Grado en Ingeniería Informática

Trabajo de Final de Grado

Proyecto feedback Frontend Móvil

Autor:
Carlos Mora Hernández

Supervisor:
Sergio Aguado González
Tutor académico:
Angel Pascual del Pobil Ferré

Fecha de lectura: 13 de Julio de 2021
Curso académico 2020/2021

Resumen

En este documento se incluye la memoria del Trabajo Final de Grado que ha sido desarrollado en el transcurso de la estancia en prácticas en Cuatroochenta. El proyecto que se ha desarrollado durante la estancia ha consistido en una aplicación móvil orientada a la realización de publicaciones por parte de gente que esta aprendiendo sobre un tópico y desean recibir una retroalimentación sobre el trabajo que han realizado. Asimismo, los usuarios de la aplicación también podrán realizar otro tipo de acciones como consultar los rankings o sus datos del perfil.

El proyecto es una aplicación Android Nativa. Es por ello que ha sido desarrollado en Kotlin y XML, para la gestión de la lógica y de las vistas respectivamente. Además el desarrollo ha seguido la metodología ágil SCRUM, en la cual se han ordenado las diferentes historias estimadas y ordenadas en una *product backlog*.

En proyecto ha sido realizado conjuntamente con otro compañero encargado de la parte del *Backend*, con el cual se ha tenido que trabajar para analizar y diseñar el sistema sobretodo durante las primeras fases del desarrollo.

Palabras clave

Móvil, Retroalimentación, vista, Android, Kotlin, Frontend.

Keywords

Feedback, product backlog, ranking, Android, Kotlin, Frontend, tags.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

En este capítulo se va a realizar una breve introducción a la empresa con la que se ha colaborado, las motivaciones a la hora de realizar este proyecto junto con sus objetivos y alcance. Finalmente se va a detallar la estructura que va a seguir esta memoria.

1.1. Contexto y motivación del proyecto

Cuatroochenta [5] es una empresa tecnológica de tamaño mediano fundada en 2011 y que se dedica al desarrollo de aplicaciones y servicios *cloud* con el propósito de prestar soluciones a los diferentes proyectos propuestos por los clientes. La empresa tiene sedes en 14 países distintos y tiene su sede central en el Espaitec de la Universidad Jaume I.

Cuatroochenta hace uso de diferentes herramientas, como pueden ser Android Studio, React Native, Webstorm o *Jira* entre otras. Con ellas se han encargado de realizar proyectos como Metro Málaga, la app oficial de Aquarama o Mundo Consum. También cuenta con una unidad de ciberseguridad dedicada a la gestión de instituciones financieras y infraestructuras críticas. En este caso, Cuatroochenta requiere de una aplicación para dispositivos móviles en la cual los usuarios puedan recibir y prestar *feedback* de diferentes temáticas a las personas que lo necesiten.

La principal motivación para la realización de este proyecto ha sido el que expertos en alguna materia puedan prestar su ayuda a personas que están en un proceso de aprendizaje, para que así puedan mejorar sus capacidades gracias a sus consejos, pautas o recomendaciones. También se busca agilizar la comunicación entre los expertos y los aprendices, así como proporcionar acceso a personas que no tienen a su alcance alguna forma de recibir ayuda sobre la temática de la cual están aprendiendo. Finalmente, se busca dar visibilidad a los expertos que más y mejores ayudas presten a los aprendices que lo requieran, para así fomentar la competitividad sana entre ellos y por lo tanto acabar favoreciendo a los aprendices que requieren ayuda.

En cuanto al estado del arte, existen aplicaciones parecidas o que tienen características similares en el mercado, foros o incluso algunas redes sociales, tienen una fisonomía parecida en

cuanto a la idea principal de la aplicación, la cual es que un usuario pueda publicar algo y otro le responda. Sin embargo no se tendrán en cuenta de manera directa para la realización de este proyecto debido a que se quiere mostrar un enfoque distinto al que ofrecen estas aplicaciones o webs. A pesar de esto, sería ilógico no obtener algunas ideas o enfoques de estas aplicaciones, debido a la experiencia que habrán obtenido al sacar estas aplicaciones al mercado y al grupo de profesionales más experimentados que las habrán desarrollado. Al fin y al cabo, un enfoque nuevo no significa no reconocer ciertos elementos cuyo funcionamiento haya sido demostrado y que son la mejor opción actual.

1.2. Objetivos del proyecto

En esta sección se va a profundizar de manera mas exhaustiva en los objetivos primordiales del proyecto y en el alcance del mismo.

El principal objetivo del proyecto es crear una plataforma para aportar y recibir *feedback* en cualquier tipo de temática.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Permitir al aprendiz publicar sus progresos sobre la materia en la que se está formando.
- Permitir a la persona experta en la materia dar una retroalimentación a los aprendices.
- Agilizar la comunicación entre los expertos y los aprendices.
- Permitir el acceso a recibir una retroalimentación sobre el trabajo realizado a personas que no tenían esa posibilidad.
- Prestar ayuda a las personas que deseen aprender.

Por lo que se refiere al alcance funcional de este proyecto, los usuarios (tanto expertos como aprendices) podrán darse de alta en la aplicación a través de un formulario donde especificarán su información personal, así como el usuario y contraseña que utilizarán para acceder a la aplicación. Una vez registrados, los usuarios tendrán en cualquier momento la opción de modificar y ver sus datos, así como darse de baja en la aplicación.

Desde dentro de la aplicación, los aprendices tendrán la opción de realizar publicaciones para poder recibir *feedback*, como datos de la publicación tendrán que especificar un título, una descripción y una categoría o tema. Opcionalmente, podrán incluir tags, imágenes, vídeos o documentos.

Las categorías a las cuales se pueden realizar publicaciones pueden corresponder a cualquier tema, siempre y cuando este aprobado por un administrador. Para ampliar el espectro de categorías, habrá un buzón de sugerencias en el cual los usuarios podrán depositar nuevas categorías a añadir. Por otra parte, las categorías pueden desglosarse en subcategorías más concretas. Las publicaciones de cada categoría o subcategoría podrán ser consultadas y si el usuario tiene el rol

de experto, podrá marcar como favoritas las que más le gusten para así poder verlas y acceder de manera más rápida a las publicaciones que le interese contestar.

Cuando se realice una publicación en una categoría específica, los expertos tendrán acceso a ella mediante un listado conjunto con el resto de publicaciones, el cual podrán filtrar mediante la categoría o mediante un buscador en el cual se podrá especificar la publicación, la categoría o el usuario con el propósito de encontrar la publicación que se desea. Una vez encontrada la publicación que se desea, los expertos podrán dar *feedback* a esa publicación. En este mensaje de respuesta, se incluirá una descripción y opcionalmente se podrán incluir archivos multimedia y documentos.

Por otra parte, cuando el aprendiz reciba *feedback* de un experto, este podrá puntuar la calidad del mismo en una escala del 1 al 5, con el propósito de crear un ranking con los expertos mejor valorados por los aprendices. También existirá un *ranking* de expertos más activos y de categorías más activas para incentivar la participación dentro de la aplicación.

Para finalizar, una publicación podrá ser marcada como indebida si los usuarios creen que no se están cumpliendo las normas de la aplicación. Por otro lado, los usuarios siempre tendrán acceso a un historial con las publicaciones que hayan realizado o dado *feedback* para consultarlas si así lo desean.

En cuanto al alcance organizativo, los departamentos que participarán en la empresa serán: el departamento de desarrollo de aplicaciones móvil, el cual se encargará de la parte del *Frontend* de la aplicación, el departamento de desarrollo de aplicaciones web el cual trabajará en conjunto en el desarrollo del proyecto y se encargará del *Backend* de la aplicación y el departamento de UX/UI que mostrará apoyo en la definición de la usabilidad y de la experiencia de usuario.

Finalmente, el alcance informático del proyecto consistirá en desarrollar una aplicación móvil mediante una tecnología como es Android [6] y un lenguaje de programación como Kotlin [19] y XML [15]. Como IDE, se hará uso de Android Studio, ya que es la herramienta predilecta para desarrollo de aplicaciones móvil nativas.

1.3. Descripción del proyecto

En esta sección se va a aportar cierta información adicional sobre el proyecto que puede ser de utilidad.

El proyecto se va a desarrollar conjuntamente con otro compañero que será el encargado de desarrollar el Backend de la aplicación, en paralelo al desarrollo del *Frontend* el cual realizo yo. El *product backlog* es común tanto para el *Backend* como para el *Frontend* en aras de aunar el desarrollo de la aplicación y cada una de las historias se dividirán en las tareas necesarias de ambas partes.

Los conocimientos en cuanto a tecnologías con los que se empezará el proyecto son básicos, dado que es la primera vez que desarrollo una aplicación Android nativa. Es por ello que durante el transcurso del proyecto se tratará de adquirir conocimientos en desarrollo de aplicaciones

móvil, tanto a nivel de organización del proyecto como a nivel de arquitectura. En la finalización del proyecto se espera obtener un resultado que haya utilizado buenas prácticas de programación así como, una aplicación que cumpla con ciertos estándares de calidad y de usabilidad de cara al usuario.

1.3.1. Herramientas

Android Studio

Esta será la herramienta que se utilizará para desarrollar la aplicación móvil. Es el IDE recomendado para desarrollar aplicaciones Android Nativas y dada la similitud con IntelliJ no debería suponer un desafío su utilización.

Jira Software

Para la organización del proyecto se utilizará *Jira Software* [2] dado que es la herramienta que utiliza la empresa internamente. En este proyecto se utilizará esta herramienta para crear la *product backlog* y los Sprints que sean necesarios, además también se computarán las horas realizadas en cada tarea y se generaran los informes que se mostraran a lo largo de esta memoria.

Github

En aras de tener un control de versiones en el proyecto y además que este alojado en la nube, se ha elegido esta herramienta para cubrir esa necesidad. Además, si se diera el caso en un futuro de colaborar con más gente en el desarrollo es una herramienta imprescindible dados los diferentes recursos que tiene para facilitar la vida a los grupos de desarrollo como la creación de ramas y pudiendo fusionarlas en un momento dado con la rama principal del proyecto.

Postman

Dado que se deberán hacer peticiones a una API REST, para probar su correcto funcionamiento así como para comprobar la forma de la respuesta, esta es una herramienta que ofrece buenos resultados. La facilidad a la hora de hacer ciertas peticiones sin tener que lanzar la *app* y la fácil repetición de las mismas han sido uno algunos de los puntos clave para elegirla.

Balsamiq Frameworks

Esta herramienta ha sido elegida para realizar los diferentes prototipos de vistas que se hagan en la app, además de la rapidez para realizar prototipos, el poder tenerlos digitalmente

facilita la utilización y búsqueda de los mismos, para poder utilizarlos en documentos como este.

1.4. Estructura de la memoria

En el capítulo 2 se muestra la metodología, planificación y seguimiento que se ha realizado del proyecto separado en Sprints.

Mientras, el capitulo 3 contiene el análisis del sistema, ilustrado con el diagrama de casos de uso y los diagramas de actividades, además también se puede visualizar el diseño del sistema a través de la guía de estilo o de algunos prototipos realizados.

Por otro lado, en el capítulo 4 se pueden observar los detalles de la implementación en los que se muestran los distintos patrones de diseño utilizados, algunas estrategias de implementación y la verificación y validación de la aplicación.

Finalmente en el capítulo 5 se exponen las conclusiones del proyecto, explicando la experiencia personal dentro de la empresa, lo que he aprendido durante todo este proceso y algunos planes de futuro.

Capítulo 2

Planificación del proyecto

2.1. Metodología

La metodología que se ha utilizado para desarrollar el proyecto es *Scrum* [1]. Esta forma parte de las metodologías ágiles. Se decidió utilizarla porqué a parte de que es la que suele utilizar la empresa internamente, era la apropiada para este tipo de proyectos los cuales tienen cierta incertidumbre en cuanto a los requisitos, dado que como se verá en siguientes secciones hay ciertas funcionalidades que se tuvieron que adaptar o incluso ampliar.

La forma de trabajar de esta metodología es creando primeramente un *product backlog* en el cual se ordenan y se estiman las diferentes tareas e historias de usuario con el propósito de extraer las más prioritarias en Sprints de una duración determinada y los cuales se realizan secuencialmente (en nuestro caso los Sprints tuvieron una duración de dos semanas). Antes, durante y después de cada Sprint se realizaron reuniones con el propósito de ordenar, fijar objetivos y organizar el desarrollo de dicho Sprint. Estas reuniones y su forma de realizarlas serán descritas más en profundidad a continuación:

- *Sprint planning* (Reunión de inicio): En la empresa, estas reuniones se realizaban antes de empezar el siguiente Sprint y acudíamos tanto el grupo de desarrollo como el *Scrum Manager* y el *Product Owner*.
- *Daily meeting*: En este caso, la reunión la hacíamos tanto mi compañero encargado del *Backend* como yo diariamente con el propósito de coordinarnos. Además, estas reuniones también servían para asegurar que las funcionalidades que se desarrollaban en el *Frontend* estuvieran disponibles en el *Backend* y por lo tanto, no impidieran su continuidad.
- *Sprint review* (Reunión de cierre): En la empresa, estas reuniones las hacíamos justo antes del las de *Sprint Planning* y mostrábamos a nuestro supervisor, en el rol de *Product owner*, la aplicación y sus progresos durante el Sprint.

Esta metodología esta formada por varios roles, cada uno de ellos con unas responsabilidades distintas y tratando de que Scrum funcione como esta previsto:

- *Product owner*: Este rol fue llevado a cabo por el supervisor de la empresa, dado que el proyecto era interno a la misma. Durante las reuniones que se llevaron a cabo con el equipo de desarrollo nos fue indicando cuales eran las funcionalidades primordiales a la hora de desarrollar el producto, y además, revisó el trabajo que habíamos realizado hasta ese momento dando su visto bueno o proponiendo ciertas mejoras que se pudieran implementar.
- *Scrum master*: Este rol lo desarrollamos una persona diferente dependiendo del Sprint en el que nos encontrásemos. Había Sprints donde lo desarrollé yo y en otros donde lo desarrolló mi compañero encargado de la parte del *Backend*. Esto como consecuencia de las pocas personas que conformábamos el proyecto y como medida para no anclarnos en un rol fijo ninguno de los dos y evitar la generación de egos.
- *Equipo de desarrollo*: En nuestro caso este rol se dividía en dos equipos, el encargado de la parte del *Frontend* que estaba compuesto por mi y el encargado del *Backend* formado por mi compañero.

2.2. Planificación

Teniendo en cuenta la metodología del proyecto, en esta sección se va a explicar la planificación, presentando el *product backlog* y la estimación de las historias que lo componen.

Con el objetivo de poder estimar con mayor conocimiento las diferentes historias del proyecto y de a su vez formarse en las tecnologías con las que se iba a trabajar, antes del comienzo del proyecto se realizó una formación la cual fue estimada en dos semanas. Dada la complejidad de las tecnologías en las cuales me estaba formando, este tiempo podría variar en algunos días.

Una vez terminada esta formación se prosiguió con la especificación y la estimación de las historias de usuario que conformarían el *product backlog*, el cual, se tomó la decisión de que fuera conjunto entre el *Backend* y el *Frontend*. Las historias de usuario fueron establecidas en una reunión junto con el *Product Owner* en la cual se nos transmitió que funcionalidades debía tener la aplicación y cuales eran las más prioritarias, es decir, las primeras que debíamos de tratar de abordar. En el cuadro 2.1 se puede observar el *product backlog* donde se encuentran las historias de usuario estimadas y priorizadas.

Para definir cuantos puntos otorgar a cada historia, se tomó como referencia una de ellas (la más compleja) y se le otorgó una puntuación acorde a su complejidad. A partir de esta historia se definieron los puntos del resto de historias teniendo en cuenta la complejidad y calculados a partir de la experiencia previa.

Finalmente, se ordenó el *product backlog* priorizando las historias que se nos indicó que eran más urgentes y que acabarían formando parte del primer Sprint. Tras ello, el resto de historias fueron ordenadas en base a criterios propios del *Scrum Manager* y del Equipo de Desarrollo, priorizando aquellas que se pensaba que aportaban más al usuario final.

Identificador	Descripción	Estimación (puntos de historia)
TA01	*Creación de la base de datos.*	-
TA02	*Diseño interfaces.*	-
HU01	*Como aprendiz.* *Quiero poder pedir feedback de los expertos utilizando categoría, título y descripción.* *Para poder mejorar en esa categoría.*	7ptos
HU02	*Como experto.* *Quiero poder tener un listado con las peticiones de feedback.* *Para poder ayudar a los usuarios que lo necesiten.*	4ptos
HU03	*Como administrador.* *Quiero poder crear categorías y subcategorías.* *Para que los usuarios y expertos puedan recibir y dar feedback.*	2ptos
HU04	*Como experto.* *Quiero poder marcar las categorías como favoritas.* *Para poder consultar con mayor celeridad las categorías que más me gustan o a las cuales soy más afín.*	3ptos
HU05	*Como experto.* *Quiero poder dar feedback a publicaciones de diferentes categorías.* *Para poder ayudar a los usuarios que necesitan ser retroalimentados.*	7ptos
HU06	*Como usuario.* *Quiero poder consultar mi historial sobre las publicaciones realizadas.* *Para poder consultarlas en un momento dado.*	4ptos
HU07	*Como usuario.* *Quiero poder consultar mis datos.* *Para poder visualizarlos en un momento dado.*	4ptos
HU08	*Como aprendiz.* *Quiero poder ver todas las categorías.* *Para poder iniciar el proceso de pedir feedback.*	2ptos
HU09	*Como experto.* *Quiero poder ver todas las categorías.* *Para poder iniciar el proceso de dar feedback.*	2ptos
HU15	*Como usuario.* *Quiero poder marcar una petición como indebida.* *Para evitar conflictos entre usuarios o en la propia aplicación.*	3ptos
HU16	*Como usuario.* *Quiero poder buscar peticiones concretas.* *Para facilitar el acceso a dicha petición.*	3ptos

Identificador	Descripción	Estimación (puntos de historia)
HU11	*Como aprendiz.* *Quiero poder dar una puntuación sobre el feedback obtenido.* *Para dar una mejor reputación a los expertos que lo merezcan*	3ptos
HU12	*Como usuario.* *Quiero poder observar un ranking de clasificación de los expertos mejor valorados.* *Para poder comprobar su reputación en la aplicación.*	4ptos
HU13	*Como usuario.* *Quiero poder observar un ranking de las categorías más activas* *Para poder tener en cuenta el tiempo aproximado en el que recibiré ayuda.*	4ptos
HU14	*Como usuario.* *Quiero poder observar un ranking de los expertos más activos.* *Para poder tener en cuenta el tiempo aproximado en el que recibiré ayuda.*	4ptos
HU10	*Como usuario.* *Quiero poder hacer sugerencias mediante un buzón.* *Para solicitar nuevas categorías.*	2ptos
HU17	*Como usuario.* *Quiero poder registrarme en la aplicación.* *Para poder usarla.*	4ptos
HU18	*Como usuario* *Quiero poder modificar mis datos.* *Para poder adaptarlos a las necesidades del momento.*	3ptos
HU19	*Como usuario* *Quiero poder borrar mis datos.* *Para poder dejar de formar parte de la aplicación.*	2ptos

Cuadro 2.1: Product backlog con las historias priorizadas y estimadas.

2.3. Identificación y gestión de riesgos

Una de las partes importantes a la hora de planificar un proyecto es la identificación y gestión de riesgos, la cual tiene como principal objetivo prever posibles problemas potenciales que puedan surgir durante el desarrollo, así como tratar de minimizar su impacto en el proyecto. Es por ello que en este proyecto se realizó una identificación de los riesgos y un posterior análisis, especificando la forma de prevenirlos y que acciones se tomarían en caso de que se manifestasen durante el desarrollo.

En el Cuadro 2.2 se pueden observar los riesgos potenciales que se identificaron, junto con el tipo de riesgo especifico de cada uno de ellos, estos son:

- Riesgo del producto: Referente a un riesgo relacionado con las tecnologías elegidas para el proyecto.
- Riesgo del proyecto: Relacionado con la organización, gestión y desarrollo general del proyecto.

ID	Descripción del riesgo	Tipo de riesgo
R01	Falta de experiencia con las herramientas utilizadas	Riesgo del producto
R02	Abandono temporal de un miembro del equipo	Riesgo del proyecto
R03	Conflictos entre los integrantes del equipo.	Riesgo del proyecto
R04	Diseño erróneo	Riesgo del producto
R05	Restricciones tecnológicas	Riesgo del producto
R06	Falta de tiempo o inexperiencia al gestionarlo.	Riesgo del proyecto
R07	Inestabilidad en el lugar de desarrollo.	Riesgo del proyecto

Cuadro 2.2: Identificación de los riesgos.

Una vez identificados todos los riesgos, en el Cuadro 2.3 se puede visualizar el análisis que se ha realizado de los riesgos, para cada uno de los riesgos se han descrito los siguientes puntos:

- *Magnitud*: Cuanto va a afectar ese riesgo en caso de que se produzca.
- *Descripción*: Una explicación más detallada sobre lo que comporta ese riesgo y sobre que afecta.
- *Impacto*: Sobre que elementos del proyecto va a afectar dicho riesgo (Ej: tiempo, coste, etc).
- *Indicadores*: Cual es la señal la cual indica que se está manifestando el riesgo.

Finalmente, en el Cuadro 2.4 se observan las medidas que se tomaron para prevenir la aparición de los riesgos durante el desarrollo (Plan de prevención/Acción) y también de que forma se minimizarían las consecuencias si aparecieran (Plan de corrección/contingencia).

ID	Análisis del Riesgo
R01	*Magnitud*: Media durante todo el desarrollo y en cada uno de los Sprints. *Descripción*: El equipo no tiene experiencia previa utilizando las herramientas con las que se desarrolla el producto lo cual puede suponer una dificultad a la hora de alcanzar los objetivos establecidos. *Impacto*: Retraso en la entrega del proyecto y/o coste adicional. *Indicadores*: El desarrollo real del proyecto no se corresponde con la planificación realizada.
R02	*Magnitud*: Alta, cuando afecta a un solo miembro. Muy alta, si afecta a más de uno. *Descripción*: Algún miembro del proyecto no se encuentra disponible por cualquier problema. Dada la situación actual, la baja por enfermedad puede darse de manera más habitual. *Impacto*: La falta de disponibilidad de los recursos humanos puede provocar retrasos en la finalización completa de los diferentes Sprints y en las historias en las que se encuentran involucrados. Teniendo en cuenta que la entrega no puede posponerse, la falta de disponibilidad de personal puede suponer una pérdida de calidad en el producto o un incremento en el coste. *Indicadores*: No procede.
R03	*Magnitud*: Media. *Descripción*: Aparición de problemas y discrepancias entre los miembros del proyecto. Falta de acuerdo en las decisiones tomadas. *Impacto*: Si los desacuerdos no son rápidamente resueltos se pueden provocar retrasos en la ejecución de los Sprints. Estos retrasos podrían suponer que el proyecto no se completara y que el trabajo de los Sprints no fuera completado. *Indicadores*: Mucho tiempo dedicado a decisiones concretas, énfasis en las posturas enfrentadas, número de enfrentamientos con respecto a una misma decisión.
R04	*Magnitud*: Alta, sobretodo en los Sprints finales *Descripción*: La interfaz de usuario no satisface los deseos del Product Owner o los usuarios no son capaces de interactuar adecuadamente con ella, o la describen como una interfaz muy complicada o extraña. *Impacto*: Puede suponer retrasos ya que se debería de rehacer la interfaz de usuario. *Indicadores*: Los resultados de las pruebas de usuario no son satisfactorios. El *Product Owner* nos comunica que no está de acuerdo con la interfaz de usuario.

ID	Análisis del Riesgo
R05	*Magnitud*: Media. *Descripción*: Debido a las restricciones a la hora de utilizar ciertas herramientas en el proyecto, se puede ver limitada la capacidad de flexibilizarlas y utilizar otras tecnologías que faciliten la realización de ciertas partes del proyecto. *Impacto*: Puede suponer retrasos a la hora de terminar el Sprint o si se encuentra en las últimas fases del proyecto puede existir un retraso temporal que incluya un coste adicional. *Indicadores*: Complicaciones especiales a la hora de implementar una característica del sistema.
R06	*Magnitud*: Alta sobretodo en los Sprints finales del proyecto *Descripción*: Saber gestionar el tiempo es clave a la hora de realizar un proyecto ya que en caso de que se haga una mala gestión o que no haya tiempo suficiente puede suponer una gran cantidad de errores que se dejan sin cubrir o incluso la imposibilidad de poder finalizar el producto *Impacto*: El tiempo es oro, y por eso mismo, ser incapaces de ajustarse al tiempo puede suponer grandes retrasos, que impliquen una monetización adicional que no se puede obtener. *Indicadores*: La resolución de las tareas no se ajusta a la gestión del tiempo establecida.
R07	*Magnitud*: Media durante todo el desarrollo. *Descripción*: El grupo tiene dificultades para poder trabajar en su lugar de trabajo. *Impacto*: La imposibilidad de trabajar en un entorno adecuado, puede suponer grandes retrasos. *Indicadores*: No procede.

Cuadro 2.3: Descripción de riesgos.

ID	Plan de prevención/Acción	Plan corrección /contingencia
R01	Realizar cursos de formación sobre las herramientas que se van a utilizar a todos los miembros del equipo.	Contactar con los expertos en la materia de la empresa para que ayuden a los miembros que están teniendo problemas.

ID	Plan de prevención/Acción	Plan corrección /contingencia
R02	Tener un margen temporal en cada sprint del proyecto para que la pausa temporal del trabajo no afecte al desarrollo del proyecto.	Buscar otras formas de trabajo para que el desarrollo no se detenga por dicho problema. Ej: Teletrabajo. En caso de que no fuera posible, se trataría de realizar tareas que no necesiten al otro miembro del equipo.
R03	Cada vez que se fije un punto de dirección en el proyecto, todo tiene que quedar totalmente claro, sin dudas y con la aceptación total de todos los miembros del grupo.	Se establecen las siguientes reglas para definir una política de toma de decisiones en caso de desacuerdo. Las cuestiones relativas a las historias de usuario se tratarán junto al Product Owner, que será quien tome la decisión. Las cuestiones de diseño o técnicas se tratarán en conjunto entre los dos equipos de desarrollo, donde cada uno aportará su opinión.
R04	Tratar de consensuar un diseño con el Product Owner y realizar múltiples tests de usuario cada ciertas iteraciones como forma de prevención.	Reunión entre todos los miembros de desarrollo, incluido el Product Owner, para consensuar las correcciones necesarias en la interfaz.
R05	Asegurarse de que el equipo se familiariza con las herramientas y corrobora su potencial antes de empezar a trabajar.	Contactar con los expertos en la materia de la empresa para que ayuden a los miembros que están teniendo problemas.
R06	Realizar un estudio sobre proyectos de calibre similar para poder realizar una gestión del tiempo más precisa.	Redefinir los tiempos conforme se va desarrollando el proyecto, para poder ser capaces de dar información más precisa sobre el tiempo restante.
R07	La inestabilidad del lugar de trabajo es imprevisible.	Medir las posibles alternativas a realizar, se podría buscar un nuevo lugar de trabajo que no suponga grandes costos adicionales. También cabe la idea de usar las funciones de teletrabajo, si los trabajadores tienen la confianza de poder trabajar adecuadamente en el entorno del hogar.

Cuadro 2.4: Prevención y corrección de cada riesgo.

2.4. Estimación de recursos y costes del proyecto

En esta sección se van a tratar los costes que han sido requeridos en el proyecto. Para ello se van a tener en cuenta el *Hardware*, el *Software* y los recursos humanos que han influido de alguna manera en el proyecto.

Dadas las características del proyecto, este cálculo se ha realizado, no solo con respecto al *Frontend móvil* que he desarrollado, sino teniendo en cuenta la totalidad del proyecto, es decir, tanto el *Backend*, como el *Frontend web* han formado parte del cálculo. Aún así, durante el progreso de cálculo se han separado los costes individuales de cada parte para posteriormente juntarlos en una tasación única.

Para tasar el coste *Hardware* de este proyecto, primeramente se han listado las diferentes herramientas utilizadas junto con el coste total de cada una de ellas (CT). Una vez obtenido este coste, se ha calculado el coste proporcional en el proyecto. Para ello, se ha fijado un tiempo de vida útil de cada uno de estos recursos (TVU). En ordenadores este tiempo ha sido fijado en 6 años, mientras que en dispositivos móviles se ha fijado en 2 años. Tras ello, se ha obtenido su coste por hora y se ha multiplicado por las horas que están previstas en este proyecto (300h) obteniendo finalmente el coste proporcional al proyecto de recursos Hardware (CP). En la Ecuación 2.1 se puede ver la fórmula utilizada y en el Cuadro 2.5, el cálculo final de todos los recursos separados en las diferentes partes del proyecto.

$$CP = (CT \div TVU \div 12 \div 30 \div 24) \times 300 \tag{2.1}$$

Proyecto	Hardware	Coste total (€)	Coste proporcional (€)
Backend	Ordenador sobremesa	2.700,00	5,21
Frontend móvil	Ordenador sobremesa	1.800,00	10,42
	Móvil OnePlus 8 Pro	700,00	12,15
Frontend web	Ordenador sobremesa	2.700,00	10,42
Total		7.900,00	38,19

Cuadro 2.5: Costes de hardware del proyecto

Tras el cálculo de costes *Hardware*, se ha proseguido calculando el coste Software del proyecto. Para ello, al igual que en el de *Hardware*, se han listado todas las herramientas utilizadas y su coste individual. Al ser la mayoría de uso gratuito, su coste ha sido nulo. En el Cuadro 2.6 se pueden observar estos costes separados en las diferentes partes del proyecto. Como se puede visualizar el coste final ha sido meramente anecdótico.

Proyecto	Software	Coste total (€)
Común	Jira Software	0,00
Backend	PHPStorm	0,00
	AWS	1,00
	Postman	0,00
	Heroku	0,00
Frontend móvil	Android Studio	0,00
	Balsamiq Frameworks Free Trial	0,00
Frontend web	Visual Studio Code	0,00
	Heroku	0,00
Total		1,00

Cuadro 2.6: Costes de software del proyecto

Una vez obtenido el coste *Hardware* y *Software*, se ha calculado el coste de los desarrolladores del proyecto. Para ello, primeramente se ha obtenido el sueldo anual (SA) de cada uno de los integrantes del proyecto. Ha sido obtenido a partir de lo que cobra un Programador Junior en Cuatroochenta [5]. A partir de este cálculo, se ha obtenido el sueldo por hora (SH) de trabajo teniendo en cuenta las semanas que tiene un año (52) y las horas de trabajo por semana (40). La fórmula utilizada se puede observar en la Ecuación 2.2.

$$SH = SA \div 52 \div 40 \tag{2.2}$$

Tras ello, se ha obtenido el coste proporcional (SP) a las horas empleadas en el proyecto, dado que trabajábamos a media jornada (4h). El sueldo hora ha sido dividido entre dos, para ajustarse a la realidad y posteriormente, se ha obtenido el resultado teniendo en cuenta las horas dedicadas al proyecto (300h) como se puede observar en la Ecuación 2.3. Los resultados obtenidos se pueden visualizar en el Cuadro 2.7 donde se encuentran los roles, el sueldo anual correspondiente a cada rol, el sueldo por hora de cada rol y su sueldo proporcional en el proyecto.

$$SP = SH \div 2 \times 300 \tag{2.3}$$

Roles	Sueldo Anual (€)	Sueldo hora (€)	Sueldo Proporcional (€)
Programador 1	16000	7,69	1153,85
Programador 2	16000	7,69	1153,85
Total	32000	15,38	2307,69

Cuadro 2.7: Costes de humanos del proyecto

Finalmente, se han sumado los costes de *Hardware* (H), *Software* (S) y Humanos (HH). Una vez obtenido el cálculo completo, se han añadido los costes indirectos asociados al proyecto, los cuales se han estimado en un 20 % del coste total. Tras ello, se ha obtenido la estimación del coste total del proyecto (CTP) y el coste por hora del mismo. En la Ecuación 2.4 se puede observar la fórmula utilizada para este cálculo y en el Cuadro 2.8 se visualizan los costes individuales de cada parte y la suma total.

$$CTP = (H + S + HH) \times 1{,}20 \tag{2.4}$$

	Presupuesto final (€)
Hardware	38,19
Software	0,00
Sueldo	2.346,89
Costes Indirectos (20 %)	469,38
Total	2.816,26

Cuadro 2.8: Costes finales del proyecto

2.5. Seguimiento del proyecto

En esta sección se va a describir el progreso del proyecto a lo largo de los diferentes Sprints recogiendo los comentarios de las reuniones de cada Sprint, así como problemas y puntos clave que hayan podido afectar de alguna manera al desarrollo.

2.5.1. Primer Sprint

Reunión de inicio

En la reunión de inicio del Sprint se definió el objetivo del sprint el cual consistía en la creación y visualización de las categorías y publicaciones, así como la creación de nuevos *feedbacks* sobre una publicación.

Tras ello se definieron las historias y tareas que formarían parte del sprint las cuales se pueden observar en el cuadro 2.9.

Problemas y puntos clave durante el Sprint

Una vez terminada la reunión se comenzó a trabajar en el Sprint. Se inició trabajando conjuntamente con el *Backend* para definir tanto la base de datos del proyecto como el diseño de las diferentes interfaces del proyecto haciendo uso de *Balsamic Frameworks* para esta última tarea.

Una vez terminadas estas tareas, se prosiguió con la primera de las historias, la cual consistía en poder hacer publicaciones de una determinada categoría. Esta fue una de las historias que más se demoró en el tiempo, aproximadamente 26 horas, mucho más de lo que se tenia previsto. Los motivos para esta demora fueron varios:

- El comienzo del proyecto: Dado que era la primera historia que se realizaba, hubo mucho trabajo previo que, si bien no tenia implicación directa en la realización de la historia, si que retraso su comienzo (Creación de ciertas clases de utilidad, configuración de *Gradle*, etc). Tal vez una manera de solucionar este problema hubiera sido tener en cuenta esta construcción de la estructura a través de una tarea, así, esta parte no habría formado parte de la primera de las historias.
- Subida de archivos: Una de las subtareas que contenía esta historia era la posibilidad de subir archivos junto con el resto de campos. Para poder subir los archivos al servidor se utilizó el tipo de archivo *multipart/form-data*. La librería que se utilizó durante todo el proyecto para hacer peticiones al servidor no servía para enviar ese tipo de archivos, eso junto con la poca documentación que existía para poder hacer uso de archivos *multipart/form-data* provocó una demora importante en el Sprint. La solución a este problema fue utilizar la librería *Fuel* [21] que, a pesar de su poca documentación, era muy sencilla de utilizar.

Identificador	Descripción	Estimación (puntos de historia)
TA01	Creación de la base de datos.	-
TA02	Diseño interfaces.	-
HU01	*Como* aprendiz. *Quiero* poder pedir *feedback* de los expertos utilizando categoría, título y descripción. *Para* poder mejorar en esa categoría.	7ptos
HU02	*Como* experto. *Quiero* poder tener un listado con las peticiones de *feedback*. *Para* poder ayudar a los usuarios que lo necesiten.	4ptos
HU03	*Como* administrador. *Quiero* poder crear categorías y subcategorías. *Para* que los usuarios y expertos puedan recibir y dar *feedback*.	2ptos
HU04	*Como* experto. *Quiero* poder marcar las categorías como favoritas. *Para* poder consultar con mayor celeridad las categorías que más me gustan o a las cuales soy más afín.	3ptos
HU05	*Como* experto. *Quiero* poder dar *feedback* a publicaciones de diferentes categorías. *Para* poder ayudar a los usuarios que necesitan ser retroalimentados.	7ptos
HU06	*Como* usuario. *Quiero* poder consultar mi historial sobre las publicaciones realizadas. *Para* poder consultarlas en un momento dado.	4ptos

Cuadro 2.9: Historias del primer Sprint

Otro problema que se encontró durante el desarrollo de este Sprint fue la navegación entre pantallas, la cual en un primer momento se realizaba de una manera y más tarde se realizó de otra. El utilizar diferentes métodos para pasar entre pantallas provocó fallos como por ejemplo no poder volver a la pagina anterior o botones que desaparecían en el paso entre paginas. El problema fue solucionado unificando la forma de navegar entre pantallas, eligiendo finalmente la que recomienda Android, *Navigation Component* [9].

El resto del Sprint se realizó sin problemas significativos aunque debido a todos los problemas que se han explicado, no se pudieron completar todas las historias que estaban previstas.

Reunión de cierre

Finalmente, en la reunión de cierre del Sprint (Sprint review) se hizo un repaso de lo ocurrido durante el Sprint. Como se puede observar en la Figura 2.1, no se acabaron las historias de las cuales estaba planeado el Sprint debido principalmente a los problemas especificados en el apartado anterior. La linea de progreso del Sprint (en rojo) no comenzó a descender hasta mediados de la segunda semana, cuando se completó la primera de las historias. Aunque en los últimos días del Sprint se finalizaron varias historias y el ritmo de progreso se aceleró, no fue suficiente para acabar totalmente el Sprint y quedaron pendientes las siguientes dos historias:

- HU03 - Como administrador quiero poder crear categorías y subcategorías para que los usuarios y expertos puedan recibir y dar *feedback*.
- HU06 - Como usuario quiero poder consultar mi historial sobre las publicaciones realizadas con la finalidad de poder consultarlas en un momento dado.

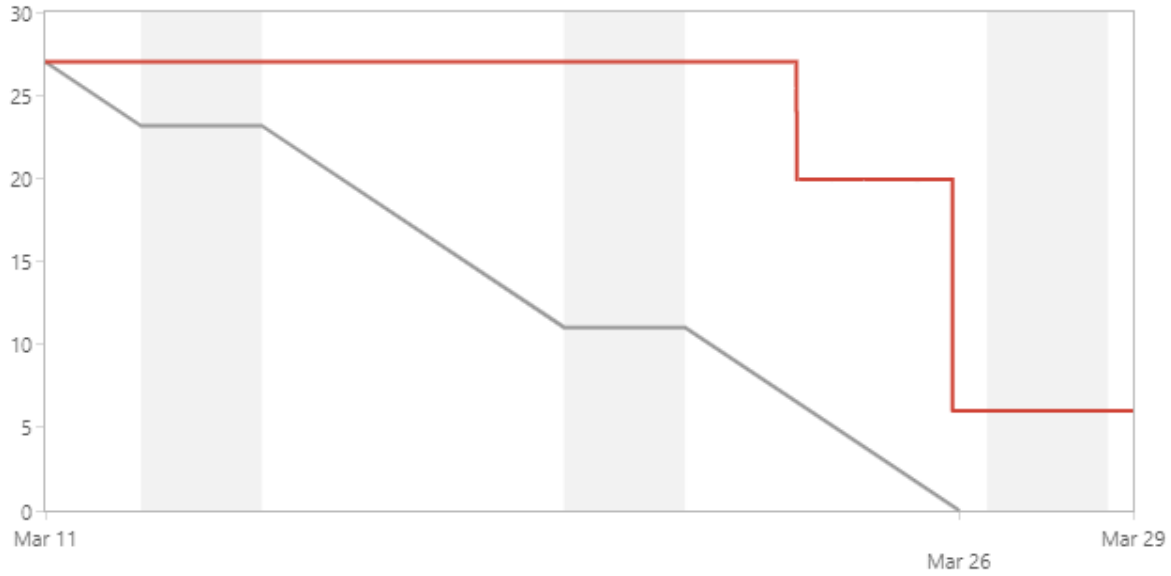


Figura 2.1: Grafico Burn Down del primer Sprint

Tras comentar esto, se realizó una pequeña demo al *Product Owner* para que se revisara el progreso. Este, evidenció un problema a la hora de crear una publicación, y es que cuando había que elegir la categoría en la cual se publicaría, la solución que se había utilizado no era óptima. El sistema consistía dos *Spinners* en los cuales al elegir una categoría en el primero de ellos, se diera la opción de elegir la subcategoría en el segundo. Ejemplo: En el primer *Spinner* se elige Deportes y entonces en el segundo aparece una segunda lista formada por Tenis, Fútbol o Baloncesto entre otros. Esta solución resultó ser bastante confusa, sobretodo porqué la actualización del segundo *Spinner* no era inmediata. Es por ello que al final se pensó en cambiar esa parte de la *app* por un sistema de pantallas clásico en el cual, para elegir la categoría, llevara al usuario a otra pantalla distinta a la de crear publicaciones. Este nuevo diseño fue aprobado por el *Product Owner*.

2.5.2. Segundo Sprint

Reunión de inicio

Antes de dar comienzo al segundo Sprint, se hizo repaso al Product Backlog y juntamente con el equipo de desarrollo, el Scrum Manager y el Product Owner se decidieron añadir dos historias más. La primera de ellas relacionada con el acceso a la aplicación, dado que al haber diferentes roles y usuarios, la app requería de una identificación individual. Además, se decidió que era prioritaria su implementación en la aplicación para que cada usuario ya pudiera realizar en la aplicación las tareas correspondientes a su rol. La otra de las historias estaba relacionada con la visualización en detalle de cada una de las publicaciones, para poder mostrar la lista de *feedbacks* recibidos y de archivos sin sobrecargar la lista de publicaciones.

En cuanto al objetivo del Sprint, se estableció que los usuarios pudieran visualizar sus datos y sus acciones en la aplicación, así como que pudieran identificarse y acceder a ella. Tras ello se decidieron las historias que formarian parte de este segundo Sprint, incluyendo las historias que no se habian completado en el anterior. En el cuadro 2.10 se puede observar toda esta información, las historias del Sprint (amarillo) y las nuevas añadidas (*).

Finalmente se especificó la duración de este Sprint, que aunque estaba planeado que durase dos semanas, se aumento su duración en cuatro días como consecuencia a los días festivos que habían durante esas dos semanas.

Identificador	Descripción	Estimación (puntos de historia)
HU03	*Como administrador.* *Quiero poder crear categorías y subcategorías.* *Para que los usuarios y expertos puedan recibir y dar feedback.*	2ptos
HU06	*Como usuario.* *Quiero poder consultar mi historial sobre las publicaciones realizadas.* *Para poder consultarlas en un momento dado.*	4ptos

Identificador	Descripción	Estimación (puntos de historia)
HU07	*Como usuario.* *Quiero poder consultar mis datos.* *Para poder visualizarlos en un momento dado.*	4ptos
HU08	*Como aprendiz.* *Quiero poder ver todas las categorías.* *Para poder iniciar el proceso de pedir feedback.*	2ptos
HU09	*Como experto.* *Quiero poder ver todas las categorías.* *Para poder iniciar el proceso de dar feedback.*	2ptos
HU15	*Como usuario.* *Quiero poder marcar una petición como indebida.* *Para evitar conflictos entre usuarios o en la propia aplicación.*	3ptos
HU20*	*Como usuario* *Quiero poder acceder a la aplicación* *Para poder utilizar todas sus funcionalidades*	2ptos
HU16	*Como usuario.* *Quiero poder buscar peticiones concretas.* *Para facilitar el acceso a dicha petición.*	3ptos
HU11	*Como aprendiz.* *Quiero poder dar una puntuación sobre el feedback obtenido.* *Para dar una mejor reputación a los expertos que lo merezcan*	3ptos
HU12	*Como usuario.* *Quiero poder observar un ranking de clasificación de los expertos mejor valorados.* *Para poder comprobar su reputación en la aplicación.*	4ptos
HU13	*Como usuario.* *Quiero poder observar un ranking de las categorías más activas* *Para poder tener en cuenta el tiempo aproximado en el que recibiré ayuda.*	4ptos
HU14	*Como usuario.* *Quiero poder observar un ranking de los expertos más activos.* *Para poder tener en cuenta el tiempo aproximado en el que recibiré ayuda.*	4ptos
HU21*	*Como usuario* *Quiero poder ver las publicaciones en detalle* *Para poder observar cual es el contenido de una publicación en concreto.*	5ptos
HU10	*Como usuario.* *Quiero poder hacer sugerencias mediante un buzón.* *Para solicitar nuevas categorías.*	2ptos

Identificador	Descripción	Estimación (puntos de historia)
HU17	*Como usuario.* *Quiero poder registrarme en la aplicación.* *Para poder usarla.*	4ptos
HU18	*Como usuario* *Quiero poder modificar mis datos.* *Para poder adaptarlos a las necesidades del momento.*	3ptos
HU19	*Como usuario* *Quiero poder borrar mis datos.* *Para poder dejar de formar parte de la aplicación.*	2ptos

Cuadro 2.10: Product backlog a fecha del 29 de marzo de 2021 junto con las historias del segundo Sprint (amarillo) y las nuevas historias introducidas (*)

Problemas y puntos clave durante el Sprint

El desarrollo de las historias correspondientes a este Sprint fue mucho más rápido de lo esperado y no se encontró ningún problema grave que detuviera el desarrollo del proyecto. Esto provocó que el trabajo previsto para este Sprint se acabara una semana antes. Para seguir trabajando en el proyecto se tomó la decisión de introducir otras cuatro historias en el Sprint, las cuales fueron:

- HU12 - Como usuario quiero poder observar un ranking de clasificación de los expertos mejor valorados para poder comprobar su reputación en la aplicación.
- HU13 - Como usuario quiero poder observar un ranking de las categorías más activas para poder tener en cuenta el tiempo aproximado en el que recibiré ayuda.
- HU14 - Como usuario quiero poder observar un ranking de los expertos más activos para poder tener en cuenta el tiempo aproximado en el que recibiré ayuda.
- HU16 - Como usuario quiero poder buscar peticiones concretas para facilitar el acceso a dicha petición.

Dada la rapidez con la que se estaban completando las historias, durante la implementación de estas cuatro se puso más énfasis en la parte visual y de usabilidad de la aplicación. Además, también se reformularon partes del código que eran confusas o que no seguían las buenas prácticas de programación.

Reunión de cierre

En la reunión de cierre del Sprint se presentó el trabajo realizado al resto del equipo, junto con el gráfico *Burn Down* del Sprint, el cual se puede observar en la Figura 2.2. En el se puede

observar como el progreso del Sprint (en rojo) es mucho más rápido que la progresión estimada (en gris). También se puede ver como al acabar el trabajo previsto se añadieron nuevas historias para poder seguir avanzando en el desarrollo.

Más tarde, se presentó una demo al Product Owner, en la cual no hubo ninguna objeción ni sugerencia.

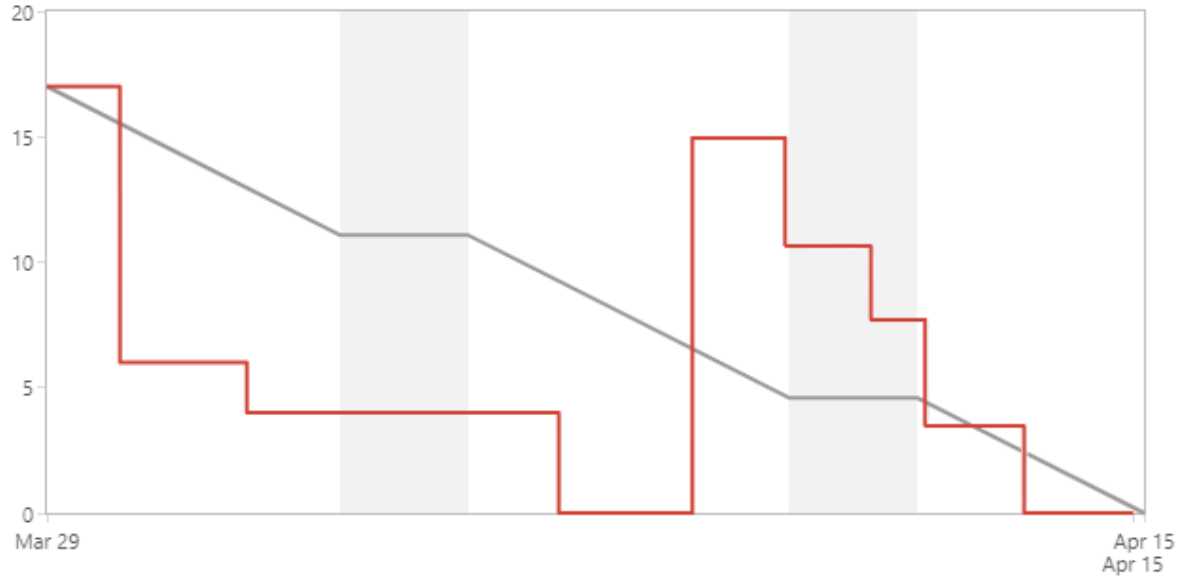


Figura 2.2: Gráfico Burn Down del segundo Sprint

2.5.3. Tercer Sprint

Reunión de inicio

En la reunión de inicio primeramente se fijaron los objetivos de este tercer Sprint, los cuales consistían en la vista en detalle de las publicaciones y *feedbacks* , junto con todo lo relacionado con la gestión de datos por parte del usuario.

Como consecuencia del gran progreso realizado en el segundo Sprint en lo que respecta a las funcionalidades de la aplicación, las historias que se incluyeron en este Sprint fueron todas las restantes del *product backlog*. Es por ello que también se empezó a pensar en posibles funcionalidades extra que añadir en los siguientes Sprints, para así poder continuar trabajando hasta alcanzar las horas previstas de este proyecto.

Junto con las historias restantes, también se decidió añadir dos tareas más. La primera de ellas se había planteado ya en anteriores reuniones, pero finalmente se formalizó en esta última. El principal objetivo de realizar esta tarea fue el de evitar la sobrecarga de memoria al guardar una lista de muchos elementos. Para evitar este problema la mejor opción era la de introducir una paginación. La segunda de estas tareas consistía en gestionar correctamente los

errores provenientes de una petición al servidor, dado que hasta este momento su gestión era muy simple y podía llevar a errores. En el cuadro 2.11 se pueden observar las historias y tareas correspondientes a este segundo Sprint.

Durante el transcurso de la reunión surgieron dudas de como organizar las pantallas de detalles de las publicaciones y los *feedbacks* . Se propusieron dos opciones:

- Poner tanto los detalles de la publicación como los detalles de los *feedbacks* en la misma pantalla.
- Dividir esa información en dos pantallas. En la primera de ellas se mostraría los detalles de la publicación junto con la información simplificada de los *feedbacks* y al pulsar sobre un *feedback* especifico te llevaría a sus detalles.

La opción elegida fue esta última debido a que la primera de ellas podía provocar sobrecarga visual en la pantalla. Tras ello se dio comienzo al Sprint.

Identificador	Descripción	Estimación (puntos de historia)
HU11	*Como aprendiz.* *Quiero poder dar una puntuación sobre el feedback obtenido.* *Para dar una mejor reputación a los expertos que lo merezcan*	3ptos
HU21	*Como usuario* *Quiero poder ver las publicaciones en detalle* *Para poder observar cual es el contenido de una publicación en concreto.*	5ptos
HU10	*Como usuario.* *Quiero poder hacer sugerencias mediante un buzón.* *Para solicitar nuevas categorías.*	2ptos
HU17	*Como usuario.* *Quiero poder registrarme en la aplicación.* *Para poder usarla.*	4ptos
HU18	*Como usuario* *Quiero poder modificar mis datos.* *Para poder adaptarlos a las necesidades del momento.*	3ptos
HU19	*Como usuario* *Quiero poder borrar mis datos.* *Para poder dejar de formar parte de la aplicación.*	2ptos
TA03	Realizar paginación en las listas	-
TA04	Depuración y gestión de errores	-

Cuadro 2.11: Historias del tercer Sprint

Problemas y puntos clave durante el Sprint

Durante los primeros días, las historias se completaron muy rápidamente. No fue hasta llegar a la historia que consistía en mostrar los datos en detalle de las publicaciones donde la progresión se frenó un poco. El principal problema que se tuvo fue a la hora de descargar archivos desde el servidor. Al igual que cuando se quería subir archivos, la documentación era escasa y se perdió mucho tiempo tratando de encontrar el correcto funcionamiento de la librería *Fuel* [21].

Tras encontrar la manera de descargar archivos, también se tuvo dificultades a la hora de almacenarlos en memoria. El problema era que al guardarlos en cache, no se conseguía acceder a ese directorio sin que la aplicación se cerrase o la imagen no se abriese correctamente. Finalmente se encontró la configuración óptima para que se pudieran visualizar los archivos modificando el *Manifest*[1] y ajustando la manera de realizar el *Intent*[2] en el sistema.

Otro problema que surgió fue a la hora de realizar la paginación. Android ofrece una librería de realizar la paginación de forma nativa, el problema es que está muy ligada a la utilización de la *Clean Architecture* de Android, la cual no se siguió íntegramente en este proyecto debido a su curva de aprendizaje pronunciada. Es por ello que la realización de paginación se tuvo que hacer a mano, haciéndola una tarea más tediosa y larga.

Reunión de cierre

Durante la reunión de cierre se presentó el trabajo realizado al Product Owner y al resto del equipo. En la Figura 2.3 se puede observar la progresión del Sprint. Cabe remarcar que los últimos días del Sprint se dedicaron a hacer las tareas de gestión de errores y de paginación, y dado que a las tareas no se les asigna puntos de historia, la linea de progresión (en rojo) permaneció plana.

Durante la reunión se mostró una demo de las funcionalidades implementadas y no hubo comentarios significativos.

2.5.4. Cuarto Sprint

Reunión de inicio

Durante esta reunión se fijaron cuales iban a ser las nuevas historias que formarían parte del desarrollo, debido a que en este momento se habían completado todas las historias previstas inicialmente. Las historias que se introdujeron fueron las que se pueden encontrar en el Cuadro 2.12. También se fijó el objetivo del Sprint, el cual consistía en que los usuarios pudieran recibir notificaciones de tipo push y que pudieran hacer sugerencias más generales.

[1] Archivo de configuración del proyecto que define los parámetros principales de la aplicación. [13]
[2] Petición que se hace a alguna aplicación distinta de la que esta abierta actualmente. [11]

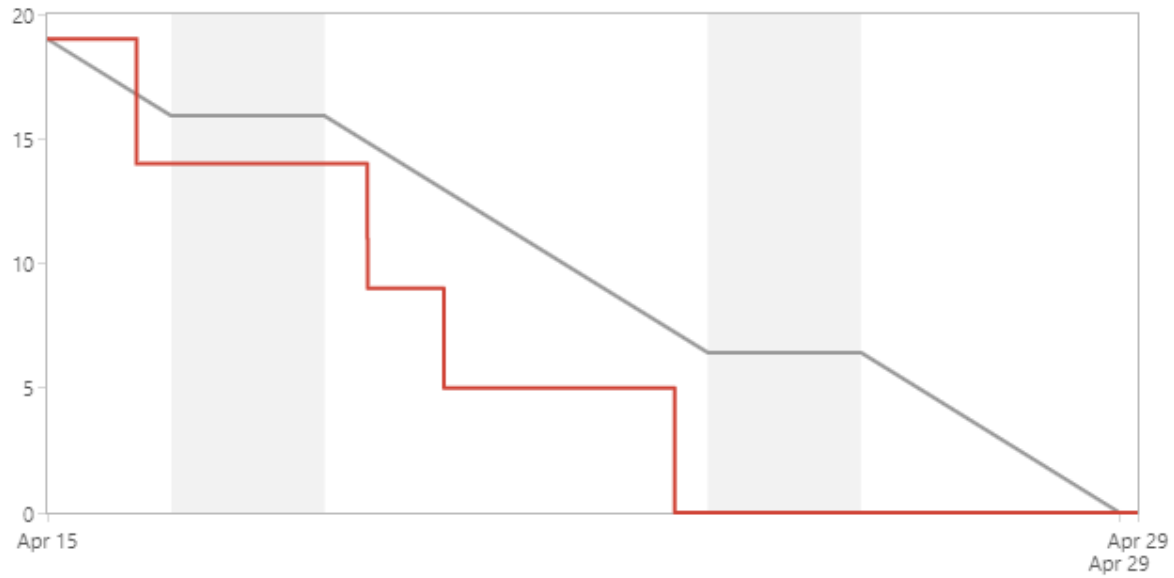


Figura 2.3: Gráfico Burn Down del tercer Sprint

Identificador	Descripción	Estimación (puntos de historia)
HU22	Como usuario Quiero poder recibir notificaciones cuando alguien interaccione conmigo Para poder enterarme y reaccionar a ello	5ptos
HU23	Como usuario Quiero poder hacer sugerencias de todo tipo Para poder mejorar la experiencia de usuario o comunicar ciertos bugs a los desarrolladores.	3ptos

Cuadro 2.12: Historias del Sprint 4

Además, y dado que se estaba finalizando el proyecto, se propusieron introducir tareas de validación para hacer tests unitarios sobre ciertas partes del software. La reunión terminó dando comienzo al Sprint.

Problemas y puntos clave durante el Sprint

Los principales problemas en este Sprint no vinieron dados por las historias que había que realizar, sino por la tarea de realizar tests unitarios. Las pruebas se querían hacer sobre los *ViewModels*. Para ello había que hacer mocks de las dependencias, pero debido a la estructura que se estaba utilizando para comunicar el modelo y el *ViewModel*, fue muy complicado llevarlo a cabo. Una de las causas por las que se manifestó este problema fue la falta de inyección de dependencias, es por ello que se estimo oportuno introducir una tarea para realizarla en un Sprint posterior.

Reunión de cierre

Durante la reunión de cierre de este Sprint se presentaron los avances en el proyecto, así como una demo completa de la aplicación. En la Figura 2.4 se puede observar la progresión del Sprint a lo largo del tiempo y como se fueron completando las dos historias que lo componían. Este gráfico es mucho más simple que en otros Sprints debido a las pocas historias de las cuales estaba compuesto. Después de esto se dio por finalizado el Sprint y el proyecto en si.

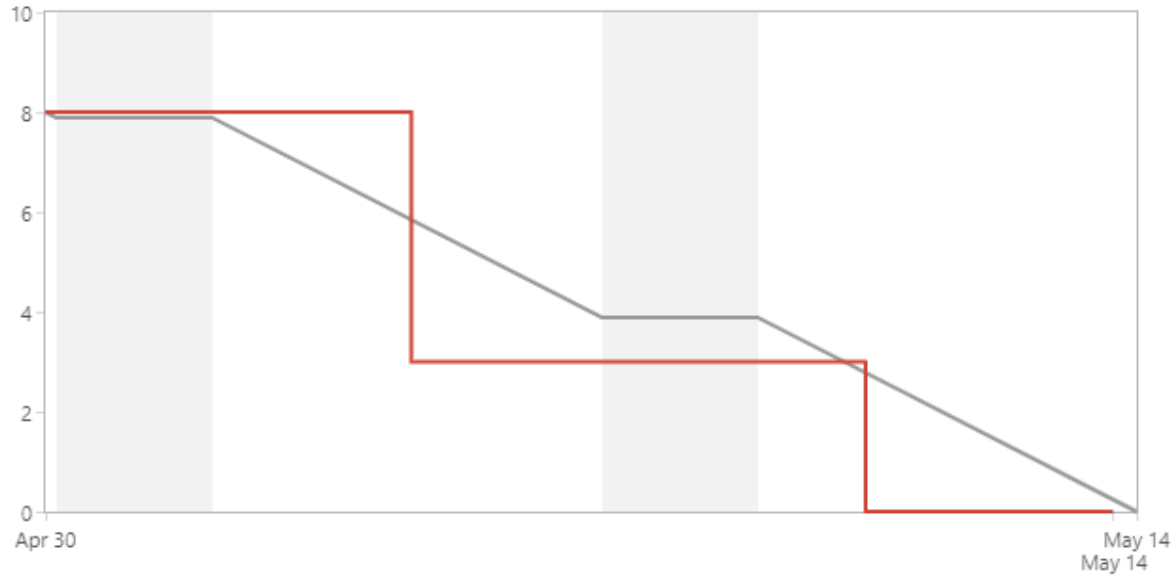


Figura 2.4: Gráfico Burn Down del cuarto Sprint

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En esta sección se van a especificar los requisitos del sistema. Para ello se realizó un diagrama de casos de uso. Este diagrama tiene la intención de mostrar como funciona el sistema desarrollado y como interactúan los actores entre las diferentes partes. Tras ello, se realizó un diagrama de actividades con el objetivo de analizar los requisitos anteriormente especificados y con tal de mostrar el flujo de las diferentes tareas a través del sistema.

3.1.1. Casos de uso

En la Figura 2.3 se puede observar el diagrama de casos de uso del proyecto, junto con sus actores y sus interacciones. Los actores han sido tres:

- Administrador: El encargado de gestionar el sistema y de realizar ciertas acciones que pueden afectar de manera indirecta al resto de actores. Además, también puede realizar interacciones con el sistema enfocadas a la visualización, como listas o datos del usuario, con el propósito de poder controlar lo que ocurre en el sistema.
- Aprendiz: Es el que realiza las acciones relacionadas con las peticiones de ayuda en una tarea y, al igual que el resto, visualiza ciertos datos y envía sugerencias a los administradores.
- Experto: Su objetivo principal es prestar ayuda a los aprendices que lo necesiten, además del resto de acciones relacionadas con el envió de ciertos mensajes al administrador y visualización de datos.

En este caso, se decidió realizar una generalización de los actores experto y aprendiz dado que habían muchas interacciones que realizaban ambos sobre una misma parte del sistema.

En cuanto a los diferentes casos de uso, se obtuvieron a partir de las historias de usuario que se especificaron en un primer momento, pero orientados más a representar las interacciones entre el sistema y los actores, en contraposición con las historias, más centradas en en el usuario y en lo que este quiere realizar en el sistema [18]. Posteriormente se realizó la especificación de los casos de uso, aquí se pueden encontrar los primeros de ellos en los cuadros 3.1 y 3.2, el resto de especificaciones se pueden encontrar en el Anexo A.

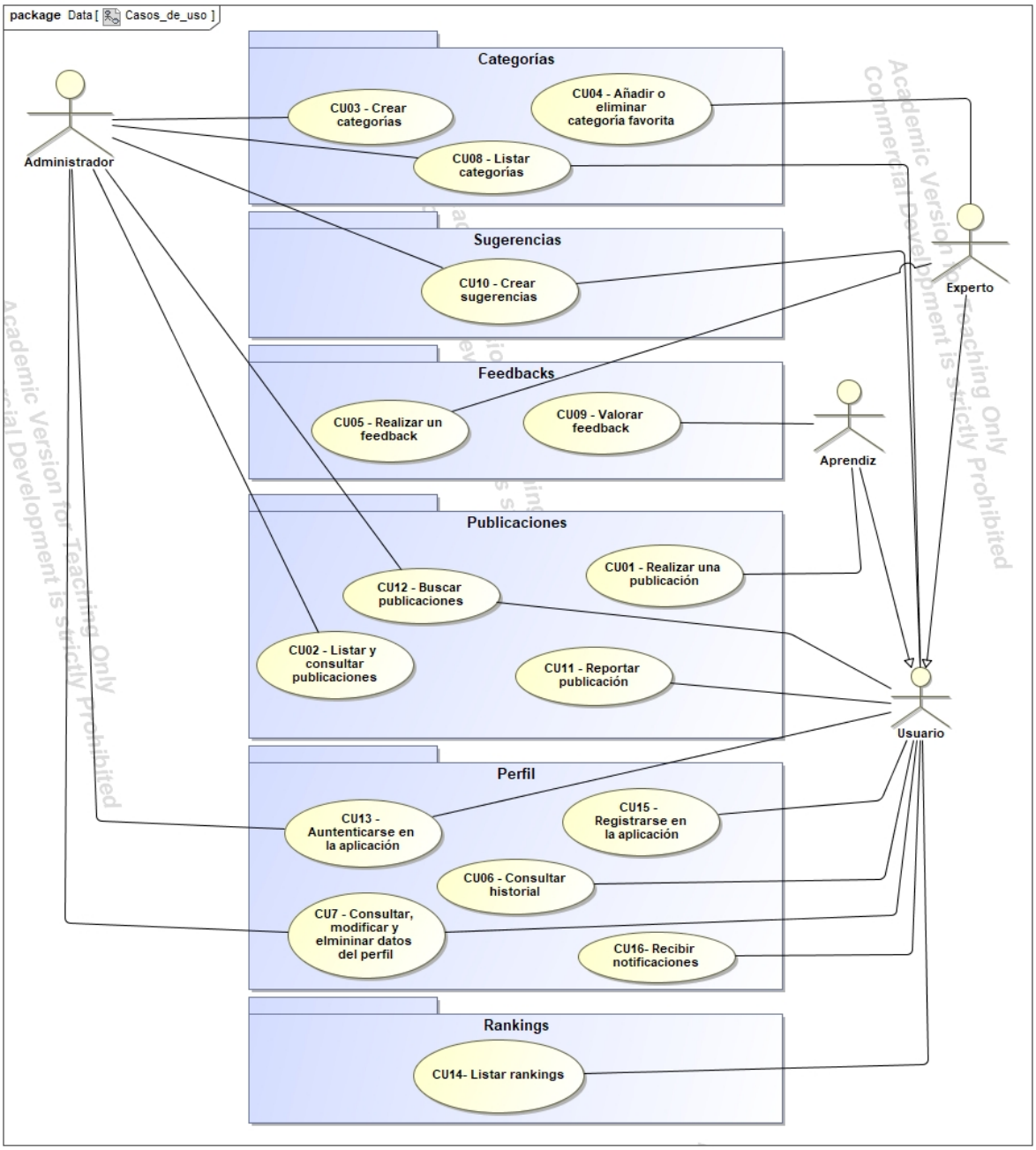


Figura 3.1: Diagrama de casos de uso del proyecto

Especificación del caso de uso	
Identificador	CU01
Nombre	Realizar una publicación.
Versión	V01
Autor	Alex Martínez Martínez
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los aprendices puedan crear una publicación para recibir feedback.
Alcance	El sistema ha de permitir que los aprendices creen una publicación a la cual puedan adjuntar archivos de vídeo, imágenes y documentos.
Nivel	Tarea principal
Actor principal	Aprendiz
Actores secundarios	N/A
Relaciones	CU02, CU11, CU12
Precondición	El aprendiz ha de estar registrado en la aplicación.
Condición fin con éxito	La publicación ha sido creada y añadida a la lista de publicaciones.
Condición fin con fracaso	La publicación no se crea y no se añade a la lista de publicaciones.
Trigger	El aprendiz desea recibir retroalimentación sobre un tema concreto.
Secuencia normal	**Acción**
1	El aprendiz se autentifica en la aplicación.
2	El aprendiz accede a la vista de crear publicación.
3	El aprendiz rellena toda la información relativa a la publicación.
4	El aprendiz adjunta archivos a la publicación.
5	El aprendiz realiza la publicación.
6	El sistema almacena la publicación y notifica al aprendiz de que esta ha sido creada satisfactoriamente.
Excepción en 5 - Categoría no encontrada	**Excepción**
1	El sistema no almacena la publicación y notifica el error.
2	El aprendiz rellena toda la información relativa a la publicación.
3	El aprendiz adjunta archivos a la publicación.
4	El aprendiz realiza la publicación.
5	El sistema almacena la publicación y notifica al aprendiz de que esta ha sido creada satisfactoriamente.
Excepción en 5 - Cancelar operación	**Excepción**
1	El aprendiz cancela la operación.
Frecuencia esperada	100 veces al día
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro 3.1: Especificación del caso de uso CU01.

Especificación del caso de uso	
Identificador	CU02
Nombre	Listar y consultar publicaciones.
Versión	V01
Autor	Carlos Mora Hernández
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los usuarios puedan listar y consultar las publicaciones creadas.
Alcance	El sistema ha de permitir que los usuarios puedan listar y consultar las publicaciones creadas, consultado tanto su información como los archivos adjuntados.
Nivel	Tarea principal
Actor principal	Experto, Aprendiz
Actores secundarios	Administrador
Relaciones	CU01, CU11, CU12
Precondición	El usuario ha de estar registrado en la aplicación.
Condición fin con éxito	Las publicaciones se listan y se puede consultar individualmente.
Condición fin con fracaso	Las publicaciones no se listan y no se puede consultar individualmente.
Trigger	El aprendiz o experto desea consultar una publicación previa para ver de que trata.
Secuencia normal	**Acción**
1	El usuario se autentifica en la aplicación.
2	El usuario accede a la vista listar publicaciones.
3	El sistema lista todas las publicaciones.
3.1	El usuario accede a una publicación concreta.
3.2	El sistema devuelve la información relativa a dicha publicación.
Excepción en 3 - Error de servidor	**Excepción**
1	El sistema no lista las publicaciones y muestra un mensaje de lista vacía.
2	El usuario accede a la vista listar publicaciones.
3	El sistema lista todas las publicaciones.
3.1	El usuario accede a una publicación concreta.
3.2	El sistema devuelve la información relativa a dicha publicación.
Excepción en 3.2 - Publicación no encontrada	**Excepción**
1	El sistema no devuelve la información sobre la publicación y notifica el error.
2	El sistema lista todas las publicaciones.
2.1	El usuario accede a una publicación concreta.
2.2	El sistema devuelve la información relativa a dicha publicación.
Frecuencia esperada	200 veces al día
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro 3.2: Especificación del caso de uso CU02.

3.1.2. Diagrama de actividades

En la Figura 3.3 se puede observar el diagrama de actividades a nivel de negocio, en el cual se ha tratado de describir un flujo de trabajo acorde con la especificación de requisitos del apartado anterior. Para realizarlo se han decidido introducir *swimlanes*[1] en las cuales se organizarán las actividades de los diferentes usuarios. Debido a que en la aplicación había muchas actividades comunes a todos los usuarios, también se decidió añadir una *swimlane* común, que ha sido llamada *Usuario general*.

Por otro lado, también se han creado algunos diagramas de actividades específicos para los casos de uso más relevantes. En la Figura 3.2 se puede observar uno de ellos, el resto se encuentran en el Anexo B.

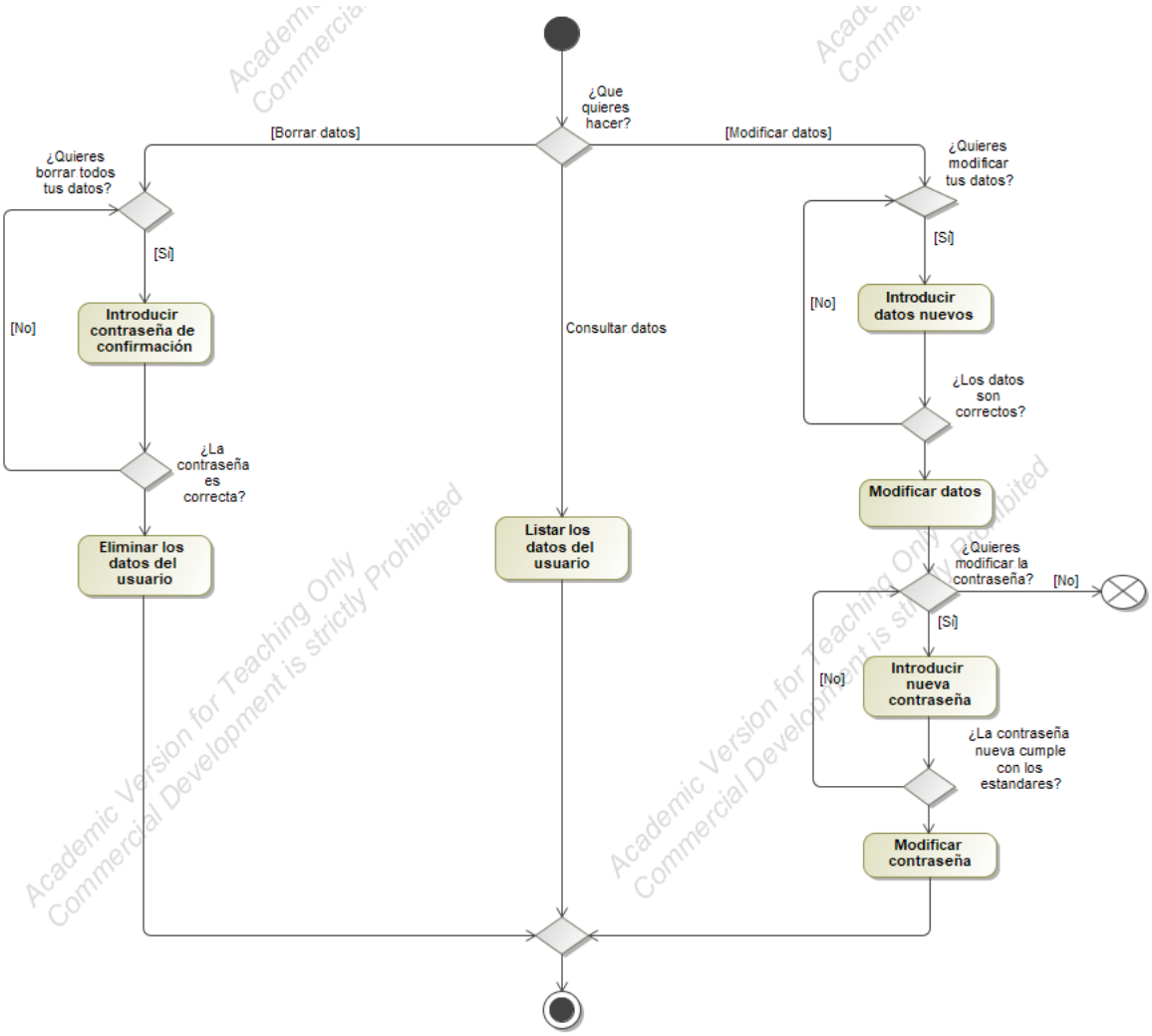


Figura 3.2: Diagrama de actividades del caso CU07.

[1]Particiones en los diagramas de actividades que tienen como propósito juntar actividades con unas características similares.

Figura 3.3: Diagrama de actividades a nivel de negocio.

3.2. Diseño de la arquitectura del sistema

En esta sección se va a profundizar en la arquitectura del sistema, la interrelación entre los diferentes componentes y ciertas características que puedan tener. La arquitectura del sistema va a estar centrada en el Frontend Móvil dado que fue la parte que desarrollé durante la estancia en practicas.

Primeramente, en el sistema se ha seguido el patrón de diseño Model-View-ViewModel (MVVM) para tratar de separar la lógica de la aplicación de la vista. Este patrón se caracteriza por eliminar la conservación de estado en la vista. El *ViewModel* se encargará ahora de conservarlo y de mantenerse sincronizado con el modelo. La vista se suscribirá a los cambios del *ViewModel* y actuará o no, en consecuencia. Esta es la arquitectura del sistema que recomienda Google y es la que se ha utilizado [14]. En la Figura 3.4 se puede observar un diagrama de la arquitectura del sistema y sus diferentes componentes.

Centrándonos en la parte del modelo, primeramente se realizan las peticiones de datos a la API REST la cual está hospedada en *Heroku* [17], a la recepción de datos, se serializan en *Data classes* que son un tipo de clases de Kotlin orientadas a la conservación de datos. Una vez convertidos en *Data Classes*, los datos llegan al *ViewModel*, el cual se encarga de conservar y de notificar a la vista de su actualización. En aras de reducir la dependencia entre clases se ha creado una capa de indirección entre el *Model* y el *ViewModel*.

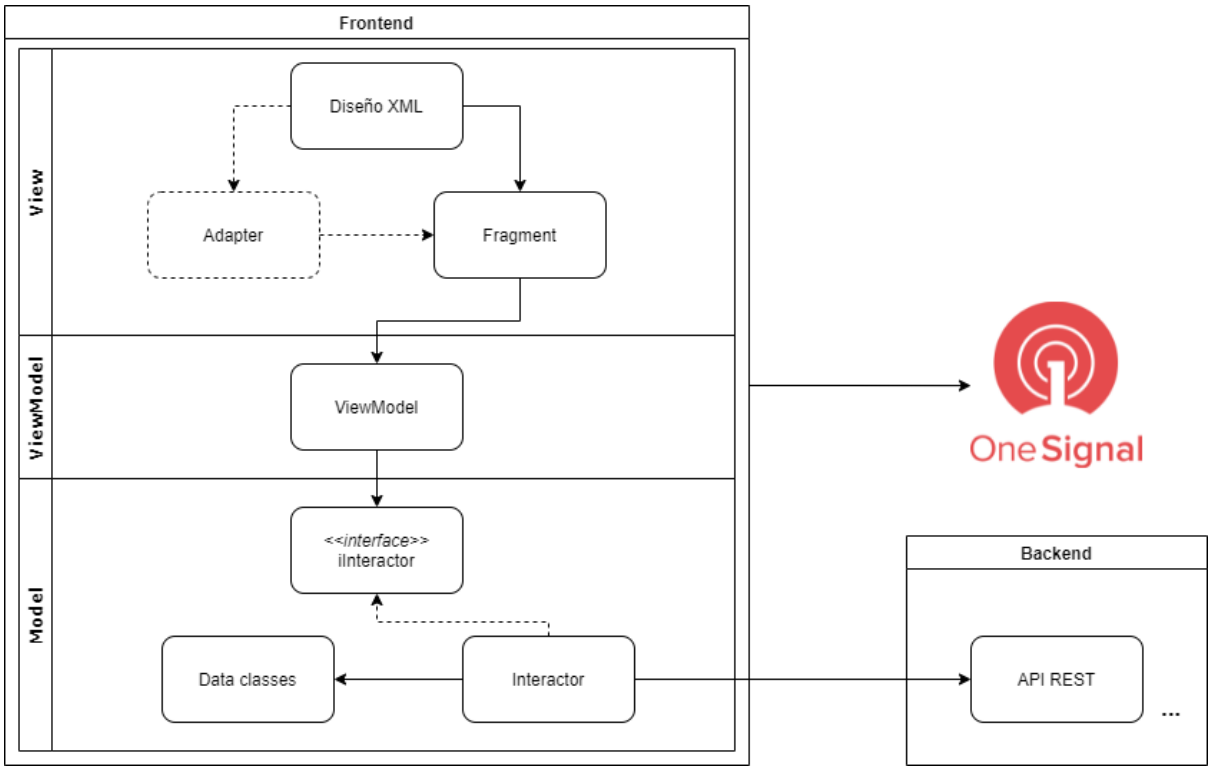


Figura 3.4: Diagrama de la arquitectura del proyecto

La vista, suscrita a los cambios en el *ViewModel* en el *Fragment* (la clase principal de la vista

y encargada de gestionar todas las acciones que realice el usuario), recibe los datos actualizados y se dispone a mostrarlos al usuario. Para mostrarlos, la vista debe decidir que tipo de datos son los que han llegado, en caso de ser una lista se requerirá del *Adapter* para mostrar los datos en la interfaz, pero si es una *Data Class* no requerirá de ningún intermediario. Este último concepto será explicado más en profundidad en el Capítulo 4. Por último, las notificaciones tipo *Push* son gestionadas por *OneSignal* [16], la aplicación móvil tan solo debe preocuparse por recibirlas y si es necesario, tomar alguna acción en cuanto lleguen.

Cabe recalcar que el sistema esta organizado en vistas, es decir, que por cada página en la aplicación existirá un *ViewModel*, un *Model* y una Vista, haciendo así más sencillo de gestionar y seguir el flujo de los datos. Como consecuencia de ello, si la página en concreto que se quiere mostrar no tiene una lista de elementos, el *Adapter* no será necesario implementarlo.

En cuanto a la gestión de errores, el encargado de realizarla es el *Model*, el cual se apoya en diferentes clases para gestionarlos y enviar una respuesta coherente al *ViewModel*, este a su vez volverá a notificar a la vista y finalmente, esta última lo mostrará al usuario para que actúe en consecuencia.

3.3. Diseño de la interfaz

En esta sección se va a describir que criterios o directrices se siguieron para realizar el diseño de la interfaz, el *Sitemap* en el cual se basó la navegación, la guía de estilo utilizada en el proyecto y finalmente, se mostraran algunos de los prototipos que se diseñaron para las diferentes interfaces.

Dado que este proyecto fue realizado íntegramente en Android, los estándares que se siguieron para el diseño de la interfaz han sido los de *Material Design* [8]. Esta es una guía que te muestra los estándares más recomendables a la hora de diseñar una interfaz, ya sea móvil o web. Además, y dado que es una propuesta de *Google*, seguir estos consejos es bastante sencillo si estas trabajando en Android, tan solo tienes que importar una dependencia en *Graddle* y a partir de ahí, tienes a tu disposición el diseño recomendado de los diferentes componentes, ya sean botones, cartas, *layouts*, etc.

Por otro lado, *Material Design* también te presenta guías para potenciar la usabilidad y la accesibilidad. Aunque seguir estas guías hace a tu aplicación mucho más diáfana y estilizada, no siempre es posible o recomendable seguirlas, dado que hay que tener en cuenta el objetivo final de tu aplicación y que se necesita mostrar en cada momento. Esto, *Material Design*, a veces no lo tiene en cuenta y seguir su guía puede resultar en una aplicación que, aunque sigue todos los estándares recomendables, es muy confusa y poco intuitiva para el usuario.

3.3.1. Guía de estilo

Como se ha mencionado anteriormente, la guía de estilo de la aplicación fue diseñada siguiendo los estándares marcados por *Material Design*.

Colores

Para la elección de colores se trató de elegir colores que tuvieran cierto contraste entre ellos pero que a su vez combinaran entre sí. Los colores se pueden diferenciar en tres grupos, el primero formado por los colores morados, el segundo por un verde azulado y el último por el negro. Esto se ha realizado así para tener colores parecidos, pero alternativos que en algunos casos podían complementarse entre sí. En la Figura 3.5 se puede observar el resultado.

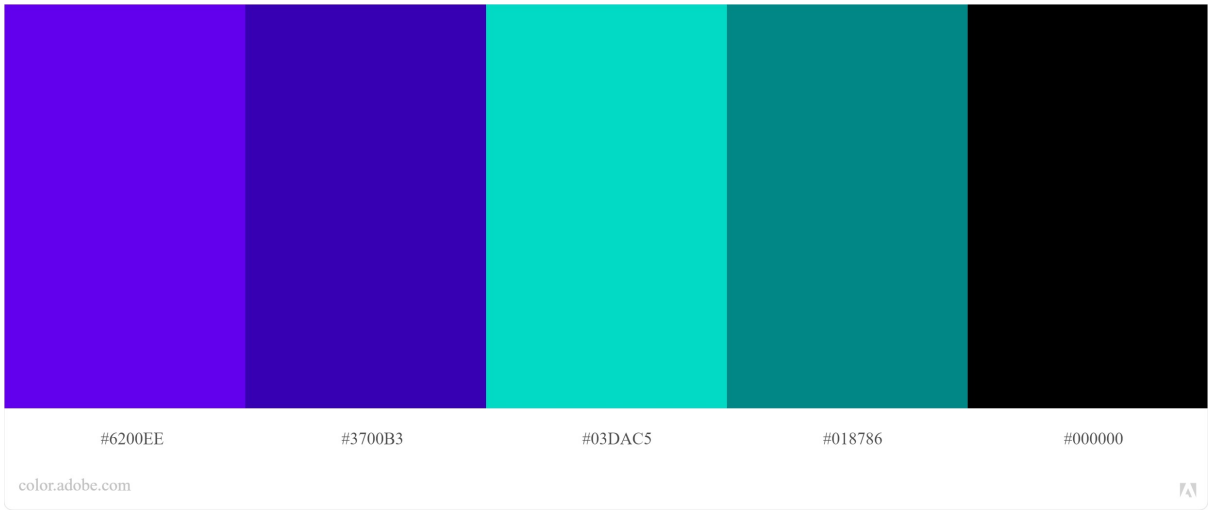


Figura 3.5: Paleta de colores utilizados

Registro utilizado

El registro utilizado para toda la aplicación ha sido el formal, refiriéndonos a los usuarios de usted con el objetivo de no mostrar prejuicios en cuanto a la edad y mostrar respeto por la persona que esta utilizando el servicio.

Tipografía

La tipografía utilizada en la aplicación fue *Roboto*, en la Figura 3.6 se puede observar un ejemplo con los diferentes tamaños que se utilizaron, a diferencia de una web convencional, en Android la medida para textos son los SP (Scalable pixels), a grandes rasgos funcionan igual que poner la medida en píxeles, pero la gran diferencia es que el tamaño de la letra se escala al ajuste del usuario, es decir, si un usuario tiene puesta la letra del sistema más grande, el texto marcado con SP se ajustará teniendo en cuenta esa medida [3].

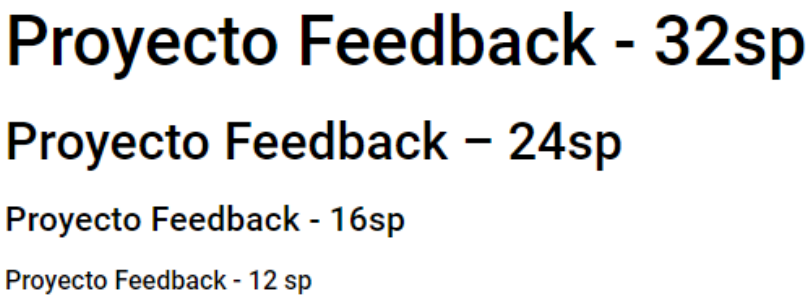


Figura 3.6: Tipografía utilizada

Iconografía

Para la iconografía dado que es una aplicación Android, *Google* tiene sus propios iconos. Es por ello que se han utilizado los *Google Fonts Icons* [7], ya que es una biblioteca muy extensa de iconos y, su estilo y versatilidad es acorde a las aplicaciones móvil. En la Figura 3.7 se pueden observar algunos ejemplos de iconos.

Figura 3.7: Ejemplo de algunos iconos de la biblioteca.

Componentes Layout

En cuanto al estilo elegido para los diferentes componentes de los *layout*, en la Figura 3.8 se puede observar el elegido para algunos. Se ha diseñado un estilo primario y uno secundario para cada uno, los cuales se han utilizado en la aplicación dependiendo de la situación y pantalla donde se encuentren.

Centrándonos en los botones, los circulares corresponden a *Fab Buttons*, normalmente situados en la parte inferior derecha de la pantalla y centrados en realizar una tarea primordial. En cuanto a al resto, los que son solo texto han sido utilizados en los *Dialogs* (pantallas emergentes) y en la *Cards* con tal de poder aprovechar el espacio al máximo en estos componentes tan pequeños.

Los *Snackbars* se han diseñado de forma que se pueda ver con claridad el mensaje en su interior debido a que normalmente mostraran mensajes importantes sobre algún aspecto de la aplicación y es importante que el usuario se entere de lo sucedido. Del mismo modo, en los *Text Fields* también se ha seguido la misma premisa en cuanto a contraste con el fondo.

Finalmente la *Tool Bar*, ha seguido el mismo estilo durante toda la aplicación y se ha fijado

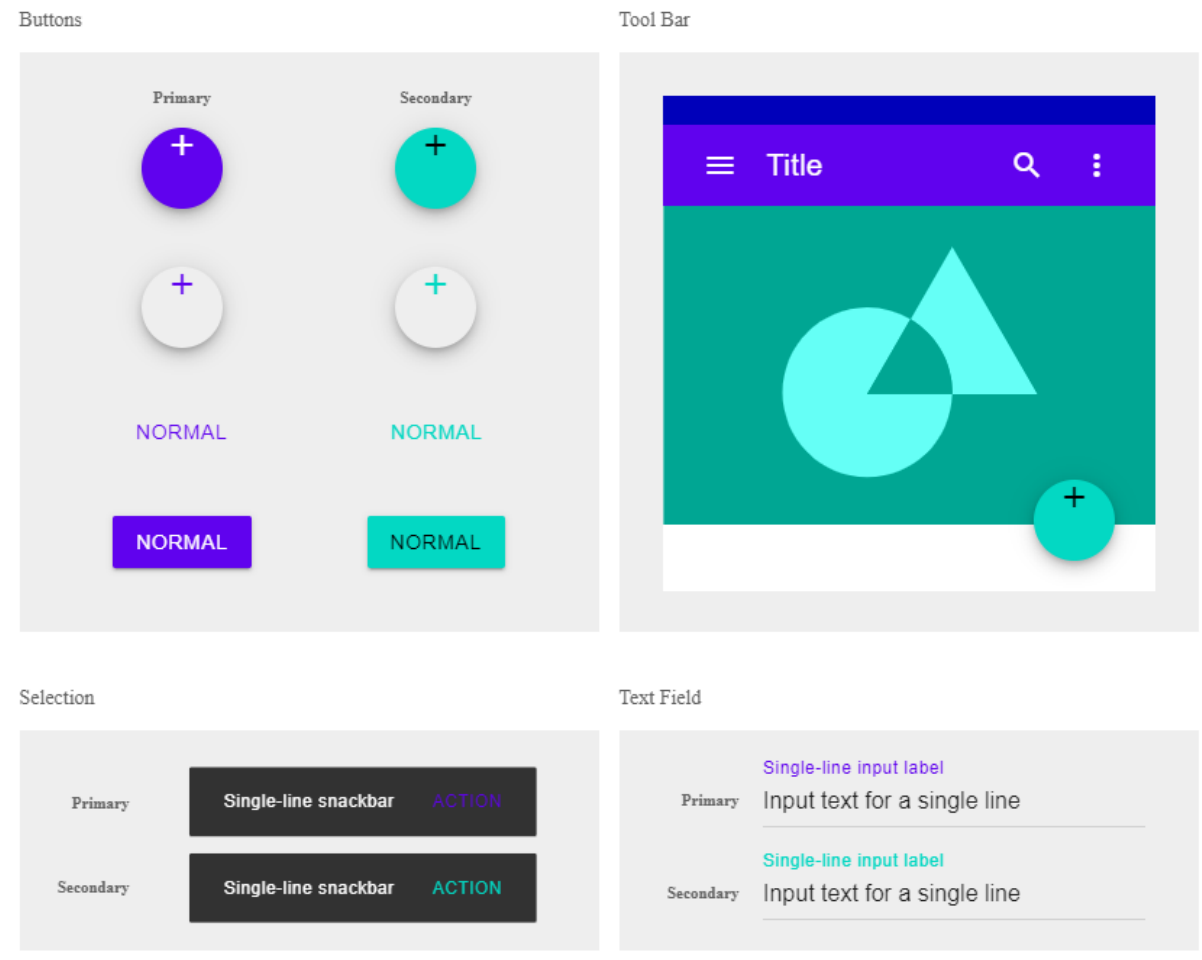


Figura 3.8: Estilo de algunos componentes del sistema.

un criterio por el cual, de izquierda a derecha, lo primero que se debería mostrar sería el botón de un menú, dado que debería aparecer también por ese lado al pulsarse. Tras ello iría el título de la página en la que nos encontramos. Después algunos iconos correspondientes a opciones importantes de la página actual y finalmente, un icono que al pulsarse se despliega un menú de opciones, para mostrar opciones secundarias de la página.

Sin embargo, ha habido casos en los que utilizar este diseño establecido no era la mejor opción, así que para casos específicos de la aplicación se ha utilizado un estilo más particular. En la Figura 3.9 se pueden observar dos ejemplos en los cuales el estilo no ha seguido con los estándares. En la Figura 3.9a se muestra el botón de cerrar sesión, este botón tiene un color distinto dado que se le quería dar cierta importancia y diferenciarlo del resto. El propósito de hacer esto no era otro que evitar que el usuario lo pudiera pulsar por error. La Figura 3.9b muestra botones de filtrado de publicaciones y se quería diferenciar del resto dado que la acción que realizaba no era la de un simple botón, sino que modificaba el contenido de la página completamente, además de que funcionan como *Radio Buttons*, es decir, solo puede haber uno pulsado en un momento determinado.

(a) Botón de cerrar sesión

(b) Botones de filtrado de publicaciones

Figura 3.9: Excepciones al estilo establecido

3.3.2. Sitemap

Con tal de diseñar la navegación a lo largo de la aplicación, se decidió realizar un *Sitemap* que lo representara. En la Figura 3.10 se puede observar el diagrama creado. Se decidió que se mostrara todas las funcionalidades sin diferenciar entre usuarios debido a que el diagrama hubiera sido muy repetitivo debido a que todos los usuarios repiten muchas funcionalidades y solo tienen unas pocas particulares. Por ejemplo, no hay ningún usuario que pueda realizar *feedback* y a la vez realizar una publicación.

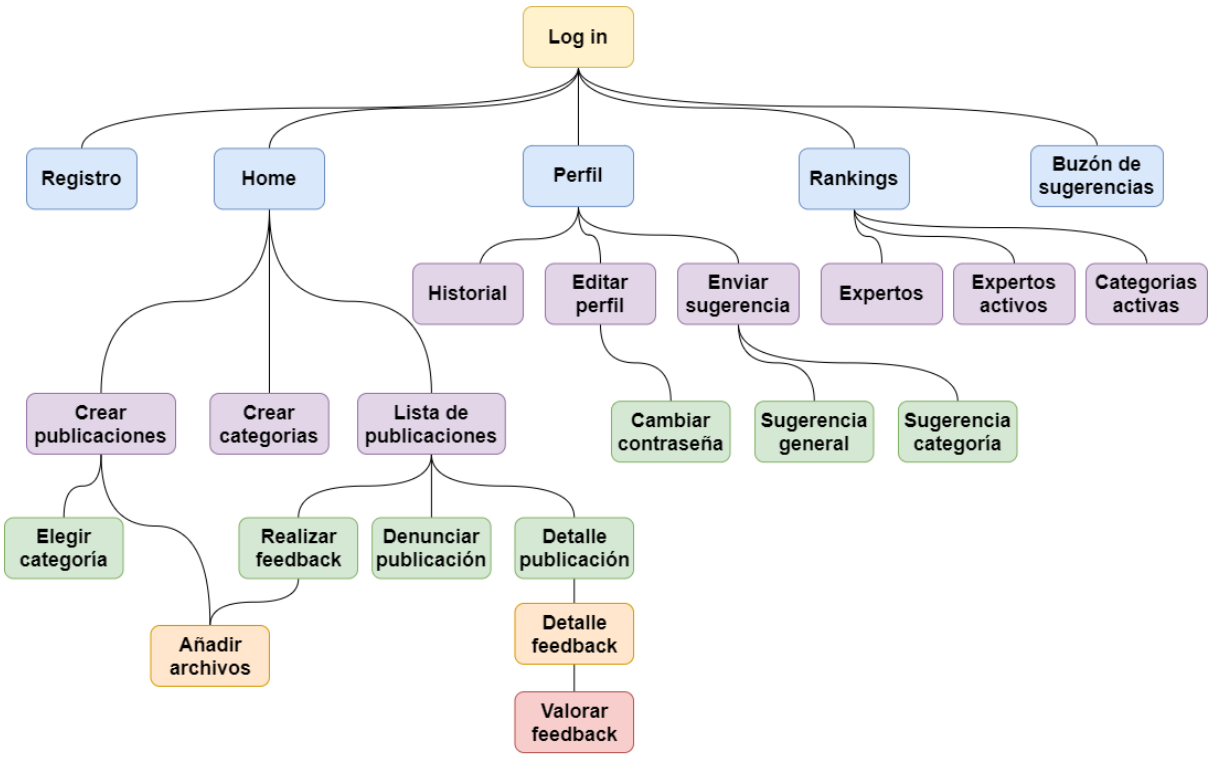
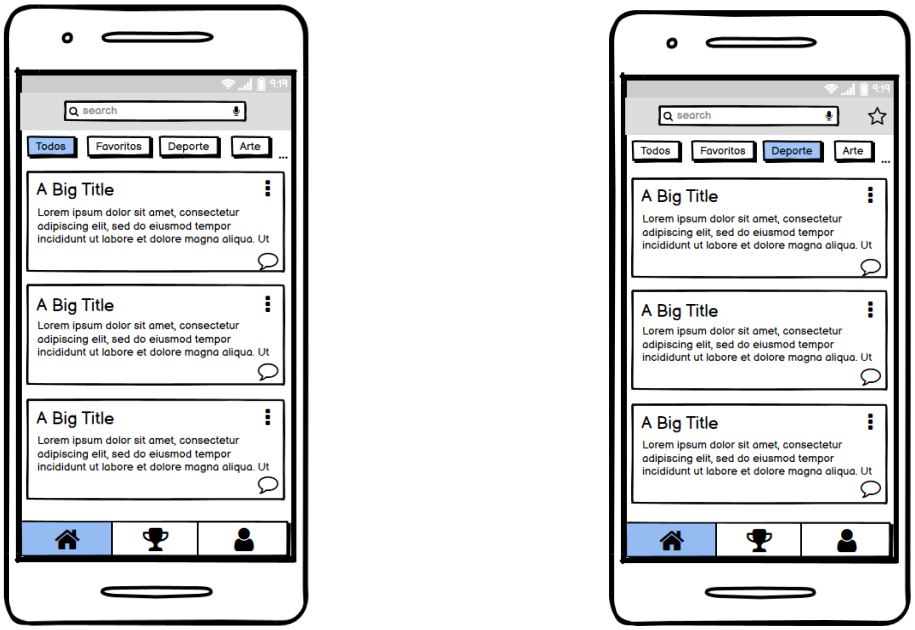


Figura 3.10: Sitemap del proyecto

3.3.3. Diseño de prototipos

Una vez realizado todo lo anterior, se decidieron crear prototipos en aras de tener una idea clara de donde debería ir cada cosa en el implementación real de la interfaz. Para ello se utilizó *Balsamiq frameworks* ya que no requiere de una curva de aprendizaje muy alta y además, es más cómodo de organizar y visualizar los prototipos que hacerlos a papel. Cabe recalcar que estos prototipos fueron diseñados antes de la implementación final y que por lo tanto, el diseño final de algunos cambió para adecuarse a las necesidades encontradas.

En la Figura 3.11 se pueden observar dos prototipos de la pantalla de listado de publicaciones. En ambas figuras se puede observar un menú inferior que se pensó con el propósito de moverse rápido a las pantallas más importantes del sistema. Encima de este menú, se puede encontrar una lista de las publicaciones que se han realizado en la *app* y tras ello una serie de botones. Estos botones muestran las diferentes categorías y al pulsar en ellos filtran las publicaciones que hay debajo, esta fue la forma más simple y más efectiva de realizar el filtrado, reduciendo considerablemente el número de pasos para hacerlo. Además, el menú superior de la Figura 3.11b es distinto con respecto a la Figura 3.11a, ya que si se encuentra en una categoría concreta el usuario tiene la opción de marcarla como favorita.



(a) Opción *Todos* seleccionada

(b) Opción categoría concreta

Figura 3.11: Pantalla de lista de publicaciones de la aplicación

Continuando, en la figura 3.12 se pueden ver las pantallas correspondientes a la creación de publicaciones y a su vista en detalle. Como datos de interés, en la primera de ellas se pensó en situar los botones importantes en la parte superior para que cuando el teclado estuviera desplegado, el usuario pudiera pulsarlos sin tener que salir atrás, situándolos abajo, al desplegarse el teclado estos quedarían solapados impidiendo que el usuario los pudiera pulsar. En la segunda figura, había mucha información que mostrar por lo que se trató de dividir la vista en

dos partes, la parte superior dedicada a los detalles de la publicación y la parte inferior dedicada a las *feedbacks* realizados.

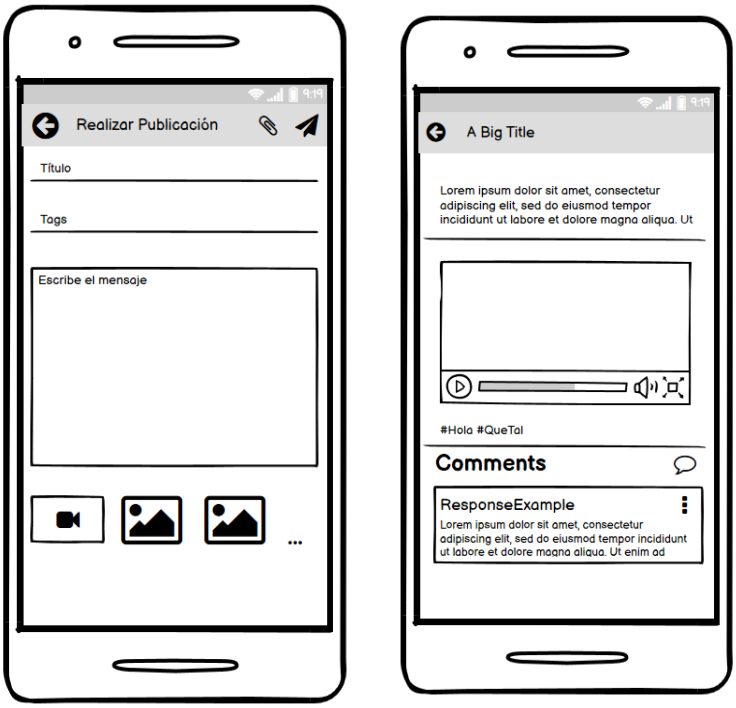


Figura 3.12: Pantallas de publicaciones.

Tras ello, en la Figura 3.13 se observan las pantallas de perfil, historial y buzón de sugerencias. Respecto a la pestaña que muestra los datos del perfil, en la parte superior derecha, existe un botón que al pulsarlo te lleva a la pestaña del historial, representada en la segunda de las tres pantallas que se muestran.

Finalmente en la Figura 3.14 se encuentran las pantallas de *rankings* y del buzón de sugerencias. Esta última, es exclusiva del administrador, apareciendo una cuarta opción en el menú inferior para que pueda acceder a ella. La pantalla de *rankings* es una simple tabla de clasificación con un menú superior que muestra el *ranking* correspondiente a la opción pulsada por el usuario.

Como se ha observado en estos prototipos, siempre se trató de diseñarlos de manera que la carga visual fuera la menor posible, para ello se utilizaron muchos iconos que sustituyeran a palabras. En pantallas pequeñas es bastante sencillo sobrecargar la vista, es por ello que el uso de iconos, a parte de dar un estilo más moderno a la aplicación, también reducen considerablemente la cantidad de texto en pantalla. También se ha tratado de aprovechar la verticalidad de las pantallas utilizando listas cuyos elementos están debajo del otro y no al lado.

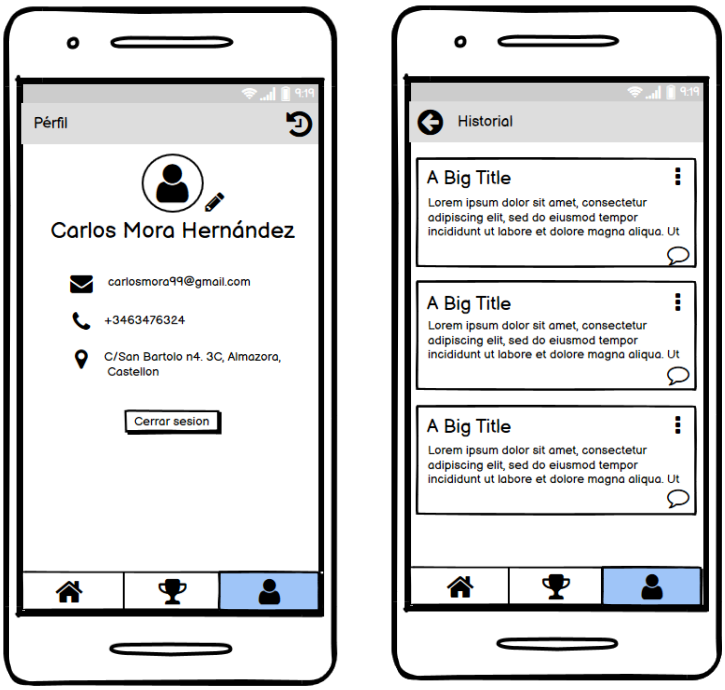


Figura 3.13: Pantallas del perfil, el historial.

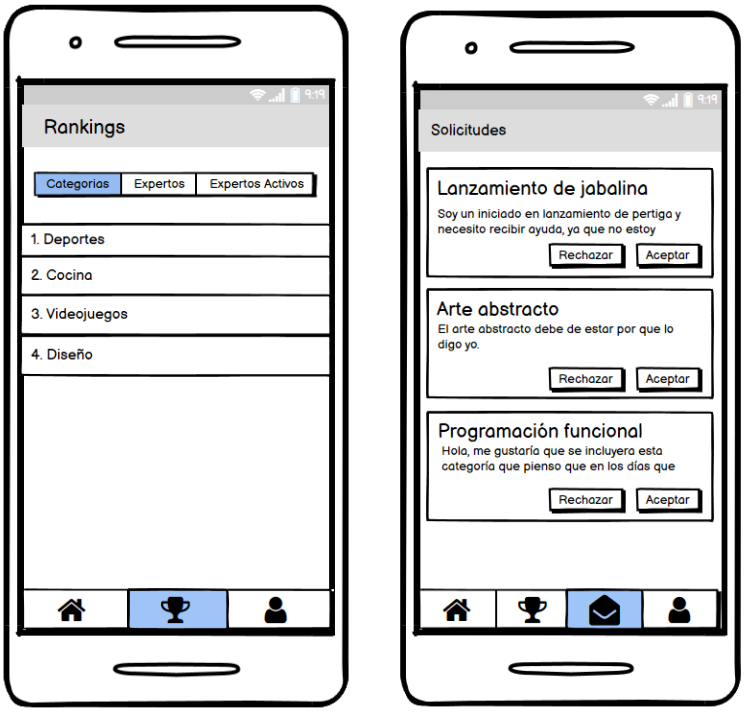


Figura 3.14: Pantallas de *rankings* y buzón de sugerencias.

Capítulo 4

Implementación y pruebas

4.1. Detalles de implementación

En esta sección se va a explicar ciertos detalles de la implementación, así como patrones de diseño, ciertas decisiones que se han tomado y el resultado final de la aplicación.

4.1.1. Organización y partes relevantes del sistema

Antes de hablar de como se han organizado las diferentes carpetas y clases del proyecto, habría que hablar de como es la estructura de un proyecto recién creado, es decir, como se organizan los distintos elementos en un proyecto Android. El proyecto se organiza en dos partes, la primera de ellas corresponde a todos los archivos escritos en Kotlin, mientras que la segunda corresponde a los recursos del sistema, como son los iconos, la letra, las vistas XML, etc.

Centrándonos ahora en la parte de Kotlin, en la Figura 4.1 se puede observar la organización que se ha seguido. Empezando por la carpeta *di*, esta contiene los contenedores que prestan la inyección de dependencias al resto de clases, su funcionamiento se explicará más adelante. Prosiguiendo con la carpeta *ui*, esta contiene el grueso del código, se ha organizado de forma que cada una de las carpetas que contiene representa a una pantalla de la aplicación. Un ejemplo del contenido de carpetas se puede ver en la Figura 4.2, corresponde a la pantalla *History* y en su interior se puede distinguir:

- Carpeta Interactor: Encargada de contener la parte del modelo que construye las peticiones de esa pantalla y la interfaz que utiliza el *ViewModel*.
- HistoryAdapter: Los adapter son los encargados de transcribir los *Data Sets* en los *RecyclerView*, si hay uno en esta carpeta significa que en la pantalla se mostrará una lista de elementos.
- HistoryFragment: Encargado de gestionar los diferentes eventos y elementos de la vista XML y tomar decisiones a partir de ellos.

- HistoryViewModel: Encargado de servir a la vista los elementos ya serializados que le llegan desde el servidor.
- OnClickHistory: Es una interfaz que implementa el *Fragment* y que tiene como principal propósito crear una capa de indirección entre él y el *Adapter* para gestionar los eventos en los diferentes ítems de la lista.

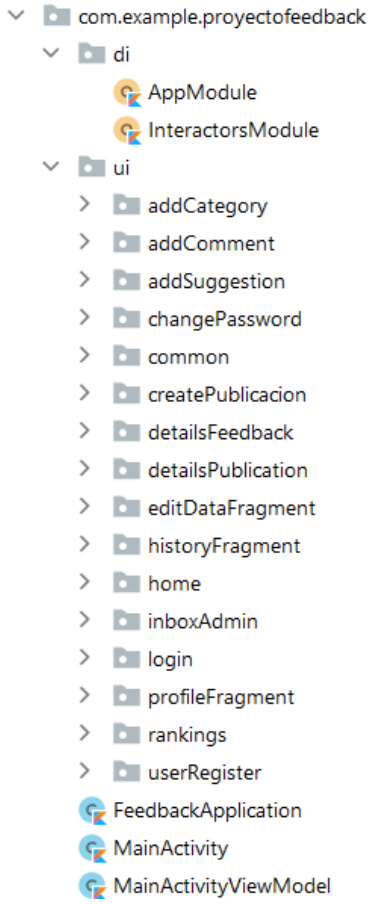


Figura 4.1: Organización del proyecto Kotlin

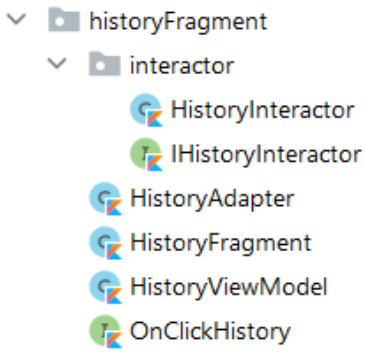


Figura 4.2: Organización del proyecto carpeta concreta Kotlin

Dentro del paquete *ui* también se encuentran tres clases más las cuales no pertenecen a ninguna pantalla. Estas son:

- FeedbackApplication: Es la clase principal de la aplicación. El estado de la misma se mantendrá durante todo el ciclo de vida de la aplicación, además es la primera clase que se ejecuta al iniciar la *app*. Se suele utilizar para guardar referencias a clases *Singleton* y es accesible desde cualquier clase o método que la necesite.
- MainActivity: Podríamos decir que para que un *Fragment* pueda ser creado, debe existir una clase *Activity* y eso es lo que hace esta clase. Su principal objetivo es ser el contenedor de los *Fragments* mostrando en cada momento el que corresponde. También tiene la capacidad de mostrar los elementos de layout comunes a todas los *Fragments* como podría ser una barra de menú.
- MainActivityViewModel: Su principal objetivo es transmitir información desde los *Fragments* hacia la clase *MainActivity*. Por ejemplo, supongamos que la clase *Activity* contiene una barra de menú la cual por defecto se muestra en todas las pantallas, sin embargo, en un *Fragment* concreto desearias que no se mostrase. Pues la manera correcta de ocultarla es a traves del *MainActivityViewModel* al cual estará suscrito la *MainActivity*.

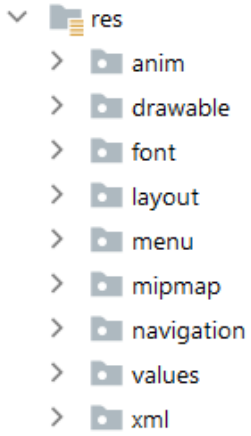


Figura 4.3: Organización de recursos

En cuanto a la parte de recursos, en la Figura 4.3 se puede observar como se divide en una serie de carpetas, cada una de ellas dedicada a un aspecto de la app. Todos los archivos contenidos en esta parte del sistema están escritos en XML. De estos paquetes, los más importantes son:

- Drawable: Contiene toda la iconografía que se utiliza en la *app* incluido el icono de la *app*, además de todos los estilos de layouts.
- Layout: Contiene todas las vistas XML del sistema.
- Navigation: Es donde, de una forma gráfica, se especifica el flujo de la aplicación, es decir, el transcurso entre pantallas que podría seguir el usuario. Esta manera de organizar el paso a través de las vistas se conoce como *Navigation Component*.

4.1.2. Patrones de diseño

A continuación se van a presentar los patrones de diseño que se han utilizado en el sistema. A este conjunto de patrones habría que sumar el utilizado para la arquitectura del sistema (Model-View-ViewModel), el cual ha sido descrito en la sección 3.2.

Singleton

Otro de los patrones utilizados en el sistema fue el patrón *Singleton*. Primeramente cabe recalcar que en el sistema existen varios modelos, cada uno de ellos para una pantalla y además, dentro de esos modelos se pueden realizar varias peticiones de forma asíncrona. Para comunicarse con la API Rest sería ineficiente tener multitud de instancias para cada uno de los modelos que se comunicaran con internet. Es por ello que la forma recomendada para realizar las peticiones al servidor es tener una sola instancia para toda la *app* la cual se encargue de hacerlo.

Para evitar que se pierdan peticiones o se desechen porque se esta realizando actualmente una, se implementó una cola de peticiones donde enviarlas de forma escalonada y que no se perdiera ninguna. En la Figura 4.4 se puede ver representada la estructura descrita, los diferentes *Interactors* son los encargados de construir la petición que se quiere realizar y la clase *FeedbackAPI* es la que solo tiene una instancia, se encarga de encolar las peticiones y enviarlas al servidor.

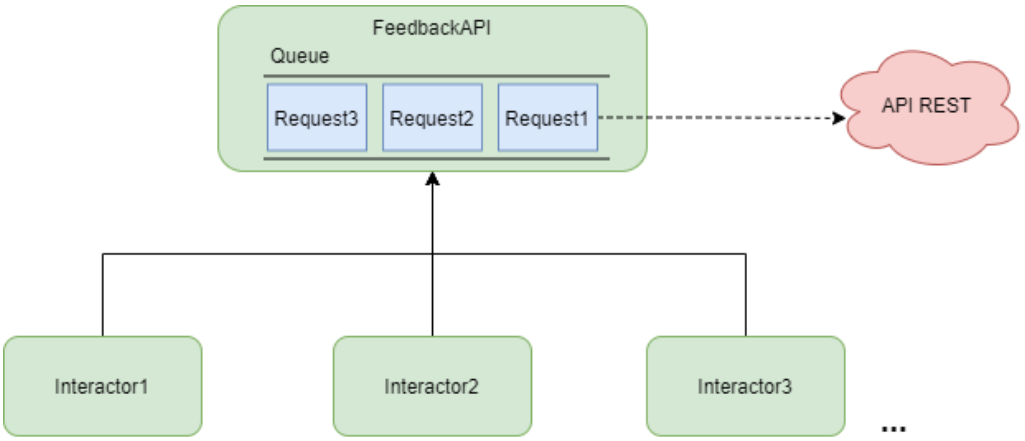


Figura 4.4: Diagrama Singleton

Adapter

En Android, normalmente el *Fragment* o la *Activity* se comunican directamente con la vista XML y son los encargados de inyectar los datos que llegan desde el servidor para que el usuario los vea. Sin embargo, hay un caso en el que esto no es posible, cuando se desea mostrar una lista de elementos se necesita adaptar los datos primero para poder introducirlos en el *RecyclerView*[1],

[1]Layout encargado de mostrar la lista de elementos al usuario.

para ello se utiliza la clase *Adapter*. Esta clase, recoge el *Data Set* que le llega desde el *Fragment* o *Activity* y se encarga adaptarlos para que el *RecyclerView* los entienda.

El funcionamiento es el siguiente: al recibir el *Data Set* desde el *Fragment*, el *Adapter* crea, para cada uno de los elementos un objeto de tipo *ViewHolder*, este elemento es el que el *RecyclerView* entiende y será el encargado no solo de mostrar los datos de cada elemento, sino de controlar los eventos de cada uno de ellos. En la Figura 4.5 se representa un diagrama mostrando el funcionamiento descrito.

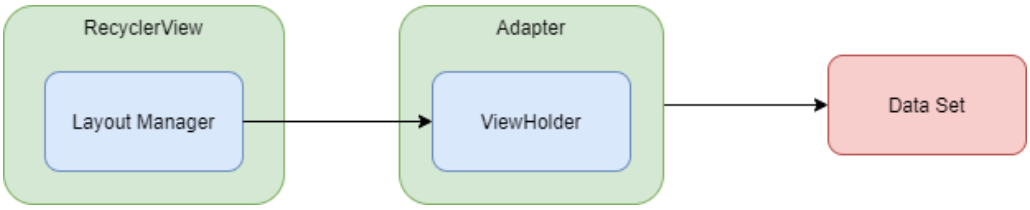


Figura 4.5: Diagrama Adapter

Observer

Muchas veces un objeto o clase quiere ser notificado cuando ciertos datos cambian. La manera más básica de realizar esto, es que el objeto que quiere recibir los datos pregunte si han cambiando. Sin embargo, esto es muy ineficiente, ya que se gastan recursos de una forma innecesaria. Es por ello, que normalmente para este tipo de casos se suele utilizar el patrón de diseño *Observer* que a diferencia de la forma explicada anteriormente, funciona de manera que el objeto que quiere recibir los datos se suscribe al objeto que los recibe y este, le notifica cuando los datos cambian haciendo la comunicación mucho más eficiente y evitando llamadas innecesarias.

En este proyecto, el patrón *Observer* se utilizo de manera más clara en la comunicación entre el *ViewModel* y el *Fragment*[2], para ello se utilizó la clase *LiveData* que funciona exactamente como un *Observer* convencional, pero que esta optimizada para el ciclo de vida de Android. El funcionamiento es el siguiente: Un objeto de esta clase es recogido por el *Fragment* y se suscribe a él. En el momento que se realiza una petición al servidor y los datos ya serializados llegan al *ViewModel*, este los introduce en el *LiveData* al cual esta suscrito el *Fragment*. Automáticamente se notifica al *Fragment* de los cambios y se realizan las acciones que se consideren oportunas. En la Figura 4.6 se visualiza un diagrama ejemplificando lo explicado.

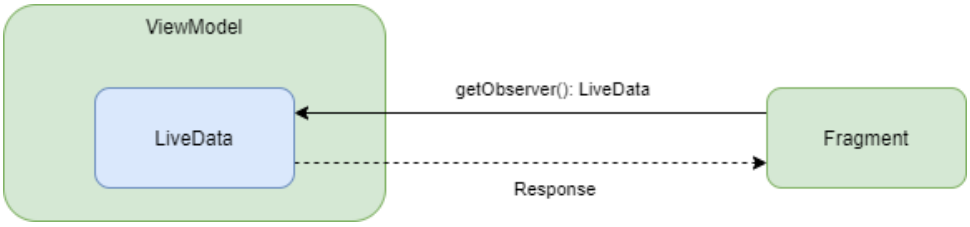


Figura 4.6: Diagrama Observer

[2]Clase encargada de controlar el comportamiento de la vista XML

Dependency Injection

Este patrón de diseño no estaba previsto introducirlo en un primer momento, pero como consecuencia del rápido progreso del proyecto y de los problemas que se tuvo al intentar realizar pruebas unitarias sobre los *ViewModels*, se decidió introducirlo para así facilitar la refactorización de código y que crear un *mock* o utilizar la clase en una prueba no fuera tan complicado. Para no tener que hacer una inyección de dependencias manual, se utilizó la biblioteca llamada *Hilt* que es la que recomienda Android para estos casos. El funcionamiento es bastante sencillo. Se crean ciertos objetos llamados contenedores que van a ser los encargados de prestar las dependencias cuando se requieran en alguna parte de la *app*. Como es habitual en Android las bibliotecas deben tener un ciclo de vida adaptado a sus necesidades de uso y así, no gastar más recursos de los que requiere, con *Hilt* [12] no iba a ser distinto. Los objetos contenedores se deben marcar como un tipo de componente dependiendo del componente que se le asigne a esa clase, su ciclo de vida variará, por ejemplo, marcar una clase contenedor con la etiqueta *SingletonComponent*, significa que va a estar viva durante todo el tiempo que la aplicación lo esté también, mientras que marcarla con una etiqueta de tipo *FragmentComponent*, significa que va a estar viva durante el tiempo en el que el Fragmento al cual está asociado también lo esté. En el código 4.1 se puede observar un ejemplo de como se marcan estos objetos, en este caso es un *SingletonComponent*.

```
@Module
@InstallIn(SingletonComponent::class)
object InteractorsModule {
    ...
}
```

Código 4.1: Ejemplo de contenedor en inyección de dependencias.

Una vez creados los objetos contenedor, estas se inyectan en las clases correspondientes, hay dos formas de hacerlo, a través de constructor o inyectándola como parámetro de la clase, se trató siempre de que se inyectara por constructor aunque no siempre fue posible. Al final se ha utilizado la inyección de dependencias para los *ViewModels*, los *Interactors* y algunas clases de utilidad. En la Figura 4.7 se puede ver un diagrama que muestra el funcionamiento de la inyección de dependencias.

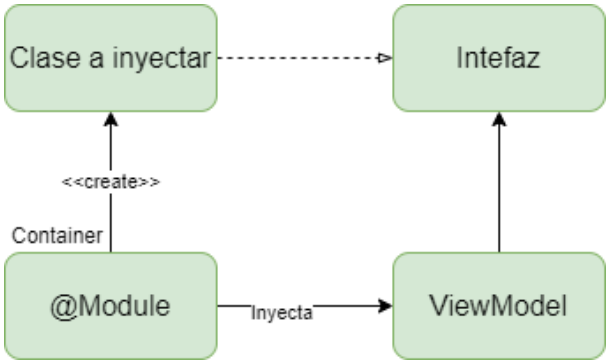


Figura 4.7: Diagrama de inyección de dependencias.

4.1.3. Decisiones y estrategias

Gestión de eventos en listas

Como se ha explicado en secciones anteriores, para mostrar una lista al usuario es necesario hacer uso de una clase *Adapter* que inyecte los datos en la vista y que si hace falta, los actualice. Sin embargo, no se ha hablado de la manera de gestionar un evento en un elemento de la lista.

Primeramente, hay que tener clara la estructura del sistema, y es que la clase *Fragment* se encarga de crear el *Adapter* e inyectarle el *Data Set* que necesita mostrar. En cuanto a responsabilidades de cada clase, la clase *Adapter* es la encargada de la gestión de datos en la lista, no de la gestión de eventos, entre otras cosas porqué es la clase *Fragment* la que normalmente se encarga de ello. Además, si fuera la clase *Adapter* quien gestionara los eventos, no se estaría respetando uno de los principios SOLID, *Single Responsability*. No obstante, los eventos de como tal llegan a la clase *Adapter*, es por ello que se ideo una estrategia para solucionar este problema y que los eventos se gestionasen en la clase *Fragment*.

Lo primero que se hizo fue crear una Interfaz que contuviera el método encargado de gestionar un evento (en este caso un click). Esa interfaz fue implementada en el *Fragment* y el se cambió el constructor del *Adapter* para que admitiera un Objeto del tipo de la interfaz creada. Por lo tanto, como el *Adapter* es creado por el *Fragment*, y el mismo es un objeto del tipo de la interfaz, al crearlo se introduce a si mismo como parámetro de constructor al *Adapter*. De esta forma, cuando salta algún evento en un elemento de la lista, se llama al *Fragment* para que lo gestione. En la Figura 4.8 se encuentra un diagrama que muestra las dependencias entre clases y lo explicado en esta sección.

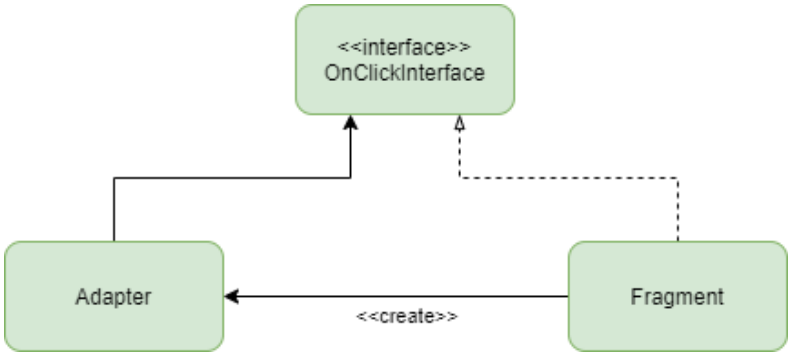


Figura 4.8: Diagrama de gestión de eventos en listas

Comunicación entre *ViewModels* e *Interactors*

Las clases *Interactors* son las encargadas de crear las peticiones, enviarlas al servidor y recibir la respuesta. Esta tarea no se puede realizar de forma síncrona dado que se bloquearía la aplicación constantemente y la experiencia de usuario sería nefasta. Además si se tuvieran que enviar multitud de peticiones en un momento concreto, se tendrían que crear una a una haciendo inútil a la clase *FeedbackAPI*, que como se explicó en la Sección 4.1.2, tiene una cola

encargada de gestionar el envío de peticiones al servidor.

Para que esto no ocurriera, se decidió utilizar los *callbacks* de *Kotlin*, que son unos métodos que se pasan como parámetros de otros y que se ejecutan de forma asíncrona. Todo eso sumado a que la biblioteca que se utiliza para las peticiones al servidor también las hace de forma asíncrona, provoca que el sistema no se quede congelado de cara al usuario.

Profundizando más en los *callbacks*, cuando se va a indicar al *Interactor* la petición que se quiere realizar, se componen introduciendo en ellos el código que se quiere ejecutar cuando llegue la respuesta desde el servidor. Normalmente en su interior se comprueba si la respuesta es un error y en caso de no serlo, se introduce en los *LiveData*. En el Código 4.2 se puede observar un ejemplo de uso de *callbacks*. En la función *createRequest* se ve como recibe un *callback* como parámetro y lo utiliza más adelante, mientras que en la función *method* se llama a *createRequest* y se define el código que se va a ejecutar en el *callback*.

```
fun createRequest(callback: (MutableList<T>) -> Unit) {
    ...
    callback(mutableListOf())
    ...
}

private fun method() {
    createRequest { list ->
        //Codigo callback
    }
}
```

Código 4.2: Ejemplo de declaración de callback.

Gestión de errores en respuestas

Cuando llega una respuesta, tanto si es lo que se había pedido o como si es un error, es importante notificárselo al usuario de alguna manera. Es por ello que para la gestión de errores se trato de idear alguna forma para que se mostraran de forma correcta.

Para ello se decidió crear una clase de utilidades que gestionara todas y cada una de las posibles respuestas de error que llegaran desde el servidor. Esta clase recoge la respuesta errónea que llega, la gestiona y devuelve un mensaje de error con información concreta de porque ha ocurrido y preparada para mostrar al usuario. Tras ello, hay un *LiveData* específico en cada *ViewModel* encargado de mostrar mensajes de error. Como este *LiveData* iba a ser exactamente igual en todos los *ViewModels*, se decidió crear una clase abstracta la cual sería extendida por cada uno de ellos y que contendría la inicialización del *LiveData* y el método que utilizaría el *Fragment* para suscribirse a él.

Paso de parámetros a través de Fragments

Para pasar parámetros de un *Fragment* a otro había dos opciones. La primera de ellas era hacer dicha comunicación a través de los *ViewModels*, que es la manera habitual de abordar este problema. Sin embargo, desde hace poco tiempo hay una manera distinta de hacer esto que es

utilizando *SafeArgs*. Para ello, el primer requisito es utilizar *Navigation Component*, elemento que se estaba utilizando en la *app*. A continuación habría que pensar que datos se quieren pasar, si las variables son de tipo primitivo simplemente hay que indicar el dato que se quiere pasar de un Fragmento a otro y cuando se quiera cambiar de pantalla, pasarle ese dato. En cambio, si el dato que se quiere pasar es una clase, hay que hacer un pequeño cambio en ella marcándola como @Parcelize e implementando la interfaz *Parcelable*, a partir de ahí se puede pasar como si fuera una clase primitiva. En el Código 4.3 se puede observar un ejemplo de como se marcó una clase para poderla pasar como parámetro a otro *Fragment*.

```
@Parcelize
data class Apprentice(var username: String): Parcelable {
    ...
}
```

Código 4.3: Ejemplo data class parcelable

Para la recogida de los argumentos en el destino tan solo hay que declarar una variable en el *Fragment* de tipo *navArgs* y automáticamente Android se encarga de rellenarla con los argumentos que le pasan.

Esto dio ciertos problemas en su implementación dado que al ser algo tan nuevo, es difícil encontrar una documentación fiable. Además, Android Studio aún no está del todo preparado para este tipo de argumentos y a veces no los detecta, mostrándote un error donde en realidad no lo hay.

4.1.4. Problemas y posibles mejoras futuras

A continuación se va a hablar de distintos problemas que se han encontrado durante la implementación del proyecto, su solución o en algunos casos su posible reemplazo en una futura revisión. Durante la sección 2.5 ya se trataron diferentes problemas que surgieron, sin embargo esta sección se va a centrar en los aspectos técnicos de los problemas que lo requieran.

Envió de peticiones a la API REST

La biblioteca que se utilizó para el envío de peticiones al servidor fue *Volley* [10], la cual es una de las recomendaciones de Android para este tipo de acciones y es relativamente fácil de usar. Sin embargo, una vez terminado el grueso del proyecto se quiso sustituir los *callbacks* que se hacían para comunicar los *Interactors* con los *ViewModels* por las corrutinas de *Kotlin*, dado que es la forma en la que se trabaja ahora para realizar partes de código de manera asíncrona. Tras estar probando varias formas de hacerlo, no se encontró la manera de que funcionase, ya que por algún motivo, *Volley* y las corrutinas, daban problemas de compatibilidad. Dado el tiempo que se tenía en ese momento se descartó el uso de corrutinas y se dejó la forma en la que estaba.

Para una futura implementación la idea sería sustituir la biblioteca *Volley* por *Retrofit*, otra de las bibliotecas que recomienda Android, la cual está actualizada para poder utilizar corrutinas e incluso es más simple de usar que *Volley*.

Pruebas unitarias sobre *ViewModels*

A la hora de diseñar estas pruebas se tenía que sustituir la dependencia del *Interactor* con un *mock*. Sin embargo, dado que la dependencia se estaba instanciando directamente en la clase, crear un *mock* era bastante complicado. Además, el uso de *callbacks* para actualizar las variables cuando llegara la respuesta lo complicaba aun más, dado que el *callback*, que era el grueso de la función se pasaba como parámetro a la llamada del *interactor*, haciendo imposible crear un *mock* y teniendo que crearlo expresamente para la prueba. Por este motivo, junto con el de la inyección de dependencias, las pruebas sobre *ViewModels* carecían de sentido y, aunque se decidió introducir la inyección de dependencias para eliminar esas dependencias duras, no se pudieron realizar.

Para solucionar este problema, habría que reestructurar las peticiones al servidor, ya que como se ha comentado antes, para sustituir los *callbacks* por las corrutinas habría también que cambiar la biblioteca de peticiones a una más actualizada.

Envió y recepción de archivos

Una de las mayores dificultades del proyecto fue el envío y recepción de archivos. Se tuvo que prescindir de utilizar *Volley* para recoger archivos ya que no soportaba nativamente el intercambio de *multipart/form-data* y las soluciones que se ofrecían estaban escritas en JAVA, que aunque se parece bastante a *Kotlin*, la traducción era bastante compleja. Al decidirse a utilizar Fuel, la falta de documentación propició que se tuviera que hacer un poco de investigación para saber que hacía cada método.

Otro problema relacionado fue que al seleccionar el archivo a enviar al servidor, la URL que se especificaba no era una dirección física y por lo tanto no se podía convertir a un archivo File que era el tipo que se requería para poder enviarse. Para solucionarlo, se tuvo que crear una clase que, dependiendo del tipo de archivo (.mp4, .jpeg, etc), convirtiera los archivos a tipo File.

Finalmente, la recepción de los archivos también fue compleja, porque aunque llegaban correctamente y se guardaban en cache, se tuvo que cambiar la configuración del *Manifest*, para que se pudieran obtener los archivos una vez guardados y así poderlos mostrar al usuario. En el Código 4.4 se pueden encontrar las líneas modificadas para que funcionase.

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths" />
</provider>
```

Código 4.4: Provider añadido al Manifest para obtención de archivos.

4.1.5. Solución final

En esta sección se va a mostrar los resultados obtenidos en el vista y se van a comentar las principales diferencias con respecto a los prototipos que se diseñaron, ya que durante el desarrollo de la app se vieron ciertas carencias o posibles mejoras sobre los diseños primigenios.

En la Figura 4.9 se puede observar el resultado de la pantalla de publicaciones del aprendiz. Como puntos cambiantes con respecto al diseño original, se pueden destacar los iconos debajo de cada publicación para indicar si existen documentos, vídeos o imágenes en esa publicación y así no tener que acceder a ella para comprobarlo. Por otro lado, al ser el aprendiz, en la parte inferior derecha se añadió un botón para crear una nueva publicación. Y por último, la imagen de la derecha refleja las pantallas de carga a lo largo de la aplicación, mostrando elementos vacíos (esqueletos) que van cargando la información progresivamente. Esto da una percepción no solo de velocidad sinó de mayor satisfacción al usuario [4].

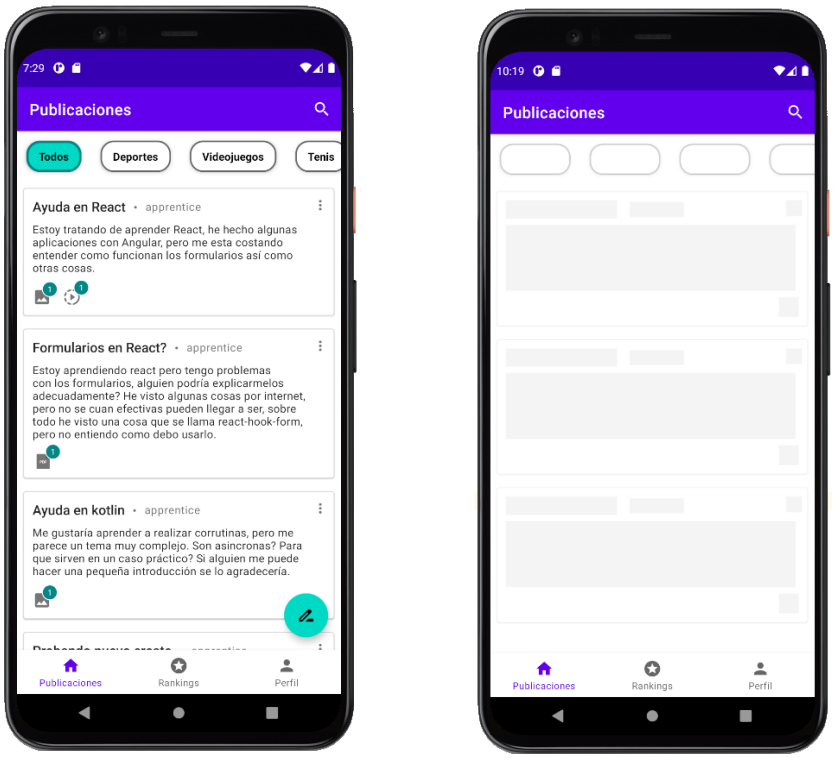


Figura 4.9: Pantallas de publicaciones aprendiz y carga

Por otra parte, en la figura 4.10 se pueden observar las pantallas de perfil y de *rankings*. En la segunda de ellas, el diseño ha variado ligeramente para dar un poco más de visibilidad a los tres primeros clasificados. Se han dispuesto en forma de podio y en una disposición diferente a las del resto de clasificados. El resto de elementos se han conservado muy parecidos a lo que se mostró en la sección 3.3.3.

En la Figura 4.11 se puede observar la pantalla de publicación con los campos requeridos completados. La estructura no ha variado en exceso con respecto a lo que se mostró en los prototipos, simplemente se ha añadido que el usuario pueda borrar una imagen ya seleccionada

y que observe el peso del archivo. En la imagen siguiente se observa una muestra de gestión de errores, en la cual se avisa al usuario que no pueden haber campos vacíos.

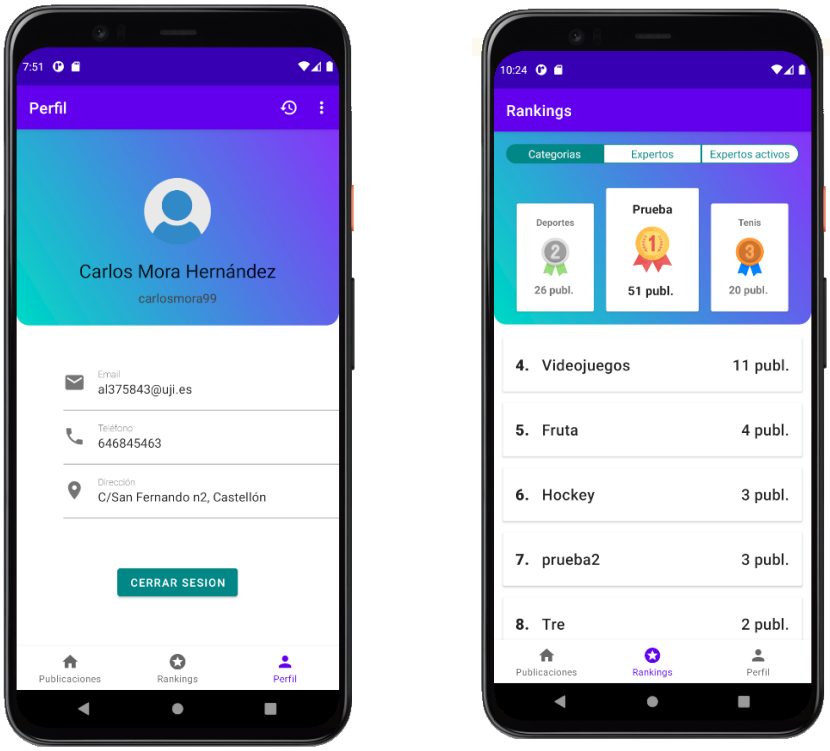


Figura 4.10: Pantallas de perfil y de *rankings*.

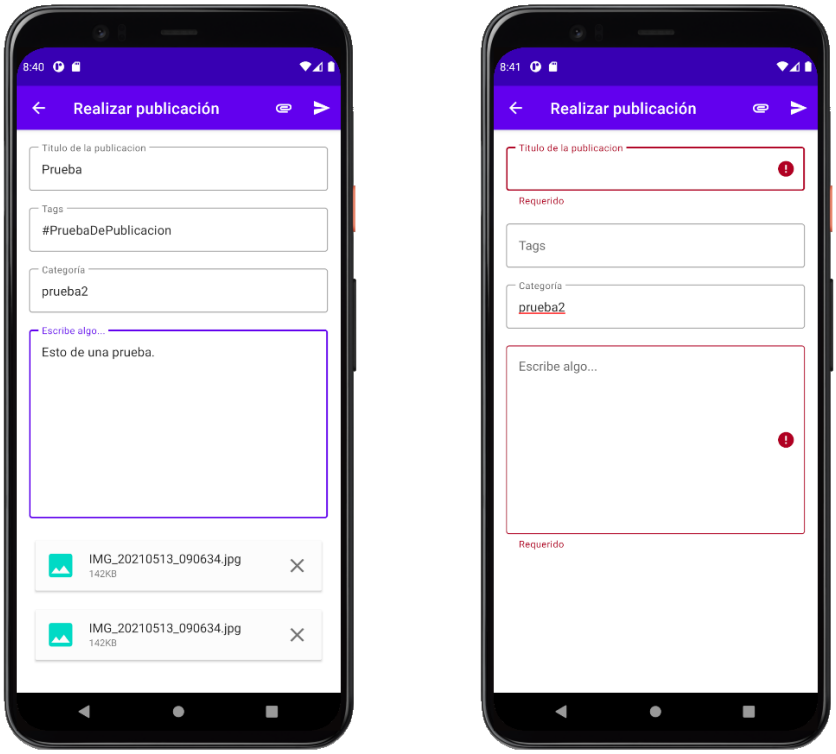


Figura 4.11: Pantalla de creación de una publicación y campos vacíos.

4.2. Verificación y validación

En este apartado se van a exponer los test que se realizaron para comprobar el correcto funcionamiento de la aplicación. Los test realizados fueron de dos tipos, test unitarios, sobre clases de utilidad y test de usuarios con tal de obtener una retroalimentación de algunos usuarios potenciales del sistema.

4.2.1. Test unitarios

Dada la complejidad que se observó al intentar realizar los test unitarios sobre *ViewModels*, finalmente se decidió realizar las pruebas sobre las clases de utilidad. Para sustituir las dependencias de las clases que se querían testear se utilizó la herramienta *Mockito*. Su funcionamiento es sencillo. Al crear un *mock* de una clase, se decide exactamente que va a devolver cuando se llame a uno de sus métodos. Posteriormente cuando la clase que realmente se quiere probar llama a ese método, devuelve algo fijo y evita que la clase mockeada influya en el resultado de la prueba.

Las pruebas unitarias se realizaron con el objetivo de comprobar que en diferentes casos se devolvía lo esperado y comprobar el correcto funcionamiento de estas clases relativamente complejas. En el Código 4.5 se puede observar un ejemplo de estas pruebas realizadas. En concreto la encargada de comprobar que los campos de texto eran válidos para el envío. En la primera parte de la prueba se configura el *mock* estableciendo que tiene que devolver un texto no vacío. Tras ello, se realiza la prueba ejecutando el método para validar los campos mockeados y finalmente se comprueba el resultado obtenido.

```
@Test
fun textOnTextField() {
    //Creacion del Mock
    val edit = MockEditable("Hola")
    val editText = mock<EditText> {
        on { this.text }.doReturn(edit)
    }
    val textInputLayout = mock<TextInputLayout> {
        on { getEditText() }.doReturn(editText)
    }

    //Ejecucion de la prueba
    val isValid = Validator.validateFields(context, textInputLayout)

    //Comprobacion de resultados
    assertEquals(true, isValid)
}
```

Código 4.5: Ejemplo de prueba unitaria

4.2.2. Test de usuario

Dadas las características del software que se estaba desarrollando, desde la empresa se recomendó que se realizaran test de usuario para validar el resultado con un grupo de usuarios

finales y potenciales. Este es el tipo de prueba más usado en los distintos proyectos de la empresa para validarlos, dado que ofrece una perspectiva de la calidad del software por parte de individuos ajenos al desarrollo y a la empresa.

Para planear estos test, lo primero que había que decidir era el objetivo de esta prueba, en este caso fue Çomprobar el correcto desempeño de las funcionalidades básicas del sistema". Tras ello, había que elegir el tipo de prueba que se iba a utilizar, se eligió QUIS [20] ya que era el tipo de test que más se adaptaba al sistema a probar, orientado a la interfaz de usuario y a la interacción del usuario con la *app*.

Tras ello se identificó a los perfiles de usuario necesarios para realizar esta prueba. Dadas las características del sistema como su semejanza con una red social, se pensó que el publico al que iría dirigido sería principalmente joven o de mediana edad. Es por ello que las personas elegidas para realizar el test fueron gente joven y que utilizaran las tecnologías de manera habitual.

Antes de comenzar a realizar las pruebas también se diseñaron las tareas o pasos que los usuarios deberían hacer al realizarla, estos fueron:

1. Crear una cuenta de aprendiz.
2. Autenticarse en la aplicación como aprendiz.
3. Listar publicaciones.
4. Realizar una publicación con archivos.
5. Buscar una publicación concreta.
6. Valorar una respuesta a una publicación.
7. Visualizar los diferentes *rankings* de la *app*.
8. Visualizar sus datos de usuario.
9. Modificar algún dato del perfil.
10. Realizar una sugerencia de categoría.
11. Darse de baja en la *app*.
12. Autenticarse en la aplicación como experto.
13. Responder a una publicación.
14. Cerrar sesión.

Una vez definidas las tareas a realizar, se realizaron las pruebas. Dadas las circunstancias actuales fueron desarrolladas de forma remota, se acordaba una hora con los participantes y se les enviaba todo el material necesario, una vez en la prueba se realizaba una explicación breve y al comenzar la prueba se hacía un seguimiento pasivo de la misma, ayudando a los usuarios solo si era totalmente necesario.

	ORS	Screen	TSI	Learning	SC	TOTAL
User 1	8,5	8,25	8,5	8,4	8,4	8,41
User 2	7,5	8,25	8,33	8,5	8,8	8,28
User 3	7	7,75	7,33	7,66	6	7,15
User 4	8,83	9	7,83	9	9	8,73
TOTAL	7,96	8,31	8,00	8,39	8,05	8,14

Cuadro 4.1: Resultados de los test de usuario.

En el Cuadro 4.1 se pueden observar los resultados de los test separados por apartados y por usuarios. Los resultados han sido muy satisfactorios obteniendo una media superior a 8 en casi todos los apartados y usuarios. El apartado con peor media es el ORS (Overall Reaction to the Software) esto se puede deber a algunos problemas que han ido surgiendo en la *app* durante el desarrollo de las pruebas y que los usuarios han experimentado. Mientras, el apartado con mejor media es el llamado Learning, que hace referencia a la facilidad de aprender como funciona la *app*, donde están situadas las cosas y mensajes que se muestran al usuario.

Durante el desarrollo de los test, los usuarios encontraron tanto puntos positivos como negativos en su experiencia, a continuación se listaran los más comunes:

Puntos negativos

- A algunos usuarios se les paraba la *app* cuando se adjuntaban archivos a las publicaciones. El problema era originado por la versión de Android de sus dispositivos, haciendo que si la versión era inferior a la 9.0, esta parte de la *app* no funcionaba correctamente.
- Al crear una categoría o enviar una sugerencia de categoría, el Spinner que da la opción a los usuarios de elegir una categoría padre creaba confusión dado que no estaba explicado en ningún punto su utilidad.

Puntos positivos

- La aplicación les dio una sensación de velocidad y las transiciones se realizaban de forma suave.
- Les resulto fácil navegar entre menús, ya que era muy intuitivo.
- El estilo utilizado era agradable a la vista.

Todos estos puntos se tendrán en cuenta para el desarrollo o mejora de futuras versiones. Esto ha sido un extracto con los resultados obtenidos tras realizar estos test, en el Anexo C se pueden observar los resultados de cada uno de los usuarios de manera más especifica.

Capítulo 5

Conclusiones

Conclusiones personales

Durante mi estancia en prácticas he aprendido a desarrollar una aplicación Android Nativa. En un primer momento elegí este proyecto porque había desarrollado pequeñas aplicaciones móvil en otros lenguajes. Pero no tenía claro si me iba a gustar, dado que no conocía la tecnología en concreto. Finalmente y tras todo el desarrollo de este proyecto puedo decir que me ha gustado mucho más de lo esperado, sobretodo en cuanto al lenguaje (Kotlin) y a las capacidades que ofrece para hacer cualquier cosa que se te ocurra. Esto también es un arma de doble filo, ya que conocer todo lo que se puede hacer, requiere muchísimo tiempo y en algunos casos puede llegar a ser frustrante.

En lo que respecta al ámbito profesional, mi experiencia en la empresa ha sido muy buena. Desde el primer momento me han ayudado en cualquier cosa que he necesitado o en cualquier problema que tuviese en el desarrollo del proyecto. La retroalimentación ha sido constante y útil, guiándome siempre hacia la mejor solución. Y mi experiencia personal también ha sido muy buena, por lo que he comentado en este párrafo y en el anterior. Todo eso ha propiciado que me sintiera cómodo haciendo este proyecto y que lo disfrutase.

Por otro lado, uno de los aspectos importantes del proyecto ha sido la coordinación que hemos tenido que tener mi compañero encargado del Backend y yo en todo momento, ya que había que seguir unos criterios fijos para desarrollar, el su parte y yo poder utilizarla en el proyecto. Además el trabajo de organización que hemos seguido ha ayudado a seguir una línea de trabajo conjunta y así poder ayudarnos el uno al otro realizando lo realmente necesario en cada momento.

Finalmente, como planes de futuro me gustaría seguir aprendiendo sobre aplicaciones móvil, no solo en Android sino en otras tecnologías como Swift o Flutter, ya que me parecen muy interesantes y dado lo que me ha gustado desarrollar en Android, seguramente estas tecnologías también me gusten y pueda adquirir conocimiento durante el proceso de aprendizaje.

Conclusiones técnicas

En cuanto a la realización del proyecto, se han completado todos los objetivos de manera satisfactoria. Además, la consecución de los mismos ha costado mucho menos tiempo de lo esperado, es por ello que se ampliaron estos objetivos y tareas con el propósito de continuar con el desarrollo del proyecto hasta alcanzar el tiempo establecido. Estos objetivos extra también fueron completados.

Las tecnologías utilizadas en el proyecto han resultado ser apropiadas para la consecución de estos objetivos y han facilitado en la mayoría de los casos su consecución. Tras la experiencia obtenida utilizándolas, hay ciertas partes del proyecto donde se podría haber mejorado su utilización (Por ejemplo a la hora de gestionar las subidas y bajadas de imágenes desde el servidor), pero el resultado no ha sido incorrecto en ningún caso.

Aunque se ha finalizado el proyecto totalmente, se pensaron algunas mejoras que podrían aportar cierto valor a la aplicación y que se podrían implementar en futuras versiones:

- Mapa de posicionamiento: Un mapa que ubicara donde se han realizado los feedbacks o publicaciones para ver que zonas del mundo han participado más en la aplicación y que así, los usuarios pudieran explorar las diferentes publicaciones de manera geográfica.
- Fotos de perfil: Una de las funcionalidades más básicas que las aplicaciones ofrecen a sus usuarios es la posibilidad de establecer una foto de perfil que los identifique, sin embargo en esta aplicación se descarto debido a la dificultad que se tuvo a la hora de subir imágenes al servidor, pero de cara a lanzar al mercado la *app*, sería una funcionalidad imprescindible.
- Busqueda avanzada: Actualmente en la *app*, aunque tiene una barra de búsqueda, buscar publicaciones por fecha o por palabras que aparezcan en la descripción no esta implementado. Es por ello que en un futuro sería interesante mejorar el sistema de búsqueda existente introduciendo estas mejoras.

Bibliografía

[1] Encarna Abellán. Metodología scrum. https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html, 2021. Consulta: 5 de marzo de 2020.

[2] Atlassian. Jira documentation. https://confluence.atlassian.com/jira, 2021. Consulta: 4 de mayo de 2021.

[3] Miguel Campos. Diferencias entre px, dp y sp en android. https://openwebinars.net/blog/diferencias-entre-px-dp-y-sp-en-android/. Consulta: 12 de marzo de 2021.

[4] Bill Chung. Everything you need to know about skeleton screens. https://uxdesign.cc/what-you-should-know-about-skeleton-screens-a820c45a571a, 2018. Consulta: 19 de octubre de 2018.

[5] Cuatroochenta. Web de cuatroochenta. https://cuatroochenta.com/.

[6] LLC Google. Android documentation. https://developer.android.com/?hl=es.

[7] LLC Google. Google fonts icons. https://fonts.google.com/icons. Consulta: 27 de abril de 2021.

[8] LLC Google. Material design guidelines. https://material.io/design/guidelines-overview. Consulta: 09 de mayo de 2021.

[9] LLC Google. Navigation component. https://developer.android.com/guide/navigation, 2019. Consulta: 27 de diciembre de 2019.

[10] LLC Google. Descripción general de volley. https://developer.android.com/training/volley?hl=es, 2020. Consulta: 03 de junio de 2020.

[11] LLC Google. Intents y filtros de intents. https://developer.android.com/guide/components/intents-filters?hl=es-419, 2020. Consulta: 03 de noviembre de 2020.

[12] LLC Google. Inyección de dependencias con hilt. https://developer.android.com/training/dependency-injection/hilt-android?hl=es-419, 2020. Consulta: 26 de junio de 2020.

[13] LLC Google. Descripción general del manifiesto de una app. https://developer.android.com/guide/topics/manifest/manifest-intro?gclid=CjwKCAjw1uiEBhBzEiwAO9B_HdMge_-bd53GjFGpstYYO3bPkP4bNP2Mj_Hnf52XtYP1QdcMH6fdwhoCi10QAvD_BwE&gclsrc=aw.ds, 2021. Consulta: 04 de mayo de 2021.

[14] LLC Google. Guía de arquitectura de apps. https://developer.android.com/jetpack/guide?hl=es-419#common-principles, 2021. Consulta: 30 de abril de 2021.

[15] LLC Google. Layouts android developers. https://developer.android.com/guide/topics/ui/declaring-layout, 2021. Consulta: 3 de mayo de 2021.

[16] OneSignal. Onesignal documentation. https://documentation.onesignal.com/docs. Consulta: 9 de mayo de 2021.

[17] Salesforce.com. Heroku documentation. https://devcenter.heroku.com/categories/reference, 2021. Consulta: 07 de mayo de 2021.

[18] Angel Luis Lozano Sanchez. Requisitos vs casos de uso vs historias de usuario. http://www.angellozano.com/requisitos-del-sistema-vs-casos-uso-vs-historias-usuario/, 2016. Consulta: 28 de febrero de 2016.

[19] JetBrains s.r.o. Kotlin documentation. https://kotlinlang.org/docs/home.html, 2021. Consulta: 5 de mayo de 2021.

[20] Sesar Joint undertaking. Questionnaire for user interface satisfaction (quis). https://ext.eurocontrol.int/ehp/?q=node/1611, 2012. Consulta: 22 de octubre de 2012.

[21] Kittinun Vantasin. Fuel repository. https://github.com/kittinunf/fuel, 2020. Consulta: 14 de diciembre de 2020.

Anexo A

Tablas de la especificación de casos de uso completa

Especificación del caso de uso	
Identificador	CU03
Nombre	Crear categorías.
Versión	V01
Autor	Alex Martínez Martínez
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los administrados puedan crear categorías.
Alcance	El sistema ha de permitir que los administrados puedan crear categorías, tanto categorías base como subcategorías.
Nivel	Tarea principal
Actor principal	Administrador
Actores secundarios	N/A
Relaciones	CU04, CU08
Precondición	El administrador ha de estar registrado como administrador en el sistema.
Condición fin con éxito	La categoría será creada y añadida a la lista de categorías.
Condición fin con fracaso	La categoría no será creada y no será añadida a la lista de categorías.
Trigger	Se quiere crear una nueva categoría para que haya más opciones al crear publicaciones.
Secuencia normal	**Acción**
1	El administrador se autentifica en la aplicación.
2	El administrador accede a la vista de crear categorías.
3	El administrador rellena toda la información necesaria sobre la categoría.
4	El administrador crea la categoría.
5	El sistema almacena la categoría y notifica al administrador que esta ha sido creada satisfactoriamente.
Excepción en 5 - Categoría padre no encontrada	**Excepción**
1	El sistema no almacena la categoría y notifica el error.
2	El administrador rellena toda la información necesaria sobre la categoría.
3	El administrador crea la categoría.
4	El sistema almacena la categoría y notifica al administrador que esta ha sido creada satisfactoriamente.
Excepción en 5 - Usuario no válido	**Excepción**
1	El sistema no almacena la categoría y notifica que el usuario no es válido.
2	El administrador rellena toda la información necesaria sobre la categoría.
3	El administrador crea la categoría.
4	El sistema almacena la categoría y notifica al administrador que esta ha sido creada satisfactoriamente.
Excepción en 4 - Cancelar operación	**Excepción**
1	El administrador cancela la operación.
Frecuencia esperada	1 vez cada dos semanas
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.1: Especificación del caso de uso CU03.

Especificación del caso de uso	
Identificador	CU04
Nombre	Añadir o eliminar categoría favorita.
Versión	V01
Autor	Carlos Mora Hernández
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los expertos puedan marcar o desmarcar categorías como favoritas.
Alcance	El sistema ha de permitir que los expertos puedan marcar (en caso de que no esté marcada) o desmarcar (en caso de que este marcada) categorías como favoritas.
Nivel	Tarea principal
Actor principal	Experto
Actores secundarios	N/A
Relaciones	CU03, CU08
Precondición	El experto ha de estar registrado en el sistema.
Condición fin con éxito	La categoría se marca o desmarca como favorita.
Condición fin con fracaso	La categoría no se marca o no se desmarca como favorita.
Trigger	Se quiere añadir o eliminar una categoría como favorita para acceder o dejar de acceder a ella.
Secuencia normal	**Acción**
1	El experto se autentifica en la aplicación.
2	El experto accede a la vista de categorías favorita.
3.1	El experto marca la categoría como favorita.
3.2	El sistema almacena la categoría como favorita y notifica al experto que esta ha sido añadida.
4.1	El experto desmarca la categoría como favorita.
4.2	El sistema almacena la categoría como no favorita y notifica al experto que esta ha sido eliminada.
Excepción en 3.2 - Categoría no encontrada	**Excepción**
1	El sistema no almacena la categoría como favorita y notifica el error.
2.1	El experto marca la categoría como favorita.
2.2	El sistema almacena la categoría como favorita y notifica al experto que esta ha sido añadida.
Excepción en 3.2 - Categoría ya es favorita	**Excepción**
1	El sistema no almacena la categoría como favorita y notifica el error.
2.1	El experto marca la categoría como favorita.
2.2	El sistema almacena la categoría como favorita y notifica al experto que esta ha sido añadida.
Excepción en 4.2 - Categoría no encontrada	**Excepción**
1	El sistema no almacena la categoría como no favorita y notifica el error.
2.1	El experto desmarca la categoría como favorita.
2.2	El sistema almacena la categoría como no favorita y notifica al experto que esta ha sido añadida.
Excepción en 4.2 - Categoría no es favorita	**Excepción**
1	El sistema no almacena la categoría como favorita y notifica el error.
2.1	El experto marca la categoría como favorita.
2.2	El sistema almacena la categoría como favorita y notifica al experto que esta ha sido añadida.
Frecuencia esperada	50 veces al día
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.2: Especificación del caso de uso CU04.

Especificación del caso de uso	
Identificador	CU05
Nombre	Realizar un feedback.
Versión	V01
Autor	Alex Martínez Martínez
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los expertos puedan realizar feedbacks sobre publicaciones creadas previamente.
Alcance	El sistema ha de permitir que los expertos puedan realizar feedbacks sobre publicaciones creadas previamente aunque no sean expertos en la categoría de la publicación permitiéndole adjuntar archivos de imagen, vídeo y documentos.
Nivel	Tarea principal
Actor principal	Experto
Actores secundarios	N/A
Relaciones	CU09
Precondición	El experto ha de estar registrado en el sistema.
Condición fin con éxito	El feedback ha sido creado y añadido a la lista de feedbacks.
Condición fin con fracaso	El feedback no se crea y no se añade a la lista de feedbacks.
Trigger	Se quiere dar retroalimentación a la publicación de algún aprendiz.
Secuencia normal	**Acción**
1	El experto se autentifica en la aplicación.
2	El experto accede a la vista de aportar feedback.
3	El experto rellena toda la información relativa al feedback.
4	El experto adjunta archivos al feedback.
5	El experto crea el feedback.
6	El sistema almacena el feedback y notifica al experto que este ha sido creado satisfactoriamente.
Excepción en 5 - Cancelar operación	**Excepción**
1	El experto cancela la operación.
Excepción en 6 - Publicación no encontrada	**Excepción**
1	El sistema no almacena el feedback y notifica el error.
2	El experto rellena toda la información relativa al feedback.
3	El experto adjunta archivos al feedback.
4	El experto crea el feedback.
5	El sistema almacena el feedback y notifica al experto que este ha sido creado satisfactoriamente.
Frecuencia esperada	100 veces al día
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.3: Especificación del caso de uso CU05.

Especificación del caso de uso	
Identificador	CU06
Nombre	Consultar historial.
Versión	V01
Autor	Carlos Mora Hernández
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que tanto expertos como aprendices puedan consultar su historial.
Alcance	El sistema ha de permitir que tanto expertos como aprendices puedan consultar su historial mostrando publicaciones y feedbacks realizados, además de valoraciones y respuestas obtenidas
Nivel	Tarea principal.
Actor principal	Aprendiz, Experto
Actores secundarios	N/A
Relaciones	N/A
Precondición	El usuario ha de estar registrado en el sistema.
Condición fin con éxito	El historial se lista y se puede consultar.
Condición fin con fracaso	El historial no se lista y no se puede consultar.
Trigger	El usuario quiere ver todo lo que ha acontecido durante su estancia en la aplicación.
Secuencia normal	**Acción**
1	El usuario se autentifica en la aplicación.
2	El usuario accede a la vista del historial.
3	El sistema lista el historial del usuario.
4	El usuario consulta su historial.
5	El sistema devuelve la información sobre el historial.
Excepción en 5 - Error de servidor	**Excepción**
1	El sistema no lista el historial del usuario y muestra mensaje de lista vacía.
2	El usuario accede a la vista del historial.
3	El sistema lista el historial del usuario.
4	El usuario consulta su historial.
5	El sistema devuelve la información sobre el historial.
Frecuencia esperada	200 veces al día
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.4: Especificación del caso de uso CU06.

Especificación del caso de uso	
Identificador	CU07
Nombre	Consultar, modificar y eliminar datos del perfil.
Versión	V01
Autor	Alex Martínez Martínez
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los usuarios puedan consultar, modificar y eliminar datos de su perfil.
Alcance	El sistema ha de permitir que los usuarios puedan consultar, modificar y eliminar datos de su perfil.
Nivel	Tarea principal
Actor principal	Aprendiz, Experto
Actores secundarios	Adminsitrador
Relaciones	N/A
Precondición	El usuario ha de estar registrado en el sistema.
Condición fin con éxito	El sistema devuelve los datos del perfil, y estos pueden modificarse y/o eliminarse.
Condición fin con fracaso	El sistema no devuelve los datos del perfil, y estos pueden modificarse y/o eliminarse.
Trigger	El usuario quiere ver sus datos personales y editarlos o eliminarlos.
Secuencia normal	**Acción**
1	El usuario se autentifica en la aplicación.
2	El usuario accede a la vista del perfil.
3	El sistema devuelve los datos del perfil.
4.1	El usuario modifica los datos de su perfil.
4.2	El sistema guarda los nuevos datos del perfil.
5.1	El usuario elimina los datos de su perfil.
5.2	El sistema elimina los datos del perfil y el usuario deja de formar parte de la aplicación.
Excepción en 4.1 y 5.1 - Cancelar la operación	**Excepción**
1	El usuario cancela la operación.
Excepción en 4.2 - Usuario ya existe	**Excepción**
1	El sistema no guarda los nuevos datos del perfil y notifica el error.
2.1	El usuario modifica los datos de su perfil.
2.2	El sistema guarda los nuevos datos del perfil.
Excepción en 5.2 - Contraseña incorrecta	**Excepción**
1	El sistema no elimina los datos del perfil y notifica el error.
2.1	El usuario elimina los datos de su perfil.
2.2	El sistema elimina los datos del perfil y el usuario deja de formar parte de la aplicación.
Frecuencia esperada	50 veces al día
Importancia	Necesaria
Prioridad	Medio plazo
Comentarios	N/A

Cuadro A.5: Especificación del caso de uso CU07.

Especificación del caso de uso	
Identificador	CU08
Nombre	Listar categorías.
Versión	V01
Autor	Carlos Mora Hernández
Fuente	Sergio Aguado González (Supervisor)
Descripción	El sistema ha de permitir que los usuarios puedan listar las categorías.
Alcance	El sistema ha de permitir que los usuarios puedan listar las categorías, tantolas categorías base como las subcategorías.
Nivel	Tarea principal
Actor principal	Aprendiz, Experto
Actores secundarios	Administrador
Relaciones	CU03
Precondición	El usuario ha de estar registrado en el sistema.
Condición fin con éxito	El sistema devuelve el listado de las categorías, y estas pueden consultarse.
Condición fin con fracaso	El sistema no devuelve el listado de las categorías, y estas no pueden consultarse.
Trigger	El usuario quiere ver un listado con todas las categorías existentes.
Secuencia normal	**Acción**
1	El usuario se autentifica en la aplicación.
2	El usuario accede a la vista del listado de categorías.
3	El sistema lista las categorías.
4	El usuario consulta las categorías.
Excepción en 3 - Error de servidor	**Excepción**
1	El sistema no devuelve el listado de las categorías y muestra un mensaje de lista vacía.
2	El usuario accede a la vista del listado de categorías.
3	El sistema lista las categorías.
4	El usuario consulta las categorías.
Frecuencia esperada	50 veces al día
Importancia	Necesaria
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.6: Especificación del caso de uso CU08.

Especificación casos de uso	
Identificador	CU09
Nombre	Valorar feedback
Versión	1.0
Autores	Alex Martínez Martínez
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El aprendiz que a recibido un feedback debe poder valorarlo para poder retroalimentar al experto
Alcance	El sistema permite que se realicen valoraciones solo si son autores de la publicación asociada y la nota variará del 1 al 10.
Nivel	Subtarea
Actor principal	Aprendiz
Actores secundarios	Administrador
Relaciones	N/A
Precondición	El aprendiz debe estar autenticado en el sistema.
Condición fin con éxito	Se envía la valoración al experto.
Condición fin con fracaso	El experto no recibe una valoración
Trigger	El experto realiza un feedback sobre una publicación.
Secuencia normal	**Acción**
1	El aprendiz se autentica en el sistema.
2	El aprendiz entra a la pantalla de un feedback recibido sobre una publicación.
3	El aprendiz valora dicho feedback.
4	Se envía la valoración al servidor.
Excepción en 3 - Valoración fuera de rango	**Excepción**
1	El aprendiz valora dicho feedback con una nota fuera de rango.
2	El sistema notifica al aprendiz y no envía la valoración
Frecuencia esperada	10 al día
Importancia	Secundaria
Prioridad	Medio plazo
Comentarios	N/A

Cuadro A.7: Especificación del caso de uso CU09

Especificación casos de uso	
Identificador	CU10
Nombre	Crear sugerencias
Versión	1.0
Autores	Carlos Mora Hernández
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El experto o aprendiz deben poder enviar a los administradores sugerencias de mejora y nuevas categorias que se pueden añadir al sistema.
Alcance	El sistema permite sugerencias de mejora y de nuevas categorias, tras ello se envian a los administradores, los cuales las reciben en un buzón.
Nivel	Subtarea
Actor principal	Aprendiz y experto.
Actores secundarios	Administrador
Relaciones	N/A
Precondición	El experto o aprendiz debe estar autenticado en el sistema.
Condición fin con éxito	El administrador recibe dichas sugerencias.
Condición fin con fracaso	El administrador no recibe dichas sugerencias.
Trigger	El experto o aprendiz encuentran una posible mejora en el sistema o una categoría que no existe.
Secuencia normal	**Acción**
1	El experto o aprendiz se autentifica en el sistema.
2	El experto o aprendiz entran en la pantalla de sugerencias.
3	El experto o aprendiz especifican la mejora o la categoría nueva que se podría introducir.
4	Se envía la sugerencia al servidor.
5	El administrador recibe dicha sugerencia y toma acciones para aplicarla.
Excepción en 3 - Sugerencia vacía	**Excepción**
1	El experto o aprendiz envían una sugerencia vacía
2	El sistema notifica al experto o aprendiz indicándole que especifique un mensaje.
Frecuencia esperada	1 a la semana
Importancia	Secundaria
Prioridad	Medio plazo
Comentarios	N/A

Cuadro A.8: Especificación del caso de uso CU10

Especificación casos de uso	
Identificador	CU11
Nombre	Reportar publicación
Versión	1.0
Autores	Alex Martínez Martínez
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El usuario debe poder reportar alguna publicación que le parezca ofensiva o inadecuada.
Alcance	El sistema debe prestar la opción al usuario de reportar una publicación por Spam o por contenido inadecuado.
Nivel	Subtarea
Actor principal	Aprendiz y experto.
Actores secundarios	N/A
Relaciones	CU02
Precondición	El usuario debe estar autenticado en el sistema.
Condición fin con éxito	Llega al servidor el reporte de la publicación.
Condición fin con fracaso	Llega al servidor el reporte de la publicación.
Trigger	El usuario encuentra una publicación inadecuada.
Secuencia normal	**Acción**
1	El usuario se autentifica en el sistema.
2	El usuario encuentra una publicación inadecuada.
3	El usuario reporta la publicación encontrada.
4	Se envía el reporte al servidor.
5	El sistema notifica al usuario de que la publicación ha sido reportada.
Excepción en 5 - Fallo de comunicación	**Excepción**
1	La petición falla y se notifica al usuario del problema que ha surgido.
Frecuencia esperada	2 al dia
Importancia	Secundaria
Prioridad	Medio plazo
Comentarios	N/A

Cuadro A.9: Especificación del caso de uso CU11

Especificación casos de uso	
Identificador	CU12
Nombre	Buscar publicaciones
Versión	1.0
Autores	Carlos Mora Hernández
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El usuario debe poder buscar las publicaciones que le interesen mediante un buscador.
Alcance	El sistema, a partir de los datos del usuario, debe poder filtrar publicaciones a partir de su nombre, el autor o los tags que contiene.
Nivel	Subtarea
Actor principal	Aprendiz y experto.
Actores secundarios	N/A
Relaciones	CU02
Precondición	El usuario debe estar autenticado en el sistema.
Condición fin con éxito	El usuario encuentra la publicación que buscaba.
Condición fin con fracaso	El usuario no encuentra la publicación que buscaba.
Trigger	El usuario accede a la vista de filtraje del sistema.
Secuencia normal	**Acción**
1	El usuario se autentifica en el sistema.
2	El usuario accede a la vista de filtraje del sistema.
3	El usuario introduce el criterio de búsqueda de una publicación.
4	El sistema busca dicha publicación.
5	El sistema muestra la/s publicación/es que coincide con ese criterio.
Excepción en 5 - Sin resultados	**Excepción**
1	El sistema muestra un mensaje especificando que no se han encontrado publicaciones que cumplan con los criterios descritos.
Frecuencia esperada	100 al dia
Importancia	Secundaria
Prioridad	Medio plazo
Comentarios	N/A

Cuadro A.10: Especificación del caso de uso CU12

Especificación casos de uso	
Identificador	CU13
Nombre	Autenticarse en la aplicación
Versión	1.0
Autores	Alex Martínez Martínez
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El usuario debe poder autenticarse en la aplicación para poder acceder a todas sus funcionalidades.
Alcance	El sistema permitirá la autenticación de usuarios a partir de un nombre de usuario y una contraseña, ambos especificados durante el registro.
Nivel	Tarea principal
Actor principal	Aprendiz, experto y administrador
Actores secundarios	N/A
Relaciones	CU15
Precondición	El usuario debe estar registrado en el sistema.
Condición fin con éxito	El usuario se autentifica en el sistema y a todas sus funcionalidades.
Condición fin con fracaso	El usuario no se autentifica en el sistema.
Trigger	Un usuario entra en la aplicación.
Secuencia normal	**Acción**
1	El usuario entra en la aplicación
2	El usuario introduce el nombre de usuario y su contraseña.
3	El sistema valida que tanto el usuario como la contraseña son correctos.
4	El usuario accede al sistema y a sus funcionalidades.
Excepción en 3 - Contraseña incorrecta	**Excepción**
1	El sistema valida que el usuario y la contraseña son incorrectos.
2	El sistema notifica al usuario que alguno de los datos es incorrecto.
Frecuencia esperada	300 al dia
Importancia	Vital
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.11: Especificación del caso de uso CU13

Especificación casos de uso	
Identificador	CU14
Nombre	Listar rankings
Versión	1.0
Autores	Alex Martínez Martínez
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El usuario debe poder ver los diferentes rankings que existen en el sistema.
Alcance	El sistema debe prestar a los usuarios tres rankings, uno de categorías más activas, otro de expertos más activos y el último de expertos mejor valorados.
Nivel	Tarea principal
Actor principal	Aprendiz y experto
Actores secundarios	N/A
Relaciones	N/A
Precondición	El usuario debe estar autenticado en el sistema.
Condición fin con éxito	La lista de rankings es visualizada por el usuario
Condición fin con fracaso	El sistema no muestra la lista al usuario.
Trigger	Un usuario entra en la pantalla de rankings.
Secuencia normal	**Acción**
1	El usuario se autentifica en el sistema.
2	El entra en la pantalla de rankings.
3	El usuario visualiza uno de los rankings y tiene la opción de ver el resto.
Excepción en 3 - No se muestra la lista	**Excepción**
1	El usuario no visualiza los rankings debido a que está vacía, se le muestra un mensaje notificándoselo.
Frecuencia esperada	200 al dia
Importancia	Vital
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.12: Especificación del caso de uso CU14

Especificación casos de uso	
Identificador	CU15
Nombre	Registrarse en la aplicación
Versión	1.0
Autores	Carlos Mora Hernández
Fuentes	Sergio Aguado González (Supervisor)
Descripción	El usuario debe poder registrarse en la aplicación introduciendo sus datos personales y eligiendo el rol que quiere realizar.
Alcance	El sistema debe permitir elegir el rol al usuario, así como que se puedan introducir los datos personales, comprobar que la contraseña cumple con los mínimos y que el nombre de usuario es único.
Nivel	Tarea principal
Actor principal	Aprendiz y experto
Actores secundarios	N/A
Relaciones	N/A
Precondición	El usuario debe entrar en el sistema.
Condición fin con éxito	El usuario esta registrado en el sistema y puede acceder a él.
Condición fin con fracaso	El usuario no se ha registrado en el sistema.
Trigger	Un usuario entra en la pantalla de registro.
Secuencia normal	**Acción**
1	El usuario entra en la pantalla de registro.
2	El usuario introduce sus datos en el sistema.
3	El sistema valida esos datos.
4	El sistema notifica al usuario indicándole que esta registrado en el sistema.
Excepción en 2 - Nombre de usuario repetido	**Excepción**
1	El sistema valida los datos.
2	El sistema notifica al usuario indicando que el nombre de usuario introducido ya está en uso.
3.1	El usuario cambia el nombre de usuario por otro.
3.2	El sistema valida esos datos.
3.3	El sistema notifica al usuario indicándole que esta registrado en el sistema.
Excepción en 2 - Contraseñas no coinciden	**Excepción**
1	El sistema valida los datos.
2	El sistema notifica al usuario indicando que la contraseña del primer campo no coincide con la del segundo campo.
3.1	El usuario cambia dichos campos para que sean iguales sus datos.
3.2	El sistema valida esos datos.
3.3	El sistema notifica al usuario indicándole que esta registrado en el sistema.
Frecuencia esperada	10 al dia
Importancia	Vital
Prioridad	Corto plazo
Comentarios	N/A

Cuadro A.13: Especificación del caso de uso CU15

Especificación casos de uso	
Identificador	CU16
Nombre	Recibir notificaciones
Versión	1.0
Autores	Carlos Mora Hernández
Fuentes	Sergio Aguado González(Supervisor)
Descripción	El sistema ha enviar una notificación de tipo push cuando alguien reaccione a una publicación o a un feedback del usuario.
Alcance	El sistema debe detectar cuando alguien reacciona a alguna publicación o feedback del usuario y debe notificar mediante un texto breve al usuario implicado.
Nivel	Subtarea
Actor principal	Aprendiz y Experto
Actores secundarios	N/A
Relaciones	CU09, CU05
Precondición	El usuario debe estar autenticado en el sistema como experto o aprendiz
Condición fin con éxito	El autor de la publicación o feedback recibe una notificación.
Condición fin con fracaso	El sistema muestra un mensaje de error al usuario que ha valorado o realizado un feedback comunicando que no se ha podido enviar la notificación push
Trigger	Un usuario realiza un feedback o una valoración
Secuencia normal	**Acción**
1	El usuario se autentica en el sistema como aprendiz.
2	El usuario busca uno de los feedbacks recibidos en sus publicaciones.
3	El usuario valora dicho feedback con un 5.
4	El sistema procesa la información y envía la información al servidor.
5	El servidor envía la notificación al usuario.
6	El usuario autor de ese feedback recibe una notificación con un mensaje que indica la nota con la que le han valorado.
Excepción en 3 - Misma nota en la valoración	**Excepción**
1	El usuario valora dicho feedback con la misma nota que anteriormente.
2	El sistema procesa la información y no envía la información al servidor.
3	El usuario autor del feedback no recibe una notificación.
Frecuencia esperada	200 al dia
Importancia	No vital
Prioridad	Largo plazo
Comentarios	N/A

Cuadro A.14: Especificación del caso de uso CU16

Anexo B

Diagramas de actividades restantes

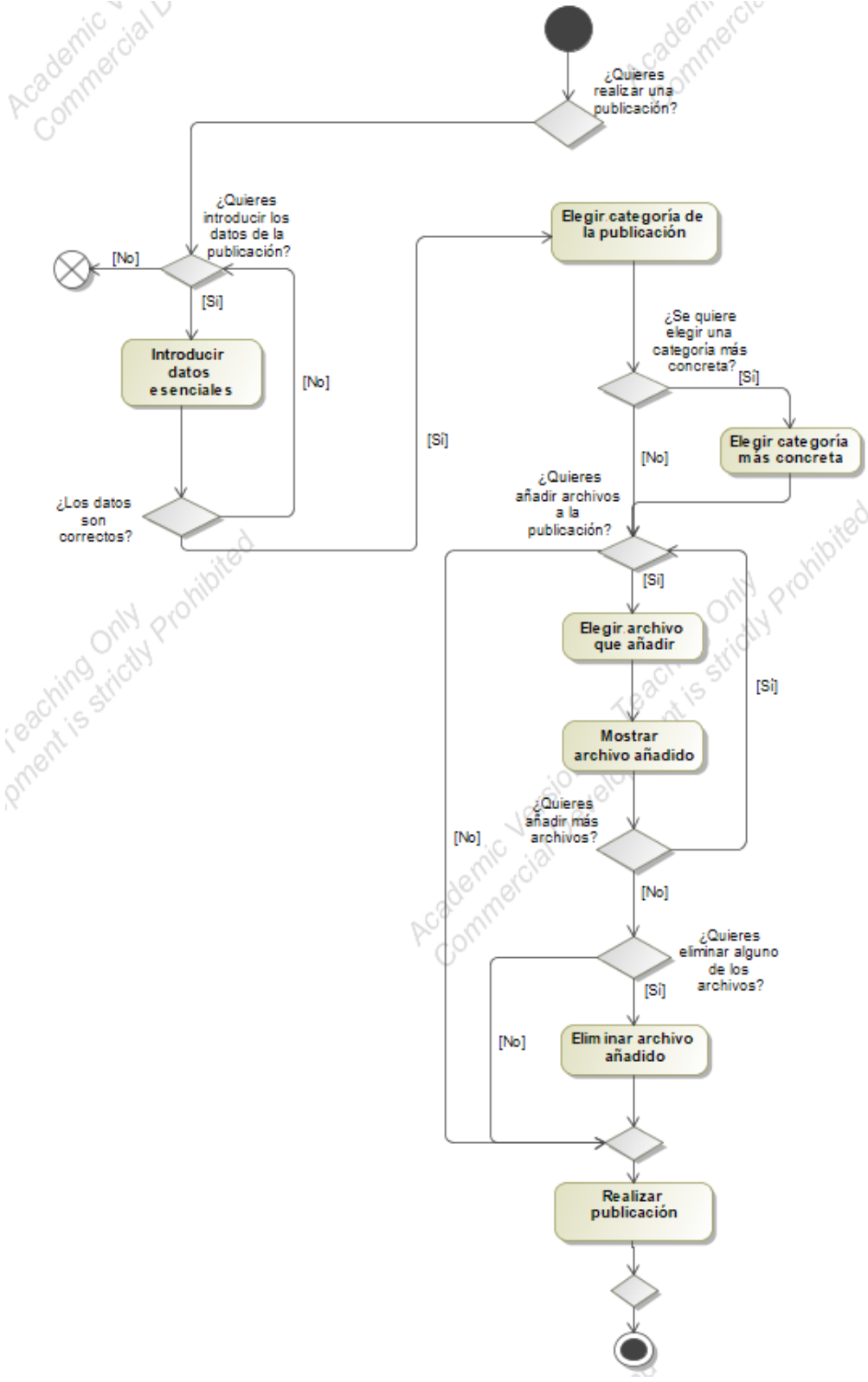


Figura B.1: Diagrama de actividades del caso CU01.

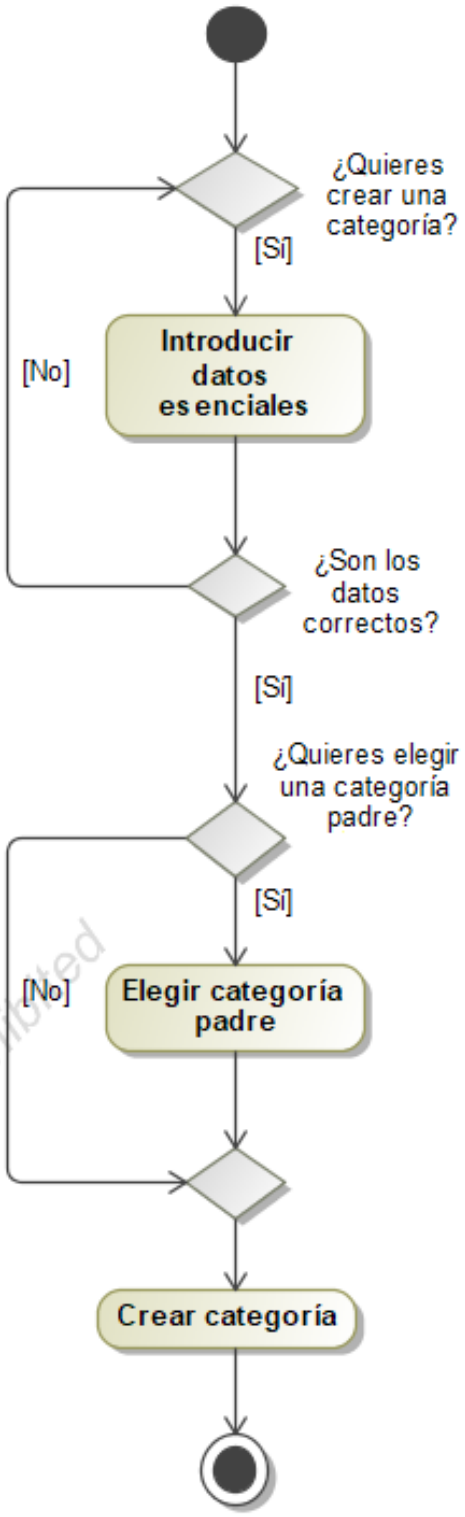


Figura B.2: Diagrama de actividades del caso CU03.

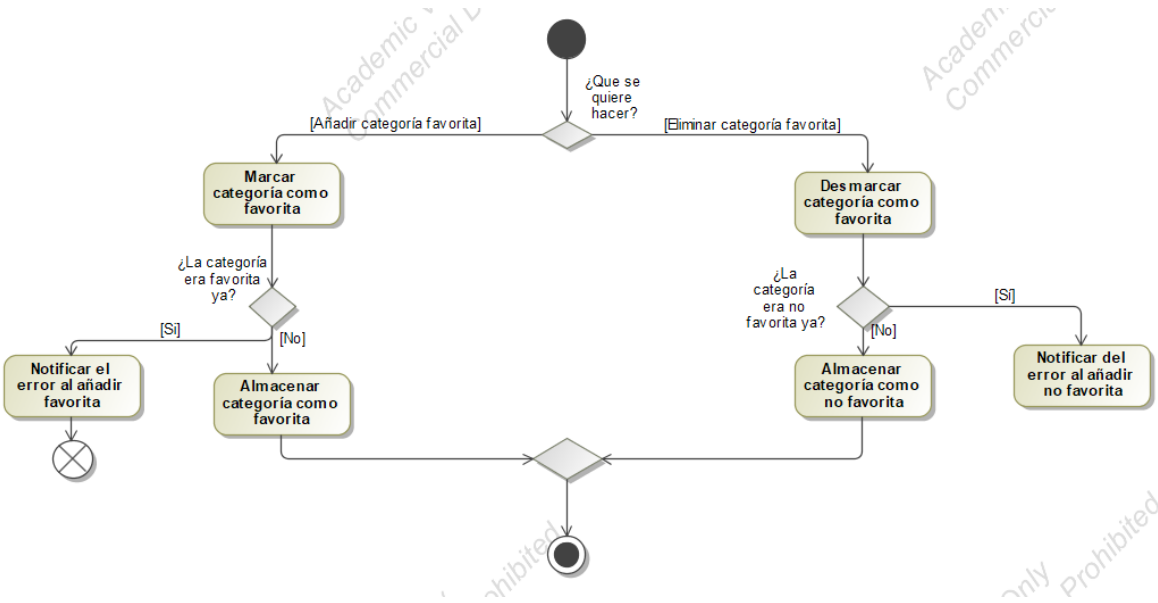


Figura B.3: Diagrama de actividades del caso CU04.

Anexo C

Test de usuario

Questionnaire for User Interface Satisfaction

OVERALL REACTIONS TO THE SOFTWARE			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
1		terrible									X		wonderful	
2		difficult									X		easy	
3		frustrating										X	satisfying	
4		inadequate										X	adequate	
5		dull										X	stimulating	
6		rigid									X		flexible	
SCREEN			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
7	Reading characters on the screen	hard to read										X	easy to read	
8	Highlighting simplifies task	not at all								X			very much	
9	Organization of information	confusing									X		very clear	
10	Sequence of screens	confusing										X	very clear	
TERMINOLOGY AND SYSTEM INFORMATION			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
11	Use of terms throughout system	inconsistent										X	consistent	
12	Rerminology related to task	never									X		always	
13	Position of messages on screen	inconsistent										X	consistent	
14	Prompts for input	confusing										X	clear	
15	Computer nforms about what its progress	never								X			always	
16	Error messages	unhelpful										X	helpful	
LEARNING			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
17	Learning to operate the system	difficult									X		easy	
18	Exploring new features by trial and error	difficult									X		easy	
19	Remembering names and use of commands	difficult										X	easy	
20	Performing task is straightforward	never										X	always	
21	Help messages on the screen	unhelpful									X		helpful	
22	Supplemental reference materials	confusing											clear	X
SYSTEM CAPABILITIES			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
23	System speed	too slow										X	fast enough	
24	System reliability	unreliable								X			reliable	
25	System tends to be	noisy										X	quiet	
26	Correcting your mistakes	difficult										X	easy	
27	Designed for all level of users	never									X		always	

	List the most **negative** aspect(s)		List the most **positive** aspect(s)
1	Sometimes the app closes for no reason.	1	Looks good overall
2	Some field are not self explanatory, e. g. "Selecciona una categoría" in "Nueva categoria"	2	Easy to navigate
3	It feels like I should be able to reply, but I'm not	3	Loads are fast

TEST PERSON INFORMATION

Age:	22
Gender:	Male
Level of education:	Graduated
Hours spent using a computer per week on average:	40
Do you blog?	No
Do you usually read a newspaper?	No
Do you know/use:	
Facebook:	Yes
Linkedin:	Yes
Github:	Yes

Questionnaire for User Interface Satisfaction

OVERALL REACTIONS TO THE SOFTWARE			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
1		terrible								X			wonderful	
2		difficult									X		easy	
3		frustrating										X	satisfying	
4		inadequate									X		adequate	
5		dull								X			stimulating	
6		rigid							X				flexible	
SCREEN			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
7	Reading characters on the screen	hard to read										X	easy to read	
8	Highlighting simplifies task	not at all									X		very much	
9	Organization of information	confusing									X		very clear	
10	Sequence of screens	confusing									X		very clear	
TERMINOLOGY AND SYSTEM INFORMATION			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
11	Use of terms throughout system	inconsistent								X			consistent	
12	Rerminology related to task	never									X		always	
13	Position of messages on screen	inconsistent									X		consistent	
14	Prompts for input	confusing										X	clear	
15	Computer nforms about what its progress	never										X	always	
16	Error messages	unhelpful										X	helpful	
LEARNING			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
17	Learning to operate the system	difficult										X	easy	
18	Exploring new features by trial and error	difficult										X	easy	
19	Remembering names and use of commands	difficult										X	easy	
20	Performing task is straightforward	never									X		always	
21	Help messages on the screen	unhelpful									X		helpful	
22	Supplemental reference materials	confusing									X		clear	
SYSTEM CAPABILITIES			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
23	System speed	too slow										X	fast enough	
24	System reliability	unreliable										X	reliable	
25	System tends to be	noisy									X		quiet	
26	Correcting your mistakes	difficult										X	easy	
27	Designed for all level of users	never										X	always	

	List the most **negative** aspect(s)		List the most **positive** aspect(s)
1	The suggestion section's name was a bit confusing	1	The animations feel smooth
2	Some menu titles were not consistant	2	The layout is very intuitive and easy to use
3	On the home menu I'd prefer some feedback on when was the post created and how many comments it has.	3	The overall theme of the application looks good.

TEST PERSON INFORMATION

Age:	22
Gender:	Male
Level of education:	University student
Hours spent using a computer per week on average:	70
Do you blog?	Yes
Do you usually read a newspaper?	No
Do you know/use:	
Facebook:	No
Linkedin:	No
Github:	Yes

Questionnaire for User Interface Satisfaction

OVERALL REACTIONS TO THE SOFTWARE			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
1		terrible								x			wonderful	
2		difficult								x			easy	
3		frustrating									x		satisfying	
4		inadequate									x		adequate	
5		dull								x			stimulating	
6		rigid						x					flexible	
SCREEN			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
7	Reading characters on the screen	hard to read									x		easy to read	
8	Highlighting simplifies task	not at all								x			very much	
9	Organization of information	confusing									x		very clear	
10	Sequence of screens	confusing									x		very clear	
TERMINOLOGY AND SYSTEM INFORMATION			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
11	Use of terms throughout system	inconsistent								x			consistent	
12	Rerminology related to task	never									x		always	
13	Position of messages on screen	inconsistent									x		consistent	
14	Prompts for input	confusing									x		clear	
15	Computer nforms about what its progress	never							x				always	
16	Error messages	unhelpful								x			helpful	
LEARNING			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
17	Learning to operate the system	difficult								x			easy	
18	Exploring new features by trial and error	difficult									x		easy	
19	Remembering names and use of commands	difficult											easy	x
20	Performing task is straightforward	never									x		always	
21	Help messages on the screen	unhelpful											helpful	x
22	Supplemental reference materials	confusing											clear	x
SYSTEM CAPABILITIES			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
23	System speed	too slow									x		fast enough	
24	System reliability	unreliable								x			reliable	
25	System tends to be	noisy					x						quiet	
26	Correcting your mistakes	difficult							x				easy	
27	Designed for all level of users	never						x					always	

	List the most **negative** aspect(s)		List the most **positive** aspect(s)
1	Opciones poco intuitivas	1	Facil de usar
2		2	Ofrece una gran variedad de temas de apoyo
3		3	

TEST PERSON INFORMATION

Age:	22
Gender:	Male
Level of education:	Graduated
Hours spent using a computer per week on average:	60
Do you blog?	No
Do you usually read a newspaper?	No
Do you know/use:	
Facebook:	X
Linkedin:	X
Github:	X

Questionnaire for User Interface Satisfaction

OVERALL REACTIONS TO THE SOFTWARE			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
1		terrible										X	wonderful	
2		difficult										X	easy	
3		frustrating										X	satisfying	
4		inadequate										X	adequate	
5		dull									X		stimulating	
6		rigid										X	flexible	
SCREEN			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
7	Reading characters on the screen	hard to read										X	easy to read	
8	Highlighting simplifies task	not at all										X	very much	
9	Organization of information	confusing										X	very clear	
10	Sequence of screens	confusing										X	very clear	
TERMINOLOGY AND SYSTEM INFORMATION			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
11	Use of terms throughout system	inconsistent									X		consistent	
12	Rerminology related to task	never				X							always	
13	Position of messages on screen	inconsistent										X	consistent	
14	Prompts for input	confusing										X	clear	
15	Computer nforms about what its progress	never										X	always	
16	Error messages	unhelpful										X	helpful	
LEARNING			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
17	Learning to operate the system	difficult										X	easy	
18	Exploring new features by trial and error	difficult										X	easy	
19	Remembering names and use of commands	difficult										X	easy	
20	Performing task is straightforward	never										X	always	
21	Help messages on the screen	unhelpful										X	helpful	
22	Supplemental reference materials	confusing										X	clear	
SYSTEM CAPABILITIES			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**NA**
23	System speed	too slow										X	fast enough	
24	System reliability	unreliable										X	reliable	
25	System tends to be	noisy										X	quiet	
26	Correcting your mistakes	difficult										X	easy	
27	Designed for all level of users	never										X	always	

	List the most **negative** aspect(s)		List the most **positive** aspect(s)
1	Que a algunos usuarios (a mi) me explota al adjuntar una imagen, aunque puede que sea por cosas de android	1	Es muy ágil y responde rapidísimo
2	Que el usuario que crea las publicaciones no puede comentar y quizá es util para añadir algun matiz	2	Visualmente es muy limpio e intuitivo
3		3	Ocupa poco y es útil

TEST PERSON INFORMATION

Age:	23
Gender:	Hombre
Level of education:	Universitario
Hours spent using a computer per week on average:	50
Do you blog?	Sí
Do you usually read a newspaper?:	En formato digital
Do you know/use:	
Facebook:	Sí
Linkedin:	Sí
Github:	Sí