en el desarrollo de aplicaciones móviles.

Los tres componentes mencionados y sus funciones son:

- **Vista:** Responsable de definir la estructura, el diseño y la apariencia de lo que ve el usuario en la pantalla. En el caso de esta aplicación, cada una de las vistas está definida mediante *Composables*, herramienta ofrecida por Jetpack Compose [9] y comentada en el Capítulo 7.
- **Modelo:** Formado por las clases no visuales que encapsulan la lógica de la aplicación. Representa el Modelo de Dominio de la aplicación, incluyendo modelo de datos junto con la lógica de dominio y acceso a datos; y la lógica de validación y de negocios. Estas clases se suelen usar junto con servicios o repositorios que encapsulan el acceso a datos y almacenamiento en caché. También se encarga de coordinar las relaciones de la vista con las clases de modelo que sean necesarias.
- **Modelo de Vista (ViewModel):** Implementa propiedades y comandos a los que la vista puede enlazar datos, y notifica a la vista los cambios de estado mediante eventos de notificación de cambios. Las propiedades y comandos que proporciona el modelo de vista definen la funcionalidad que ofrece la interfaz de usuario, pero la vista determina cómo se va a mostrar esa funcionalidad.

En la Figura 6.1 se puede observar el flujo existente entre los tres componentes y la navegabilidad. La Vista conoce el Modelo de Vista y el Modelo de Vista conoce el Modelo, pero el modelo desconoce el modelo de vista y el modelo de vista desconoce la vista. Por lo tanto, el modelo de vista aísla la vista del modelo y permite que el modelo evolucione independientemente de la vista.

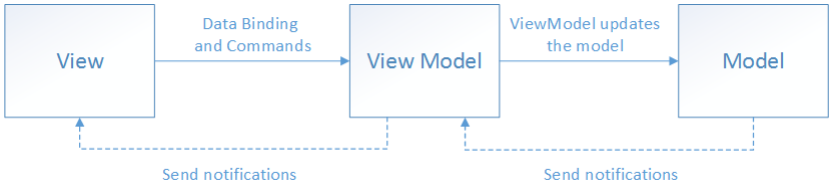

Figura 6.1: Flujo arquitectura MVVM

¿Qué es realmente ViewModel y cómo funciona?

ViewModel [1] es una clase que fue diseñada con el fin de almacenar y administrar datos relacionados con la Interfaz de Usuario (IU) de manera optimizada para los ciclos de vida. Si el sistema destruye o recrea un controlador de IU, como un *Activity* o *Fragment*, se perderán todos los datos relacionados con la IU que se almacene en el controlador.

Un *Fragment* representa una parte reutilizable de la IU de la app; define y administra su propio diseño, tiene su propio ciclo de vida y puede administrar sus propios eventos de entrada.

Una *Activity* es el punto de entrada de interacción con el usuario. Representa una pantalla individual con una interfaz de usuario.

La clase ViewModel permite que se conserven los datos después de cambios de configuración, como pueden ser las rotaciones de pantallas.

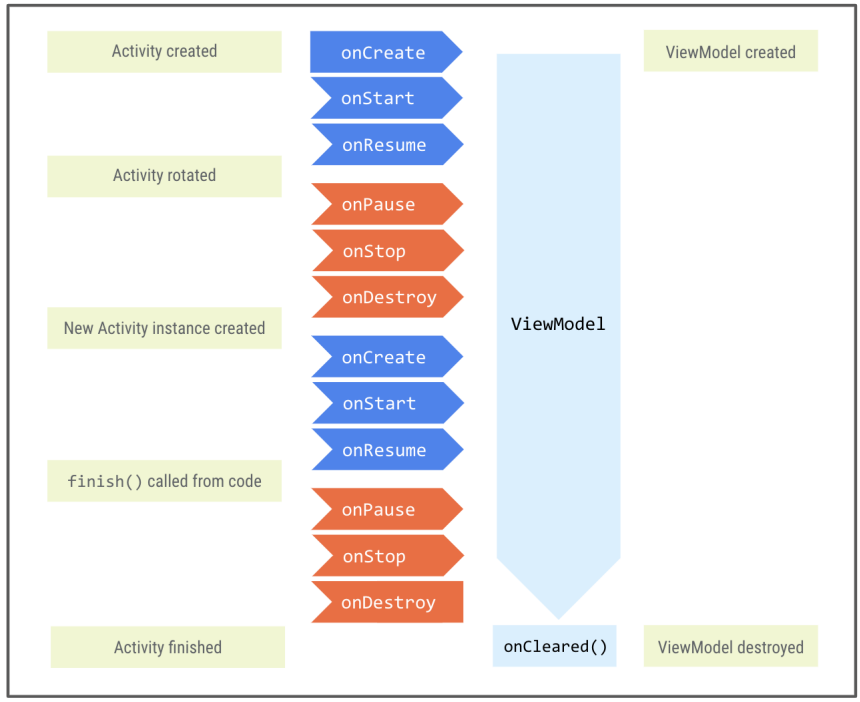

Figura 6.2: Ciclo de vida de un ViewModel. Extraído de [1].

Como se puede observar en la Figura 6.2, a pesar de que la actividad asociada al ViewModel sufre diversos cambios de configuración, este conserva siempre el mismo estado, hasta que es finalmente destruido junto a la actividad.

Para poder usar la clase ViewModel, basta con crear una que herede de ella de manera que se puedan crear los métodos necesarios en nuestro contexto, utilizando siempre las propiedades que ViewModel proporciona. Por lo general, las clases que lo utilizan vienen dadas por un prefijo asociado con la actividad o fragmento del que almacenarán información, y el sufijo -ViewModel. Un ejemplo para la pantalla `LoginScreen` sería `LoginViewModel`.

6.1.2. Principios de Diseño

Inversión de dependencias

Para el diseño del sistema se ha hecho uso del principio **Inversión de Dependencias**, dentro de los principios SOLID [4]. Este principio establece que un componente no debe depender de implementaciones, sino de abstracciones; es decir, interfaces. Con esto se logra que los módulos de alto nivel sean totalmente independientes de la implementación.

En el caso de esta aplicación, este principio se aplica a los módulos asociados de:

- **UI**, los casos de uso dependerán de una interfaz, de esta forma, si existe un cambio en la implementación del repositorio, el módulo UI junto con su lógica quedaría intacto, o bien si se decide añadir alguna implementación distinta en un futuro.
- **Datos**, para facilitar el cambio de implementación de los repositorios, por ejemplo, si queremos cambiar el cliente http, el módulo datos quedaría intacto.

6.1.3. Diagrama de Paquetes

En la Figura 6.3 se muestra el Diagrama de Paquetes a utilizar haciendo uso del **patrón *MVVM*** descrito anteriormente. En este caso los *ViewModel* se encuentran en el paquete UI junto con las Vistas. Los *ViewModel* son capaces de implementar su funcionalidad gracias a la lógica proporcionada por los paquetes *Data*, *Local* y *Remote*, y consiguen acceder a ella mediante la inyección de dependencias ubicada en *DI*.

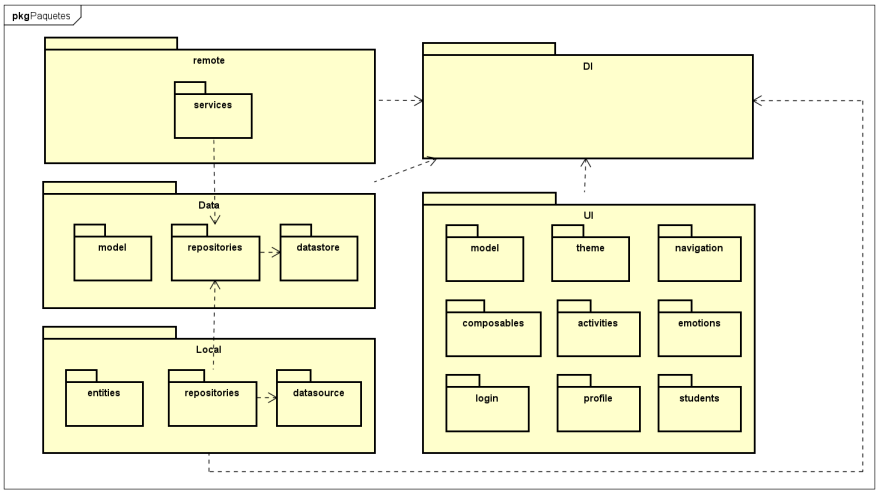

Figura 6.3: Diagrama de Paquetes.

6.2. Patrones de Diseño

6.2.1. Patrón DAO

El **patrón DAO (*Data Access Object*)** [2] u Objeto de Acceso a Datos, es un patrón de diseño que propone separar por completo la lógica de negocio de la lógica de acceder a datos. La clase **DAO** es la encargada de proporcionar una interfaz con todas las operaciones a realizar con la información proporcionada por la fuente de datos.

En la figura 6.4 se muestra la representación de este patrón

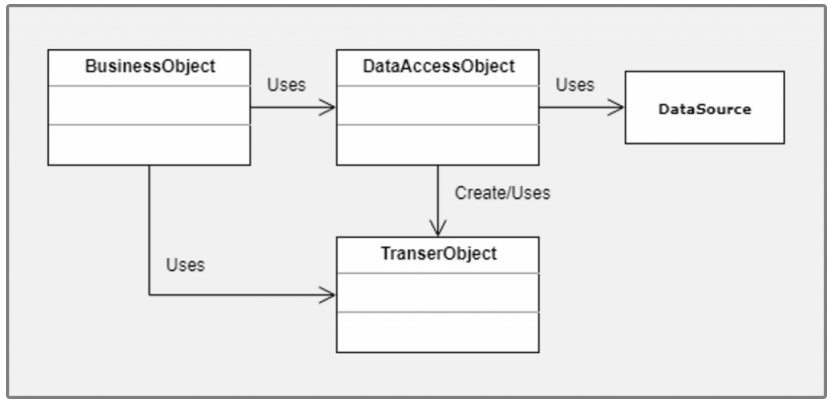

Figura 6.4: Patrón DAO. Extraído de [2].

Uso en el proyecto

En la aplicación, en el módulo asociado a la capa de local, existe un paquete llamado *datasource* que contiene las diferentes clases DAO utilizadas, las cuales cuentan con las diferentes operaciones a realizar en las entidades de la base de datos.

Estas clases son utilizadas por los diferentes repositorios, que a su vez son llamados por los **ViewModels**, los cuales acceden a la información de la base de datos, previamente adaptada a la capa correspondiente.

Esta adaptación se refiere a las clases del paquete **entities**. Una *entity* representa los tipos de objeto a almacenar en la base de datos local, con sus atributos e información. Como estos objetos pertenecen a la capa de **Local** y el **Dominio** no puede trabajar con ellos, se realiza una transformación a su objeto correspodiente en la capa Dominio.

6.2.2. Patrón Factory

El **patrón de diseño *Factory*** [3] permite crear objetos en una superclase, a la vez que permite a las subclases alterar el tipo de objetos que se crearán. Se utiliza cuando es necesario crear objetos sin tener que especificar su clase exacta.

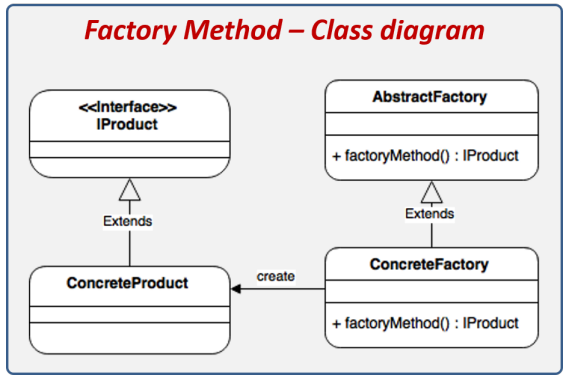

Figura 6.5: Patrón Factory. Extraído de [3].

Uso en el proyecto

En este proyecto el patrón se utiliza para la clase *User*. Esta clase proporciona los usuarios del sistema, que pueden ser de distintos tipos, como *Student* (Estudiante/Alumno) o *Teacher* (Profesor), según quién haya iniciado sesión. La creación de estas subclases con diferentes atributos como la emoción o el curso asociado se puede llevar a cabo gracias a este patrón. También se utiliza en la inyección de dependencias, cuando se instancia la Base de Datos local.

6.3. Arquitectura física del sistema

La arquitectura física del sistema representa los elementos o servicios que son necesarios para desplegar la aplicación. En este caso, los servicios backend utilizados por la aplicación son enteramente proporcionados por Firebase. A continuación se muestra el diagrama de despliegue de la aplicación en la Figura 6.6.

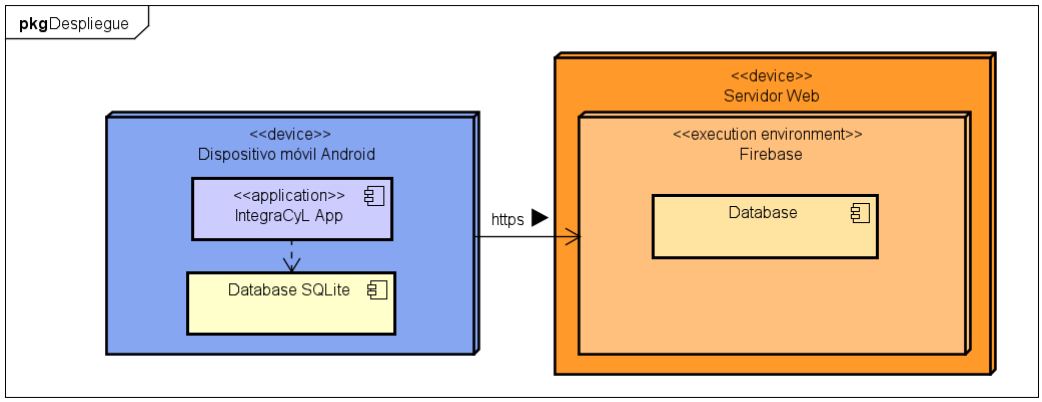

Figura 6.6: Arquitectura Física del Sistema.

6.4. Diseño de la interfaz gráfica

Para el Diseño de la Interfaz Gráfica, se han realizado *mocks* o prototipos de diseño de la interfaz de usuario (IU) de cada una de las pantallas que componen la aplicación. Estos bocetos tienen como finalidad:

- Ayudar en el desarrollo final de la interfaz, siendo una especie de guía a seguir. Es decir, es una base.
- Facilitar a los clientes o *stakeholders* de la aplicación una vista previa del planteamiento inicial de la aplicación, sujeta a posibles cambios.

Las siguientes figuras muestran los bocetos originales usados, diseñados con la herramienta Balsamiq.

- Login en la aplicación

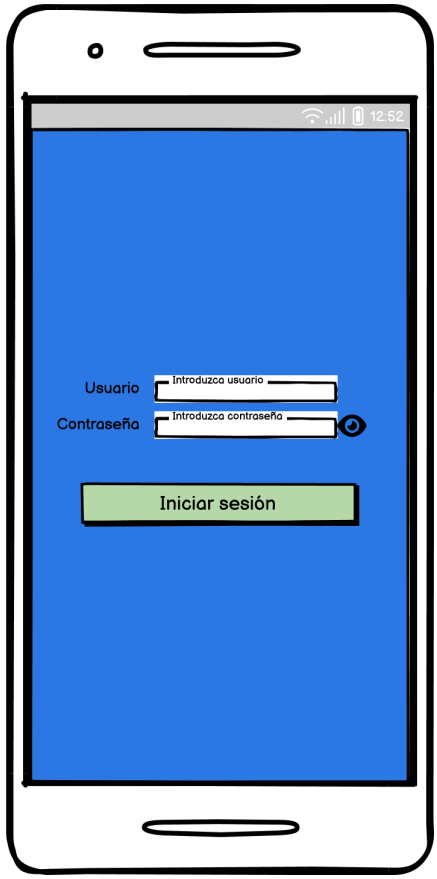

Figura 6.7: Boceto pantalla Login.

Pantalla que aparecerá siempre y cuando el usuario no haya iniciado sesión en el sistema. Aparecen dos **Campos de Texto** para introducir los credenciales, es decir el nombre de usuario y la contraseña, y un **Botón** para validar los datos y pasar al inicio de la aplicación.

- Perfil

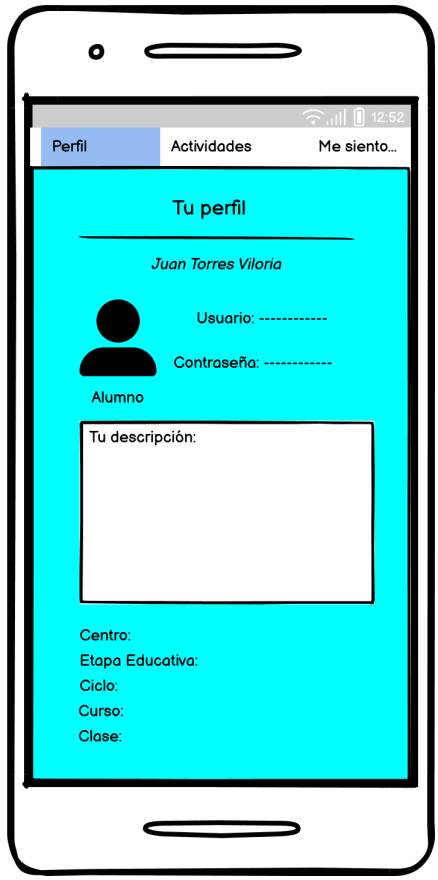

Figura 6.8: Boceto pantalla Perfil.

Esta pantalla será la inicial y aparecerá siempre que el usuario esté registrado e inciado en el sistema. Cuenta con dos **Campos de texto** para introducir el nombre de usuario y la contraseña, y un **Botón** para validar los credenciales e iniciar sesión.

- Listado de actividades

Figura 6.9: Boceto pantalla Actividades.

Se mostrará una vez iniciada la sesión como página principal, en ella aparece un **Texto** que muestra el título de la sección y distintos **Checkbox** que servirán como filtros para ordenar las Actividades. Ordenadas en formato **Cajón** o **Tarjeta** aparecen las Actividades, formadas por una **Imagen** asociada a la asignatura correspodiente, un **Texto** indicando el nombre de la asignatura y otro **Texto** a modo de pequeña descripción.

Figura 6.10: Boceto descripción de una Actividad.

Aparece una vez el usuario haya seleccionado una Actividad, se repite la **Imagen** de la asignatura junto con un **Texto** del nombre de la misma. El **Texto** que aparece esta vez es una descripción detalla de los pasos a seguir de la Actividad. Un **Botón** para iniciar la Actividad y otro **Botón** para completar el cuestionario asociado.

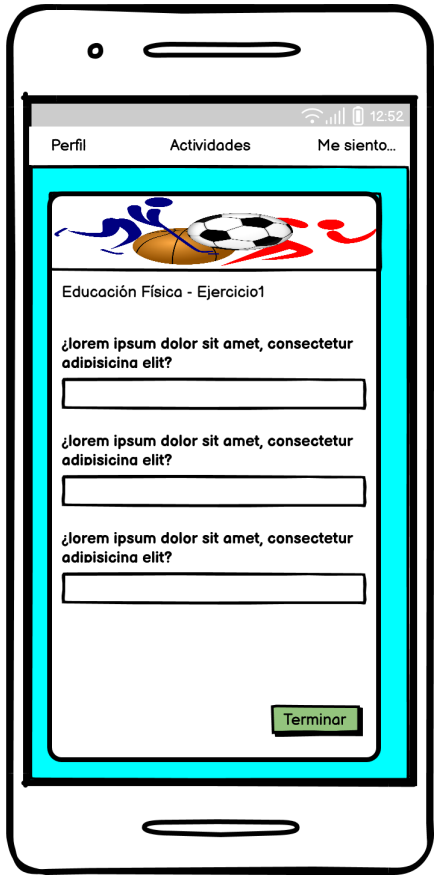

Figura 6.11: Boceto cuestionario de una Actividad.

Imagen y **Texto** con nombre de la asignatura. Para cada apartado del cuestionario existe un **Texto** con la pregunta y un **Campo de Texto** para introducir la respuesta. **Botón** para finalizar el cuestionario.

- Crear y asignar Actividades (Usuario Profesor)

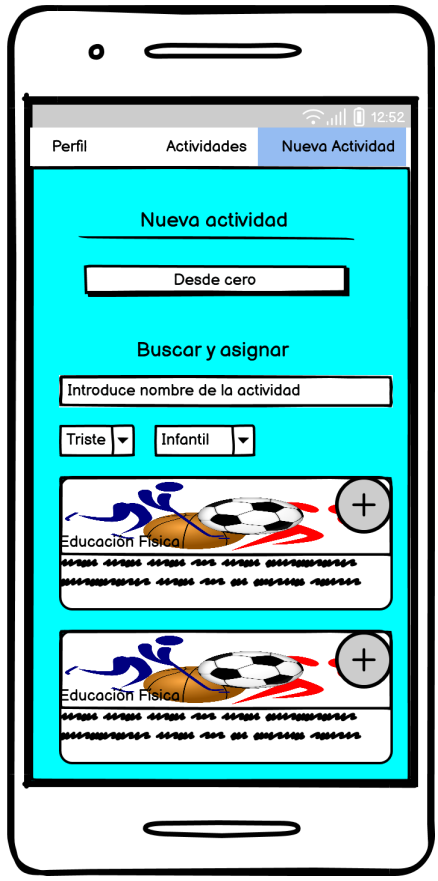

Figura 6.12: Boceto pantalla Crear y asignar Actividades.

Esta pantalla aparece solo en caso de ser un usuario Profesor. **Texto** con título de la sección y **Botón** para crear una Actividad nueva. La segunda mitad de la pantalla está enfocada en asignar Actividades a Alumnos, por lo que se indica con un **Texto**. Para localizar Actividades a asignar, contará con un **Campo de Texto** donde introducir el nombre de la Actividad y dos **ComboBox** para filtrar según Emociones o Cursos.

Finalmente aparece el listado de Actividades siguiendo el patrón visto en la Figura 6.9, añadiendo un **Botón con Icono (IconButton)** para asignar la actividad a un Alumno.

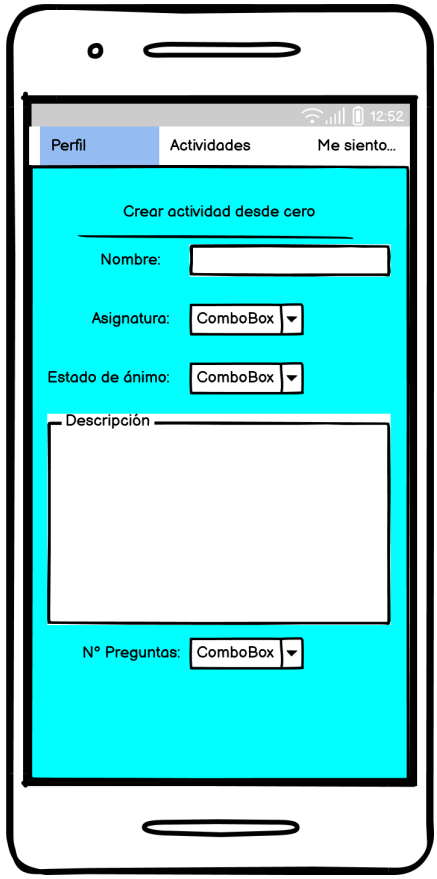

Figura 6.13: Boceto Crear Actividad desde cero.

Primeramente en la pantalla aparecerá el título de la sección con un **Texto**. Los campos a rellenar que tendrá el usuario Profesor son: **Campo de Texto** con el nombre de la Actividad, **ComboBox** en caso de querer seleccionar una asignatura relacionada y otro **ComboBox** para asociar a una Emoción, todos ellos acompañados de **Texto** indicando estas opciones. Un **Campo de Texto** a rellenar con la descripción detalla de la Actividad y un **ComboBox** para seleccionar cuantas preguntas quiere incluir en el cuestionario, siguiendo el formato visto en la Figura 6.11.

Figura 6.14: Boceto Asignar Actividad.

Cuando el usuario Profesor decide asignar una Actividad a un Alumno, se muestra un mensaje superpuesto en pantalla. En él aparecen un **Texto** indicando la opción que se está realizando, un **Texto** con el nombre de la Actividad seleccionada y un **Texto** con la Asignatura de la Actividad.

Un **Texto** para indicar que se debe elegir a un Alumno y un **Campo de Texto** para introducirlo, seguido de un **ComboBox** en caso de querer seleccionar al Alumno de este modo. Un **Texto** para indicar la fecha límite de la Actividad y dos **Botones**, en caso de cancelar o confirmar la decisión.

- Emociones (Usuario Alumno)

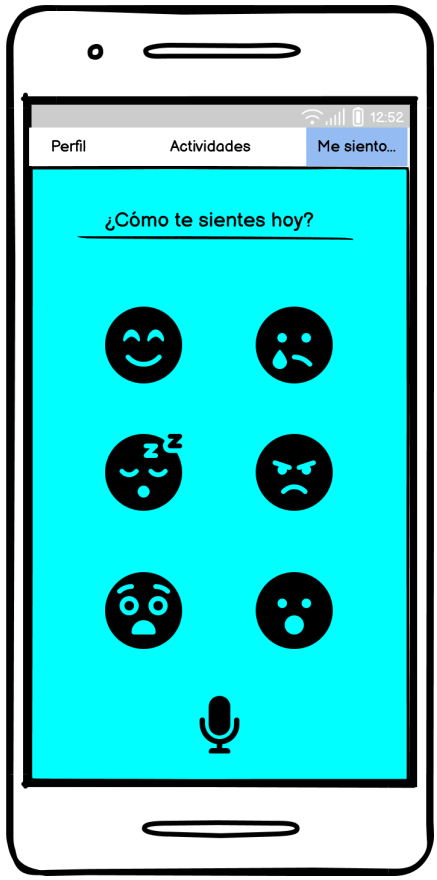

Figura 6.15: Boceto pantalla Emociones.

Un **Texto** indicando el título de la sección mostrada en pantalla y seis Botones con Icono para seleccionar la **Emoción**.

6.5. Diseño Base de Datos

6.5.1. Base de Datos remota

Para la realización de este proyecto, se ha decidido utilizar como herramienta y gestor de Base de Datos para desarrollar e implementar los servicios remotos y el apartado de persistencia a Firebase [10]. La singularidad aquí reside en que se pueden diseñar bases de datos de tipo no relacional, es decir, no usan SQL como lenguaje principal para las consultas y el almacenamiento de información no se realiza en tablas relacionadas.

Base de Datos NoSQL

Una base de datos **NoSQL** tiene como principal característica ser no relacional. Originalmente, SQL no se usó como API (*Application Programming Interfaces*) para acceder a los datos, sin embargo, la ubicuidad y utilidad de SQL hizo que muchas bases de datos **NoSQL** agregaran soporte para SQL. Hoy en día se acepta comúnmente que NoSQL significa **No Solo SQL** [11].

Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad). Entre las ventajas que nos ofrece este tipo de sistema de gestión de bases de datos, nos encontramos [12]:

- Manejar enormes cantidades de datos.
- No generan cuellos de botella.
- Escalamiento sencillo.
- Diferentes DBs NoSQL para diferentes proyectos.

6.5.2. Base de Datos local

Para poder trabajar con los datos a obtener por los servicios y que cualquier ViewModel tenga acceso a los datos siguiendo la arquitectura y no pasándoselos entre ellos de manera directa, se hace uso de una Base de Datos local. Cuando un dato remoto es obtenido en el proyecto, es almacenado directamente en local para poder ser usado en cualquier otro lugar del proyecto.

Para la implementación de la Base de Datos se usa *SQLite* con ayuda de la librería *Room*, es decir en este caso sí que será relacional.

A continuación, en la Figura 6.16 se muestra un Diagrama Relacional de la estructura de la base de datos empleada, basada en el modelo conceptual del Modelo de Dominio 5.1.

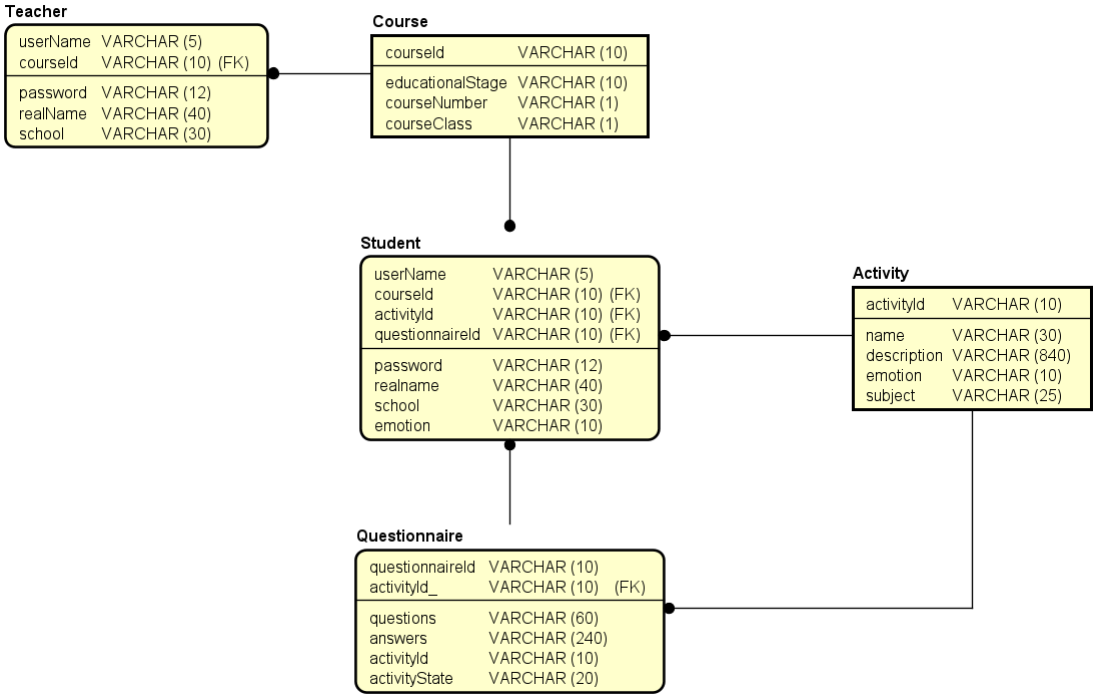

Figura 6.16: Diseño Relacional de la Base de Datos.

Capítulo 7

Tecnologías utilizadas

En este capítulo se presentan las tecnologías utilizas para la realización del proyecto en cada una de sus fases, desde la planificación, pasando por la implementación de la aplicación hasta la realización de este documento.

7.1. Planificación

Para la planificación del proyecto y sus respectivos esquemas y diagramas, las herramientas utilizadas son Microsoft Project, Microsoft Excel y Trello.

7.1.1. Microsoft Project

Figura 7.1: Logo Microsoft Project.

Microsoft Project [13] (o MSP) es un software de administración de proyectos desarrollado y vendido por Microsoft. Está diseñado para ayudar a un administrador de proyectos a desarrollar un cronograma, asignar recursos a las tareas, realizar un seguimiento del progreso, administrar el presupuesto y analizar las cargas de trabajo.

7.1.2. Microsoft Excel

Figura 7.2: Logo Microsoft Excel.

Microsoft Excel [14] se trata de un software que permite realizar tareas contables y financieras, entre otras, gracias a sus funciones, desarrolladas específicamente para ayudar a crear y trabajar con hojas de cálculo. Cuenta con cálculos, gráficas, tablas calculares y un lenguaje de programación macro llamado Visual Basic para aplicaciones.

7.1.3. Trello

Figura 7.3: Logo Trello.

Trello [15] es un software de administración de proyectos con interfaz web para organizar proyectos que implementa un tablero *Kanban*. Permite a los equipos gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas. En este proyecto ha servido como organización para clasificar las tareas y sus estados.

7.2. Análisis y Diseño

7.2.1. Astah

Figura 7.4: Logo Astah.

Astah es una herramienta de modelado UML que permite la realización de los diversos diagramas para la fase de Diseño y la fase de Análisis del proyecto. Además cuenta con la funcionalidad de generar los diagramas como imágenes en formato vectorial *.svg*, entre otras, lo que permite una mejor calidad.

7.3. Entorno de desarrollo

7.3.1. Android Studio

Figura 7.5: Logo Android Studio.

Android Studio [16] es un entorno de desarrollo integrado o IDE (*Integrated Development Environment*) para el desarrollo de aplicaciones para Android y está basado en IntelliJ IDEA. Android Studio ofrece funciones que aumentan la productividad a la hora de desarrollar aplicaciones para Android, como las siguientes:

- Un sistema de compilación flexible basado en Gradle.
- Un emulador rápido y cargado de funciones.
- Un entorno unificado donde poder desarrollar para todos los dispositivos Android.
- Ediciones en vivo para actualizar elementos componibles en emuladores y dispositivos físicos en tiempo real.
- Integración con GitHub y plantillas de código para ayudar a compilar funciones de apps comunes e importar código de muestra.
- Variedad de marcos de trabajo y herramientas de prueba.

7.4. Lenguaje de Programación

7.4.1. Kotlin

Figura 7.6: Logo Kotlin.

Kotlin es un lenguaje de programación de código abierto creado por JetBrains que se ha popularizado gracias a que se puede utilizar para programar aplicaciones Android [17]. Es de tipado estático y corre sobre la máquina virtual de Java, y también puede ser compilado a código fuente de JavaScript.

Entre algunas de sus principales características, se encuentran [18]:

- **Expresivo y conciso**: se centra en la expresión de ideas y reducir código duplicado.
- **Código más seguro**: incluyendo los tipados *@Nullable* y *@NonNullable* ayuda a evitar *NullPointerExceptions*: las aplicaciones Android que usan **Kotlin** tienen un 20 % menos de probabilidad de *crashear*, es decir fallar en tiempo de ejecución.
- **Interoperabilidad**: **Kotlin** es 100 % interoperable con Java.
- **Concurrencia estructurada**: las corrutinas de Kotlin agilizan la programación asíncrona, haciendo que las tareas comunes como llamadas de red y actualizaciones de bases de datos sean simples y eficientes.

7.4.2. Jetpack Compose

Figura 7.7: Logo Jetpack Compose.

Jetpack Compose [9] es el kit de herramientas moderno de Android para compilar IU nativas. Simplifica y acelera el desarrollo de la IU en Android.

Sus principales características son:

- **Menos código**: realiza más tareas con menos código y evita clases internas de errores a fin de que el código sea simple y fácil de mantener.
- **Intuituvo**: solo es necesario describir la IU. A medida que cambie el estado de la app, la IU se actualizará automáticamente.
- **Acelera el desarrollo**: es compatible con todo el código existente para que se pueda adoptar cuandoy donde se desee. Permite iterar rápidamente con vistas previas en vivo y goza de una compatibilidad completa con Android Studio.
- **Potente**: permite crear apps atractivas con acceso directo a las API de la plataforma de Android y tiene compatibilidad integrada para Material Design [19], tema oscuro [20], animaciones [21] y mucho más.

Jetpack Compose destaca por hacer uso de unas funciones llamadas *Composables*. Estas funciones permiten diseñar interfaces de manera programática, definiendo su diseño. Cada una de estas funciones cuentan con un estado, por lo que cuando existan cambios de estado, se recompone esa función (el elemento *Composable*) y no toda la interfaz.

7.5. Control de versiones

7.5.1. Github

Figura 7.8: Logo GitHub.

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git [22].

GitHub ofrece a los desarrolladores la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura, facilitando la organización de proyectos y permitiendo la colaboración de varios desarrolladores en tiempo real [23].

7.5.2. SourceTree

Figura 7.9: Logo SourceTree.

SourceTree es un software que permite la visualización y administración de repositorios basados en Git mediante una interfaz sencilla e intuitiva [24].

Representa los cambios realizados en los diferentes ficheros, ya sea en una rama (branch) o en un *commit* específico, y facilita la ejecución de las tareas más básicas del manejo de ramas, como *push*, *pull* o *commit*. De igual manera, dispone de un terminal sencillo para poder seguir realizando estas funciones a mano. Todo esto se puede observar en la Figura 7.10

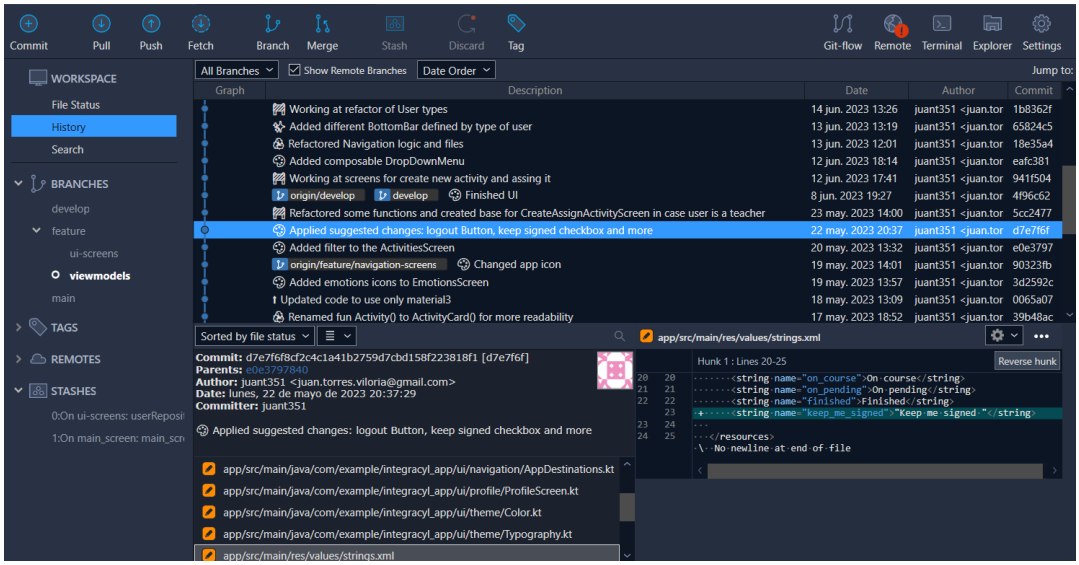

Figura 7.10: SourceTree.

7.5.3. Gitmoji

Figura 7.11: Logo Gitmoji.

Gitmoji [25] es una guía de emoticonos para los mensajes *commit* en GitHub.

El uso de emoticonos en los mensajes *commit* proporciona una manera fácil de identificar el propósito o la intención del mensaje con solo mirar los emoticonos utilizados [26]. Sin necesidad de leer el código, ayuda a entender y clasificar los *commit*, con un vistazo rápido se puede saber a qué parte del código está afectando. En la Figura 7.12 se puede observar la página principal con la guía de emoticonos:

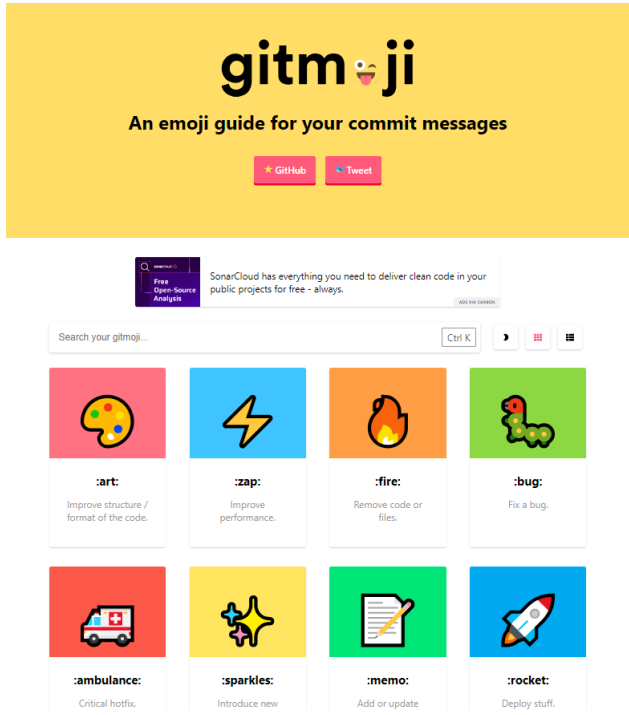

Figura 7.12: Gitmoji.

7.6. Documento de Memoria

7.6.1. Overleaf

Figura 7.13: Logo Overleaf.

Overleaf es una herramienta de publicación y redacción colaborativa en línea de documentos LaTeX que hace que todo el proceso de redacción, edición y publicación de documentos científicos sea mucho más rápido y sencillo. **Overleaf** brinda la conveniencia de un editor LaTeX fácil de usar con colaboración en tiempo real y la salida totalmente compilada producida automáticamente en segundo plano a medida que se escribe.

Capítulo 8

Implementación

Este capítulo recoge el proceso de codificación empleado en el proyecto.

8.1. Formación previa

Para este proyecto, la gran mayoría de tecnologías utilizadas son prácticamente nuevas para el equipo de desarrollo. Este apartado es vital para un desarrollo seguro y estable del proyecto, por lo que se ha dedicado gran tiempo a adquirir los conocimientos necesarios.

La estrategia empleada para la formación ha sido la realización de varios cursos gratuitos de Internet para agilizar el aprendizaje.

8.1.1. Kotlin

Para la formación de este moderno lenguaje de programación, dominante en aplicaciones Android se ha optado por:

- ***Codelabs***: cursos cortos para establecer los conceptos básicos y poder programar en Android. De manera progresiva se presentan las nuevas herramientas y las similitudes con otros lenguajes, hasta llegar a los conceptos de más grosor, como pueden ser las funciones de extensión, flujos *LiveData* y las corrutinas. Los cursos realizados han sido:
  - **Android Basics in Kotlin** [27]
  - **Kotlin Bootcamp for Programmers** [28]

  Al final de cada *Codelab* se propone realizar un ejercicio sin solución para poder probar lo aprendido y manejarse con el lenguaje de manera autosuficiente.
- **Kotlin Koans** [29]: serie de ejercicios para familiarizarse con la sintaxis de Kotlin.

8.1.2. Jetpack Compose

Jetpack Compose es el conjunto de librerías a utilizar para el diseño de la interfaz gráfica de la aplicación.

De igual manera que para la formación en el lenguaje de Kotlin, para Jetpack Compose se han realizado varios cursos donde se ha podido aprender los conceptos básicos y fundamentales para desarrollar una aplicación Android. Los cursos han sido:

- **Aspectos básicos de Android con Compose** [30]: introducción a Jetpack Compose y realización de pequeñas aplicaciones para practicar y asentar conocimientos.
- **Jetpack Compose para desarrolladores de Android** [31]: realización de aplicaciones más consistentes gracias al uso de herramientas más potentes.

8.2. Configuración inicial del proyecto

Previamente a la implementación del proyecto es necesario la creación del mismo y la configuración del ***Gradle***. *Gradle* [32] es una herramienta de compilación avanzada de código abierto que permite la automatización de la compilación del proyecto.

Inicialmente se seleccionan las librerías y herramientas que se usarán a lo largo del proyecto, y a medida que se avance se podrán ir añadiendo más. *Gradle* facilita mucho todo este proceso, ya que simplemente especificando una versión válida y compatible de una librería con nuestro proyecto, se encarga de todo el proceso de compilación y de subtareas.

8.3. Estructura del código

Para la implementación del código de la aplicación se ha seguido la estructura básica que define *Android Studio* al crear un proyecto de cero, aplicando las pertinentes modificaciones para adaptarse al **patrón arquitectónico MVVM**.

Teniendo en cuenta paquetes principales que se definen por defecto como son *.idea*, *gradle* o *build*, en este caso se hablará del paquete ***app***. Dentro de este paquete se encuentran *src*, el cual está formado por *androidTest*, ***main*** y *test[unitTest]*. En *unitTest* se guardan las clases usadas para realizar pruebas unitarias, como por ejemplo las realizadas para comprobar el correcto funcionamiento de la inyección de dependencias.

En ***main*** se encuentran *res* y *java*. El primero sirve para almacenar los diferentes recursos utilizados en la aplicación, como los *String*, colores de la aplicación, fuentes de texto o imágenes e iconos. En *java* se encuentra el código funcional de la aplicación separado en paquetes, los cuales son: *data*, *di*, *local*, *remote* y *ui*.

En *data* se encuentra el código relacionado con el tratamiento de datos, los cuales se obtienen bien de *remote* en caso de ser datos externos o de *local* si se usa la Base de Datos local. En *remote* se encuentran los servicios empleados para la funcionalidad, como la obtención de Usuarios, Actividades y otros datos guardados en *Firebase*. El paquete *ui* contiene lo relacionado con la interfaz, siendo esto la parte visual diseñada con *Jetpack Compose* y los *viewModels* que gestionan la lógica de la parte visual con ayuda del resto de paquetes, los cuales tiene acceso gracias a *di*, quien se encarga de la inyección de dependencias.

8.4. Diseño de interfaz con Jetpack Compose

En esta sección se muestran los diseños iniciales de la UI presentada en el Capítulo de Diseño 6 adaptados a los requisitos finales e implementado con **Jetpack Compose**.

- **Pantalla de Iniciar Sesión**

  Esta pantalla se muestra en la Figura 8.1.

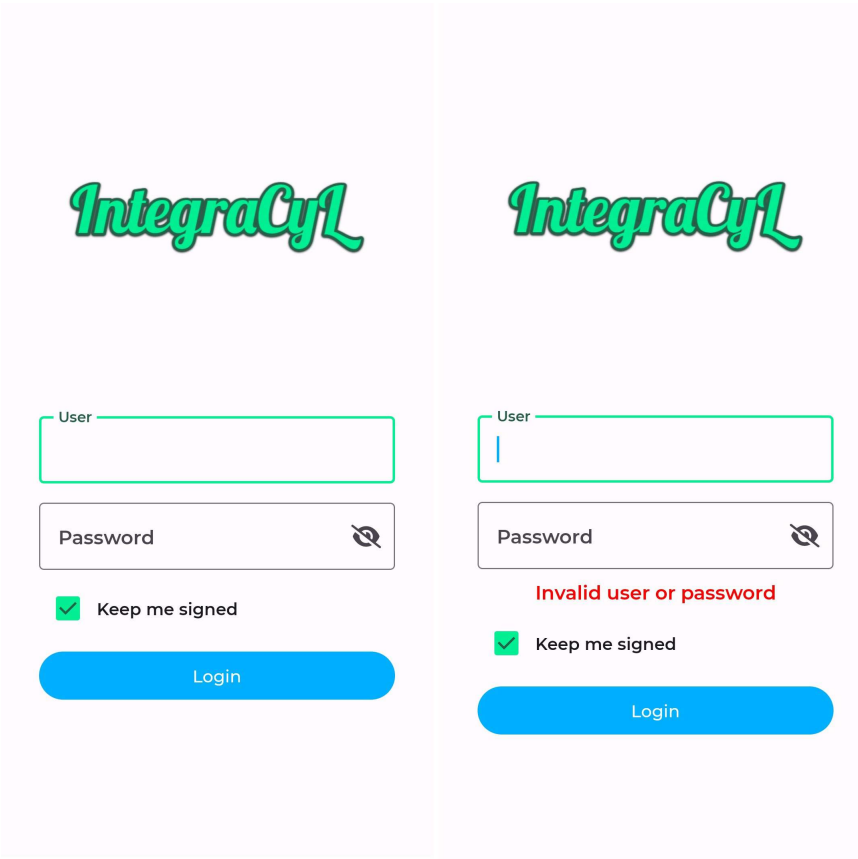

Figura 8.1: Iniciar Sesión.

- **Pantalla Perfil**

  Esta pantalla se muestra en la Figura 8.2.

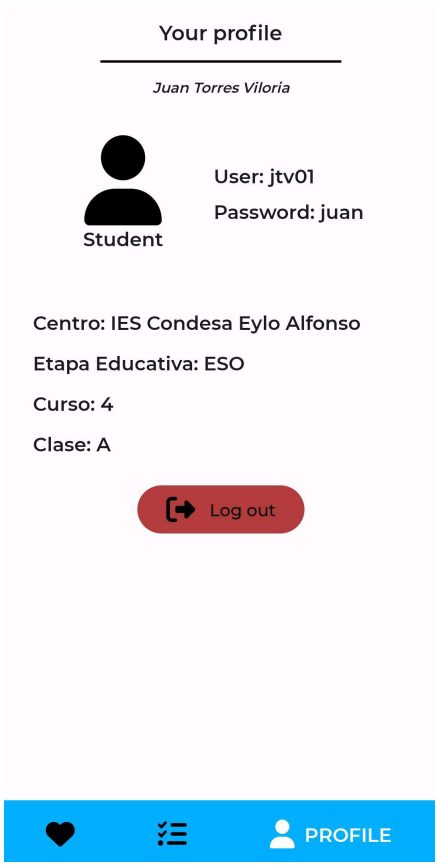

Figura 8.2: Perfil.

- **Pantalla Listado de Actividades**

  Esta pantalla para el Alumno se muestra en la Figura 8.3 y en la Figura 8.4. Para el usuario Profesor se muestra en la Figura 8.5.

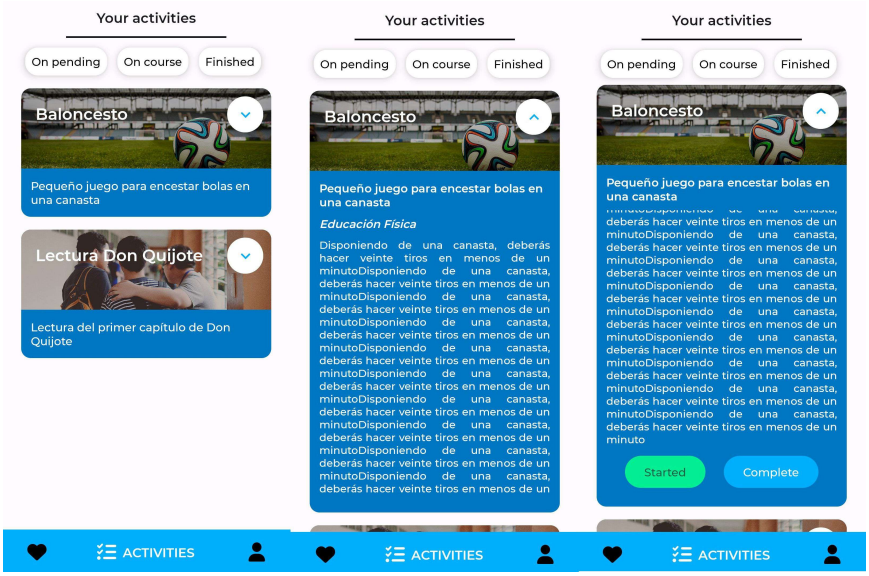

Figura 8.3: Listado de Actividades para Alumno.

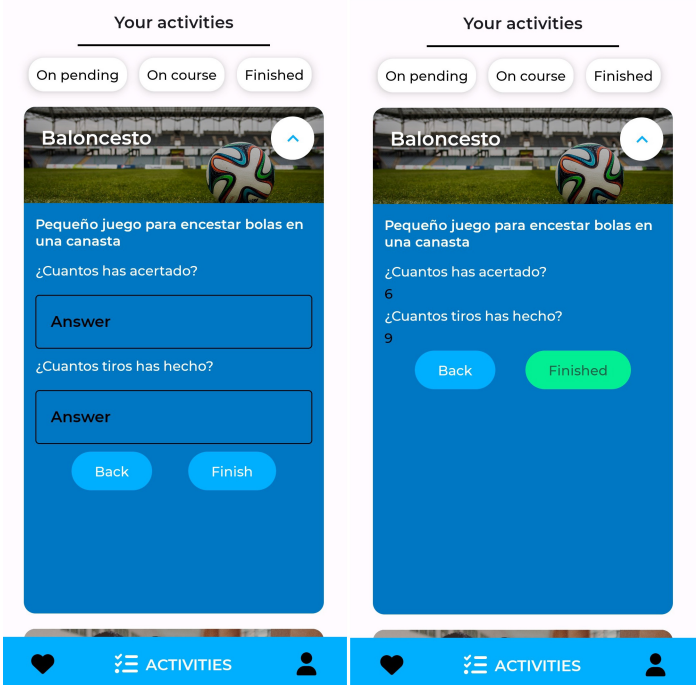

Figura 8.4: Completar Actividad para Alumno.

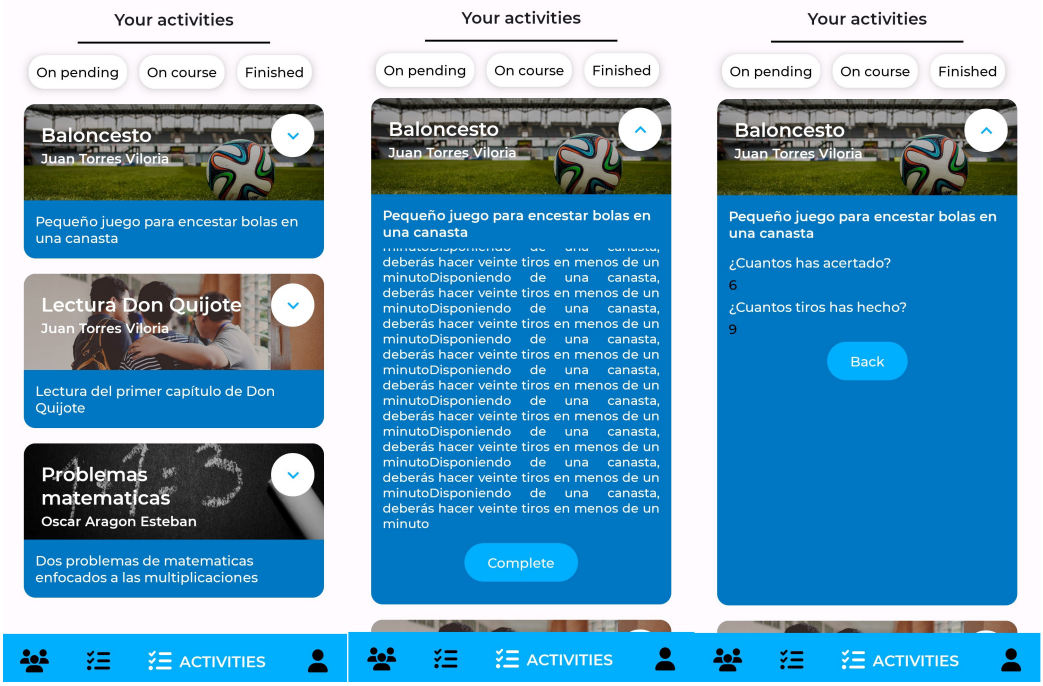

Figura 8.5: Listado de Actividades para Profesor.

- **Pantalla Emociones**

  Esta pantalla se muestra en la Figura 8.6.

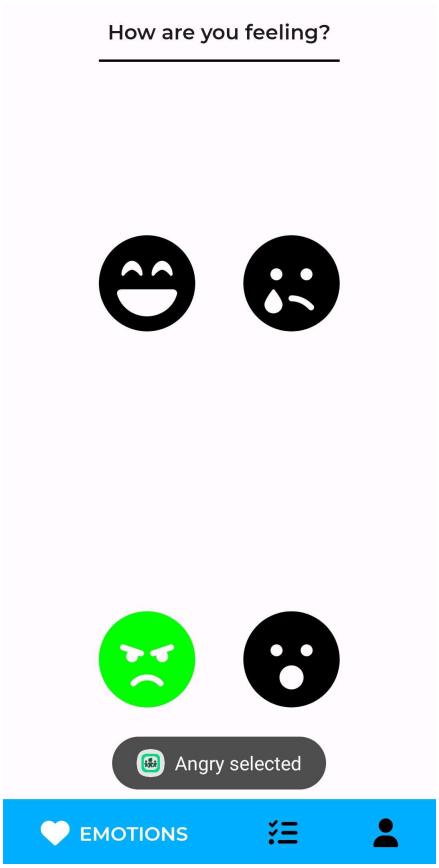

Figura 8.6: Listado de Emociones para Alumno.

- **Pantalla Estudiantes**

  Esta pantalla se muestra en la Figura 8.7.

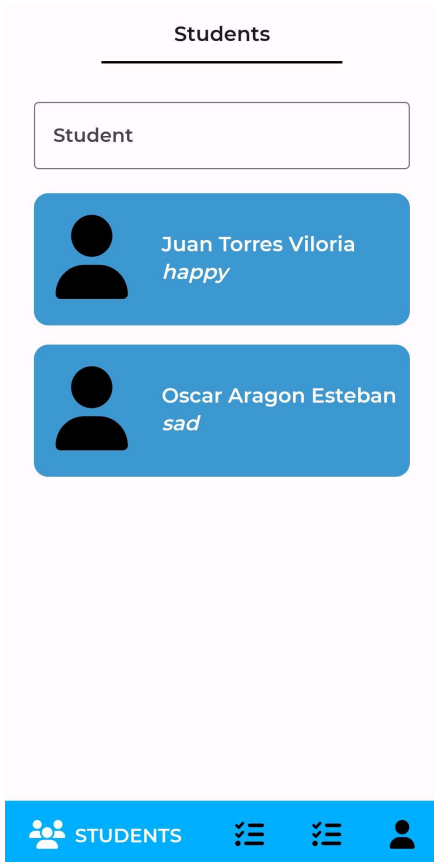

Figura 8.7: Listado de Alumnos.

- **Pantalla Creación y Asignación de Actividades**

  La pantalla para asignar una Actividad a un Alumno se muestra en la Figura 8.8 y para crear una Actividad desde cero se muestra en la Figura 8.9.

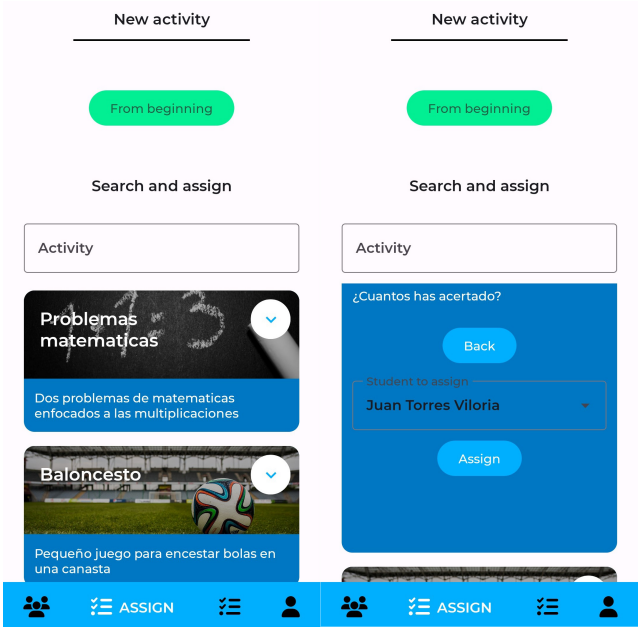

Figura 8.8: Asignar Actividad a Alumno.

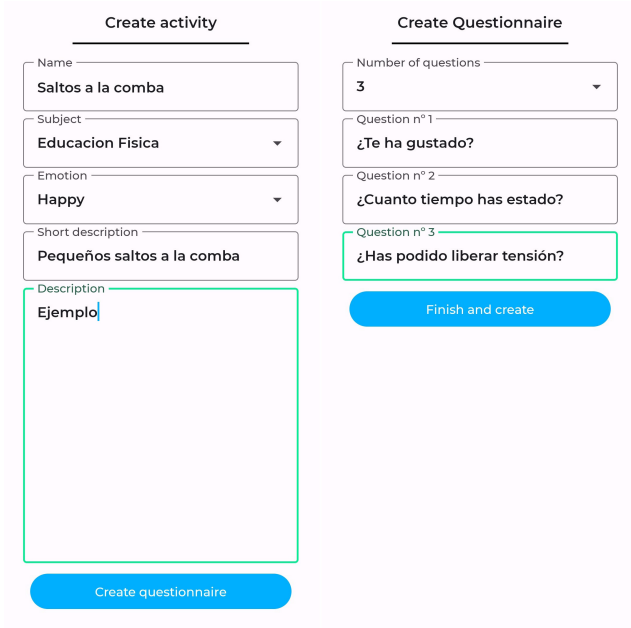

Figura 8.9: Crear nueva Actividad.

8.5. Shared Preferences

Uno de los recursos que más se utiliza a lo largo de todo el proyecto en su implementación, son las *Shared Preferences*. Las *Shared Preferences* de Android son un mecanismo de almacenamiento ligero y persistente ampliamente utilizadas en el desarrollo de aplicaciones móviles para almacenar y recuperar datos clave-valor de manera eficiente. Estas preferencias se guardan en un archivo XML en el almacenamiento interno de la aplicación y pueden ser accedidas en cualquier momento durante la ejecución de la aplicación [33].

En el contexto de este proyecto, las *Shared Preferences* han sido utilizadas como una opción de almacenamiento para la configuración de la aplicación. Esto ha permitido almacenar y recuperar de manera eficiente las preferencias del usuario, como el tema de la interfaz de usuario, la opción de mantener sesión iniciada y otras preferencias personalizadas.

8.6. Transformaciones entre capas

Como se ha comentado previamente, la arquitectura de la aplicación consta de diferentes capas que interactuan entre ellas. Para que esta comunicación sea posible y, por ejemplo, un objeto de la capa de datos remota se pueda almacenar en la capa de datos local (de Firebase a Room), existe la clase *Mapper.kt*

La clase *Mapper.kt* se utiliza para realizar transformaciones entre clases de diferentes capas en el código. Su propósito principal es convertir objetos de una capa de datos a objetos de otra capa, como el modelo, los datos y la capa local. Estas transformaciones permiten la comunicación y el intercambio de información entre las diferentes capas de la aplicación de manera eficiente.

La clase *Mapper.kt* incluye una serie de funciones que implementan estas transformaciones. Cada función está diseñada para realizar una transformación específica entre dos clases. A continuación, se presentan algunos ejemplos de transformaciones implementadas en la clase *Mapper.kt*, para los objetos relacionados con el Alumno a lo largo de las diferentes capas:

```
1  fun DataStudent.toStudent(): Student {
2      val activitiesAndQuestionnairesId: HashMap<Int, Int> = hashMapOf()
3      for (i in 0 until this.activitiesId.size) {
4          activitiesAndQuestionnairesId[this.activitiesId[i]] =
           ↪  this.questionnairesId.getOrElse(i) { 0 }
5      }
6      return Student(
7          userName = this.userName,
8          password = this.password,
9          realName = this.realName,
10         school = this.school,
11         activitiesAndQuestionnairesId = activitiesAndQuestionnairesId,
12         course = this.course.toCourse(),
13         emotion = this.emotion.toEmotion()
14     )
```

```
}

fun Student.toDataStudent(): DataStudent {
    val activitiesId: MutableList<Int> = mutableListOf()
    val questionnairesId: MutableList<Int> = mutableListOf()

    for ((key, value) in activitiesAndQuestionnairesId) {
        activitiesId.add(key)
        questionnairesId.add(value)
    }

    return DataStudent(
        userName = this.userName,
        password = this.password,
        realName = this.realName,
        school = this.school,
        activitiesId = activitiesId,
        questionnairesId = questionnairesId,
        course = this.course.toDataCourse(),
        emotion = this.emotion.toEmotionString()
    )
}
```

Transformaciones para comunicar el objeto Alumno entre el dominio y la capa de datos remota.

```
    fun StudentEntity.toDataStudent() =
    DataStudent(
        userName = userName,
        password = password,
        realName = realName,
        school = school,
        activitiesId = activitiesId.toListId(),
        questionnairesId = questionnairesId.toListId(),
        course = course.toDataCourse(),
        emotion = emotion
    )

    fun DataStudent.toStudentEntity() =
    StudentEntity(
        userName = userName,
        password = password,
        realName = realName,
        school = school,
        activitiesId = activitiesId.toListIdString(),
        questionnairesId = questionnairesId.toListIdString(),
        course = course.toCourseString(),
        emotion = emotion
    )
```

Transformaciones para comunicar el objeto Alumno entre la capa de datos remota y la capa de datos local.

8.7. Inicio de Sesión

Para la implementación del inicio de sesión se han tenido en cuenta dos puntos importantes en base a sistemas de este estilo, como puede ser el Campus Virtual de la UVa [34] o la página web de la Escuela de Ingeniería Informática [35].

El primero de ellos es que al ser un servicio académico y con funciones meramente lectivas, no existe la opción de crear un usuario de cero, es algo que no está al alcance de los usuarios recurrentes. Todos los usuarios deben ser previamente introducidos al sistema para que este les pueda reconocer y dar validez.

El segundo punto es referido a los credenciales de inicio de sesión. Al igual que los sistemas comentados previamente, los nombres de usuario siguen todos un modelo específico y la contraseña es una asignada por defecto. Los nombres de usuario vienen dados por la concatenación de las iniciales del nombre y los dos primeros apellidos del usuario junto con los dos últimos dígitos de su año de nacimiento. Por ejemplo, para el usuario Juan Torres Viloria nacido en 2001, su nombre de usuario es jtv01. En caso de coincidencia, se añadiría la segunda letra del nombre al último usuario registrado. El usuario Jaime Torres Velasco, nacido también en 2001 será jatv01. Este proceso se aplica en función de los parámetros que coincidan, como último ejemplo, el usuario Jaime Trigeros Velasco del 2001 será jtrv01.

Para la implementación existe una clase *UserService* que se encarga de comprobar los datos introducidos por el usuario haciendo uso de la Base de Datos remota *Firebase*. En caso de coincidencia, se obtienen la información del usuario en formato *DataUser*, un objeto para almacenar los datos remotos de un usuario y se realiza una tranformación a *UserEntity*, objeto diseñado para guardar la información del usuario en Base de Datos local y poder acceder a su información desde cualquier ventana de la aplicación.

El formato del DataUser es:

```
sealed interface DataUser {
    var userName: String
    var password: String
    var realName: String
    var school: String
}

data class DataStudent(
    override var userName: String = "",
    override var password: String = "",
    override var realName: String = "",
    override var school: String = "",
    var activitiesId: MutableList<Int> = mutableListOf(),
    var questionnairesId: MutableList<Int> = mutableListOf(),
    var course: DataCourse = DataCourse(),
    var emotion: String = ""
) : DataUser

data class DataTeacher(
    override var userName: String = "",
```

```
    override var password: String = "",
    override var realName: String = "",
    override var school: String = "",
    var courses: Set<DataCourse> = mutableSetOf(),
) : DataUser
```

Finalmente se comprueba si el usuario ha decidido mantener la sesión iniciada, guardando esta decisión en *SharedPreferences* junto con el nombre de usuario. Por último se carga la ventana principal de la aplicación y sus diferentes rutas según el usuario.

8.8. Vista del Perfil

Cuando un usuario desea acceder a la ventana de Perfil y obtener información sobre sus datos, internamente se accede a la Base de Datos local y se obtiene dicha información.

Para ello, primero es necesario saber de qué usuario hay que obtener la información, por lo que se consulta la variable `userName` guardada en *SharedPreferences*, que se había obtenido previamente en el inicio de sesión. A partir de ese momento, se busca en la base de datos local el usuario que coincida con ese nombre y se obtiene su *UserEntity*.

```
    val sharedPreferences : SharedPreferences = koinInject()
    val userName = sharedPreferences.getString("userName", "")

    LaunchedEffect(true) {
        userName?.let { profileViewModel.setUserName(it) }
        profileViewModel.onCheckProfile()
    }
```

Para mostrar los datos por pantalla se transforma *UserEntity* en *User* y de este último objeto se despliega la información de sus diferentes campos.

```
    val student: Student? = localStudentDatastore.getStudentByUserName(userState
    ↪  .value.userName)?.toStudent()
    val teacher: Teacher? = localTeacherDatastore.getTeacherByUserName(userState
    ↪  .value.userName)?.toTeacher()
```

8.9. Vista de Estudiantes

Cuando el usuario es un Profesor y se carga la vista de Estudiantes, se realiza una búsqueda de Alumnos/Estudiantes asociados a ese Profesor.

Para ello, primero es necesario obtener los datos del Profesor, entre los que figuran los diferentes cursos a los que puede estar asignado. Se repite el proceso de obtener su nombre mediante *SharedPreferences*:

```
    val sharedPreferences: SharedPreferences = koinInject()
    val userName = sharedPreferences.getString("userName", "")

    LaunchedEffect(true) {
        userName?.let { studentsScreenViewModel.setTeacherName(it) }
        studentsScreenViewModel.onCheckStudents()
    }
```

Con los datos del Profesor, se obtiene un listado de Estudiantes que coincidan en Centro Educativo, Curso y alguna Asignatura con el profesor. Para obtener los estudiantes asociados a un curso se vuelve a utilizar la clase *UserService* y se forma un listado de los estudiantes que forman los diferentes cursos.

```
    // Get all the students that match the teacher courses and subjects
            teacher?.courses?.forEach { course ->
                val studentsTemp = userRepository.studentsByCourse(teacher.school,
                ↪  course)
                students += studentsTemp
                _studentsState.update {
                    it.copy(
                        students = students
                    )
                }
            }
```

Finalmente, para cada Estudiante se muestra utilizando la API (*Application Programming Interface*) *CardView* de Android su nombre real completo y su estado de ánimo obtenidos del objeto *Student*.

8.10. Consultar Actividades

En caso de querer consultar sus Actividades y cargarse la pantalla respectiva, el proceso es similar a lo visto hasta ahora. Primero se obtiene el nombre de usuario del que se desea obtener la información a partir de *SharedPreferences*.

En caso de ser Alumno, mediante el atributo *activitiesAndQuestionnairesId*, que es un *HashMap* de identificadores Actividad-Cuestionario, se buscan esas actividades una a una con su identificador mediante la clase *UserService*. Además las Actividades obtenidas junto con los Cuestionarios asociados se guardan en Base de Datos local mediante el uso de *activitiesDatastore* y *questionnaireDatastore*.

```
    if (student != null) {
            val activities: MutableList<IntegraCyLActivity> = mutableListOf()
            val questionnaires: MutableList<Questionnaire> = mutableListOf()
            // For each Clave-Valor actividad-cuestionario del alumno
            student.activitiesAndQuestionnairesId.forEach { activityAndQuestionnaireId
            ↪  ->
```

```
                val activityTemp =
                    userRepository.getActivityById(activityAndQuestionnaireId.key)
                val questionnaireTemp =
                    userRepository.getQuestionnaireById(activityAndQuestionnaireId
                    ↪ .value)
                // Se añade la actividad en local
                activityTemp?.let {
                    activities.add(it)
                    activitiesDatastore.insertActivity(it.toDataIntegraCyLActivity())
                }
                // Se añade el cuestionario en local
                questionnaireTemp?.let {
                    questionnaires.add(it)
                    questionnaireDatastore.insertQuestionnaire(it
                    ↪ .toDataQuestionnaire())
                }
            }
            // ...
```

Para un Profesor, primero se seleccionan sus Alumnos asignados (mismo proceso que en la sección anterior) y para cada uno de ellos se cargan sus Actividades.

```
else if (teacher != null) {
                val students: MutableList<Student> = mutableListOf()
                val studentsWithActivities: HashMap<Student, IntegraCyLActivity> =
                ↪ hashMapOf()
                // Busqueda de alumnos asignados al profesor.
                teacher.courses.forEach { course ->
                    val studentsTemp = userRepository.studentsByCourse(teacher.school,
                    ↪ course)
                    students += studentsTemp
                }
                val activities: MutableList<IntegraCyLActivity> = mutableListOf()
                val questionnaires: MutableList<Questionnaire> = mutableListOf()
                // Para cada alumno asignado al profesor
                students.forEach { student ->
                    // Para cada par clave-valor actividad-cuestionario
                    student.activitiesAndQuestionnairesId.forEach {
                    ↪ activityAndQuestionnaireId ->
                        val activityTemp =
                            userRepository.getActivityById(activityAndQuestionnaireId
                            ↪ .key)
                        val questionnaireTemp =
                            userRepository
                            ↪ .getQuestionnaireById(activityAndQuestionnaireId.value)

                        activityTemp?.let {
                            activities.add(it)
                            activitiesDatastore.insertActivity(it
                            ↪ .toDataIntegraCyLActivity())
                            studentsWithActivities.put(student, it)
                        }
                        questionnaireTemp?.let {
                            questionnaires.add(it)
```

```
                            questionnaireDatastore.insertQuestionnaire(it
                            ↪  .toDataQuestionnaire())
                        }
                    }
                }
                // ...
```

Una vez se tiene la información de cada Actividad, de nuevo se recurre a la API *CardView* para mostrar su contenido en pantalla. En caso de usuario Profesor, el *CardView* mostrará además a qué Alumno pertenece esa Actividad.

8.11. Crear Actividad

Cuando un Profesor decide crear una Actividad desde cero, se realiza una navegación a una pantalla con un formulario para rellenar los datos de la Actividad.

Cuando se pulsa el botón para *Crear Cuestionario* y pasar a la siguiente pantalla, se llama a una función del *ViewModel* que realiza unas comprobaciones y genera la Actividad.

```
    Button(
            onClick = {
                createAssignActivityViewModel.createActivity(
                    activityName,
                    activityShortDescription,
                    activityDescription,
                    if(selectedEmotion== -1) "" else emotions[selectedEmotion],
                    if(selectedSubject == -1) "" else subjects[selectedSubject],
                    createQuestionnaire
                )
            },
            //...
```

Esta función lo primero que hace es comprobar si los datos introducidos son correctos. Una Actividad es opcional que tenga una Emoción o Asignatura asignada, por lo que se comprueba si se ha seleccionado alguna de las disponibles y en caso negativo se rellena con la cadena vacía.

Como en los requisitos no se especificó un nivel de seguridad o comprobación para entradas de usuario, simplemente se comprueba que los tres campos obligatorios (nombre de la Actividad, pequeña descripción y descripción detallada) no estén vacíos.

Si alguno de esos campos está vacío, se actualiza el estado del ViewModel marcando que ha habido un error en la creación de la Actividad, por lo que en pantalla se activa un cuadro de texto informando del error.

```
    _assignCreateState.update {
            it.copy(
                activityError = true,
            )
    }
```

```
    if(createAssignState.activityError){
        item {
            Text(
                text = "Invalid activity",
                color = Color.Red,
                modifier = Modifier.fillMaxWidth().align(Alignment.CenterHorizontally)
            )
        }
    }
```

En caso de que todos los campos sean correctos, se crea una nueva Actividad a partir de los datos introducidos y se guarda en la Base de Datos local y en remoto Firebase. Acto seguido se navega a la pantalla para crear el Cuestionario de la Actividad.

```
    val newActivity = IntegraCyLActivity(
                id = newId,
                name = typedName,
                shortDescription = typedShortDescription,
                description = typedDescription,
                emotion = typedEmotion?.toEmotion(),
                subject = typedSubject?.toSubject()
            )

    //...
    viewModelScope.launch(Dispatchers.IO) {
            activitiesDatastore.insertActivity(newActivity.toDataIntegraCyLActivity())
            userRepository.setActivity(newActivity.toDataIntegraCyLActivity())
    }
```

Cuando aparece la pantalla de crear un Cuestionario, el proceso es muy similar al de crear una Actividad. Mediante un *LargeDropDownMenu* (un menú saliente con diferentes opciones, en este caso números del uno al diez) el usuario Profesor selecciona el número de preguntas que va a crear de un máximo de diez. A partir de ese número de preguntas se generan cuadros de texto que permiten al profesor introducir las preguntas, y cuando finaliza y pulsa el botón *Terminar y Crear* de nuevo se llama a una función del *ViewModel* que se encarga de manejar la lógica a partir de una lista de *String* formada por las preguntas y la función que permite la navegación al punto de partida en caso de que no haya errores.

```
    Button(
            onClick = {
                createAssignActivityViewModel.createQuestionnaire(allQuestions,
                ↪  goBackToActivities)
            },
            //...
```

La comprobación de error es ciertamente sencilla, únicamente se tiene en cuenta si la lista de preguntas no está vacía y si alguna de ellas es cadena vacía. Si se cumple alguna de estas condiciones, se actualiza el estado del *ViewModel* indicando error en la creación del Cuestionario.

```
    if(typedQuestions.any { it == "" } || typedQuestions.isEmpty()){
            _assignCreateState.update {
                it.copy(
                    questionnaireError = true,
                )
            }
    }
```

Al igual que en el anterior formulario, si hay error se muestra un diálogo de texto informando del error y obligando a introducir las preguntas de manera correcta.

En caso de no haber error, se genera un nuevo Cuestionario, asociándole la nueva Actividad creada en el paso anterior y estableciendo el estado de la misma como *Pendiente*. Como un Cuestionario está formado por preguntas y respuestas almacenadas en un atributo *HashMap* donde las claves son las preguntas y los valores de cada una de ellas son las respuestas, se inicializa uno vacío introduciendo las preguntas introducidas y respuestas vacías.

Por último, al crearse como Cuestionario nuevo, se establece que es el Cuestionario por defecto para esa Actividad. Esto quiere decir que cuando la Actividad es asignada a un Alumno, se genera una copia exacta del Cuestionario para así no modificar el existente y que el Alumno pueda usar la copia como el suyo. Cada Actividad única solo tiene una instancia, pero para cada Actividad existe un Cuestionario por Alumnno y el Cuestionario base.

Esto se logra estableciendo el atributo *isBase* como verdadero al crear un nuevo Cuestionario.

```
    for(question in typedQuestions){
        newQuestionsAndAnswers[question] = ""
    }
    val newQuestionnaire = Questionnaire(
        id = newId,
        isBase = true,
        activityId = newActivityId,
        activityState = IntegraCyLActivityState.OnPending,
        questionsAndAnswers = newQuestionsAndAnswers
    )
```

Por último, se llaman a las funciones encargadas de insertar el nuevo Cuestionario en la Base de Datos local y en repositorio remoto de Firebase.

```
viewModelScope.launch(Dispatchers.IO) {
    questionnaireDatastore.insertQuestionnaire(newQuestionnaire.toDataQuestionnaire())
    userRepository.setQuestionnaire(newQuestionnaire.toDataQuestionnaire())
}
```

8.12. Asignar Actividad

Para que un Profesor pueda asignar una Actividad a uno de sus Alumnos, desde la pantalla de crear y asignar Actividades, selecciona la Actividad a asignar y una vez vistos los detalles de la Actividad y el Cuestionario asociado, dispone de un *LargeDropDownMenu* para seleccionar a uno de sus Alumnos y un botón para confirmar.

Al pulsar el botón, se llama a una función del *ViewModel* encargada de la lógica, proporcionándola el identificador de la Actividad, el Cuestionario de la Actividad y el Alumno a asignar.

```
    // Assign button
    Button(
        onClick = {
            isAssigned = !isAssigned
            createAssignActivityViewModel.assignActivity(
                activity.id,
                questionnaire,
                if (selectedStudent != -1) students[selectedStudent] else null
            )
        },
        //...
```

Dentro de esta función, la comprobación que se hace es si se ha seleccionado algún Alumno, en caso negativo se actualiza el estado del ViewModel indicando error en la asignación.

La lógica dentro de la función en caso de haber un Alumno seleccionado es la siguiente. Primero, se crea un nuevo Cuestionario exactamente igual al asignado por defecto a la Actividad pero con un identificador nuevo y estableciendo que no es un Cuestionario base, es decir, que es modificable.

```
    val newQuestionnaire = questionnaire.copy(
        id = newId,
        isBase = false
    )
```

Una vez hecho esto, se actualiza el Alumno añadiendo en su *HashMap* de actividades y cuestionarios los identificadores de la Actividad a asignar y del nuevo Cuestionario.

```
    // Updated the HashMap of activities and questionnaires IDs, adding the newOnes
    val updatedStudentActivitiesQuestionnaires = student.activitiesAndQuestionnairesId
    updatedStudentActivitiesQuestionnaires.put(activityId, newId)

    // Update the student
    val updatedStudent = student.copy(activitiesAndQuestionnairesId =
    ↪  updatedStudentActivitiesQuestionnaires)
```

Se actualiza el estado del ViewModel introduciendo el nuevo Cuestionario y el Alumno actualizado, se hacen llamadas a los diferentes repositorios para reflejar estos cambios en la Base de Datos local y en remoto usando Firebase.

```
    viewModelScope.launch(Dispatchers.IO) {
            userRepository.setStudent(updatedStudent.toDataStudent())
            userRepository.setQuestionnaire(newQuestionnaire.toDataQuestionnaire())
            questionnaireDatastore.insertQuestionnaire(newQuestionnaire
            ↪  .toDataQuestionnaire())
    }
```

Capítulo 9

Pruebas

Este capítulo recoge las pruebas establecidas y sus resultados para la comprobación del correcto funcionamiento de la aplicación.

9.1. Pruebas de Sistema

Las pruebas de Sistema son una etapa crítica y esencial para garantizar que el software creado cumpla con los requisitos y funcionalidades esperadas. Estas pruebas constituyen una fase crucial en el proceso de asegurar la calidad del producto final antes de su lanzamiento y puesta en producción.

El objetivo primordial de las pruebas de sistema es verificar que el conjunto de módulos y componentes integrados funcione de manera coherente y satisfaga los Casos de Uso y Requisitos especificados en los requisitos del proyecto, en el Capítulo 4.

Durante esta fase, el enfoque se desplaza desde el código fuente y los detalles internos hacia la perspectiva del usuario final, evaluando la aplicación en su totalidad como un sistema completo. Esto permite detectar posibles errores, fallos de interoperabilidad, comportamientos inesperados o deficiencias en el cumplimiento de los requerimientos funcionales y no funcionales.

Las pruebas de sistema abarcan diferentes aspectos de la aplicación, como la navegación, la interfaz de usuario, la funcionalidad principal y las características secundarias. Además, se verifica que el sistema se comporte de manera adecuada ante diferentes escenarios y condiciones, simulando situaciones reales a las que se enfrentaría durante su uso en producción.

Para este proyecto, se realizarán diversas pruebas de sistema sobre la aplicación informática en cuestión, con el fin de validar su comportamiento y rendimiento según los casos de uso establecidos previamente. Mediante la ejecución de estos escenarios de prueba, se podrá de-

mostrar la eficiencia y eficacia del software, asegurando su robustez y adecuación para su despliegue en un entorno de producción.

9.2. Casos de Prueba

Los siguientes casos de prueba (Figuras 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9, 9.10, 9.11 y 9.12) se han realizado una vez completada la implementación de la aplicación. Se comprobarán las funcionalidades más importantes y cómo se comporta el sistema ante su uso en el mundo real.

Identificador	PS-01.
Nombre	Inicio de sesión con datos correctos.
Prueba	El usuario introduce los credenciales de prueba correctos.
Resultado	El servidor comprueba los datos y la aplicación navega hasta la pantalla de permisos.
Valoración	Correcta.

Tabla 9.1: PS-01. Inicio de sesión con datos correctos.

Identificador	PS-02.
Nombre	Inicio de sesión con datos incorrectos.
Prueba	El usuario introduce los credenciales de prueba incorrectos.
Resultado	El servidor devuelve error y se muestra un cuadro de diálogo informativo.
Valoración	Correcta.

Tabla 9.2: PS-02. Inicio de sesión con datos incorrectos.

Identificador	PS-03.
Nombre	Mantener sesión iniciada.
Prueba	El usuario selecciona mantener sesión iniciada.
Resultado	El usuario cierra la aplicación y al volver a iniciarla no es necesario identificarse de nuevo.
Valoración	Correcta.

Tabla 9.3: PS-03. Mantener sesión iniciada.

Identificador	PS-04.
Nombre	Obtener actividades.
Prueba	El usuario navega a la vista de actividades.
Resultado	Las actividades son mostradas al usuario con todos sus datos.
Valoración	Correcta.

Tabla 9.4: PS-04. Obtener actividades.

Identificador	PS-05.
Nombre	Ver perfil.
Prueba	El usuario accede a la opción de ver su perfil.
Resultado	Se muestra la información del perfil del usuario con todos sus detalles.
Valoración	Correcta.

Tabla 9.5: PS-05. Ver perfil.

Identificador	PS-06.
Nombre	Completar actividad.
Prueba	El usuario selecciona una actividad y la completa según las indicaciones.
Resultado	La actividad se registra como completada en el sistema y se actualiza su estado.
Valoración	Correcta.

Tabla 9.6: PS-06. Completar actividad.

Identificador	PS-07.
Nombre	Ver emoción (usuario alumno)
Prueba	El usuario navega a la vista de emociones
Resultado	Se muestran las emociones disponibles y la actual del alumno resaltada en verde.
Valoración	Correcta.

Tabla 9.7: PS-07. Ver emoción (usuario alumno).

Identificador	PS-08.
Nombre	Seleccionar emoción (usuario alumno).
Prueba	El usuario alumno selecciona una emoción.
Resultado	La emoción seleccionada se registra correctamente y se actualiza resaltandose en la pantalla. Un pequeño mensaje confirma el cambio.
Valoración	Correcta.

Tabla 9.8: PS-08. Seleccionar emoción (usuario alumno).

Identificador	PS-09.
Nombre	Crear nueva actividad (usuario profesor).
Prueba	El usuario profesor crea una nueva actividad con todos los campos requeridos junto con un cuestionario base asociado.
Resultado	La actividad se registra correctamente en el sistema junto con el cuestionario y se muestra en la lista de actividades disponibles.
Valoración	Correcta.

Tabla 9.9: PS-09. Crear nueva actividad (usuario profesor).

Identificador	PS-10.
Nombre	Asignar actividad (usuario profesor).
Prueba	El usuario profesor asigna una actividad a un alumno específico.
Resultado	La actividad se muestra en la lista de actividades asignadas al alumno correspondiente y en la lista de actividades del profesor.
Valoración	Correcta.

Tabla 9.10: PS-10. Asignar actividad (usuario profesor).

Identificador	PS-11.
Nombre	Buscar actividad (usuario profesor).
Prueba	El usuario profesor realiza una búsqueda de una actividad específica mediante la barra de búsqueda.
Resultado	Se muestra una lista de actividades que coinciden con el nombre introducido en la barra de búsqueda.
Valoración	Correcta.

Tabla 9.11: PS-11. Buscar actividad (usuario profesor).

Identificador	PS-12.
Nombre	Ver listado de alumnos (usuario profesor).
Prueba	El usuario profesor accede al listado de alumnos que tiene asignados.
Resultado	Se muestra una lista de alumnos asignados con su nombre completo y emoción actual.
Valoración	Correcta.

Tabla 9.12: PS-12. Ver listado de alumnos (usuario profesor).

Capítulo 10

Seguimiento del proyecto

Durante la planificación inicial se realizó una estimación de las fases en las que se dividiría el proyecto, acorde a la Figura 3.2. Una vez finalizado el proyecto se observan los tiempos cumplimentados, y los que no han alcanzado la fecha propuesta en un principio, como se muestra en la Tabla 10.1.

Fase	Inicio estimado	Inicio real	Fin estimado	Fin real
Análisis de requisitos	13/02/2023	13/02/2023	21/02/2023	21/02/2023
Análisis del sistema	22/02/2023	22/02/2023	24/02/2023	25/02/2023
Diseño de sistema	27/02/2023	04/03/2023	15/03/2023	22/03/2023
Codificación	16/03/2023	23/03/2023	09/06/2023	23/06/2023
Pruebas	12/06/2023	24/06/2023	23/06/2023	06/07/2023

Tabla 10.1: Comparación de tiempo de la completitud de las fases.

Como se puede observar las fechas estimadas no se han podido completar acorde al plan de trabajo establecido. Esto se debe a la ejecución de los riesgos 3.3, 3.6 y 3.8. Debido a que se ha sobrepasado el límite establecido para el día 23 de junio de 2023, fecha para la entrega del proyecto, se ha tenido que aplazar al día 11 de julio de 2023 para la convocatoria extraordinaria.

Capítulo 11

Conclusiones y trabajo futuro

Este capítulo recoge las distintas conclusiones que se han obtenido una vez finalizado este proyecto Trabajo Fin de Grado y una valoración personal por parte del autor.

11.1. Conclusiones

A partir de los objetivos descritos en la Sección 1.2 se hace un análisis del grado de consecución de cada uno de ellos.

Para los objetivos específicos sujetos al desarrollo del proyecto:

1. **Desarrollar una aplicación móvil.** Se ha conseguido diseñar e implementar en su totalidad una aplicación móvil que cumple unos requisitos y casos de uso especificados previamente.
2. **Seguir todas las etapas del desarrollo software.** Durante el desarrollo del proyecto se ha conseguido seguir unas pautas predefinidas para facilitar el proceso y conseguir un resultado más logrado y prometedor.
3. **Desarrollar una documentación fidedigna.** Se ha conseguido mantener una documentación actualizada que refleje el desarrollo del proyecto y que cuente con todas las etapas definidas para el desarrollo de un proyecto software.

Para los objetivos académicos impuestos por el desarrollador:

1. **Conocer una tecnología actual y nueva para el alumno**. Gracias a la formación previa realizada y comentada en el Capítulo 8, se ha conseguido desarrollar una aplicación móvil mediante el uso de tecnologías de vanguardia y desconocidas en un principio para el autor, como pueden ser *Kotlin* o *Jetpack Compose*.

2. **Crear una interfaz agradable e intuitiva al usuario**. Haciendo uso de *Jetpack Compose* se ha logrado una interfaz que cumple con los requisitos establecidos, manteniendo la sencillez.

3. **Seguir una estructura de aplicación de acuerdo con los principios de *Clean Architecture* [4].** Se ha logrado cumplir este objetivo haciendo uso y aplicando el patrón arquitectónico *MVVM* y patrones de diseño como *DAO* o *Factory*, descritos en el Capítulo 6.

En general la balanza final es muy positiva, y gracias al trabajo dedicado se han conseguido cumplir los objetivos especificados en la Sección 1.2.

11.2. Valoración personal

En primer lugar, la idea de hacer un proyecto tan completo y amplio, marcando el final de una etapa educativa asustó, pensé que quizás me quedaba algo grande y que no estaba lo suficientemente preparado.

Una de las dificultades iniciales fue crear de cero la aplicación usando tecnologías completamente desconocidas en ese momento para mí como eran *Kotlin* y *Jetpack Compose*. Gracias a mi etapa de prácticas universitarias, pude conocer mejor estas tecnologías y el mundo de desarrollo móvil, disfrutándolo cada vez más y siendo a día de hoy mi meta a futuro, dentro de todos los campos de la informática. La fase de formación fue sin duda el empujón y motivación que necesitaba para el desarrollo de la aplicación.

A medida que avanzaba en el proyecto fui entendiendo la importancia de las diferentes fases que alberga un desarrollo de estas magnitudes: planificación, análisis o diseño, entre ellas. Ahora comprendo lo necesarias que son para poder lograr un trabajo digno y resultados convincentes.

Aunque me quede un mal sabor de boca por no haber conseguido entregar el proyecto en el primer plazo establecido de entrega y por ello haber estado las últimas semanas con más prisas, estoy bastante orgulloso con el resultado y con todo lo aprendido por el camino, de manera activa o pasiva.

11.3. Trabajo futuro

A partir de la funcionalidad final de la aplicación y una vez terminado el desarrollo de la misma, se ha realizado una reflexión junto con el tutor del Trabajo de Fin de Grado en donde se han debatido posibles líneas futuras para la aplicación. Como mejoras futuras se podrían añadir:

- **Asignar de manera automática actividades**. Se podría implementar que, cuando un alumno seleccionase cierta emoción o estado de ánimo en el que se encuentra, se le asignaran actividades asociadas a esa emoción por defecto.
- **Diferentes roles de profesores.** Otra opción que podría ser interesante es la posibilidad de crear diferentes roles para los usuarios profesores, como por ejemplo director o tutor, y que un director pudiera ver todos los alumnos matriculados en su centro educativo.
- **Integración con *iOS***. Al ser una aplicación escrita en *Kotlin*, sería sencillo una migración para que la aplicación esté también para el sistema operativo de *Apple*.
- **Evaluación de actividades**. El usuario profesor podría corregir las actividades realizadas por sus alumnos mediante la aplicación y que el resultado de esa evaluación fuese visible para el alumno.
- **Tiempo de realización**. Incorporar la opción de establecer una fecha de entrega límite para ciertas actividades.
- **Actividades privadas.** A día de hoy cualquier actividad creada por un profesor es pública y cualquier otro profesor puede acceder a ella. Introducir la posibilidad de crear actividades individualizadas a ciertos alumnos, proporcionando así más personalización.
- **Personalización de cuestionarios**. Actualmente los cuestionarios están formados por un máximo de diez preguntas. Se podrían implementar diferentes tipos de cuestionarios, como selección de imagenes o preguntas tipo test.
- **Disponibilidad de varios idiomas**. Actualmente la aplicación está desarrollada para el idioma inglés. Ya que la aplicación está destinada para alumos refugiados y cabe la posibilidad de que sólo hablen en su idioma natal, se propone la implementación de una traducción integrada para la selección de un idioma específica.

Apéndice A

Manuales

A.1. Manual de despliegue e instalación

Normalmente para la instalación de una aplicación Android, el proceso es sencillo, basta con acceder a la Google Play Store [36] o una similar disponible en el dispositivo y en el buscador introducir el nombre de la aplicación.

Para este caso, el proceso es algo más singular. Para instalar la aplicación se requiere un **smartphone o tablet Android** con versión mínima de **Android 9.0 (Pie)**.

El proceso de instalación se empieza accediendo al repositorio de *GitHub* cuyo enlace se encuentra en el Apéndice B y **generando el archivo .apk y descargándolo** en el dispositivo Android. Hay que ejecutar ese archivo para que comience el proceso de instalación siguiendo las instrucciones del asistente de instalación del dispositivo.

A.2. Manual de mantenimiento

Antes de empezar con el proceso de instalación del código y del proyecto, hay varias consideraciones a tener en cuenta.

La primera de ellas es que el proyecto ha sido desarrollado con *Jetpack Compose*, por lo tanto es necesario disponer de una versión actualizada del mismo, disponible en [37].

La segunda es que es necesario contar con *Android Studio* en nuestro dispositivo para poder interactuar y modificar en caso que fuera necesario el código del proyecto. Para instalar *Android Studio* basta con acceder a [38].

Una vez se tenga una versión estable de *Jetpack Compose*, se siguen los siguentes pasos:

1. Descargar en formato *.zip* el código del proyecto disponible en el Apéndice B.
2. Extraer el proyecto en la carpeta que se desee.
3. Abrir el proyecto en *Android Studio*. *File* → *Open*, → seleccionar la carpeta *root* del proyecto.
4. Esperar unos pocos minutos a *Gradle* finalice la configuración del proyecto.
5. El código ya está disponible para su mantenimiento.

A.3. Manual de usuario

La primera vez que se accede a la aplicación, lo primero que se muestra en una pantalla de inicio de sesión. En caso de ser un usuario registrado en el sistema y con disponibilidad de acceder a la aplicación, es necesario introducir el nombre de usuario y la contraseña asignados. El nombre de usuario está formado por la concatenación de las primeras letras del nombre y los dos primeros apellidos, y los dos últimos dígitos del año de nacimiento.

Antes de iniciar sesión, existe la posibilidad de marcar una casilla para mantener la sesión iniciada. Al seleccionarla, aunque se cierre la aplicación, el inicio de sesión ya no volverá a ser necesario.

Al clickar sobre el botón de iniciar sesión, en caso de algún dato incorrecto, se mostrará por pantalla el error y se tendrán que volver a introducir los datos. En caso contrario, la aplicación avanzará a la ventana principal, la cual muestra el listado de actividades.

A.3.1. Inicio de sesión como Alumno

En caso de haber iniciado sesión con el rol de Alumno, la barra de navegación inferior de la aplicación contará con tres pestañas a disposición del usuario. La primera de ellas es la **ventana de Actividades**, y es la que se muestra como principal una vez se ha iniciado sesión.

La sección de Actividades está formada por un **listado de Actividades asignadas al Alumno** y un filtro de búsqueda. El filtro de búsqueda, situado en la parte superior de la pantalla, está formado por tres opciones cliqueables a modo botón. Las opciones definen los tres estados por los que puede pasar una actividad, siendo estos Pendiente (*On Pending*), En curso (*On course)* o Finalizada (*Finished*). Al pulsar sobre uno de los filtros, el listado de Actividades se actualiza y sólo aparecen las Actividades cuyo estado coincida con el seleccionado.

Las Actividades son mostradas con un fondo con una imagen de la asignatura asociada, el nombre de la Actividad, un botón para desplegarla y una pequeña descripción.

El usuario Alumno puede clickar el botón con la opción de desplegar con el que cuenta cada una de las Actividades. Al hacerlo, la Actividad se agranda ocupando gran parte de la ventana y se muestra la descripción detallada de la actividad, el nombre de la asignatua asociada en caso de haberla y el nombre de la emoción/estado de ánimo asociado en caso de haberlo. Además aparece un botón *Empezar*, el cuál al pulsarlo hace que la Actividad pase al estado de En Curso

El texto de la descripción es deslizable, y al llegar al final aparece un botón con la opción *Completar*. Al seleccionar esta opción aparece el **Cuestionario de la Actividad**, el cual está formado por un máximo de diez preguntas a las que hay que introducir respuesta en un campo de texto. Una vez completado el Cuestionario y pulsando el botón *Finalizar* que aparece debajo de la última pregunta, el estado de la Actividad pasa a *Finalizada* y se vuelve a la pantalla inicial con el listado de Actividades.

Si desde la barra de navegación inferior se accede al **Perfil**, la pantalla se actualiza mostrando el nombre completo del alumno, es decir nombre y dos primeros apellidos, su rol (Alumno), su nombre de usuario y contraseña, y sus datos del centro educativo. Estos últimos son el nombre del centro, etapa educativa (ESO, Bachillerato, Primaria o Infantil), curso y clase.

Desde la barra de navegación también se puede acceder a **Emociones**. Cuando esto ocurre, la pantalla muestra cuatro iconos de cuatro caras que muestran emociones: Feliz, Triste, Enfadado y Sorprendido. La emoción que esté seleccionada se representa con el icono correspondiente en color verde, mientras que el resto aparecen en negro. Al pulsar sobre cualquiera de ellos, se actualiza el color y el sistema guarda la emoción seleccionada como emoción actual del alumno.

A.3.2. Inicio de sesión como Profesor

En caso de haber iniciado sesión con el rol de Profesor, la barra de navegación inferior de la aplicación contará con cuatro pestañas a disposición del usuario. La primera de ellas es la **ventana de Actividades**, y es la que se muestra como principal una vez se ha iniciado sesión.

Esta primera ventana funciona exactamente igual que para un usuario Alumno. Las diferencias a destacar es que el listado de Actividades estará formado por todas las Actividades asignadas a los Alumnos del Profesor. Otra diferencia es que al pasar a la parte del Cuestionario de la Actividad, no habrá opción para rellenar las respuestas y contestar al Cuestionario, si no que ya saldrá el Cuestionario completo para su corrección.

Otra sección que se mantiene para el usuario Profesor en la barra de navegación inferior es la pantalla de **Perfil**. La única diferencia es que no se muestran los datos del curso completo, solamente el Centro Educativo asignado.

Desde la barra de navegación, se puede seleccionar el apartado Alumnos. La pantalla se actualiza para mostrar un **listado de los Alumnos asignados al Profesor**, para cada Alumno se muestra su nombre completo y su estado de ánimo actual. Esto último le sirve

de guía al Profesor para asignar Actividades. Esa misma pantalla cuenta con una barra de búsqueda que muestra los alumnos cuyo nombre completo coincida con lo introducido.

La última sección disponible en la barra de navegación inferior para el Profesor es la de Creación y Asignación de Actividades. Cuando esta ventana es seleccionada, la pantalla se actualiza mostrando un botón para **crear una Actividad de Cero** seguido de un listado de Actividades existentes disponibles para **asignar**.

Si se pulsa *Crear de Cero*, aparece en pantalla un formulario para completar los campos anteriormente mencionados de una Actividad: nombre, pequeña descripción, descripción detallada, asignatura o emoción asociadas. Al seleccionar el botón *Siguiente*, se muestra un segundo formulario para introducir un máximo de diez preguntar que constituirán el Cuestionario de la Actividad. Al seleccionar *Completar* se **crea la nueva Actividad** y estará disponible en el listado de Actividades a asignar.

Si se desea **asignar una Actividad a un Alumno**, en la el listado inferior que aparece en pantalla, al desplegar la información completa de una Actividad y pulsar *Asignar*, aparece un listado con los Alumnos asignados al Profesor. Al seleccionar uno, la Actividad se asignará y al Alumno le aparecerá la Actividad con el estado Pendiente.

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://github.com/juant351/TFG_app`.

Bibliografía

[1] Android Developers. Viewmodel. https://developer.android.com/topic/libraries/architecture/viewmodel?hl=es-419&authuser=3. (2023).

[2] Oscar Blancarte. Patron daol. https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/. (2023).

[3] Digital Guide IONOS. ¿qué es el patrón factory. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-factory/. (2023).

[4] Robert C. Martin (Uncle Bob). Clean architecture: A craftsman's guide to software structure and design. (2023).

[5] Tecnología Educativa del Laboratorio de Idiomas FILO:UBA. ablapp. https://ablapp.org/. (2023).

[6] kiddoware. Kids story creator. https://play.google.com/store/apps/details?id=com.kiddoware.ai.little.kids.bedtime.story.books.generator. (2023).

[7] Anastasia Stsepanets. Desarrollo en cascada. https://blog.ganttpro.com/es/metodologia-de-cascada/. (2023).

[8] Microsoft. Modelo-vista-modelo de vista (mvvm). https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm. (2023).

[9] Android Developers. Jetpack compose. https://developer.android.com/jetpack/compose?hl=es-419&authuser=3. (2023).

[10] Android Developers. Google for developers. https://firebase.google.com/?hl=es. (2023).

[11] Computer Weekly. Base de datos nosql. https://www.computerweekly.com/es/definicion/NoSQL-o-base-de-datos-No-Solo-SQL. (2023).

[12] Wikipedia. Nosql ventajas. https://es.wikipedia.org/wiki/NoSQL. (2023).

[13] Wikipedia. Microsoft project. https://es.wikipedia.org/wiki/Microsoft_Project. (2023).

[14] Wikipedia. Microsoft excel. https://es.wikipedia.org/wiki/Microsoft_Excel. (2023).

[15] Wikipedia. Trello. https://es.wikipedia.org/wiki/Trello. (2023).

[16] Meet Android Studio. Android Studio. https://developer.android.com/studio/intro. (2023).

[17] Elena Canorea. ¿qué es kotlin y para qué sirve? https://www.plainconcepts.com/es/kotlin-android/. (2023).

[18] Android Developers. Kotlin y android. https://developer.android.com/kotlin?hl=es-419. (2023).

[19] Material Components. Build beautiful, usable products with material components for android, flutter, ios, and the web. https://material.io. (2023).

[20] Material Components. Dark theme. https://material.io/design/color/dark-theme.html. (2023).

[21] Android Developers. Introduction to animations. https://developer.android.com/training/animation/overview. (2023).

[22] Wikipedia. Github. https://es.wikipedia.org/wiki/GitHub. (2023).

[23] Diego Camacho. Qué es github y cómo usarlo para aprovechar sus beneficios. https://platzi.com/blog/que-es-github-como-funciona/. (2023).

[24] SourceTree. Sourcetree. https://www.sourcetreeapp.com/. (2023).

[25] Carlos Cuesta. Gitmoji. https://gitmoji.dev. (2023).

[26] Carlos Cuesta. Gitmoji - about. https://gitmoji.dev/about. (2023).

[27] Android Developers. Android basics in kotlin. https://developer.android.com/courses/android-basics-kotlin/course. (2023).

[28] Udacity. Kotlin bootcamp for programmers. https://www.udacity.com/course/kotlin-bootcamp-for-programmers--ud9011. (2023).

[29] JetBrains. Kotlin koans. https://play.kotlinlang.org/koans/overview. (2023).

[30] Android Developers. Aspectos básicos de android con compose. https://developer.android.com/courses/android-basics-compose/course. (2023).

[31] Android Developers. Jetpack compose para desarrolladores de android. https://developer.android.com/courses/jetpack-compose/course. (2023).

[32] Gradle Inc. What is gradle? https://docs.gradle.org/current/userguide/what_is_gradle.html. (2023).

[33] Anupam Chugh. Android shared preferences example tutorial. https://www.digitalocean.com/community/tutorials/android-shared-preferences-example-tutorial. (2023).

[34] Universidad de Valladolid. Campus virtual uva. https://campusvirtual.uva.es/. (2023).

[35] Escuela de Ingeniería Informática de Valladolid. Escuela de ingeniería informática de valladolid. `https://www.inf.uva.es/`. (2023).

[36] Google Play. Google play store - apps. `https://play.google.com/store/apps`. (2023).

[37] Google Developers. Inicio rápido - jetpack compose. `https://developer.android.com/jetpack/compose/setup`. (2023).

[38] Google Developers. Download android studio app tools. `https://developer.android.com/studio`. (2023).