Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

Ingeniería inversa de una app Android open-source y migración de XML a Jetpack Compose de una de sus vistas

---

Alumno: Jorge Aguado Recio

Tutores: Yania Crespo González-Carvajal
Juan Carlos Garrote Gascón

A mi familia, que me ha ayudado y apoyado en todo momento.

Agradecimientos

A mi familia, que siempre ha estado ahí y me ha animado para conseguir lo que me propusiese.

A mis amigos, que también me han apoyado y se han preocupado por mi a lo largo de todos estos años.

A mis compañeros de carrera que se han convertido en mis amigos, y con su ayuda todos estos años han sido más llevaderos.

A mi tutora y profesora Yania, la cual me ha orientado y ayudado en la gestión de este proyecto y en varias asignaturas a lo largo del grado.

A mi tutor Juan Carlos por estar día tras día ayudándome en cualquier aspecto y facilitándome muchas tareas en todos los ámbitos.

Gracias a todos.

Resumen

El objetivo de este trabajo es conseguir ejecutar una modernización parcial, de tipo migración (tipo de modernización cuyo objetivo es moverse por completo a una nueva tecnología), en un proyecto open-source. Se trata de una app Android con la interfaz de usuario desarrollada en XML que se desea ir migrando a Jetpack Compose, el framework moderno de creación de IU utilizado de manera oficial en Android.

Para conseguir este objetivo, se realizará un trabajo de ingeniería inversa sobre el estado actual de la aplicación, con el objetivo de obtener documentación de diseño. A partir de ese punto se abordará la modernización parcial de la app mediante migración. La contribución al proyecto open-source será doble, aportando documentación de diseño, que está desactualizada en este momento, y la propia modernización en sí.

Abstract

The objective of this work is to achieve a partial modernization, of the migration type (a modernization approach aimed at fully transitioning to new technology), of an open-source project. It involves an Android app with its user interface developed in XML that is intended to be gradually migrated to Jetpack Compose, the officially endorsed modern UI creation framework for Android.

To accomplish this goal, a reverse engineering work will be undertaken on the current state of the application, with the aim of obtaining design documentation. From that point, the partial modernization of the app will be addressed through migration. The contribution to the open-source project will be twofold, providing design documentation, which is outdated at the moment, and the actual modernization itself.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Este trabajo de fin de grado se va a llevar a cabo junto con Izertis [21], una empresa multinacional especializada en el sector tecnológico. Su sede principal se encuentra en Gijón, aunque posee varias sedes y oficinas en otras partes de la península como Valencia o Valladolid. Además presenta alguna sede en otros países de diferente continente como, por ejemplo, México.

El equipo que trabaja en Valladolid es puramente nativo de Android. En concreto, se encuentran desarrollando la aplicación Android de *ownCloud* [36], aplicación open-source sobre la que se va a trabajar en el TFG. *ownCloud* es un servicio de almacenamiento y sincronización de archivos multiplataforma que se puede alojar en un servidor propio. La principal ventaja que tiene esta aplicación respecto a sus principales competidoras como *Google Drive*, *OneDrive* o *Dropbox*, es que al estar instalado en nuestro propio hosting permite diseñar la estructura de todo el sistema de almacenamiento, además de tener el control total sobre la privacidad de todos los archivos existentes [54].

1.2. Motivación

La principal motivación para la realización de este trabajo es la aplicación de todos los conocimientos aprendidos durante las horas de prácticas curriculares en la mejora de la aplicación open-source sobre la que se va a trabajar.

Durante los últimos años *ownCloud* ha sufrido bastantes modificaciones en cuanto a su estructura se refiere, por lo que la documentación no se encuentra totalmente actualizada. A partir de un proceso de ingeniería inversa se actualizará y se documentará parte de la aplicación actual. Además, actualmente la incorporación de nuevo personal al equipo del

proyecto requiere un proceso de formación intenso utilizando horas de trabajo de un ingeniero senior, en el que se explique la arquitectura y estructura de la aplicación, la forma de trabajo del equipo o incluso las decisiones que se han tomado para el desarrollo. Contar con una documentación actualizada permitiría la autoformación de los nuevos integrantes, siendo sólo necesario algunas sesiones para aclarar dudas con los ingenieros senior que se encuentren dentro del equipo.

Por otro lado, el actual framework de interfaz de usuario con el que se trabaja en la mayoría de las actuales aplicaciones en Android es Jetpack Compose [2], dejando de lado el desarrollo de interfaces basado en XML como se venía haciendo anteriormente. En la aplicación Android para *ownCloud* se desarrollaron las interfaces de usuario en el formato XML, por lo que durante el desarrollo de este trabajo de fin de grado se migrará una de sus pantallas al framework anteriormente mencionado.

1.3. Proyectos open-source

Un proyecto open-source en un entorno informático [33] es aquel proyecto cuyo código está diseñado para que sea accesible a todo el mundo. Todos pueden ver, modificar y distribuir el código de la forma en la que vean conveniente. Estos proyectos se desarrollan de forma diferente a otros proyectos convencionales. Todo depende de la revisión de las múltiples personas que trabajen en el proyecto, además del apoyo de la propia comunidad. Otra de las características que tienen los proyectos open-source respecto a cualquier otro tipo de proyecto es que suelen ser más económicos, flexibles y duraderos [43, 18].

En un proyecto open-source se encuentran varios roles especializados:

- **Autor**: Persona(s) que creó(crearon) el proyecto.
- **Dueño**: Personas que poseen la propiedad administrativa sobre la organización o el repositorio en el que se está desarrollando todo el proyecto.
- **Encargados**: Personas encargadas de dirigir y administrar la organización del proyecto.
- **Colaboradores**: Cualquier persona que haya aportado con alguna contribución durante el desarrollo del proyecto. En inglés se conoce con el término *contributor*.
- **Comunidad**: Personas que utilizan el producto creado. Pueden expresar su opinión acerca de la dirección que está tomando el proyecto.

Aparte de los diferentes roles, también se utilizan distintas herramientas. Estos medios facilitan el trabajo a todos los implicados en el proyecto. Las herramientas que se utilizan son las siguientes [34]:

- **Issue Tracker**: Medio a través del cual las personas que conforman el proyecto discuten los problemas relacionados con el mismo.

- **Pull request**: Lugar donde las personas discuten y revisan los cambios que han tenido lugar durante el desarrollo.
- **Foros de discusión**: Algunos proyectos utilizan estas herramientas para iniciar hilos de conversación de diversos tópicos/temas.
- **Canal de chat síncrono**: Medio a través del cual todos los participantes del proyecto pueden comunicarse diariamente para la organización interna y colectiva.

1.4. Ingeniería inversa

La ingeniería inversa es un proceso de desarrollo en el sector de la tecnología que consiste en obtener información de cómo está construido o cómo funciona un programa, objeto o sistema ya creado. Para ser más exactos el origen de la ingeniería inversa se encuentra en la ingeniería mecánica. Si en la ingeniería se diseñan los componentes de un producto para su montaje posterior, la ingeniería inversa consiste en el proceso inverso [19, 52].

1.4.1. Ingeniería inversa de software

El primer paso para llevar a cabo ingeniería inversa sobre un producto software es entender el código. Sin la comprensión del mismo resulta imposible obtener conclusiones. Durante este proceso, el código no se modifica como tal, sino que se sacan una serie de conclusiones para la reestructuración o rediseño del producto. Al aplicar ingeniería inversa sobre un producto software se puede llegar a varias conclusiones como la reducción de la complejidad del sistema, recuperación de información/documentación perdida o desactualizada o incluso la reutilización de sistemas completos [20]. Con un proceso de ingeniería inversa en el caso del software, se parte de código y los resultados del proceso serán artefactos de niveles de abstracción superior, hasta el nivel que se plantee llegar, como diseño, análisis, requisitos.

1.5. Objetivos

Los objetivos principales que se quieren conseguir con la realización de este TFG son los siguientes:

- **Comprensión y documentación de la aplicación open-source.** Como se va a trabajar sobre una aplicación open-source, será necesario entender el código así como la estructura de la propia aplicación. Una vez comprendidos los aspectos fundamentales, se llevará a cabo la actualización de la documentación a través de diagramas tanto del marco estático como dinámico.
- **Comprensión y aprendizaje de Jetpack Compose**. No se había utilizado este framework de Android anteriormente durante el transcurso del grado, por lo que la

dificultad aumentará en cierta medida. Además, por este motivo será necesario una previa formación sobre dicho framework para comprender el funcionamiento básico del mismo.

- **Migración de interfaz de usuario (IU) XML a Jetpack Compose.** Tras la formación en el framework recomendado actualmente por Android, se realizará una migración de la interfaz de usuario de una pantalla de la aplicación open-source *ownCloud* a Jetpack Compose.

1.6. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 1: Introducción. En este capítulo se describe el contexto la aplicación sobre la que se va a trabajar, al igual que la principal motivación para su desarrollo. Además se exponen los diferentes objetivos que se pretenden conseguir a lo largo de todo el desarrollo.

Capítulo 2: Tecnologías utilizadas. En este capítulo se describen todas las tecnologías que se han utilizado para el desarrollo del trabajo de fin de grado.

Parte I: Ingeniería inversa. Comprende todos los capítulos relacionados con el proceso de ingeniería inversa que se ha llevado sobre la aplicación Android de *ownCloud*.

Capítulo 3: Estudio de *ownCloud* y Clean Architecture. En este capítulo se describe el funcionamiento de *ownCloud* además del tipo de arquitectura que está asociada a la aplicación y sobre la que se va a trabajar.

Capítulo 4: Ingeniería inversa de la arquitectura de *ownCloud*: vista estática. En este capítulo se muestra la estructura de la aplicación junto con las clases que participan en los casos de usos descritos y sus relaciones de manera estática.

Capítulo 5: Ingeniería inversa de la arquitectura de *ownCloud*: vista dinámica. En este capítulo se muestra la interacción entre los diferentes objetos que participan en cada uno de los casos de usos descritos en el proyecto.

Parte II: Reingeniería: migración a Jetpack Compose. Comprende todos los capítulos relacionados con el proceso de reingeniería de uno de los casos de uso expuestos además de la migración de XML a Jetpack Compose de una de las vistas.

Capítulo 6: Rediseño de un caso de uso. En este capítulo se describe todo el proceso de rediseño del caso de uso sobre el que se ha realizado la migración. Además se explica el método que se ha utilizado para ejecutar la migración.

Capítulo 7: Implementación y pruebas del caso de uso rediseñado. En este capítulo se describe cómo se ha realizado la migración de XML a Jetpack Compose de una de las vistas de la aplicación *ownCloud*.

Capítulo 8: Conclusiones. En este capítulo se describen las conclusiones que se han obtenido durante el desarrollo y algunas posibles mejoras de cara al futuro.

Anexos. Comprende los anexos relacionados con la planificación y gestión del proyecto, la organización del trabajo y el desarrollo de todo el TFG.

Anexo A: Planificación del proyecto. En este anexo se describe el marco de trabajo que se va a utilizar así como la adaptación a este proyecto en concreto. También se define la planificación de todo el TFG y los riesgos asociados.

Anexo B: Seguimiento del proyecto. En este anexo se describe cómo se ha ido avanzando en el proyecto, qué incidencias o eventos han ocurrido y el detalle de todas las tareas que se han llevado a cabo en cada uno de los sprints.

Anexo C: Resumen de enlaces adicionales. En este apéndice se incluyen enlaces de interés del proyecto: el repositorio en el que se ha llevado a cabo la migración de XML a Jetpack Compose y el repositorio de la aplicación Android de *ownCloud*.

Capítulo 2

Tecnologías utilizadas

2.1. Tecnologías para la comunicación

2.1.1. Rocket.chat

Rocket.chat [47] es una plataforma de comunicación en tiempo real y de código abierto. Permite a equipos y comunidades comunicarse y colaborar de manera efectiva a través de mensaje de texto, mensajes de voz o incluso vídeo. Una de las ventajas que permite esta herramienta es que se pueden crear canales personalizados en los que se pueden discutir temas de diferente índole, dependiendo de la personalización. En este caso en particular se tiene un canal de dudas para el seguimiento del proyecto y otro de reuniones para acordar fecha y hora con los tutores. Otra de las ventajas que tiene esta herramienta es que puedes alojarlo en tu propio servidor, por lo que siempre habrá mayor control sobre la seguridad y privacidad de los datos. Esta aplicación está proporcionada por la propia *Escuela de Ingeniería Informática de Valladolid.*

2.1.2. Microsoft Teams

Microsoft Teams [31] es una plataforma de colaboración y comunicación desarrollada por Microsoft, que forma parte del paquete de aplicaciones de Microsoft 365. Está diseñada para facilitar el trabajo en equipo mediante la integración de chats, videollamadas, almacenamiento de archivos y aplicaciones en un solo sitio. Teams se ha convertido en una herramienta esencial para muchas organizaciones, facilitando la comunicación, colaboración y productividad en un entorno cada vez más digitalizado y remoto. A su vez, es una aplicación gratuita que se ha utilizado para poder comunicarse con los tutores de forma remota todos aquellos días en los que no había disponibilidad para realizar las tutorías de forma presencial. Esta herramienta también es proporcionada por la Universidad de Valladolid.

2.1.3. Google Meet

Google Meet [15] es una plataforma de videoconferencia desarrollada por Google. Permite a las personas realizar reuniones virtuales en tiempo real, ya sea para trabajar, estudiar u otro propósito. Ofrece varias funciones como la compartición de pantalla, un chat en directo, grabación de las reuniones, subtítulos en tiempo real o incluso incluir otras aplicaciones externas de productividad como pueden ser Google Calendar o Google Drive. Esta herramienta se empezó a utilizar sobre todo con la aparición del COVID-19. Es una aplicación gratuita a manos de todo el público y que en este proyecto se ha utilizado para realizar videollamadas con los tutores aquellos días que no era posible realizar las tutorías de forma presencial. El motivo por el que se han utilizado dos plataformas de comunicación diferentes, es porque en los primeros días de desarrollo no se tenía las credenciales para poder acceder a la cuenta de Teams, y por tanto ha sido necesario emplear esta herramienta para llevar a cabo las tutorías de forma remota.

2.1.4. Vysor

Vysor [53] es una aplicación que permite reflejar la pantalla de tu dispositivo Android e iOS en el ordenador. Con esta aplicación, los usuarios son capaces de controlar los dispositivos Android directamente a través de una conexión Wi-Fi o con un cable. La finalidad de esta aplicación es facilitar tareas como la visualización de contenido, la ejecución de aplicaciones o incluso el control del dispositivo en una pantalla mucho más grande además de tener la comodidad del teclado y del ratón. Vysor es sumamente útil para desarrolladores de aplicaciones móviles ya que permite realizar pruebas y desarrollar código de forma simultánea y más visual. Es una aplicación gratuita a manos de todo el público. En este trabajo se ha utilizado para mostrar el avance del desarrollo de la pantalla que se ha migrado a Jetpack Compose.

2.2. Tecnologías para la gestión del proyecto

2.2.1. GitHub

GitHub [13] es una plataforma en línea que utiliza Git (ver Apartado 2.2.2) como base para el control de versiones y ofrece una variedad de servicios adicionales relacionados con el desarrollo de software colaborativo. Fue fundada en 2008 y actualmente se ha convertido en una de las plataformas líderes para alojar y compartir proyectos de código abierto y privados. Algunas características a destacar son:

- **Repositorios**: Son espacios de almacenamiento virtual en los que se encuentran los ficheros de un proyecto y permiten a los usuarios alojar proyectos Git de forma remota. Cada repositorio puede contener un código fuente diferente, archivos de documentación, archivos multimedia y más contenido variado.

- **Colaboración**: Facilita la comunicación entre desarrolladores permitiéndoles trabajar de forma conjunta en proyectos compartidos. Los colaboradores pueden enviar `pull request` para proponer cambios en el código, y los propietarios del repositorio pueden revisar y fusionar estos cambios.
- **Seguimiento de problemas y solicitudes de funciones**: Proporciona herramientas para rastrear problemas (bugs) y solicitudes de funciones (feature request), lo que permite a los desarrolladores mantener un registro de las tareas pendientes y comunicarse de manera efectiva sobre los problemas y mejoras del proyecto.
- **Wikis y páginas de proyectos**: Se pueden incluir wikis para documentación adicional y páginas de proyectos para comunicar el estado y la dirección del proyecto a usuarios y colaboradores.
- **Integración continua y entrega continua (CI/CD)**: Integra herramientas CI/CD que permiten automatizar pruebas, compilaciones y despliegues de código para garantizar la calidad y la entrega eficiente del software.
- **Comunidad y descubrimientos**: Es una comunidad activa de desarrolladores que comparten y colaboran en una amplia gama de proyectos. Los usuarios pueden descubrir nuevos proyectos, contribuir en proyectos ya existentes y aprender de otros desarrolladores.

Esta plataforma se ofrece de manera gratuita y está accesible para cualquier usuario. En este proyecto, se ha empleado esta herramienta para facilitar la migración de la interfaz de usuario, aprovechando su sistema de control de versiones integrado. Además, se ha utilizado la funcionalidad de tablón para organizar el proyecto de manera efectiva. En la Figura 2.1 se muestran varias tareas que se encontraban en desarrollo durante alguno de los primeros sprints.

Cada tarea ha sido etiquetada según su naturaleza: aquellas relacionadas con documentación se han marcado como **Documentation**, mientras que las relacionadas con la creación de diagramas de la estructura de la aplicación han sido etiquetadas como **Design**. Además, se han identificado tareas de desarrollo y migración de XML a Jetpack Compose, todas ellas etiquetadas como **Development**. También existe una etiqueta denominada **Sprint**, que se pone en todas aquellas tareas que se encuentren dentro del alcance del sprint actual.

También se puede apreciar cómo el tablero presenta varias columnas que reflejan el estado de las tareas. La columna **To Do** contiene las tareas pendientes. **In Progress** muestra aquellas en proceso de desarrollo. **Review** indica las tareas en espera de revisión, y **Done** refleja las tareas completadas.

Otra cosa a mencionar sobre esta tecnología es que para el desarrollo del TFG se ha realizado un `fork` de todo el proyecto de *ownCloud*. Un `fork` consiste en una copia de un repositorio de código almacenado en la plataforma. Es comúnmente utilizado en colaboración y desarrollo de software de código abierto. Para este caso, simplemente hay que ir al repositorio principal de *ownCloud* y pulsar en uno de los botones superiores donde pone `fork`. En ese momento ya se tendrá una copia idéntica de todo el proyecto sobre la cual poder trabajar.

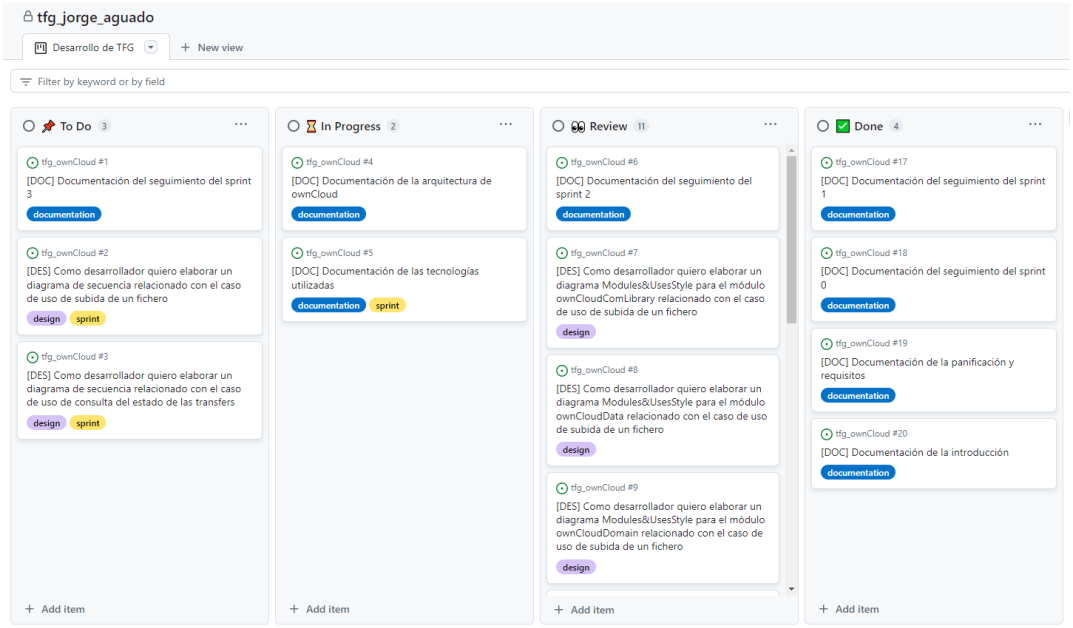

Figura 2.1: Tablero de GitHub

Una vez que ya se ha creado el `fork`, tenemos la posibilidad de crear una `issue`. Una `issue` consiste en un registro de una tarea, problema o sugerencia relacionada con el proyecto de software que se esté gestionando en *GitHub*. En este trabajo se abre una `issue` por cada tarea que se tenga que hacer para la migración de XML a Jetpack Compose. Para cada una de esas tareas se abre un `pull request` en el repositorio. Su principal función es permitir a los colaboradores proponer cambios en el código, por tanto cada vez que se vaya a realizar algún cambio sobre una rama se creará un `pull request`. Cada uno de los `pull request` incluye los cambios propuestos además de una descripción en los que se explican los motivos de esos cambios y cualquier información para los revisores. Los colaboradores pueden consultar un `pull request`, realizar comentarios, sugerir modificaciones o incluso aprobarlo para que los cambios se fusionen con la rama principal. Estos elementos se pueden asociar mutuamente pudiendo llevar a cabo un seguimiento de cada uno de los cambios que se han realizado respecto a una tarea en concreto.

2.2.2. Git

Git [12] es un sistema de control de versiones distribuido, diseñado para manejar todo tipo de proyectos, desde los más pequeños hasta los más grandes, con velocidad y eficiencia. Fue creado por Linus Torvalds en 2005 para el desarrollo del kernel de Linux, pero desde entonces se ha convertido en uno de los sistemas de control de versiones más populares y ampliamente utilizado en todos los tipos de proyectos de software. Se basa en un sistema de repositorios y ramas. Si lo que queremos es añadir por ejemplo una nueva funcionalidad a nuestra aplicación, lo que tenemos que hacer es crear una nueva rama a partir de la rama

de desarrollo normalmente nombrada como **develop**. En el caso de *ownCloud* cada rama de funcionalidad surge a partir de la rama principal, denominada **master**. Su funcionamiento se puede ver de forma gráfica en la Figura 2.2.

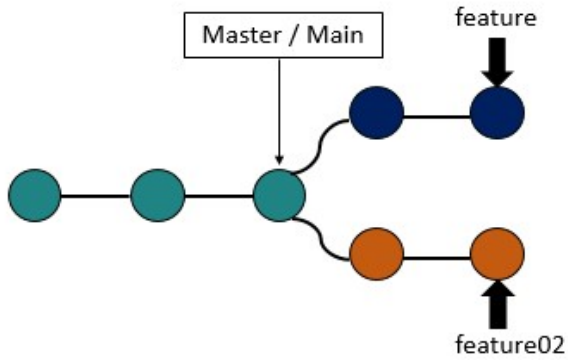

Figura 2.2: Funcionamiento principal de las ramas de Git [11]

Una vez que se ha trabajado en ese cambio y después de haber pasado satisfactoriamente por varias fases de aceptación, revisión e integración se puede fusionar esa rama en la rama master. Con esto se consigue añadir esa funcionalidad que se ha desarrollado en la aplicación principal y por tanto que ese cambio que se ha hecho, lo pueda ver el usuario que utiliza dicha aplicación en este caso. Para poder llevar a cabo todo este proceso, se han de seguir unos pasos (forma de trabajo establecida en *ownCloud*) para poder tener la rama actualizada respecto a la rama principal además de evitar cualquier tipo de conflicto en el código. A continuación se muestran algunos comandos [6] útiles para dicho proceso:

- `git fetch`: Se utiliza para recuperar los últimos cambios de un repositorio remoto sin fusionarlos automáticamente con la rama local actual. Esto significa que el comando obtiene la última versión de todas las ramas del repositorio remoto que no se tienen localmente y actualiza la información de seguimiento de las ramas remotas, pero no modifica el trabajo local en absoluto.

- `git rebase`: Se utiliza para reorganizar y modificar la historia de commits en una rama. En lugar de fusionar como lo hace `git merge`, reescribe la historia de la rama actual para que parezca que se basa en el punto más reciente de la rama especificada. El uso común de este comando es integrar cambios de una rama secundaria en una rama principal de manera más limpia y ordenada, evitando así líneas de tiempo confusas y commits redundantes. Adicionalmente existe una forma de utilizar este comando de forma interactiva. Esto se hace con `git rebase -i <commit>`. Esta funcionalidad te permite revisar y modificar la historia de commits antes de fusionarlos con otra rama. Al ejecutar este comando aparecerá una lista de todos los commits a partir del cual se ha ejecutado y una serie de instrucciones para poder manipular esa historia. También se puede dar el caso de que salte un conflicto de la rama en la que se esté trabajando con la rama principal. En el momento en el que salta ese conflicto será necesario resolverlo.

Una vez que se ha resuelto se comprueba con el comando `git status`, el cual muestra si existen más conflictos en los archivos respectivos. Si todo está correcto se comprueba el siguiente commit utilizando el comando `git rebase --continue`. Estos pasos se repiten hasta que se hayan solucionado todos los conflictos.

- `git add`: Se utiliza para agregar cambios realizados en archivos específicos al área de preparación (*staging area*). Este área de preparación es un paso intermedio antes de confirmar los cambios mediante `git commit`.
- `git commit`: Se utiliza para confirmar los cambios realizados en los archivos de un repositorio local. Al ejecutarlo, se abre un editor de texto donde puedes escribir un mensaje que describe los cambios que se han realizado durante esta confirmación. Este mensaje es sumamente importante ya que proporciona contexto sobre todos los cambios realizados. La principal ventaja que tiene Android Studio (herramienta que se explicará más adelante) es que presenta un botón que fusiona tanto el comando `git add` como el comando `git commit`, facilitando la tarea en gran medida. El funcionamiento de este comando junto con el comando `git add` se puede apreciar en la Figura 2.3.

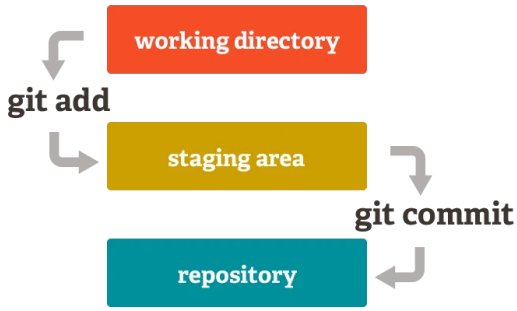

Figura 2.3: Funcionamiento de `git add` y `git commit` en el área de preparación, árbol de trabajo y repositorio [29]

- `git log`: Se utiliza para mostrar el historial de commits en una rama específica. Al ejecutarlo se despliega una lista de los commits realizados en esa rama, mostrando información como el autor, la fecha y hora y el mensaje asociado a ese commit.
- `git reset`: Se utiliza para deshacer cambios en el área de preparación y/o en el árbol de trabajo. Se puede utilizar de varias formas dependiendo del contexto. `git reset <nombre del archivo>` para quitar un archivo específico del área de preparación, manteniendo los cambios en el árbol de trabajo. `git reset --hard` para deshacer todos los cambios en el árbol de trabajo y en el área de preparación.
- `git checkout`: Se utiliza para cambiar entre ramas, restaurar archivos y deshacer cambios en el directorio de trabajo. La funcionalidad varía dependiendo de los argumentos que se le pase. `git checkout <nombre de la rama>` para cambiar a otra rama en el repositorio. Esto permite trabajar en diferentes ramas de desarrollo de forma independiente. `git checkout --<nombre del archivo>` para restaurar ese archivo a su estado tal como estaba en el último commit. `git checkout -b <nombre de la nueva rama> <hash del commit>` para crear una nueva rama basada en un commit específico.

- `git push`: Se utiliza para enviar los cambios locales a un repositorio remoto. Esto permite sincronizar el trabajo, compartiendo los commits con otros colaboradores o actualizando el estado del repositorio remoto con los cambios locales. Este comando no permite subir el contenido cuando existen diferencias entre lo hay que en remoto y lo que hay en local. Por este motivo, existe una opción `git push -f` que se utiliza para forzar la subida de los archivos. Esta opción es peligrosa, pero en muchas ocasiones resulta necesaria, ya que sin ella el propio Git no te permite subir los archivos al repositorio remoto.

2.3. Tecnologías para el análisis, diseño y documentación

2.3.1. Overleaf

Overleaf [35] es una plataforma en línea que permite a los usuarios escribir, editar y colaborar en documentos LaTeX en tiempo real. LaTeX es un sistema de composición de textos ampliamente utilizado para la creación de documentos científicos y técnicos ya que ofrece un alto control sobre el diseño y la estructura del documento. Esta aplicación simplifica el proceso de escritura además de tener una compilación automática, control de versiones, plantillas predefinidas y la capacidad de compartir documentos con otros usuario en tiempo real, además de trabajar de forma colaborativa. Es una aplicación gratuita a manos de todo el público y que ha servido en este trabajo para desarrollar toda la memoria.

2.3.2. Astah Professional

Astah Professional [5] es una herramienta de modelado visual y diseño de software utilizada por desarrolladores y profesiones de la ingeniería del software para crear diagramas UML, diagramas de flujo de datos, diagramas de clases, diagramas de secuencia y otro tipo de diagramas que ayudan en el proceso de diseño y documentación de software. Esta herramienta no es gratuita y es necesario pagar una licencia para uso, pero en este caso la Escuela de Ingeniería Informática proporciona una licencia para estudiantes. En este proyecto la utilidad de esta herramienta ha sido para crear todos los diagramas relacionadas con la estructura de *ownCloud*.

2.4. Tecnologías para el desarrollo del proyecto

2.4.1. Android Studio

Android Studio [10] es un entorno de desarrollo integrado (IDE) creado por Google para desarrollar aplicaciones móviles Android. Proporciona herramientas y recursos para escribir código, depurar, compilar y probar aplicaciones Android de manera eficiente. Además integra

funcionalidades como el diseño de interfaces de usuario, gestión de versiones y compatibilidad con diferentes dispositivos Android. Este entorno de desarrollo es gratuito y está a disposición de todo el mundo. Se ha utilizado la versión `Android Studio Hedgehog | 2023.1.1 Patch 2` para el desarrollo del TFG.

2.4.2. Android

Android [1] es un sistema operativo pensado para dispositivos móviles al igual que otros sistemas operativos como pueden ser iOS, Symbian o incluso Blackberry OS. La diferencia de Android respecto a estos sistemas operativos es que está basado directamente en Linux. En un principio fue desarrollado por Android Inc, sin embargo más tarde fue adquirido por Google LLC en el año 2005. Dos años después fue presentado junto con la fundación de Open Handset Alliance para avanzar los estándar abiertos de los dispositivos móviles. Respecto algunos temas del TFG, en Android existen dos formas de para desarrollar interfaces de usuario. La primera y más antigua es a través de XML, en el que se van definiendo etiquetas junto con sus atributos. La otra forma de hacerlo es la que actualmente se aplica en muchas aplicaciones Android y lo que la documentación de Android Developers considera como oficial: Jetpack Compose. Ambas tecnologías se explicarán en el Apartado 2.4.4 y Apartado 2.4.5 respectivamente.

2.4.3. Kotlin

Kotlin [26] es un lenguaje de programación moderno y estáticamente tipado que se ejecuta en la máquina virtual de Java (JVM) y también puede ser compilado a JavaScript o código nativo. Fue desarrollado por JetBrains y se anunció por primera vez en 2011. Kotlin es completamente interoperable con Java, lo que significa que puede llamar y ser llamada desde código Java sin ningún tipo de problema. Esta característica es uno de sus mayores puntos fuertes. Además está diseñado para reducir la cantidad de código necesario para escribir programas en comparación con Java. Esto incluye menos verbosidad en las declaraciones y la eliminación de patrones de diseño redundantes. Por otro lado, Kotlin introduce el manejo de nulabilidad a nivel de lenguaje y funciones de orden superior (lambdas). Es el lenguaje recomendado para desarrollar Android nativo y que, hoy en día, ha sustituido a Java. En este proyecto es el lenguaje que se ha utilizado para llevar a cabo la migración de la interfaz de usuario.

2.4.4. XML

XML [55] es un lenguaje de marcado similar a HTML. Sus siglas significan Extensible Markup Language. En Android su principal uso es para definir la disposición y los elementos que conforman la interfaz de usuario de una aplicación. Esto incluye la disposición de elementos visuales como pueden ser botones, campos de texto, imágenes, etc.. Android proporciona por defecto un conjunto de etiquetas para definir cada uno de esos elementos como

pueden ser: `LinearLayout`, `RelativeLayout`, `TextView`, `Button`, entre otras muchas. Además de aplicarse a la interfaz de usuario también se utiliza para definir otro tipo de recursos estáticos como cadenas de texto, estilos, colores, dimensiones, menús, animaciones, etc... Este tipo de recursos se definen en archivos separados y se utilizan en la aplicación para mantener una separación clara entre la lógica de la aplicación y los elementos visuales y estéticos. En XML el enfoque es imperativo, es decir: se describe la estructura directamente y se aplican los cambios manualmente cuando es necesario. Además para poder ver los cambios que se realizan es necesario compilar y ejecutar la aplicación, lo que puede resultar un proceso bastante lento para la iteración y desarrollo de la IU.

2.4.5. Jetpack Compose

Jetpack Compose [2] es un conjunto de bibliotecas modernas de Android que simplifica y acelera el desarrollo de la interfaces de usuario nativas. Está diseño para reemplazar el tradicional enfoque basado en XML y ofrecer una forma más intuitiva y declarativa de construir interfaces de usuario en Android. Algunas de las características que presenta Jetpack Compose son:

- **Declarativo y funcional**: En lugar de definir directamente las interfaces y manipularlas a través de un código procedural permite describir directamente como debería ser la interfaz de usuario y de forma automática se encarga de renderizarla eficientemente.
- **Funciones Composables**: Todas las interfaces de usuarios se construyen a partir de funciones composables las cuales reciben la etiqueta `@Composable`. Estos composables pueden tener a su vez otros composables, lo que permite construir interfaces complejas y modulares de manera muy intuitiva.
- **Preview en tiempo real**: Permite visualizar cómo se verá y se comportará la interfaz de usuario en tiempo real mientras se está desarrollando. Esto agiliza el proceso de diseño y desarrollo al proporcionar la retroalimentación instantánea al realizar los cambios respectivos.
- **Interoperabilidad**: Es totalmente compatible con cualquier código y bibliotecas existentes de Android, lo que significa que se puede integrar fácilmente en proyectos existentes. Al mismo tiempo en un proyecto se permite que convivan simultáneamente tanto XML como Jetpack Compose sin ningún problema, resultando muy útil en la migración de aplicaciones de gran tamaño como es en el caso de este trabajo.
- **Soporte para el ciclo de vida Android**: Se integra perfectamente con el ciclo de vida de cualquier aplicación Android, gestionando automáticamente la actualización y destrucción de las vistas de la interfaz de usuario.

De forma resumida, se podría decir que Jetpack Compose ofrece una forma moderna y potente, con una experiencia de desarrollo más fluida, productiva y mantenible en comparación con aquellas interfaces de usuario desarrolladas a partir de XML y código procedural.

Para el desarrollo de parte de este trabajo de fin de grado se ha tomado como referencia el Trabajo de Fin de Máster (TFM) de **Juan Carlos Garrote Gascón** [23], el cual consistía en una guía para migrar interfaces de usuario de XML a Jetpack Compose.

2.4.6. Gradle

Gradle [17] es una herramienta de automatización de construcción de proyectos de código con soporte para diferentes acciones del ciclo de vida del código, y muy importante para la gestión de dependencias. Es especialmente popular en el ecosistema de desarrollo de aplicaciones Android, donde se utiliza para construir, probar y desplegar aplicaciones. El proceso de construcción se basa directamente en la compilación, enlazado y empaquetado del código. Es muy popular por el hecho de que es capaz de soportar varios lenguajes como por ejemplo Java, Scala, C/C++ y Groovy, pero como se ha mencionado anteriormente, su fuerte se encuentra en aplicaciones Android. [49]

Parte I

Ingeniería inversa

Capítulo 3

Estudio de ownCloud y Clean Architecture

3.1. ownCloud

La aplicación open-source *ownCloud* [36] es sobre la que gira todo el proyecto. A pesar de que ya se ha explicado ligeramente su funcionamiento principal en el Apartado 1.1, en esta sección se llevará a cabo una descripción con más detalle.

ownCloud es un servicio de almacenamiento en la nube y sincronización de archivos multiplataforma que se puede alojar en un servidor propio. La principal ventaja, como se mencionó en el Capítulo 1, que tiene esta aplicación sobre el resto de aplicaciones con una funcionalidad parecida es que, al estar instalado en nuestro propio *hosting*, permite diseñar la estructura de todo el sistema de almacenamiento además de tener el control total sobre la privacidad de todos los archivos existentes.

Actualmente *ownCloud* cuenta con varios clientes dependiendo de la plataforma que se utilice. Tiene una versión web, una versión de escritorio, una versión de Android y una versión de iOS. Todo lo que se va a desarrollar en este trabajo está relacionado con la aplicación en su versión de Android. A diferencia del resto de equipos de desarrollo, los cuales se encuentran en Alemania, el equipo de Android tiene su sede en las oficinas de Izertis en Valladolid. Además Izertis también cuenta con un desarrollador back-end, pero se encuentra en un equipo totalmente distinto. Continuando con el back-end, existen dos tipos de servidores: *oC10* (*ownCloud* 10) y *oCIS* (*ownCloud* Infinite Scale) [38].

En primer lugar, *oC10* es la versión de *ownCloud* que presenta una funcionalidad similar a otras aplicaciones del estilo. Permite el almacenamiento de cualquier tipo de archivo en una estructura de ficheros que haya definido el usuario. Por otro lado, está *oCIS* (*ownCloud* Infinite Scale). La principal diferencia de esta versión respecto a la mencionada anteriormente es que su arquitectura está basada en microservicios en la nube. No depende directamente

de paquetes externos de PHP o incluso de una base de datos. Asimismo, esta versión cuenta con *spaces* [37], cuyo principal propósito es el trabajo colaborativo. Un *space* es un elemento en la estructura de ficheros que se puede crear como cualquier otra carpeta o archivo cuya finalidad es favorecer a los equipos, organizaciones e instituciones que trabajan conjuntamente de forma remota. Un usuario con el rol de *Admin* o *Spaces admin* puede crear un *space* con total libertad, además de invitar a todas aquellas personas que desee con un simple enlace. Todos aquellos que tengan permiso para acceder al *space* en cuestión tendrán en todo momento el acceso a los archivos que haya en su interior. En el caso de que se produzca algún abandono en un equipo, es suficiente con negar el acceso a ese *space*, pues ya no podrá ver ni modificar su contenido. Por último se debe comentar, que como cualquier aplicación o sistema de funcionalidad similar, la sincronización de los archivos es el aspecto más importante de *ownCloud*. Esto permite a un usuario tener todo su contenido actualizado independientemente del cliente en el que se hayan hecho las acciones.

Por otro lado existen otras funcionalidades que son idénticas para ambas versiones. *ownCloud* permite la creación de la estructura de directorios que quiera el propio usuario. Se pueden crear carpetas y descargar y subir archivos a libre albedrío. También se pueden compartir todos esos elementos a través de un enlace público (con contraseña o fecha de expiración) o directamente añadir invitados o grupos de usuarios a partir de su nombre. Otra características a destacar es que se pueden reproducir vídeos en streaming con la propia aplicación. También se pueden tener varias cuentas en un mismo dispositivo siendo posible conectarse a varios servidores y acceder a los elementos con un par de clicks.

En cuanto a la estructura de la aplicación Android, hace unos años era monolítica, sin una arquitectura definida, en la que todas las clases estaban dentro de un mismo módulo. Esto resultaba caótico ya que el tamaño del sistema era inmenso y, por tanto, no había claridad en cuanto a su estructura. Las clases de interfaz de usuario accedían directamente a las bases de datos, a llamadas de red o incluían lógica de negocio. Por este motivo se tuvo que llevar a cabo un proceso de reestructuración total, a manos de dos desarrolladores de media aproximadamente. Con este número de personas trabajando se tardó unos 4 años en finalizar todo el proceso. Éste incluía el cambio de lenguaje de Java a Kotlin, aunque actualmente no se haya transformado al 100 % todas las clases (se podría decir que la aplicación está reestructurada al 90 %). Para no tener la aplicación parada por completo mientras se llevaba a cabo este proceso se decidió llevar a cabo el desarrollo en una rama totalmente separada de la principal. En primer lugar se cambió por completo los *shares* (archivos compartidos). Más tarde se trabajó en la autenticación, y por último en la sincronización de ficheros. Con esta reestructuración se consiguió pasar de un módulo a cuatro totalmente diferentes, de los cuales se hablará en el Apartado 4.1. De esta manera se consiguió tener una arquitectura más clara y desacoplada como esqueleto principal de la aplicación.

3.2. Clean Architecture

Este tipo de arquitectura, documentada por Robert Martin [46], permite construir un sistema mantenible, escalable y de calidad basándose en la separación de responsabilidades. En todo momento se busca alta cohesión y bajo acoplamiento. El foco del sistema en una arquitectura limpia se centra en la lógica de negocio. El diagrama principal de esta arquitectura se asemeja ligeramente a una cebolla, en la que el centro de la misma sería esa capa de lógica de negocio que se ha mencionado. Todo se centra en esa capa, lo que implica que las relaciones y dependencias que existen entre las capas externas tiene que ir hacia el centro del diagrama. De ninguna manera es posible tener dependencia desde la capa central, de dominio, hacia las capas tanto de interfaz de usuario como de datos. Tampoco es posible tener dependencias de la capa de datos a la capa de interfaz de usuario. Toda esta estructura se puede apreciar en la Figura 3.1.

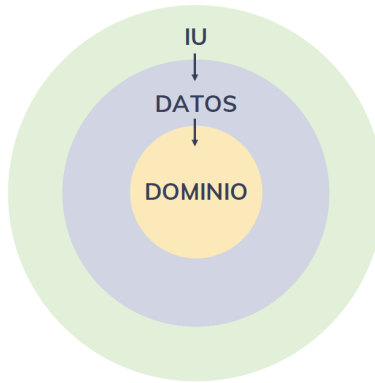

Figura 3.1: Estructura general de una arquitectura limpia

Empezando por la parte superior o capa más externa de la estructura, se tiene la **capa de interfaz de usuario** o capa de presentación. Esta capa, tal y como se muestra en la Figura 3.2, contiene los componentes principalmente relacionados con lo que se va a mostrar por pantalla al usuario. Dentro de esta capa se pueden distinguir dos elementos principales:

- **Elementos de IU**. Clases directamente relacionadas con la interfaz de usuario, como por ejemplo: `Activities, Fragments, Adapters..`
- **State Holders**. Clases encargadas de hacer los cálculos que necesita la interfaz de usuario y de comunicarse con las capas inferiores (`ViewModels`)

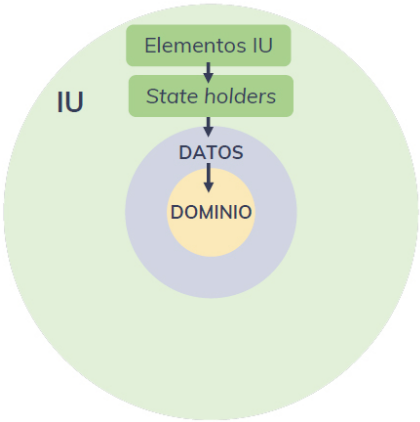

Figura 3.2: Estructura de la capa de presentación de una arquitectura limpia

En este caso el patrón arquitectónico utilizado en *ownCloud* y que tienen la mayoría de aplicaciones en Android es el MVVM (Model-View-ViewModel). Este patrón permite llevar a cabo una programación reactiva. Las tres partes en detalle que conforman este patrón son las siguientes:

- **Model** (M): Todos los datos que se manejan en la aplicación. En este caso, puesto que MVVM se encuadra dentro de la capa de IU, el modelo se refiere a la capa de datos.
- **View** (V): Interfaz de usuario, encargada de mostrar por pantalla todos los datos del modelo.
- **ViewModel** (VM): Clases encargadas de realizar los cálculos para la interfaz de usuario y preparar los datos observables para notificar de cambios a la vista. En caso de que se produzca algún cambio en los datos, automáticamente notificará a la vista para que esos cambios le lleguen al usuario final a través de la interfaz.

Justo debajo de la capa de interfaz de usuario (si consideramos el diagrama de cebolla) se encuentra la **capa de datos**, la cual contiene la lógica encargada de recuperar los datos de sus respectivas fuentes. Dentro de esta capa se encuentra la implementación de los repositorios, que se encargan de trabajar con distintas fuentes de donde se recuperan los datos para devolverlos en estado usable (patrón repositorio [41]), las interfaces e implementaciones de las clases data source, que se encargan de acceder a una sola fuente de datos tanto remotas o locales, y también contiene las clases de utilidad para acceder a fuentes de datos (`DAOs`) y entidades para bases de datos. Todo esto se puede ver reflejado en la Figura 3.3.

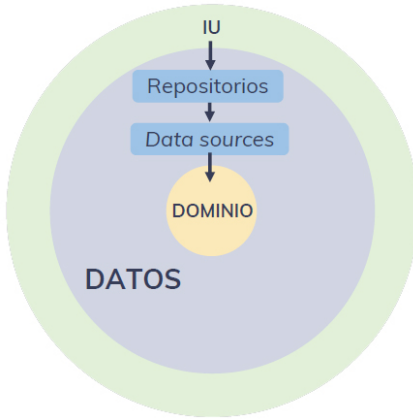

Figura 3.3: Estructura de la capa de datos de una arquitectura limpia

Por último se encuentra la **capa de dominio** o capa de la lógica de negocio, la cual es la parte central de una arquitectura limpia. Esta capa contiene la lógica de negocio de toda la aplicación. Constituye la capa base y no posee dependencias con otras capas. Tal y como se muestra en la Figura 3.4, dentro de esta capa se pueden encontrar los siguientes elementos:

- **Modelo**. Son todas las clases de dominio que conforman la aplicación en su totalidad.
- **Casos de uso**. Son las clases que utilizan el modelo y que realizan las acciones de la aplicación requeridas por el usuario.
- **Interfaces de los repositorios de la capa de datos**. De esta manera los casos de uso tienen dependencias hacia estas interfaces, que se encuentran en esta capa, y no hacia su implementación, que se encuentran en la capa de datos.

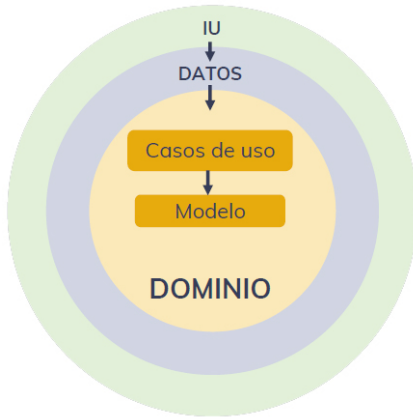

Figura 3.4: Estructura de la capa de dominio de una arquitectura limpia

Dentro de una Clean Architecture, para sistemas orientados a objetos se toman como referencia los principios *SOLID* [42] (Figura 3.5). A continuación se irá entrando más a detalle en cada uno de los principios.

- **Single Responsability** (S): Única responsabilidad o razón para cambiar. En otras palabras, cada clase debe centrarse en realizar única y exclusivamente una sola cosa. Esta propiedad es la más importante dentro de todos los principios y favorece la cohesión. Un contraejemplo de cumplimiento de este principio sería contar con una una única clase encargada de la llevar toda la lógica del programa, de almacenar cualquier tipo de dato en la base de datos, realizar llamadas a la red o incluso pintar la interfaz y mostrarle información por pantalla al usuario.
- **Open/Closed** (O): Clases abiertas a extenderse (añadir nueva funcionalidad) y a cerradas a modificarse (cambiar el código). Para conseguir cumplir este principio se pueden añadir nuevos métodos a las clases correspondientes pero, en estos casos, resulta mucho más útil utilizar la herencia.
- **Liskov Substitution** (L): Las clases deben ser totalmente sustituibles por sus subclases sin alterar el funcionamiento del sistema. Cuando se crea una subclases, no se debe alterar el comportamiento de la misma. Como última opción, redefinir el comportamiento siempre y cuando sea una sustitución totalmente válida.
- **Interface Segregation** (I): Las interfaces deben ser tan específicas como podamos. Es mejor separar la funcionalidad del sistema en varias interfaces específicas que en unas pocas de propósito general. Un claro contraejemplo sería una interfaz que tenga muchos métodos que las clases que la implementan, los dejen vacíos o incluso lancen excepciones porque no se necesitan.
- **Dependency Inversion** (D): Mejor depender de abstracciones que de concreciones. Siempre que sea posible, las dependencias deben ser hacia interfaces en lugar de clases concretas. Nuevamente se muestra un ejemplo de incumplimiento de este principio: una clase, que implementa un determinado servicio en una aplicación, instancia en su constructor objetos de clases concretas, además de llamar a sus métodos directamente. De esta manera se estaría creando un alto acoplamiento entre todas esas clases que no son interfaces.

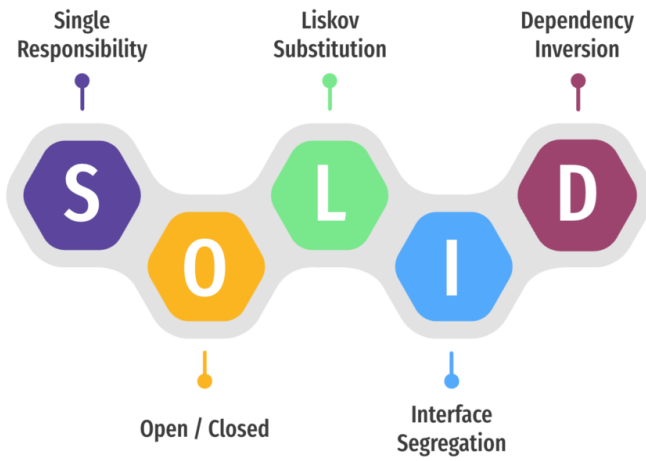

Figura 3.5: Principios *SOLID* [30]

La Clean Architecture presenta una serie de ventajas y desventajas respecto a otros tipos de arquitectura. Dentro de las ventajas destacan:

- Se consigue mantenibilidad, calidad y escalabilidad de todo el sistema.
- Resulta más fácil trabajar en equipo dentro de la misma aplicación sin que tengan lugar conflictos en el código.
- Es mucho más fácil probar la aplicación, las clases se encuentran mucho más aisladas.
- Los fallos se pueden investigar de una forma más metódica. El motivo es el mismo que la ventaja anterior: todo se encuentra más localizado.

Estas son las principales desventajas que presenta la arquitectura limpia o Clean Architecture:

- La curva de aprendizaje puede ser bastante elevada en comparación con sistemas que no presentan esta arquitectura.
- A veces, seguir al pie de la letra la guía nos puede llevar a la sobreingeniería para cosas que se pueden hacer de una manera mucho más simple.

Capítulo 4

Ingeniería inversa de la arquitectura de ownCloud: vista estática

4.1. Clean Architecture en ownCloud: vista estática

Una vez que ya se ha hecho una introducción de lo que es *ownCloud* (en el Apartado 3.1) y cómo es una arquitectura limpia (en el Apartado 3.2), se mostrará cómo están ambas partes relacionadas entre sí.

Para empezar, el proyecto se encuentra dividido en cuatro módulos totalmente diferentes. Está el módulo *owncloudApp* el cual corresponde con la capa de interfaz de usuario, los módulos *owncloudData* y *owncloudComLibrary*, los cuales entrarían dentro de la capa de datos y por último el módulo *owncloudDomain*, el cual corresponde corresponde con la capa de dominio o lógica de negocio.

Entrando más a detalle en la estructura de la aplicación, se mostrarán varios diagramas en los que se observa de forma clara todo lo mencionado anteriormente. Debido a la gran amplitud de la propia aplicación se han elegido dos casos de uso para mostrar la arquitectura, sus clases y todas las relaciones que existen entre las diferentes capas. Esos dos casos de uso son: **consulta de estado de las subidas** (pantalla en la que se trabajará en la migración) y **subida de un fichero**. El hecho de que haya dos casos de uso, es porque el primero no utiliza el módulo *owncloudComLibrary*, pues no presenta ninguna llamada de red hacia un servidor externo, mientras que en el segundo caso de uso sí que se utiliza la arquitectura en su totalidad.

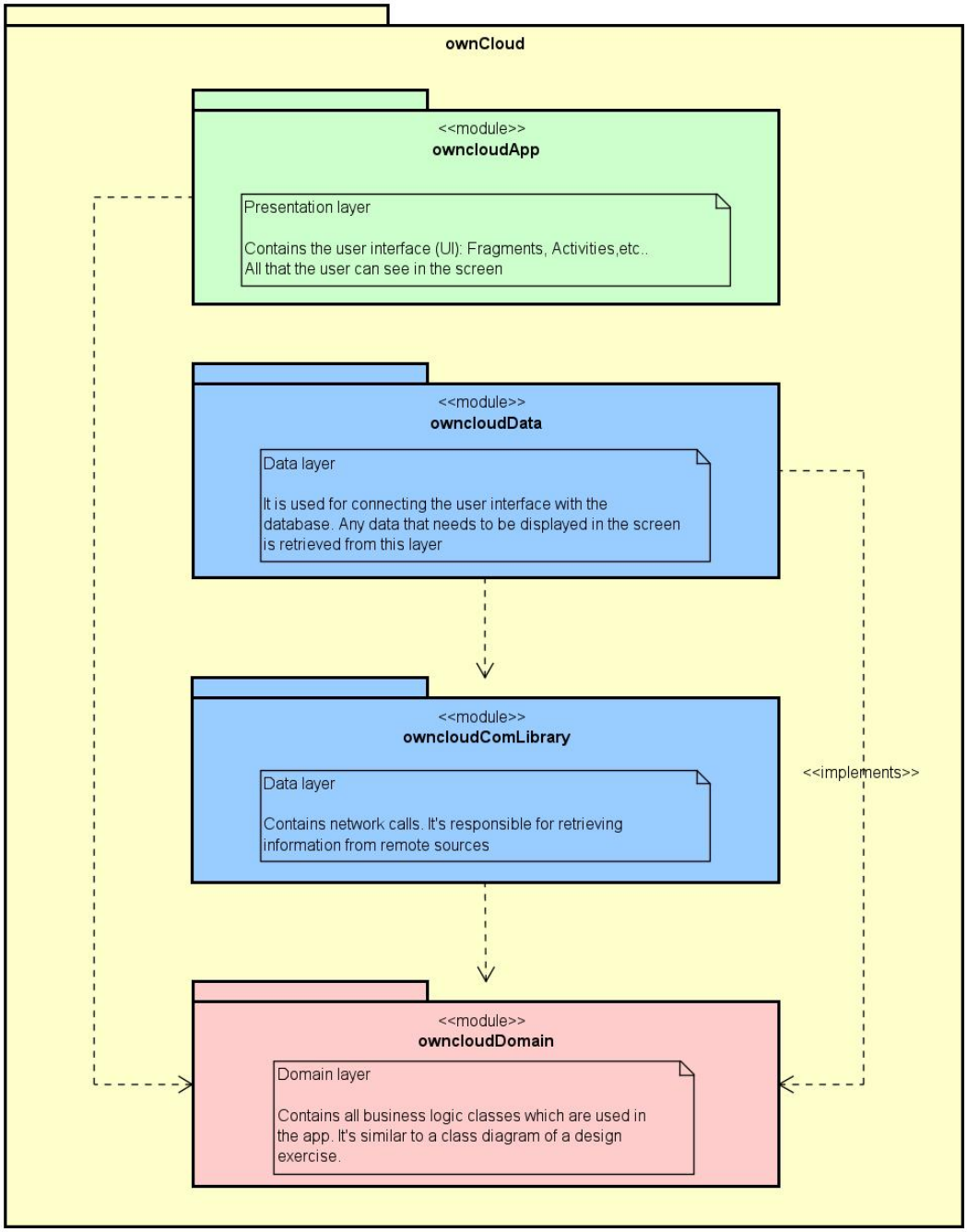

Figura 4.1: Arquitectura principal de *ownCloud*

4.1.1. Caso de uso genérico

Como se puede ver en la Figura 4.1, la arquitectura de la app Android de *ownCloud* sigue esa Clean Architecture de la que se ha hablado en el Apartado 3.2. La capa de dominio (*owncloudDomain*) corresponde al centro de la arquitectura, considerándose la capa "sumidero" pues ninguna relación puede salir de ese módulo hacia módulo superiores.

Se han establecido una serie de colores para asignar a cada módulo de la aplicación, dependiendo de la capa a la que correspondan, en los diagramas que documentan la arquitectura. Esto también se verá reflejado en las clases que se encuentran en cada uno de los módulos. El color verde corresponde a todo aquello relacionado con la capa de presentación, en este caso al módulo *owncloudApp* (el único en dicha capa). El color azul se corresponde a todo aquello que pertenezca a la capa de datos, ya sea parte del módulo *owncloudData* u *owncloudComLibrary*. Finalmente, el color rojo está asociado a todo lo que se encuentre dentro de la capa de dominio. En este caso particular todo aquello que esté dentro del módulo *owncloudDomain*.

La estructura de un caso de uso genérico se puede observar en la Figura 4.2. La explicación del mismo seguirá la disposición de las capas de la clean architecture. Dentro del módulo de presentación, se identifican dos clases principales: un `Fragment/Activity` (dependiendo del caso de uso) y un `ViewModel`. El `Fragment/Activity` constituye la parte visual de la aplicación, todo aquello que el usuario ve a través de la pantalla. Cuando se produce alguna interacción del usuario con la interfaz, se llama directamente al `ViewModel`. En todos los casos de uso, se establece una asociación entre un `Fragment` o `Activity` con uno o varios `ViewModels`, de ahí esa relación *1...**. También existe la posibilidad de que haya varias `Activities` o `Fragments` asociados entre sí. Ocurre lo mismo en la otra dirección, un `ViewModel` puede estar relacionado con una o más clases de IU de forma simultánea. Dependiendo de cada caso de uso se pueden tener más o menos clases con diferentes funciones, pero todas ellas cumplen al menos con esta estructura mencionada.

La siguiente capa a tratar es la capa de datos, más concretamente se iniciará con el módulo *owncloudData*. Aquí hay algo en particular en el diagrama, y es que existen clases que presentan un color más claro que la clase `Repository`. Esto es debido a que todas esas clases no siempre se utilizan en todos los casos de usos. Dependiendo de la funcionalidad, se prescindirá de algunas y por ese motivo se ha querido reflejar en el diagrama. Dentro de este primer módulo, siempre nos encontraremos con el `Repository`. Esta clase es la implementación del repositorio presente en la capa de dominio, la cual se definirá en último lugar. Esta implementación puede tener una relación directa con un `LocalDataSource` y un `RemoteDataSource` (no siempre se dan los dos casos de forma simultánea). La primera se utiliza siempre y cuando se hagan llamadas a la base de datos local mientras que la segunda se utiliza cuando es necesario realizar una llamada de red para llevar a cabo cualquier tipo de operación remota. Tomando el camino de la base de datos local, se encuentra el `DAO`. Esta clase es el nexo entre la base de datos y la aplicación. Se encarga de realizar las operaciones necesarias para transformar las clases que pertenecen a la propia aplicación en entidades que se pueden almacenar en base de datos. Explicando la relación *1...** a *1...**, no quiere decir que un `LocalDataSource` esté relacionado con varios `DAOs` de un mismo tipo, sino que puede haber varios de diferente tipo. Al mismo tiempo, un `DAO` puede estar presente en varios `LocalDataSource`. Finalmente, si se sigue el flujo, se encuentra la `LocalEntity`. Como ya se

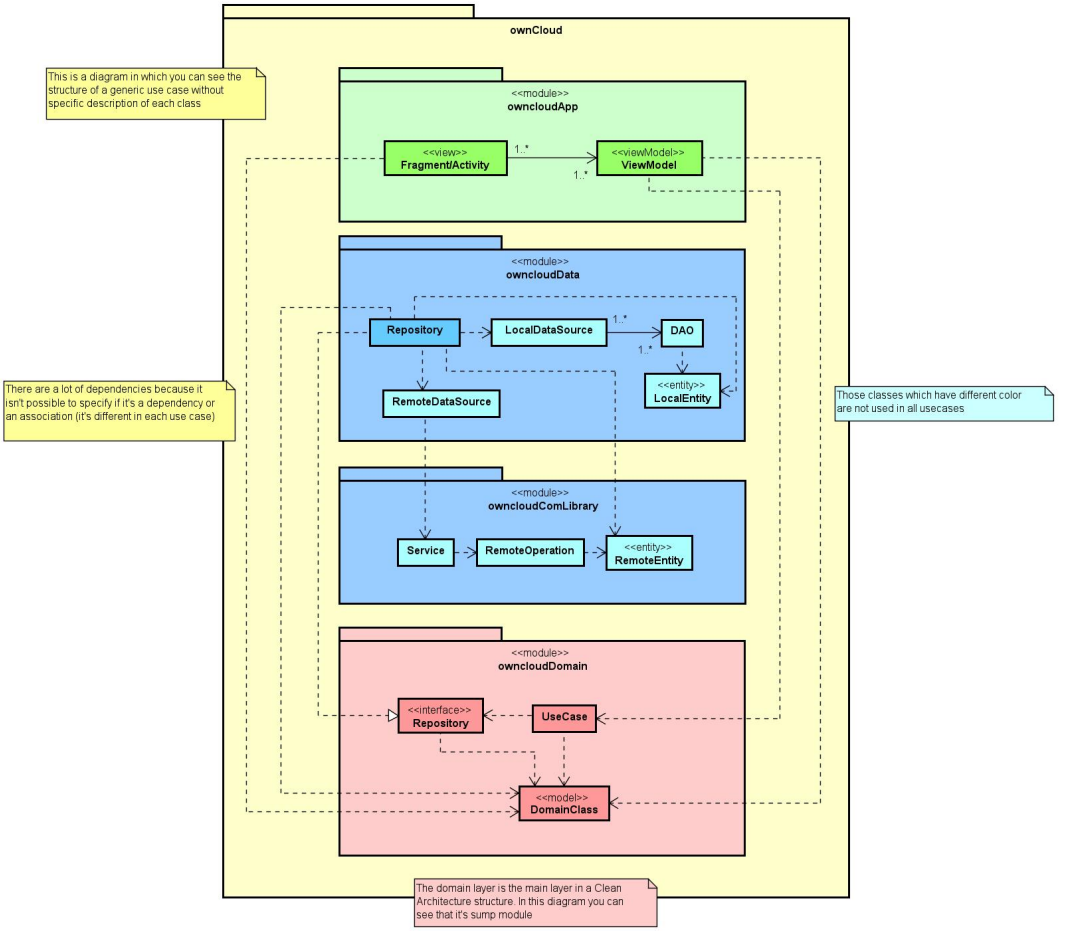

Figura 4.2: Ejemplo de caso de uso genérico

ha mencionado anteriormente, esta es la entidad que se almacenará en base de datos. Dicha clase únicamente presenta los tipos de datos básicos de Kotlin, teniendo que haber convertido anteriormente cualquier tipo definido en la aplicación en uno de estos. Si se sigue el otro flujo, se encuentra el `RemoteDataSource`. A diferencia del otro data source, este presenta una relación directa con una clase de otro módulo. Aquí es donde entra en juego el módulo *owncloudComLibrary*, encargado de realizar todas las llamadas de red. En primer lugar está el `Service`. Es una clase que reúne todas la operaciones que se necesitan realizar sobre el servidor externo. Dentro de esta clase, justo como se acaba de mencionar, están todas las operaciones de tipo `RemoteOperation`. Éstas pueden ser tanto de lectura como de escritura. Finalmente se puede apreciar como existe una asociación con la clase `RemoteEntity`. Esta clase es la encargada de mapear todos los datos que provienen del servidor remoto y que son necesarios para realizar ciertas operaciones en la aplicación.

Por último, se encuentra el módulo *owncloudDomain*, el cual siempre tiene tres partes

principales y bien definidas. En primer lugar está el `UseCase`. Esta clase es llamada directamente por el `ViewModel` localizado en la capa de presentación. Existen tantos casos de uso como funcionalidades haya en la aplicación. Esta clase tiene una dependencia directa con otras dos clases. Estas dos son el modelo o `Model` y el `Repository`. El modelo en este diagrama aparece como una única clase, pero en un ejemplo real corresponde con varias clases relacionadas entre sí, incluidas aquellas que se encuentran en módulos diferentes.

4.1.2. Caso de uso: Subida de un fichero

Tras haber hablado de la arquitectura general de la aplicación, se detallará toda la información relacionada con el caso de uso de **subida de un fichero**. Siguiendo el mismo orden que antes, se mostrará en un primer lugar el módulo *owncloudApp*. Tras ello se mostrará toda la información asociada a *owncloudData*. Seguido a esto se explicará todo lo relacionado con el módulo *owncloudComLibrary*. Finalmente se entrará en detalle en el módulo *owncloudDomain*, siendo este el centro de toda la estructura de la aplicación. Por otro lado se seguirá con la relación de colores que se había definido en el Apartado 4.1.1. Las clases pertenecientes al módulo *owncloudApp* llevan el color verde. Las clases de los módulos *owncloudData* y *owncloudComLibrary* el azul y las clases de *owncloudDomain* el color rojo. El resto de clases que tienen un color más claro corresponden a aquellas que no se encuentran reflejadas en la Figura 4.2 y por tanto no son las clases más importantes a pesar de tener una relación directa con el caso de uso y formar parte del mismo.

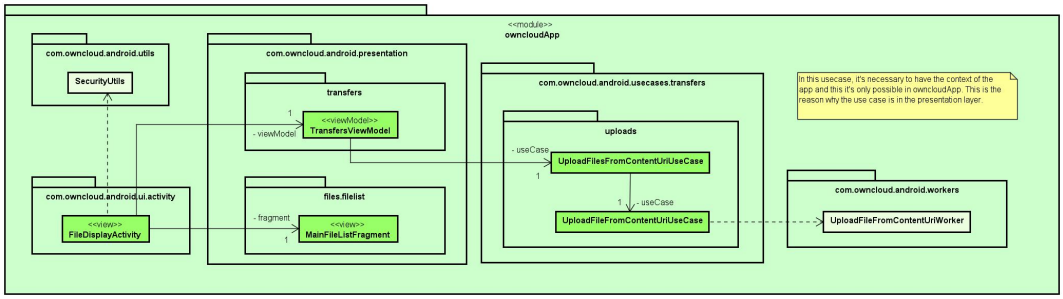

Figura 4.3: Modules&UsesStyle del módulo *owncloudApp* perteneciente al caso de uso de subida de un fichero

En la Figura 4.3 se tienen todas las clases participantes en este caso de uso que pertenecen al módulo *owncloudApp*. En primer lugar se tiene `FileDisplayActivity`, la cual será la clase en la que se implemente toda la lógica relacionada con las acciones del usuario en caso de que quiera subir cualquier tipo de archivo. Esa clase tiene como atributo un `Fragment`, el cual recibe el nombre de `MainFileListFragment`. Esta clase mostrará por pantalla toda la interfaz relacionada con la lista principal en la cual se ven todos los archivos del *Personal space*. Otra de las dependencia que se tiene con la `Activity` es el `ViewModel`. En este caso la dependencia es con `TransfersViewModel`.

En el momento en el que el usuario haya seleccionado el archivo que quiere subir desde el sistema de ficheros, se ejecutará el `UseCase` correspondiente. En este ejemplo concreto

se tienen dos casos de uso diferentes: `UploadFilesFromContentUriUseCase` y `UploadFileFromContentUriUseCase`. El hecho de que estas dos clases se encuentren en la capa de presentación y no en la capa dominio, es porque se necesita un contexto de la aplicación para realizar ciertas acciones. En Android esto sólo es posible conseguirlo en esta capa, por lo que se ha hecho una excepción respecto a la estructura que aparece reflejada en la Figura 4.2 (esto resulta algo fuera de lo normal, y que se podría tener en cuenta a la hora de rediseñar un caso de uso o incluso la aplicación). En primer lugar, se tiene un caso de uso para todos los archivos seleccionados, los cuales se guardarán en base de datos y luego se tiene el otro caso de uso por fichero individual. Este será el encargado de subir el archivo al servidor correspondiente para que más tarde se haga la sincronización y el resto de operaciones oportunas. Esas operaciones se harán a través de un `Worker`, que en este caso recibe el nombre de `UploadFileFromContentUriWorker`. Esto es simplemente para hacer todo el trabajo en segundo plano y no interferir en la secuencia principal.

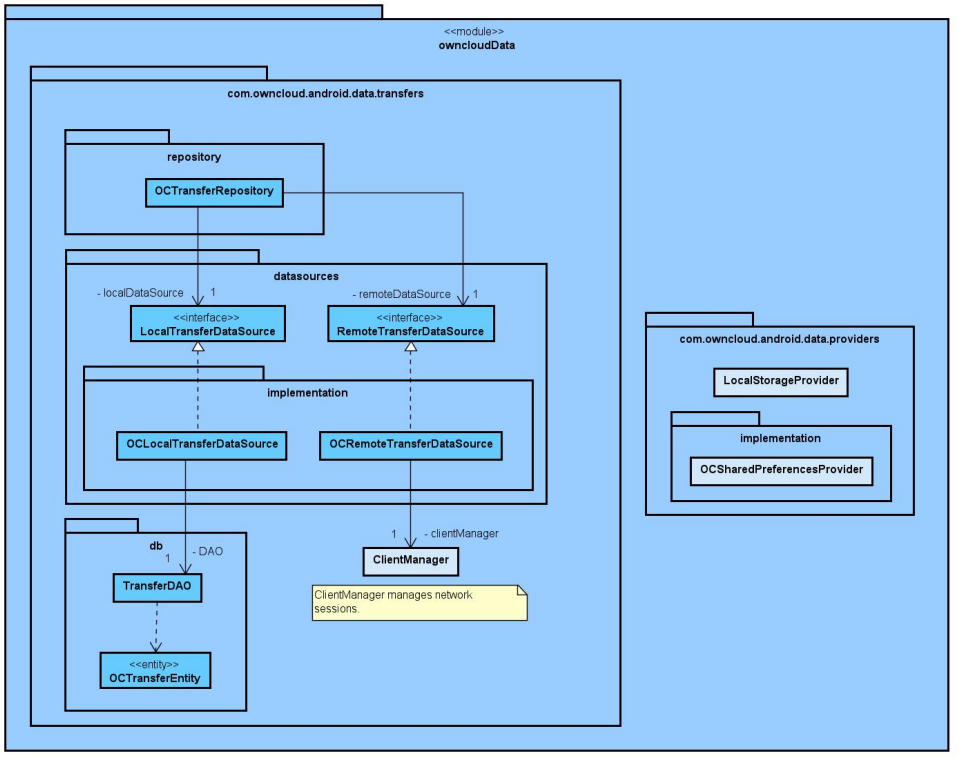

Figura 4.4: Modules&UsesStyle del módulo *owncloudData* perteneciente al caso de uso de subida de un fichero

Respecto al módulo *owncloudData*, todas las clases y sus relaciones se encuentran reflejadas en la Figura 4.4. En primer lugar se encuentra la implementación de la interfaz del repositorio. Esta clase recibe el nombre `OCTransferRepository`. Ésta tiene dos atributos o propiedades bien definidas. Por un lado está el `LocalTransferDataSource`, clase encargada de realizar cualquier tipo de operación de forma local. Por el otro lado está la interfaz que sirve como conexión para cualquier operación que se haga de forma remota, recibiendo el nombre

de `RemoteTransferDataSource`. Respecto a la parte de datos local, la implementación de `LocalTransferDataSource` se encuentra en la clase `OCLocalTransferDataSource` presente en el paquete `implementation`. Esta clase tiene una asociación con la clase `TransferDAO`, siendo una propiedad de la primera. El `DAO` servirá para poder almacenar en base de datos todos aquellos ficheros que se quieran subir, los cuales están convertidos en objetos de tipo `OCTransferEntity`.

Siguiendo el camino de las operaciones en remoto, se tiene que la implementación de `RemoteTransferDataSource` se encuentran en el mismo paquete que la implementación local. En este caso, la clase que lo implementa recibe el nombre de `OCRemoteTransferDataSource`, siguiendo el patrón del prefijo OC. Para acabar con este módulo, existe una clase denominada `ClientManager` la cual se encarga de gestionar todas las sesiones para conectarse a la red y que se encuentran directamente asociada con la implementación del data source remoto, siendo un atributo de la misma. De forma aislada se pueden ver las clases `LocalStorageProvider` y `OCSharedPreferencesProvider`. Ambas clases se utilizan en el caso de uso de forma complementaria para conseguir información que será utilizada en la lógica del caso de uso.

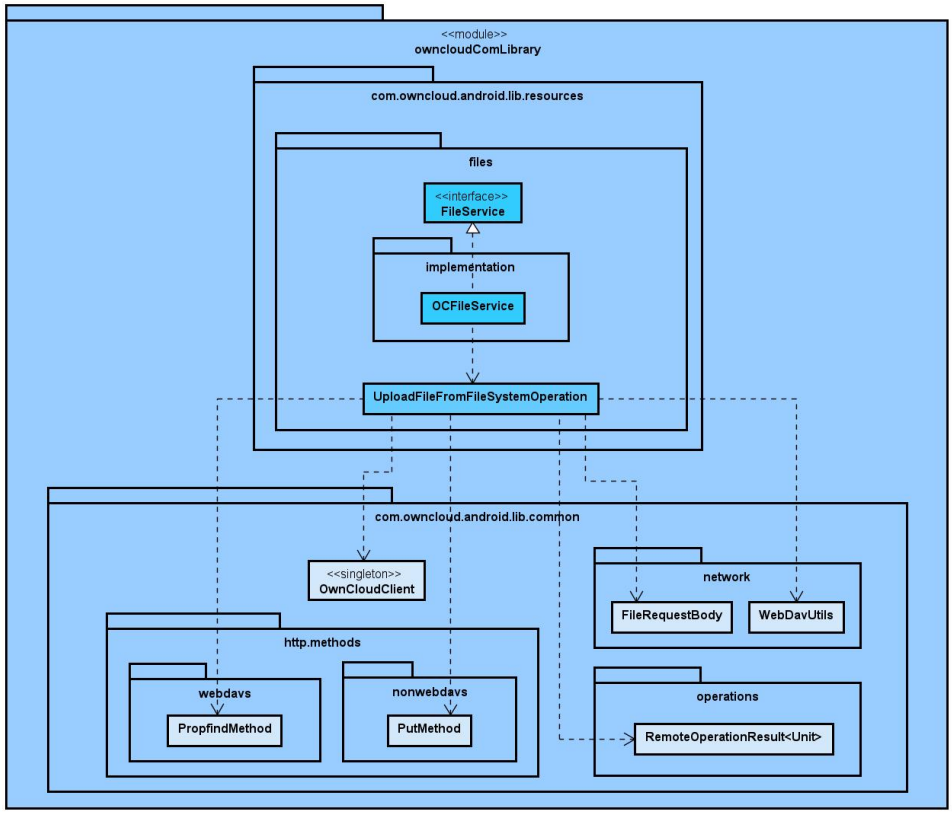

Figura 4.5: Modules&UsesStyle del módulo *owncloudComLibrary* perteneciente al caso de uso de subida de un fichero

Siguiendo el orden que se había establecido previamente, todas las clases y relaciones pertenecientes al módulo *owncloudComLibrary* aparecen en la Figura 4.5. En primer lugar se tiene tres clases bien diferenciadas que participan en este caso de uso, todas ellas dentro del paquete `files`. La primera de ellas es `FileService`. Esta clase es simplemente una interfaz, la cual está implementada por `OCFileService`. Ésta última es la encargada de llevar a cabo todas las operaciones remotas dependiendo del caso de uso. Dentro de estas operaciones, se tiene la clase `UploadFileFromFileSystemOperation` cuya lógica consiste en subir los respectivos ficheros al servidor. En la parte inferior se pueden ver varias clases dentro del paquete `common`. Todas ellas participan en el caso de uso pues están relacionadas con la operación, pero no forman parte de la estructura general que se había mostrado en la Figura 4.2

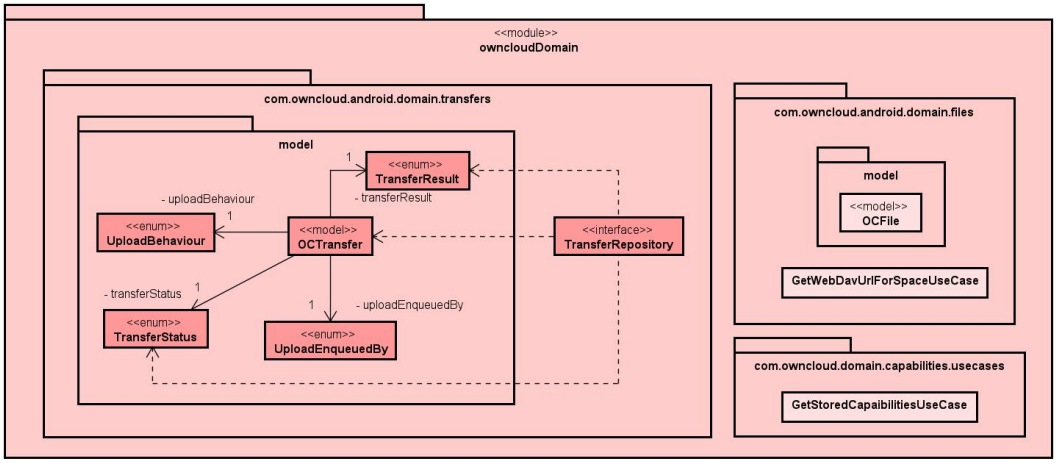

Figura 4.6: Modules&UsesStyle del módulo *owncloudDomain* perteneciente al caso de uso de subida de un fichero

Para el módulo *owncloudDomain* se tiene una gran diferencia respecto al otro caso de uso. Como se puede apreciar en la Figura 4.6 no se tienen los casos de uso en este módulo. Destacando entre todas las clases, está `TransferRepository` la cual es la interfaz a partir de la cual se harán todas las operaciones respectivas. A su vez se tiene las clases de `OCTransfer`, `TransferResult`, `UploadEnqueuedBy`, `TransferStatus` y `UploadBehaviour`. La relación que existe entre todas ellas es que la clase `OCTransfer` tiene un atributo de cada una de las otras clases además de otros atributos de tipos primitivos, de ahí la multiplicidad de 1 solamente en una dirección. Por otro lado también existe dependencia entre el repositorio y otras clases que no sean la de la propia subida, debido a que existen ciertos métodos que dependen de esas clases para que se ejecuten de forma correcta.

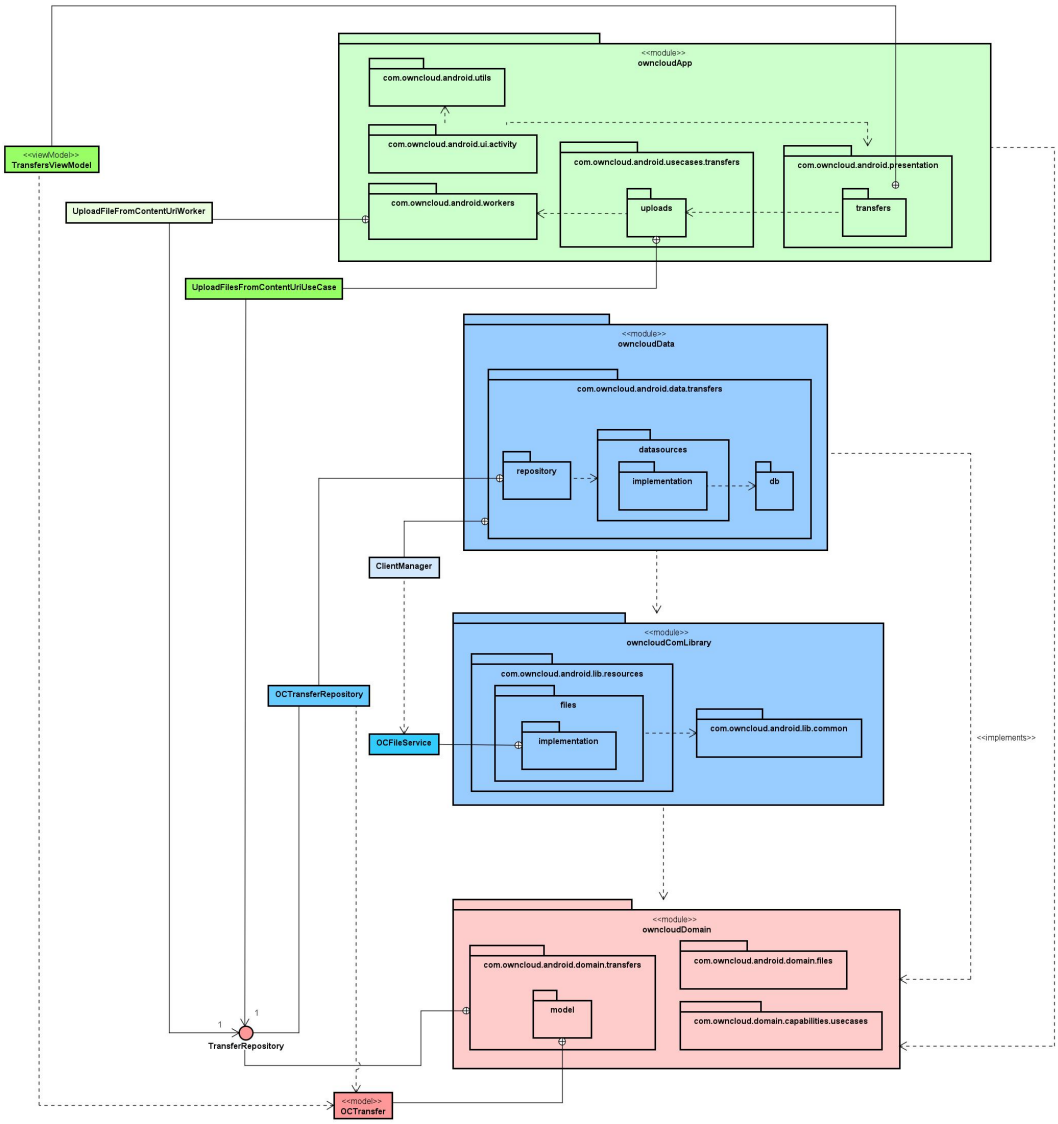

Figura 4.7: Modules&UsesStyle del caso de uso de subida de un fichero

En la Figura 4.7 se pueden apreciar todos los módulos junto con las clases del caso de uso completo. Se ha decidido mostrar únicamente las clases que están relacionadas entre sí pero pertenecen a diferentes módulos, ya que se quiere reflejar la alta cohesión y el bajo acoplamiento que existe. Como para este caso de uso los `UseCase` estaban el módulo de presentación no existe un `ViewModel` relacionado directamente con la capa de dominio. Es por ello que se tiene una relación con el repositorio desde el `Worker`. Por otro lado también se puede apreciar como existe una dependencia entre el módulo *owncloudData* y el módulo *owncloudComLibrary*. Esta dependencia viene dada por la relación que hay desde la clase

`ClientManager` hasta `FileService`, debido a que la primera clase es aquella que se encarga de gestionar todas las sesiones para poder realizar llamadas de red tal y como se menciona en la Figura 4.4.

4.1.3. Caso de uso: Consulta del estado de las subidas

Después de haber explicado el caso de uso correspondiente a la subida de un fichero, se definirá el caso de uso que no utiliza el módulo *owncloudComLibrary*: **consulta del estado de las subidas**.

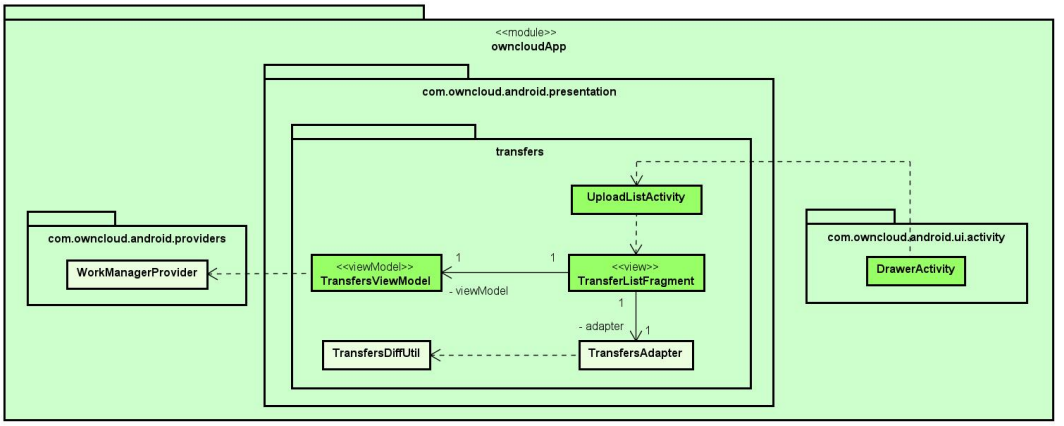

Figura 4.8: Modules&UsesStyle del módulo *owncloudApp* para el caso de uso de consulta del estado de las subidas

Comenzando con el módulo *owncloudApp* (reflejado en la Figura 4.8), la clase a partir de la cual se inicia todo recibe el nombre `DrawerActivity`. En esta clase se define la barra inferior (`bottom bar`) con las múltiples opciones que existen la aplicación. Una vez que el usuario ha seleccionado la opción de subidas se crea la clase `UploadListActivity`. De ahí esa dependencia que existe entre ambas clases mencionadas. En esta última se define el comportamiento principal de la pantalla sobre la que se basa el caso de uso, creando al mismo tiempo la interfaz que se mostrará el usuario. Esta vista corresponde a la clase `TransferListFragment`, la cual tiene como propiedad un `ViewModel` (`TransfersViewModel`). Esta clase se encarga de implementar la lógica cuando el usuario interactúa con la pantalla a la vez que llama al caso de uso dependiendo de dicha acción. Por otro lado se encuentra la clase `TransferAdapter`, la cual sirve para mostrar como lista todas aquellas subidas que se encuentran almacenadas de forma local y pertenecen al usuario que está utilizando la aplicación. Además, existe una dependencia con `TransfersDiffUtil`, la cual sirve de utilidad y realiza las operaciones necesarias para mostrar un cambio de una lista a otra.

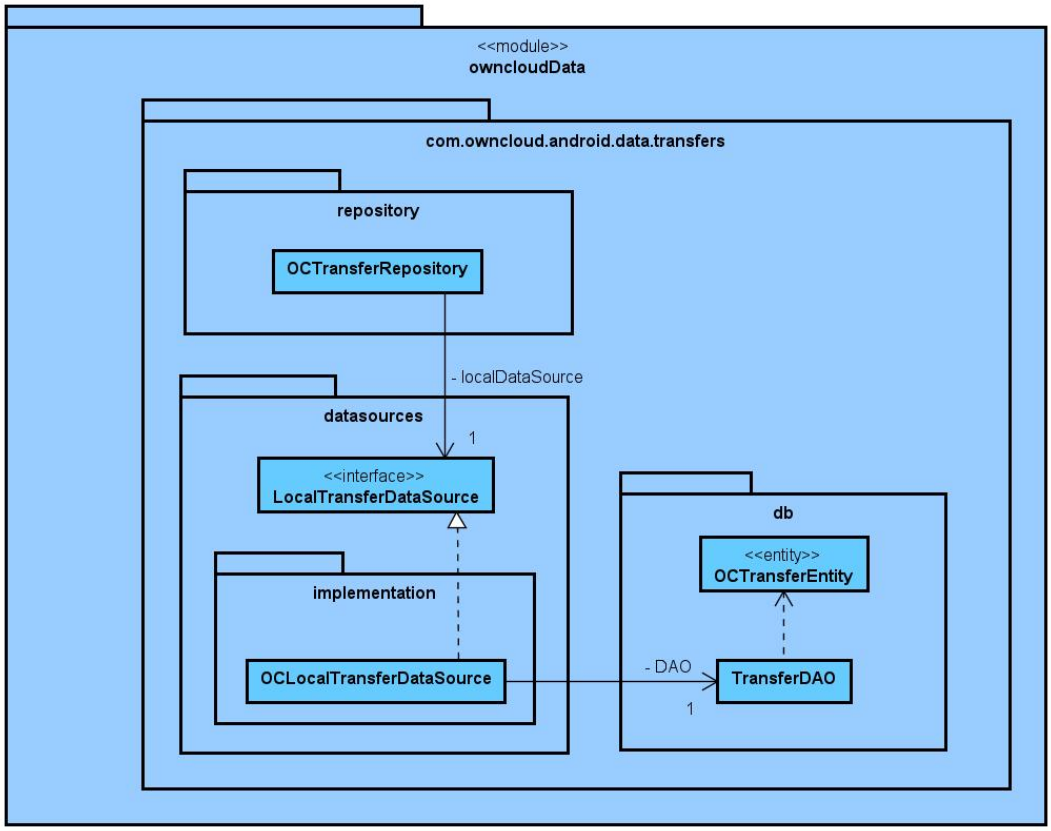

Figura 4.9: Modules&UsesStyle del módulo*owncloudData* para el caso de uso de consulta del estado de las subidas

Respecto al módulo *owncloudData* (el cual se encuentra reflejado en la Figura 4.9), se tiene la clase `OCTransferRepository` la cual es la implementación de la interfaz del repositorio presente en la capa de dominio. Esta clase tiene como propiedad una interfaz, que recibe el nombre de `LocalTransferDataSource`. La implementación de dicha interfaz se encuentra en otro paquete anexo dentro de todos los datasources, cuyo nombre es `OCLocalTransferDataSource`. Como se puede apreciar todas las implementaciones de las interfaces reciben el prefijo OC (*ownCloud*). Esa clase de implementación tiene como propiedad otra clase de tipo `TransferDAO`. Como la finalidad de este caso de uso es comprobar el estado de todas las subidas que ha hecho el usuario, entonces será necesario sacar esa información de la base de datos local. Esto se consigue a través de esa clase `DAO` y junto con la clase a la que se tiene dependencia, denominada `OCTransferEntity`. Esta clase tiene una particularidad y es que todos los atributos son tipos primitivos, debido a que se han almacenado en base de datos. Para poder trabajar correctamente una vez que se hayan conseguido todos los datos será necesario transformarlo en una clase del modelo de dominio.

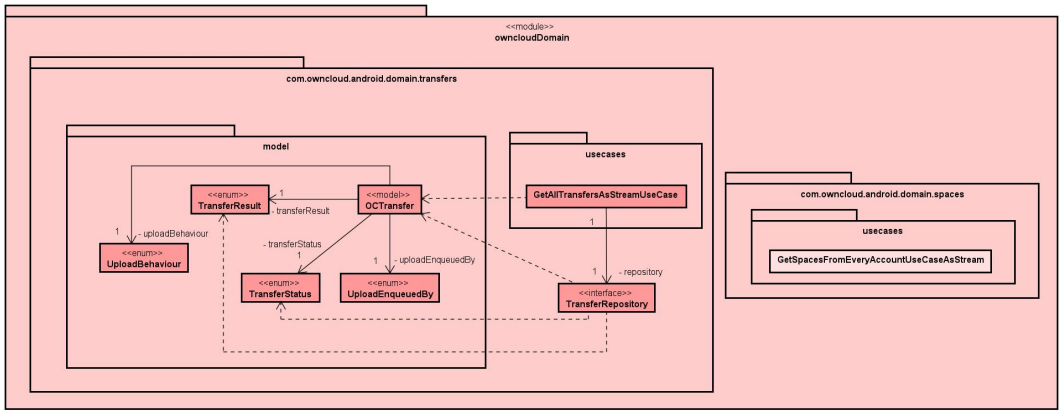

Figura 4.10: Modules&UsesStyle del módulo *owncloudDomain* para el caso de uso de consulta del estado de las subidas

En la Figura 4.10 se puede ver todas las clases que intervienen en este caso de uso y que pertenecen a la capa de dominio. En primer lugar se tiene una clase que recibe el nombre de `GetAllTransfersAsStreamUseCase`, la cual se ejecutará con el fin de poder obtener todas las subidas que tenga el usuario en su cuenta. Esta clase se encuentra en un paquete aparte denominado `usecases`, en el cual se encuentran el resto de casos de usos que se ejecuten directamente desde el `ViewModel` y tengan relación con las subidas. Tal y como se ha mencionado en otros apartados, este caso de uso tiene una dependencia con el `TransferRepository`. Esta clase representa una interfaz, cuya implementación se encuentra el módulo *owncloudData*. También se puede ver como existe otro `UseCase` de forma aislada en otro paquete. Esta clase recibe el nombre de `GetSpacesFromEveryAccountUseCaseAsStream`. Su funcionalidad es conseguir todos los `spaces` que estén asociados a la cuenta del usuario para más tarde relacionar las subidas con esos `spaces`.

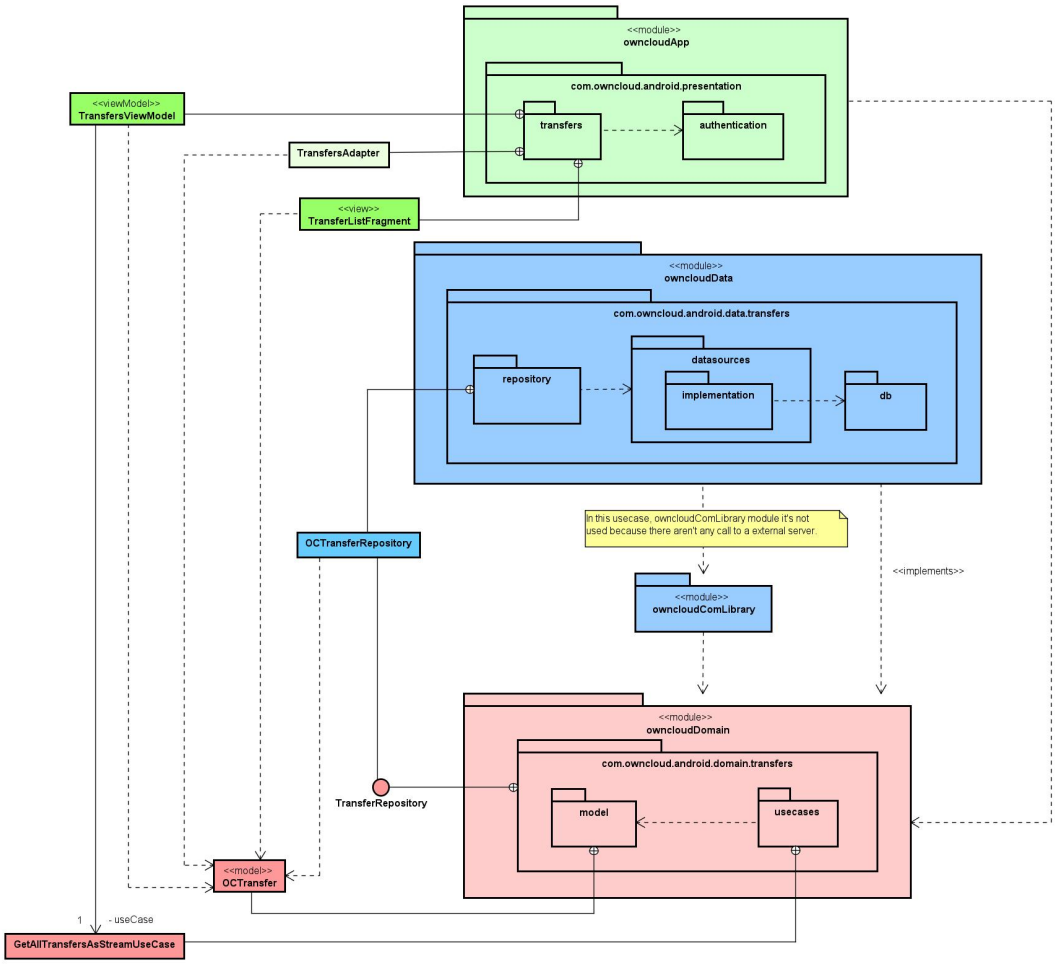

Figura 4.11: Modules&UsesStyle del caso de uso de consulta del estado de las subidas

En la Figura 4.11 se observan todos los módulos que intervienen en este caso de uso además de todas las clases que se relacionan con clases de otro módulo. El hecho de que se muestre este diagrama es que se pueda apreciar la alta cohesión y el bajo acoplamiento que existe en la app Android de *ownCloud*. En primer lugar se tienen las tres clases pertenecientes al módulo *owncloudApp*, las cuales son `TransfersAdapter`, `TransfersViewModel` y `TransfersListFragment`. Todas ellas tiene una dependencia directa con el modelo, en concreto con la clase `OCTransfer`. Además de esa dependencia el `TransfersViewModel` tiene una dependencia con el caso de uso `GetAllTransfersAsStreamUseCase`. Siguiendo la Clean Architecture, no pueden salir dependencias desde la capa de dominio hacia otras capas, de ahí que se tenga `OCTransferRepository` relacionado con `TransferRepository` pues es su implementación. Aparte de estas clases no existe ninguna otra relación entre todos los módulos. Como en la propia nota del diagrama se indica, el módulo *owncloudComLibrary* no participa en este caso de uso pues no existe ninguna llamada de red y todo se hace de forma local.

Capítulo 5

Ingeniería inversa de la arquitectura de ownCloud: vista dinámica

5.1. Clean Architecture en ownCloud: vista dinámica

A lo largo de este capítulo se tratarán los dos casos de uso sobre los que se ha trabajado, mostrando la interacción de todos los objetos que participan en cada uno de ellos. Para llevar el mismo orden que en el anterior capítulo, se comenzará con la descripción del caso de uso: **subida de un fichero** y tras ello se detallará el caso de uso: **consulta del estado de las subidas**. Se seguirá la misma convención que ya se venía usando en los diagramas de módulos en relación al uso de colores. El color verde se aplicará a todas aquellas clases que pertenezca al modelo *owncloudApp*. El color azul para las clases que pertenecen a los módulos *owncloudData* y *owncloudComLibrary*. A su vez, el color rojo será utilizado en todas aquellas clases del módulo *owncloudDomain*. Como novedad respecto a los diagramas de módulos, es que en los diagramas de secuencia aparecen ciertas clases de color gris. Éstas pertenecen al contexto de Android y no son desarrolladas por *ownCloud*.

5.1.1. Caso de uso: Subida de un fichero

Para que se inicie el caso de uso, el usuario tiene que interactuar con el botón flotante que se encuentra ubicado en la parte inferior derecha de la pantalla. Este botón se ha creado en la clase `MainFileListFragment`. En el momento en el que se pulsa esa opción se abrirá un cuadro de texto en esa misma parte de la pantalla, cuya secuencia corresponde al diagrama de la Figura 5.1.

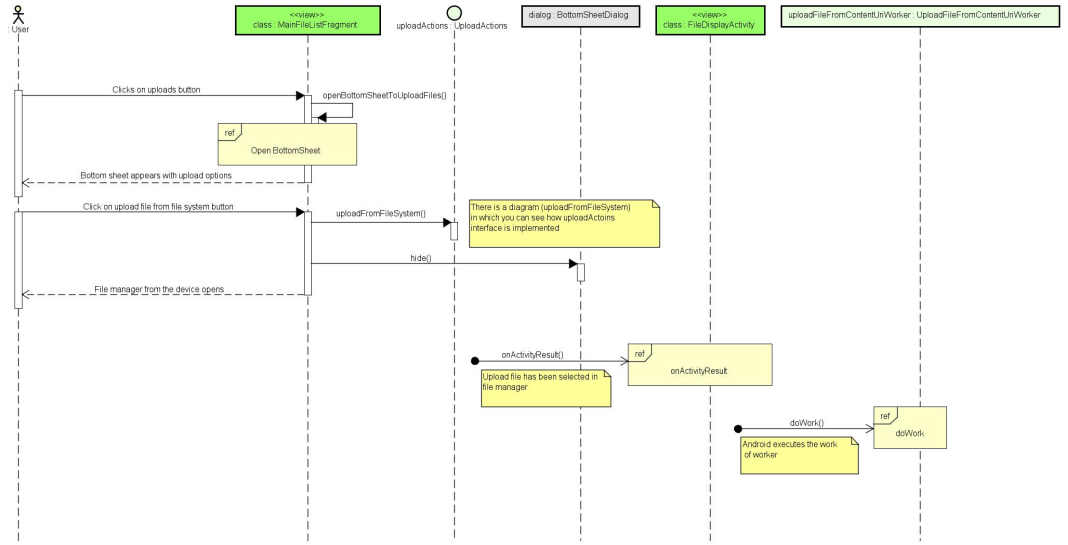

Figura 5.1: Diagrama de secuencia principal correspondiente al caso de uso de subida de un fichero.

La función `openBottomSheetToUploadFiles`, la cual se encuentra desarrollada en la Figura 5.2 tiene un funcionamiento bastante sencillo. En primer lugar lo que se hace es un *inflate* sobre el `LayoutInflater`. Una vez que se ha hecho esto, se creará el diálogo que se va a mostrar por pantalla. Cuando ya se ha creado, se mostrarán las dos vistas posibles que tiene el diálogo: subida de ficheros desde el almacenamiento del sistema o subida de ficheros a través de la cámara. Tras haber construido el diálogo con las dos posibles opciones, junto con el texto correspondiente, se mostrará por pantalla.

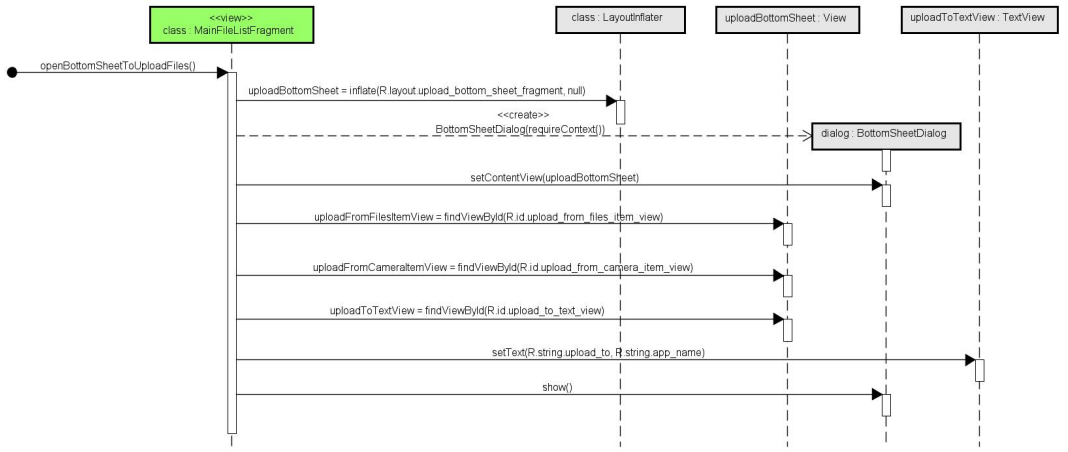

Figura 5.2: Diagrama de secuencia correspondiente a la función `openBottomSheetToUploadFiles` de la Figura 5.1

A partir de este momento, el usuario seleccionará la opción de subida de ficheros desde el sistema de archivos. Tal y como se muestra en la Figura 5.1, cuando el usuario lleva a cabo la acción correspondiente se ejecutará un nuevo método llamado `uploadFromFileSystem` además de ocultar el diálogo que había aparecido anteriormente. Esta llamada forma parte de una interfaz alojada en la clase `MainFileListFragment` cuya implementación se encuentra en la clase `FileDisplayActivity`. Por este motivo se ha mostrado toda la información asociada a este método en la Figura 5.3.

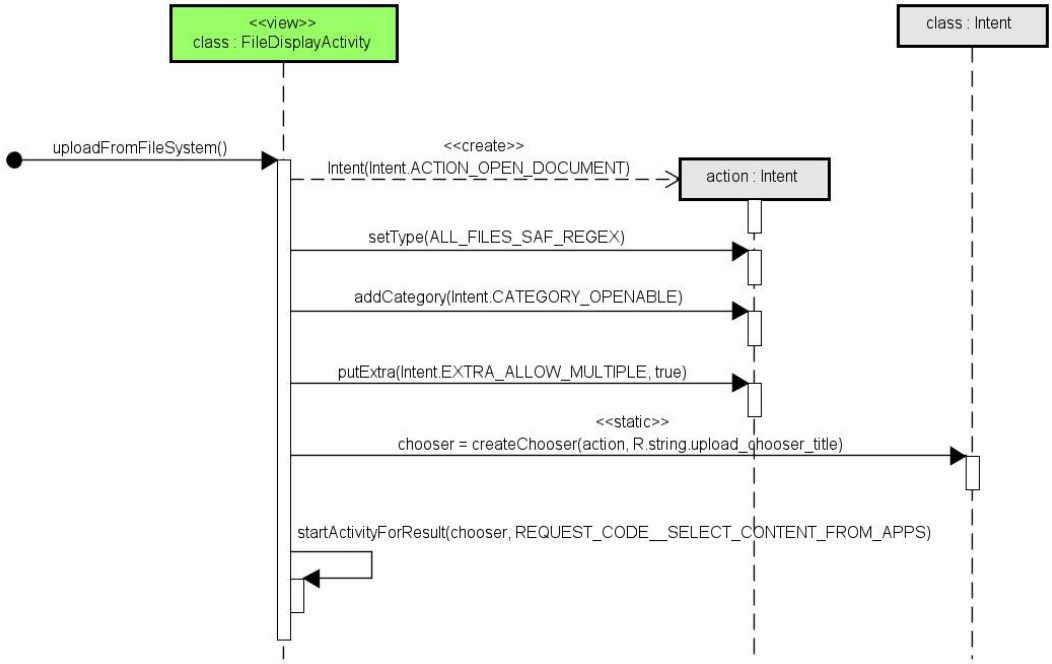

Figura 5.3: Diagrama de secuencia correspondiente a la función `uploadFromFileSystem` de la Figura 5.1

Una vez que se realiza la llamada al método correspondiente, se crea un `Intent` implícito, pues no se determina a que componente se va a llamar. Tras haber creado el `Intent`, se establecen una serie de parámetros sobre ese componente y a partir de ahí se lanzará su ejecución haciendo que se abra el sistema gestor de ficheros del dispositivo. En este caso la función asociada a esa acción es `startActivityForResult`. El funcionamiento interno es que una función *callback* recibirá un aviso una vez que ya se haya terminado de elegir los ficheros que se quieren subir. A partir de este momento, se procesará toda la información que venga en el propio *callback*. Esta llamada se puede apreciar en el diagrama de la Figura 5.1, en el cual existe una llamada asíncrona sobre `onActivityResult` perteneciente a la clase `FileDisplayActivity`.

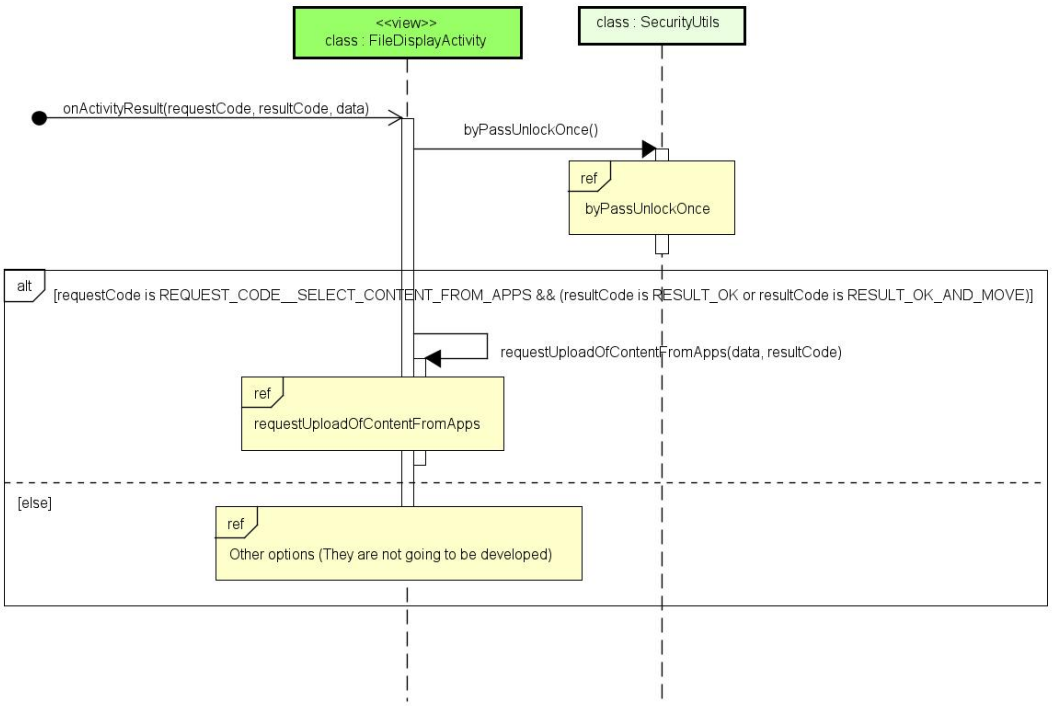

Figura 5.4: Diagrama de secuencia correspondiente a la función `onActivityResult` de la Figura 5.1

Tal y como se puede apreciar en la Figura 5.4, la llamada *callback* a esa función llega con tres parámetros diferentes. El primero de ellos es el `requestCode`, el cual indica el tipo de llamada que se está haciendo. Por otra lado se tiene el `resultCode`, el cual indica si las operaciones que se han hecho previamente son correctas y no ha habido ningún fallo. Por último está la variable `data`, en la cual viene toda la información asociada a los ficheros que se han seleccionado en el sistema gestor de archivos. Una vez que se recibe esa llamada, se llama a la función `byPassUnlockOnce`, cuyo funcionamiento se encuentra reflejado en la Figura 5.5.

Un usuario tiene la posibilidad de establecer una contraseña cada vez que accede a *ownCloud*, teniendo una mayor seguridad respecto a sus datos. El método de `byPassUnlockOnce`, lo que hace es desactivar ese paso si el tiempo en el que ha estado buscando los archivos que quiere subir ha excedido el tiempo límite para que vuelva a saltar la pantalla de introducir la contraseña. Después de haber ejecutado este método, se tiene una estructura condicional bastante larga y que para este caso de uso sólo aplica la primera condición[1].

[1]El resto de los casos de esa estructura condicional no se van a desarrollar por motivos de extensión, además de no estar directamente relacionados con el caso de uso expuesto.

Figura 5.5: Diagrama de secuencia correspondiente a la función `byPassUnlockOnce` de la Figura 5.4

En caso de que el `requestCode` tenga un valor relacionado con la selección de contenido de cualquier otra aplicación, y el `resultCode` sea correcto, se ejecutará la función `requestUploadOfContentFromApps`, cuya implementación se ve reflejada en la Figura 5.6. En primer lugar, se crear un ArrayList de `URIs` totalmente vacío. Tras ello se recorrerá toda la lista de elementos que vienen en el data del `Intent` y se irán añadiendo a ese ArrayList que se ha creado anteriormente. Una vez que se ha formado correctamente toda la lista con las `URIs` de los elementos que se van a subir a la aplicación, se obtiene el directorio sobre el que se va a realizar dicha subida.

Cuando el usuario no esté en una determinada ruta dentro de su sistema de ficheros en *ownCloud*, se subirá a la dirección `root` que está por defecto (/). Para cada uno de esos ficheros se necesitará obtener todos los permisos correspondientes. Esta funcionalidad se abstrae en el *ref* denominado `getPermission`, cuyo desarrollo se documenta en la Figura 5.7.

Para cada uno de los elementos que pertenecen a la lista de `URIs` antes mencionada, se comprueba que se tenga permiso de escritura de todos los ficheros que se van a tratar. En caso de que esto no sea así, saltará una excepción de tipo `RemoteException`. La zona del código en la que se captura esta excepción está marcada en el fragmento combinado *alt*, el cual simplemente llamará a la clase `Timber` para generar un mensaje de error en forma de log. Una vez que se han conseguido todos los permisos de los archivos que se van a subir, se llama a la función `uploadFilesFromContentUri`, la cual se encuentra alojada en el `TransfersViewModel`. El desarrollo de esta función se documenta en la Figura 5.8.

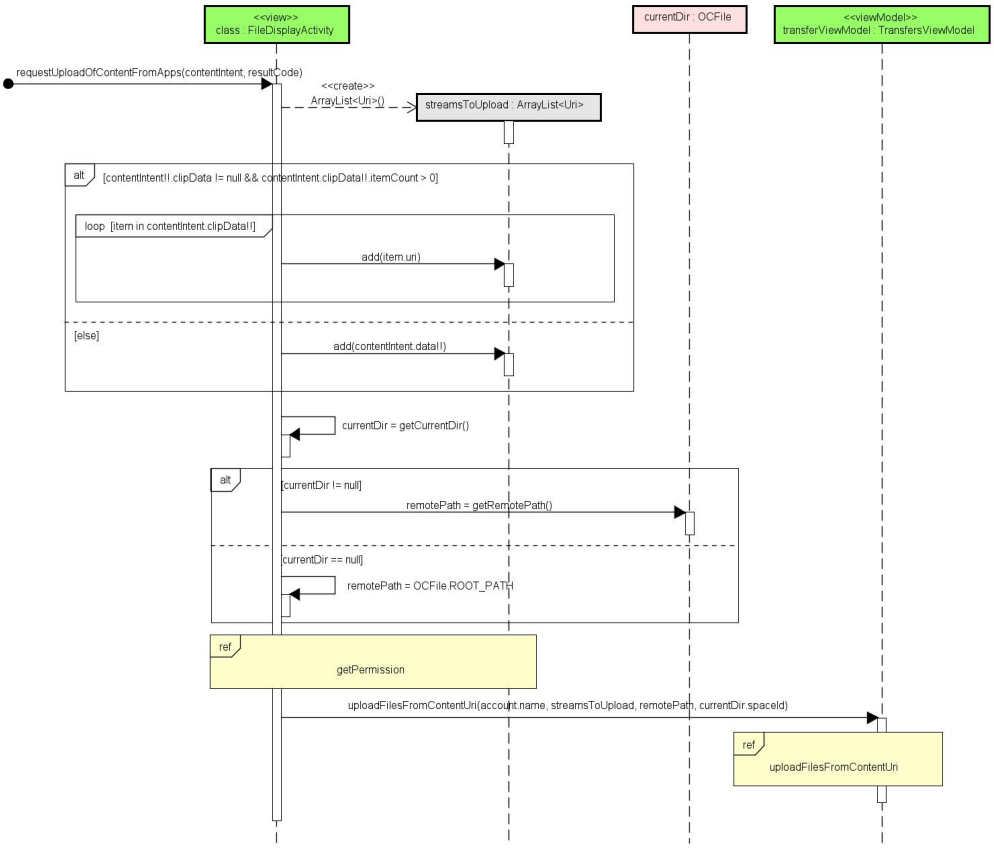

Figura 5.6: Diagrama de secuencia correspondiente a la función `requestUploadOfContentFromApps` de la Figura 5.4.

Después de recibir la llamada por parte del `FileDisplayActivity`, el `ViewModel` realizará una llamada a la clase de Android `CoroutineScope` de modo que ejecutará el caso de uso en otro hilo totalmente distinto al principal de forma asíncrona. Una particularidad que tiene Kotlin y que se ha utilizado en este proyecto son los *operator*. Un *operator* permite agregar funcionalidades nuevas a clases existentes sin tener que heredar de ellas o modificar su código fuente. Para este caso concreto se ha creado un *operator* relacionada con el método `invoke` de un caso de uso. Únicamente será necesario escribir el nombre del caso de uso con los parámetros necesarios sin la notación punto para poder ejecutarlo. Después de que se haya ejecutado de forma asíncrona, se recorrerá nuevamente la lista de todas las `URIs` creando un objeto de tipo `DocumentFile` para cada uno de los elementos. Cada elemento se guardará en base de datos local además de subirse al servidor remoto. En caso de que la creación del archivo sea fallida saltará un error en consola. Para llevar un orden de todo el caso de uso, primero se verá la funcionalidad del guardado del fichero en base de datos local la cual recibe el nombre de `storeInUploadsDatabase`. Luego se detallará toda la implementación de la función `enqueueSingleUpload`, la cual corresponde a la subida de un único fichero hacia el servidor.

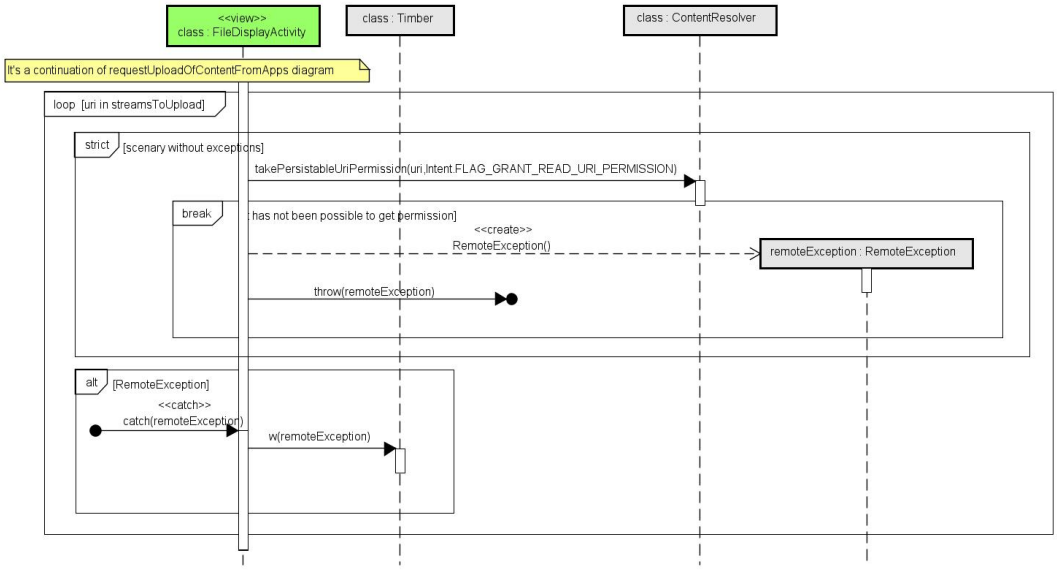

Figura 5.7: Diagrama de secuencia correspondiente al ref `getPermission` de la Figura 5.6

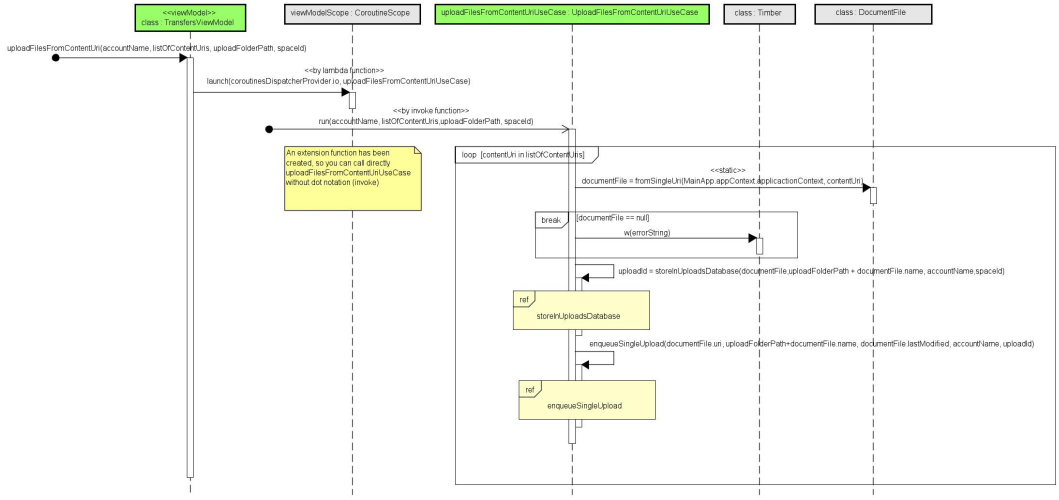

Figura 5.8: Diagrama de secuencia correspondiente a la función `uploadFilesFromContentUri` de la Figura 5.6

En la Figura 5.9 se puede apreciar la relación que teníamos entre la capa de presentación y la capa de dominio. El caso de uso `UploadFilesFromContentUriUseCase`, a partir de los parámetros que se le pasan, crea un objeto de tipo `OCTransfer`. Una vez que se ha creado ese objeto, llamará al repositorio `TransferRepository`, cuya implementación se encuentra en otra capa totalmente distinta. Al igual que antes, se utiliza un diagrama diferente para mostrar la implementación de esa interfaz, junto con el resto de llamadas que se hagan en

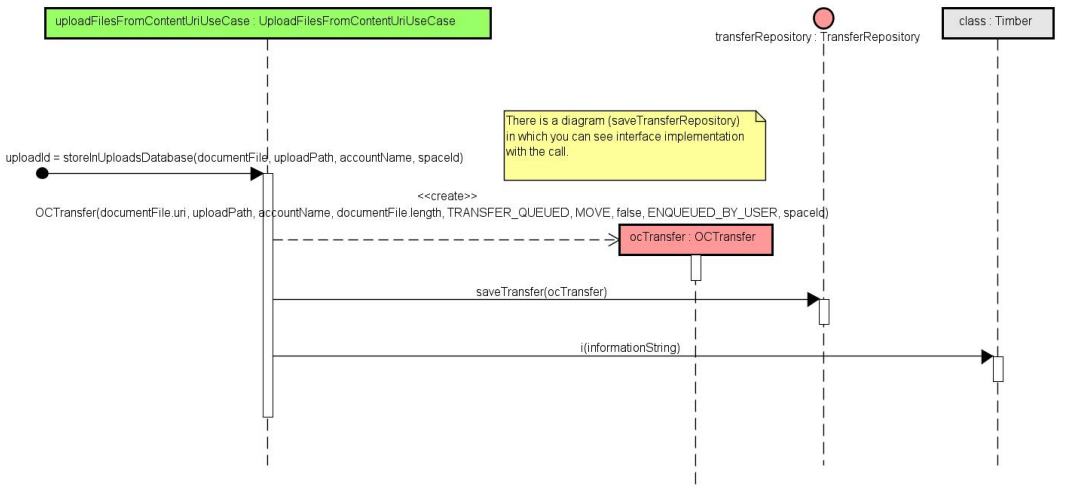

Figura 5.9: Diagrama de secuencia correspondiente a la función `storeInUploadsDatabase` de la Figura 5.8

el flujo de actividad. Además de llamar al repositorio, se muestra un log con la información correspondiente a la creación del objeto y la llamada a la interfaz.

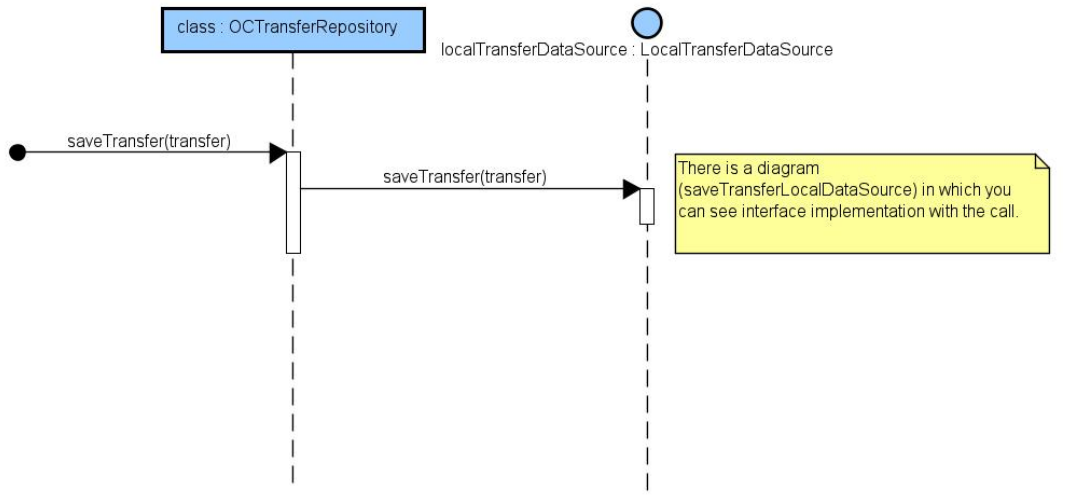

Figura 5.10: Diagrama de secuencia correspondiente a la implementación de la interfaz `TransferRepository` de la Figura 5.9

La implementación del repositorio simplemente consisten en realizar una llamada a la clase `LocalTransferDataSource`, la cual tiene una relación directa con el `DAO`. En la Figura 5.10 se puede ver esa llamada. Al igual que antes, al tener una interfaz existe otro diagrama en el que ya se muestra todo su implementación. Todo ello se encuentra reflejado en la Figura 5.11.

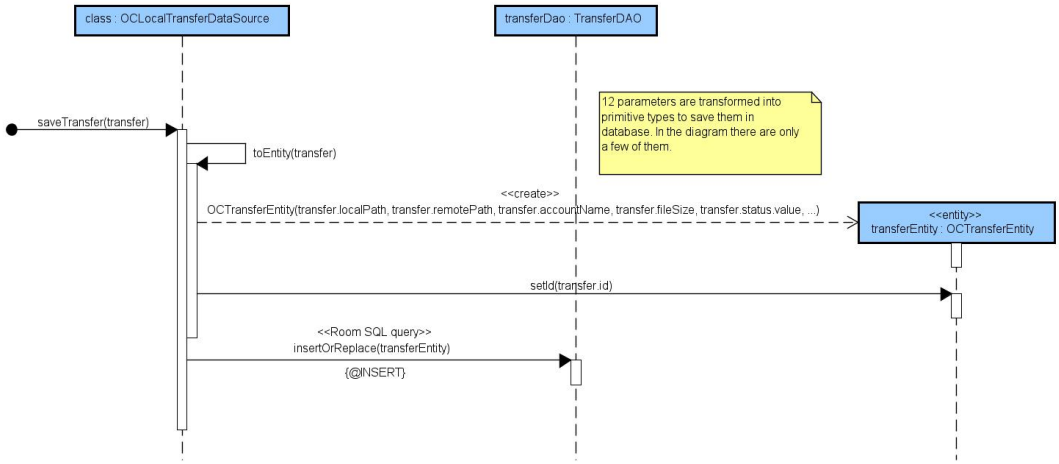

Figura 5.11: Diagrama de secuencia correspondiente a la función `saveTransfer` de la Figura 5.10

Desde la implementación del `LocalTransferDataSource`, se recibe el objeto `OCTransfer` que se había creado en pasos anteriores, exactamente en la Figura 5.9. Para guardar este objeto en base de datos local será necesario transformar todas sus propiedades en tipos primitivos, pues son los únicos tipos admitidos. Esa transformación corresponde a la función `toEntity`. Una vez que se ha creado el objeto de tipo `OCTransferEntity` se llamará a la clase `TransferDAO`, la cual se encarga de realizar un `insert` con ese objeto y guardarlo directamente en la base de datos local a través de un `insert`, operación que añade una nueva entrada, pues es una base de datos relacional. En caso de que ya exista la entrada, reemplazará la que había previamente sobrescribiéndola, de ahí que la función sea `insertOrReplace`. Para conseguir todo esto se ha utilizado Room, una biblioteca nativa de Android que se utiliza en *ownCloud* para la construcción de la base de datos local. A partir de este momento ya se tiene guardado en base de datos la información asociada al archivo que se va a subir. Tras ello, y siguiendo el flujo que se muestra en la Figura 5.8, ahora hay que guardarlo en el servidor remoto a través de una llamada de red. Sin embargo, antes de esta llamada hay una serie de pasos intermedios como, por ejemplo, la implementación de la función `enqueueSingleUpload`.

Desde el caso de uso `uploadFilesFromContentUriUseCase` se llama, con el *operator* que se había creado, al otro caso de uso que está relacionado directamente, el cual recibe el nombre de `uploadFileFromContentUriUseCase`. Desde este caso de uso se construirá un `Worker` (clase encargada de realizar operaciones en segundo plano), el cual será el encargado de realizar todas las acciones relacionadas con la llamada de red cuando el sistema operativo lo indique. Antes de crear el `Worker` se definen una serie de variables que son útiles para su construcción. Todo ello se ha abstraído en un *ref* con el nombre de *Creation of variables of the worker*. Esto se hace con el fin de evitar alargar demasiado el diagrama y, como consecuencia de esto, tener dificultad para apreciar todo lo importante referente al diagrama.

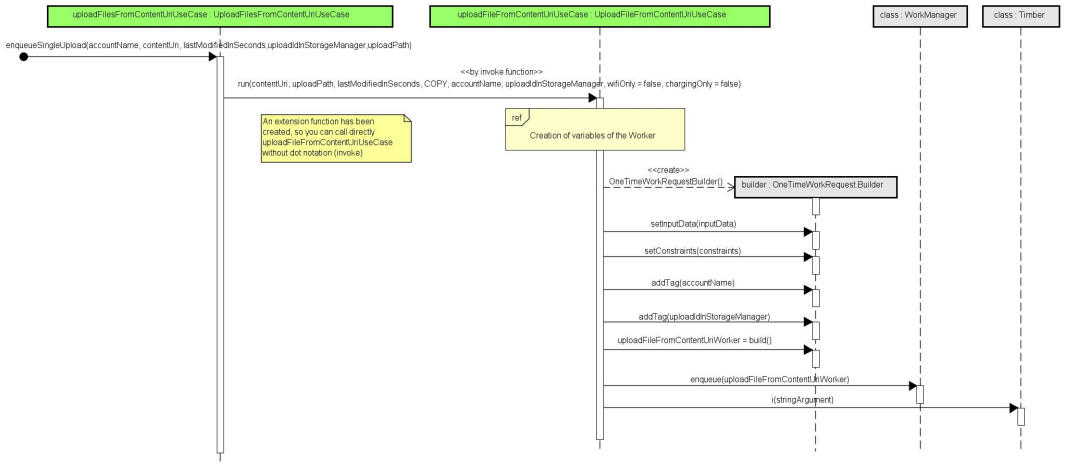

Figura 5.12: Diagrama de secuencia correspondiente a la función `enqueueSingleUpload` de la Figura 5.8

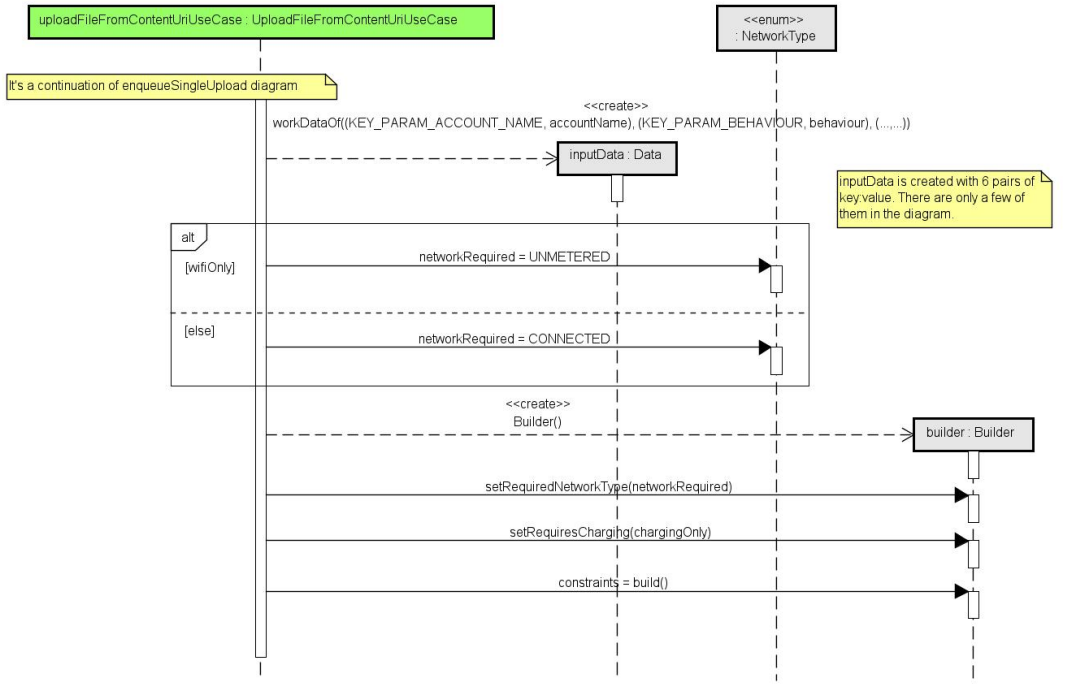

Figura 5.13: Diagrama de secuencia correspondiente al ref `Creation of variable of the worker` de la Figura 5.12

En la Figura 5.13 se aprecia la creación de todas esas variables, que se mencionaban anteriormente respecto a la construcción del `Worker`. En primer lugar, se creará un objeto de

tipo `Data`, el cual tiene seis pares de atributos de la forma *<clave:valor>*. En el diagrama no están reflejados todos ellos pero esto es debido a que el diagrama se hacía demasiado extenso y, por tanto, ilegible debido a su tamaño. El siguiente paso es comprobar si está habilitada la opción del Wi-Fi. Dependiendo del caso se elegirá un valor u otro. Una vez hecho esto, se inicializarán todos los valores del `Builder` para que más tarde pueda construir el `Worker` correctamente. Siguiendo la secuencia de la Figura 5.12, se inicializan todas esas variables que se habían creado en el ref y, finalmente, se hace un `build` construyendo el `Worker` con la información correspondiente y encolándolo en el `WorkManager` para que el sistema operativo lo ejecute siguiendo sus propios criterios internos. Volviendo nuevamente al diagrama de secuencia principal, reflejado en la Figura 5.1, se puede apreciar como existe una llamada asíncrona a la función `doWork` correspondiente a la clase `UploadFilesFromContentUriWorker`. El desarrollo de dicho método se encuentra en la Figura 5.14.

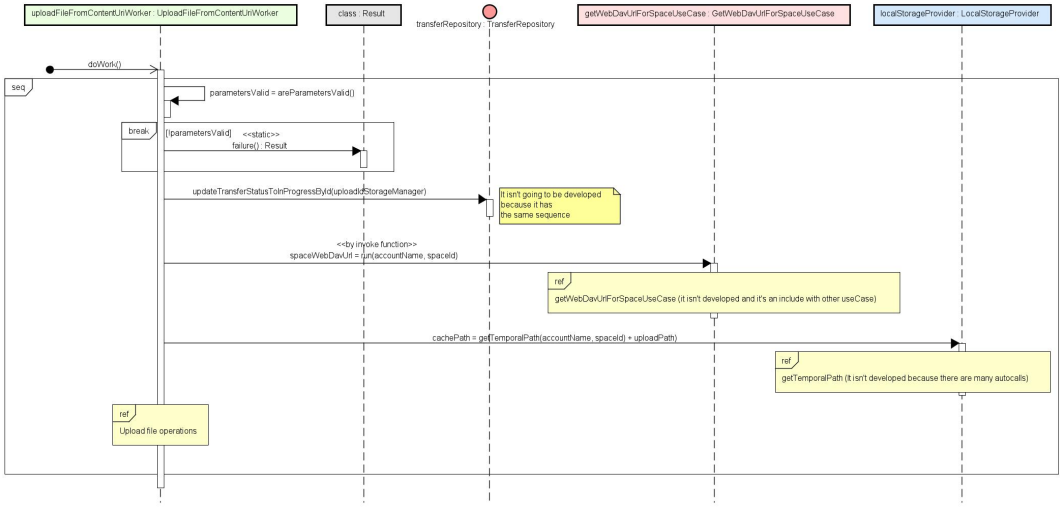

Figura 5.14: Diagrama de secuencia correspondiente a la función `doWork` de la Figura 5.1

En primer lugar, se comprueba que todos los parámetros que recibe el `Worker` son correctos. Una vez que se haya comprobado, en caso de no ser correcto se devolverá un fallo y, por tanto, el caso de uso será cancelado ya que no se puede realizar correctamente la subida de los ficheros. Una vez que se tienen todos los parámetros, se cambiará el estado de la subida a *In Progress* a través de una llamada al repositorio. En este caso no se ha desarrollado todo lo correspondiente a este método pues sigue la misma secuencia que se ha descrito para guardar cada uno de los ficheros en base de datos local. Una vez que se ha modificado ese estado, se ejecutará otro caso de uso, el cual recibe el nombre de `GetWebDavUrlForSpacesUseCase`. Para no extender demasiado todos los diagramas, se ha referenciado ese caso de uso sin entrar a detalle. Este caso de uso se ha representado con otro color más claro, ya que a pesar de no ser el centro del caso de uso también es importante e interviene en el mismo, siendo parte de un *include* hacia el `UseCase` principal. Tras llamar a ese caso de uso se consigue el `temporalPath`, que tampoco se ha desarrollado pues incluía muchas autollamadas, las cuales no son demasiado importantes, pues el objetivo principal de estos diagramas es representar la comunicación entre objetos. Tras haber realizado todas estas operaciones se llevará a cabo un proceso de subida de fichero el cual se encuentra referenciado en el cuadro con nombre

`Upload file operations`. El desarrollo de ese *ref* se encuentra reflejado en la Figura 5.15.

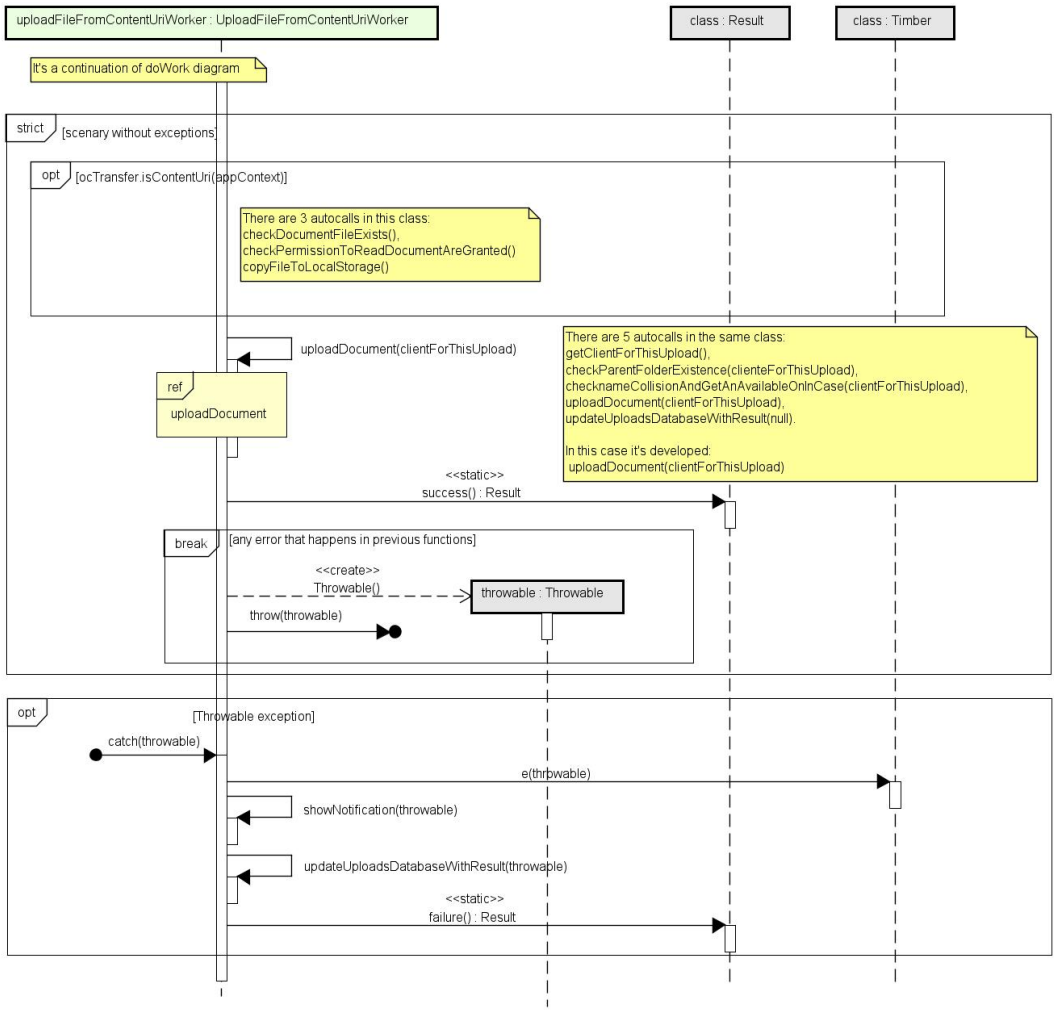

Figura 5.15: Diagrama de secuencia correspondiente al *ref* `Upload File Operations` de la Figura 5.14

En el diagrama de la Figura 5.15 se modela una estructura de tipo `try-catch` combinando un fragmento combinado `strict` con un fragmento combinado `break`. Esto significa que, cuando se cumpla la condición de guarda del `break`, se realizan los mensajes en su interior (en este caso se lanza una excepción) y se interrumpe el fragmento combinado exterior. Antes del `break` hay tres funciones agrupadas en un fragmento combinado `opt`, las cuales son autollamadas. Por este motivo no se han desarrollado y se han dejado simplemente las funciones anotadas. Después de ese bloque alternativo se tiene otras cinco llamadas a funciones que también se encuentran dentro de la misma clase por lo que no se han desarrollado todas, a excepción de una: `uploadDocument`. Esta función tiene la lógica asociada a la subida del fichero al servidor y por tanto se puede apreciar con bastante facilidad la comunicación que

existe entre los diferentes módulos. Cualquier de estas funciones que se encuentran dentro del bloque `strict` pueden generar errores. En caso de que esto suceda, en cualquiera de los casos se lanzará una excepción de tipo `Throwable`, la cual será tratada en el bloque `catch`, representado en el diagrama por un fragmento `opt`. El tratamiento simplemente consiste en mostrar una notificación de que algo ha fallado, además de actualizar el estado de cualquiera de las subidas a fallido.

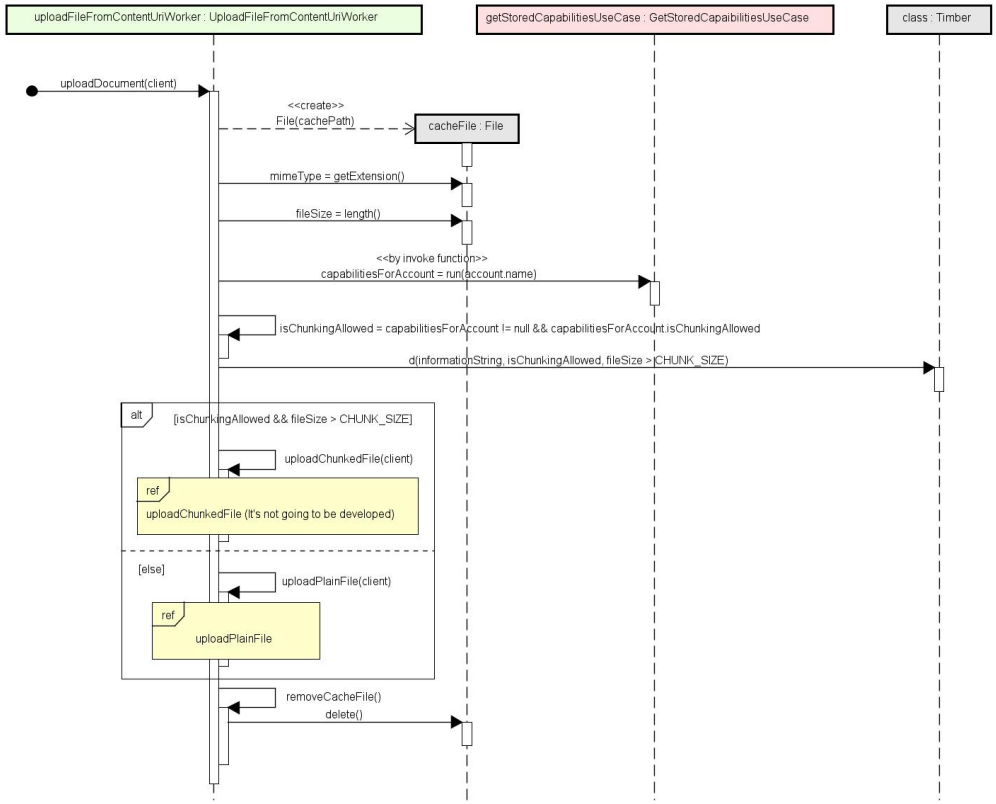

Figura 5.16: Diagrama de secuencia correspondiente a la función `uploadDocument` de la Figura 5.15

En la Figura 5.16 se ve el desarrollo de la función `uploadDocument`. En primer lugar se crea un objeto de tipo `File` a partir de una variable que se encuentra dentro del worker (`cachePath`). Este objeto recibe el nombre de `cacheFile`. Sobre ese objeto se obtiene una serie de variables útiles para el desarrollo del caso de uso. Una vez que se ha trabajado con ese objeto se obtienen las `capabilities` de la aplicación a partir de un caso de uso denominado `getStoredCapabilitesUseCase`. De esas capabilities se determina si hay que subir el fichero en pequeños trozos (*chunking*) o simplemente habrá que subir el fichero en su totalidad. En este caso de uso se hace especial atención a la subida del fichero completo, pues en el caso del *chunking* existe una lógica más compleja y que extendería todos los diagramas. Después de hacer las operaciones correspondientes se elimina ese objeto que se había creado al principio del método.

El método `uploadPlainFile`, que se iba a tratar en este caso de uso, está desarrollado en la Figura 5.17. La lógica de este método es bastante simple pues sigue el mismo patrón que cualquier otra operación que tenga dependencias con el resto de capas de la aplicación. Desde el `Worker` se llama al repositorio, el cual sigue siendo `transferRepository`. Su implementación se encuentra en el módulo *ownCloudData*, por lo que se muestra otro diagrama aparte con la respectiva llamada y la implementación correspondiente. Esto se encuentra detallado en la Figura 5.18.

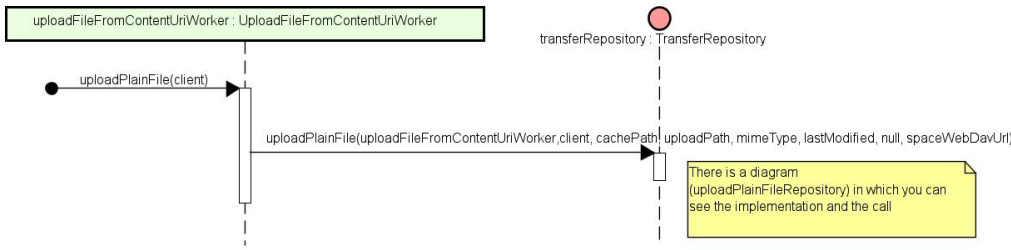

Figura 5.17: Diagrama de secuencia correspondiente a la función `uploadPlainFile` de la Figura 5.16

Como ya se había mencionado, esa implementación se encuentra en la capa de datos, de ahí que el color de la clase que participan en primer lugar nada más hacerse la llamada sea de color azul. Este método recibe varios parámetros que son importantes a la hora de realizar la operación de red correspondiente a la subida del archivo. Además se puede apreciar como existe una interfaz denominada `remoteTransferDataSource`. Como se lleva haciendo con todos, la implementación de la interfaz se encuentra en otro diagrama diferente.

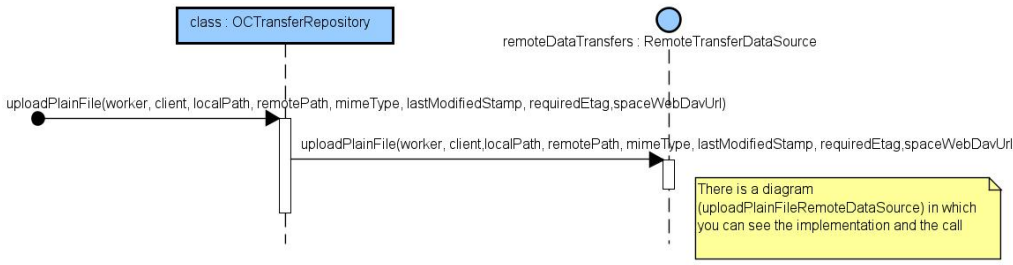

Figura 5.18: Diagrama de secuencia correspondiente a la implementación de `transferRepository` de la Figura 5.17

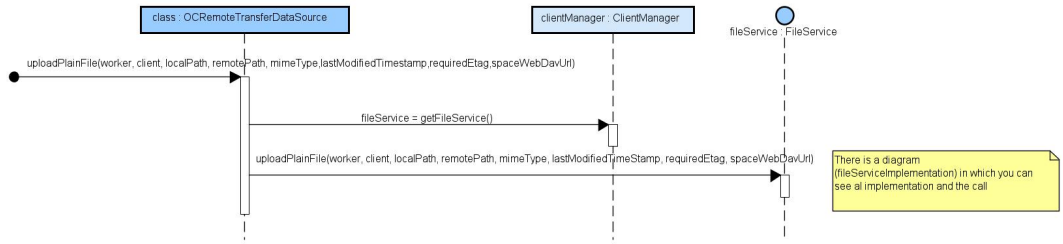

Figura 5.19: Diagrama de secuencia correspondiente a la implementación de la interfaz `remoteDataTransfer` de la Figura 5.18

A partir del diagrama de la Figura 5.19, ya se puede ver como comienza a entrar en juego el módulo que diferencia este caso de uso del otro, consulta del estado de las subidas, (*owncloud-ComLibrary*). Una vez que se hace la llamada a la implementación del `RemoteDataSource` se llama a la clase `ClientManager`, la cual se encarga de gestionar todos los servicios disponibles en la aplicación. A partir de que ya se ha obtenido ese servicio, se realiza una llamada a la interfaz `fileService` con todos los parámetros que se habían transmitido a lo largo de todo el flujo y que resultan necesarios para poder completar el caso de uso.

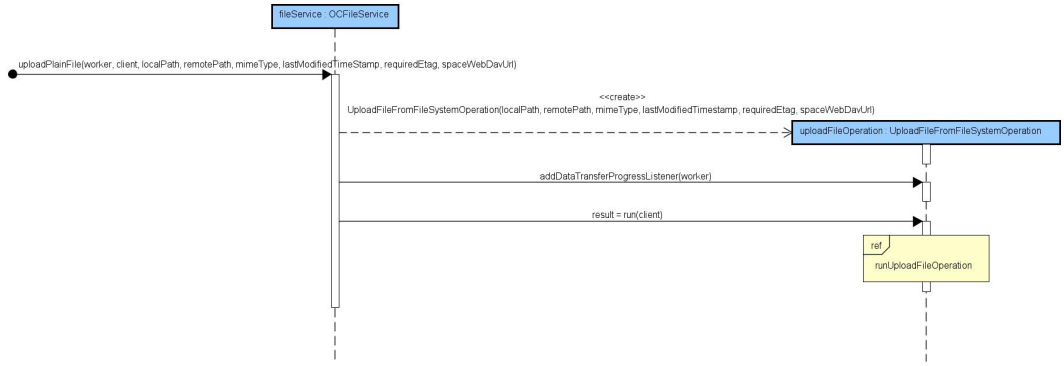

Figura 5.20: Diagrama de secuencia correspondiente a la implementación de interfaz `fileService` de la Figura 5.19

La Figura 5.20 muestra la creación de una operación `UploadFromFileSystemOperation` con los parámetros que se le habían pasado al método y tras ello se le añade toda la información que traía el `Worker`. Por último, se ejecuta el método `run` de la propia operación para ejecutar toda la lógica que tiene implementada, la cual se encuentra reflejada en la Figura 5.21.

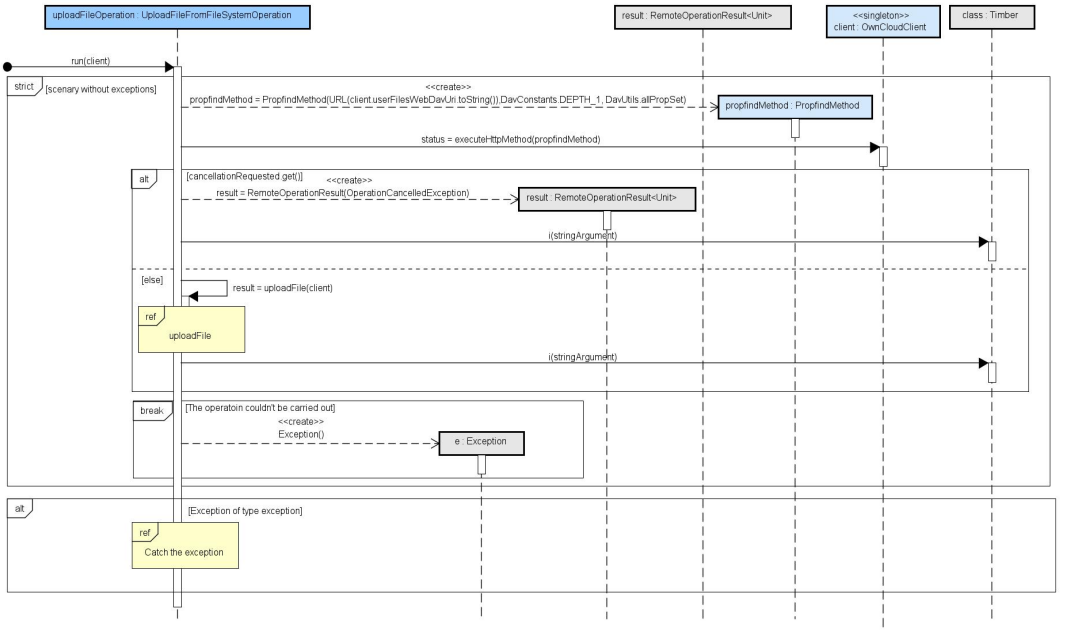

Figura 5.21: Diagrama de secuencia correspondiente a la implementación la función `run` de la Figura 5.20

Dentro de la propia función `run` se desarrolla una lógica bastante larga, pero que en esencia va haciendo casi todo el rato lo mismo hasta finalizar el caso de uso en su totalidad. En primer lugar, se crea un objeto de tipo `Propfind` y se ejecuta a través de una función denominada `executeHttpMethod`. Dependiendo del estado del resultado, se hace una cosa u otra. En cualquier caso, se puede producir una excepción de tipo `Exception`, cuyo tratamiento se ve reflejado en la Figura 5.22. Si todo sale como se esperaba, y sin ningún tipo de cancelación, se ejecutará todo el código que está asociado al *ref* denominado `uploadFile`. El desarrollo de ese bloque se encuentra reflejado en las Figuras 5.23 y 5.24.

Dentro de la función `uploadFile` se puede ver como existen construcciones de varios tipos con toda la información que se viene arrastrando y que se encuentra dentro de la clase `uploadFileOperation`. Primero, se crea el cuerpo de la solicitud que se va a enviar al servidor con los distintos parámetros. Tras ello, se crea la operación de tipo `PutMethod`. Una vez que se tiene todo creado, se ejecuta desde la clase `OwnCloudClient` y, dependiendo de su resultado (satisfactorio o fallido), se crea un objeto de tipo `RemoteOperationResult`. De esta manera acabaría el caso de uso de subida de un fichero.

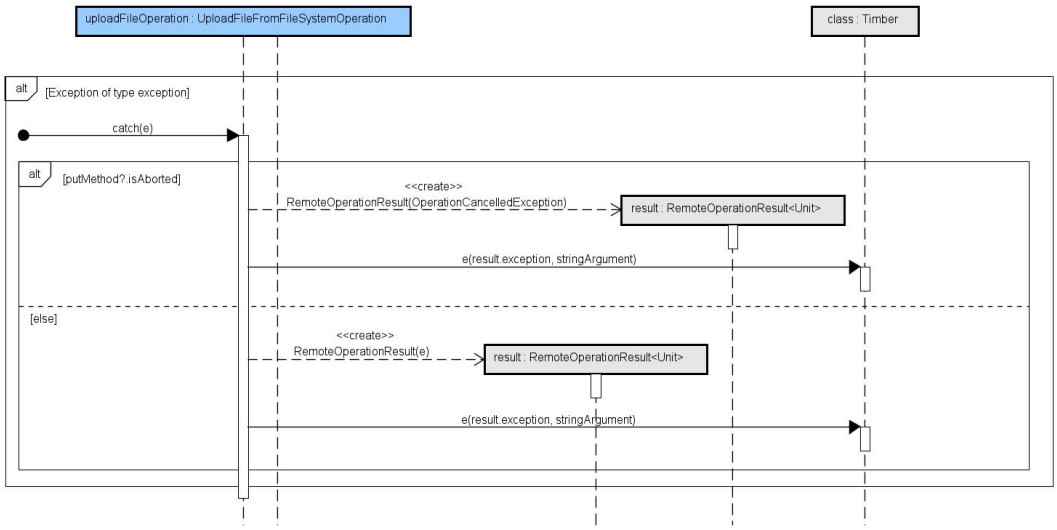

Figura 5.22: Diagrama de secuencia correspondiente a la captura de excepción de la Figura 5.21

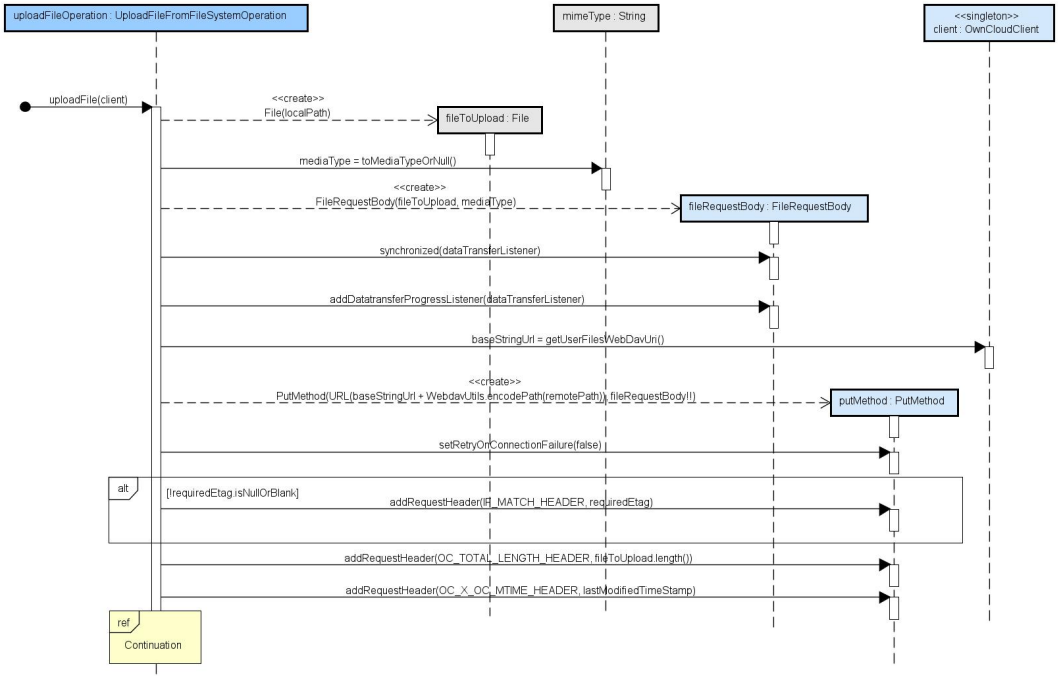

Figura 5.23: Diagrama de secuencia correspondiente a la función `uploadFile` de la Figura 5.22

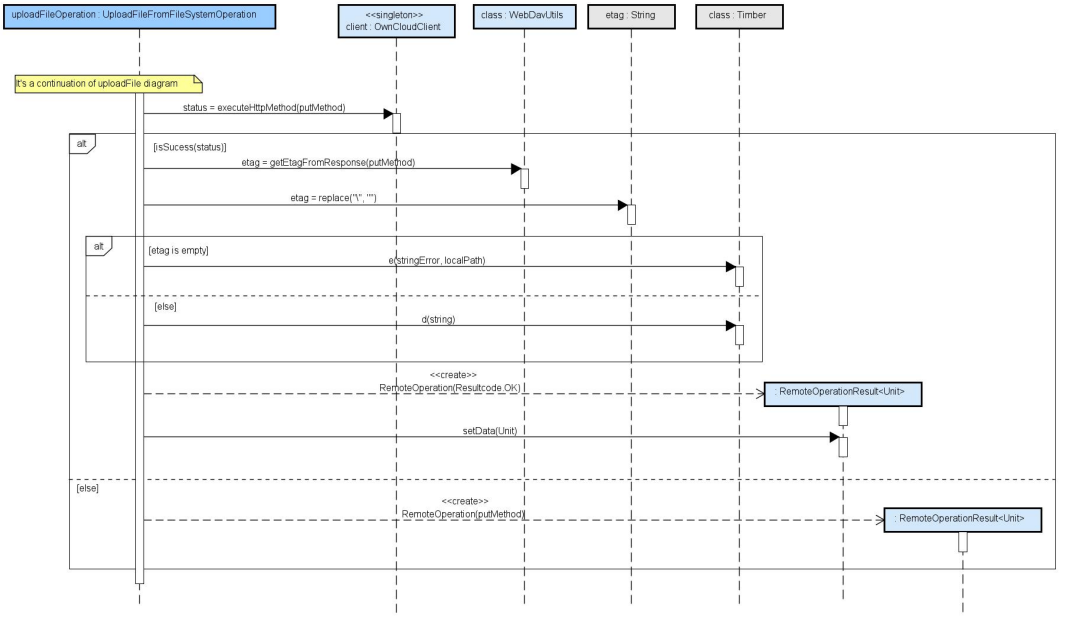

Figura 5.24: Continuación del diagrama de secuencia correspondiente a la Figura 5.23

5.1.2. Caso de uso: Consulta del estado de las subidas

El caso de uso se inicia cuando el usuario pulsa sobre la opción de las subidas, la cual está situada en la barra inferior de la aplicación. Esta primera interacción se puede ver en la Figura 5.25.

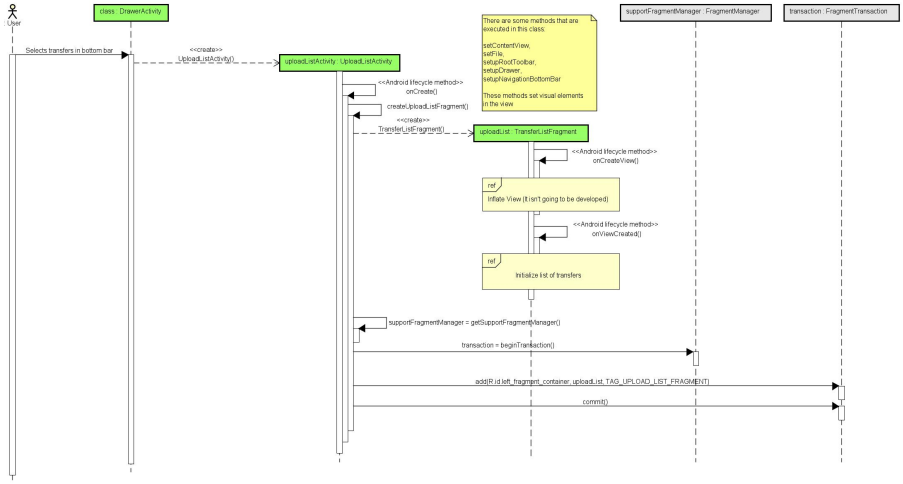

Figura 5.25: Diagrama de secuencia principal correspondiente al caso de uso de consulta de estado de las subidas

Una vez que el usuario ha pulsado sobre esa acción, se crea la actividad `UploadList-Activity`. Nada más crearse se ejecutan una serie de métodos, los cuales sirven para inicializar todos los elementos visuales que van a aparecer por pantalla. Una vez que se han creado todos estos elementos se crea un *fragment*, más concreto un *fragment* de tipo `TransferListFragment`, el cual recibe el nombre `uploadList`. Después de haberse creado el *fragment*, se llama al `FragmentManager` y se muestra ese *fragment* por pantalla. La creación del `uploadList` se puede ver en la Figura 5.26.

Lo primero que se hace dentro de ese *fragment*, es crear el ViewModel. En este caso es de tipo `TransfersViewModel` y recibe el nombre de `transfersViewModel`. La creación de esa clase está reflejada en la Figura 5.27. Siguiendo con el flujo, el siguiente paso que se realiza es crear el `transfersAdapter`. Esta clase será la encargada de distribuir todas las subidas, que se recuperen más adelante, en la lista que aparecerá por pantalla. Tras ello, se inicializan varios elementos de la vista. Por último, lo que se tiene son dos funciones diferentes.

En primer lugar está la función denominada `collectLatestLifecycleFlow`. Su funcionalidad es estar escuchando constantemente, comprobando si se ha producido algún cambio en la lista de las subidas que se van a recuperar de base de datos local. En caso de que se produzca alguna modificación, el *adapter* se encargará de pintar toda esa la lista en la pantalla correspondiente. La última función que aparece ahí comprueba si alguna descarga está en progreso y notifica al *adapter* sobre algún cambio que haya, modificando de esta manera el estado de cada subida en pantalla.

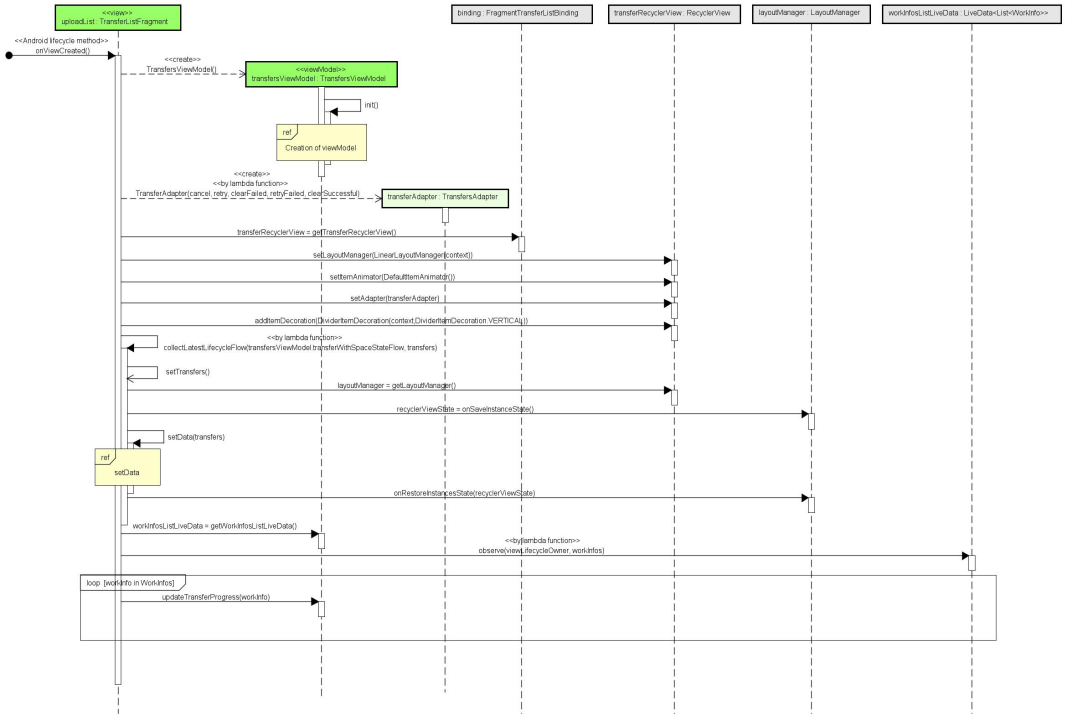

Figura 5.26: Diagrama de secuencia correspondiente a la creación del `TransferListFragment`

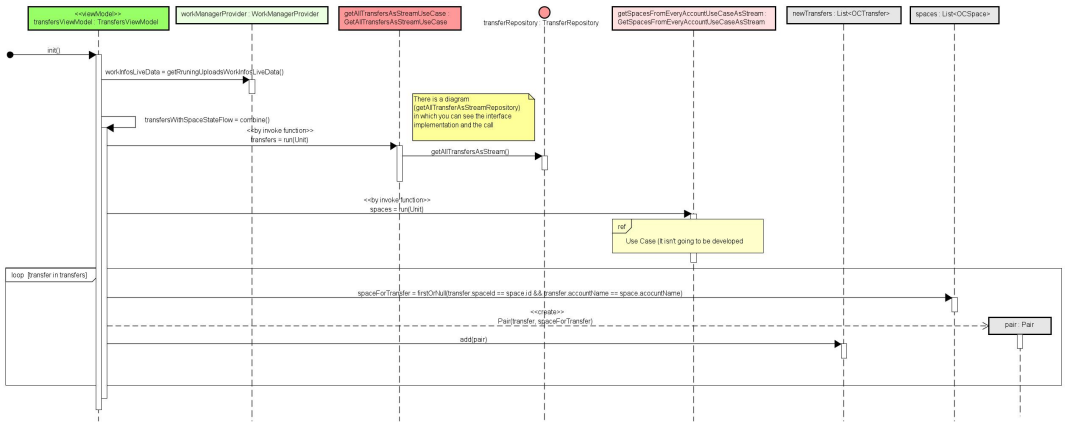

Figura 5.27: Diagrama de secuencia correspondiente a la creación del `TransfersViewModel`

En la Figura 5.27 se muestra todo aquella relacionado con la creación de la clase denominada `transfersViewModel`. Dentro del `init`, la cual es la función que se inicia nada más crear la clase se inicializan dos atributos, que como ya se había mencionado anteriormente, se va estar observando para pintar lo correspondiente por pantalla. El primer de

ellos (`workInfosLiveData`) indica el progreso de todas las subidas que estén teniendo lugar en el sistema. El segundo de los atributos que se recuperan nada más crear la clase son todas las subidas que haya en base de datos local y si ha habido algún cambio sobre ellas (`transfersWithSpaceStateFlow`). Este atributo es un `combine` de varios elementos. En primer lugar se obtiene todos los subidas que haya en base de datos local a través del caso de uso `GetAllTransfersAsStreamUseCase`. La implementación de la interfaz (`transferRepository`) se encuentra reflejada en la Figura 5.28. Después de haber ejecutado ese caso de uso, se ejecutará otro caso de uso diferente el cual recibe el nombre de `GetSpacesFromEveryAccountUseCaseAsStream`. La funcionalidad de esta caso de uso es conseguir todos los spaces que estén asociados a la cuenta del usuario. Una vez que se tiene la lista tanto de las subidas como de los spaces, se va a crear un objeto `Pair` por cada par de elementos. Se va comprobando a que space está asociado cada subida y se van asociando entre sí, para luego añadirlo a la lista correspondiente. Esta lista será la que el `transfersAdapter` utilice para poder mostrar todos los elementos por pantalla y con la información correspondiente.

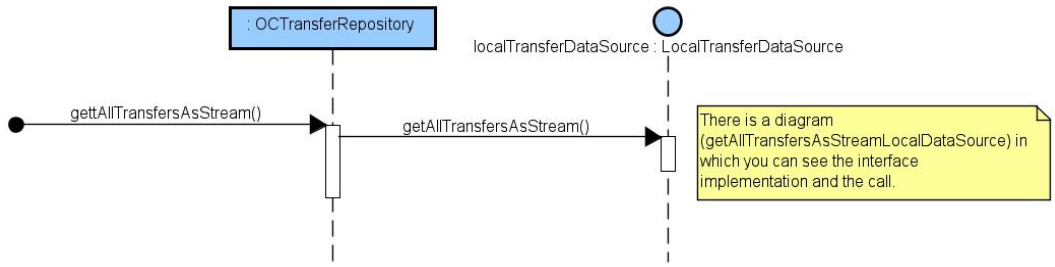

Figura 5.28: Diagrama de secuencia correspondiente a la implementación de la interfaz `transferRepository` de la Figura 5.27

Siguiendo el flujo de los diagramas anteriores, en la Figura 5.28, se muestra la implementación de la interfaz para este caso de uso. Siguiendo el mismo patrón que el caso de uso de subida de un fichero, dicha clase presenta el prefijo `OC`. La llamada que se realiza es sobre el `LocalTransferDataSource`, el cual es un atributo de esa clase. A esa llamada no se pasará ningún parámetro, pues simplemente consiste en obtener todos los elementos de base de datos local. Nuevamente al ser una interfaz, se tiene otro diagrama en el que se muestre dicha implementación. En este caso corresponde con la Figura 5.29.

En primer lugar se llama a la clase `transferDao` para realizar una consulta a la base de datos local y obtener todas las tranfers que haya almacenadas en dicho sistema. Tras ello se crea una nueva lista en la que se irá guardando cada uno de esos elementos para ya transformados en el modelo del sistema (`OCTransfer`). Tras haber transformado todas las `transfersEntities` al correspondiente modelo, se procederá a ordenarlas según su tipo. Sobre esa lista que se había creado se ejecuta la función `groupBy` con el atributo `status` como argumento de dicha función. Los dos siguientes pasos son crear dos listas en los que se irán guardando todas las subidas ya ordenadas. La primera de esa lista es un `ArrayList` mientras que la segunda es de tipo `MutableList`. Una vez que se han creado, se procederá a ordenarlas. En la primera posición del `ArrayList` se almacenarán aquellas subidas que están en progreso. En la siguiente posición de la lista se guardan aquellas que estén en cola. Seguidas a esas, se encuentran todas las subidas que hayan fallado durante todo el proceso

de subida. Por último, están almacenadas todas aquellas subidas que han sido satisfactorias y que ya han terminado todo el proceso de subida. Cuando ya se tiene ordenadas sobre esa lista se guardarán una a una en la `MutableList` que se había creado anteriormente.

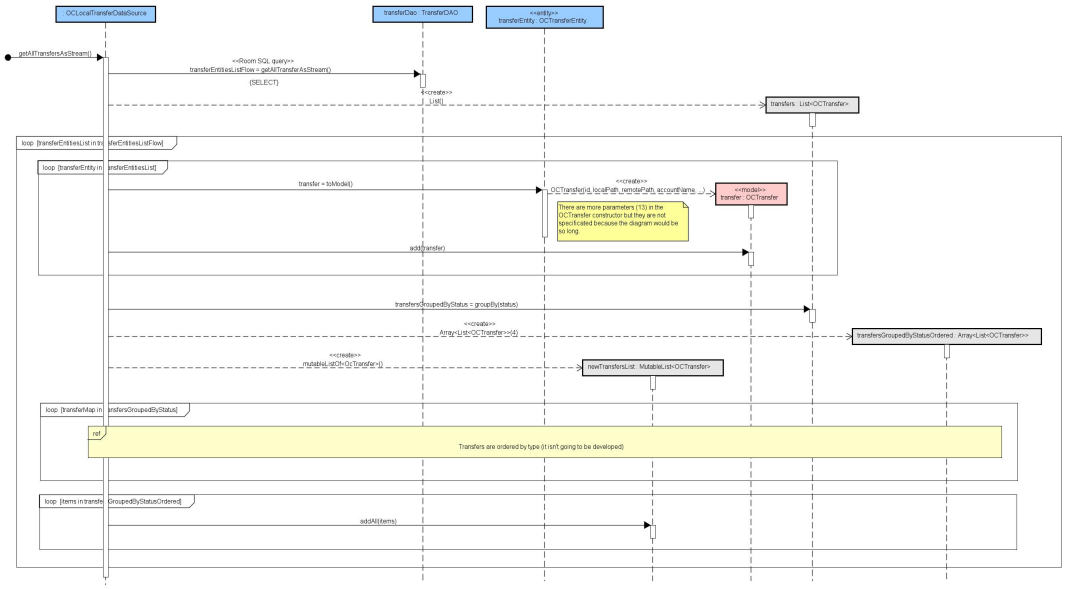

Figura 5.29: Diagrama de secuencia correspondiente a la implementación de la interfaz `localTransferDataSource` de la Figura 5.28

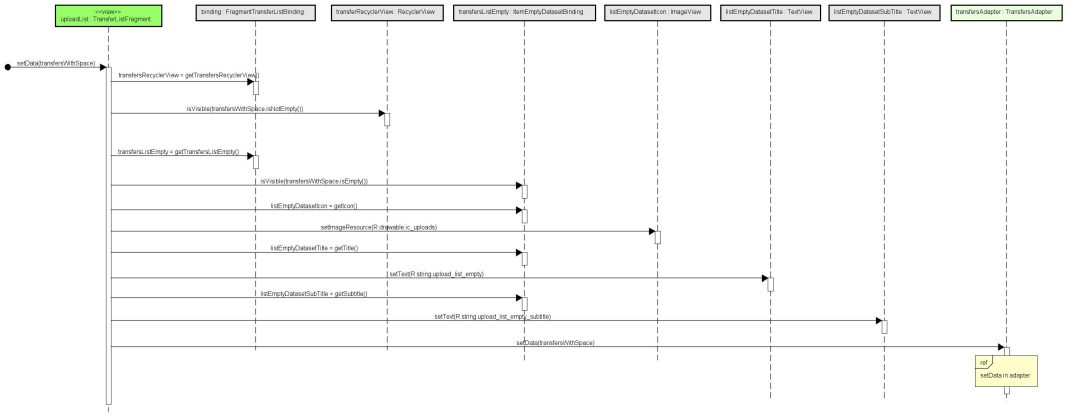

Figura 5.30: Diagrama de secuencia correspondiente a la implementación de la función `setData` de la Figura 5.26

Ahora viene toda la secuencia relacionada con mostrar toda la información que se ha sacado de la base a datos. En la Figura 5.30 se puede ver como todo esto inicia con la función `setData`. Tras ello, a través del binding, se setean todos los elementos iniciales que corresponde a la lista vacía. Finalmente se llama al `transfersAdapter` con todas las subidas asociadas a cada space para que lo pinte por pantalla. Toda esta lógica se encuentra desarrollada en la Figura 5.31.

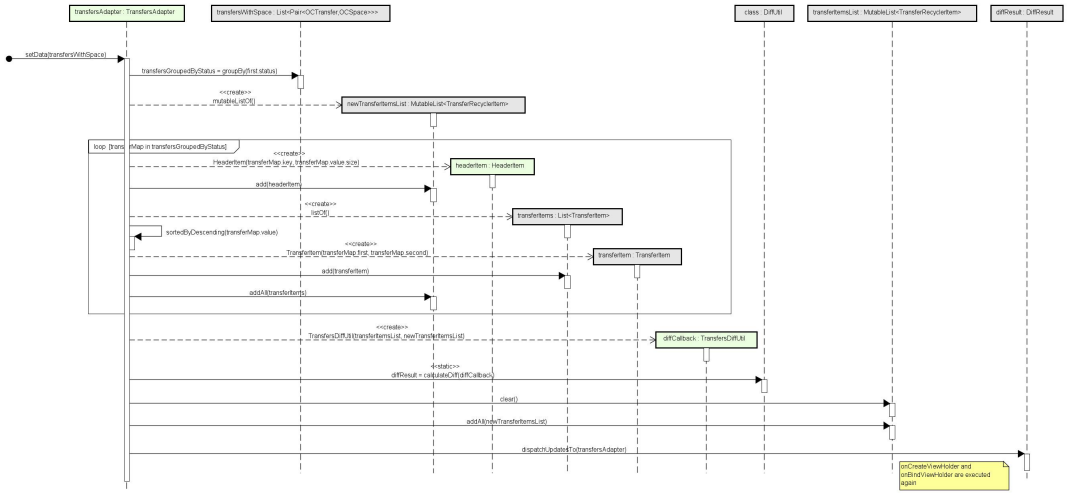

Figura 5.31: Diagrama de secuencia correspondiente a la implementación de la función `setData` de la Figura 5.30

Parte II

Reingeniería: migración a Jetpack Compose

Capítulo 6

Rediseño de un caso de uso

El caso de uso que se ha decidido migrar, y en consecuencia, rediseñar es: **consulta del estado de las subidas**. Dentro de todo este proceso no se ha contemplado cambiar el aspecto visual de la pantalla por lo que la apariencia de la pantalla es similar a la que ya se tenía antes de todo el proceso. Para ello se han realizado una serie de tareas hasta conseguir el resultado final. El primer paso que se ha llevado a cabo es la lectura de la *Guía de migración de XML a Jetpack Compose* de **Juan Carlos Garrote Gascón** [23], en la cual se exponían diferentes formas para llevar a cabo la migración, lo que implica tener que rediseñar el caso de uso en cuestión. Existen dos formas diferentes de poder llevar a cabo esa migración:

- **Migración completa.** Se realiza una migración completa de toda la aplicación a la nueva tecnología, manteniendo en todo momento la lógica.

- **Migración por pantallas.** Se realiza una migración progresiva por pantallas, manteniendo la lógica de cada una de ellas, teniendo una aplicación híbrida hasta que se complete todo el proceso.

Para este caso concreto se ha decidido realizar la **migración por pantallas**, ya que el caso de uso que se desea rediseñar únicamente aplica a una pantalla en toda la aplicación. Dentro de esta técnica de migración existen otras dos posibles opciones: la pantalla a migrar sea una actividad de Android o sea un fragmento de Android alojado en una actividad. Dependiendo del caso que se tenga, se realizará de una u otra manera. En este caso se realizará la migración de un fragmento ya que la clase antes del rediseño (`TransferListFragment`) es un fragmento nativo de Android, como su propio nombre indica.

6.1. Rediseño de la vista estática del caso de uso

Algo a tener en cuenta en la migración, es que ésta solamente se aplica sobre la capa de presentación, que en el caso de *ownCloud* corresponde con el módulo *owncloudApp*. Por ello, tanto a nivel estático como a nivel dinámico, se han rediseñado únicamente las clases que se encuentran dentro de este módulo. El resultado se puede apreciar en la Figura 6.1. Previamente, tal y como se puede ver en la Figura 4.8 la clase sobre la que se va a realizar la migración recibía el nombre de `TransferListFragment`. Esta clase presentaba una asociación *1-1* con otra clase denominada `TransfersAdapter`. Esta última clase era la responsable de pintar por pantalla cada una de las subidas que se obtenían de la base de datos local, además de organizar cada uno de los elementos de la interfaz de usuario. A su vez, esta clase presentaba una dependencia con `TransfersDiffUtil`, ya que necesitaba, a través de un *callback*, conocer cuando las subidas se habían actualizado en la base de datos local. Tras realizar el rediseño, se puede apreciar que el fragmento de Android ha cambiado de nombre a `TransferListComposeFragment`. Además se ha eliminado esa asociación que tenía con la clase `TransferAdapter`, ya que ahora toda la lógica asociada con los elementos de la interfaz de usuario se encuentra alojada en la misma clase. Obviamente, si ya no se tiene una relación con `TransferAdapter`, la clase `TransfersDiffUtil` queda suprimida por igual, la cual servía para que el `RecyclerView` tuviera una animación y fuera más eficiente. Esto es debido a que en Jetpack Compose cada *composable* lleva a cabo una recomposición en caso de que alguno de los datos que reciba como parámetro cambie. Todo esto se explicará en el Apartado 7.3.4.

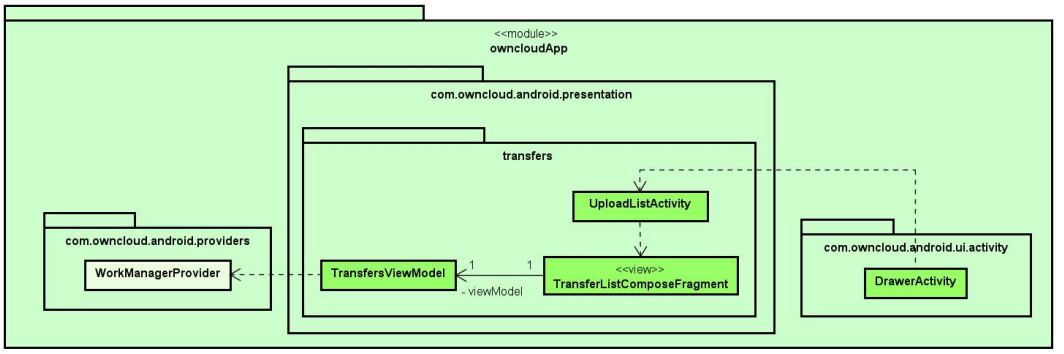

Figura 6.1: Modules&UsesStyle del módulo *owncloudApp* para el caso de uso de consulta del estado de las subidas después del rediseño

6.2. Rediseño de la vista dinámica del caso de uso

En cuanto a la parte dinámica del caso de uso (diagramas de secuencia), se ha llevado a cabo un cambio bastante apreciable, ya que en este caso toda la lógica de la IU relacionada con el caso de uso se encuentra, tal y como se había mencionado antes, alojada en una misma clase: `TransferListComposeFragment`. La clase `UploadListActivity` será la encargada de crear una instancia de `TransferListComposeFragment` y ya ella será la encargada de pintar por pantalla los elementos y transferir la orden a otras clases encargadas, en caso de que el usuario interactúe con la interfaz.

En la Figura 6.2 se ve como existe una similitud con respecto a la Figura 5.25. La única diferencia que existe es que en el fragmento solamente implementa una de las funciones correspondientes al ciclo de vida de un fragmento en Android. En este caso esa función es `onCreateView`. Una vez que se ha creado el fragmento correspondiente, se lleva a cabo la creación de la clase `TransfersViewModel`. Este proceso es exactamente idéntico al que se tenía antes del rediseño por lo que no se muestra ningún diagrama relacionado con ello. El gran cambio que se había mencionado, viene dentro de la propia función `onCreateView`. Anteriormente, era necesario crear una clase de tipo `TransferAdapter` para poder mostrar por pantalla todos los elementos, dependiendo del estado de las subidas. Esto va a cambiar totalmente, ya que tras el rediseño únicamente se hará una llamada a la función `Transfers`, la cual es de tipo *composable*. Esta función será la que se encargue de crear la interfaz dependiendo de ciertos valores que provengan del `TransfersViewModel`. Este proceso se ve reflejado en la Figura 6.3.

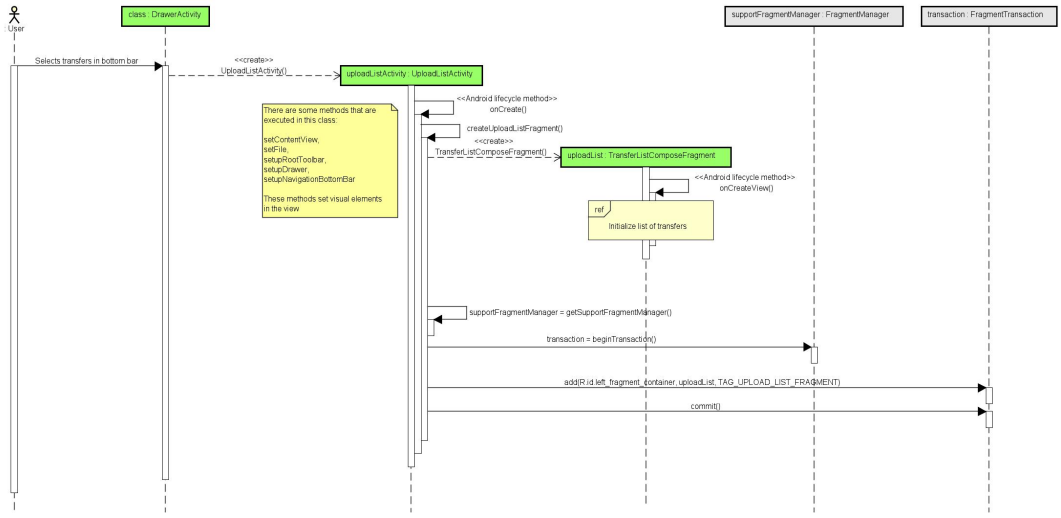

Figura 6.2: Diagrama de secuencia principal correspondiente al caso de uso: Consulta del estado de las subidas

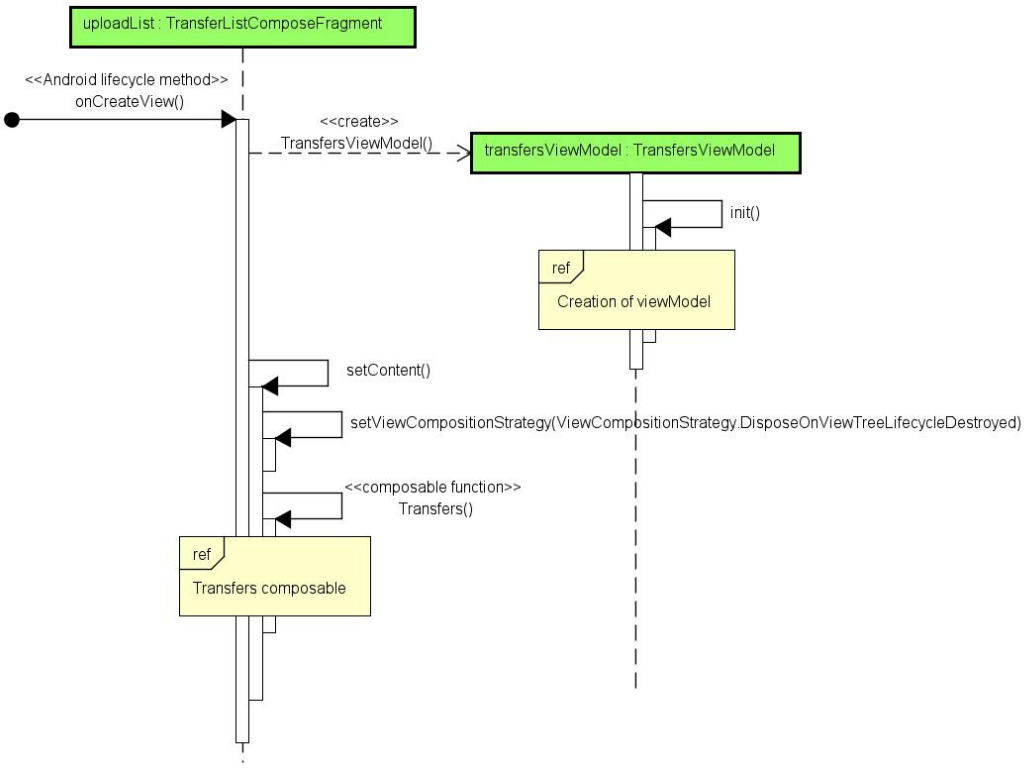

Figura 6.3: Diagrama de secuencia correspondiente a la creación del `TransferListComposeFragment` de la Figura 6.2

Una vez que se llama a la función *composable* `Transfers`, se llevan a cabo dos pasos que resultan esenciales para el correcto funcionamiento del caso de uso. En primer lugar se crea una variable con el nombre `transfersState` en la cual se irá almacenando toda la información relacionada con todas las subidas que están almacenadas en la base de datos local junto con el `space` al que están asociadas. Esta variable se encontrará constantemente esperando a algún cambio que se produzca en la base de datos local, y en el momento en el que esto ocurra la propia función *composable* se volverá a ejecutar pero con otros parámetros diferentes, y por tanto, con una representación visual diferente. Por otro lado, también existe una variable que recibe el nombre de `workInfos`, en la cual se almacenará el estado del proceso que se está encargando de subir un fichero y su progreso en caso de que se se encuentra en cola o en progreso. Tras ello se comprobará si el conjunto de subidas no está vacío. En caso de que no haya ninguna subida en la base de datos local se llamará a la función *composable* `EmptyList`, la cual se encargará de pintar por pantalla toda la interfaz de usuario con un mensaje de lista vacía. En caso de que esa lista no sea vacía entonces se llamará a otra función *composable* denominada `TransferList`, pasándola por parámetro las dos variables que se habían definido previamente. Todo esto se puede ver en la Figura 6.4.

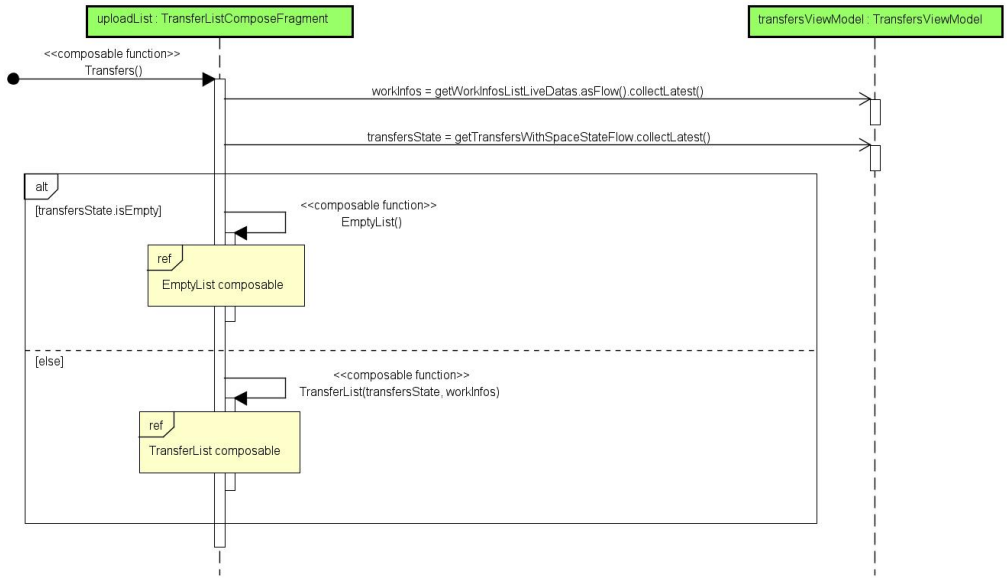

Figura 6.4: Diagrama de secuencia correspondiente a la función *composable* `Transfers`

En caso de que la lista de las subidas no se encuentre vacía, todas aquellas subidas (que se encuentran en la base de datos local) se ordenarán según su estado. Una vez que se encuentran ordenadas se mostrará por cada estado una cabecera con cierta funcionalidad dependiendo del tipo. Esta cabecera corresponde con la función *composable* `UploadGroup`. Esta función recibe dos parámetros únicamente: el número de subidas que hay en un determinado estado y ese estado en cuestión. Por cada cabecera o grupo de subidas, se ordenan todos los elementos según su tiempo de vida de más recientes a más antiguas y tras ello, se llamará a la función *composable* `TransferItem` con tres parámetros diferentes: la subida a pintar, el `space` en el que se encuentra y la información de progreso de cada subida (`workInfo`) con la cual se pintará la barra de progreso en caso de que se esté subiendo el archivo.

La composición de la interfaz de usuario asociada a la lista vacía (no hay subidas en la base de datos local) se puede ver desarrollada en la Figura 6.5. En ella se puede apreciar como en primer lugar existe una función *composable* `Column`, la cual sirve para organizar el resto de elementos en forma de columna. Dentro de esa función se tienen otras tres funciones *composables* correspondientes a los tres elementos que se muestran por pantalla: el icono, el título y el subtítulo.

Por otro lado, la composición de la interfaz asociada a la lista de las subidas en caso de que no sea vacía, se puede ver desarrollada en la Figura 6.6. Al igual que se ha mencionado antes, se mostrará una cabecera por cada estado de las subidas una vez que ya están agrupadas y, tras esa cabecera, todas las subidas que se encuentren en ese determinado estado. La particularidad que se puede ver aquí, es que no es necesario tener un elemento `RecyclerView` en la interfaz y una clase denominada `TransfersAdapter` para mostrar todas los elementos en forma de lista. Toda esta lógica se hace de forma automática con la función *composable* `LazyColumn`.

Figura 6.5: Diagrama de secuencia correspondiente a la función *composable* `EmptyList`

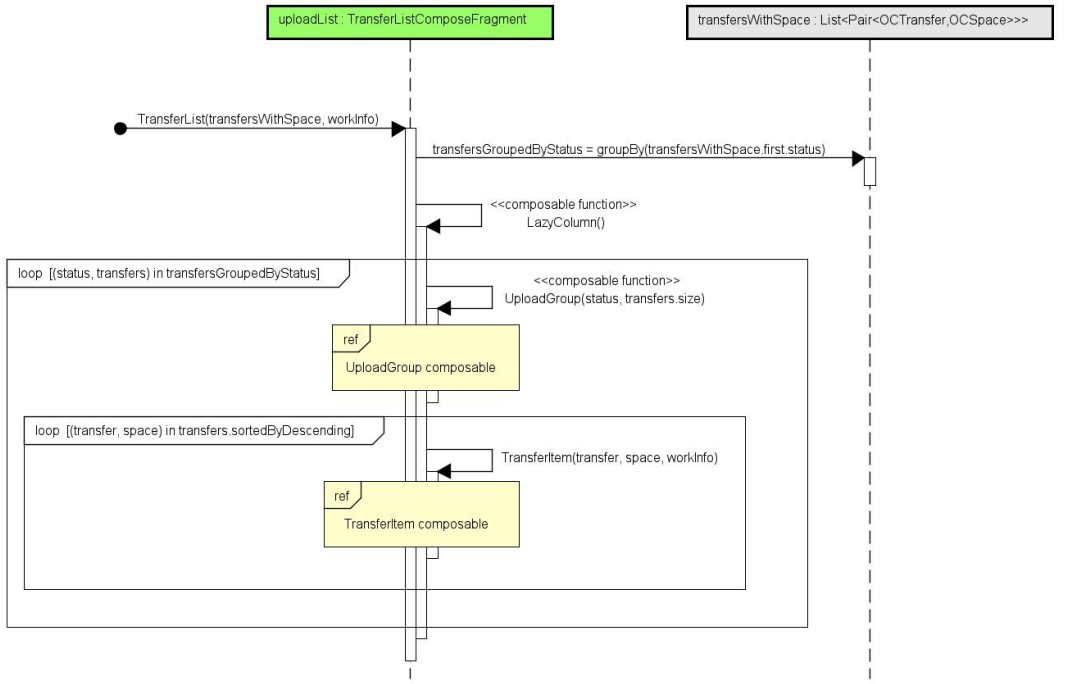

Figura 6.6: Diagrama de secuencia correspondiente a la función *composable* `TransferList`

Cada una de esas cabeceras que se han mencionado están estructuradas según se muestra en la Figura 6.7. Dependiendo de la posición en la que se quieran organizar los elementos se utilizarán las funciones *composables* `Row` o `Column`. Dentro de cada una de estas funciones se van añadiendo elementos de IU según se vayan requiriendo. En este caso las funciones que se han utilizado han sido `Text`, `Spacer`, `Divider` y `Button`, cada uno con sus respectivos

parámetros para personalizar la apariencia. Una particularidad que se puede apreciar es que dependiendo del estado de la subida se mostrarán diferentes elementos gráficos por la pantalla. En caso de que el estado sea *failed* o *succeeded* se mostrará por pantalla un botón para borrar todos los ficheros que se encuentren en ese estado. Por otra parte, en caso de que no se haya subido correctamente, se mostrará un botón para poder reintentar la subida. Esta lógica es la correspondiente al segundo fragmento alternativo de la Figura 6.7. Por último se mostrará una línea divisoria únicamente en los dos primeros casos que se han expuesto previamente: el estado es satisfactorio o fallido. Para el resto de los casos se mostrará la base de la interfaz sin todos estos botones.

Una vez que se ha creado la cabecera, el siguiente paso es mostrar cada una de las subidas que se encuentran en la base de datos local. Cada uno de estos elementos corresponde con la función *composable* `TransferItem`. En esta función se componen todos los elementos que mostrarán la información asociada a cada uno de los ficheros. Tal y como se puede apreciar en la Figura 6.8, el primer paso es crear un objeto de tipo `File` a partir de la variable `remotePath` almacenada en cada uno de los objetos de tipo `OCTransfer`. A partir de ese momento se comenzará a crear la interfaz de usuario usando las correspondientes funciones *composables*. Como la extensión de esta función es bastante larga, únicamente se hará mención de todas aquellas cosas que resultan interesantes y diferentes respecto a otras funciones *composables* que ya han aparecido. Dependiendo del estado en el que se encuentre el archivo que se quiere subir, se mostrará un texto u otro. Por ejemplo: en caso de que esa subida no haya fallado se mostrará por pantalla el intervalo de tiempo que ha transcurrido desde que se ha subido hasta la actualidad. En caso de que la subida no se haya realizado de forma satisfactoria ya sea porque ha fallado o porque está en progreso se mostrará un pequeño texto con su estado correspondiente. Por otro lado, en caso de que la subida esté en progreso se mostrará una linea de progreso permitiendo al usuario saber cuánto falta para que el fichero se termine de transferir. Aquí es donde entra en juego la variable `workInfos`. Existe una función dentro del propio fragmento que se encarga de buscar el elemento, que corresponde a la subida en cuestión, en esa variable y actualizar el estado de la barra de progreso dependiendo del valor que provenga de la clase encargada de gestionar esa variable, en este caso el `Worker`. Por último mencionar que existe una función *composable* denominada `SpacePathLine`, la cual únicamente se ejecuta en caso de que exista un `space` asociado a esa subida. Esto es debido a que la propia aplicación de *ownCloud* presenta dos tipos de versiones diferentes: *oCIS* y *oC10*. En el caso de que el usuario se encuentre en una versión *oC10*, la funcionalidad de los `spaces` no existe y por tanto el valor de la variable `space` que recibe la función *composable* tendrá un valor `null`. En este caso no se mostrará toda esa información asociada ya que resultaría inútil, pues no existe ningún valor asociado. En caso contrario (el usuario está utilizando la versión *oCIS*) la variable `space` sí que contendrá información relevante y por tanto, se mostrará todo eso por pantalla.

La función *composable* `SpacePathLine` tiene una estructura bastante simple, tal y como se puede apreciar en la Figura 6.9. En primer lugar, existe una función *composable* `Row` que sirve para distribuir el resto de los elementos en forma de fila. En su interior existen tres funciones *composables* diferentes: dos *composables* de tipo `Text` y un *composable* `Icon`. Éste último mostrará un icono dependiendo del `space` en el que se encuentre el archivo. Existen dos posibles opciones: el archivo se encuentra en el `personal space` o se encuentra en un `space` compartido. En cualquiera de estos dos casos se mostrará la ruta en la que se encuentra además del nombre del `space`, lo cual corresponde a las dos funciones *composables* `Text`.

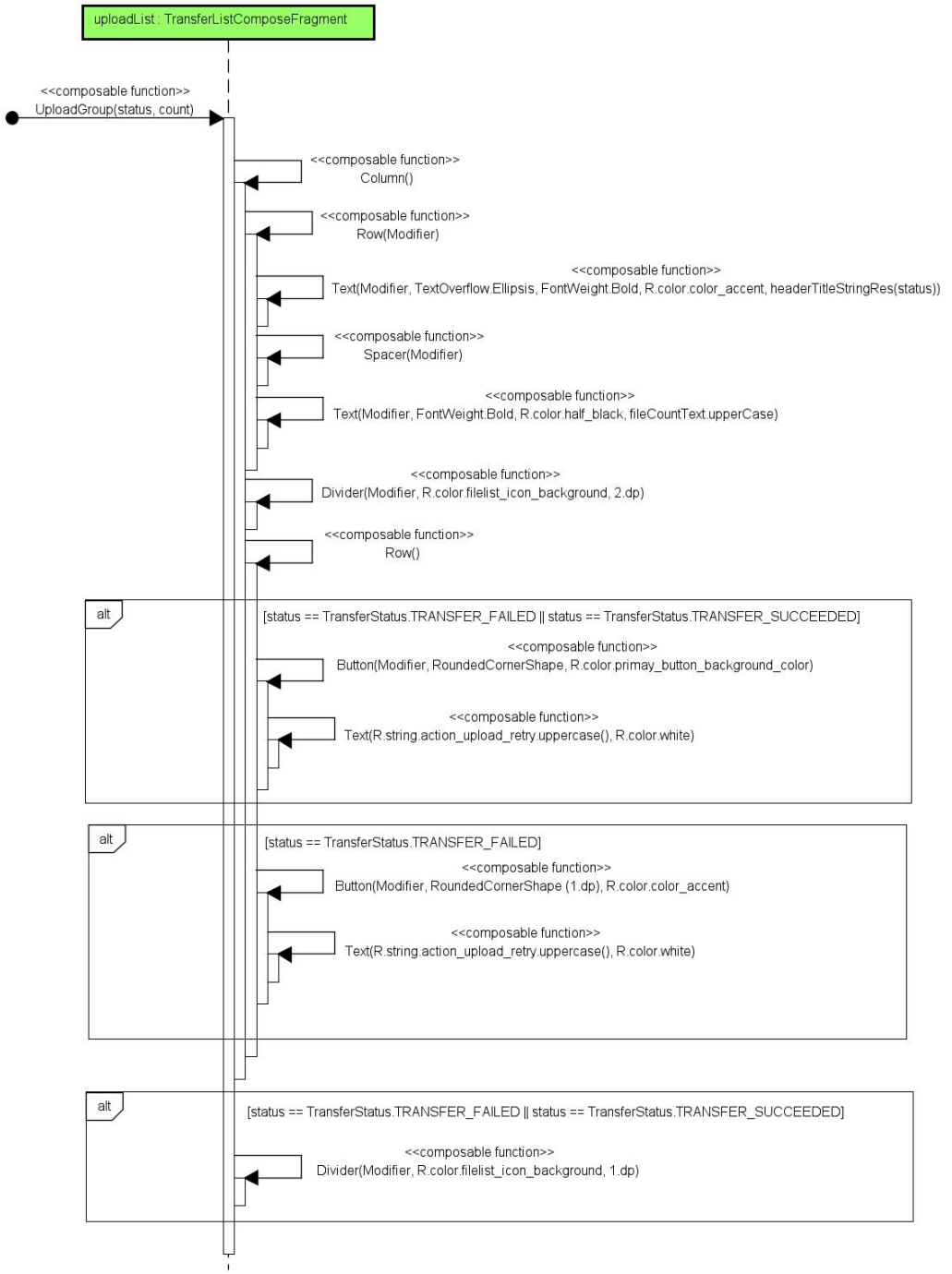

Figura 6.7: Diagrama de secuencia correspondiente a la función *composable* `UploadGroup`

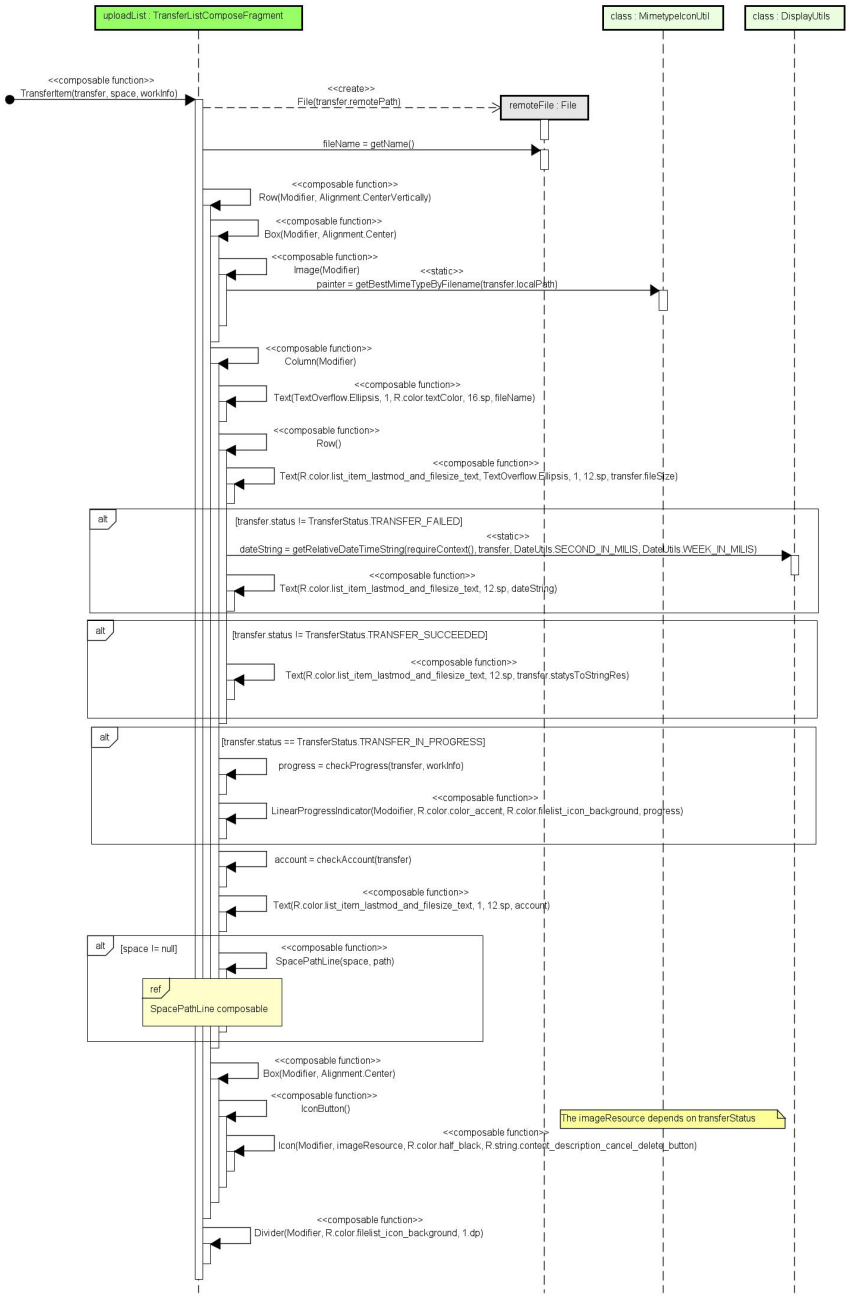

Figura 6.8: Diagrama de secuencia correspondiente a la función *composable* `TransferItem`

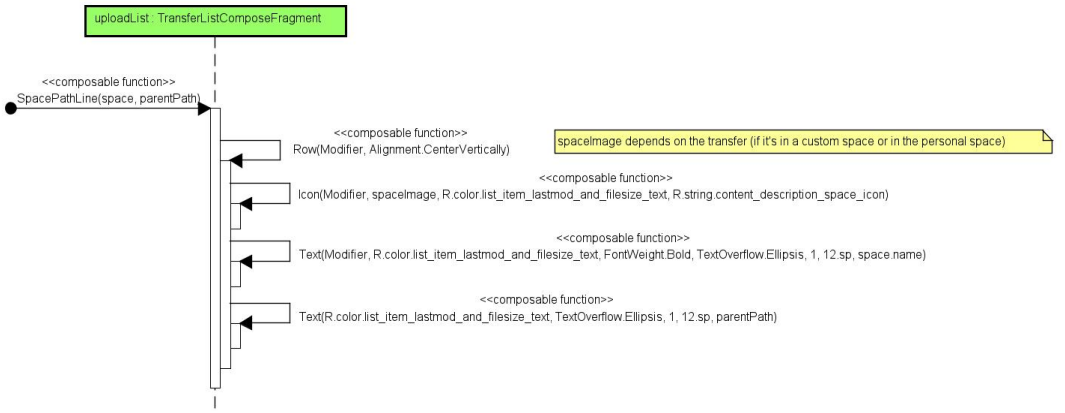

Figura 6.9: Diagrama de secuencia correspondiente a la función *composable* `SpacePathLine`

Capítulo 7

Implementación y pruebas del caso de uso rediseñado

7.1. Implementación

7.1.1. Proceso seguido

El proceso que se ha seguido para completar la migración ha sido la **migración por pantallas**, en contraposición a la construcción de la aplicación por completo en Jetpack Compose, tal y como se había mencionado en el Capítulo 6.2. El motivo por el que se ha decidido seguir este método es porque la migración únicamente se va a realizar sobre una pantalla, en este caso la pantalla relacionada con el caso de uso: **consulta del estado de las subidas**. Además, como se indica en la *Guía de migración de XML a Jetpack Compose* de **Juan Carlos Garrote Gascón** [23] existen dos posibles opciones en la migración por pantallas: la pantalla a migrar es una actividad o es un fragmento de Android. Para este caso concreto, se tiene un fragmento de Android y ésta será la única clase en la que se van a realizar los cambios.

El primer paso es crear un nuevo fragmento de Android, el cual recibe el nombre de `TransferListComposeFragment`. Cuando se tenga creado en su respectivo paquete, se deberá crear una instancia de ese fragmento en el lugar que corresponde y eliminar la anterior instancia que estaba definida con vistas XML. La clase encargada de crear ese fragmento es `UploadListActivity` y el lugar concreto donde se realiza dicha operación es dentro de la función `createUploadListFragment`. En el Fragmento de código 7.1 se puede ver esta lógica. Además, se puede apreciar cómo la anterior creación del fragmento se encuentra comentada, debido a que se busca realizar una comparación entre lo que se tiene actualmente y lo que se tenía antes de la migración. Por brevedad, los fragmentos de código que se muestran omiten partes no relativas a lo que se va a explicar.

```
1       public class UploadListActivity extends FileActivity {
2
3       ...
4
5       private void createUploadListFragment() {
6           TransferListComposeFragment uploadList = new
                TransferListComposeFragment();
7           //TransferListFragment uploadList = new TransferListFragment
                ();
8           ...
9       }
```

Fragmento de código 7.1: Creación del fragmento de Android en la clase `UploadListActivity`

Tras haber creado la instancia del fragmento, se deberá construir la propia clase a partir de las funciones del ciclo de vida de un fragmento de Android. En este caso sólo será necesario crear la función `onCreateView`, en la cual se va a definir la interfaz de usuario y su respectivo comportamiento a través de funciones *composables*.

En el Fragmento de código 7.2 se puede ver cómo se realiza una llamada a la función `Transfers`, la cual es de tipo *composable* y presenta su propia lógica interna.

```
1   class TransferListComposeFragment : Fragment() {
2
3       private val transfersViewModel by viewModel<TransfersViewModel
            >()
4
5       override fun onCreateView(
6           inflater: LayoutInflater,
7           container: ViewGroup?,
8           savedInstanceState: Bundle?
9       ): View {
10          return ComposeView(requireContext()).apply {
11              setContent {
12                  ...
13                  Transfers()
14              }
15          }
16      }
17      ...
18  }
```

Fragmento de código 7.2: Función `onCreateView` de la clase `TransferListComposeFragment`

Llegado este punto existen dos estrategias posibles para llevar a cabo la migración a través de funciones *composables*:

- **De abajo hacia arriba (*bottom-up*):** Se comienza migrando los elementos de IU más pequeños, y de forma progresiva se va construyendo la pantalla completa.
- **De arriba hacia abajo (*top-down*):** Se comienza migrando los elementos de IU que contienen a todos los demás, y de forma progresiva se van migrando todos los elementos contenidos.

Para este caso se ha escogido la estrategia denominada ***bottom-up***, comenzando con los elementos más pequeños para más tarde juntarlos y con ello tener definida la interfaz en su totalidad. Es por ello que en primer lugar se definirá la función composable `SpacePathLine`, la cual es la más pequeña de todas las existentes en el fragmento.

```
1     @Composable
2     fun SpacePathLine(space: OCSpace, parentPath: String) {
3         val spaceName: String
4         val spaceImage: Int
5
6         if (space.isPersonal) {
7             spaceName = stringResource(id = R.string.
                  bottom_nav_personal)
8             spaceImage = R.drawable.ic_folder
9         } else {
10            spaceName = space.name
11            spaceImage = R.drawable.ic_spaces
12        }
13
14
15        Row(...)
16        {
17            Icon(
18                ...
19                painter = painterResource(id = spaceImage),
20                ...
21            )
22            Text(
23                ...
24                text = spaceName
25            )
26            Text(
27                ...
28                text = parentPath
29            )
30        }
31    }
```

Fragmento de código 7.3: Función *composable* `SpacePathLine`

En el Fragmento de código 7.3 se encuentra reflejada la estructura y el comportamiento de la función *composable* `SpacePathLine`. Esta función recibe dos parámetros diferentes con los que se trabajará a lo largo de toda la lógica implementada. En primer lugar se recibe el `space` al que se encuentra asociada una subida en concreto (la que se está pintando en la lista en ese momento) junto con la variable `parentPath`, la cual indica la ruta en la que se guardará cuando el archivo se suba exitosamente. Antes de comenzar con el resto de las funciones composables se realiza una comprobación del `space`. En caso de que sea el `personal space`, el nombre que se va a mostrar por pantalla viene determinado por una cadena de texto que está definida en el fichero `strings.xml`, además de establecer la imagen de una carpeta como icono principal. El hecho de que esa variable se encuentre en el fichero `strings.xml` es debido a que es una cadena de texto traducible. Por otro lado, si ese `space` no es el personal entonces el nombre que se mostrará por pantalla será aquel que está asociado a la variable que ha llegado como parámetro (nombre no traducible) de la función *composable* `TransferItem`, además de establecer el icono predeterminado de los `spaces`. Una vez que se han hecho estas comprobaciones se comenzará con la estructuración de toda la interfaz.

En primer lugar se introduce una función *composable* `Row`, la cual servirá para agrupar todos los elementos en forma de fila. Esta función tiene varios parámetros relacionados con la posición de los elementos en el interior, haciendo que se complete el ancho de la pantalla y se centren verticalmente. Dentro de esa función se tiene una función composable `Icon`. Esta función tiene una serie de parámetros que determinarán el aspecto del icono como, por ejemplo, el color, la descripción o incluso la imagen que se ha escogido previamente. Tras esa función se tiene dos composables de tipo `Text` con una estructura bastante similar entre sí. Se determina el color del texto, *overflow* (omitir partes de texto en caso de que sea demasiado grande y no se pueda ver correctamente en el espacio disponible), número de líneas, tamaño de la fuente y el texto. Este último parámetro viene determinado por las dos variables `spaceName` y `parentPath`. El resultado de esta función se encuentra reflejado en la Figura 7.1, donde se puede ver claramente el icono, el nombre del `space` y la ruta en la que se encuentra un determinado archivo.

Figura 7.1: Elemento de IU asociado a la función *composable* `SpacePathLine`

El siguiente paso que hay que realizar es crear cada uno de los elementos que conformarán la lista de todas las subidas. La función *composable* encargada de mostrar por pantalla este elemento de IU recibe el nombre de `TransferItem`. La explicación de esta función *composable* seguirá el orden que se encuentra reflejado en el Fragmento de código 7.4.

```
1       @Composable
2       fun TransferItem(transfer: OCTransfer, space: OCSpace?, workInfo
            : List<WorkInfo>) {
3           val remoteFile = File(transfer.remotePath)
4           var fileName = remoteFile.name
5           if (fileName.isEmpty()) {
6               fileName = File.separator
7           }
8
```

```
9           var path = ""
10          remoteFile.parent?.let {
11              path = if (it.endsWith("${OCFile.PATH_SEPARATOR}")) {
12                  it
13              } else {
14                  "$it${OCFile.PATH_SEPARATOR}"
15              }
16          }
17
18          Row(...)
19          {
20              Box(...)
21              {
22                  Image(...)
23              }
24              Column(...)
25              {
26                  Text(
27                      ...
28                      text = fileName
29                  )
30                  Row {
31                      Text(...)
32                      if (transfer.status != TransferStatus.
                            TRANSFER_FAILED) {
33                          ...
34                          Text(...)
35                          }
36                      }
37                      if (transfer.status != TransferStatus.
                            TRANSFER_SUCCEEDED) {
38                          Text(...)
39                      }
40                  }
41
42                  if (transfer.status == TransferStatus.
                        TRANSFER_IN_PROGRESS) {
43                      LinearProgressIndicator(
44                          ...
45                          progress = checkProgress(transfer, workInfo)
                                / 100f
46                      )
47                  }
48
49                  Text(
50                      ...
51                      text = checkAccount(transfer)
52                  )
53                  if (space != null) {
54                      SpacePathLine(space = space, parentPath = path)
55                  }
```

```
56                }
57                Box(...)
58                {
59                    if (transfer.status != TransferStatus.
                          TRANSFER_SUCCEEDED) {
60                        ...
61                        IconButton(onClick = { transfersViewModel.
                              cancelUpload(transfer) }) {
62                            Icon(...)
63                        }
64                    }
65                }
66            }
67            Divider(...)
68        }
```

Fragmento de código 7.4: Función *composable* `TransferItem`

A esta función `TransferItem` llegan, como parámetros, el objeto `OCTransfer`, que será necesario para obtener toda la información y pintarla por pantalla, el objeto de tipo `OCSpace`, que muestra el `space` en el que se encuentra ubicado el archivo, y, por último, la variable `workInfo`, en la cual viene toda la información asociada al progreso de subida de todos los archivos. Comenzando con la implementación de la función, en primer lugar se crea un objeto de tipo `File` a partir del atributo `remotePath` de la variable `transfer`. Tras ello, se obtiene el nombre del archivo que se ha subido, guardándolo en una variable con el nombre `fileName`. Si ese nombre está vacío, se pondrá la ruta del directorio raíz (/). El siguiente paso es determinar la ruta en la que se encuentra ese archivo para más tarde enviárselo a la función *composable* `SpacePathLine`. El valor de la variable `path` viene determinado por el atributo `parent` del objeto `remoteFile`. Dependiendo de su valor, se construirá la cadena, de una u otra forma, con el fin de mostrarlo correctamente en todo momento.

Una vez que se ha determinado el valor de esas dos variables, `fileName` y `path`, se procederá a la creación de la interfaz de usuario con el resto de funciones *composables*. En primer lugar, se tiene una función `Row` encargada de distribuir al resto de elementos en forma de fila. Una de las características que tiene esta función, respecto al resto que aparecen en todo el fragment, es que en caso de que el estado de la subida sea *failed*, se activa la opción *clickable*. Esto permite al usuario interactuar con todo el elemento, independientemente de donde toque, pudiendo reanudar la subida del archivo. De forma anidada, se encuentra otra función de tipo `Box`, la cual será la encargada de centrar la miniatura del archivo. Esta miniatura se consigue reproducir por pantalla a través de la función *composable* `Image`, la cual tiene una serie de atributos que definirán su apariencia. Saliendo del alcance de la función `Box`, se encuentra una función `Column` organizando los elementos de interfaz de usuario que tenga anidados en forma de columna.

Como se puede apreciar, todos los elementos del *composable* se distribuyen tanto en columna como en fila según se vaya requiriendo. Lo siguiente que hay que mostrar en la pantalla es el nombre del archivo, almacenado en la variable `fileName` ya comentada anteriormente. El resto de información que hay que mostrar está relacionada con el tamaño del archivo, el tiempo que ha transcurrido desde que se ha subido hasta la actualidad, la cuenta a la que se

encuentra asociada y el `space` al que se encuentra asociado. Esto último se corresponde con la función *composable* `SpacePathLine`, además de depender de la versión que esté utilizando el usuario (*oCIS* / *oC10*). Además, en caso de que la subida del fichero se encuentre en progreso se mostrará una línea de progreso, permitiendo al usuario conocer el estado actual de la misma y cuánto falta aproximadamente para que el fichero se termine de transferir. Este progreso viene determinado por la información almacenada dentro de la variable `workInfo`. Para conseguir este proceso se ha creado una función auxiliar denominada `checkProgress`. Junto a ese indicador se muestra un icono de cancelación, permitiendo al usuario interrumpir el proceso en cualquier momento. Esta funcionalidad sólo está disponible para aquellas subidas que no se hayan guardado de forma satisfactoria en el servidor. Por otro lado, para conseguir la información de la cuenta a la que se encuentra asociada esa subida, se ha creado una función auxiliar denominada `checkAccount`.

Para concluir con la función `TransferItem`, se tiene una función composable `Divider` encargada de mostrar por pantalla un separador entre los diferentes elementos. La apariencia de este divisor viene determinado por todos los parámetros de la función, como por ejemplo el color, el grosor o la longitud del mismo. El resultado de la función *composable* `TransferItem` en su totalidad y en los diferentes estados que existen, se puede ver reflejado en la Figura 7.2, Figura 7.3, Figura 7.4, y Figura 7.5.

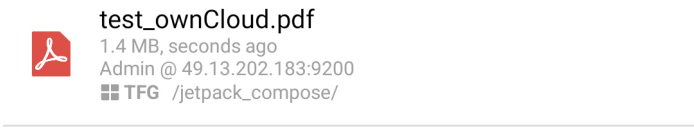

Figura 7.2: Resultado de la función *composable* `TransferItem` en el estado *Succeded*

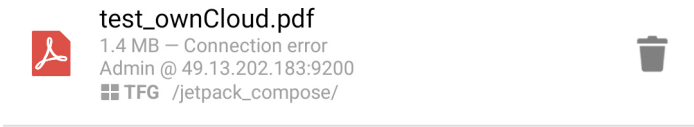

Figura 7.3: Resultado de la función *composable* `TransferItem` en el estado *Failed*

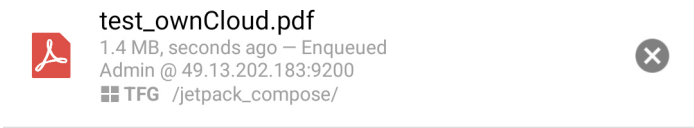

Figura 7.4: Resultado de la función *composable* `TransferItem` en el estado *Enqueued*

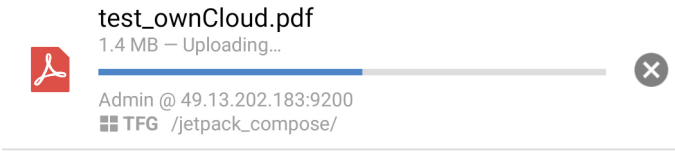

Figura 7.5: Resultado de la función *composable* `TransferItem` en el estado *In Progress*

Siguiendo con la estrategia, el próximo elemento que hay que crear es la cabecera que agrupará todas aquellas subidas que presentan el mismo estado. Esta cabecera se encuentra definida en otra función *composable*, la cual recibe el nombre de `UploadGroup` y cuya implementación se puede ver en el Fragmento de código 7.5.

```
1   @Composable
2   fun UploadGroup(status: TransferStatus, count: Int) {
3       val stringResFileCount =
4           if (count == 1) R.string.
                uploads_view_group_file_count_single else R.string.
                uploads_view_group_file_count
5       val fileCountText: String = String.format(requireContext().
            getString(stringResFileCount), count)
6
7       Column {
8           Row(...) {
9               Text(
10                  ...
11                  text = requireContext().getString(
                        headerTitleStringRes(status)).uppercase()
12              )
13              Spacer(...)
14              Text(
15                  ...
16                  text = fileCountText.uppercase()
17              )
18          }
19          Divider(...)
20          Row {
21              if (status == TransferStatus.TRANSFER_FAILED ||
                    status == TransferStatus.TRANSFER_SUCCEEDED) {
22                  Button(...) {
23                      Text(...)
24                  }
25              }
26              if (status == TransferStatus.TRANSFER_FAILED) {
27                  Button(...)
28                  {
29                      Text(...)
30                  }
31              }
```

```
32
33                  }
34              }
35              if (status == TransferStatus.TRANSFER_FAILED || status ==
                    TransferStatus.TRANSFER_SUCCEEDED) {
36                  Divider(...)
37              }
38          }
```

Fragmento de código 7.5: Función *composable* `UploadGroup`

En primer lugar se reciben dos únicos parámetros, los cuales son: el estado (`status`) de todas las subidas que se han agrupado y (`count`) el número de elementos que se encuentran en ese determinado estado. Si el número de ficheros solamente es uno, entonces se establecerá la cadena de texto en singular. En caso contrario, se creará la cadena de texto con el número de elementos, además de mostrar el texto `Files` en plural. Una vez que se han hecho estas comprobaciones se comenzará con la creación de la interfaz de usuario. Al igual que en las otras funciones composables, la distribución de todos los elementos se ha determinado a partir de las funciones `Column` y `Row`. Dentro de estas funciones es donde se definen el resto de elementos que conforman la interfaz en su totalidad. En primer lugar se tiene un `Text`, cuyo valor está determinado por el estado en el que se encuentre el grupo de las subidas a mostrar. Esta cadena de texto viene dada a través de una función auxiliar con el nombre `headerTitleStringRes`, pasándole por parámetro la variable `status`. Tras ello, se mostrarán los botones dependiendo del estado en el que se encuentre. Si la cabecera pertenece a todos aquellos archivos que se encuentran en estado *failed* o *succeeded* entonces se mostrará un botón que permite borrar los elementos en la base de datos local, y por este motivo se actualiza la IU. Por otro lado si la cabecera corresponde únicamente a los archivos que se encuentran en estado *failed* entonces se mostrará un botón que implementa la funcionalidad para reintentarlo, pero para todas las subidas fallidas. Esa es la principal diferencia que existe entre este botón y la funcionalidad implementada a la hora de pulsar sobre una celda individual. En los otros dos casos (*enqueued* o *in progress*) no se mostrará ningún botón dentro de la cabecera. El resultado de todo ello se puede apreciar en la Figura 7.6, Figura 7.7, Figura 7.8, y Figura 7.9.

Figura 7.6: Resultado de la función *composable* `UploadGroup` en el estado *Succeded*

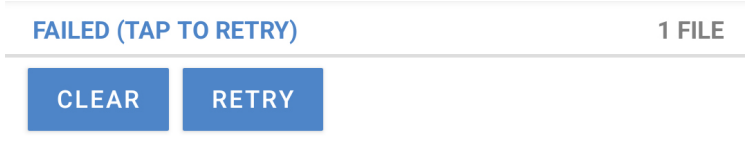

Figura 7.7: Resultado la función *composable* `UploadGroup` en el estado *Failed*

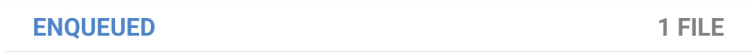

Figura 7.8: Resultado la función *composable* `UploadGroup` en el estado *Enqueued*

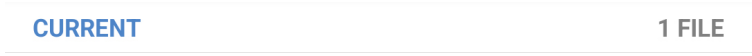

Figura 7.9: Resultado la función *composable* `UploadGroup` en el estado *In Progress*

El siguiente paso es juntar los dos tipos de composables de forma dinámica, de modo que se muestre por pantalla las subidas (`TransferItem`) agrupadas por estado junto a la cabecera (`UploadGroup`). Todo este funcionamiento se encuentra dentro de la función *composable* `TransferList`, la cual se encuentra desarrollada en el Fragmento de código 7.6.

```
1  @Composable
2  fun TransferList(transfersWithSpace: List<Pair<OCTransfer,
       OCSpace?>>, workInfo: List<WorkInfo>) {
3      val transfersGroupedByStatus = transfersWithSpace.groupBy {
           it.first.status }
4      LazyColumn {
5          transfersGroupedByStatus.forEach { (status, transfers)
               ->
6              item {
7                  UploadGroup(status = status, count = transfers.
                       size)
8              }
9              transfers.sortedByDescending { it.first.
                   transferEndTimestamp ?: it.first.id }
10                 .forEach { (transfer, space) ->
11                     item {
12                         TransferItem(transfer = transfer, space
                               = space, workInfo = workInfo)
13                     }
14                 }
15         }
16     }
17 }
```

Fragmento de código 7.6: Función *composable* `TransferList`

En primer lugar, se ordenan todas las subidas junto a sus `spaces`, según el estado en el que se encuentren. Tras ello, se mostrará la cabecera que englobará a todos los elementos que tengan ese estado. Por cada cabecera se recorrerá la lista de todas las subidas, y en caso de que el estado sea el mismo se mostrarán en forma de lista. El hecho de que se muestre en una lista en la que se puede desplazar (siempre y cuando haya un número de elementos como para que no quepan todos en la misma pantalla) es una funcionalidad interna y que se gestiona de forma automática a través de la función *composable* `LazyColumn`. El resultado de esta función se puede apreciar en la Figura 7.10 y en la Figura 7.11.

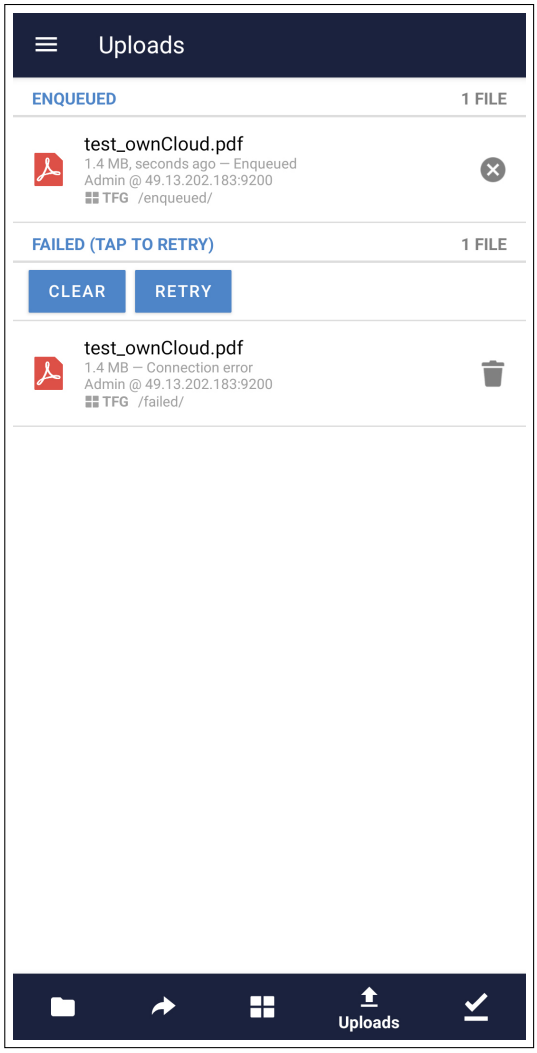

Figura 7.10: Resultado de la función *composable* `TransferList` con los estados *Enqueued* y *Failed*

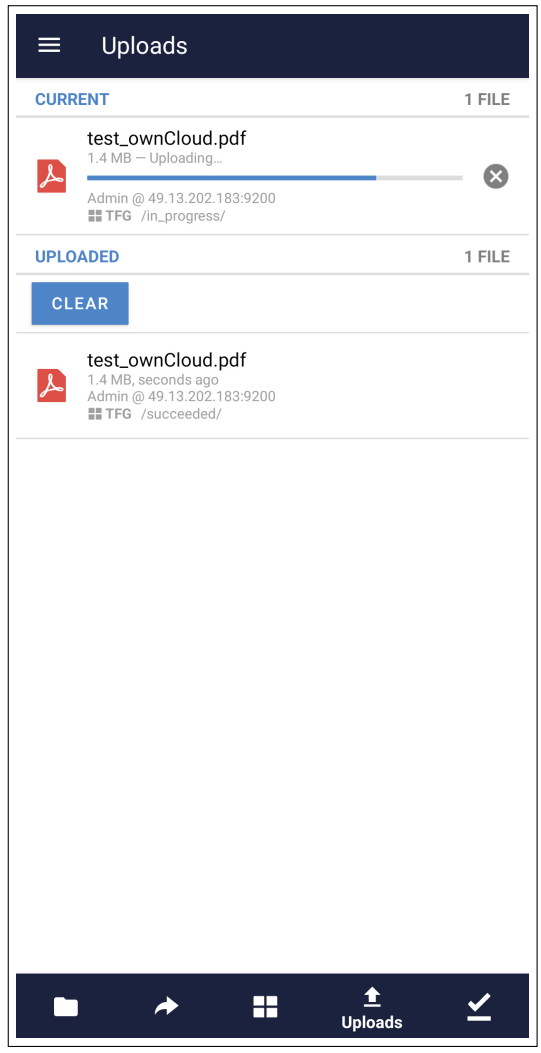

Figura 7.11: Resultado de la función *composable* `TransferList` con los estados *In progress* y *Succeeded*

A pesar de tener un resultado consistente en el que se muestren todas las subidas en sus respectivos estados, existe la posibilidad que no haya ninguna subida por parte del servidor, o incluso cambie el estado de alguna de ellas debido a que, por ejemplo, ha habido una pérdida de red. Por este motivo, la información que debe mostrar cada *composable* viene determinada por el comportamiento de las subidas, todo ello implementado en la función *composable* `Transfers` correspondiente al Fragmento de código 7.7.

```
1       @Composable
2       fun Transfers() {
3           var transfersState by remember { mutableStateOf(emptyList<
                Pair<OCTransfer, OCSpace?>>()) }
4           val workInfos by transfersViewModel.workInfosListLiveData.
                asFlow().collectAsState(initial = emptyList())
5
6           LaunchedEffect(transfersViewModel) {
7               transfersViewModel.transfersWithSpaceStateFlow.
                    collectLatest { transfers ->
8                   transfersState = transfers
9               }
10          }
11
12          if (transfersState.isEmpty()) {
13              EmptyList()
14          } else {
15              TransferList(transfersWithSpace = transfersState,
                    workInfo = workInfos)
16          }
17      }
```

Fragmento de código 7.7: Función *composable* `Transfers`

Dentro de esta función existen dos variables, cuyo valor está en constante cambio. En primer lugar se tiene la variable `transferState`. Esta variable almacena el valor de todas las subidas junto con el `space` al que se encuentren asociado, y el cual puede cambiar a lo largo del tiempo por ciertos motivos. Por ello, se ha utilizado la palabra reservada `remember` a la hora de definir la variable. Esto sirve para almacenar el valor de una variable de manera que su estado persista a través de la recomposición. La recomposición es el proceso mediante el cual Jetpack Compose vuelve a renderizar una parte de la interfaz de usuario cuando se detecta un cambio de estado que afecta a esa parte [39]. Si no se añadiese la palabra `remember` entonces la interfaz siempre se construiría con el mismo valor y por tanto no habría un cambio apreciable para el usuario, a pesar de que el estado de las subidas se actualiza en el lado del servidor. El cambio de valor de esta variable viene dentro del bloque `LaunchedEffect`, en el que se observa constantemente la variable `transfersWithSpaceStateFlow` perteneciente a la clase `TransfersViewModel`. Cada vez que se produzca un cambio en esta variable, influirá en el valor de la variable `transferState` y, por tanto, Jetpack Compose se encargará de volver a construir la interfaz de usuario recomponiendo únicamente aquellos *composables* que hayan sido afectados por esos cambios. Por otro lado, está la variable `workInfos` cuyo funcionamiento es muy similar al anterior. La única diferencia que existe es que no se utiliza la palabra reservada `remember` para guardar el estado anterior. En este caso se trata la variable como si fuese de tipo `Flow` en el que se está escuchando constantemente si existe algún cambio. Esto se hace a través de la directiva `collectAsState`. Una vez que se tiene toda la información actualizada y lista para mostrarse por pantalla, se realiza una comprobación para ver si la lista que se ha conseguido por parte del servidor no se encuentra vacía. En caso de que sí lo esté, se llamará a la función *composable* denominada `EmptyList`. En caso contrario, se llama a la función composable `TransferList` con las dos variables que se han

comentado (`transferState` y `workInfos`).

```
@Composable
fun EmptyList() {
    Column(...) {
        Icon(
            ...
            painter = painterResource(id = R.drawable.ic_uploads
                ),
            ...
        )
        Text(
            ...
            text = stringResource(id = R.string.
                upload_list_empty)
        )
        Text(
            ...
            text = stringResource(id = R.string.
                upload_list_empty_subtitle)
        )
    }

}
```

Fragmento de código 7.8: Función *composable* `EmptyList`

En el Fragmento de código 7.8 se puede ver con claridad la estructura simple que sigue el *composable* `EmptyList`. Como siempre, se tiene una función para distribuir todos los elementos según un patrón determinado. En este caso se ha utilizado la función `Column` para agrupar los elementos uno encima de otro. Estos elementos son: un *composable* de tipo `Icon` y dos *composables* de tipo `Text`. Aparte de los elementos que conforman la interfaz, no existe una lógica adicional asociada a esta función ya que únicamente se precisa su uso de manera estática. El resultado de esta función *composable* se puede ver en la Figura 7.12.

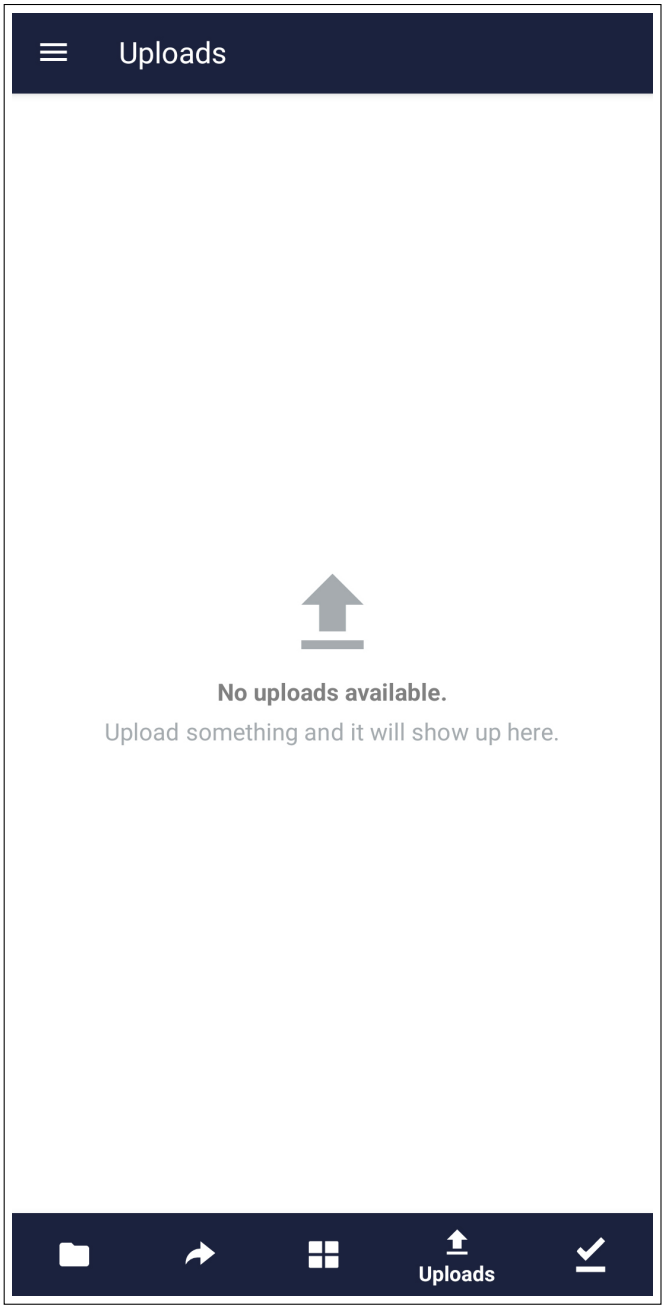

Figura 7.12: Resultado de la función *composable* `EmptyList`

7.1.2. Resultado

El resultado que se ha obtenido es la pantalla *"Transfers"* de la aplicación Android de *ownCloud*, la cual ha sido migrada de un paradigma imperativo con la interfaz de usuario definida en XML y código Kotlin a un paradigma declarativo utilizando exclusivamente código Kotlin y el kit de herramientas de IU de Jetpack Compose. En la Figura 7.13 y Figura 7.14 se puede ver una comparativa entre la pantalla original y la pantalla resultante para un mismo estado. En esa comparación se puede apreciar como ambas pantallas son relativamente idénticas, diferenciándose por alguna característica de tamaño, espacio o distribución de algún elemento. Con ello se puede concluir que se puede realizar una migración a Jetpack Compose sobre una pantalla sin influir en su aspecto.

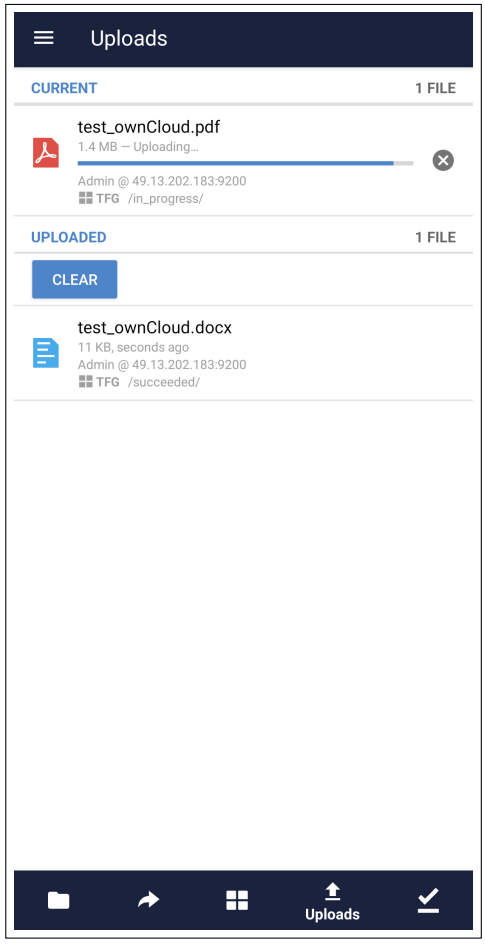

Figura 7.13: Pantalla de las subidas definida en XML

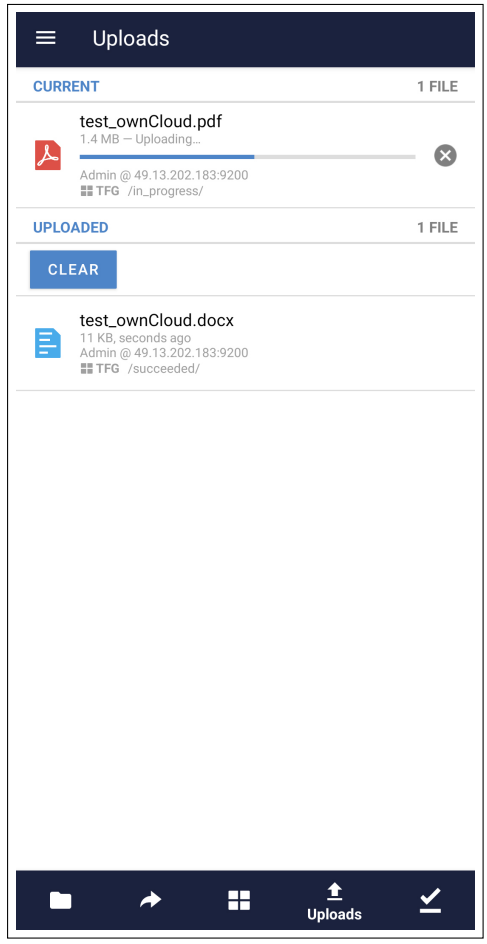

Figura 7.14: Pantalla de las subidas definida en Jetpack Compose

7.2. Pruebas

Tras terminar con la implementación, el código de la pantalla asociada al caso de uso en cuestión ha sido sometido a dos rondas de revisión a manos de un ingeniero experto en Android y en la aplicación de *ownCloud*, sugiriendo algún cambio. Tras terminar con esa revisión, se han llevado a cabo una serie de pruebas para comprobar que todo funcionaba a la perfección. Toda esta serie de pruebas recibe el nombre de plan de pruebas, el cual ha sido creado por un ingeniero QA (*Quality Assurance*) perteneciente al equipo Android de *ownCloud* de Izertis. Algo a destacar, es que a pesar de que el trabajo principal se ha centrado en la migración de la interfaz de usuario, se ha separado todo el plan de pruebas en varias partes. En primer lugar se ha probado cómo se veía la interfaz por cada estado disponible (*In progress, failed, succeeded y enqueued*). Tras ello se ha comprobado si el flujo era correcto para un sólo archivo. Finalmente se han hecho pruebas con varios ficheros de forma simultánea con estados diferentes.

En todas las pruebas que se han hecho relativas al estado *failed* solamente se ha considera el error *Connection error*. A pesar de que hay más errores que pueden reproducirse, e influir en el estado de la subida, el cambio en la interfaz de usuario es el mismo y, por tanto, no surgirían nuevos problemas respecto a las pruebas. En las Tablas 7.1, 7.2, 7.3, 7.4, 7.5, 7.6 y 7.7 se pueden apreciar todas las pruebas que se han llevado a cabo separados por estados y flujos.

Título	***Pasos a seguir***	***Resultado esperado***	***Resultado***	***Comentarios***
Lista vacía	1. Instalar la aplicación y añadir una cuenta 2. Abrir la sección de *Uploads* dando click en el icono de la flecha vertical situada en la barra inferior	*No uploads available*	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)

Tabla 7.1: Plan de pruebas referente a la lista vacía

Título	*Pasos a seguir*	*Resultado esperado*	*Resultado*	*Comentarios*
Subir 1 archivo en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Uploaded* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha sido subido - *Personal* como `space` - / como carpeta El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Uploaded* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - *Personal* como `space` - / como carpeta El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro de una carpeta del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Uploaded* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha subido - *Personal* como `space` - Carpeta destino de la subida El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro de una carpeta del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Uploaded* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - *Personal* como `space` - Carpeta destino de la subida El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Uploaded* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha subido - Nombre del `space` compartido - / como carpeta El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Uploaded* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - Nombre del `space` compartido - / como carpeta El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Uploaded* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha subido - Nombre del `space` compartido - Carpeta destino de la subida El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Uploaded* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - Nombre del `space` compartido - Carpeta destino de la subida El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Borrar lista	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el botón *Clear*	3. Sección con el nombre *Uploaded* y *5 Files* 4. Lista y sección *Uploaded* eliminadas	Correcto	

Tabla 7.2: Plan de pruebas referente al estado de subida: *Succeeded*

Título	*Pasos a seguir*	*Resultado esperado*	*Resultado*	*Comentarios*
Subir 1 archivo en el root del Personal Space	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Enqueued* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha sido subido - *Personal* como `space` - / como carpeta - Botón *X* visible El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Enqueued* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - *Personal* como `space` - / como carpeta - Botón *X* visible para cada archivo El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro de una carpeta del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Enqueued* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha subido - *Personal* como `space` - Carpeta destino de la subida - Botón *X* visible El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro del root del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Enqueued* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - *Personal* como `space` - Carpeta destino de la subida - Botón *X* visible para cada archivo El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Enqueued* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha subido - Nombre del `space` compartido - / como carpeta - Botón *X* visible El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Enqueued* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - Nombre del `space` compartido - / como carpeta - Botón *X* visible para cada archivo El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica de un space compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Enqueued* y *1 File* El archivo que se ha subido debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se ha subido - Nombre del `space` compartido - Carpeta destino de la subida - Botón *X* visible El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)

(Continúa en la siguiente página)

(Empieza en la página anterior)

Título	***Pasos a seguir***	***Resultado esperado***	***Resultado***	***Comentarios***
Subir 10 archivos en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Enqueued* y *10 Files* Todos los archivos que se han subido deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y fecha de la subida - Cuenta en la que se han subido - Nombre del `space` compartido - Carpeta destino de la subida - Botón *X* para cada archivo El botón *Clear* debe estar visible	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Eliminar algunos archivos que están en cola	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el botón *X* de dos archivos 5. Recuperar la conexión del dispositivo	3. Sección con el nombre *Enqueued* y *5 Files* 4. 3 archivos restantes en la sección *Enqueued* 5. Los archivos restante pasan a la sección *Current* y tras ello a la sección *Uploaded*	Correcto	
Eliminar todos los archivos que están en cola	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el botón *X* de todos los archivos	3. Sección con el nombre *Enqueued* y *5 Files* 4. La sección *Enqueued* desaparece por completo	Correcto	

Tabla 7.3: Plan de pruebas referente al estado de subida: *Enqueued*

Título	*Pasos a seguir*	*Resultado esperado*	*Resultado*	*Comentarios*
Subir 1 archivo grande en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Current* y *1 File* mientras el archivo se está subiendo. El archivo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - *Personal* como `space` - / como carpeta - Botón *X* visible - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Current* y *10 Files*. El número va decrementando a medida que los archivos se van subiendo correctamente. Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - *Personal* como `space` - / como carpeta - Botón *X* visible para cada archivo - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro de una carpeta del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Current* y *1 File* mientras el archivo se está subiendo. El archivo que se está subiendo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - *Personal* como `space` - Carpeta destino de la subida - Botón *X* visible - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Current* y *10 Files*. El número va decrementado a medida que los archivos se van subiendo correctamente. Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - *Personal* como `space` - Carpeta destino de la subida - Botón *X* visible para cada archivo - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Current* y *1 File* mientras el archivo se está subiendo. El archivo que se está subiendo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - Nombre del `space` compartido - / como carpeta - Botón *X* visible - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)

(Continúa en la siguiente página)

(Empieza en la página anterior)

Título	***Pasos a seguir***	***Resultado esperado***	***Resultado***	***Comentarios***
Subir 10 archivos en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Current* y *10 Files*. El número va decrementando a medida que los archivos se van subiendo correctamente Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - Nombre del `space` compartido - / como carpeta - Botón *X* visible para cada archivo - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del space compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Current* y *1 File* mientras el archivo se está subiendo. El archivo que se está subiendo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se está subiendo - Nombre del `space` compartido - Carpeta destino de la subida - Botón *X* visible - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 ficheros en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Current* y *10 Files*. El número va decrementando a medida que los archivos se van subiendo correctamente. Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Uploading...* - Cuenta en la que se han subido - Nombre del `space` compartido - Carpeta destino de la subida - Botón *X* para cada archivo - Barra de progreso	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Eliminar algunos archivos en progreso	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el botón *X* de dos archivos antes de que finalicen	3. Sección con el nombre *Current* y *5 Files* 4. 3 archivos restantes en la sección *Current*. Cuando terminan pasan a la sección *Uploaded* y la sección *Current* desaparece.	Correcto	
Eliminar todos los archivos que están en progreso	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el botón *X* de todos los archivos	3. Sección con el nombre *Current* y *5 Files* 4. La sección *Current* desaparece por completo	Correcto	

Tabla 7.4: Plan de pruebas referente al estado de subida: *In progress*

Título	***Pasos a seguir***	***Resultado esperado***	***Resultado***	***Comentarios***
Subir 1 archivo en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Failed* y *1 File*. El archivo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - *Personal* como `space` - / como carpeta - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Failed* y *10 Files*. Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - *Personal* como `space` - / como carpeta - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro de una carpeta del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Failed* y *1 File*. El archivo que se está subiendo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - *Personal* como `space` - Carpeta destino de la subida - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)

(Continúa en la siguiente página)

(Empieza en la página anterior)

Título	***Pasos a seguir***	***Resultado esperado***	***Resultado***	***Comentarios***
Subir 10 archivos en una ruta específica del *Personal Space*	1. Abrir el botón flotante dentro del *root* del *Personal Space* 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Failed* y *10 Files.* Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - *Personal* como `space` - Carpeta destino de la subida - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Failed* y *1 File* El archivo que se está subiendo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - Nombre del `space` compartido - / como carpeta - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 archivos en el *root* de un `space` compartido	1. Abrir el botón flotante dentro del *root* del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Failed* y *10 Files.* Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - Nombre del `space` compartido - / como carpeta - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 1 archivo en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar cualquier fichero del sistema	Sección con el nombre *Failed* y *1 File.* El archivo que se está subiendo debe cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se está subiendo - Nombre del `space` compartido - Carpeta destino de la subida - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Subir 10 ficheros en una ruta específica de un `space` compartido	1. Abrir el botón flotante dentro de una carpeta del `space` compartido 2. Hacer click en *Upload* y seleccionar *Files* 3. Seleccionar 10 ficheros del sistema	Sección con el nombre *Failed* y *10 Files.* Todos los archivos que se están subiendo deben cumplir las siguientes características: - *Thumbnail* o icono del tipo de fichero - Nombre del fichero - Tamaño y *Connection error* - Cuenta en la que se han subido - Nombre del `space` compartido - Carpeta destino de la subida - Icono de la papelera visible Los botones *Clear* y *Retry* deben estar visibles	Correcto	Comprobado en las dos orientaciones posibles (vertical y horizontal)
Eliminar algunos archivos fallido. Recuperar el proceso con el botón *Retry*	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el icono de la papelera en 2 archivos fallidos 5. Levantar el servidor 6. Hacer click en el botón *Retry*	3. Sección con el nombre *Failed* y *5 Files* 4. 3 archivos restantes en la sección *Failed.* 6. 3 archivos en la sección *Current.* La sección *Failed* desaparece	Correcto	
Eliminar algunos archivos fallido. Recuperar el proceso dando click al elemento.	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el icono de la papelera en 2 archivos fallidos 5. Levantar el servidor 6. Hacer click sobre cada uno de los elementos fallidos	3. Sección con el nombre *Failed* y *5 Files* 4. 3 archivos restantes en la sección *Failed.* 6. 3 archivos en la sección *Current.* La sección *Failed* desaparece	Correcto	
Eliminar todos los archivos fallidos.	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el icono de la papelera en todos los archivos fallidos	3. Sección con el nombre *Failed* y *5 Files* 4. La sección *Failed* desparece	Correcto	
Limpiar la lista	1. En cualquier carpeta, abrir el botón flotante 2. Dar click en *Upload* y seleccionar *Files* 3. Seleccionar 5 archivos del sistema 4. Hacer click en el botón *Clear*	3. Sección con el nombre *Failed* y *5 Files* 4. Se vacía la lista de los elementos y la sección Failed desaparece		

Tabla 7.5: Plan de pruebas referente al estado de subida: *Failed*

´Título	Pasos a seguir	Resultado esperado	Resultado	Comentarios
Flujo completo	1. Abrir el botón flotante dentro de una ruta cualquiera 2. Hacer click en *Upload* y seleccionar *Files* 3. Quitar la conexión del dispositivo 4. Seleccionar un archivo grande del sistema 5. Activar la conexión del dispositivo 6. Apagar el servidor 7. Hacer click en el botón *Retry* 8. Levantar el servidor 9. Hacer click en el botón *Retry* 10. Esperar que termine la subida	4. Archivo en la sección *Enqueued* 5. Archivo en la sección *Current* 6. Archivo en la sección *Failed* 7. Archivo en la sección *Failed* 8. Archivo en la sección *Failed* 9. Archivo en la sección *Current* 10. Archivo en la sección *Uploaded*	Correcto	

Tabla 7.6: Plan de pruebas referente al flujo de subida de un sólo archivo en todos los estados posibles

´Título	Pasos a seguir	Resultado esperado	Resultado	Comentarios
Archivos en todos los estados posibles	1. Terminar alguna subida 2. Apagar el servidor y añadir más archivos para subir 3. Levantar el servidor 4. Apagar la conexión del dispositivo y seleccionar un archivo grande para subir 5. Seleccionar más archivos para subir 6. Recuperar la conexión del dispositivo	4. Archivos en las secciones *Uploaded*, *Failed* y *Enqueued* 5. El archivo grande comienza a subirse y el resto de archivos en las secciones *Uplodaded*, *Failed* y *Current*	Correcto	

Tabla 7.7: Plan de pruebas con todos los estados posibles simultáneamente

7.3. Conclusiones de la migración

Con la migración de XML a Jetpack Compose completada se ha llegado a la conclusión de que esta última tecnología presenta varias ventajas respecto a la que se venía utilizando anteriormente. En la *Guía de migración de XML a a Jetpack Compose* [23] se exponen cuatro hipotéticas ventajas respecto a la migración, las cuales son: **menos código, más intuitivo, acelera el desarrollo y más potente.**

7.3.1. Menos código

Si se tiene en cuenta el sistema de vista XML de la pantalla asociada al caso de uso sobre el que se ha trabajado, se tiene un total de **5 clases diferentes** con el siguiente número de líneas de código (LLOC):

- `TransferListFragment.kt`: 133 LLOC
- `TransferAdapter.kt`: 304 LLOC

- `fragment_transfer_list.xml`: 24 LLOC
- `upload_list_item.xml`: 125 LLOC
- `item_empty_dataset.xml`: 54 LLOC

Sumando el total de líneas de código entre todas las clases que participan en la interfaz de usuario relacionada con el caso de uso, se tiene un total de **640 LLOC**. Por el contrario, tras haber migrado la interfaz de usuario a Jetpack Compose se tiene **una única clase** denominada `TransferListComposeFragment.kt` la cual presenta un total de **499 LLOC**. De esta manera se puede apreciar como no sólo se ha reducido el número de clases que intervienen para mostrar la interfaz de usuario sino que también se ha reducido el número de líneas de código (un 23 % aproximadamente). A partir de estos datos se puede concluir que Jetpack Compose **necesita menos líneas de código** que XML para definir la misma pantalla de la aplicación.

7.3.2. Más intuitivo

Tal y como se ha expresado en el Apartado 7.3.1, se ha disminuido notablemente la cantidad de clases que intervienen en el caso de uso, hasta tal punto de tener una única clase encargada de todo. De esta forma, en caso de que sea necesario algún cambio o reestructuración de la lógica del caso de uso será mucho más fácil trabajar con el código en mismo lugar. Además, Jetpack Compose permite estructurar toda la interfaz de usuario de forma más clara, jugando con el anidamiento entre los diferentes elementos gráficos. Por estos motivos, se puede concluir que Jetpack Compose resulta ser **más intuitivo**.

7.3.3. Acelera el desarrollo

Debido a que la creación de la pantalla en XML asociada al caso de uso sobre el que se ha trabajado ha sido relativamente moderna, se tiene el tiempo total que se ha tardado para desarrollarla en su totalidad. Según un juicio de experto (no se tienen registros al respecto), dicha pantalla fue creada en aproximadamente **55 horas** por un desarrollador experto en Android y que conoce a la perfección la estructura de la aplicación. Como contraste, se ha medido el tiempo total que ha llevado la migración de esa misma pantalla, teniendo un total de **21 horas**, teniendo en cuenta que no se ha llevado a cabo por un desarrollador experto. Con todos estos datos se puede ver que el mismo trabajo se ha podido completar en un intervalo de tiempo mucho menor (reducido un 70 % aproximadamente), respaldando de esta manera que Jetpack Compose sí **acelera el desarrollo** de las interfaz de usuario en aplicaciones Android.

7.3.4. Más potente

Tras el desarrollo del caso de uso de estudio, se ha llegado a la conclusión de que la utilización de Kotlin de forma exclusiva en la definición de la interfaz de usuario permite

que todo se cree de forma más dinámica, aprovechando en todo momento las características que tiene el propio lenguaje. El ejemplo más claro de ello se puede apreciar a la hora de definir una lista de elementos indefinidos. En el caso de la vista creada a partir de XML es necesario definir el contenedor donde irán definidos todos esos elementos que conforman la lista. Para este caso de uso, la clase que corresponde con dicho contenedor recibe el nombre de `fragment_transfer_list.xml`. A su vez es necesario definir otro archivo en XML para cada uno de los elementos que se desean mostrar en forma de lista. Este archivo recibe el nombre de `upload_list_item.xml` y, además, se necesita una clase `Adapter`, que se encargue de toda la lógica para mostrar por pantalla de forma correcta todos los elementos, y un fragmento o actividad que lo contenga a su vez. Para este caso de uso las clases son `TransfersAdapter.kt` y `TransfersListFragment.kt` respectivamente. En Jetpack Compose todo es mucho más sencillo. Existe una función composable predefinida que recibe el nombre de `LazyColumn`, cuya lógica ya se encuentra implementada de forma automática. Tan sólo es necesario crear un bucle en el que se vaya creando cada uno de los elementos a representar, que son por igual funciones *composables*, tal y como se puede ver en el Fragmento de código 7.6.

Por otro lado, gracias a la flexibilidad que ofrece Kotlin, el código es muy modularizable. En este proyecto, se ha decidido crear cada una de las funciones *composables* dentro del mismo fichero que comprende el fragmento. Sin embargo, cada función podría haberse escrito en un fichero distinto, mejorando de esta manera la organización del código de la aplicación, además de permitir la reutilización de los elementos gráficos en otras pantallas con total libertad.

De forma adicional, Jetpack Compose aplica ciertos criterios de accesibilidad por defecto, ayudando a crear una interfaz de usuario con mayor calidad y evitando escribir código que de otra manera sería necesario. Algún ejemplo de ello sería, que existen ciertos elementos *composables* en los que es obligatorio definir una descripción del elemento (*contentDesription*). Por igual, los elementos de tipo `IconButton` reciben un tamaño mínimo de 48dp x 48dp, teniendo una zona lo suficientemente grande como para poder hacer *click* sobre el botón. En XML el tamaño hay que definirlo de forma manual teniendo en cuenta los criterios de accesibilidad.

Por último, se debe mencionar que se aplican por defecto los principios de *Material Design* [16] (sistema de diseño utilizado y recomendado por Google) proporcionando una interfaz de usuario usable, consistente y con una fácil personalización.

A partir de todos estos argumentos, se puede concluir que Jetpack Compose es **más potente** que la solución basada en XML para el desarrollo de interfaces en Android.

Capítulo 8

Conclusiones

Tras haber finalizado el proyecto, se puede concluir que éste ha sido costoso, debido a la cantidad de esfuerzo que se ha tenido que dedicar para sacarlo adelante. Además, durante los primeros sprints tampoco se tenía una idea clara de cómo iba a resultar el desarrollo del proyecto, por lo que el ritmo ha sido menor que en las etapas finales del trabajo.

Por otro lado también ha resultado gratificante ya que se han conseguido completar todos los objetivos que se habían establecido. Además, se han aprendido muchos conceptos que previamente se desconocían, sobre todo relacionados con la tecnología que se ha utilizado a la hora de la implementación: Jetpack Compose.

A nivel personal se han superado las expectativas sobre todo en la parte de la migración, ya que se ha conseguido completar el objetivo con un número de horas mucho menor del que se había estimado. Además, se ha mejorado el uso de herramientas que ya se habían visto en el grado como, por ejemplo, Astah. También se han aprendido conceptos que no se podían aplicar a nivel de una asignatura de grado como es la organización del trabajo (sprints, reuniones semanales, tablero en GitHub, etc...)

En cuanto a los objetivos que se habían establecido puede decirse que:

- Se ha conseguido realizar un estudio sobre dos casos de uso de la aplicación para luego llevar a cabo un proceso ingeniería inversa y poder documentar la estructura de la aplicación junto a las clases que intervienen en cada caso.
- Se ha conseguido comprender y aprender sobre la tecnología principal que se ha utilizado en la migración de la interfaz de pantalla: Jetpack Compose.
- Se ha conseguido realizar la migración de pantalla de XML a Jetpack Compose con una apariencia casi idéntica a la interfaz de usuario que se tenía previamente en la aplicación.

Algo a destacar es que, como consecuencia de todo el trabajo que se ha llevado a cabo, se han conseguido sacar unas conclusiones sobre la arquitectura limpia y que afectan de

forma directa a la aplicación Android de *ownCloud*. Tras realizar el análisis en uno de los dos casos de uso, se ha observado que el patrón que sigue esta arquitectura no se aplica en todos los casos y, por tanto, no se cumple al 100 % todos los principios que se estipulan en la clean architecture. Si no se hubiera hecho este trabajo de ingeniería inversa, estudio análisis y documentación sobre la aplicación, no se podría haber apreciado estas características que resultan pocos comunes en un arquitectura como esta. Por otro lado, todo el trabajo que se ha llevado a cabo sirve como documentación de la estructura de la aplicación, siendo necesario actualizarla en caso de que se produzcan cambios que impacten sobre la misma. Por último, toda la información que se refleja en este proyecto puede servir como complemento a la formación base de nuevos trabajadores que se incorporen en el equipo Android de *ownCloud*. Gracias a los diagramas se puede tener una visión mucho más clara de cómo está estructurada la aplicación y las relaciones que existen entre los diferentes módulos que la componen.

8.1. Líneas de trabajo futuras

Como trabajo futuro se plantean las siguiente propuestas que, a pesar de que estaban fuera del alcance del proyecto, resultan interesantes para la mejora de *ownCloud* de cara al usuario final.

- **Nueva interfaz de usuario para la pantalla de las subidas**. El rediseño del caso de uso que se ha tratado en la parte de la migración, no incluía la modificación y mejora de la interfaz de usuario y por tanto se ha tratado de crear la pantalla lo más parecida a lo que se tenía anteriormente. Dicha interfaz presentaba ciertos elementos inadecuados a nivel de aspecto o colocación, por lo que se podría realizar un rediseño de la interfaz de usuario completa para mejorar su aspecto de cara al usuario.

- **Mejor adaptación de la arquitectura limpia en algunos casos de uso**. Tal y como se puede ver en el Apartado 4.1.2, en algunos casos de uso existen ciertas clases que no se encuentran en su respectiva capa si se sigue la Figura 4.2. Por tanto, una de las mejoras a futuro sería conseguir que la arquitectura limpia se aplique 100 % a la aplicación, y no existan excepciones en algunos caso de uso.

- **Elaboración de un artículo sobre la arquitectura limpia en aplicaciones Android**. Después de haber llevado a cabo todo el proceso de ingeniería inversa se han conseguido afianzar todos los conceptos sobre la arquitectura limpia en aplicaciones Android. Es por ello que se podría preparar un artículo sobre este tema y publicarlo en la web, de forma que desarrolladores puedan tomar como referencia a la hora de realizar una rearquitectura de una aplicación o bien comenzar desde cero.

- **Subir al repositorio de *ownCloud* la documentación sobre la arquitectura**. A día de hoy, no se tiene una documentación sobre la arquitectura dentro del repositorio principal de la aplicación Android de *ownCloud*, por lo que se podría crear un documento a partir de la información de este trabajo y subirlo al repositorio. De esta forma se conseguiría que esté a manos de cualquiera que quiera aportar en el trabajo o incluso resultar útil para la formación de nuevos trabajadores.

- **Actualización de la documentación**. Incorporar en los procesos de desarrollo de *ownCloud* la actualización de la documentación de la arquitectura cuando se introduzcan cambios que impacten en la misma.

Bibliografía

[1] Android. Android. https://www.android.com/intl/es_es/. Accessed: 2024-05-01.

[2] Android Developers. Jetpack Compose UI App Development Toolkit - Android Developers. https://developer.android.com/jetpack/compose. Accessed: 2024-03-06.

[3] Apple. MacBook Pro 2023 (M3). https://www.apple.com/es/shop/buy-mac/macbook-pro/14-pulgadas-gris-espacial-chip-m3-de-apple-con-cpu-de-8-n%C3%BAcleos-y-gpu-de-10-n%C3%BAcleos-8gb-de-memoria-1tb. Accessed: 2024-03-09.

[4] Astah. Pricing Individual. https://astah.net/pricing/individual/. Accessed: 2024-03-09.

[5] Astah. Sofware Design Tool. Astah Professional. https://astah.net/products/astah-professional/. Accessed: 2024-04-13.

[6] Atlassian. Desarrollo de software. Comandos Git. https://www.atlassian.com/es/git/glossary#commands. Accessed: 2024-04-14.

[7] Campus Training. Alba Naveira. Sueldo de un desarrollador Android: todo lo que necesitas saber. https://www.campustraining.es/cursos-de-informatica/programacion-android/sueldo/. Accessed: 2024-03-09.

[8] Deloitte. Artefactos scrum: las 3 herramientas claves de gestión. https://www2.deloitte.com/es/es/pages/technology/articles/artefactos-scrum.html. Accessed: 2024-03-02.

[9] Deloitte. Scrum: roles y responsabilidades. https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html. Accessed: 2024-03-03.

[10] Developers. Android Studio. https://developer.android.com/studio. Accessed: 2024-04-104.

[11] Espai. Como trabajar con Ramas en Git y Github. https://www.espai.es/blog/2021/05/como-trabajar-con-ramas-en-git-y-github/. Accessed: 2024-04-13.

[12] Git. Fast version control. https://git-scm.com/. Accessed: 2024-04-14.

[13] GitHub. Let's build from here. https://github.com/. Accessed: 2024-04-13.

[14] Glassdoor. Sueldos para el puesto de Desarrollador Android en España. https://www.glassdoor.es/Sueldos/desarrollador-android-sueldos-SRCH_KO0,21.htm#:~:text=En%20Espa%C3%B1a%2C%20el%20sueldo%20medio,que%20trabajan%20de%20Desarrollador%20android. Accessed: 2024-03-09.

[15] Google. Google Meet. https://meet.google.com/. Accessed: 2024-04-12.

[16] Google. Material Design. https://m3.material.io/. Accessed: 2024-07-02.

[17] Gradle. Gradle Build Tool. https://gradle.org/. Accessed: 2024-04-104.

[18] IBM. ¿Qué es el software de código abierto? https://www.ibm.com/es-es/topics/open-source. Accessed: 2024-02-28.

[19] Ingenieros Asesores. Ingeniería Inversa. Conceptos y aplicaciones. https://ingenierosasesores.com/actualidad/ingenieria-inversa-concepto-aplicaciones/. Accessed: 2024-03-02.

[20] Ionos. La ingeniería inversa del software. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/ingenieria-inversa-de-software/. Accessed: 2024-03-02.

[21] Izertis. Facilitamos la transformación digital. https://www.izertis.com/es/. Accessed: 2024-03-06.

[22] Juan Carlos Garrote Gascón. AppGenda: app Android como agenda virtual de las PYMEs de servicios de un ayuntamiento. https://uvadoc.uva.es/handle/10324/50084. Accessed: 2024-02-21.

[23] Juan Carlos Garrote Gascón. Elaboración de una guía de migración de XML a Jetpack Compose en aplicaciones Android: un caso de estudio de modernización de software. https://uvadoc.uva.es/handle/10324/57389. Accessed: 2024-05-01.

[24] Ken Schwaber y Jeff Sutherland. La guía definitiva de Scrum: Las reglas del juego. https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf. Accessed: 2024-03-06.

[25] Kent Beck, Mike Beedle, Arie van Bennekum y otros 14 autores más. Manifiesto por el desarrollo ágil de Software. https://agilemanifesto.org/iso/es/manifesto.html. Accessed: 2024-03-02.

[26] Kotlin. Kotlin Programming. https://kotlinlang.org/. Accessed: 2024-07-01.

[27] Leburo. Precio de coworking en Valladolid. https://leburocowork.es/precio-coworking-valladolid. Accessed: 2024-03-09.

[28] María Robles del Blanco. Evolución y mejora en la traducción de código Vensim a código Python: ingeniería inversa y mantenimiento de PySD. https://uvadoc.uva.es/handle/10324/50438. Accessed: 2024-02-21.

[29] Medium | Lucas Maurer. Git Gud: The Working Tree, Staging Area, and Local Repo. https://medium.com/@lucasmaurer/git-gud-the-working-tree-staging-area-and-local-repo-a1f0f4822018. Accessed: 2024-04-14.

[30] Medium (Juvinaojesus). Los principios SOLID. https://medium.com/@juvinaojesusd/los-principios-solid-256f106b311a. Accessed: 2024-03-11.

[31] Microsoft Teams. Microsoft Teams. https://www.microsoft.com/es-es/microsoft-teams/group-chat-software. Accessed: 2024-06-25.

[32] OnePlus. OnePlus Nord 3 (5G). https://www.oneplus.com/es/oneplus-nord-3-5g. Accessed: 2024-03-09.

[33] Open Source. Open Source Initative. https://opensource.org/. Accessed: 2024-02-26.

[34] Open Source Guides. Cómo contribuir con el código abierto. https://opensource.guide/es/how-to-contribute/. Accessed: 2024-02-26.

[35] Overleaf. Overleaf, a LaTeX editor. https://www.overleaf.com/. Accessed: 2024-04-12.

[36] ownCloud. A Kiteworks Company. https://owncloud.com/. Accessed: 2024-03-02.

[37] ownCloud. Feature. Spaces. https://owncloud.com/features/spaces/. Accessed: 2024-03-18.

[38] ownCloud. Introduction to Infinity Scale. https://doc.owncloud.com/ocis/next/. Accessed: 2024-03-18.

[39] Paradigma Digital. Estados y recomposición en Jetpack Compose. https://www.paradigmadigital.com/dev/estados-recomposicion-jetpack-compose/. Accessed: 2024-06-28.

[40] PC Componentes. MSI Alpha 15 A3DDK-001XES. https://www.pccomponentes.com/msi-alpha-15-a3ddk-001xes-amd-ryzen-7-3750h-16gb-512gb-ssd-rx-5500m-156. Accessed: 2024-03-09.

[41] Per-Erik Bergman. Medium. Repository Pattern. https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30. Accessed: 2024-07-01.

[42] Profile. SOLID: Los 5 principios que te ayudarán a desarrollar software de calidad. https://profile.es/blog/principios-solid-desarrollo-software-calidad/. Accessed: 2024-03-09.

[43] Redhat. ¿Qué es open-source? https://www.redhat.com/es/topics/open-source/what-is-open-source#:~:text=Originalmente%2C%20la%20expresi%C3%B3n%20open%20source,la%20forma%20que%20consideren%20conveniente. Accessed: 2024-02-26.

[44] Repsol. Tipos de ordenadores y su gasto mensual. https://www.repsol.es/particulares/asesoramiento-consumo/cuanto-consume-ordenador/#:~:text=Por%20lo%20general%2C%20un%20ordenador,cuenta%20el%20tiempo%20de%20uso. Accessed: 2024-03-09.

[45] Richard Albán Fernández. Tutorías InfUVa: una web y bot de Telegram para facilitar la gestión de tutorías para alumnos y profesores de la Escuela de Ingeniería Informática. https://uvadoc.uva.es/handle/10324/57206. Accessed: 2024-02-21.

[46] Robert Martin. Clean Architecture Craftsmans Software Structure. https://www.amazon.com/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164. Accessed: 2024-07-01.

[47] Rocket.chat. Rocket Chat. https://es.rocket.chat/. Accessed: 2024-04-12.

[48] Scrum.org. What is Scrum? https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events. Accessed: 2024-03-03.

[49] Simpl¡learn. What is Gradle? Why do we use gradle? Explained. https://www.simplilearn.com/tutorials/gradle-tutorial/what-is-gradle#:~:text=Gradle%20is%20a%20build%20automation,help%20of%20build%20automation%20tools. Accessed: 2024-04-104.

[50] UEMC (Ángela de Toro). ¿Qué es scrum? Conoce el framework que aguluza el trabajo en equipo. https://www.escueladenegociosydireccion.com/revista/business/scrum-framework-agiliza-trabajo-equipo/. Accessed: 2024-03-11.

[51] Universidad de Valladolid. Guía docente. Trabajo de Fin de Grado. https://apps.stic.uva.es/guias_docentes/uploads/2023/545/46976/1/Documento.pdf. Accessed: 2024-03-08.

[52] Universitat Oberta de Catalunya. Ingeniería Inversa. Qué es, herramientas y técnicas. https://blogs.uoc.edu/informatica/ingenieria-inversa-que-es-herramientas-y-tecnicas/#:~:text=La%20ingenier%C3%ADa%20inversa%20es%20el,fue%20el%20proceso%20de%20fabricaci%C3%B3n. Accessed: 2024-03-02.

[53] Vysor. Vysor, a window to your Phone. https://www.vysor.io/. Accessed: 2024-04-12.

[54] webEmpresa. ¿Qué es ownCloud y para qué sirve? https://www.webempresa.com/blog/que-es-owncloud-y-para-que-sirve.html. Accessed: 2024-03-02.

[55] Wikipedia. Extensible Markup Language (XML). https://es.wikipedia.org/wiki/Extensible_Markup_Language. Accessed: 2024-07-01.

Anexos

Apéndice A

Planificación del proyecto

A.1. Scrum

El desarrollo del proyecto se llevará a cabo mediante el marco de trabajo **Scrum** [24]. Es un marco de trabajo a partir del cual las personas pueden abordar problemas complejos creando de forma simultánea productos de calidad. Este marco de trabajo se lleva utilizando desde la década de los 90 aproximadamente. De forma resumida, Scrum consiste en una serie de iteraciones en la que cada uno de los integrantes del equipo, los cuales algunos presentan roles diferentes, se coordinan y cooperan entre sí para generar un incremento de producto en el periodo de tiempo que se ha establecido. Hay que mencionar también que el marco de trabajo Scrum se encuentra dentro de la metodología ágil, la cual se caracteriza por los siguientes cuatro valores según el *Manifiesto Ágil* [25]:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Dentro de Scrum se definen artefactos, eventos y roles tal y como se puede ver en la Figura A.1.

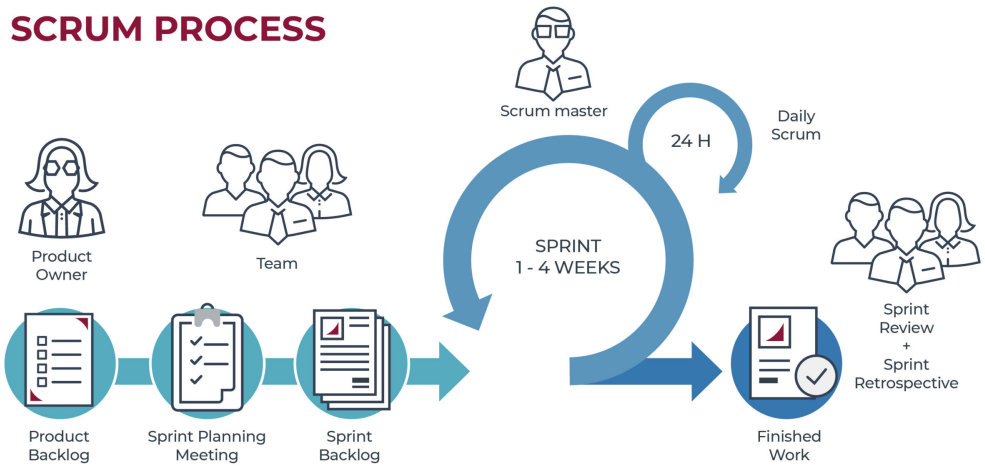

Figura A.1: Artefactos, Roles y Eventos de Scrum [50]

A.1.1. Artefactos

En el marco de trabajo Scrum se define como artefacto a aquellos resultados que surgen a partir del trabajo realizado durante un cierto período de tiempo previamente establecido. Al mismo tiempo estos artefactos proporcionan transparencia y registro de la aplicación Scrum. Existen tres artefactos principales [8] dentro de este marco de trabajo, los cuales son:

- **Product Backlog**: Consiste en una lista ordenada que contiene cualquier tipo de trabajo que haya que realizar en el producto. Es la principal fuente de información del mismo. El responsable de este artefacto es exclusivamente el Product Owner.
- **Sprint Backlog**: Consiste en una lista de elementos pertenecientes al Product Backlog que se deben de llevar a cabo durante un sprint.
- **Incremento del producto**: Se trata del resultado de un sprint. Consiste en la suma de todas las tareas, casos de uso, historias de usuario u otro tipo de trabajo que se haya realizado durante el sprint.

A.1.2. Roles

Un equipo de trabajo Scrum está conformado por diferentes roles. Cada uno de ellos posee unas responsabilidades y debe de trabajar de una forma en particular. Los tres roles existentes [9] dentro de este marco de trabajo son los siguientes:

- **Product Owner**: Encargado de optimizar y maximizar el producto. Es el rol encargado de gestionar el valor del producto a través del Product Backlog. De forma tradicional este rol se ha asemejado al de un cliente, siendo la persona que establece qué se debe hacer hacer o cuál es la funcionalidad que se requiere conseguir.
- **Scrum Master**: Tiene dos funciones dentro del marco de trabajo. La primera de ellas es gestionar el proceso Scrum. Es el rol encargado de asegurar que se están llevando a cabo todas las etapas de manera correcta. Otra de las funciones que posee el Scrum Master es de eliminar cualquier tipo de impedimento que surja durante el desarrollo del proyecto.
- **Equipo de desarrollo**: Este equipo suele estar formando entre 3 y 9 profesionales. Son los encargados de, como su propio nombre indica, desarrollar el proyecto en el que están involucrados. El equipo de desarrollo se coordinará de forma que puedan conseguir el incremento del producto en el tiempo establecido.

A.1.3. Eventos

Dentro del marco de trabajo Scrum existen diversos eventos en los cuales el producto se va desarrollando de forma progresiva. Algunos de estos eventos son simplemente un análisis de cómo va todo mientras que otros eventos se centran solamente en el desarrollo de la aplicación. Los eventos [48] con más detalle son los siguientes:

- **Sprint**: Es el evento representativo de este marco de trabajo. Un sprint tiene una duración fija entre 1 semana y 4 semanas, en el cual el equipo de desarrollo realiza su trabajo a partir de unos objetivos que se han fijado previamente. Durante el sprint los requisitos no se pueden cambiar, sin embargo es posible hacer retrospectiva del mismo y tratar puntos que no se habían cubierto en los sprints siguientes.
- **Sprint Planning**: Reunión de todo el equipo Scrum en el que se debate acerca de qué ítems del Product Backlog se tienen que desarrollar durante el sprint que dará comienzo tras este evento. La duración de este evento no debería durar más de 2 horas por cada semana de sprint.
- **Daily Scrum**: Evento diario en el cual se reúne el equipo de desarrollo durante aproximadamente 15 minutos. Cada integrante del equipo expone al resto en lo que ha trabajo el día anterior y si ha tenido cualquier tipo de dificultad que le haya impedido terminar con el trabajo asignado.
- **Sprint Review**: Evento que tiene lugar al final de cada sprint en el que se reúne todo el equipo Scrum junto con el cliente para determinar qué objetivos se han completado durante el sprint.
- **Sprint Retrospective**: Con este evento se finaliza el sprint. El equipo de desarrollo se reúne y debate acerca de qué aspectos se pueden mejorar para el siguiente sprint o si ha habido algún problema entre el equipo que haya influido a la hora de conseguir el trabajo establecido previamente.

A.2. Adaptación de Scrum al proyecto

Aunque sí que se utilizará en este proyecto el marco de trabajo Scrum, éste se adaptará en base a unas ciertas necesidades.

En primer lugar los roles. En este caso la responsabilidad del *Scrum Master* recae sobre la tutora por parte de la Universidad, Yania Crespo. Respecto al rol de *Product Owner*, este recae sobre el tutor de empresa, Juan Carlos Garrote. Como trabajador de *ownCloud*, el desarrollo que se va a realizar en este trabajo beneficia directamente a la empresa. Al mismo tiempo, añadir que el resto de integrantes del equipo de desarrollo de *ownCloud* también se ve beneficiado por este TFG. Por último mencionar que el rol de *Equipo de Desarrollo* recae sobre el propio alumno, siendo el encargado de desarrollar en su totalidad todo el proyecto.

Respecto a los eventos, la adaptación a este TFG es la siguiente. Tanto *Sprint Review*, *Sprint Planning* y *Sprint Retrospective* se realizarán cada dos semanas, en concreto los lunes de cada semana. Los sprints tendrán una duración de 2 semanas cada uno, estimando unas 40 horas por cada uno de ellos. De esta manera se llega a las 300 horas totales que aparecen reflejadas en la guía docente [51]. Al mismo tiempo y a diferencia de lo que dicta el marco de trabajo de Scrum, no se llevarán a cabo *Daily Scrum*. En este caso se sustituirán por una *Weekly Scrum*. Este evento será una reunión de seguimiento en el medio del sprint, pudiendo solucionar cualquier tipo de duda o impedimento que surja durante el desarrollo de todo el proyecto.

Por otra parte, en cada sprint se tendrán unas historias de usuario que cumplir. Éstas se estimarán durante el *Sprint Planning* utilizando la sucesión de Fibonacci. Cada punto de historia estimado corresponde a 5 horas de trabajo aproximadamente.

Finalmente, relativo a los artefactos, mencionar que el incremento de producto se conseguirá al final de cada sprint.

A.3. Planificación y calendarización

Teniendo en cuenta que la asignatura del Trabajo de Fin de Grado está estimada en unas 300 horas de trabajo, se ha hecho una planificación de modo que se cumplan ese número determinado de horas. Tal y como se ha dicho en el Apartado A.2, cada sprint presenta una duración de 2 semanas, teniendo un total de **8 sprints** (contando el sprint 0) hasta la primera convocatoria. Cada sprint, al ser de dos semanas de duración se tiene un total de 4 horas diarias sin contar fines de semana. Esto haría un total de **40 horas por cada sprint**, todo ello sin tener en cuenta el sprint 0, el cual tiene un esfuerzo de 20 horas al sprint total. Con esta planificación se cumplirían las 300 horas requeridas. En caso de no presentar el trabajo en la convocatoria ordinaria existe un sprint extra de 2 semanas de duración igualmente, permitiendo alargar las horas de dedicación en cada sprint para poder llegar a tiempo.

La justificación de que el sprint 0 esté estimado con menos horas de esfuerzo, es que en dicho sprint no se llevan a cabo tareas de desarrollo puramente. Está enfocado en la planificación y organización del TFG.

Para hacerlo más fácil, en la Tabla A.1 se detalla la planificación con las diferentes fechas y eventos a lo largo del tiempo hasta la fecha final de entrega en segunda convocatoria.

Sprint	*Fecha Inicio*	*Fecha Fin*	Anotaciones
Sprint 0	19/02/2024	04/03/2024	
Scrum Weekly	26/02/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	04/03/2024		
Sprint 1	04/03/2024	18/03/2024	
Scrum Weekly	11/03/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	18/03/2024		
Sprint 2	18/03/2024	08/04/2024	Vacaciones de Semana Santa del 23/03/2024 al 02/04/2024. Por este motivo el sprint terminará más tarde que el resto
Scrum Weekly	02/04/2024		Los días de reunión son el lunes, pero al ser vacaciones esta semana se pasa para el martes 02/04
Sprint Review, Sprint Retrospective & Sprint Planning	08/04/2024		
Sprint 3	08/04/2024	22/04/2024	
Scrum Weekly	15/04/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	22/04/2024		
Sprint 4	22/04/2024	06/05/2024	
Scrum Weekly	29/04/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	06/05/2024		
Sprint 5	06/05/2024	20/05/2024	
Scrum Weekly	13/05/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	20/05/2024		
Sprint 6	20/05/2024	03/06/2024	Estudio de ERSS para la convocatoria ordinaria
Scrum Weekly	27/05/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	03/06/2024		
Sprint 7	03/06/2024	17/06/2024	Los días 17, 18, 19 y 20 serán de revisión para la convocatoria ordinaria. El día 5 es la convocatoria ordinaria de ERSS
Scrum Weekly	10/06/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	17/06/2024		
Límite solicitud defensa convocatoria ordinaria	20/06/2024		Extraordinaria de ERSS
Sprint Extra 1	24/06/2024	07/07/2024	Este sprint sólo será necesario en caso de que el trabajo no se presente en la convocatoria ordinaria
Scrum Weekly	01/07/2024		
Sprint Review, Sprint Retrospective & Sprint Planning	06/07/2024		
Límite solicitud defensa convocatoria extraordinaria	07/07/2024		

Tabla A.1: Planificación del calendario

A.4. Plan de riesgos

Antes del desarrollo de todo el TFG, es importante llevar a cabo un proceso de análisis de riesgos. Para que este análisis sea correcto se han de detectar las siguientes características dentro de cada riesgo:

- **Probabilidad**: Posibilidad de que un riesgo se pueda llegar a materializar. Se utiliza

una escala con los siguientes valores: baja, media y alta.

- **Impacto**: Efecto que puede llegar a producir el riesgo en caso de que se materialice. Se utiliza una escala con los siguientes valores: bajo, medio y alto.
- **Plan de mitigación**: Conjunto de acciones que se pueden llevar a cabo para reducir la probabilidad de que un riesgo se materialice.
- **Plan de contingencia**: Conjunto de acciones que se tienen que llevar a cabo después de que un riesgo se haya materializado.

A partir de la Tabla A.2 hasta la Tabla A.9 se muestran todos los riesgos que se han detectado y están relacionados directamente con el proyecto. En todos ellos aparecen las características que se han definido anteriormente.

Riesgo R01	
Título	Trabajo individual
Descripción	El desarrollo del proyecto es llevado a cabo solamente por una persona
Probabilidad	Alta
Impacto	Bajo
Plan de mitigación	Pedir ayuda a compañeros que hayan hecho el TFG previamente
Plan de contingencia	Replanificar todo el trabajo y presentarlo en una fecha en la que se llegue a tiempo

Tabla A.2: Riesgo R01. Trabajo individual.

Riesgo R02	
Título	Dominio de la tecnología
Descripción	No se domina la tecnología que se va a utilizar durante el desarrollo del TFG
Probabilidad	Media
Impacto	Alto
Plan de mitigación	Apoyo en cursos o información en la red para adquirir los conocimientos necesarios. Además, preguntar al tutor todas las dudas que se tenga
Plan de contingencia	Elegir otro tema de TFG sobre el que sí se tenga un control sobre la tecnología a usar

Tabla A.3: Riesgo R02. Dominio de la tecnología.

Riesgo R03	
Título	Ausencia temporal por enfermedad o vacaciones
Descripción	El estudiante cae enfermo o se va de vacaciones durante un determinado período de tiempo
Probabilidad	Media
Impacto	Medio
Plan de mitigación	En horas muertas y momentos aburridos, avanzar con el trabajo restante aprovechando ese tiempo
Plan de contingencia	Dedicarle más horas de las usuales al día, incluso sacrificando otras actividades o momentos del día en el que se esté más relajado

Tabla A.4: Riesgo R03. Ausencia temporal del estudiante por enfermedad.

Riesgo R04	
Título	Asignaturas pendientes
Descripción	Tener durante el cuatrimestre asignaturas pendientes por aprobar
Probabilidad	Alta
Impacto	Medio
Plan de mitigación	Tener un plan de estudio de modo que se pueda compaginar todo lo pendiente en el cuatrimestre
Plan de contingencia	Aprobar las asignaturas en sus respectivas convocatorias y presentar el TFG en la última convocatoria

Tabla A.5: Riesgo R04. Asignaturas pendientes.

Riesgo R05	
Título	Confinamiento debido al COVID-19
Descripción	La situación actual del COVID-19 empeora, provocando un confinamiento total de la población
Probabilidad	Baja
Impacto	Bajo
Plan de mitigación	Tener un espacio de trabajo que te permita trabajar de forma online
Plan de contingencia	Realizar todas las tutorías de forma remota con los tutores a través de las herramientas que se tenga a disposición

Tabla A.6: Riesgo R05. Confinamiento debido al COVID-19.

Riesgo R06	
Título	Indisponibilidad del equipo de trabajo
Descripción	Imposibilidad de poder acceder al equipo en el que se está realizando el TFG
Probabilidad	Baja
Impacto	Bajo
Plan de mitigación	Llevar todo al día para que en caso de que sea solamente unos días, no tener demasiado retraso en el proyecto
Plan de contingencia	Obtener otro dispositivo de la empresa lo más rápido posible

Tabla A.7: Riesgo R06. Indisponibilidad del equipo de trabajo.

Riesgo R07	
Título	Pérdida de información
Descripción	Se pierde la información y el contenido del TFG debido a problemas externos y ajenos al estudiante
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Tener un backup en otro dispositivo
Plan de contingencia	Replanificar el proyecto acortando ciertos objetivos con el fin de poder entregarlo en las fechas estipuladas

Tabla A.8: Riesgo R07. Pérdida de información.

Riesgo R08	
Título	No entender el código de la aplicación open-source
Descripción	El estudiante no entiende el código que se le proporciona para llevar a cabo el proyecto de ingeniería inversa
Probabilidad	Media
Impacto	Alto
Plan de mitigación	Preguntar todas las dudas que vayan surgiendo al momento
Plan de contingencia	Reservar varios días para dedicarlos exclusivamente a la comprensión del código con el tutor de forma presencial

Tabla A.9: Riesgo R08. No entender el código de la aplicación open-source.

A.5. Presupuesto

Finalizando esta parte de planificación y calendarización, se ha elaborado un plan de presupuesto relacionado directamente con el proyecto. Este presupuesto se ha dividido en dos partes. Por un lado se tiene el **presupuesto simulado** y por otro el lado el **presupuesto real**. El presupuesto real es aquel que se estima si esto fuese un proyecto de una empresa real. El presupuesto simulado se ha adaptado a los materiales educativos y a las condiciones que se van a dar durante el desarrollo del TFG.

A.5.1. Presupuesto simulado

En primer lugar lo que se tiene en cuenta es el sueldo de un desarrollador de Android actualmente. Consultando varias fuentes de información [14] [7] se ha llegado a la conclusión de que el sueldo medio anual de un desarrollador Android es de 30.000€brutos anuales. A ese sueldo hay que aplicarle el 31,4 % que le costaría a la empresa por la Seguridad Social. Con esto el valor del empleado para la empresa sería de 39.420€. Por otro lado, teniendo en cuenta que la jornada laboral del desarrollador es de 8 horas de lunes a viernes, se tiene que el número de horas trabajadas al mes es de 176 horas (22 días al mes). A partir del sueldo bruto que se ha especificado anteriormente se calcula que el trabajador cobra 14,2€/hora aproximadamente. En este proyecto en particular, se dedicarán 300 horas por lo que el sueldo total quedaría reflejado en una cifra total de **4.260€**.

Respecto al equipo de trabajo se utilizará un *MacBook Pro 2023 (M3)*[3]. El precio completo es de 2.259€. Teniendo en cuenta que el dispositivo tiene una vida útil de 4 años se llega a la conclusión de que el equipo cuesta 47,06€/mes. Como el proyecto se desarrollará durante 5 meses, la amortización del dispositivo se ve reflejada en una cantidad de **235,30€**.

Por otra parte, se utilizará también un dispositivo móvil para llevar a cabo todas las pruebas. El dispositivo elegido es un *One Plus Nord 3 (5G)* [32]. Su precio total es de 449€. Al igual que el equipo portátil de trabajo, se estima que su vida útil es de 4 años teniendo un precio de 9,35€/mes. De esta manera, contando con los 5 meses de duración del proyecto, se tiene una amortización de **46,75€**.

El desarrollo del proyecto se llevará a cabo en un co-working ubicado en Valladolid [27]. La tarifa elegida será de media jornada (4 horas al día de lunes a viernes) por lo que el coste por el espacio de trabajo es de 85€/mes. Teniendo en cuenta los 5 meses de desarrollo, el precio total del co-working asciende a **425€**.

Finalmente, es necesario incluir en el presupuesto estimado el coste de las licencias de todos los programas que se van a utilizar durante el desarrollo del proyecto. En este caso la única licencia que hay que pagar es la de Astah Professional [4]. Su coste es de 9,99€/mes, por lo que aplicando al igual que los anteriores casos la duración del proyecto se tendría un precio final de **49,95€**.

Concepto	***Precio***	***Cantidad***	***Total***
Coste del empleado	14,2€/hora	300 horas	4.260€
Equipo de trabajo (*MacBook Pro 2023*)	47,06€/mes	5 meses	235,3€
Dispositivo móvil (*One Plus Nord 3*)	9,35€/mes	5 meses	46,75€
Licencia *Astah Professional*	9,99€/mes	5 meses	49,95€
Espacio de trabajo (*Co-Working*)	85€/mes	5 meses	425€
TOTAL			**5017€**

Tabla A.10: Presupuesto simulado.

A.5.2. Presupuesto real

A diferencia del presupuesto estimado, ciertos elementos no se aplican al presupuesto real como por ejemplo el coste del empleado. Como el equipo de desarrollo en este caso es el propio estudiante, el coste del mismo es de 0€. Sucede lo mismo con el co-working. El estudiante llevará a cabo el desarrollo del proyecto desde su casa por lo que no existirá ningún gasto de alquiler respecto a la zona de trabajo. Sin embargo, referente a este concepto, hay que aplicar un gasto que no se había mencionado anteriormente. Es el caso de la luz. Con el presupuesto real se tenía incluido el gasto de luz dentro de la tarifa de la oficina de co-working pero en este caso habrá que tener en cuenta la tarifa de luz y el consumo de los dispositivos para calcular la cifra total.

Aproximadamente el consumo de un ordenador portátil es de 16kWh mensuales [44], lo cual sería un consumo por hora de 0,022kWh (aplicando que un mes tiene 30 días). Si se tiene en cuenta que se emplearán 300 horas de trabajo aproximadamente, el consumo total del dispositivo es de 6,6kWh. Con una tarifa de 0,130435 €/kWh el gasto total es de **0,86€**.

Por otra parte, el equipo utilizado por el estudiando es un *MSI Alpha 15* [40]. Su precio es de 1099,84€. Teniendo en cuenta que la vida útil del dispositivo es de 4 años (48 meses), el precio del mismo saldría a 22,91€/mes. Para cubrir los 5 meses de desarrollo del TFG se tiene que el coste total del equipo de trabajo es de **114,55€**.

Respecto al smartphone que se va a utilizar para realizar las pruebas correspondientes, se cuenta con un Motorola Moto G14. Su precio es de 124€actualmente por lo que, contando con los 4 años de vida útil que tiene el dispositivo su precio saldría a 2,58€/mes. Al igual que en los anteriores casos hay que multiplicar esa cantidad por el número total de meses de desarrollo (5 meses). El precio total ascendería a **12,92€**.

Finalmente, no se aplicará ningún coste relacionado a las licencias de todos los programas que se utilizaran para el desarrollo ya que la universidad las proporciona de forma gratuita.

Concepto	*Precio*	*Cantidad*	*Total*
Equipo de trabajo (*MSI Alpha 15*)	22,91€/mes	5 meses	114,55€
Dispositivo móvil (*Motorola Moto G14*)	2,58€/mes	5 meses	12,92€
Coste de la luz	0,1304€/kWh	6,6kWh	0,86€
TOTAL			**128,33€**

Tabla A.11: Presupuesto real.

A.6. Product Backlog Inicial

Tal y como se ha mencionado en el Apartado A.1.2, el rol del *Product Owner* será desempeñado por Juan Carlos. Este tendrá la responsabilidad de definir el product backlog inicial. Como se está utilizando un marco de trabajo Scrum, cada una de las épicas se definen como historias de usuario. La estructura de una historia de usuario es la siguiente: "*Como <stateholder> quiero <objetivo> para <razón para conseguir ese objetivo>*". Siguiendo este modelo a continuación se muestra el product backlog inicial.

Número	**Épica**
EP01	Como equipo de desarrollo quiero ser capaz de migrar una interfaz de la aplicación la cual está definida en XML al framework que se utiliza hoy en día en Android: Jetpack Compose.
EP02	Como equipo de desarrollo quiero poder documentar y actualizar la arquitectura de la aplicación a través de un proceso de ingeniería inversa.

Tabla A.12: *Product Backlog* inicial

A.7. Product Backlog Final

Desde el inicio del proyecto hasta su finalización, el Product Backlog ha sufrido diferentes cambios, pues se han ido modificando algunas historias de usuario, se han añadido otras y también se han eliminado aquellas que no resultaban interesantes o por ciertos motivos debían agruparse con otras. En la Tabla A.13 se muestran todas aquellas historias de usuario relacionadas con la épica **EP02: Ingeniería Inversa** mientras que en la Tabla A.14 se muestran todas las historias de usuario relacionadas con la **EP01: Migración de XML a Jetpack Compose**.

Código	*Historia de usuario*	*Puntos*
HU01	Como desarrollador quiero documentarme sobre la arquitectura, funcionalidad, metodología y otros aspectos importantes de ownCloud	2
HU02	Como desarrollador quiero elaborar un diagrama inicial en el que se muestre la arquitectura principal de ownCloud	1
HU03	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para un caso de uso generalizado	2
HU04	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo owncloudApp relacionado con el caso de uso de consulta del estado de las subidas	1
HU05	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudDomain relacionado con el caso de uso de consulta del estado de las subidas	1
HU06	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudData relacionado con el caso de uso de consultado del estado de las subidas	1
HU07	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo owncloudApp relacionado con el caso de uso de subida de un fichero	1
HU08	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudDomain relacionado con el caso de uso de subida de un fichero	1
HU09	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo owncloudData relacionado con el caso de uso de subida de un fichero	1
HU10	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo owncloudComLibrary relacionado con el caso de uso de subida de un fichero	2
HU11	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos del caso de uso de subida de un fichero	5
HU12	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos del caso de uso de consulta del estado de las subidas	3

Tabla A.13: Historias de usuario asociadas a la épica 2 (EP02: Ingeniería inversa)

Código	*Historia de usuario*	*Puntos*
HU13	Como desarrollador quiero documentarme y aprender sobre la tecnología de Jetpack Compose	2
HU14	Como desarrollador quiero migrar la pantalla de subidas de la aplicación ownCloud de XML a Jetpack Compose	5
HU15	Como desarrollador quiero poder hacer una comparativa después de haber migrado la pantalla de las subidas	2

Tabla A.14: Historias de usuario asociadas a la épica 1 (EP01: Migración de XML a Jetpack Compose)

Apéndice B

Seguimiento del proyecto

B.1. Introducción

En este capítulo se ha detallado el seguimiento real de todo el proyecto sprint a sprint. Se ha establecido un apartado por cada sprint, en el cual se entra más a detalle con el progreso de cada uno. A excepción del Sprint 0, todos los demás sprint tendrán una estructura similar.

El proyecto comenzó el día 19 de febrero de 2024, y según la planificación inicial que se ha realizado terminará el 20 de junio de 2024 como muy pronto (primeras fechas) o el 7 de julio de 2024 como muy tarde (segundas fechas).

B.2. Sprints

B.2.1. Sprint 0 (19/02/2024 - 04/03/2024)

La duración de este sprint es de 2 semanas al igual que el resto, sin embargo, como ya se ha mencionado en el capítulo A.7 este sprint es bastante diferente al resto. La estimación inicial que se le ha dado a este sprint es de **20 horas**, que a diferencia del resto es la mitad. Durante este sprint se ha llevado a cabo la planificación y organización del TFG. Además, durante este período de tiempo se ha comenzado con la lectura de varios TFG con el mismo método de trabajo, los cuáles pertenecían a **Juan Carlos Garrote Gascón** [22], **María Robles del Blanco** [28] y **Richard Albán Fernández** [45]. En especial, se ha puesto bastante atención en el TFG de María Robles pues trataba temas de ingeniería inversa de un proyecto open-source. Estas son las principales tareas que se han llevado a cabo durante este sprint, las cuales se encuentran detalladas según las horas de trabajo y estado de cada una en la Tabla B.1.

Nombre de la tarea	***Tiempo estimado***	***Tiempo dedicado***	***Estado***
Lectura del TFG de Juan Carlos Garrote	4h	4h 45min	Completada
Lectura del TFG de María Robles	4h	7h	Completada
Lectura del TFG de Richard Albán	4h	4h	Completada
Planificación del TFG	4h	3h	Completada
Organización de los capítulos del TFG	4h	3h15min	Completada
Total	**20h**	**22h**	**4/4**

Tabla B.1: Tareas del Sprint 0

B.2.2. Sprint 1 (04/03/2024 - 18/03/2024)

Durante este sprint ya se ha comenzando con las tareas de desarrollo del proyecto. En primer lugar, se llevó a cabo una serie de presentaciones para la formación. La primera de todas ha sido la descripción general del proyecto y la puesta en marcha. Durante esta sesión de formación, se ha explicado en líneas generales cuales son los diferentes productos que tiene la marca al igual que su estructura a grandes rasgos (*back-end*, *front-end*, *mobile*). Tras su finalización se ha hecho la instalación del proyecto en el dispositivo personal para poder trabajar en él. Tras ello se ha llevado a cabo una segunda sesión de formación, esta vez sobre el método de trabajo del equipo de *ownCloud*. En esta presentación se ha explicado cuál es la principal herramienta que se utiliza para la organización del proyecto (*GitHub*) al igual que otros detalles más a fondo como pueden ser los distintos estados en los que se encuentra cada tarea. Tras esa segunda sesión, se ha llevado a cabo una tercera. En esta sesión se ha explicado a fondo cual era la arquitectura de toda la aplicación. Para acabar, mencionar que durante este sprint se han dedicado **30h** en total en las cuales se incluyen, no sólo las historias de usuario que se habían metido para este sprint sino también tareas de documentación.

Durante este sprint se ha tenido el problema de la asignatura pendiente, contemplado en el riesgo A.5 y siendo imposible dedicarle las horas necesarias para completar todas las tareas. A pesar de ello se ha conseguido sacar adelante bastante trabajo. Todas aquellas tareas que no se han terminado, se desarrollarán durante sprints futuros, teniendo la necesidad de dedicarle más horas, siempre y cuando se den las condiciones necesarias. Además, existe una semana de vacaciones que en principio no se iba a hacer nada, pero se cuenta con la posibilidad de usar esos días para poder avanzar tareas atrasadas.

Historia de usuario	*Descripción de la tarea*	*Tiempo empleado*	*Estado*
HU01	Como desarrollador quiero documentarme sobre la arquitectura, funcionalidad, metodología y otros aspectos importantes de ownCloud	5h 30min	Completada
HU02	Como desarrollador quiero elaborar un diagrama inicial en el que se muestre la arquitectura principal de ownCloud	4h	En progreso
HU03	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle generalizado en el que se muestre la arquitectura en su totalidad	2h 30 min	En progreso
-	Documentación de la introducción	5h 30min	Completada
-	Documentación de requisitos y planificación	6h	Completada
-	Documentación de la arquitectura de ownCloud	2h	En progreso
-	Documentación del seguimiento del Sprint 0	2h	Completada
-	Documentación del seguimiento del Sprint 1	2h 30min	Completada
Total		**30h**	**5/8**

Tabla B.2: Tareas del Sprint 1

B.2.3. Sprint 2 (18/03/2024 - 08/04/2024)

Tal y como se mencionó anteriormente, la finalización de este sprint ha sido en una fecha más tardía que las 2 semanas estipuladas en la planificación por cada sprint por el simple motivo de que entre medias estaba la semana de vacaciones de Semana Santa. Durante este sprint, aparte de terminar aquellas historias de usuario que quedaron pendientes se han desarrollado todos los diagramas Modules&UsesStyle (**HU04, HU05, HU06, HU07, HU08, HU09, HU10**) de cada uno de los módulos que componen la aplicación de *ownCloud*, separados por casos de usos. Una vez que se han terminado los diagramas por módulos se ha creado otro diagrama Modules&UsesStyle para cada uno de los casos de usos en el que únicamente se muestra la relación entre todas las capas que conforma la aplicación. En este caso, se han especificado dos casos relacionados con la pantalla, la cual más tarde se migrará de XML a Jetpack Compose. El hecho de que haya dos casos de usos es que en uno de ellos no utilizaba uno de los módulos (*owncloudComLibrary*) mientras en el otro caso de uso sí se realizaban llamadas de red hacia servidores externos, y por tanto se podía apreciar la relación entre todos los módulos de la aplicación. A continuación se muestra un tabla detallada con todas las historias de usuario que se han desarrollado durante este sprint, así como su tiempo dedicado y el estado en el que se encuentran. En este caso no es necesario pasar al siguiente sprint ninguna tarea, ya que todas se encuentran en estado completado. Como mucho, realizar algunos retoques en los diagramas a partir del *Sprint Review* que da por finalizado este sprint y comienza el siguiente. Por otra parte, se puede apreciar cómo se ha necesitado menos tiempo del que se había estimado inicialmente, por lo que en sprints posteriores se tendrá la posibilidad de dedicar más tiempo a tareas que sean más difíciles o largas para hacer. A pesar de no haber cumplido con las horas que se habían estipulado en

la planificación (40 horas), se ha conseguido sacar adelante todo el trabajo, a excepción de la documentación que está pendiente de diagramas, que se había planificado teniendo un total de **31h** dedicadas entre historias de usuario y otras tareas de documentación.

Historia de usuario	***Descripción de la tarea***	***Tiempo empleado***	***Estado***
HU02	Como desarrollador quiero elaborar un diagrama inicial en el que se muestre la arquitectura principal de ownCloud	1h 30min	Completada
HU03	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle generalizado en el que se muestre la arquitectura en su totalidad	4h	Completada
HU04	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudApp relacionado con el caso de uso de consulta de estado de las subidas	4h	Completada
HU05	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudDomain relacionado con el caso de uso de consulta de estado de las subidas	3h 30min	Completada
HU06	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudData relacionado con el caso de uso de consulta de estado de las subidas	3h	Completada
HU07	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudApp relacionado con el caso de uso de subida de un fichero	3h	Completada
HU08	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudDomain relacionado con el caso de uso de subida de un fichero	2h 30min	Completada
HU09	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo ownCloudData relacionado con el caso de uso de subida de un fichero	2h 15min	Completada
HU10	Como desarrollador quiero elaborar un diagrama Modules&UsesStyle para el módulo owncloudComLibrary relacionado con el caso de uso de subida de un fichero	2h	Completada
-	Documentación del seguimiento del Sprint 2	2h	Completada
-	Documentación de la arquitectura de ownCloud	3h 15min	En progreso
Total		**31h**	**10/11**

Tabla B.3: Tareas del Sprint 2

B.2.4. Sprint 3 (08/04/2024 - 22/04/2024)

Durante el transcurso de este sprint se ha tenido en cuenta el riesgo de la asignatura pendiente A.5, contemplado en el apartado de los riesgos posibles para este proyecto. Por este motivo no se ha podido dedicar el tiempo suficiente como para sacar adelante todas las tareas que se habían planificado para este sprint. A pesar de ello se ha comenzado con la historia de usuario **HU11**. Para ir desarrollando el diagrama de secuencia, se ha ido mirando lentamente todas las llamadas a los diferentes métodos de las clases implicadas además de los argumentos que iban dentro de cada llamada. También se han modificado

ligeramente los diagramas realizados en el anterior sprint, cosa que se tenía prevista y por tanto entraba dentro de la planificación de este sprint. De forma adicional, y con las historias de usuario pendientes para terminar en el siguiente sprint, se han llevado a cabo tareas de documentación. En primer lugar se ha documentado el apartado de las tecnologías utilizadas casi en su totalidad, faltando solamente realizar una comparación inicial entre XML y Jetpack Compose. Otra tarea de documentación que se ha llevado a cabo ha sido la explicación parcial de los diagramas de clases que se terminaron durante el sprint anterior. Debido a esa falta de tiempo para poder dedicarle a este sprint, ese apartado también ha quedado incompleto por lo que se irá terminando a lo largo de los siguientes sprints según se vayan acabando todos los diagramas pendientes. Teniendo en cuenta todos los inconvenientes que se han tenido durante sprint, se ha dedicado un total de **29h 50min**. Todas las tareas que se tenían previstas en este sprint se terminarán en el siguiente sprint.

Historia de usuario	***Descripción de la tarea***	***Tiempo empleado***	***Estado***
HU11	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos para el caso de uso de subida de un fichero	11h	En progreso
HU12	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos para el caso de uso de consulta del estado de las subidas	-	No iniciada
-	Documentación de la arquitectura de ownCloud	6h 15min	En progreso
-	Documentación de las tecnologías utilizadas	6h 35min	En progreso
-	Modificaciones en los diagramas Modules&UsesStyle	4h	Completada
-	Documentación del seguimiento del Sprint 3	2h	Completada
Total		**29h 50min**	**2/6**

Tabla B.4: Tareas del Sprint 3

B.2.5. Sprint 4 (22/04/2024 - 06/05/2024)

De las dos semanas que conforman el sprint, sólo ha sido posible trabajar durante la segunda semana del mismo debido al riesgo A.5, el cual ya estaba contemplado en el apartado respectivo. A pesar de no haber tenido una semana con disponibilidad para adelantar parte de las tareas que se habían movido para este sprint, se han conseguido terminar 3 de las cinco que se habían planificado. La primera de ellas ha sido la historia de usuario **HU11**. Se han desarrollado todos los diagramas de secuencia que correspondía al caso de uso de subida de un fichero, el cual incluía tanto parte de registro de datos local como registro de datos en remoto, a través de llamadas de red. La otra tarea que también se ha terminado durante este sprint ha sido la documentación del apartado de las tecnologías utilizadas en todo el TFG. También mencionar que la tarea de documentación de la arquitectura de *ownCloud* lleva varios sprints en progreso. Esto es debido a que según se van haciendo los diagramas se va plasmando toda la información que corresponda en el informe y por tanto no es posible terminar con esta tarea hasta que se hayan desarrollado todas las historias de usuario relacionadas con la épica **EP01**. Por otra parte se ha comenzado con el desarrollo del diagrama de secuencia para el caso de uso de consulta del estado de las subidas. Este trabajo va a ser bastante más fácil que para el otro caso de uso, pues muchas cosas se pueden referenciar con el otro diagrama además de no tener toda la parte de llamadas de red hacia un servidor externo a la aplicación.

Teniendo en cuenta que todavía se encuentra en desarrollo se pasará al siguiente sprint. Para acabar, mencionar que las horas que se han dedicado a este sprint han sido **27h**, las cuales se detallan más a fondo en la Tabla B.5.

Historia de usuario	***Descripción de la tarea***	***Tiempo empleado***	***Estado***
HU11	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos para el caso de uso de subida de un fichero	14h 45min	Completada
HU12	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos para el caso de uso de consulta del estado de las subidas	4 h	En progreso
-	Documentación de las tecnologías utilizadas	2h 15min	Completada
-	Documentación de la arquitectura de ownCloud	4h	En progreso
-	Documentación del seguimiento del Sprint 4	2h	Completada
Total		**27h**	**3/5**

Tabla B.5: Tareas del Sprint 4

B.2.6. Sprint 5 (06/05/2024 - 20/05/2024)

El principal trabajo que se ha desarrollado a lo largo del sprint ha sido la historia de usuario **HU12** correspondiente a la elaboración los diagramas de secuencia mostrando la interacción de todos los objetos que participan en el caso de uso de consulta del estado de las subidas además de documentar todo lo relativo a los diagramas de secuencia correspondiente al caso de uso de subida de un fichero. Durante este sprint y, como se lleva apreciando a lo largo de otros sprints, se ha manifestado el riesgo A.5. Como consecuencia de esto, no se ha podido realizar suficiente trabajo y, por tanto, no se ha conseguido terminar las tareas que pertenecían al sprint anterior. Además, después de una reunión con los tutores y viendo el alcance de todas las tareas que quedaban por hacer, se ha decidido utilizar el Sprint extra que se había planificado en la Tabla A.1 como sprint de refuerzo en caso de contingencia. Además, se ha decidido dedicar el siguiente sprint entero a documentación de la arquitectura de *ownCloud*, dando por terminada la primera parte del trabajo. Por tanto durante los Sprint 7 y Sprint extra se llevará a cabo el trabajo de la migración de Jetpack Compose junto con su respectiva documentación.

En resumen, en este sprint se han dedicado un total de **36h** repartidas en tareas según indica la Tabla B.6.

Historia de usuario	*Descripción de la tarea*	*Tiempo empleado*	*Estado*
HU12	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos para el caso de uso de consulta del estado de las subidas	10h 40min	En progreso
-	Documentación de la arquitectura de ownCloud	15h 10min	En progreso
-	Documentación del seguimiento del Sprint 5	2h	Completada
-	Modificaciones en los diagramas relacionados con la historia de usuario HU11	8h 10min	Completada
Total		**36h**	**2/4**

Tabla B.6: Tareas del Sprint 5

B.2.7. Sprint 6 (20/05/2024 - 03/06/2024)

El principal trabajo que se ha desarrollado durante este sprint es la documentación de toda la arquitectura de *ownCloud* tanto de la vista estática como de la vista dinámica. Por otro lado también se ha terminado con la historia de usuario **HU12** la cual era la última asociada a la épica **EP01**. Respecto al tiempo dedicado, ha sido el primer sprint en el que se han completado todas las horas que se habían estimado durante la planificación, habiendo dedicado un total de **44h**. Todas las tareas y su respectivo tiempo se encuentran reflejadas en la Tabla B.7.

Historia de usuario	*Descripción de la tarea*	*Tiempo empleado*	*Estado*
HU12	Como desarrollador quiero elaborar diagramas de secuencia que describan la interacción entre objetos para el caso de uso de consulta del estado de las subidas	4h	Completada
-	Documentación de la arquitectura de ownCloud	38h	Completada
-	Documentación del seguimiento del Sprint 6	2h	Completada
Total		**44h**	**3/3**

Tabla B.7: Tareas del Sprint 6

B.2.8. Sprint 7 (03/06/2024 - 17/06/2024)

Este sprint se ha dedicado exclusivamente a la parte de la migración de XML a Jetpack Compose, la cual engloba las historias de usuario **HU13**, **HU14** y **HU15**. Antes de comenzar con el proceso de migración de la pantalla del caso de uso, se ha tenido un período de formación en Jetpack Compose (**HU13**), sobre la cual no se tenía nada de conocimiento previo. La formación se ha basado en realizar los *codelabs* de Android, en los que se ha trabajado sobre varias de las funcionalidades o características de esta tecnología en cuestión, habiendo dedicado un total de aproximadamente **15 horas y 15 minutos** de trabajo. Una vez que se ha terminado con todo el proceso de formación, el siguiente paso ha sido realizar la migración de la pantalla (**HU14**). Todo este proceso se ha conseguido completar en un tiempo menor que el estimado, teniendo un total de **21 horas** de trabajo. Algo que hay que mencionar es que durante la segunda semana del sprint se ha tenido una ausencia durante aproximadamente cuatro días, por lo que el número de horas que se han podido dedicar a este

sprint ha sido menor que el que se tenía estimado. A pesar de ello, este problema ya se había contemplado en el riesgo A.4 y el trabajo se finalizó correctamente antes de esa ausencia. Una vez terminado todo el proceso de implementación se ha llevado a cabo un proceso de revisión de código por parte de Juan Carlos, en el cual se han cambiado algunas líneas del código. Adicionalmente, aún debe pasar un proceso de pruebas por parte del ingeniero *QA* en el cual pueden surgir nuevos cambios, de ahí que el estado de esa historia de usuario sea: **En progreso**. En la Tabla B.8 se encuentran todas las tareas que se han mencionado anteriormente de forma esquemática junto con el total de horas dedicadas y el estado de cada una de ellas.

Historia de usuario	***Descripción de la tarea***	***Tiempo empleado***	***Estado***
HU13	Como desarrollador quiero documentarme y aprender sobre la tecnología Jetpack Compose	15h 15min	Completada
HU14	Como desarrollador quiero migrar la pantalla de subidas de la aplicación *ownCloud* de XML a Jetpack Compose	21h	En progreso
-	Documentación del seguimiento del Sprint 7	2h	Completada
Total		**38h 15min**	**2/3**

Tabla B.8: Tareas del Sprint 7

B.2.9. Sprint Extra (17/06/2024 - 01/07/2024)

Durante este sprint se ha trabajado en varias tareas de diferente tipo. Tras el reporte proporcionado por un ingeniero QA perteneciente al equipo de *ownCloud* en Izertis, se ha llegado a la conclusión de que los dos problemas que habían surgido no se resolverán por varios motivos. El primero de ellos por problemas técnicos, ya que la tecnología que se ha utilizado (Jetpack Compose) no permite llevar a cabo esos cambios que se habían propuesto. Por otro lado, el segundo de los reportes tampoco se ha desarrollado debido a que era un problema que venía de un desarrollo anterior y no tenía ninguna relación directa con la funcionalidad de la interfaz de usuario, por lo que se sale del alcance de este proyecto. El tiempo total que se ha dedicado a estas tareas, las cuales han sido más de investigación que de desarrollo, ha sido de **5 horas**. Con esto terminado se daría por completada la historia de usuario **HU14**. Por otro lado también se ha llevado a cabo la historia de usuario **HU15**, en la que se explicaba las ventajas que tenía Jetpack Compose respecto a la creación de interfaces en vistas XML. Todo ello ha quedado reflejado en el Apartado 7.3. Terminando con las tareas del sprint, también se ha trabajado en la documentación del Capítulo 6.2 y Capítulo 7.3.4. Finalmente se ha llevado a cabo un proceso exhaustivo de corrección por parte de los tutores, modificando algunas cosas del documento que resultaban confusas o incluso era erróneas en el contexto en el que se estaba trabajando. Todas estas tareas se pueden ver desarrolladas en la Tabla B.9 con más detalles, mostrando las horas que se han dedicado además del estado en el que se encuentran.

Historia de usuario	*Descripción de la tarea*	*Tiempo empleado*	*Estado*
HU14	Como desarrollador quiero migrar la pantalla de subidas de la aplicación ownCloud de XML a Jetpack Compose	5h	Completada
HU15	Como desarrollador quiero poder hacer una comparativa después de haber migrado la pantalla de las subidas	9h	Completada
-	Documentación del rediseño del caso de uso e implementación de la migración	29h	Completada
-	Correciones finales en el documento	15h	Completada
-	Documentación del seguimiento del Sprint extra	2h	Completada
Total		**60h**	**5/5**

Tabla B.9: Tareas del Sprint extra

B.3. Resumen de la ejecución del proyecto

Tras haber finalizado el proyecto, resulta interesante realizar una comparación entre la planificación inicial y cómo se ha desarrollado realmente el proyecto. Además también resulta interesante realizar una comparativa entre el tiempo estimado que iba a llevar el desarrollo del proyecto y el tiempo empleado para poder conseguir todos los objetivos. Es por ello, que en esta sección se mostrarán todos estos puntos más con más detalle.

B.3.1. Calendarización planificada y real

El proyecto se ha desarrollado desde el día **19 de febrero** hasta el **4 de julio**, siendo una duración aproximada de 5 meses de trabajo.

La calendarización que se había estimado en las etapas iniciales del proyecto, y que se puede ver en la Tabla A.1, ha sido respetada casi en su totalidad. Al final ha sido necesario el uso del Sprint extra, ya que no se llegaba a tiempo para entregar el proyecto en las primeras fechas. La carga de trabajo de cada uno de los sprints ha sido adaptada según las circunstancias del momento, cosa que ya se tuvo en cuenta en su día a la hora de la planificación.

Algo a mencionar respecto a este tema, es que ha habido varias reuniones semanales (tanto *Scrum Weekly* como *Sprint Review*) que se han tenido que mover de día que se tenían planeadas, debido a la disponibilidad por parte de la tutora. Además, durante el Sprint 7 y el Sprint extra, todas estas reuniones se llevaron a cabo de forma remota. Esto no influyó para nada en el desarrollo del proyecto, pues la mecánica de las reuniones era exactamente la misma: mostrar los avances, dudas y cosas a mejorar del sprint en el que se estuviera trabajando o los previos al mismo.

Otro detalle, es que se utilizó el período de vacaciones de Semana Santa para avanzar el trabajo que se había estimado de modo que el Sprint 2 tuvo una duración de aproximadamente 3 semanas.

Además, se hizo un pequeño cambio en las fechas de inicio y fin del Sprint extra. En un principio se tenía estimado que comenzase el día 24 de junio y terminase el 7 de julio. Debido a que, antes del comienzo de este sprint, ya se conocía que iba a ser necesario utilizarlo, se adelantó la fecha de inicio al día 17 de junio. De esta manera la finalización del Sprint extra fue el día 1 de julio. Como consecuencia, los días que se tenían reservados para la corrección y revisión de la memoria de trabajaron se movieron a la última semana de entrega del proyecto: del 1 al 4 de julio.

B.3.2. Tiempo real estimado

Según la guía docente, la realización del TFG requiere 300 horas de trabajo, por lo que se planificó ese número de horas en caso de que se entregase en primeras fechas: 40 horas dedicadas en los 7 sprints de desarrollo y 20 horas en el sprint inicial de preparación. En caso de que se entregase en segundas fechas, se añadió un sprint extra el cual estaba estimado en 20 horas de trabajo, teniendo un total de 320h.

Sumando todo el tiempo dedicado en cada uno de los sprints desarrollados en el Apartado B.2, el cómputo total de tiempo dedicado al proyecto es de **318 horas**.

B.3.3. Costes finales

Teniendo en cuenta el tiempo de trabajo (318 horas) que se ha dedicado en el proyecto, se ha recalculado el coste real final y se ha contrastado con el presupuesto real estimado. Por otro lado el presupuesto simulado se mantiene igual ya que no ha habido una variación en los recursos que se han utilizado, el precio unitario de cada uno de ellos o incluso el período de tiempo durante el que se han utilizado.

Coste real final

A pesar de que los recursos que se encuentran en esta sección son exactamente los mismos que cuando se hizo la estimación, ha habido un ligero cambio respecto a la cantidad de kWh que se han utilizado. Teniendo en cuenta que el consumo real de un portátil en 1 hora es de 0,022 kWh (teniendo en cuenta una media de 30 días en un mes), la cantidad de total en las 318 horas de trabajo ha sido de 6,996 kWh.

Concepto	***Precio***	***Cantidad***	***Total***
Equipo de trabajo (*MSI Alpha 15*)	22,91€/mes	5 meses	114,55€
Dispositivo móvil (*Motorola Moto G14*)	2,58€/mes	5 meses	12,92€
Coste de la luz	0,1304€/kWh	6,996 kWh	0,912€
TOTAL			**128,382€**

Tabla B.10: Coste real final

El total estimado en el presupuesto real era de 128,33€, el cual al final ha resultado ser **128,382€**. Éste es ligeramente superior al que se había estimado debido a que se han empleado más horas de trabajo que las que se tenían estimadas.

Apéndice C

Resumen de enlaces adicionales

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

- Repositorio del proyecto: `https://github.com/joragua/tfg_ownCloud/tree/compose/transfers`.
- Repositorio de la aplicación Android de *ownCloud*: `https://github.com/owncloud/android`.