Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

DESARROLLO DE UN PROTOTIPO DE AGREGADOR DE ALMACENAMIENTO EN LA NUBE

Alumna

María Isabel Cabrera Bermejo

Tutor

Ángel Luis García Fernández
(Departamento de Informática)

Septiembre, 2021

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Ángel Luis García Fernández tutor del Trabajo Fin de Grado titulado: '**Desarrollo de un prototipo de agregador de almacenamiento en la nube**', que presenta Doña María Isabel Cabrera Bermejo, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2021

La alumna:	El tutor:
CABRERA BERMEJO MARIA ISABEL - 15512884M Firmado digitalmente por CABRERA BERMEJO MARIA ISABEL - 15512884M Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-15512884M, givenName=MARIA ISABEL, sn=CABRERA BERMEJO, cn=CABRERA BERMEJO MARIA ISABEL - 15512884M Fecha: 2021.09.10 08:15:39 +02'00'	Firmado por GARCIA FERNANDEZ ANGEL LUIS - 26038790F el día 10/09/2021 con un certificado emitido por AC FNMT Usuarios
María Isabel Cabrera Bermejo	Ángel Luis García Fernández

(Página intencionalmente en blanco)

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Sin especialidad
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	María Isabel Cabrera Bermejo
Fecha de asignación	11/11/2020
Descripción corta	El objetivo del TFG es realizar el análisis, diseño e implementación de una aplicación de escritorio multiplataforma que permita gestionar de forma transparente varias cuentas de proveedores de almacenamiento en la nube.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

La computación en la nube o *cloud computing* es una nueva tecnología que nos permite el acceso remoto a distintos servicios, desde acceso a software, almacenamiento de archivos hasta el procesamiento de datos a través de la red. Por tanto, nos permite utilizar diversos servicios de computación de manera virtual sin tener que conectarnos a un ordenador personal o a un servidor local. La red utilizada es comúnmente Internet debido a su amplia implantación en nuestro día a día.

Con esta nueva tecnología se aboga por el uso de servicios descentralizados con servidores remotos y software que permitan el acceso desde cualquier dispositivo en lugar de equipos o servidores locales individuales, evitando así el gasto tanto económico como de espacio e infraestructura que esto suponía antes.

Surgido como concepto básico en la década de 1960 por John McCarthy (quien introdujo también el término de Inteligencia Artificial) con la idea de "tiempo compartido" (refiriéndose a la compartición concurrente de un recurso computacional), la computación en la nube comenzó a utilizarse en algunas empresas de Estados Unidos de forma interna proporcionando recursos compartidos entre sus empleados, y a pesar de la falta de popularidad que sufrió pocos años después por la falta de avances tecnológicos tanto en hardware como en software y en comunicaciones, ha conseguido elevarse como una de las tecnologías más empleadas a día de hoy.

Existen distintos tipos de servicios computacionales que nos brinda esta tecnología y que podemos ver agrupados en los siguientes bloques:

- IaaS (Infraestructure as a Service), permite acceso a una infraestructura hardware proporcionada por un proveedor y que es compartida entre los distintos usuarios que acceden a ella.
- PaaS (Platform as a Service), proporciona una plataforma para el desarrollo de software, mantenimiento o gestión del mismo de manera virtual y sin complejidades en las capas de infraestructura o redes al no trabajar con ellas.
- SaaS (Software as a Service), orientado más al usuario final ya que permite el acceso a software alojado en la nube, ignorando tareas de mantenimiento y/o desarrollo.

La computación en la nube hace uso de servidores remotos a los que se conectan los usuarios a través de una gran variedad de dispositivos conectados a Internet, permitiendo el acceso al servicio que desee utilizar (almacenamiento, tratamiento de datos etc.). Dentro de esta tecnología y según diversos factores, existen tres tipos de nube:

- Nube pública: se trata del tipo más común de nube. Se encuentra disponible para cualquier usuario y todos los datos están alojados en los mismos servidores y sistemas de almacenamiento. Su principal característica es que todos los recursos que la forman son propiedad de un proveedor que se encarga de su mantenimiento y gestión.
- Nube privada: consta de una serie de recursos computacionales utilizados únicamente por unos usuarios concretos (como una empresa u organización), pudiendo estos gestionar la configuración de la misma según sus necesidades internas y manteniendo una mayor seguridad tanto en datos como en software.
- Nube híbrida: este modelo de nube ofrece la flexibilidad y escalabilidad de la nube pública y la seguridad de la nube privada, siendo el usuario propietario de unas partes de la nube (como en la nube privada) y compartiendo otras con el resto de usuarios (como en la nube pública). En este modelo se utilizan conjuntamente distintos modelos de implementación de nube (como puede ser pública o privada) manteniendo una organización o integración entre ellas.

- Multinube: se trata de un modelo de nube en el que se encuentran presentes al menos dos implementaciones de un modelo de nube (pública o privada) de distintos proveedores. De esta manera los usuarios obtienen una mayor expansión de sus servicios y/o organizaciones.

Dentro de la computación en la nube uno de los servicios (SaaS) más demandados a día de hoy es el almacenamiento en la nube o *cloud storage*. Este servicio permite tanto el almacenamiento como la organización y distribución de todo tipo de archivos que se encuentran en un servidor remoto, proporcionado por un proveedor, al que accedemos a través de Internet.

Al tratarse de un servicio que hace uso de la "nube" para almacenar los datos a los que acceden los usuarios, se puede clasificar de manera similar a como se ha hecho con la computación en la nube según se utilice para ello una nube pública, privada, híbrida o multinube.

A la hora de almacenar nuestros archivos en un proveedor u otro existen distintos motivos por los que podamos seleccionar una plataforma u otra. La cantidad de espacio que nos ofrecen de manera gratuita (sin tener que pagar una cuota o alquiler para mantener nuestros archivos en la nube), la seguridad con la que nuestros archivos se almacenan o la facilidad que tenemos para acceder a ellos son algunos de los factores que más se tienen en cuenta. Sin embargo, es cada vez más común que un mismo usuario haga uso de distintas plataformas de almacenamiento en la nube o de distintas cuentas dentro de una misma plataforma. Esto puede ocurrir por motivos de seguridad (para no tenerlo todo en la misma cuenta por si esta acaba siendo robada, perdida etc.), por motivos de orden (para mantener en una cuenta archivos personales y en otra archivos de trabajo) o por motivos económicos (para no estar pagando cuotas elevadas por almacenar una cantidad de archivos importante en una sola plataforma y con una única cuenta).

Debido a esta tendencia cada vez más generalizada, se ha creado una nueva necesidad entre los usuarios de este tipo de servicios: el acceso a todas las plataformas de almacenamiento en la nube en las que se almacenan algunos de sus datos de manera simultánea y a través de una única aplicación. De esta manera se genera un entorno multinube en el que gracias a una única interfaz de usuario, este

puede conectar con más de un proveedor a la vez y tener acceso a todos sus archivos independientemente de la plataforma en la que se almacenan para una mejor gestión y organización de los mismos, evitando los costos en tiempo que puede suponer para un usuario tener que buscar entre las distintas plataformas un archivo, abrir y cerrar conexiones con distintas aplicaciones etc.

1.2 Objetivos del trabajo

El objetivo final de este trabajo es realizar el análisis, diseño e implementación de una aplicación de escritorio multiplataforma que permita gestionar de forma transparente varias cuentas de proveedores de almacenamiento en la nube.

Con la creación de esta aplicación el usuario final trabajaría inmerso en un entorno multinube que le permitiría el acceso a todos sus archivos de manera fácil, sencilla e intuitiva para un mejor control y organización de los mismos, además de facilitar la búsqueda entre ellos.

La falta de herramientas que presten este servicio y la creciente popularización de la nube y del almacenamiento en la misma hacen que el desarrollo de una aplicación que cumpla estas características sea muy interesante tanto a nivel de adquisición de conocimiento, ya que la computación en la nube es una de las tecnologías más presentes a día de hoy, como a nivel de utilidad, ya que proporciona un servicio cada vez más demandado.

1.3 Aplicaciones similares

Son pocas las herramientas que encontramos disponibles en el mercado para trabajar con un entorno multinube. Podemos clasificarlas según la plataforma en la que se desarrollan.

- Aplicaciones web.
    - MultCloud[1]: se trata de una página web cuya funcionalidad principal es el control y la administración de distintas cuentas de almacenamiento en la nube, permitiendo el traspaso de ficheros entre distintas nubes, la sincronización de cualquiera de ellas, etc. (Ilustración 1) Tiene soporte para la mayoría de los proveedores de almacenamiento que podemos

[1] https://www.multcloud.com/

encontrar actualmente (Google Drive, OneDrive, Dropbox, Mega, Box etc.). Además, ofrecen una extensión de Google Chrome. Pero sin embargo, presenta bastante limitaciones y desventajas ya que contiene cuatro modelos de uso de los cuales solo uno es gratuito. Este modo tiene limitados los GigaBytes de tráfico permitidos a 30GB/mes, lo cual puede ser escaso y empeorar nuestro trabajo con la nube. Además, este modelo también tiene limitados el número de hilos de trasferencia de archivos (a dos) y permite una sola tarea de sincronización (excluyendo la simultaneidad). Si bien es cierto que existen versiones de pago (que según aumenta el precio disminuyen todas estas limitaciones), no todo el mundo se puede permitir el pago de las cuotas de este servicio.

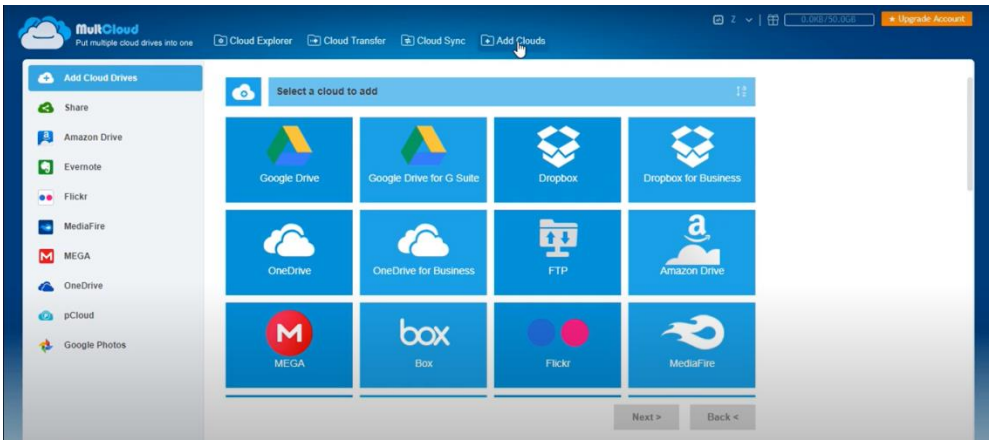

Ilustración 1: MultCloud

- Aplicaciones móviles.
  - FolderSync[2]: se trata de una aplicación disponible únicamente para Android, centrada principalmente en la sincronización de carpetas locales con el almacenamiento en la nube de distintos proveedores de manera simultánea, siendo posible la conexión tanto para la subida como para la bajada de archivos entre el dispositivo móvil y el proveedor deseado. (Ilustración 2) Consta de dos modos de uso, siendo uno de pago y otro gratuito, este último con anuncios y limitaciones a la hora de trabajar con la aplicación.

[2] https://play.google.com/store/apps/details?id=dk.tacit.android.foldersync.lite&hl=es&gl=US

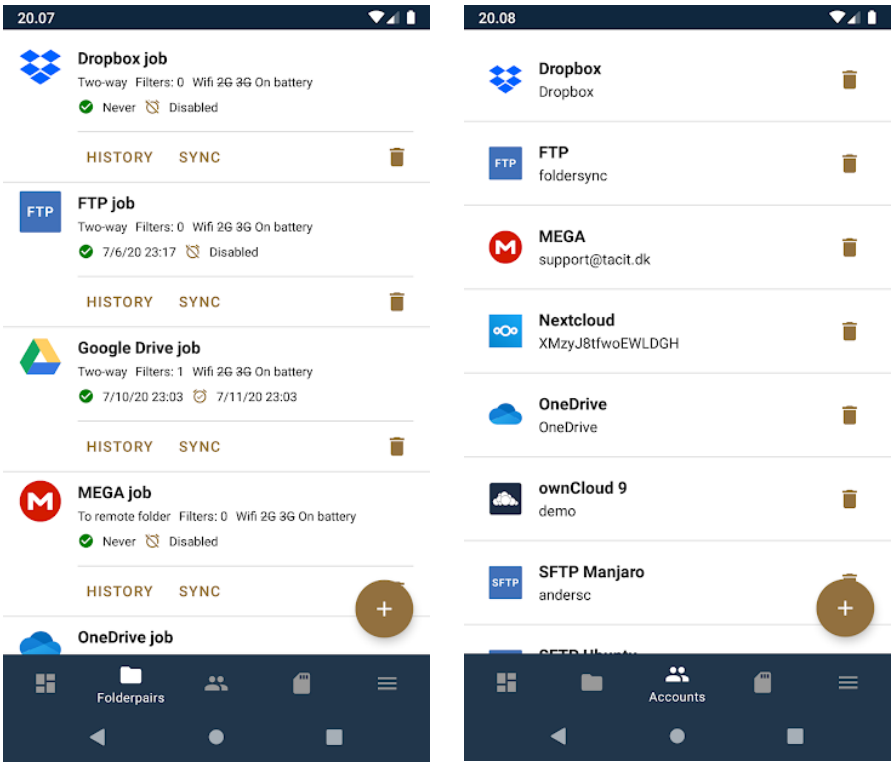

Ilustración 2: FolderSync

- Syndoc Cloud Manager[3]: es una aplicación disponible tanto para Android como para iOS, que permite el control y gestión de archivos almacenados en distintos proveedores además del traspaso de archivos entre nubes. (Ilustración 3) Existen distintas versiones: Syndoc, Syndoc Pro y Syndoc Business, siendo la primera la única gratuita, y que incluye limitaciones a la hora de realizar distintas funciones.

[3] https://syndoc.com/html/products.html

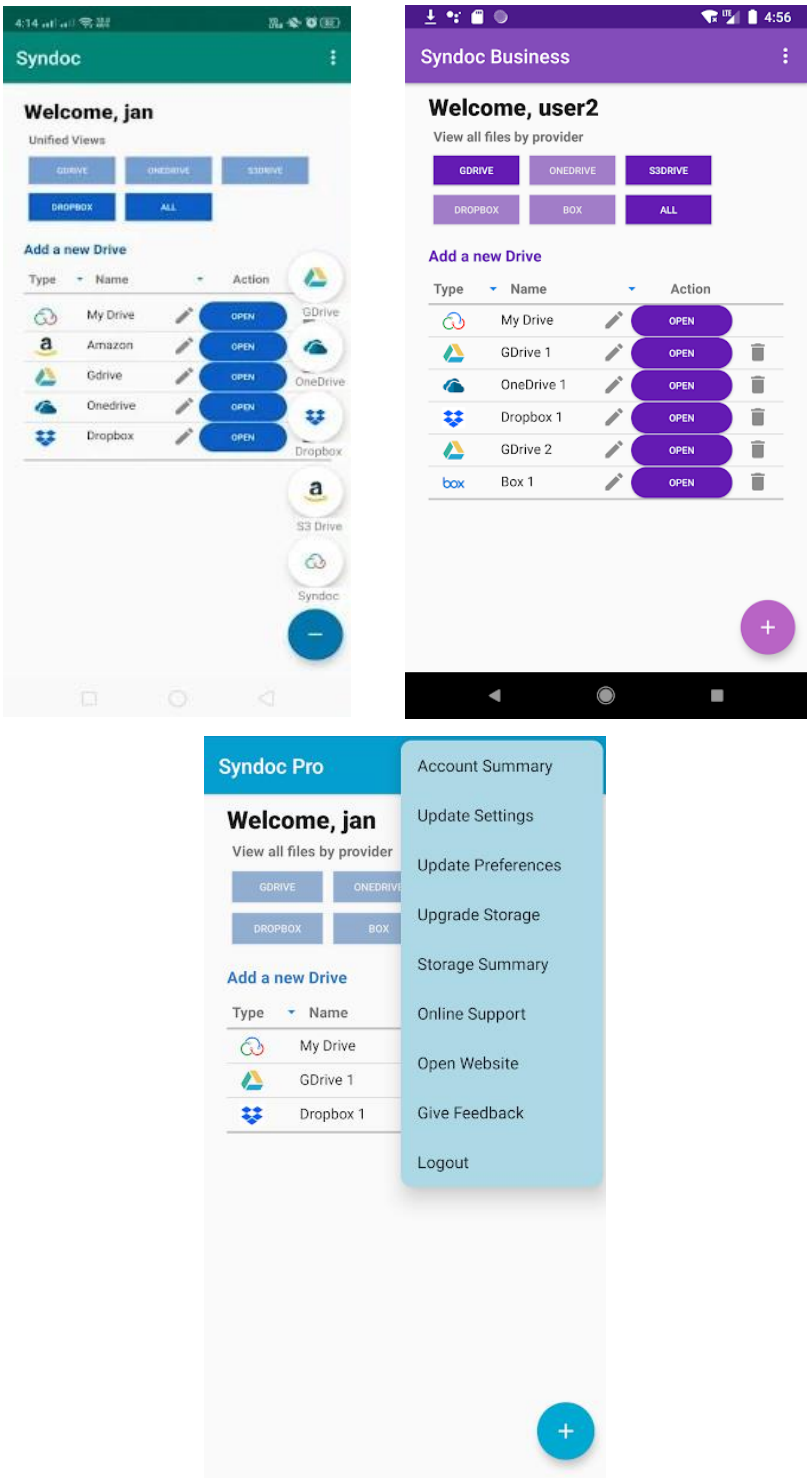

Ilustración 3: Syndoc

- Aplicaciones de escritorio. Cabe destacar una serie de tres aplicaciones creadas por una misma empresa, Iniciativas Informáticas y de Comunicación, por encima del resto de aplicaciones del mercado, ya que las versiones de pago son más asequibles económicamente y tienen un continuo mantenimiento y una mayor funcionalidad. Solo una de las tres

aplicaciones, la primera, tiene versión para Windows y MacOS, mientras que el resto solo tienen versión para Windows.

- Air Explorer[4]. Trabaja con la mayoría de proveedores de almacenamiento en la nube actualmente disponibles, permitiendo la gestión de archivos en cada nube y entre nubes, además de la sincronización. (Ilustración 4) Cuenta con una versión gratuita y distintas versiones de pago. En la primera solo se permite una cuenta por proveedor y dos subidas/bajadas simultáneas. En las versiones de pago esto no está limitado y además incluye funciones extra como el encriptado de los datos.

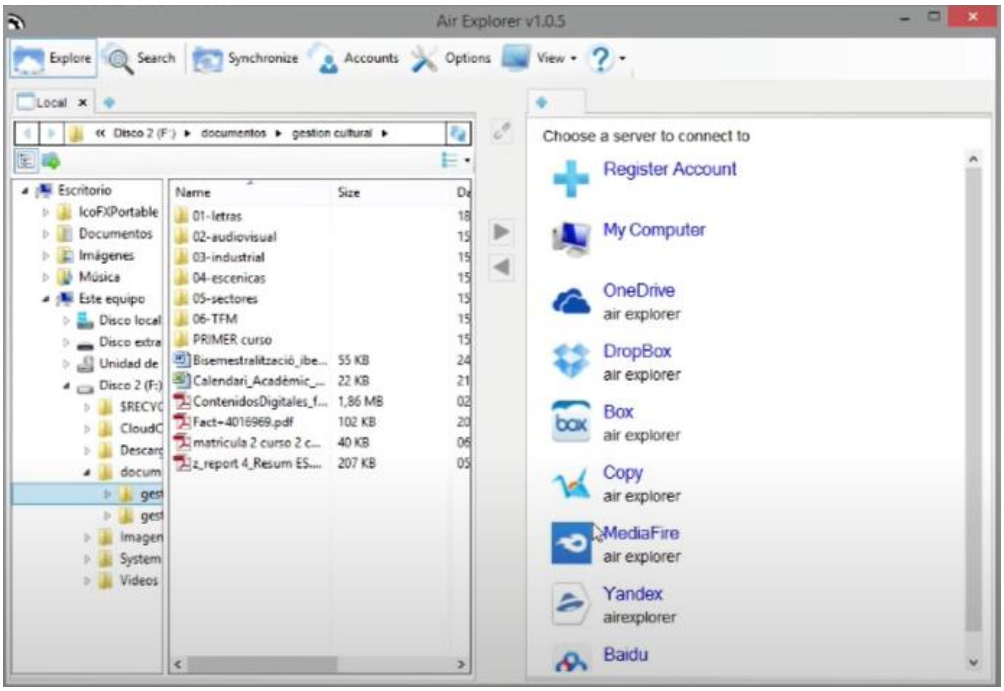

Ilustración 4: Air Explorer

- Air Live Drive[5]. La ventaja de esta aplicación respecto a la anterior es que nos permite trabajar con nuestras nubes directamente desde nuestro ordenador como discos duros internos, evitando así el proceso de sincronización. (Ilustración 5) Consta de una versión gratuita, donde solo se nos permite el uso de un disco por nube y tres discos simultáneamente, y dos versiones de pago en las que podemos incluir

[4] https://www.airexplorer.net/es/
[5] https://www.airlivedrive.com/es/

una contraseña para entrar y donde los discos simultáneos y los discos por nube no están limitados.

Ilustración 5: Air Live Drive

- Air Cluster[6]. Permite la unificación de varias nubes en una sola, de manera que se gestione el espacio conjunto de todas ellas y los archivos que contienen. (Ilustración 6) Sin embargo, la versión gratuita solo permite un máximo de dos clusters (cada uno con un máximo de tres cuentas) además de limitar el número de subidas/bajadas simultáneas a cinco. Con las versiones de pago esto no está limitado y se incluyen funcionalidades extra como el encriptado de datos, contraseña para entrar etc.

[6] https://www.aircluster.org/es/

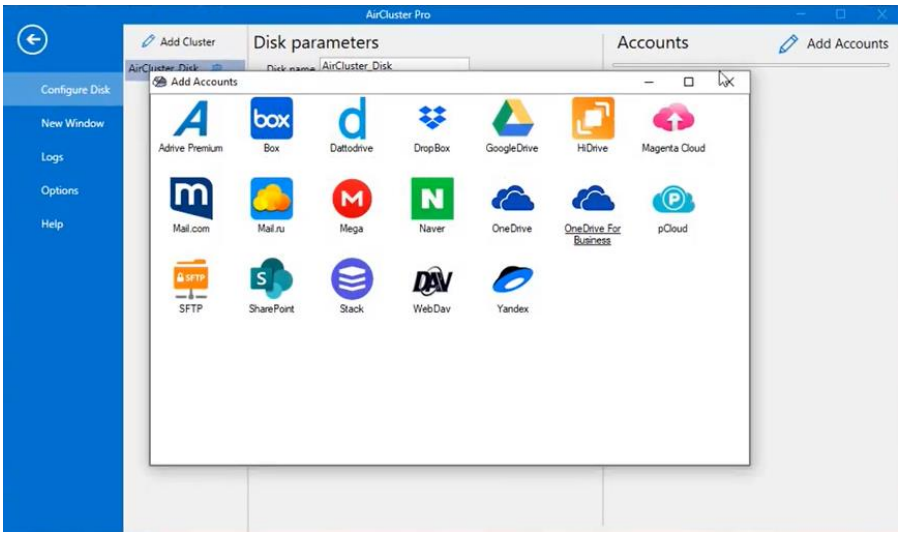

Ilustración 6: Air Cluster

Como podemos ver, la mayoría de las aplicaciones que encontramos en el mercado, independientemente de la plataforma en la que se trabaje, son aplicaciones cuyas versiones gratuitas están limitadas en cuanto a funcionalidad. Aunque esta puede ser suficiente para un pequeño uso del almacenamiento en la nube, este servicio puede quedarse corto, de manera que parte de la población quede excluida de su uso por su coste, que aunque no sea excesivamente elevado, puede suponer un impedimento para aquellas personas con menor riqueza económica, aumentando así la brecha digital.

Podríamos decir que el hecho de tener que pagar una cuota por cualquier funcionalidad un poco más avanzada es la principal carencia de este tipo de aplicaciones, ya que no solo hay pocas alternativas sino que la mayoría presta una amplia funcionalidad siempre que se pague por su uso. Por ello, en este trabajo se opta por el desarrollo de una aplicación gratuita que aunque no se va a comercializar (al no ser este el objetivo del proyecto), en caso de hacerse, estaría al alcance de cualquiera evitándose así un costo extra para una funcionalidad que cada vez se usa más en nuestro día a día.

1.4 Requisitos iniciales

La aplicación a desarrollar debe permitir al usuario trabajar bajo un entorno multinube en el que pueda tener presentes todos los archivos de las distintas

cuentas vinculadas a los proveedores correspondientes. Para ello, además debe poderse asociar más de una cuenta de un mismo proveedor.

El usuario debe poder observar los archivos de cada una de las cuentas que se encuentren ligadas a la aplicación en el momento correspondiente, ya sea de manera conjunta (mostrando todos los archivos de todas las cuentas) o de manera individual, pudiendo realizar una búsqueda de archivos por cualquiera de los atributos mostrados (nombre, cuenta a la que pertenece etc.).

Sobre cada archivo hay una serie de funcionalidades principales que se debe poder llevar a cabo: descargar, eliminar y subir (en las dos primeras acciones siempre que se tengan los permisos necesarios sobre el fichero).

Además, la aplicación debe permitir al usuario vincular o desvincular cualquier cuenta de los proveedores disponibles siempre que este lo desee, guardando en todo momento las cuentas emparejadas a la aplicación, de manera que si esta se cierra y se abre se mantenga el estado de la aplicación.

1.5 Alcance

A la finalización del proyecto, se espera obtener una aplicación totalmente funcional que permitirá al usuario realizar conexiones con más de una cuenta en al menos dos proveedores de almacenamiento en la nube distintos de manera simultánea. Además de esto, se podrán realizar distintas operaciones como búsqueda, subida, bajada o eliminación sobre cualquier archivo. Todo esto de manera sencilla e intuitiva, permitiendo desvincular y controlar cualquiera de las cuentas vinculadas en cada momento a la aplicación.

Los resultados obtenidos a la finalización de este trabajo, podrán apreciarse con más detalle en el conjunto de entregables que se describen a continuación.

- El código fuente de la aplicación, con todos los archivos necesarios para su compilación y ejecución, en el lenguaje de programación seleccionado para el desarrollo de este trabajo. El código de los distintos ficheros fuente irá documentado correctamente para una mejor comprensión de cada una de las funciones llevadas a cabo.

- Una versión ejecutable, para que la aplicación pueda ser usada por los usuarios que así lo deseen, de manera cómoda y sencilla sin la necesidad

de compilar y ejecutar el código. Si fuera necesario se incluiría un instalador para su uso.

- La memoria del TFG, es decir, este documento con todos los aspectos importantes para garantizar una comprensión íntegra de los detalles a tener en cuenta en este proyecto (motivación, metodología, desarrollo, diseño etc.). Además, se incluirá un manual de usuario para que este pueda tener una guía para un correcto uso de la aplicación.

1.6 Hipótesis y restricciones

El TFG se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

Por otro lado, dado que el desarrollo de este proyecto se centra en la unificación de distintos proveedores de almacenamiento en una misma aplicación para poder trabajar con ellos de manera homogénea y sin necesidad de utilizar distintos programas, son dos las hipótesis de partida que se definen previas a la implementación de este software, además de la limitación temporal previamente comentada.

En primer lugar, se supone como cierto que las APIs (Interfaz de Programación de Aplicaciones) de los distintos proveedores de servicios con los que se trabajará serán estables, mínimo durante el periodo abarcado dentro de la realización del TFG, permitiendo por tanto llevar a cabo las tareas necesarias sin ningún tipo de inconvenientes de esta índole.

En segundo lugar, se supone como cierto que todas las APIs con las que se interactuará para ofrecer este servicio al usuario tendrán una funcionalidad principal similar, es decir, que dejando de lado las diferencias entre ellas, todas permitirán llevar a cabo una serie de funciones base como la obtención de archivos o la eliminación o descarga de los mismos. Esto se supone cierto para poder presentar un software homogéneo que permita realizar las mismas funciones con archivos de distintos proveedores.

1.7 Estudio de alternativas y viabilidad

Para la producción de este tipo de programas, una aplicación de escritorio con interfaz de usuario, son necesarias una serie de tecnologías que nos permiten llevar a cabo todo el proceso y obtener así el producto finalizado.

En primer lugar, se estudia el lenguaje de programación con el que se escribirá el código fuente de la aplicación. Al trabajar con APIs externas, hay que tener en cuenta con qué lenguajes trabajan y cuál es el más demandado entre estas, ya que atendiendo a este criterio, estamos aumentando el número de proveedores de servicios de almacenamiento que podremos incorporar a la aplicación.

Java, C++, Swift, Python y C# son algunos de los lenguajes de programación más utilizados para la creación de aplicaciones de escritorio actualmente. Entre estos y el resto de posibilidades existentes, vamos a realizar un análisis de las APIs de los principales proveedores de almacenamiento en la nube y el funcionamiento de cada una para elegir el lenguaje más apropiado.

Para este análisis, se ha llevado a cabo una búsqueda de las APIs de los principales servicios de almacenamiento en la nube, siendo uno de ellos Google Drive. Este es uno de los más conocidos en el mundo y también de los más utilizados. Ofrece tres APIs distintas según las acciones que necesitemos llevar a cabo: el control de ficheros almacenados, es decir, realizar acciones sobre ellos, obtener información acerca de la actividad de los usuarios, y un widget que podemos incluir en nuestra aplicación para acceder a los ficheros almacenados en Google Drive. De estas tres, la más interesante para la creación de la aplicación que se está desarrollando es la primera, por tratarse de una API que permite obtener información de ficheros, realizar subidas, bajadas, etc. Se puede incorporar en distintos lenguajes de programación: Python, Java y Node.js.

Otro de los principales proveedores de almacenamiento que podemos encontrar actualmente es Dropbox. Su API nos permite incorporar distintas funcionalidades, abarcando todas las necesarias para nuestra aplicación. Se puede emplear en distintos lenguajes como Swift, Objective-C, Python, .NET, Java, JavaScript y HTTP.

Además, Box, Mega y MediaFire, todas bastantes populares a día de hoy, se pueden incorporar a una aplicación externa mediante su API. Mega permite utilizarla

solamente con C++. Box permite su inclusión con más lenguajes de programación como Java, Python, .NET, Node.js, Objective-C y Swift, además de mediante línea de comandos. De igual manera, MediaFire también permite el uso de su API a través de distintos lenguajes como Java, JavaScript, Python, PHP, C++ y Objective-C.

Sin embargo, Amazon Cloud Drive, que está experimentando un auge en popularidad en los últimos años, solo permite la utilización de su API mediante tres sistemas, a través de una RESTful API, es decir, gracias a peticiones HTTP, o por medio de una SDK (Kit de Desarrollo de Software) para Android o para iOS.

Para utilizar OneDrive desde una aplicación externa es necesario utilizar Microsoft Graph, la API que nos permite conectar y utilizar funcionalidades de distintos productos de Microsoft. Se puede utilizar en una gran variedad lenguajes como Python, PHP, Java, JavaScript, Angular, Swift, Objective-C, Node.js, .NET, React y Ruby.

De esta manera, podríamos resumir el uso de los lenguajes en las APIs de los distintos proveedores de almacenamiento en la nube en la Tabla 1.

	Google Drive	Dropbox	Box	Mega	MediaFire	Amazon Cloud Drive	OneDrive
Python	X	X	X		X		X
Java	X	X	X		X		X
Node.js	X		X				X
Swift		X	X				X
Objective-C		X	X		X		X
.NET		X	X				X
JavaScript		X			X		X
HTTP		X				X	
PHP					X		X
C++				X	X		
Angular							X
React							X
Ruby							X

Tabla 1: proveedores de almacenamiento y lenguaje de APIs

Una vez visto todo esto, es más sencillo decidir con qué lenguaje de programación se va a desarrollar la aplicación, siendo este el que nos permita utilizar cómodamente un mayor número de APIs de proveedores distintos. Tanto Java como

Python son los más repetidos (Google Drive, Dropbox, Box, MediaFire y OneDrive), tal y como se puede ver en Tabla 1. Entre estos dos, Java es finalmente el lenguaje elegido, ya que es con el que tengo una mayor experiencia, además de presentar más facilidades a la hora de la implementación de una aplicación de escritorio (con bibliotecas como Java Swing o JavaFX para el diseño de la interfaz de usuario). De esta manera, los proveedores que podrían incorporarse son los que permiten, a través de Java, acceder a la API, siendo Google Drive y Dropbox los que comenzarán integrándose en la aplicación por ser los más populares, y siguiendo a continuación con el resto de proveedores disponibles que trabajan con Java en sus APIs en la medida en que el desarrollo del proyecto lo permita. Además, el proyecto se desarrollará en todo momento teniendo en cuenta facilitar al máximo la incorporación de nuevos proveedores de almacenamiento; para ello, se documentará debidamente los pasos a seguir.

Una vez elegido el lenguaje de programación, es vital la selección de una plataforma que nos permita desarrollar la interfaz gráfica para interactuar con el usuario. Las más utilizadas a día de hoy son dos bibliotecas gráficas de Java que pueden utilizarse conjuntamente en un mismo proyecto, además de ser multiplataforma.

Java Swing es la más antigua, sustituyó a AWT (Abstract Window Toolkit, era la herramienta por excelencia para la creación de interfaces) y se convirtió en la más estable para trabajar, incluso permitiendo el uso de la arquitectura MVC (Modelo-Vista-Controlador). Sin embargo, con el paso de los años apareció una nueva biblioteca, JavaFX, creada para ser la sustituta natural de Java Swing, ya que incorpora elementos con un look & feel más evolucionado y moderno, permite la utilización de CSS (Hoja de estilos en cascada) y crea las vistas como ficheros FXML (Lenguaje de declaraciones basado en XML), implementando así directamente el patrón MVC al asignarle la clase controladora. Además, para facilitar la creación de las vistas, se puede utilizar Scene Builder como herramienta para evitar las dificultades que pueden surgir con la construcción de la interfaz gráfica mediante código.

Ya que JavaFX es la más moderna de estas bibliotecas, de la que más actualizaciones se realizan y la que más ventajas presenta, será la utilizada para la generación de la interfaz de usuario.

Finalmente, necesitamos un IDE (Entorno Integrado de Desarrollo) que nos permita utilizar el lenguaje de programación y la biblioteca elegida, facilitándonos la tarea de desarrollo de software. Aunque en el mercado encontramos una gran variedad, son tres los que soportan JavaFX: NetBeans, IntelliJ IDEA y Eclipse. De estas tres opciones, he elegido NetBeans ya que es con el que mayor experiencia y habilidades tengo a la hora de trabajar y con el que mejor me desenvuelvo, evitando así dedicar un tiempo extra al proceso de aprendizaje sobre la utilización del entorno.

1.8 Tecnologías utilizadas

Una vez revisadas las alternativas de las que disponemos en los ámbitos necesarios para el desarrollo de esta aplicación, es decir, lenguaje de programación, biblioteca de soporte para la interfaz e IDE de programación, podemos observar las seleccionadas con más detalle a continuación, junto a los agregados que nos faciliten el proceso de desarrollo.

- Java: es el lenguaje de programación multiplataforma utilizado para el desarrollo de este proyecto. Tras el análisis de las APIs de los principales proveedores de almacenamiento, se ha observado que este es el lenguaje en el que más APIs han sido desarrolladas y por tanto, su uso permite la integración de un mayor número de servicios de almacenamiento distintos. La versión utilizada ha sido la JDK 11.0.10[7].

- Gradle: es la herramienta de automatización de código que se usará para la creación de este trabajo. Ha sido elegida por ser la más moderna y la que actualmente se encuentra más en uso al haber sido creada sobre los conceptos de Ant y Maven (sus predecesoras). Además, es la utilizada en la documentación aportada sobre la API de Google Drive con Java, que será la inicial sobre la que comenzará el proyecto. De esta manera, se está facilitando el inicio de desarrollo de la aplicación mientras, de manera simultánea, se está utilizando una herramienta moderna y actualizada que presenta ventajas significativas como la rapidez en el tiempo de

[7] https://www.oracle.com/es/java/technologies/javase/jdk11-archive-downloads.html

construcción o la incorporación de scripts más breves y limpios, sobre el resto de herramientas de automatización.

- NetBeans: es el IDE que permitirá desarrollar la aplicación y generar el ejecutable. Soporta JavaFX y permite integrar Scene Builder. La versión utilizada es la 12.0[8].
- JavaFX: es una biblioteca gráfica de Java que permite crear una interfaz con un estilo moderno y personalizable al permitir la incorporación de hojas de estilo CSS, con una gran variedad de elementos y la inclusión directa del patrón de arquitectura MVC (con vistas en ficheros FXML enlazadas a sus respectivos controladores). Por todo esto y por ser la sustituta de Java Swing, ya que en un futuro esta podría quedarse obsoleta para un mantenimiento de la aplicación, ha sido elegida como plataforma para generar la interfaz de usuario. La versión empleada ha sido la versión LTS (Long Term Support) por ser la que mantendrá el soporte durante un mayor periodo de tiempo. En este caso es la 11.0.2[9].
- Scene Builder: permite la modificación de los archivos FXML de manera cómoda e intuitiva para crear la interfaz gráfica con JavaFX y evitar las dificultades y contratiempos derivados de la creación de esta mediante código. Además, puede integrarse fácilmente con NetBeans. Para ello, solo hay que instalar ambos programas en el ordenador y abrir los archivos deseados (.fxml) desde NetBeans. En caso de que haciendo esto no se ejecute Scene Builder, solo hay que ir a Tools → Options → Java → JavaFX e incluir la ruta donde está instalado el programa. La versión usada es la 15.0.1[10]. Requiere el uso de Java 11 o superior.

Además, para la elaboración de los diagramas que ilustran este documento y que son necesarios para la elaboración de un correcto diseño, se utilizarán principalmente dos herramientas, Pencil[11] y UMLet[12]. La primera de ellas es una aplicación de código abierto que proporciona los recursos necesarios para la creación de prototipos de GUI (Interfaz Gráfica de Usuario) y por tanto será útil para

[8] https://netbeans.apache.org/download/nb120/nb120.html
[9] https://gluonhq.com/products/javafx/
[10] https://gluonhq.com/products/scene-builder/
[11] https://pencil.evolus.vn/
[12] https://www.umlet.com/

la creación de storyboards que describan el diseño de la interfaz. La segunda herramienta, UMLet, también de código abierto, permite la composición de diagramas de todo tipo (de clases, de paquetes, de secuencia, de actividad, etc.), lo cual resultará significativamente útil para el diseño arquitectónico y de clases del sistema, pudiendo detallar la estructura interna del software a desarrollar.

1.9 Metodología de desarrollo de software

Alternativamente a realizar un desarrollo siguiendo la metodología clásica en cascada, he decidido aplicar un enfoque ágil, basado en iteraciones. Esto se ha debido a diversos motivos.

Una de las razones es la facilidad que presenta esta metodología para realizar cualquier tipo de cambio en el diseño a lo largo de las distintas iteraciones, en caso de ser necesario. Además, como tras la finalización de cada iteración se van cumpliendo las historias de usuario que se definirán más adelante, es posible observar visualmente, mediante diagramas Burndown, el desarrollo del proyecto y como este va progresando y cumpliendo funcionalidades, sin tener que esperar al momento final de comprobación de la metodología clásica en cascada. Esto también nos permite, además de detectar errores en una etapa temprana, poder solucionarlos con más facilidad.

Otro de los motivos para elegir esta metodología frente a otras, es que, al tratarse de un trabajo limitado por tiempo para la realización del mismo, usar una metodología ágil basada en iteraciones permite que, si a posteriori se desease incorporar una mayor funcionalidad, podría hacerse incluyendo nuevos requisitos en forma de historias de usuario y realizando un mayor número de iteraciones donde estas se vayan llevando a cabo.

Además, al final de cada iteración se obtiene una aplicación utilizable, por lo que aunque no diera tiempo a completar todas las historias de usuario, siempre se podría entregar un producto dentro del plazo indicado (aunque no cumpla con absolutamente todos los requisitos).

1.10 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFG, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas, unas 300). Por

consiguiente, los cálculos de tamaño del proyecto están supeditados al tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

Dado que se trabaja con una metodología ágil, se incluirán, en el apartado 2.1, las historias de usuario que definen las especificaciones del sistema. Estas, contienen unos puntos de historia que serán utilizados como métrica para determinar tanto el tamaño del proyecto como la dificultad estimada de las diferentes iteraciones. De estas, se realizarán tantas como sean necesarias hasta cumplir con la puntuación total del proyecto (salvo imprevistos que puedan surgir durante el desarrollo del mismo). Esta estimación puede ir cambiando según vayan surgiendo nuevas historias de usuario a lo largo de las distintas iteraciones.

1.11 Planificación temporal

Teniendo como fecha de inicio de este proyecto el 01/02/2021, se estipula un plazo de realización de 5 meses, por lo que la finalización de este proyecto queda establecida el 01/07/2021.

Debido al uso de una metodología ágil basada en iteraciones, por el propio fundamento de esta, no se detallan en la planificación temporal las especificaciones del sistema que se cumplirán en fechas concretas dado que esto irá siendo definido en las iteraciones.

Tal y como se puede ver en la Ilustración 7, el proyecto dará comienzo con unos trabajos previos que incluyen todo el proceso de análisis e investigación necesario antes de comenzar a implementar la funcionalidad que la aplicación debe llevar a cabo. Esta tarea tiene una duración de un mes y una vez finalizada se da pie a un periodo de iteraciones en el que se llevarán a cabo tantas como sean necesarias para completar con las especificaciones del sistema que serán definidas mediante historias de usuario.

Durante todo el proceso, tanto de trabajos previos como a lo largo de las distintas iteraciones, se elabora este documento para la redacción de la memoria, con todos los aspectos importantes y a tener en cuenta.

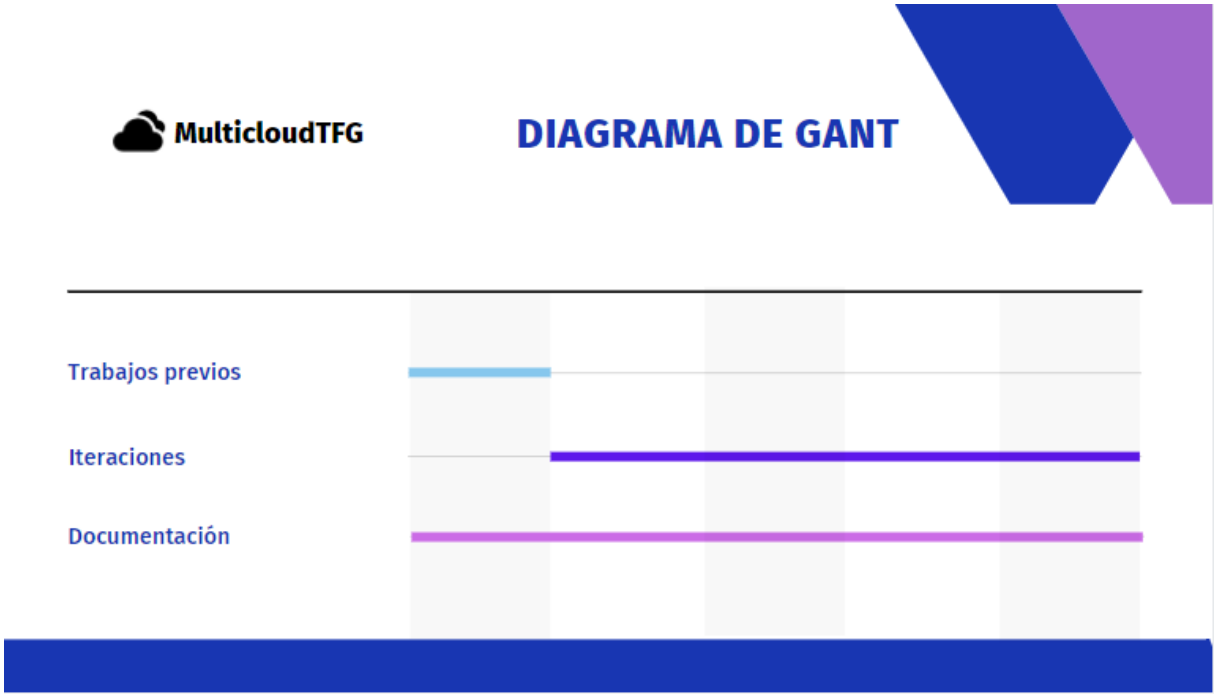

Ilustración 7: diagrama de Gant

1.12 Presupuesto

Los costes asociados al desarrollo de este proyecto se distribuyen entre el software, el personal y el hardware necesario que hagan posible la finalización de la aplicación.

- Costes hardware (Tabla 2):
    - Un ordenador que permita llevar a cabo todas las tareas de análisis y programación para completar con éxito la aplicación objetivo de este proyecto, así como la redacción de esta memoria. Para ello se ha utilizado un portátil Acer Aspire E 15 E-5-575G-7492 con un procesador Dual-Core Intel i7-7500U de 2.7Ghz, con una memoria RAM de 8GB y un almacenamiento interno de 1TB de HDD (ampliado con la incorporación de 250GB de SSD de 2.5" Samsung 870 EVO), pantalla de 15,6" con resolución 1366x768 de retroiluminación LED, tarjeta gráfica dedicada NVIDIA GeForce 940MX de 2GB y una batería de 2800mAh con 4 celdas de litio. Este equipo se obtuvo en 2017 por un precio de 772’90€ más 131’30€ por la SSD incluida como extra (904’20€ en total). Considerando un periodo de amortización de 6

años, el coste mensual del equipo es de 12’55€/mes y teniendo en cuenta un uso de 5 meses de duración del proyecto, la amortización parcial del portátil utilizado es de 62’79€.

Herramienta	Costo
Ordenador	62’79€
TOTAL	**62’79€**

Tabla 2: coste hardware de la aplicción

- Costes software (Tabla 3):
  - Utilización de todas las herramientas que han sido indicadas y descritas en el apartado 1.8.

Herramienta	Costo
Java	0€
Gradle	0€
NetBeans	0€
JavaFX	0€
Scene Builder	0€
Pencil	0€
UMLet	0€
TOTAL	**0€**

Tabla 3: coste software de la aplicación

- Costes de personal humano (Tabla 4): todos los papeles y roles que aquí se describan serán adoptados por una única persona al tratarse de un trabajo TFG. Los salarios de los profesionales que harían falta para este proyecto han sido obtenidos del convenio colectivo estatal[13].

[13] Convenio colectivo estatal: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

- Un analista que adapte y diseñe una solución adecuada para el desarrollo de una aplicación multinube. Con un salario de 23.505’72€/año en 14 pagas, el coste que este operario supondría por horas sería de 10’49€/h (suponiendo unas 40 horas semanales como jornada en la que se obtiene el cobro total indicado antes). De las 300 horas de duración del trabajo, el analista trabajará un 20% (unas 60 horas) por lo que el coste que supone es de 625’61€.
- Un programador junior que lleve a cabo la implementación de la aplicación según lo establecido en el diseño aportado por el analista, además de realizar las pruebas de verificación necesarias para verificar el correcto funcionamiento de la aplicación. Con un salario de 14.800’66€/año con 14 pagas, el coste de este trabajador supondría por horas 6’6€/h (suponiendo unas 40 horas semanales como jornada en la que se obtiene el cobro total indicado antes). De las 300 horas de duración del trabajo, el analista trabajará un 80% (unas 240 horas) por lo que el coste que supone es de 1.585’78€.

Personal	Costo
Analista	625’61€
Programador	1.585’78€
TOTAL	**2.211’39€**

Tabla 4: coste de personal humano de la aplicación

- Otros costes (Tabla 5):
  - Costes energéticos derivados del consumo del ordenador (permanentemente conectado a la red eléctrica). Teniendo en cuenta que el equipo tiene un consumo de 73’44Wh, un precio de la luz medio de 0.35€/kWh y que estará trabajando aproximadamente 300 horas, este coste supone un total de 7’71€.

Herramienta	Costo
Electricidad	7’71€

TOTAL	**7’71€**

Tabla 5: otros costes de la aplicación

Además de los gastos, hay que incluir un beneficio que se debe obtener tras la finalización de la aplicación (dado que no todo serán gastos) y el IVA que hay que pagar de impuestos, este último es el 21% del total y como beneficio supondremos un 10%.Todos los gastos se unifican en la Tabla 6 (sin IVA ni beneficios) y en la Tabla 7, donde se aprecia el presupuesto total que supondría el proyecto.

Recurso	Costo
Hardware	62’79€
Software	0€
Personal humano	2.211’39€
Otros costes	7’71€
TOTAL	**2.281,89€**

Tabla 6: coste total del proyecto sin IVA ni beneficio

Recurso	Costo
IVA (21%)	479’19€
Beneficio (10%)	228’18€
Total sin IVA ni beneficios	2.281’89€
TOTAL	2.989’26€

Tabla 7: coste total del proyecto con IVA y beneficio

1.13 Mecanismos de control de calidad

El aseguramiento de la calidad en la redacción del trabajo implica la verificación de la completitud (falta de omisiones), la integridad de la documentación, así como una redacción clara, concisa y entendible por todos los participantes e interesados en dicho trabajo.

Al estar el trabajo desarrollado por una sola persona y verificado por un/a tutor/a, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, se han utilizado mecanismos básicos para la

verificación de la integridad y completitud, que incluyen un control de versiones a nivel de documento y un control sencillo de la trazabilidad de especificaciones y requisitos.

Tanto la funcionalidad como la calidad de la aplicación serán verificadas a la finalización de las distintas iteraciones al estar trabajando con una metodología ágil basada en iteraciones. Para ello, se utilizarán los criterios de aceptación definidos junto a las historias de usuario en el apartado 2.1. Estos criterios validan el correcto funcionamiento de la aplicación con la calidad esperada de esta. Además, para dar por concluido el desarrollo de este trabajo, deben verificarse todos los criterios de aceptación que todo el conjunto de historias de usuario contenga.

2 DISEÑO INICIAL

En las siguientes subsecciones se hace una descripción del diseño inicial previo a la primera iteración. Incluye una definición de las especificaciones del sistema en forma de historias de usuario, además de un diseño arquitectónico con los diagramas de clases que estructuran el funcionamiento del software y cómo este se desarrollará al comienzo de la implementación del mismo.

Todo el contenido de este primer diseño podría ser modificado en las distintas iteraciones siempre que sea necesario.

2.1 Especificaciones del sistema

Se definirán las especificaciones del sistema en forma de historias de usuario, esto se ha decidido así ya que utilizar esta técnica permite definir desde la perspectiva del usuario la funcionalidad que debe cumplir la aplicación. Además, con ayuda de los story points (o puntos de historia) asignados a cada historia en base a la dificultad estimada, se podrá comprobar el ritmo de trabajo tras la finalización de cada entregable mediante diagramas Burndown. De igual manera, las historias de usuario irán siendo asignadas a las distintas iteraciones dividiendo el trabajo equitativamente, de manera que tras terminar todas las iteraciones se hayan llevado a cabo todas las historias de usuario, cumpliendo correctamente con los criterios de aceptación de las mismas.

La puntuación de cada una de las historias que se detallan a continuación se ha fijado realizando una estimación del coste en dificultad y tiempo utilizando la serie de Fibonacci (escala creciente 1, 2, 3, 5, 8, 13…) como medidor de puntuación para establecer la dificultad de las historias de usuario de manera realista, marcando la diferencia entre tareas pequeñas y grandes, antes de comenzar con el desarrollo de la aplicación.

Hay que tener en cuenta que en la aplicación a desarrollar solo existe un perfil de usuario, por lo que, todos los usuarios tienen las mismas características y no existe ningún tipo de distinción por grupos o categorías.

HU - 1	Visualización de archivos unificados	3 puntos
COMO usuario QUIERO poder ver todos los archivos de todas mis cuentas juntos PARA tener acceso a todos los archivos sin tener que entrar en la aplicación web de cada proveedor con cada cuenta		
Criterios de aceptación: • CA - 1.1: la aplicación debe permitir ver una lista unificada con todos los archivos de las cuentas de almacenamiento. • CA - 1.2: al abrir la aplicación se muestran todos los archivos de todas las cuentas de almacenamiento.		

Tabla 8: HU-1 Visualización de archivos unificados

HU - 2	Visualización de archivos de una sola cuenta	2 puntos
COMO usuario QUIERO poder ver los archivos de cada una de mis cuentas de manera individual PARA tenerlos clasificados según la cuenta a la que pertenecen		
Criterios de aceptación: • CA - 2.1: la aplicación ofrece la opción de ver los archivos de una sola cuenta • CA - 2.2: al elegir una cuenta, la lista de archivos muestra solo los archivos de esa cuenta		

Tabla 9: HU-2 Visualización de archivos individualmente

HU - 3	Acciones sobre archivos: subir, bajar y eliminar	8 puntos
COMO usuario QUIERO poder subir, descargar y eliminar cualquier archivo desde la aplicación PARA evitar tener que acceder a otras aplicaciones para realizar estas acciones		
Criterios de aceptación: • CA - 3.1: desde la ventana de archivos, la aplicación dispondrá de un modo de subir nuevos archivos, permitiendo elegir la cuenta de almacenamiento a utilizar. • CA - 3.2: al subir un archivo, la lista de archivos se actualiza, incluyendo el nuevo archivo subido • CA - 3.3: desde la ventana de archivos, se podrá descargar cualquiera de los archivos listados (si es posible) y guardarlo en el ordenador. • CA - 3.4: desde la ventana de archivos, se podrá eliminar cualquiera de los archivos listados (si es posible). • CA - 3.5: después de seleccionar el borrado de un archivo, la lista de archivos se actualiza y el archivo eliminado deja de aparecer		

Tabla 10: HU-3 Acciones sobre archivos: subir, bajar y eliminar

HU - 4	Búsqueda de archivos	3 puntos
COMO usuario QUIERO poder buscar entre todos los archivos PARA encontrar fácilmente cualquier archivo		
Criterios de aceptación: • CA - 4.1: en la vista de la lista de archivos hay un control para realizar una búsqueda • CA - 4.2: al realizar una búsqueda utilizando el control, solo se muestran los archivos que cumplen con el criterio indicado • CA - 4.3: al limpiar los criterios de la barra de búsqueda, se vuelve a mostrar		

la lista completa de archivos

Tabla 11: HU-4 Búsqueda de archivos

HU - 5	Vinculación de múltiples cuentas	3 puntos
COMO usuario QUIERO poder vincular múltiples cuentas de un proveedor de almacenamiento en la nube a la aplicación PARA tener acceso a los archivos de todas mis cuentas de dicho proveedor		
Criterios de aceptación: • CA - 5.1: la aplicación dispondrá de una opción que permita vincular una cuenta del proveedor de almacenamiento (pudiendo realizar esta acción tantas veces como queramos). • CA - 5.2: una vez añadida la nueva cuenta, la lista de archivos se actualiza, incluyendo los de la añadida		

Tabla 12: HU-5 Vinculación de múltiples cuentas

HU - 6	Visualización de cuentas vinculadas	5 puntos
COMO usuario QUIERO poder ver las cuentas que tengo vinculadas a la aplicación PARA tener un control de las mismas y poder saber cuáles tengo en cada momento		
Criterios de aceptación: • CA - 6.1: la aplicación dispondrá de una opción para ver todas las cuentas que tengo vinculadas a la aplicación, a qué plataforma pertenecen e información relevante. • CA - 6.2: en el menú de la aplicación, se dispondrá de tantas opciones como cuentas haya vinculadas a la aplicación para acceder a ellas de manera independiente.		

Tabla 13: HU-6 Visualización de cuentas vinculadas

HU - 7	Desvincular cuentas	3 puntos
COMO usuario QUIERO poder desvincular cualquier cuenta de la aplicación PARA no tener acceso a sus archivos desde la aplicación		
Criterios de aceptación: • CA - 7.1: la aplicación dispondrá de una opción para poder desvincular cualquiera de las cuentas que se encuentran enlazadas a la aplicación. • CA - 7.2: al eliminar una cuenta, los archivos de la misma dejarán de aparecer en el listado de archivos		

Tabla 14: HU-7 Desvincular cuentas

HU - 8	Guardar estado de la aplicación	5 puntos
COMO usuario QUIERO que al volver a abrir la aplicación, las cuentas vinculadas lo sigan estando PARA no tener que vincular todas las cuentas cada vez que cierro y abro de nuevo la aplicación		
Criterios de aceptación: • CA - 8.1: cada vez que se abra la aplicación, aparecerán todas las cuentas que estaban vinculadas antes de cerrarla la última vez en la opción que nos permita ver el listado de todas nuestras cuentas enlazadas. • CA - 8.2: al abrir la aplicación aparecerán tantas opciones en el menú como había antes de cerrar la aplicación la última vez, es decir, tantas opciones en el menú como cuentas había (y hay) vinculadas a la aplicación. • CA - 8.3: al abrir la aplicación aparecerá la vista de archivos (que es la que se muestra inicialmente al abrir la aplicación) mostrando todos los archivos de las cuentas que estaban (y continúan estando) vinculadas al software antes de cerrarlo la última vez.		

Tabla 15: HU-8 Guardar estado de la aplicación

2.2 Análisis y diseño del sistema

Una vez definidas las historias de usuario para fijar las especificaciones en cuanto a funcionalidad que el sistema debe comprender, se procede a determinar el diseño, tanto a nivel arquitectónico como de la interfaz de usuario.

Al trabajar con una metodología ágil basada en iteraciones, todo lo expuesto en las siguientes subsecciones hace referencia a una primera instancia global, es decir, a un diseño previo a la realización del trabajo y que, por tanto, en caso de ser necesario, podrá ser modificado a lo largo de las distintas iteraciones. En cada momento en que sea necesario cambiar este diseño, se describirá en la iteración correspondiente el cambio, así como los motivos que han llevado a hacerlo.

2.2.1 Diseño de la interfaz y storyboards

La descripción de la interfaz de usuario se ha llevado a cabo mediante un primer boceto o sketch en el que se puede ver como se desarrollarán las distintas funciones para las que se crea la aplicación y a las que hay que dar cabida. En las ilustraciones 8, 9, 10, 11, 12 y 13 se muestra este boceto con más detalle, observando además, la inclusión, en cada diseño de ventana, de comentarios que aclaran los componentes y su funcionamiento.

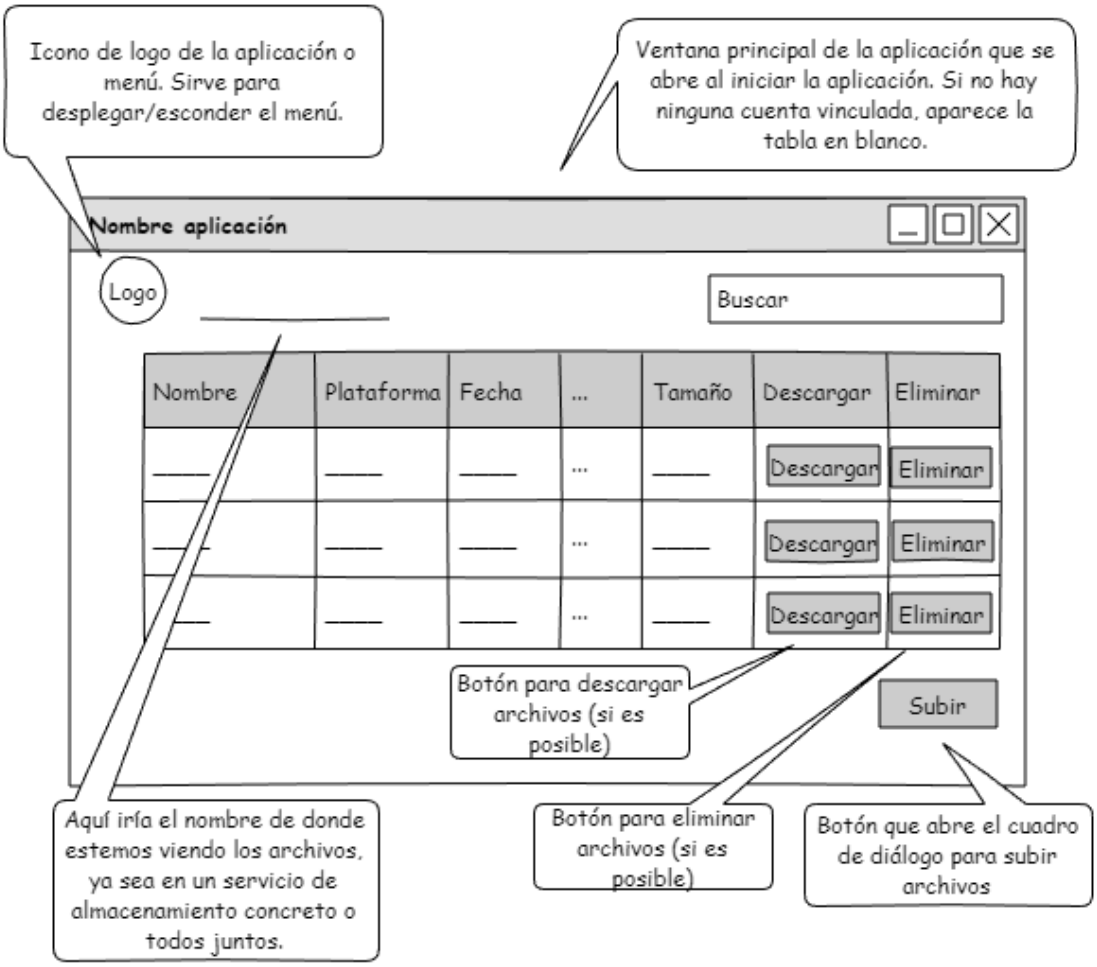

Ilustración 8: ventana principal

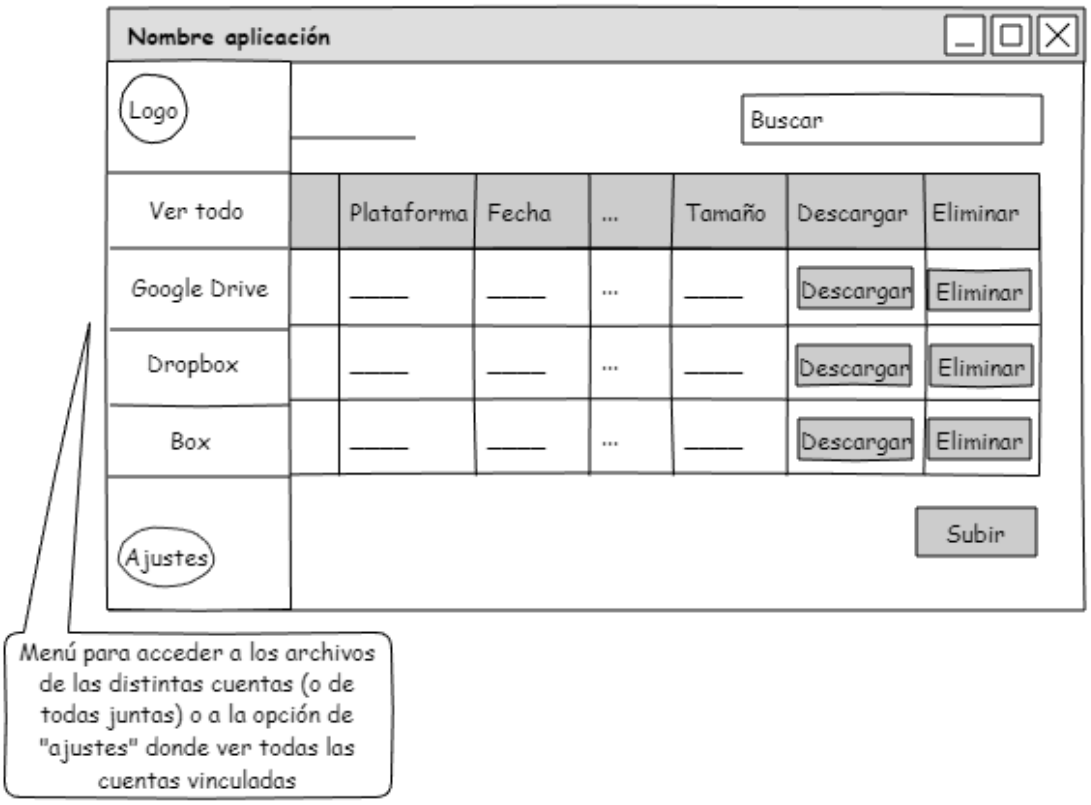

Ilustración 9: ventana principal con menú desplegado

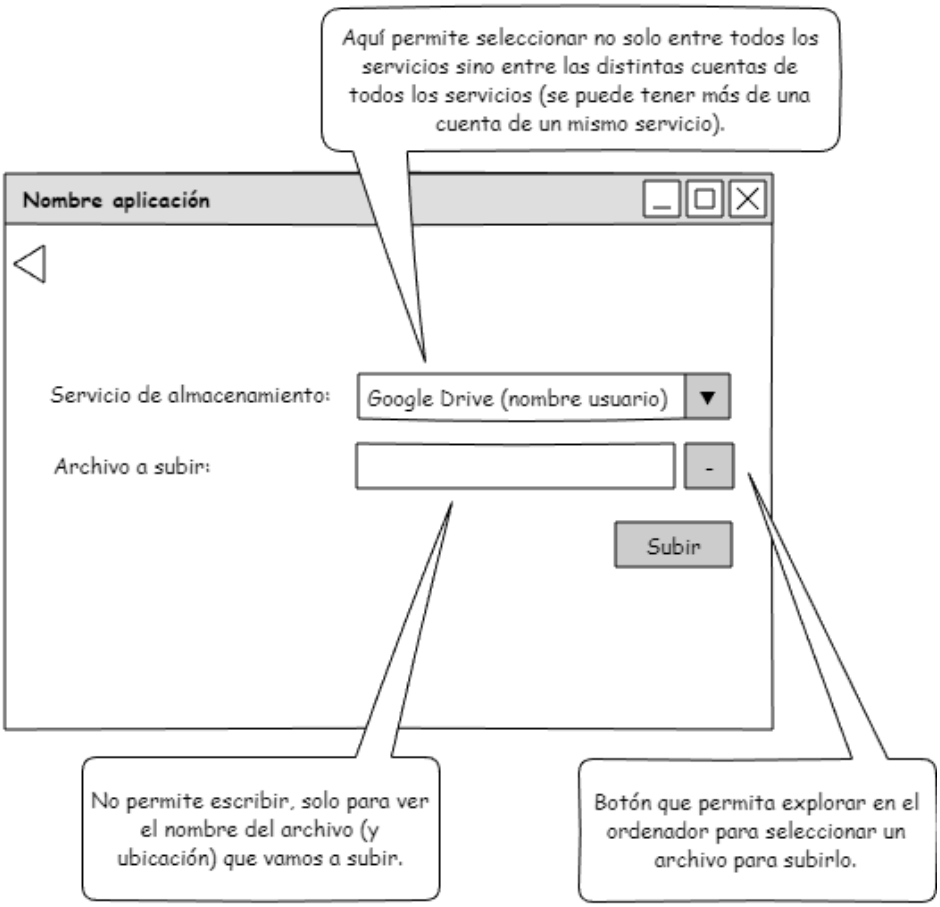

Ilustración 10: cuadro de diálogo para subir archivo

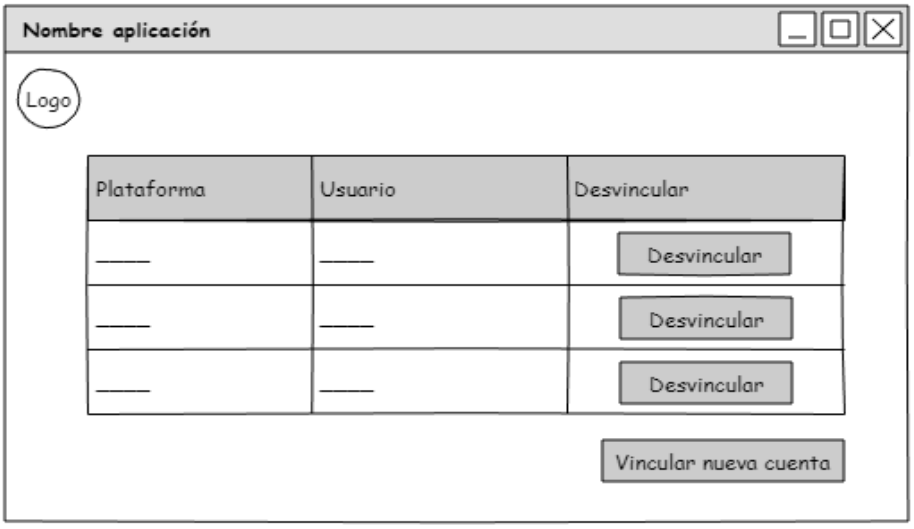

Ilustración 11: ventana de ajustes

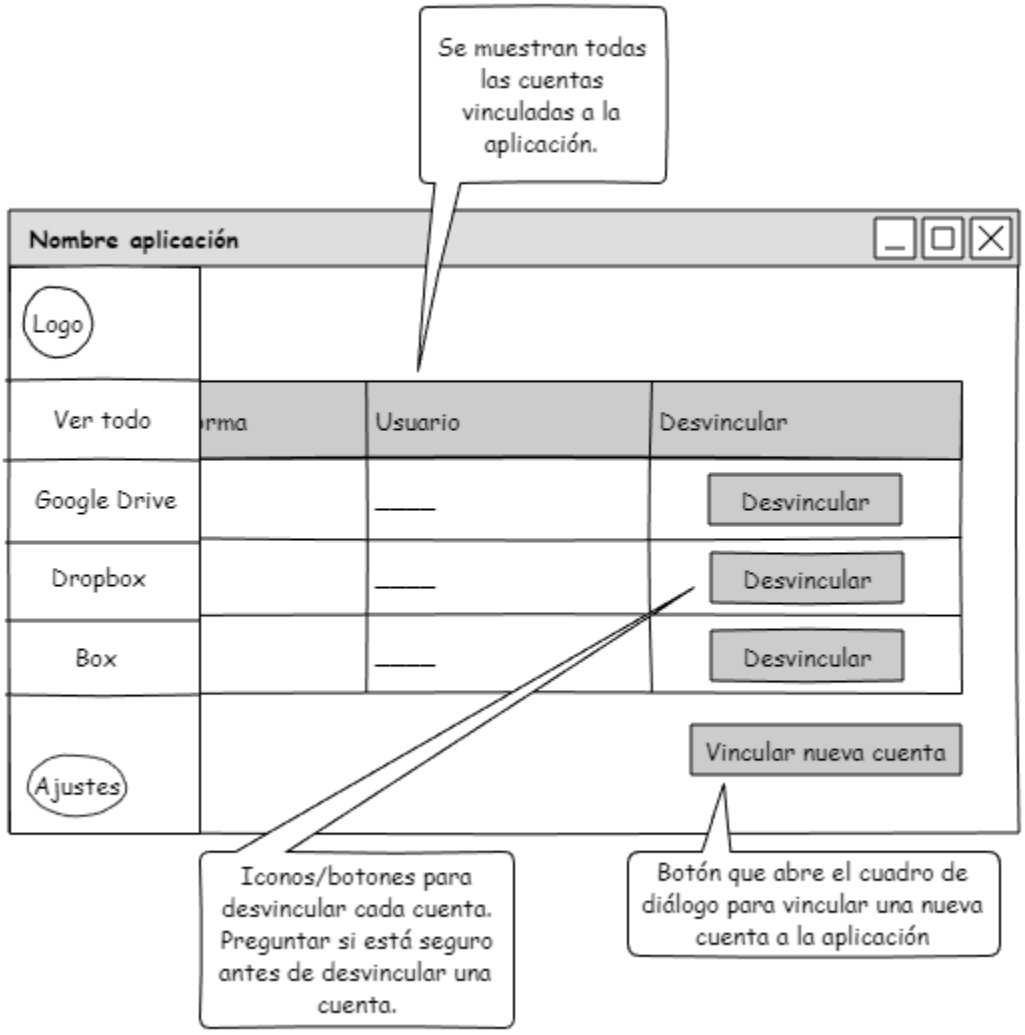

Ilustración 12: ventana de ajustes con menú desplegado

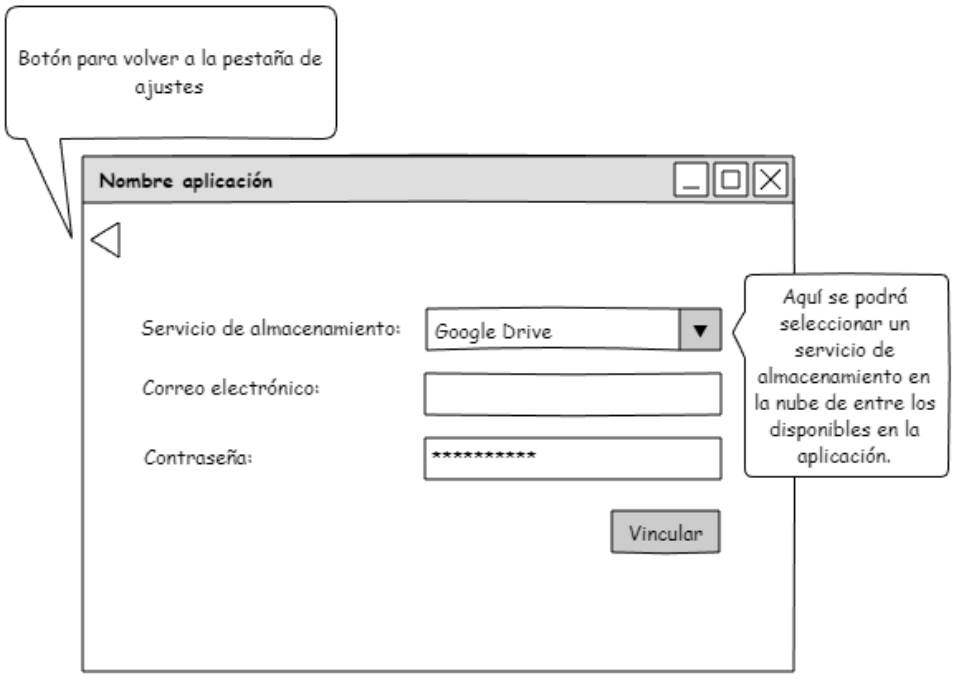

Ilustración 13: cuadro de diálogo para vincular una nueva cuenta

2.2.2 Diseño arquitectónico del sistema

El diseño arquitectónico nos permite definir cómo debe organizarse el sistema, así como su estructura global. Como patrón a seguir, se ha elegido el MVC (Modelo-Vista-Controlador) para desarrollar la aplicación de manera modular y mantenible, facilitando de esta manera la posterior introducción de nuevas funcionalidades en caso de desearlo tras el trabajo realizado durante este TFG.

Sabiendo esto, en primer lugar se han identificado los principales componentes que estructurarán el sistema, clasificándolos según la función que llevarán a cabo dentro del mismo. De esta manera, se ha generado un diagrama de paquetes (Ilustración 14) donde se establecen las conexiones entre los modelos, las vistas y los controladores, dado que son las tres capas en las que se estructura el patrón MVC, para un correcto funcionamiento.

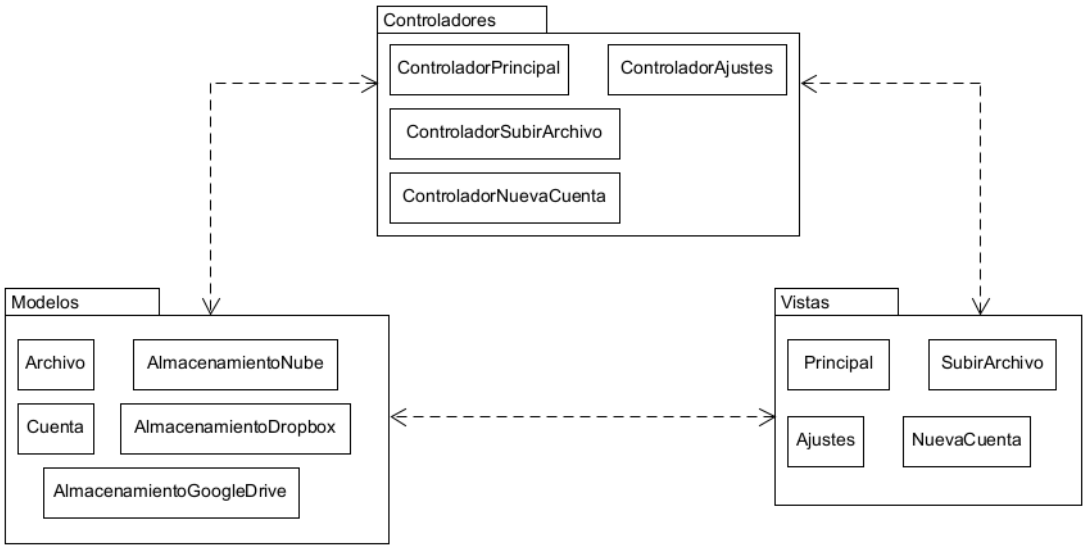

Ilustración 14: diagrama de paquetes

Estando ya separadas las clases de las distintas capas que componen este patrón, se crean los diagramas de clases correspondientes dentro de cada uno de estos paquetes mostrando las relaciones entre los distintos objetos de las clases de una misma capa. Las ilustraciones 15, 16 y 17 muestran estos diagramas de clases.

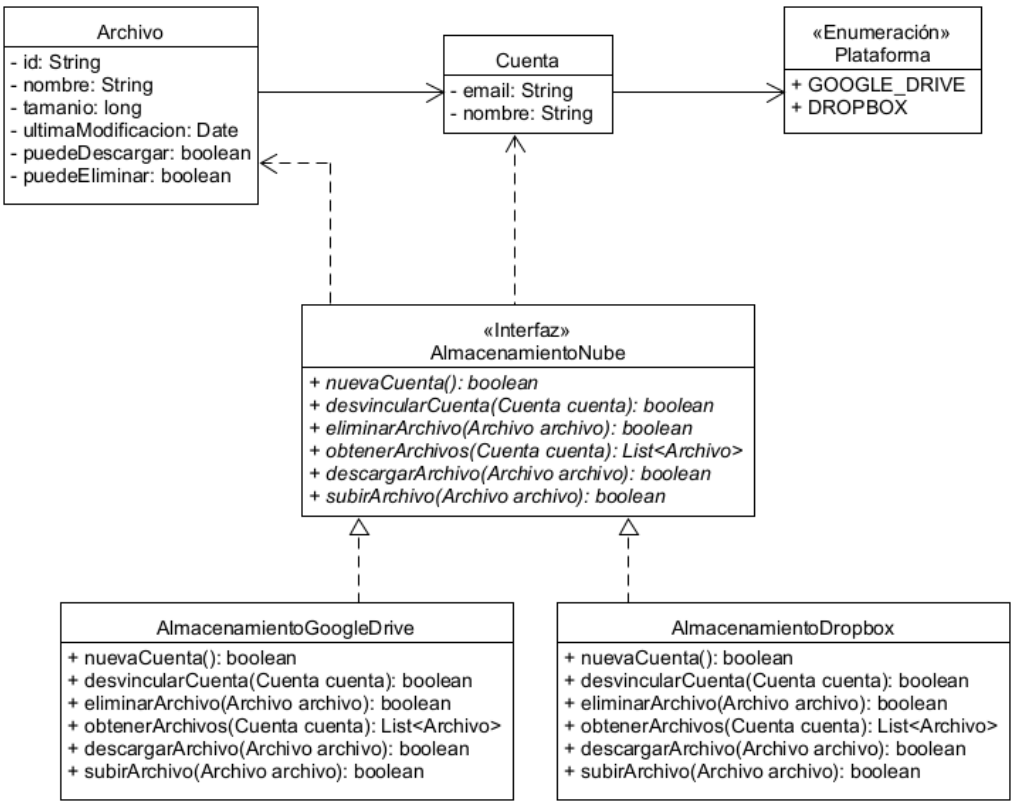

Ilustración 15: diagrama de clases del paquete Modelos

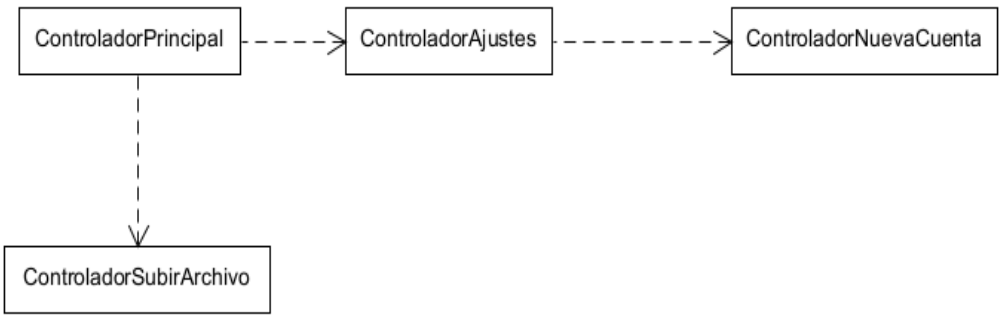

Ilustración 16: diagrama de clases del paquete Controladores

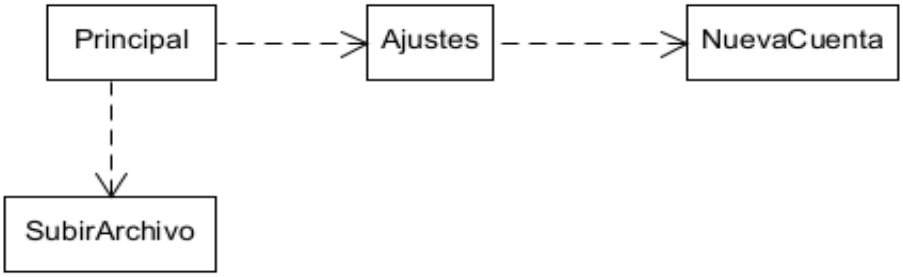

Ilustración 17: diagrama de clases del paquete Vistas

Como se puede apreciar, tanto en el diagrama de clases del paquete *Controladores* (Ilustración 16) como del paquete *Vistas* (Ilustración 17) no se han rellenado las clases con ningún tipo de atributo o método. Esto es debido al tipo de objetos con los que estamos tratando. En primer lugar, los objetos de las clases del paquete *Controladores* son aquellos que llevarán a cabo la funcionalidad de cada una de las vistas, siendo por tanto sus métodos y atributos los asociados a dicha vista (se puede decir que los atributos corresponden a los componentes que aparecen en la interfaz de usuario y los métodos corresponden a la funcionalidad, por ejemplo, al pulsar un botón).

De igual manera, los objetos de las clases del paquete *Vistas* tienen como atributos los componentes de la interfaz de usuario de la vista correspondiente y ningún método (al ser cada vista un fichero FXML que contiene únicamente los objetos que la conforman, debido a la utilización de JavaFX para el desarrollo de la interfaz).

3 DESARROLLO

Al trabajar con una metodología ágil basada en iteraciones, todos los aspectos relativos al desarrollo del software que se está llevando a cabo se detallarán a lo largo de distintas iteraciones, cada una de ellas descrita en las siguientes subsecciones. En cada una de estas iteraciones se pretende dar solución a historias de usuario (repartidas según su puntuación para que todas las iteraciones tengan una dificultad estimada similar) cumpliendo con los criterios de aceptación de

las mismas que ratifican un correcto funcionamiento de la aplicación, obteniendo un software funcional tras la finalización de cada iteración.

3.1 Trabajos preliminares

La primera iteración puede considerarse como la iteración cero o trabajos preliminares previos al desarrollo de la aplicación, ya que en ella se realizan las primeras labores para la puesta en marcha de este proyecto, siendo por tanto ligeramente distinta al resto y con una duración estimada de un mes.

En esta iteración inicial se realiza una pequeña tarea de investigación acerca de los tipos de servicios en la nube, las nubes existentes, las aplicaciones que podemos encontrar en el mercado y un análisis de las APIs de los proveedores de almacenamiento en la nube más demandados. Además, se elabora el diseño de un boceto inicial de la interfaz de usuario con toda la funcionalidad esperada y una primera conexión con un proveedor de almacenamiento en la nube.

Para iniciar el desarrollo de un prototipo de agregador de almacenamiento en la nube es vital comenzar definiendo conceptos como la computación en la nube, los tipos de servicios que esta nueva tecnología ofrece y los tipos de nubes existentes. Todo este conocimiento e información adquirida está incluida de manera resumida en el apartado 1.1 para una mejor y más sencilla comprensión de los términos abordados.

Una vez definidos los objetivos de este trabajo, se lleva a cabo una búsqueda de las aplicaciones que proporcionan un servicio similar, clasificándolas según la plataforma para la que han sido desarrolladas: web, móvil o escritorio; y analizando las ventajas y desventajas para detectar las carencias que se podrían solventar, siendo la principal el pago por uso que plantean como modelo de negocio todos los programas existentes, como se puede leer más detalladamente en el apartado 1.3.

Una vez hecho esto, y al tratarse de un proyecto que permite trabajar con distintos proveedores de almacenamiento en la nube, es necesario obtener información de las herramientas que estos nos proporcionan para hacer uso de sus servicios desde una aplicación externa. Esto nos permite ver las similitudes y diferencias que presentan para seleccionar los proveedores que son compatibles

entre sí y que se podrían unificar dentro de una misma aplicación. Para ello, existen las APIs que nos permiten utilizar desde nuestro propio software otro distinto con una capa intermedia de abstracción. Dentro del apartado 1.7 se lleva a cabo un análisis del funcionamiento de las APIs de los proveedores de almacenamiento en la nube más demandados a día de hoy. Esto es necesario para seleccionar el lenguaje de programación a utilizar, de manera que nos permita la inclusión de un mayor número de servicios. La elección del lenguaje de programación condiciona, como no podría ser de otra manera, la elección de otras herramientas como el entorno de desarrollo o las herramientas de automatización de proyectos. Todas las alternativas que se han tenido en cuenta, y sus respectivas valoraciones se pueden observar más detalladamente en el apartado 1.7. Finalmente, las que se utilizarán están brevemente descritas en el apartado 1.8, y una vez que estas sean instaladas, se puede comenzar con las tareas de implementación, realizando una primera conexión con uno de los proveedores. Se comenzará con Google Drive ya que es el más demandado y el que aporta una mayor documentación.

Para utilizar la API que nos ofrece, es necesario registrar la aplicación, creando un proyecto dentro de Google Cloud Platform[14] y habilitando la API de Google Drive desde la sección “API y servicios”. Después se crean las credenciales de tipo “ID de clientes OAuth 2.0” desde esta misma sección, especificando que se trata de una aplicación de escritorio (ya que, tal y como se explica en la documentación de la API, las credenciales son las que identifican a la aplicación, permitiendo a los usuarios finales autenticarse para acceder[15]). Una vez creadas, se descargan y, siguiendo el ejemplo expuesto en el tutorial de Google Drive[16], se guardan en la carpeta indicada, se inicializa el proyecto con Gradle como herramienta de automatización de compilación del código fuente, y se añaden las dependencias necesarias para utilizar las bibliotecas proporcionadas por la API y que nos permiten acceder a la funcionalidad necesaria.

Finalmente, y siguiendo el ejemplo, se introduce el código modificando el nombre de la aplicación y la ruta del fichero de credenciales para que al ejecutarlo funcione correctamente. Se comprueba que el funcionamiento es el adecuado

[14] Google Cloud Platform: https://cloud.google.com/
[15] Tipos de credenciales Google: https://cloud.google.com/docs/authentication?hl=es
[16] Tutorial Google Drive con Java: https://developers.google.com/drive/api/v3/quickstart/java

mostrando los diez primeros archivos de la cuenta con la que realizamos la conexión, dando por finalizada esta toma de contacto con la API de Google Drive. De esta manera, al solicitar los permisos al usuario, se nos muestra una alerta indicando una falta de verificación, tal y como se puede ver en la Ilustración 18. Por el momento y dado el estado en el que nos encontramos actualmente, pondremos la aplicación en un estado de prueba (dentro de la sección de "APIS y servicios", en "Pantalla de consentimiento de OAuth" en la plataforma de Google Cloud), pasando a publicar la aplicación una vez que esta esté finalizada y verificada (en caso de ser necesaria su publicación).

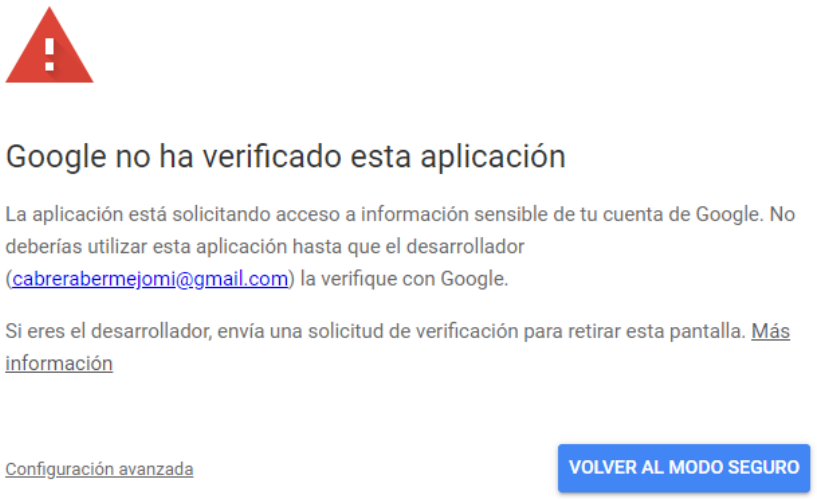

Ilustración 18: aplicación no verificada

Como parte final de esta iteración se realiza una interfaz de usuario con un único botón que lance la funcionalidad anteriormente introducida. Para ello, siguiendo el tutorial de JavaFX[17], se incluyen los módulos, plugins y la versión a utilizar de JavaFX en el fichero build.gradle. Una vez creada la ventana de prueba con su correspondiente controlador, fichero .fxml y .css, siguiendo los pasos y tras comprobar su correcto funcionamiento, se introduce un botón añadiéndole como funcionalidad la conexión previamente establecida con Google Drive y verificamos que al pulsar el botón se realiza la conexión de manera adecuada.

De esta manera se puede dar por terminada esta primera toma de contacto con el proyecto, con una conexión inicial con Google Drive mediante una interfaz de usuario básica.

[17] Tutorial JavaFX con Gradle en NetBeans: https://openjfx.io/openjfx-docs/#IDE-NetBeans

3.2 Primera iteración

Una vez establecida una conexión inicial con Google Drive, en esta iteración se completarán las tareas que se deben poder realizar con este proveedor de almacenamiento en la nube, es decir, permitir la conexión de más de una cuenta de manera simultánea y la obtención de todos los archivos que contienen, además de poder realizar todas las acciones esperadas sobre estos. Incluyendo una interfaz de usuario base, que se mejorará posteriormente, pero servirá para poder apreciar el funcionamiento correcto de la aplicación y la funcionalidad que se va desarrollando. También se implementará el sistema de guardado para almacenar las cuentas vinculadas en todo momento. Esta iteración tiene una duración aproximada de un mes.

3.2.1 Cambios de diseño

Ha sido necesario incluir una nueva clase llamada *FilaTabla* dentro del paquete *Modelos*, para poder incluir y modificar los datos de la tabla de manera dinámica, en tiempo de ejecución. Esta clase (Ilustración 19) contiene la cuenta y el archivo correspondiente a cada fila, para obtener la información propia y poder rellenar los datos. Además, también se incluyen dos botones para las funciones de eliminar y descargar que serán creados siempre que se pueda realizar la acción, así como la imagen correspondiente a la plataforma a la que pertenece el archivo. Finalmente se han incluido dos métodos públicos que establecen la funcionalidad necesaria (teniendo en cuenta el almacén al que sea necesario acceder) para la eliminación y descarga de archivos y que serán llamados cuando se esté rellenando la tabla para definir dicha funcionalidad en caso de poderse llevar a cabo cada una de estas tareas.

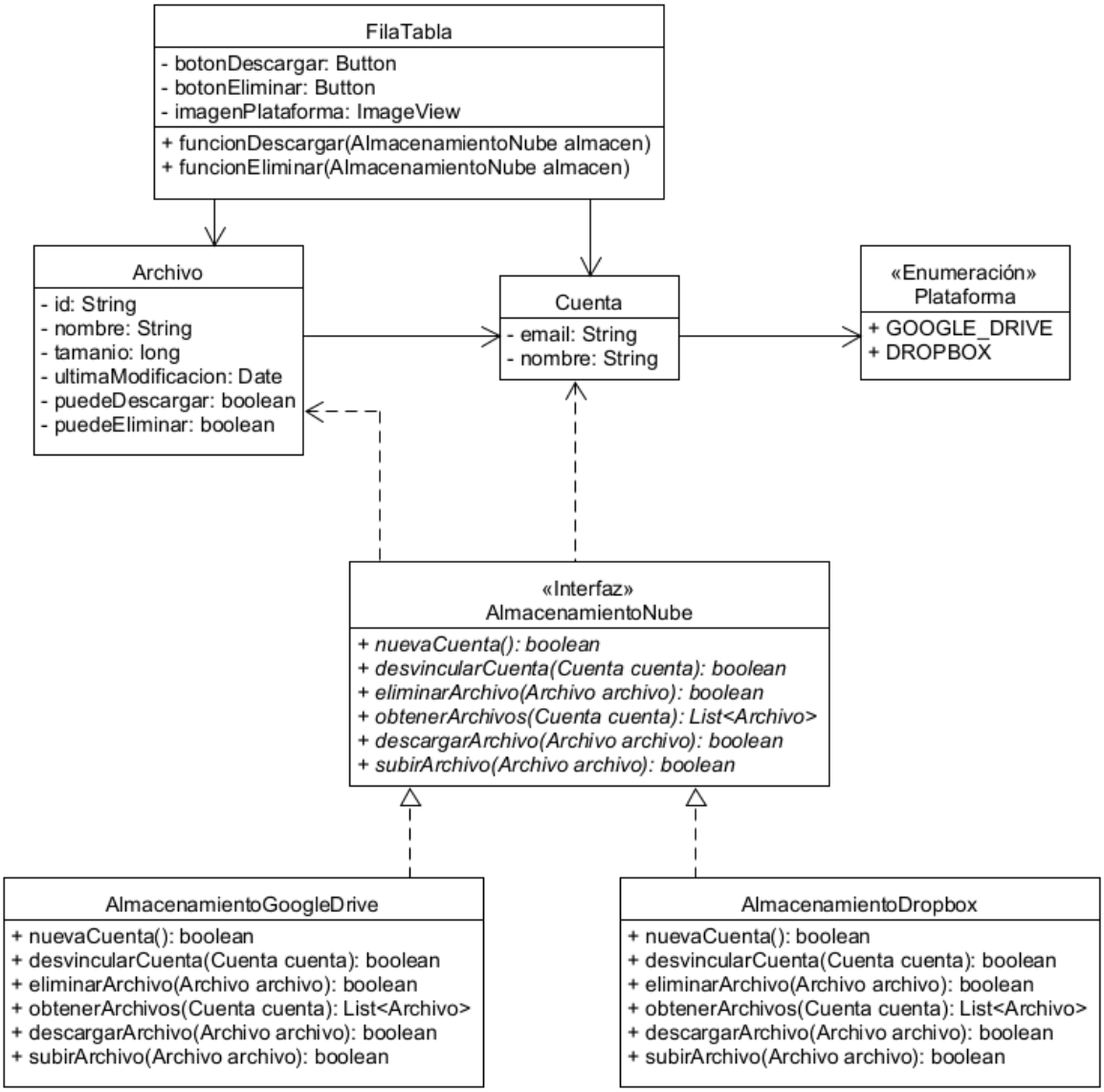

Ilustración 19: diagrama de clases del paquete Modelos modificado (FilaTabla)

3.2.2 Historias de usuario

Los objetivos a cumplir en esta iteración son los reflejados en las historias HU-3, HU-5 y HU-8 que se detallan en el apartado 2.1. Además, para poder apreciar los resultados y visualizar los archivos de forma correcta también se pretende llevar a cabo la historia HU-1 que nos permitirá realizar el resto de tareas más fácilmente.

Todas las especificaciones que se pretenden cumplir en esta iteración son relativas al proveedor de almacenamiento en la nube Google Drive.

De esta manera, en primer lugar se implementará el estado de guardado de la aplicación (HU-8), procediendo después a abordar la historia HU-5, estableciendo

los permisos que los usuarios deben ceder a la aplicación para poder acceder a las cuentas de Google Drive y permitiendo la vinculación de más de una de estas cuentas de manera simultánea. Finalmente se desarrollará la funcionalidad descrita en la historia HU-3, permitiendo la obtención de todos los archivos y pudiendo realizar las acciones sobre estos.

3.2.3 Detalles de implementación

Antes de comenzar con las tareas que se desarrollarán dentro de esta iteración, cabe destacar la creación de la clase *Constantes*, de la que harán uso todas las clases tanto del paquete *Controladores* como del paquete *Modelos* y que contendrá todas las constantes necesarias para el desarrollo de la aplicación (por ejemplo, las rutas del archivo de credenciales de Google Drive y de las distintas vistas de la aplicación). Además, dentro de esta clase se incluye la clase de enumeración *Plataforma* que es utilizada fundamentalmente en el paquete *Modelos* por la clase *Cuenta* (Ilustración 15). Se ha decidido hacer así para unificar todas las constantes en una misma clase, facilitando la parametrización y evitando los posteriores errores que pudieran surgir.

En primer lugar, hay que definir los permisos que se requieren por parte del usuario para el correcto funcionamiento de la aplicación. Para ello, desde Google Cloud Platform → "APIS y servicios" → "Pantalla de consentimiento de OAuth" → "Editar App", editamos la aplicación que estamos desarrollando, modificando tanto el nombre de la aplicación (que en este caso se llamará MulticloudTFG) como los permisos. Para el tipo de aplicación que se está desarrollando, son necesarios permisos que nos permitan el acceso a información relativa a la cuenta del usuario, además de todos los detalles de los archivos que estas contengan, incluyendo la posibilidad de realizar modificaciones sobre estos.

A la hora de seleccionar los permisos, hay que tener en cuenta los tipos que existen y que se diferencian dentro de las APIs de Google[18], siendo necesario seguir un proceso de verificación por parte de Google, más o menos extenso, dependiendo de los que sean solicitados, en caso de ser alguno de tipo sensible o restringido. Para el funcionamiento deseado de la aplicación que estamos desarrollando se han

[18] Tipos de permisos Google API y verificación:
https://support.google.com/cloud/answer/9110914#sensitive-scopes

incluido los permisos incluidos en las ilustraciones 20 y 21 de entre todos los disponibles para las distintas APIs de Google[19] y en concreto para Google Drive[20].

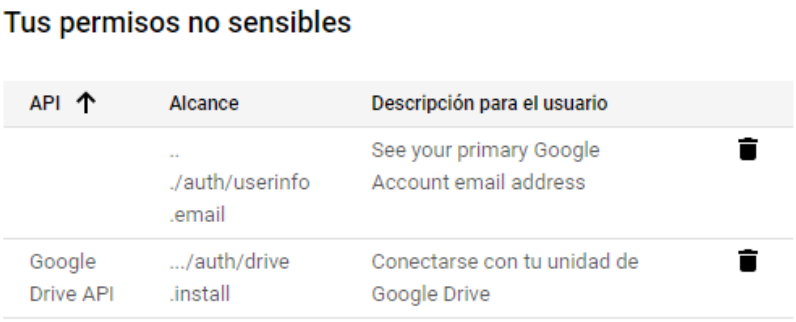

Ilustración 20: permisos no sensibles

Ilustración 21: permisos sensibles

Una vez establecidos todos los permisos necesarios dentro de Google Cloud, hay que definirlos en el código. Para ello se ha utilizado la clase DriveScopes[21], que contiene distintos permisos, por lo que se incluirán los necesarios al inicializar la

[19] Permisos Google APIs: https://developers.google.com/identity/protocols/oauth2/scopes
[20] Permisos Google Drive API: https://developers.google.com/identity/protocols/oauth2/scopes#drive
[21] https://developers.google.com/resources/api-libraries/documentation/drive/v2/java/latest/com/google/api/services/drive/DriveScopes.html

aplicación (DRIVE, DRIVE_READONLY y DRIVE_METADATA). Estos serán los que se solicitarán a cada uno de los usuarios que deseen vincular su cuenta de Google Drive a la aplicación.

Para permitir la conexión de más de una cuenta de cualquier proveedor de almacenamiento en la nube de manera simultánea es necesario en primer lugar almacenar los tokens que se irán generando, y que dan acceso a las cuentas sin verificarlas continuamente, para su posterior lectura, manteniendo así el estado de la aplicación. Por ello, se ha diseñado un sistema de carpetas que contendrá la aplicación (Ilustración 22) donde se guardará el estado de la misma.

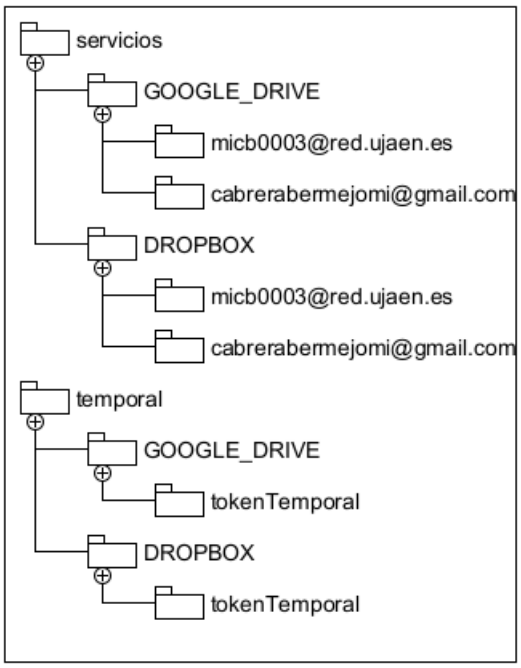

Ilustración 22: diagrama organización carpetas

De esta manera, mientras se realiza el proceso de vinculación de una cuenta, el archivo correspondiente a esta se encontrará en la carpeta *temporal* del servicio al que se esté conectando. Tras finalizar este proceso, se eliminará de esta ubicación para pasar a la carpeta respectiva dentro de *servicios*. Para cada cuenta se creará una carpeta con su correo y que contendrá el token que permite establecer la conexión.

Se ha diseñado de esta manera porque, dado que el proceso de vinculación se realiza a través del navegador, es posible que se produzca cualquier fallo durante el mismo (derivado de una falta de conexión, de un fallo del proveedor o de un error

humano por parte del usuario). Con este sistema de carpetas se garantiza que en caso de ocurrir, al no estar finalizada la vinculación, no habrá almacenados como datos de cuentas vinculadas aquellas que realmente no lo estén; y solo se almacenará el archivo temporal que, en caso de fallo será eliminado para que, posteriormente, al intentar realizar una nueva conexión, se cree de nuevo, evitando así fallos en el sistema.

Para la conexión de nuevas cuentas (sin límite en cuanto a número) con el proveedor de almacenamiento en la nube Google Drive, se implementa el método correspondiente (*nuevaCuenta*) dentro de la clase encargada de la gestión de este proveedor (*AlmacenamientoGoogleDrive*). Cada vez que se intente vincular una cuenta se abrirá el navegador para establecer la conexión y una vez que esta finalice, se realiza el proceso de guardado del archivo que contiene el token obtenido en la carpeta correspondiente tal y como se ha comentado antes (comprobando que no se está vinculando una cuenta que ya está enlazada, y lanzando un mensaje de error en caso de que esto ocurra).

Google Drive utiliza el protocolo OAuth2.0[22] para la autentificación y autorización, generando un flujo seguro para establecer las conexiones con las cuentas que se deseen enlazar a la aplicación. Para acceder a una cuenta que ya está vinculada, evitando el proceso que se lleva acabo al introducir una nueva, el flujo utilizado es ligeramente distinto, dado que en lugar de crear un fichero temporal para almacenar los nuevos datos, se leen los ya existentes en la carpeta correspondiente a la cuenta a la que se desea acceder. Para la generación de este flujo se ha añadido un nuevo método llamado *getCredentials* en la clase *AlmacenamientoGoogleDrive*, modificando ligeramente el diagrama de clases del paquete *Modelos* (Ilustración 23).

[22] Protocolo OAuth 2.0: https://oauth.net/2/

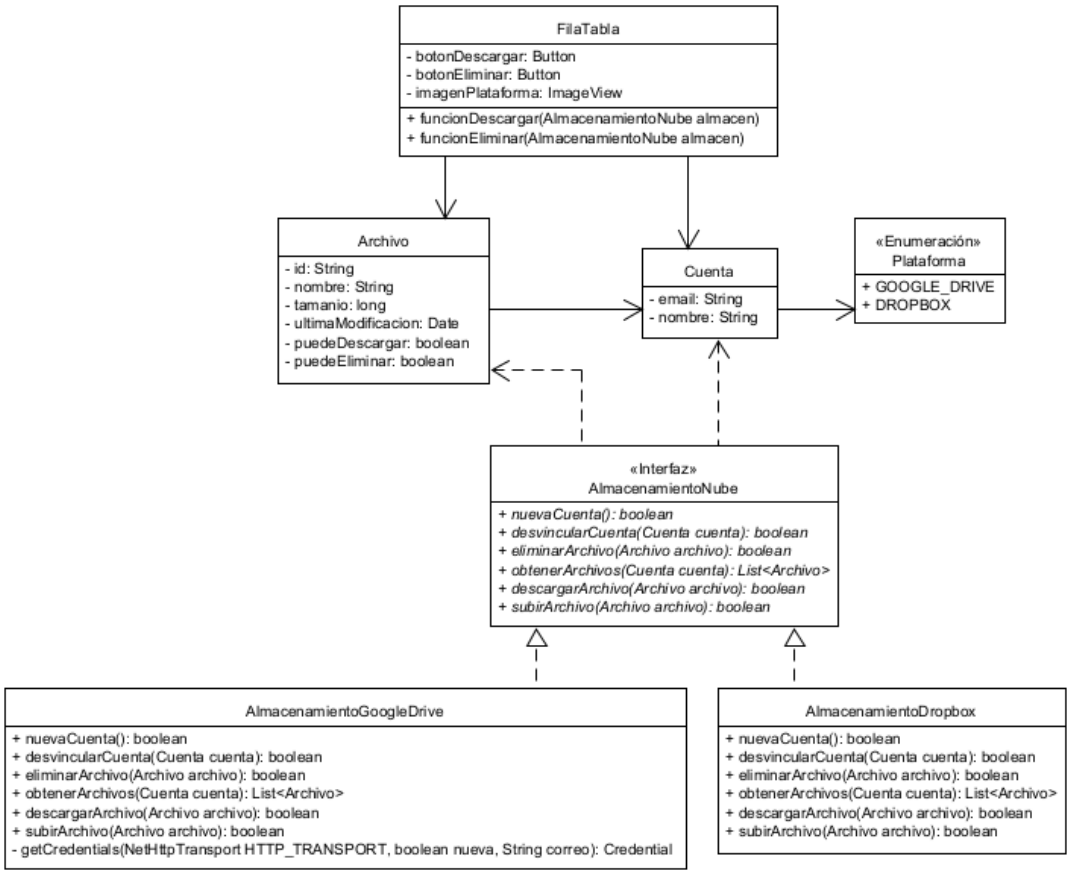

Ilustración 23: diagrama de clases del paquete Modelos modificado (getCredentials)

Una vez hecho esto, se modifica el código utilizado en los trabajos preliminares, cuando se estableció una primera toma de contacto con la API de Google Drive, para la obtención de los archivos. Se obtienen todos los ficheros almacenados en Drive (eliminando la limitación del número máximo) siendo necesario recorrer, a través de tokens, todas las páginas de archivos obtenidas con las consultas realizadas al proveedor (dado que solo permite un total de 100 archivos por consulta). Esto, que puede llegar a demorar algo de tiempo dependiendo del número de archivos que tengamos almacenados, nos permitirá obtener todos los ficheros reunidos en la cuenta con la que deseemos conectar. Además, se obtienen todos los campos de los ficheros (no solo id y nombre como estaba establecido en un primer momento) ya que posteriormente serán necesarios para definir la funcionalidad que se puede llevar a cabo con cada uno de los archivos y para mostrar la información más relevante a los usuarios. Esta funcionalidad para

obtener todos los ficheros de una cuenta está incluida dentro del método *obtenerArchivos* de la clase *AlmacenamientoGoogleDrive*.

Gracias a esto tenemos toda la información de todos los archivos de Google Drive de la cuenta con la que estemos vinculando la aplicación. Para mostrar esta información al usuario se comenzará creando una tabla similar a la diseñada en el boceto inicial (Ilustración 8) con los campos considerados más importantes relativos a los archivos. Estos son: el nombre, el tipo de archivo (si se trata de un fichero o una carpeta, según el tamaño en bytes, siendo una carpeta siempre que este tenga un valor 0), el tamaño en su unidad correspondiente (realizando la conversión necesaria), la fecha de última modificación, la nube a la que pertenece (con un icono de la misma para una visualización más clara) y la cuenta; finalmente, se incluyen también los dos botones para descargar y eliminar el archivo en caso de tener sobre este los permisos necesarios. Las columnas se establecen en este mismo orden para tener en primer lugar los datos relativos al fichero, en segundo lugar los correspondientes a la nube y cuenta a la que pertenece y finalmente las acciones que se pueden llevar a cabo.

Cada vez que se inicie la aplicación, el proceso que se llevará a cabo será el mismo: abrir la ventana inicial y leer las carpetas incluidas dentro de */servicios*, obteniendo cada una de las cuentas de todos los proveedores que estén guardados dentro de esta carpeta (restaurando el estado anterior de la aplicación en todo momento). Después se obtienen los archivos de cada una de las cuentas utilizando el proveedor de almacenamiento al que correspondan (ahora mismo solo se está utilizando Google Drive pero se establece lo necesario para utilizar distintos proveedores, de manera que su posterior inclusión sea más sencilla) y se rellena la tabla con tantas filas como archivos haya.

Por cada uno de los archivos obtenidos de todas las cuentas vinculadas a la aplicación, se crea un objeto de la clase *FilaTabla* que es el que se incluye en la tabla (que es una TableView<FilaTabla>) y se comprueba si es posible su descarga o borrado de la nube para crear los botones correspondientes e implementar su funcionalidad.

En caso afirmativo, para la descarga de archivos, se establece que al pulsar el botón se abra una ventana de diálogo con los parámetros necesarios y que

permite guardar el archivo en el dispositivo. Este será el utilizado para copiar en él los datos del fichero de la nube que se descarga mediante el método *descargarArchivo* del proveedor de almacenamiento en la nube correspondiente (en este caso Google Drive que es con el que estamos trabajando). Para la descarga de archivos, se ha implementado un método privado llamado *descargaRecursiva* (Ilustración 24) que comprobando si se trata de una carpeta o no, realiza iteraciones recursivas para obtener todos los archivos que esta contenga o descargar el fichero correspondiente. Además, también se comprueba si el archivo que se pretende descargar pertenece a una de las aplicaciones online de Google (documentos, hojas de cálculo, presentaciones etc.), convirtiéndolo a su equivalente para proceder a la descarga. Se comprueban y se convierten de Google Docs a Microsoft Word, de Google Sheets a Microsoft Excel, de Google Drawing a imagen en PNG, de Google Jamboard a PDF y de Google Slides a Microsoft PowerPoint. Pero en estos casos, la API de Google Drive solo nos permite exportarlos al formato que deseemos si no superan los 10MB[23], en caso contrario, no se pueden descargar. La aplicación ha sido adaptada a ello para que cuando esto ocurra se informe al usuario del error mediante el mensaje conveniente.

[23] https://developers.google.com/drive/api/v3/reference/files/export

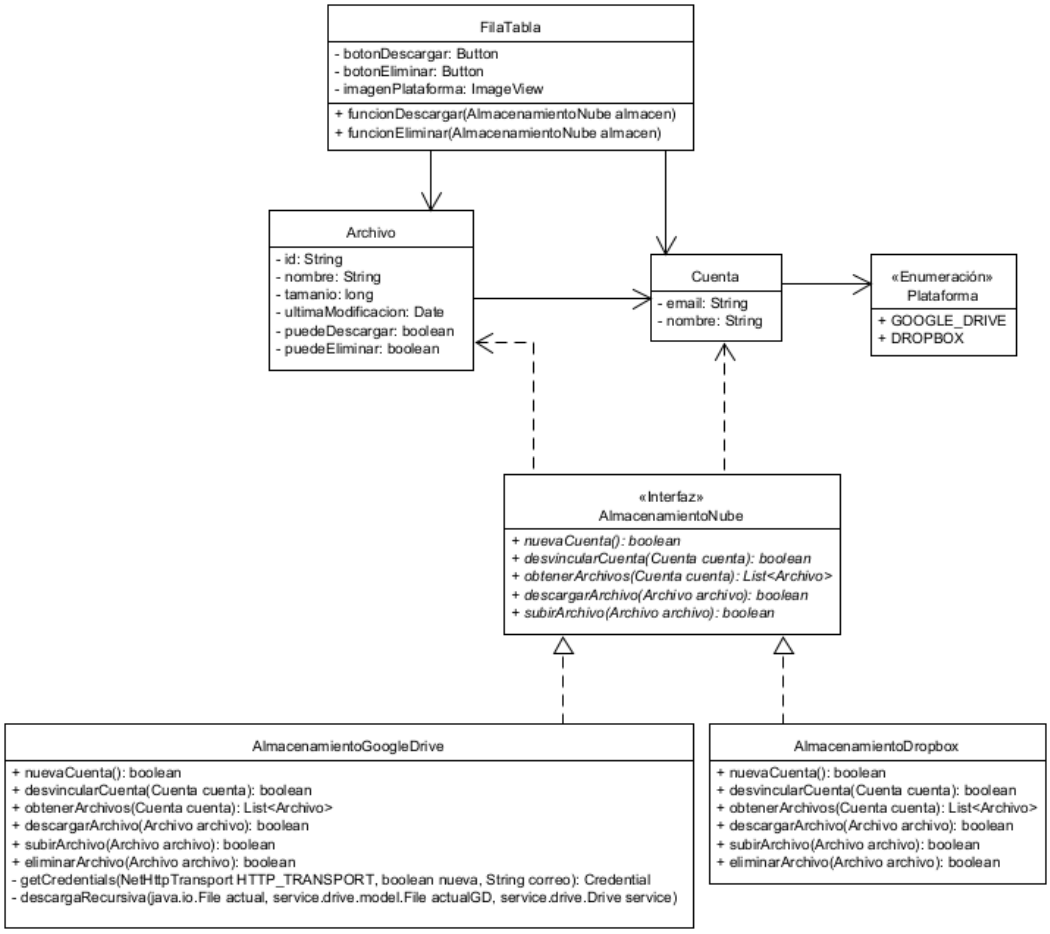

Ilustración 24: diagrama de clases del paquete Modelos modificado (descargaRecursiva)

Por otro lado, el borrado de archivos no se lleva a cabo de manera permanente, sino que el archivo o carpeta se envía a la papelera de Google Drive. Para ello se modifica el valor del atributo (*trashed*) que indica si se encuentra en la papelera y se actualiza su valor. Una vez hecho esto, se actualiza la tabla que se muestra al usuario (donde ya no aparecerá el archivo eliminado) y se muestra un diálogo de alerta que informa al usuario de lo que acaba de ocurrir.

Con las funcionalidades de eliminar y descargar archivos totalmente completadas, es la de subir archivos la que da por finalizada esta iteración. Esta es ligeramente diferente a las anteriores, ya que en ella interviene una nueva ventana para la selección de archivos y la cuenta a la que se desean subir. Durante la creación de esta, ha sido ligeramente modificada respecto a su diseño inicial (Ilustración 10) cambiando el orden de la selección, dejando en primer lugar la

selección de archivos (a través de un control que nos permite explorar en el equipo) y seguidamente la selección de la cuenta a la que se quieren subir los archivos mediante un ChoiceBox de JavaFX. Estos han sido cambiados de sitio respecto al diseño inicial para evitar que al seleccionar una de las cuentas disponibles se cubra excesivamente la selección de archivos. Además de este cambio, también se ha eliminado el botón en forma de flecha que permitía volver atrás y se ha incorporado un botón para cancelar la operación junto al que finaliza la subida de archivos. Además, al abrirse la ventana con la modalidad APPLICATION_MODAL[24], se bloquea la ventana principal para que no se pueda utilizar hasta que no se cierre la ventana que se ha abierto para subir archivos, por lo que cobra más sentido un botón de cancelación de operación que uno para volver a la ventana anterior (ya que solo se puede volver a esta cuando se cierre la que se acaba de abrir). Todos estos cambios realizados se pueden apreciar en la Ilustración 25, imagen obtenida de la ventana creada con Scene Builder.

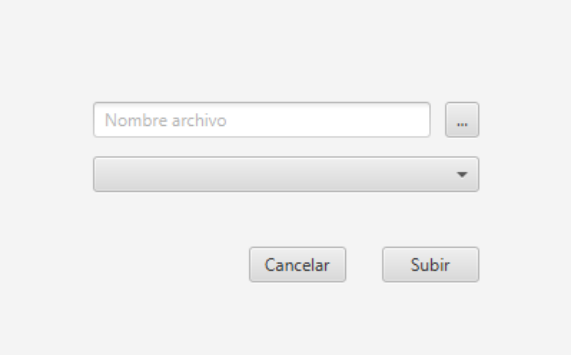

Ilustración 25: modificaciones en el cuadro de diálogo para subir archivos

Tras abrir la ventana para subir archivos a través del botón correspondiente en la ventana principal, se rellena el selector (ChoiceBox) donde seleccionaremos la cuenta a la que queremos subir los archivos de entre todas las cuentas que se encuentren actualmente vinculadas a la aplicación. Estas aparecerán acompañadas de la plataforma a la que pertenecen, apareciendo seleccionada por defecto la primera de ellas.

En el controlador encargado de la funcionalidad de esta ventana (*ControladorSubirArchivo*) se implementan los métodos correspondientes a los botones presentes en ella: el que nos abre la ventana de diálogo donde se

[24] Tipos de modalidades de una ventana JavaFX: https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html

seleccionan los archivos del dispositivo que se desean subir a la nube, estableciendo los parámetros necesarios y obteniendo los elegidos; el botón para cancelar la operación, que simplemente se limita a cerrar la ventana; y el botón que será pulsado para finalmente subir los archivos, en cuyo caso se comprueba que haya una cuenta seleccionada así como archivos para mostrar un mensaje de error si fuese necesario y de nuevo, proceder a su cierre.

Cuando se pulse el botón de *Subir*, se comprueba si algún archivo ha sido seleccionado así como la cuenta a la que se desea subir. En caso de existir, se procede a la subida de los mismos por parte del almacén correspondiente (en este caso Google Drive) a través de la clase encargada de su gestión. Dentro del método *subirArchivo* de la clase *AlmacenamientoGoogleDrive* se crea un fichero de Google Drive con el contenido del archivo que se desea subir y completando sus metadatos únicamente con el nombre. Este proceso se repite para tantos archivos como hayan sido seleccionados por el usuario en la ventana de diálogo. Dentro de los tres tipos de subida que permite la API de Google Drive[25], se ha utilizado la subida multiparte que permite subir archivos de 5MB o menos (aunque en la práctica, tras su implementación, se han llegado a subir archivos de hasta 25MB) con los metadatos asociada correspondiente ya que la subida reanudable que permite subir archivos de más de 5MB utiliza peticiones HTTP y en el desarrollo de la aplicación se está utilizando Java y esto complicaría y alargaría más el proceso.

Finalmente, se actualiza la tabla que se muestra al usuario para que este perciba los cambios ocurridos y cómo los archivos se han subido correctamente a la cuenta de Google Drive.

3.2.4 Pruebas de verificación y validación

Antes de dar por concluida esta iteración, se comprueba que la aplicación cumple con lo indicado en las historias de usuario seleccionadas al comienzo de la iteración. Para esto se comprueba que se cumplen los criterios de aceptación definidos en el apartado 2.1. Comenzamos con los de la historia HU-8, que ha sido la primera que se ha implementado.

El primero de ellos, el CA-8.1, no puede darse por cumplido ya que no se ha desarrollado todavía el apartado que muestra al usuario el listado de todas las

[25] Subida de archivos con Google Drive API: https://developers.google.com/drive/api/v3/manage-uploads

cuentas enlazadas, por lo que queda pendiente para una posterior revisión en la iteración en la que se lleve a cabo esta tarea. De igual manera, el CA-8.2 no puede verificarse dado que el apartado del menú de la aplicación no ha sido desarrollado todavía, y de igual manera se comprobará en la iteración correspondiente al desarrollo del mismo. Por otro lado, el CA-8.3 sí puede darse por verificado ya que efectivamente, una vez vinculada una cuenta a la aplicación, aunque esta se cierre, al volver a iniciar el sistema, reaparece tal y como se dejó la última vez que fue cerrada, conservando en todo momento el estado de la misma y mostrando todos los archivos, en la tabla de la pantalla inicial, correspondientes a todas las cuentas vinculadas a la aplicación. Por tanto, se da por verificado únicamente el CA-8.3 dentro de la historia HU-8 y el resto quedan pendientes para las iteraciones correspondientes. De esta manera, no se contabilizarán los puntos de historia correspondientes a la historia HU-8 dado que no se verifican todos sus criterios de aceptación.

Siguiendo el orden de desarrollo de las distintas historias de usuario, se pasa a verificar los criterios de aceptación de la historia HU-5 con respecto al proveedor de almacenamiento Google Drive. En este caso, el CA-5.1 se cumple, dado que existe un botón que permite vincular tantas cuentas de Google Drive como deseemos a la aplicación. Al igual que el último criterio, el CA-5.2, ya que la lista de archivos es actualizada incluyendo los archivos de la nueva cuenta vinculada a la aplicación. Aunque estéticamente el botón que desarrolla esta funcionalidad esté colocado de manera provisional en la pantalla inicial y no en su lugar correspondiente, podemos dar por verificada esta historia dado que la funcionalidad la cumple a la perfección, siendo únicamente necesario colocar el botón en la ventana correspondiente cuando se implemente en posteriores iteraciones.

Finalmente, se comprueban que se verifiquen los criterios de aceptación de la historia HU-3 respecto al proveedor Google Drive. En este caso, se cumplen todos los criterios de aceptación descritos sobre esta historia. En la ventana de archivos o ventana principal donde se muestran todos los archivos pertenecientes a las cuentas vinculadas a la aplicación, junto a cada archivo aparece un par de botones para su eliminación o descarga (siempre que sea posible) que cumplen con esta funcionalidad a la perfección, mandando el fichero a la papelera dentro de nuestra nube de Google Drive u obteniendo el archivo y creándolo en nuestro dispositivo al descargarlo. Además, existe un botón que abre una ventana modal que permite

seleccionar un archivo en nuestro directorio local y subirlo a la nube que deseemos (seleccionándola previamente), controlando además todos los errores que puedan ocurrir en cualquiera de estos procesos y advirtiendo al usuario en todo momento. Finalmente, cada vez que se sube o elimina un archivo de entre los listados, la tabla de actualiza acorde a la operación realizada.

También es necesario comprobar y verificar la historia HU-1 dado que se ha creado la tabla donde se listan los archivos de todas las cuentas vinculadas a la aplicación que nos permite visualizarlos de manera unificada. De nuevo, se cumplen los dos criterios de aceptación que componen esta historia de usuario, dado que la tabla que unifica y muestra todos los archivos ya ha sido generada y creada en la ventana principal, que es la que aparece al abrir el sistema y en la que se cargan los datos de la aplicación para restaurar el estado de la misma.

Una vez realizadas las pruebas de verificación siguiendo los criterios de aceptación de las historias de usuario tratadas en esta iteración, se crea una tabla resumen (Tabla 16) de lo que se va cumpliendo y que se irá rellenando en posteriores iteraciones hasta que se cumplan todos los criterios de aceptación y se puedan dar por finalizadas las historias de usuario. Tanto la historia HU-3 como la historia HU-5 detallan requisitos que dependen del proveedor de almacenamiento por lo que se incluirán tantas como proveedores se incorporen a la aplicación. En un primer momento se comenzará con Google Drive y Dropbox y después se pasará al resto de proveedores en caso de dar tiempo, por lo que actualmente se incluyen dos variantes de las mismas, HU-3.1, HU-3.2, HU-5.1 e HU-5.2 donde el .1 indica que es referido a Google Drive y el .2 hace referencia a Dropbox.

		Iteración 1
HU-1	CA-1.1	X
	CA-1.2	X
HU-2	CA-2.1	
HU-3.1	CA-3.1.1	X
	CA-3.1.2	X
	CA-3.1.3	X

HU-3.2	CA-3.2.1	
	CA-3.2.2	
	CA-3.2.3	
HU-4	CA-4.1	
HU-5.1	CA-5.1.1	X
HU-5.2	CA-5.2.1	
HU-6	CA-6.1	
	CA-6.2	
HU-7	CA-7.1	
HU-8	CA-8.1	
	CA-8.2	
	CA-8.3	X

Tabla 16: tabla resumen iteración 1

Así, y habiendo completado las historias HU-1 (3 puntos), HU-3.1 (8 puntos) e HU-5.1 (3 puntos), y teniendo en cuenta que la aplicación se desarrollará con la planificación temporal indicada en el apartado 1.12, se crea el diagrama Burndown correspondiente habiendo completado un total de 13 puntos en esta iteración (y en total también al tratarse de la primera) de los 43 que consta el conjunto de historias de usuario teniendo en cuenta las repetidas.

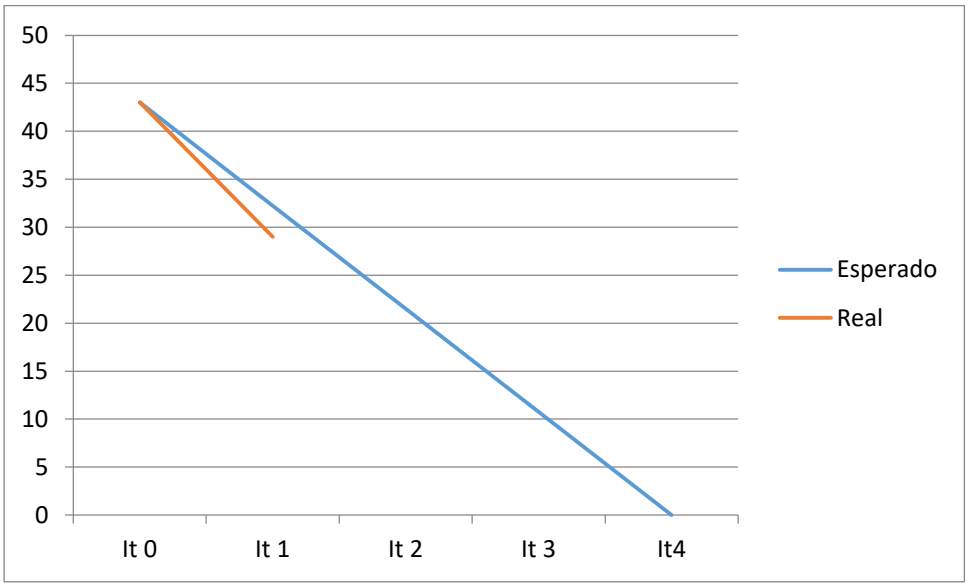

Ilustración 26: diagrama Burndown iteración 1

En la Ilustración 26 se puede observar como, durante el desarrollo de los trabajos preliminares que se llevaron a cabo previos a la primera iteración, se definieron las historias de usuario así como la puntuación de estas, siendo este momento el marcado en la iteración cero en el diagrama. Además de verse como posteriormente, durante esta primera fase de implementación de la aplicación, se ha avanzado respecto a lo esperado en el proyecto. Este seguimiento se hará cada vez que se finalicen las sucesivas iteraciones hasta dar por concluido el desarrollo de este software.

3.3 Segunda iteración

Tras concluir con el proveedor de almacenamiento en la nube Google Drive y disponiendo de todas las funciones esperadas de este, se procede en esta iteración a llevar a cabo estas mismas tareas con otro proveedor distinto, en este caso Dropbox. De esta manera, tras finalizar esta iteración debe poderse establecer conexión con más de una cuenta y obtener todos los archivos de estas, además de poder realizar todas las acciones necesarias sobre estos, es decir, descargar y eliminar cuando sea posible y subir archivos a la nube. Esta iteración tiene una duración estimada de un mes.

3.3.1 Historias de usuario

Los objetivos a cumplir en esta iteración son los reflejados en las historias de usuario HU-3 y HU-5 descritas detalladamente en el apartado 2.1. Haciendo referencia ambas historias, en todo momento, al proveedor de almacenamiento en la nube Dropbox, que es el siguiente que se incluirá en la aplicación tras finalizar con Google Drive en la iteración anterior.

De manera similar a como se procedió anteriormente, se comenzará desarrollando la historia HU-5, definiendo los permisos que requiere la aplicación para la conexión con el proveedor y permitiendo la vinculación de múltiples cuentas. Una vez hecho esto, se desarrollan las funcionalidades comprendidas en la historia HU-3, pudiendo obtener y visualizar todos los archivos de las cuentas enlazadas a la aplicación y existiendo la posibilidad de descargar y eliminar cualquiera de estos

siempre que sea posible, así como subir múltiples archivos desde nuestro dispositivo a la nube.

3.3.2 Detalles de implementación

Al igual que se hizo con Google Drive, para conectar Dropbox como proveedor de almacenamiento en la nube a nuestra aplicación, el primer paso es registrar la aplicación dentro de la consola de Dropbox, obtener las credenciales correspondientes y definir los permisos que se requerirán por parte del usuario.

Para ello, entramos en Dropbox Console[26] y creamos una nueva aplicación seleccionando en primer lugar *Scoped access* y eligiendo el tipo de acceso que necesitamos. Se nos presentan dos posibilidades, *App folder* o *Full Dropbox*, siendo la primera de ellas acceso único a una carpeta concreta, por lo que seleccionamos la segunda de estas opciones que nos permite acceder de manera completa a cuentas en Dropbox. Finalmente asignamos el nombre a nuestra aplicación y entramos en ella, apreciando distintas subsecciones dentro.

En *Settings*, comprobamos que el software se encuentra en estado de desarrollo y obtenemos las credenciales de la aplicación, que constan de dos códigos *App key* y *App secret*. Estos códigos debemos copiarlos como un fichero .json de manera análoga a como se hizo con las credenciales de Google Drive (en la misma carpeta) con el formato correspondiente, incluyendo dos parámetros o valores: *key* y *secret*. Además se incluye la ruta en la clase *Constantes* de manera análoga a lo que se hizo anteriormente con Google Drive. De esta manera tendremos almacenadas las credenciales de Dropbox como *credencialesDropbox.json* y las de Google Drive como *credencialesGD.json* en la misma carpeta en la que se incorporarán y se leerán tantas credenciales como proveedores de almacenamiento en la nube contenga la aplicación.

Una vez hecho esto, se definen los permisos necesarios para acceder a las cuentas que el usuario desee vincular. Esto se hace dentro de la subsección *Permissions*, añadiendo a los que ya vienen marcados, los permisos de *files.medatada.write* para modificar la información de todo tipo de ficheros, *files.content.write* y *files.content.read* para tanto editar como ver el contenido de los archivos almacenados en Dropbox, y finalmente *sharing.write* que nos permite

[26] Dropbox Console: https://www.dropbox.com/developers/

controlar los permisos y ajustes de todos los archivos. De esta manera, quedan marcados como permisos de la aplicación los que se muestran en la Ilustración 27. Además, es importante destacar que el resto de permisos de los apartados contenidos en *Team Scopes* son aquellos incluidos dentro de la versión Business de Dropbox y por tanto utilizan la versión Business de la API.

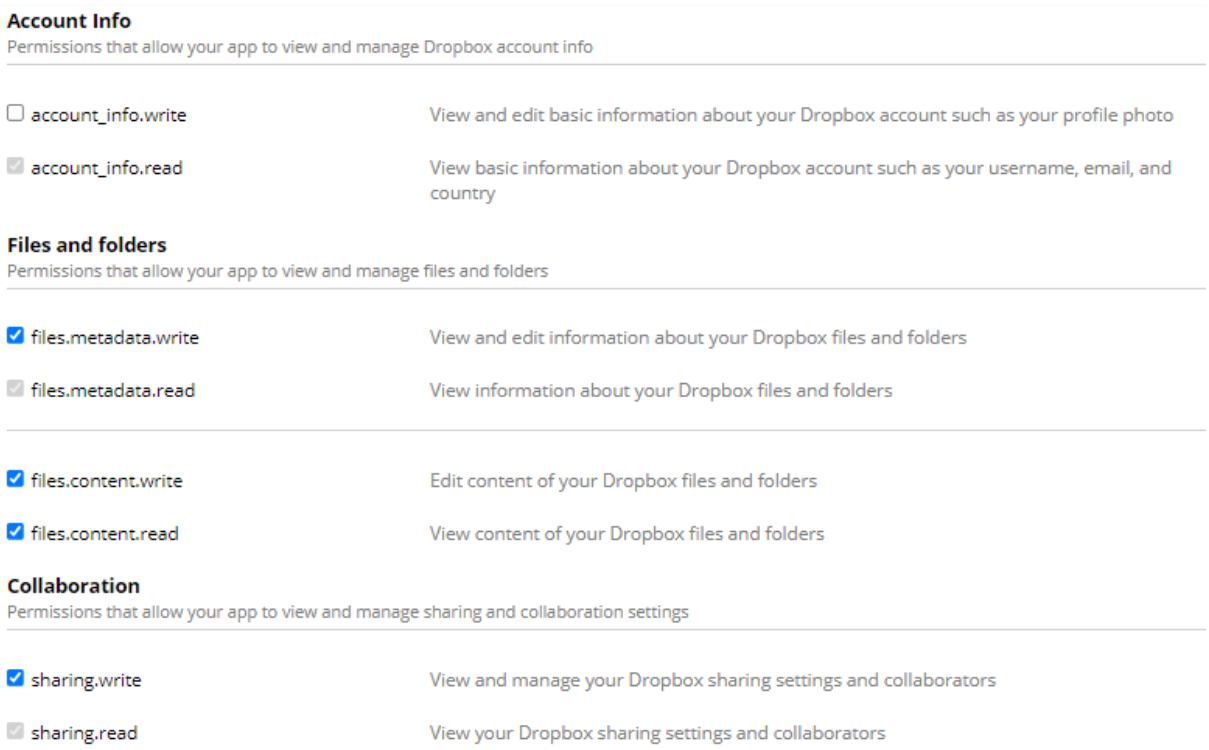

Ilustración 27: permisos Dropbox

Dado que Dropbox también utiliza OAuth 2.0 como protocolo de autorización para permitir el acceso por parte de la aplicación a la cuenta en Dropbox del usuario sin conocer las credenciales del mismo, se utilizan de nuevo tokens de acceso que serán almacenados y tratados de la misma manera que los de Google Drive para guardar el estado de la aplicación. Además, se puede establecer el periodo de caducidad de estos tokens dentro de la consola de Dropbox, existiendo dos posibilidades, la no caducidad de los mismos o una expiración temprana tras el trascurso de cuatro horas. Se ha decidido establecer una no caducidad para los tokens, ya que el periodo de cuatro horas puede resultar excesivamente breve y en caso de que el usuario no acceda a la aplicación durante días esto puede resultar problemático.

El primer paso para conectar con Dropbox nuestra aplicación es incluir las dependencias correspondientes a la API en el fichero *build.gradle* tal y como se indica en el tutorial[27] proporcionado por el propio proveedor. Para vincular cuentas y obtener el token, se crea un botón (como se procedió para la conexión de cuentas con Google Drive) en la ventana principal de la interfaz que, al pulsarlo, nos permita conectar cuentas de Dropbox con la aplicación. Esto posteriormente será modificado e incluido en la ventana de ajustes tal y como se muestra en el diseño inicial, pero para comprobar su correcto funcionamiento se hace de esta manera en un primer momento.

Sin embargo, la conexión con Dropbox para la obtención de un token es ligeramente distinta a la de Google Drive. Esto se debe a que para obtener el token que nos permita acceder a la cuenta del usuario, este es dirigido al navegador donde debe introducir las credenciales de la cuenta que desea vincular a la aplicación. Una vez hecho esto, le aparecerá en la misma pantalla del navegador un código que debe introducir en la aplicación para obtener el token a partir de este. Esto es así puesto que esta manera de obtención de tokens está pensada para aplicaciones web en las que sí sería posible pasar este código de manera automática sin necesidad de acciones por parte del usuario. Debido a esta característica especial, se ha diseñado una ventana informativa (Ilustración 28) que se abrirá antes de conectar con Dropbox para informar al usuario de los pasos que debe seguir. De igual manera, se comprueba siempre, antes de finalizar la conexión y dar por vinculada la cuenta a la aplicación, que se ha introducido un código y que la cuenta que se pretende enlazar no lo está ya (gracias a la información interna que contiene dicho código).

[27] Tutorial Dropbox API: https://github.com/dropbox/dropbox-sdk-java#setup

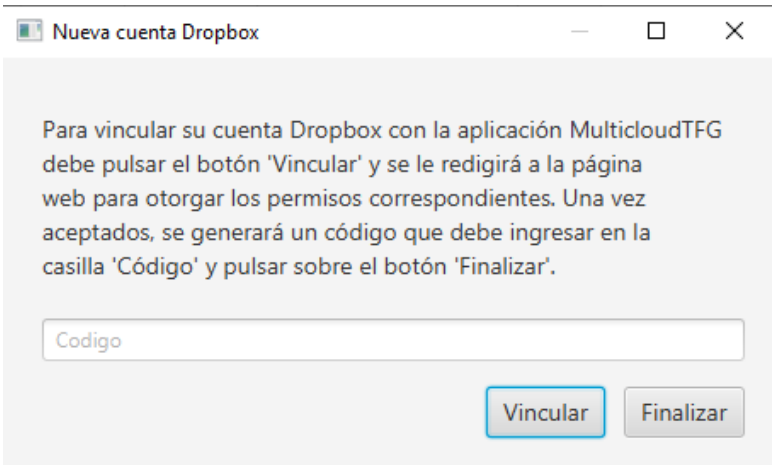

Ilustración 28: ventana informativa al conectar con Dropbox

Debido a esto, ha sido necesario modificar la clase *AlmacenamientoDropbox* (Ilustración 29) encargada de la gestión de todo lo referente al nuevo proveedor. Se ha incluido un nuevo método, *obtenerURLToken* que, con las credenciales de la aplicación obtiene y devuelve la URL a la que se redirige al usuario para la obtención del código que posteriormente proporcionará el token. Además, también se ha incluido un método *guardarNuevaCuenta* que con el código introducido por el usuario accede a Dropbox para obtener el token correspondiente, almacenándolo en la carpeta necesaria para mantener el estado de la aplicación con la nueva cuenta que acaba de ser vinculada o lanzando un mensaje de error en caso de que esta ya exista. Por ello, el método abstracto *nuevaCuenta* se ha dejado sin implementar, dado que se utilizan los dos comentados anteriormente por necesitar devolver u obtener unos parámetros para establecer la conexión. Este proceso puede repetirse tantas veces como se desee, permitiendo la conexión de más de una cuenta de Dropbox a la aplicación.

La obtención de archivos de Dropbox mediante la API genera listados de archivos por carpetas. Debido a esto, ha sido necesario ir recorriendo recursivamente todas las carpetas contenidas en la nube de este proveedor comenzando por la carpeta raíz para obtener así todos los archivos. Para ello, se comprueba en todo momento si se trata de una carpeta o de un fichero para continuar con la iteración recursiva o no. Además, también se comprueba en todo momento si se pueden descargar y eliminar todos los archivos y carpetas, para asignar los valores necesarios a los atributos correspondientes y así crear posteriormente los botones para realizar estas acciones en los casos que

correspondan. Esta obtención de archivos recursiva se realiza mediante un nuevo método privado *obtenerArchivosRecursivamente* que ha sido añadido a la clase *AlmacenamientoDropbox* (Ilustración 29).

Tras esto y con la base generada en la primera iteración para el tratamiento de los distintos archivos ya sean de un proveedor u otro para manejarlos a todos de igual manera mediante la interfaz *AlmacenamientoNube*, solo es necesario implementar los métodos que permiten descargar, eliminar y subir archivos a Dropbox dentro de la clase *AlmacenamientoDropbox*.

Comenzando con la eliminación de archivos, se desarrolla el método, heredado de la interfaz, *eliminarArchivo* en la clase *AlmacenamientoDropbox* utilizando la API de Dropbox para mandar el archivo que haya sido seleccionado a la papelera para que la eliminación sea de manera homogénea sea el proveedor que sea.

Para la descarga de archivos, se ha implementado un nuevo método llamado *descargaRecursiva* (Ilustración 29) que permite la descarga recursiva de ficheros pudiendo bajar carpetas con todo su contenido. Para la iteración recursiva se ha realizado de manera similar a como se ha hecho en la obtención de archivos.

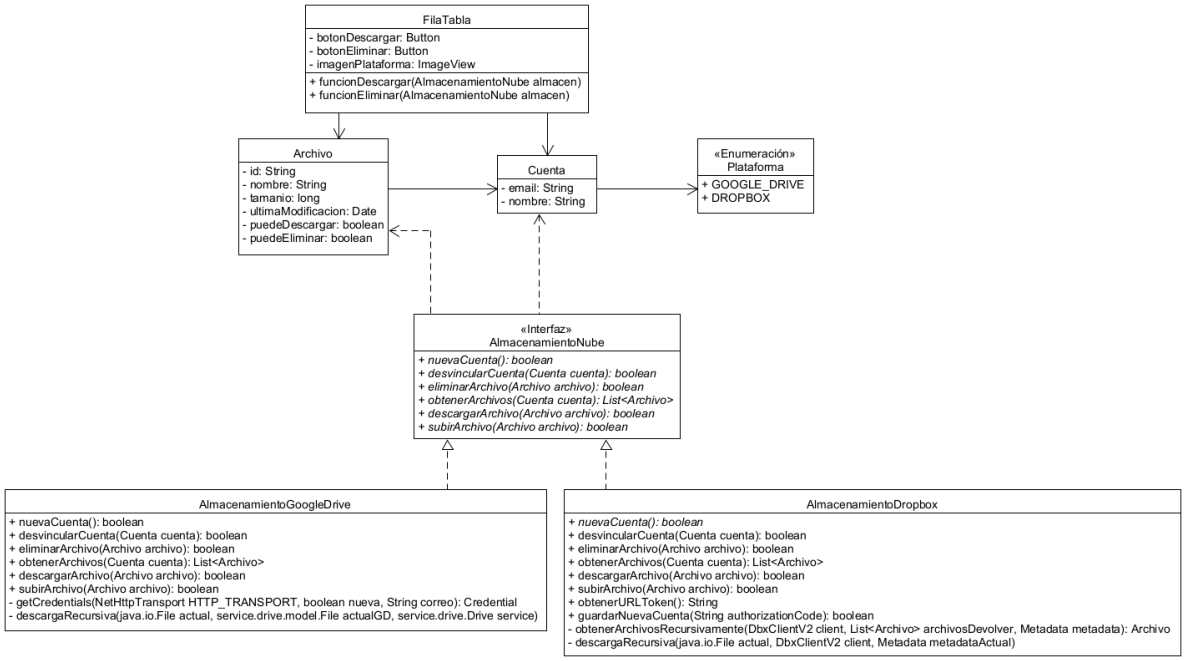

Ilustración 29: diagrama de clases del paquete Modelos modificado (*obtenerURLToken + guardarNuevaCuenta *+ descargaRecursiva + obtenerArchivosRecursivamente)

Finalmente, para la subida de archivos, se utiliza la ventana que permite llevar a cabo esta operación, tal y como se hizo antes con la subida de archivos con Google Drive. Para repetir el proceso con Dropbox solo es necesario implementar el método *subirArchivo* de la clase *AlmacenamientoDropbox*, donde se establece en primer lugar la conexión con el proveedor desde la cuenta seleccionada en la ventana mencionada anteriormente. Una vez hecho esto, se crea el archivo con los metadatos y se sube a la nube. Con este método se pueden subir múltiples archivos en una única selección por parte del usuario, al igual que ocurre con Google Drive.

Como se puede observar, con la parte general construida en la iteración anterior, en esta solo es necesario implementar los métodos correspondientes a la interfaz y realizar los ajustes necesarios para incorporar las características de esta nueva API. Con todas las funcionalidades descritas en las historias HU-3 y HU-5 finalizadas, se puede dar por concluida esta iteración.

3.3.3 Pruebas de verificación y validación

Una vez finalizadas las tareas de implementación de las funcionalidades descritas en las historias de usuario objetivo de esta iteración, se procede a verificar los criterios de aceptación de las mismas, comprobando su correcto funcionamiento. Siendo las historias HU-5 y HU-3 las que deben haberse completado durante el periodo que esta iteración ha durado, y haciendo ambas referencia al proveedor de almacenamiento en la nube Dropbox (anteriormente se llevaron a cabo estas historias pero referidas a Google Drive) se comenzará por la primera de estas, la historia HU-5, al ser la primera que se desarrolló y se llevó a cabo.

El primer y único criterio de aceptación de esta historia de usuario, el CA-5.1 se cumple correctamente, con un botón generado en la ventana principal de la aplicación que permite conectar tantas cuentas de Dropbox como deseemos a la aplicación, actualizando en todo momento el estado del sistema y facilitando su guardado y mantenimiento. Aunque no será esta la forma final que dará acceso a la vinculación de cuentas, sí queda de esta manera concluido lo referente a esta funcionalidad. En iteraciones posteriores, se incluirá esta función, que ya ha sido desarrollada e implementada, en la ventana correspondiente tal y como se indica en el diseño inicial de la interfaz de usuario.

Finalmente, se comprueba la otra historia de usuario, la historia HU-3 en relación a Dropbox, contenida en esta iteración. De nuevo, se cumplen todos los criterios de aceptación que verifican la finalización de esta historia. En primer lugar, y haciendo uso de la ventana ya implementada en la iteración anterior, se permite la subida de archivos a cualquier cuenta de Dropbox, cumpliendo el CA-3.1. Además, se observan en la tabla de la ventana inicial de la aplicación todos los archivos pertenecientes a las cuentas de Dropbox que se encuentren vinculadas; junto a estos, existen botones que permiten tanto la descarga (de archivos individuales o de carpetas enteras) como la eliminación, cumpliendo y quedando verificados también los CA-3.2 y CA-3.3.

Después de examinar y validar los criterios de aceptación que correspondían, se actualiza la tabla resumen (Tabla 17), incluyendo una nueva columna referente a la iteración actual. En este caso, y dado que se han completado las historias de usuario HU-3 y HU-5 con respecto al proveedor Dropbox, aparecen marcadas como completas las historias HU-3.2 y HU-5.2. Para las posteriores incorporaciones de otros proveedores de almacenamiento en la nube, se incluirán en esta tabla siguiendo la regla utilizada para Google Drive (.1) y Drobpox (.2).

		Iteración 1	Iteración 2
HU-1	CA-1.1	X	
	CA-1.2	X	
HU-2	CA-2.1		
HU-3.1	CA-3.1.1	X	
	CA-3.1.2	X	
	CA-3.1.3	X	
HU-3.2	CA-3.2.1		X
	CA-3.2.2		X
	CA-3.2.3		X
HU-4	CA-4.1		
HU-5.1	CA-5.1.1	X	
HU-5.2	CA-5.2.1		X
HU-6	CA-6.1		

	CA-6.2		
HU-7	CA-7.1		
HU-8	CA-8.1		
	CA-8.2		
	CA-8.3	X	

Tabla 17: tabla resumen iteración 2

Finalmente, en la Ilustración 30 podemos ver el diagrama Burndown para comprobar el avance seguido respecto a lo esperado, habiendo completado las historias HU-3.2 (8 puntos) y HU-5.2 (3 puntos), con un total de 11 puntos en esta iteración.

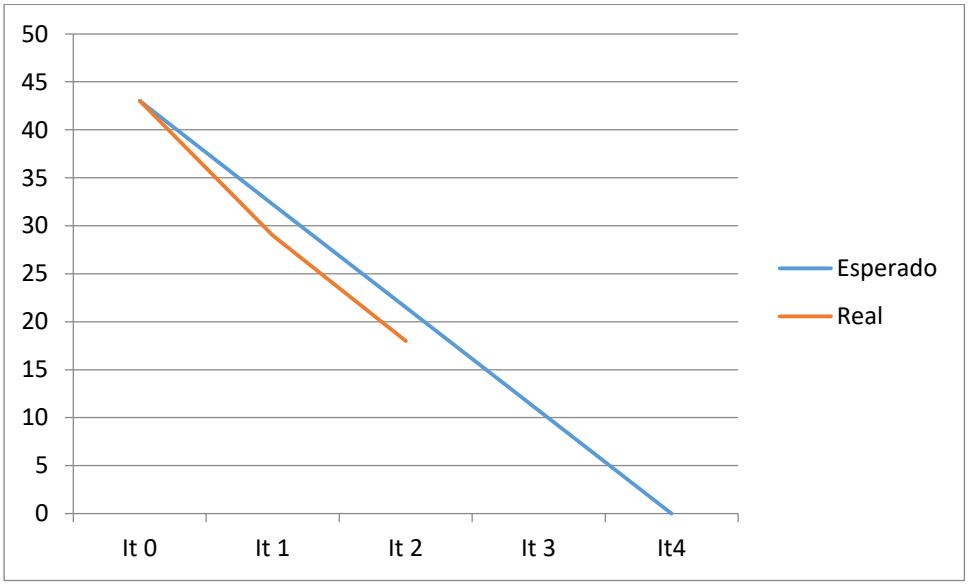

Ilustración 30: diagrama Burndown iteración 2

3.4 Tercera iteración

Estando ya finalizadas todas las funcionalidades en relación a los primeros dos proveedores de almacenamiento en la nube incorporados a la aplicación, se procede a realizar el menú que muestre tantas opciones como cuentas tengamos vinculadas (además de dos opciones que estarán siempre presentes, una para ver la ventana principal y otra para la de ajustes), cargando en cada opción los datos que correspondan. Además, se incorporará la barra de búsqueda que permitirá filtrar las tablas de archivos por cualquiera de los campos de información que se pueden

observar en las distintas columnas. La duración aproximada de esta iteración es de un mes.

3.4.1 Historias de usuario

Estas tareas reflejan los objetivos a cumplir en las historias de usuario HU-2 y HU-4. El desarrollo de esta iteración comenzará por la HU-4, implementando primero la búsqueda de archivos para proceder posteriormente a la creación del menú de manera dinámica, creándose en tiempo de ejecución y actualizándose cada vez que se vincule una nueva cuenta a la aplicación para que esta aparezca en el menú.

3.4.2 Detalles de implementación

Comenzando con la incorporación de una barra de búsqueda para filtrar los datos de la tabla de archivos por cualquier columna de esta. Se ha utilizado un TextField donde el usuario introducirá aquello que desee buscar. El valor de este control se limpia cada vez que se actualiza la tabla de archivos (tras realizar algunas operación como subida, bajada o borrado de archivos, así como después de vincular una nueva cuenta).

Para el filtrado de archivos, se ha creado un nuevo método *establecerFiltroBusqueda* dentro de la clase *ControladorPrincipal* que se llama cada vez que se rellena la tabla de archivos. En él, se utiliza una lista filtrada según un predicado donde se compara lo que va introduciendo el usuario con los valores de todas las filas en todas las columnas, independientemente de si se ha introducido en mayúscula o en minúscula. La lista una vez filtrada, con los datos que contenga coincidentes con lo que el usuario está buscando, se establece como contenido de la tabla. Este proceso se realiza cada vez que el usuario escribe o borra cualquier letra en la barra de búsqueda.

El siguiente paso en esta iteración es realizar un menú que se desplegará gracias a un botón que al pulsarlo mostrará al usuario las opciones de las que dispone en el menú de la aplicación. En primer lugar se mejora la interfaz de usuario que teníamos hasta ahora en la ventana principal, incorporando los layouts necesarios y diferenciando dos partes superpuestas en la ventana (Ilustración 31), la

parte izquierda del menú (incorporada dentro de un AnchorPane) y la parte central donde se incluyen la tabla, los botones correspondientes y la barra de búsqueda (encapsulado todo en otro AnchorPane). El botón encargado de desplegar el menú está incorporado en el segundo AnchorPane, de manera que en el AnchorPane del menú únicamente se incluyen todas las opciones del menú para poder desplegarlo correctamente

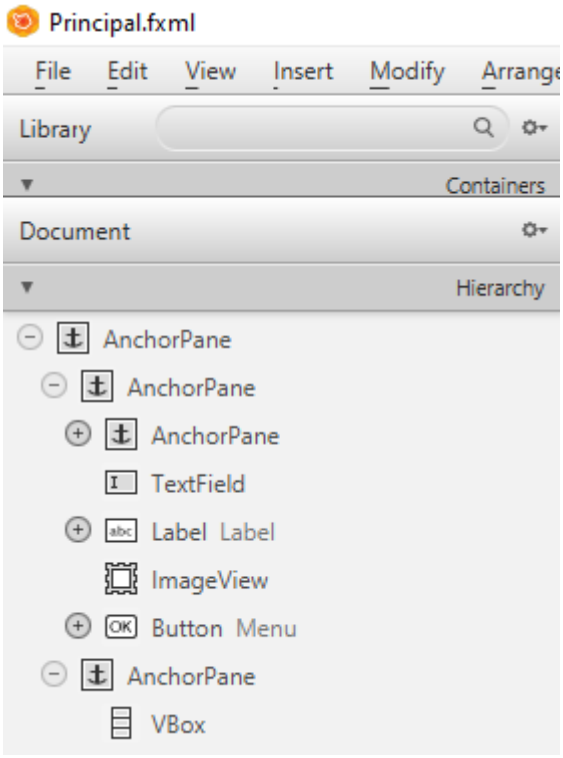

Ilustración 31: jerarquía de elementos de la vista principal

Al iniciar la aplicación, se posiciona el AnchorPane que contiene el menú (el que contiene únicamente el VBox) escondido de la vista del usuario. Finalmente, para desplegarlo, se utiliza un método asociado a la pulsación del botón que dependiendo de la posición en la que se encuentre el menú, los traslada a una u otra posición para mostrarlo o hacerlo visible al usuario mediante una transición de 0.5 segundos.

Una vez hecho esto, se rellena el menú proceduralmente con tantas opciones como cuentas estén vinculadas a la aplicación, además de dos opciones que siempre aparecerán: la opción de ajustes (que se incorporará siempre al final y permitirá acceder a la ventana de ajustes para poder vincular y desvincular cuentas) y la opción que nos proporciona el acceso a la ventana principal (incluida siempre como la primera) para ver de manera simultánea los archivos de todas las cuentas.

Para incluir tantas opciones como deseemos en el menú, se añaden en tiempo de ejecución tantos botones como correspondan al VBox mencionado anteriormente. Los botones correspondientes a las cuentas del usuario son creados incorporándoles el correo de la cuenta y un icono de la plataforma a la que esta corresponde, además de establecer los parámetros necesarios. Los botones para las opciones del menú *Todo* y *Ajustes* se hace de manera similar pero con los iconos y nombres apropiados. Este proceso se realiza al iniciar la aplicación y cada vez que se vincule o desvincule cualquier cuenta del sistema.

La tabla mostrada en la ventana principal se utilizará no solo para mostrar los archivos de todas las cuentas unificados en un solo listado, sino además, para mostrar los archivos de las cuentas individualmente en las opciones del menú correspondientes a cada cuenta. Para ello se ha creado un nuevo método *rellenarTablaCuenta* dentro de la clase *ControladorPrincipal* al que se le pasa la cuenta de la opción seleccionada en el menú para obtener y mostrar un listado de los archivos únicamente pertenecientes a la cuenta seleccionada. Cada vez que se seleccione cualquier opción que contenga la tabla de archivos, esta se limpia y se rellena con los datos correspondientes. Y en la opción que muestra los archivos de todas las cuentas se rellena la tabla como hasta ahora se había hecho.

Además, se ha incorporado en la ventana principal un Label junto a un ImageView encima de la tabla, a la derecha del botón del menú, que indicará al usuario donde se encuentra en cada momento. Esto aparece de manera general en el boceto inicial de la interfaz de usuario (Ilustración 8) y para que se vaya modificando según se cambie entre las distintas opciones del menú, se actualiza dentro de los métodos que rellenan la tabla de archivos con los datos de una cuenta, o todas.

Una vez hecho esto y para terminar por completo el menú, es necesario realizar el cambio de la tabla de archivos a la tabla de cuentas al entrar en *Ajustes* y viceversa al entrar en cualquier otra opción. Tras realizar distintas pruebas con JavaFX sobre diferentes maquetaciones que puedan cambiarse en tiempo de ejecución sin alterar el orden y las dimensiones de ninguna de las partes, se ha optado por crear un nuevo archivo .fxml con el tamaño correspondiente a la parte de la tabla y los botones inferiores (excluyendo la parte de la búsqueda y el Label de la

posición actual del usuario) consistente en un AnchorPane que contiene un botón para vincular nuevas cuentas y un TableView con las columnas necesarias: nombre de la cuenta, plataforma a la que pertenece, fecha en la que se vinculó a la aplicación, así como una opción para permitir la desvinculación cuando se desee. Este AnchorPane se añade y se elimina del AnchorPane que contiene la ventana principal dependiendo de la opción elegida por el usuario en el menú, es decir, en caso de seleccionar la opción de *Ajustes* se eliminará la tabla de archivos y se incluirá la de cuentas. De igual manera, en caso de seleccionar cualquier otra opción, se hará a la inversa.

En esta iteración solo se ha incluido el cambio de una ventana a otra, con todos los componentes que estas contengan pero sin la funcionalidad de los mismos. Esta tarea se llevará a cabo en la siguiente iteración.

3.4.3 Pruebas de verificación y validación

Después de concluir con las tareas de desarrollo e implementación descritas anteriormente a lo largo del periodo de esta iteración, es necesaria una verificación de los criterios de aceptación de las historias de usuario para comprobar el avance de la aplicación, siguiendo el orden en el que las historias de usuario objetivo de esta iteración se han desarrollado. La primera de ellas es la historia HU-4. Todos los criterios de aceptación se cumplen correctamente dado que se ha incluido una barra de búsqueda que permite al usuario filtrar y buscar en sus archivos por cualquiera de las columnas de la tabla, mostrando solo los archivos que contienen los datos buscados. Además de mostrar de nuevo todos los archivos cuando se eliminen los datos de la barra de búsqueda y poder buscar no solo en la ventana principal donde se nos muestran todos los archivos de las cuentas unificadas, sino que en las distintas opciones del menú (en las secciones individuales de cada cuenta) también es posible utilizar esta función.

Por último, se comprueban los criterios de aceptación de la historia HU-2, que ha sido la segunda y última en desarrollarse. El primero de los criterios que esta contiene, el CA-2.1 se cumple a la perfección, ya que la aplicación incorpora un menú con tantas opciones como cuentas haya vinculadas, de manera que al entrar en una u otra opción, se filtran los archivos, mostrando únicamente los de la cuenta

seleccionada. El segundo, el CA-2.2, queda satisfecho igualmente ya que al pasar entre las opciones del menú seleccionando la cuenta deseada, se listan los archivos pertenecientes únicamente a esta.

Tras comprobar que podemos dar por finalizadas correctamente las historias de usuario que se propusieron como objetivo de esta iteración, se actualiza la tabla resumen (Tabla 18) incluyendo los criterios que podemos dar por concluidos a la finalización de esta iteración. Además de los mencionados anteriormente, también se incluyen los CA-6.2 y CA-8.2 (ya que el .2 corresponde al proveedor de almacenamiento Dropbox). El primero de ellos se añade por hacer referencia al visionado de tantas opciones en el menú como cuentas haya vinculadas a la aplicación, tarea que ya ha sido implementada. El segundo, menciona el hecho de que al cerrar la aplicación y volver a abrirla, en el menú sigan estando las opciones que se encontraban al cerrar la aplicación, esto ha sido desarrollado para que sea de esta manera, por lo que damos por concluido también este criterio de aceptación.

		Iteración 1	Iteración 2	Iteración 3
HU-1	CA-1.1	X		
	CA-1.2	X		
HU-2	CA-2.1			X
HU-3.1	CA-3.1.1	X		
	CA-3.1.2	X		
	CA-3.1.3	X		
HU-3.2	CA-3.2.1		X	
	CA-3.2.2		X	
	CA-3.2.3		X	
HU-4	CA-4.1			X
HU-5.1	CA-5.1.1	X		
HU-5.2	CA-5.2.1		X	
HU-6	CA-6.1			
	CA-6.2			X
HU-7	CA-7.1			
HU-8	CA-8.1			

	CA-8.2			X
	CA-8.3	X		

Tabla 18: tabla resumen iteración 3

Por tanto y finalizando esta iteración, se actualiza también el diagrama Burndown (como podemos ver en la Ilustración 32) restando los puntos obtenidos en esta iteración tras concluir las historias HU-4 (3 puntos) y HU-2 (2 puntos).

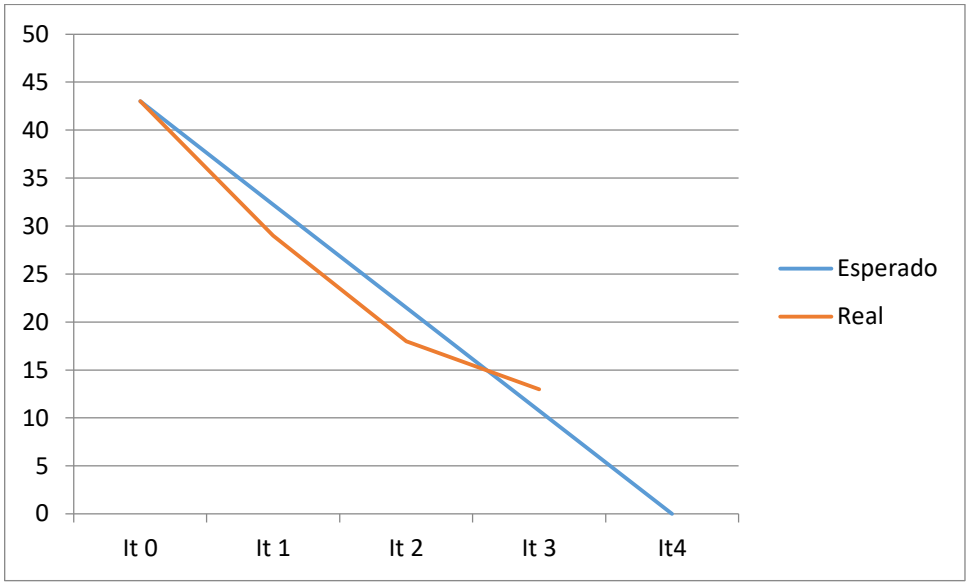

Ilustración 32: diagrama Burndown iteración 3

3.5 Cuarta iteración

Tras la creación del menú y el desarrollo de la navegación entre las distintas opciones que este dispone, así como la carga de unos u otros datos de archivos según la opción seleccionada, se pretende, en esta iteración, completar la vista de ajustes de manera que al abrirla, se muestre un listado con todos las cuentas vinculadas a la aplicación y la información más relevante de estas, además de la posibilidad de desvincularlas. También se incluirá la ventana para la vinculación de múltiples cuentas seleccionando el proveedor que se desea enlazar (eliminando los botones provisionales que se utilizaron al inicio del desarrollo de la aplicación y llevándolo a cabo de manera definitiva como se indicaba en el diseño inicial). La duración aproximada de esta iteración es de un mes.

3.5.1 Cambios de diseño

Comenzando con el rellenado de la tabla de ajustes, es necesaria una nueva clase para incluir toda la información necesaria relativa a cada cuenta como se hizo anteriormente con la tabla de archivos de la ventana principal. Para ello, tal y como se muestra en la Ilustración 33, se ha creado la nueva clase *FilaTablaCuentas*, donde se almacenarán los datos relativos a cada una de las cuentas enlazadas a la aplicación, incluyendo el botón que permite su desvinculación. Además, se ha incluido un nuevo atributo en la clase *Cuenta* llamado *fechaVinculacion* donde se almacenará la fecha exacta en la que realizó la vinculación. También se le ha cambiado el nombre a la clase *FilaTabla* utilizada en la tabla de archivos sustituyéndolo por *FilaTablaArchivos*.

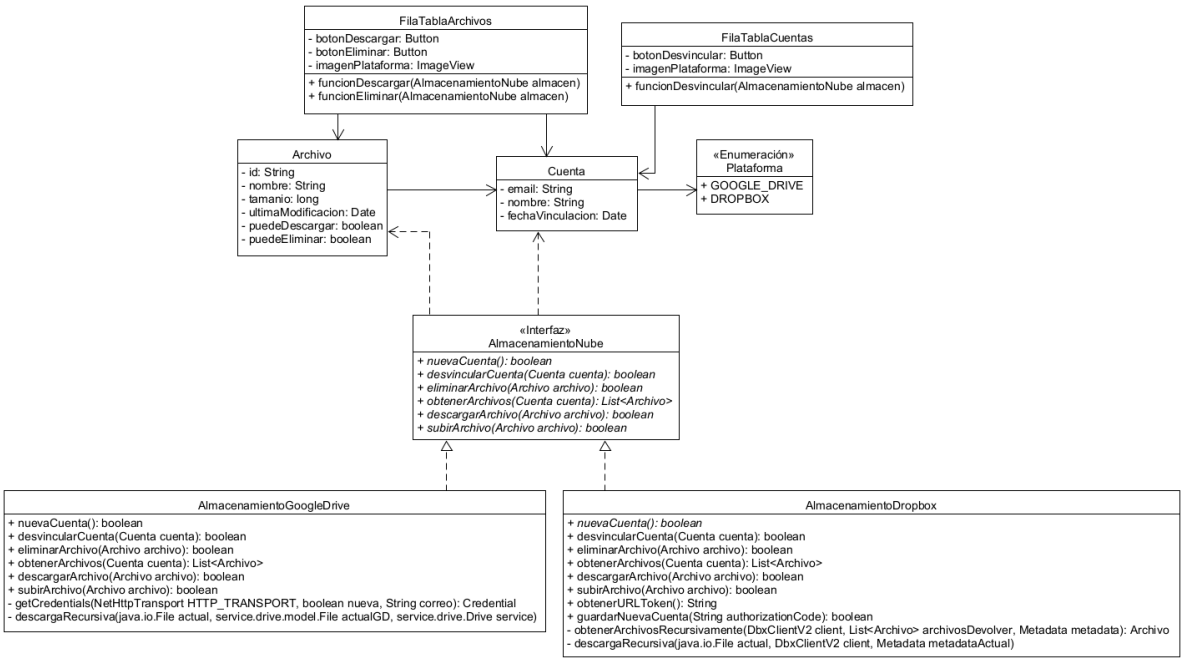

Ilustración 33: diagrama de clases del paquete Modelos modificado (FilaTablaCuentas)

De esta manera, cada vez que se lean las carpetas locales de la aplicación donde se almacenan los tokens de cada una de las cuentas vinculadas a esta, se obtiene la fecha de creación de los mismos, incluyéndola como atributo en cada uno de los objetos de la clase *Cuenta* almacenados en la estructura de datos correspondiente.

Con estos datos, cada vez que se pulse la opción de *Ajustes* dentro del menú, además de modificar el AnchorPane para eliminar la ventana principal e incluir los elementos correspondientes a la ventana de ajustes, se inicializarán todos los que

esta contenga, rellenando la tabla con los datos necesarios e implementando la funcionalidad de aquellos elementos que lo necesiten.

3.5.2 Historias de usuario

Las historias de usuario objetivo de esta iteración son las historias HU-6 y HU-7. Se comenzará en primer lugar por la historia HU-6, completando la tabla de cuentas con todos los datos pertinentes e incluyendo los botones que permitirán la desvinculación. Después se implementará dicha funcionalidad y finalmente se creará la ventana que permitirá vincular cuentas a la aplicación (y a la que se accede a través de un botón en la ventana de ajustes), dejando esta parte de la aplicación finalizada por completo.

3.5.3 Detalles de implementación

Para rellenar la tabla, en primer lugar se obtiene esta del archivo .fxml (es decir, de la vista) así como sus respectivas columnas, estableciendo los valores de las mismas según los distintos atributos de la clase *FilaTablaCuentas*. De manera similar a como se hizo con la tabla de datos, y haciendo uso de la misma barra de búsqueda, se implementa un filtro de búsqueda por cada una de las columnas que la tabla de cuentas contiene, pudiendo realizar búsquedas sobre esta. Finalmente, se incluyen en la tabla tantas filas como cuentas se encuentren en dicho momento vinculadas a la aplicación. Además, se modifica el Label superior que indica en qué opción del menú se encuentra el usuario, estableciendo la sección de *Ajustes* con el icono correspondiente como la actual.

Según se crean cada una de las filas que completan la tabla, se establece también la función de desvinculación de las mismas. Para ello, la clase *FilaTablaCuentas* incluye un método público llamado *funcionDesvincular* (Ilustración 33) al que se llamará al ir añadiendo cuentas a la tabla y al que se le pasa el proveedor de almacenamiento correspondiente, ya que desde este se llamará al método *desvincularCuenta* del proveedor de almacenamiento en la nube al que pertenezca la cuenta.

Para la desvinculación de una cuenta con Google Drive se establece un flujo de conexión con la cuenta que se desea desvincular para acceder al almacenamiento del token correspondiente en Google Drive, procediendo a la

eliminación del mismo, así como al borrado de los archivos locales propios de la aplicación donde se almacenaba el token. De manera similar pero haciendo uso de las herramientas proporcionadas por la API de Dropbox, se elimina el token del usuario almacenado en Dropbox así como el archivo local correspondiente. Sin embargo, ambos proveedores limitan la posibilidad de desvincular cuentas al no permitir su completa realización, de manera que aunque la aplicación desarrollada ya no tenga acceso a la nube del usuario, seguirá apareciendo como vinculada dentro de la aplicación web del propio proveedor. Para avisar de esto al usuario, se lanza una ventana de alerta indicando lo sucedido. Y finalmente, una vez hecho todo esto, se actualiza la tabla de cuentas, mostrando la desvinculación de la cuenta tal y como se había deseado, además del menú, donde también se elimina la opción de la cuenta que se acaba de eliminar.

Tras terminar esta parte del desarrollo, se crea una nueva vista que se abrirá al pulsar el botón *Vincular* de la ventana de ajustes y que permitirá seleccionar con qué proveedor de los soportados por la aplicación se desea vincular una nueva cuenta. Esta selección se lleva a cabo mediante un selector (un ChoiceBox de la librería JavaFX) en el que se incluirán tantas opciones como proveedores haya implementados en la aplicación, comprobando para ello la clase de enumeración *Plataforma* que podemos observar en el diagrama de clases del paquete *Modelos*. La ventana ha sido ligeramente modificada, tal y como se puede ver en la Ilustración 34, respecto a la original del boceto inicial (Ilustración 13), ya que como las conexiones se realizan a través de la web, se han eliminado los campos de correo electrónico y contraseña donde introducir las credenciales. Según sea seleccionado por el usuario uno u otro proveedor de entre todos los posibles, al pulsar el botón de *Vincular* se llamará al método que realiza una nueva conexión, llevándose a cabo toda la funcionalidad implementada en la primera y segunda iteración para poder enlazar múltiples cuentas tanto de Google Drive como de Dropbox. Con todo esto completado y totalmente funcional, esta iteración se da por finalizada, quedando todo lo relativo a la ventana de ajustes finalizado.

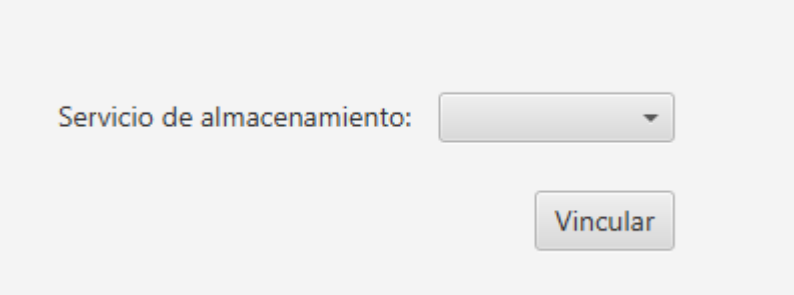

Ilustración 34: cambio en la ventana que permite vincular nuevas cuentas

3.5.4 Pruebas de verificación y validación

Una vez implementadas y desarrolladas todas las funcionalidades objetivo de esta iteración, se procede a la verificación de su correcta finalización mediante la comprobación de los criterios de aceptación de las historias de usuario que se propusieron al inicio de esta iteración, comenzando con la historia HU-6, por ser la primera en la que se ha trabajado y que se ha concluido.

De los dos criterios de aceptación que contiene esta historia de usuario, el segundo de ellos, el CA-6.2, ya había sido completado y comprobado en la tercera iteración, al crear el menú de la aplicación con la inclusión de opciones para cada una de las cuentas vinculadas en la aplicación. El CA-6.1 puede verificarse en esta iteración dado que en la ventana de ajustes se permite al usuario visualizar todas las cuentas enlazadas a la aplicación en una tabla junto a información relevante a las mismas, es decir, la fecha en la que se realizó la conexión, el proveedor de almacenamiento en la nube al que pertenecen y la opción de desvincularlas. Con este criterio de aceptación también satisfecho, esta historia de usuario está finalizada y se pasa a la verificación de la siguiente, de la historia HU-7.

El CA-7.1, está correctamente completado con la funcionalidad esperada, dado que en la tabla de cuentas de la ventana de ajustes se incluye para cada una de las cuentas existentes un botón para su desvinculación. Este proceso se puede llevar a cabo dado que la desvinculación de cuentas ha sido implementada para Google Drive y Dropbox, los dos proveedores actualmente existentes en la aplicación. Además, el último criterio de aceptación de esta historia, el CA-7.2, también se satisface correctamente ya que una vez desvinculada una cuenta y eliminados sus datos de la aplicación, los archivos que pertenecen a esta dejan de aparecer en el listado de archivos.

De esta manera, podemos ver en la Tabla 19 de forma resumida los criterios de aceptación cumplidos a lo largo de esta iteración y como se han completado con estos, todos los criterios de aceptación de las historias de usuario descritas al comienzo del desarrollo de este proyecto. También se puede ver cómo, a pesar de no haber sido comentado anteriormente, se ha visto cumplido el CA-8.1 puesto que una vez cerrada y abierta de nuevo la aplicación, al entrar en la ventana de ajustes aparecerán en todo momento reflejadas las cuentas vinculadas en todo momento a la aplicación, manteniendo el estado de la misma.

		Iteración 1	Iteración 2	Iteración 3	Iteración 4
HU-1	CA-1.1	X			
	CA-1.2	X			
HU-2	CA-2.1			X	
HU-3.1	CA-3.1.1	X			
	CA-3.1.2	X			
	CA-3.1.3	X			
HU-3.2	CA-3.2.1		X		
	CA-3.2.2		X		
	CA-3.2.3		X		
HU-4	CA-4.1			X	
HU-5.1	CA-5.1.1	X			
HU-5.2	CA-5.2.1		X		
HU-6	CA-6.1				X
	CA-6.2			X	
HU-7	CA-7.1				X
HU-8	CA-8.1				X
	CA-8.2			X	
	CA-8.3	X			

Tabla 19: tabla resumen iteración 4

Por tanto y dando por finalizadas las historias HU-6 (5 puntos), HU-7 (3 puntos) y HU-8 (5 puntos), completando un total de 13 puntos en esta iteración, se

actualiza el diagrama Burndown, tal y como observamos en la Ilustración 35, alcanzando el objetivo de 0 puntos restantes.

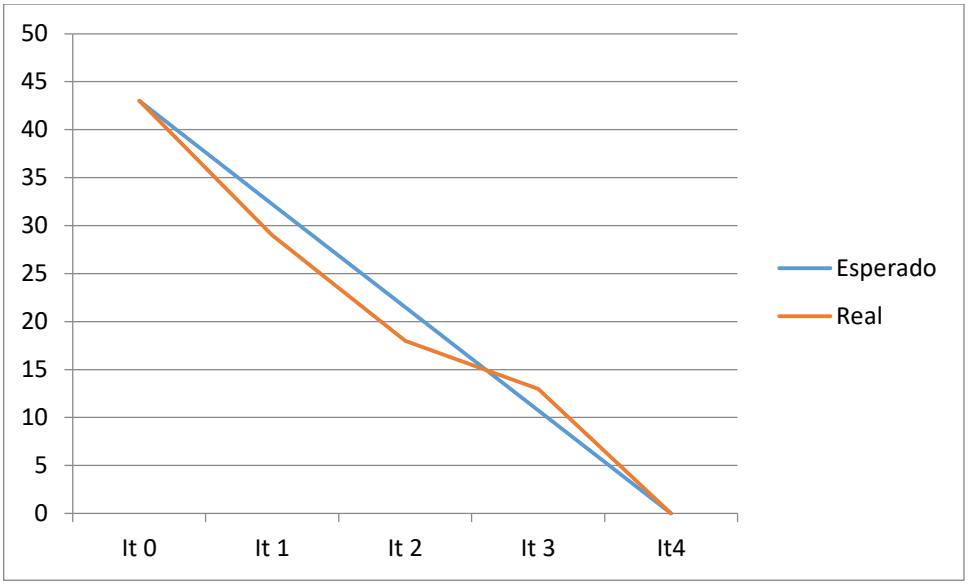

Ilustración 35: diagrama Burndown iteración 4

De esta manera se finalizan todas las historias de usuario que se fijaron como especificaciones del sistema al comienzo del desarrollo de este proyecto, dando por concluido también el desarrollo de la aplicación.

3.6 Trabajos finales

Una vez terminada toda la funcionalidad propia de la aplicación definida en el apartado 2.1 y dado que para la entrega queda un plazo sobrante, se han añadido unos trabajos finales que incorporan mejoras a la aplicación como la introducción del número de archivos totales en cada una de las opciones de la aplicación, es decir, tanto en el listado de archivos unificados como de cuentas individuales.

Además, esta parte de trabajos finales se ha centrado especialmente en una mejora de la interfaz de usuario, dándole color a todos los elementos, incluyendo también efectos como cambiar el color al pasar sobre un elemento o al hacer click en él. Todo esto comprobando que los colores sean claramente perceptibles y distinguibles para todo tipo de personas independientemente de los problemas

visuales (como daltonismo) que puedan tener. Esto se ha confirmado gracias a la página web Paletton[28] que nos ofrece ver todas las diferencias perceptibles.

Esta tarea de finalización de la interfaz de usuario se ha llevado a cabo incluyendo en el archivo styles.css que se creó inicialmente, todas las clases necesarias y los estilos que requieren cada una de estas.

También se han incluido unos Tooltip, es decir, unas etiquetas de información sobre herramientas, en los botones de descarga y eliminación de archivos (dado que el nombre de la columna ha sido borrado) que aparecen sobre dichos botones cuando el ratón se deja sobre estos.

La interfaz de usuario concluida por completo y de manera satisfactoria con toda la funcionalidad implementada puede observarse en las ilustraciones 36, 37, 38, 39, 40 y 41.

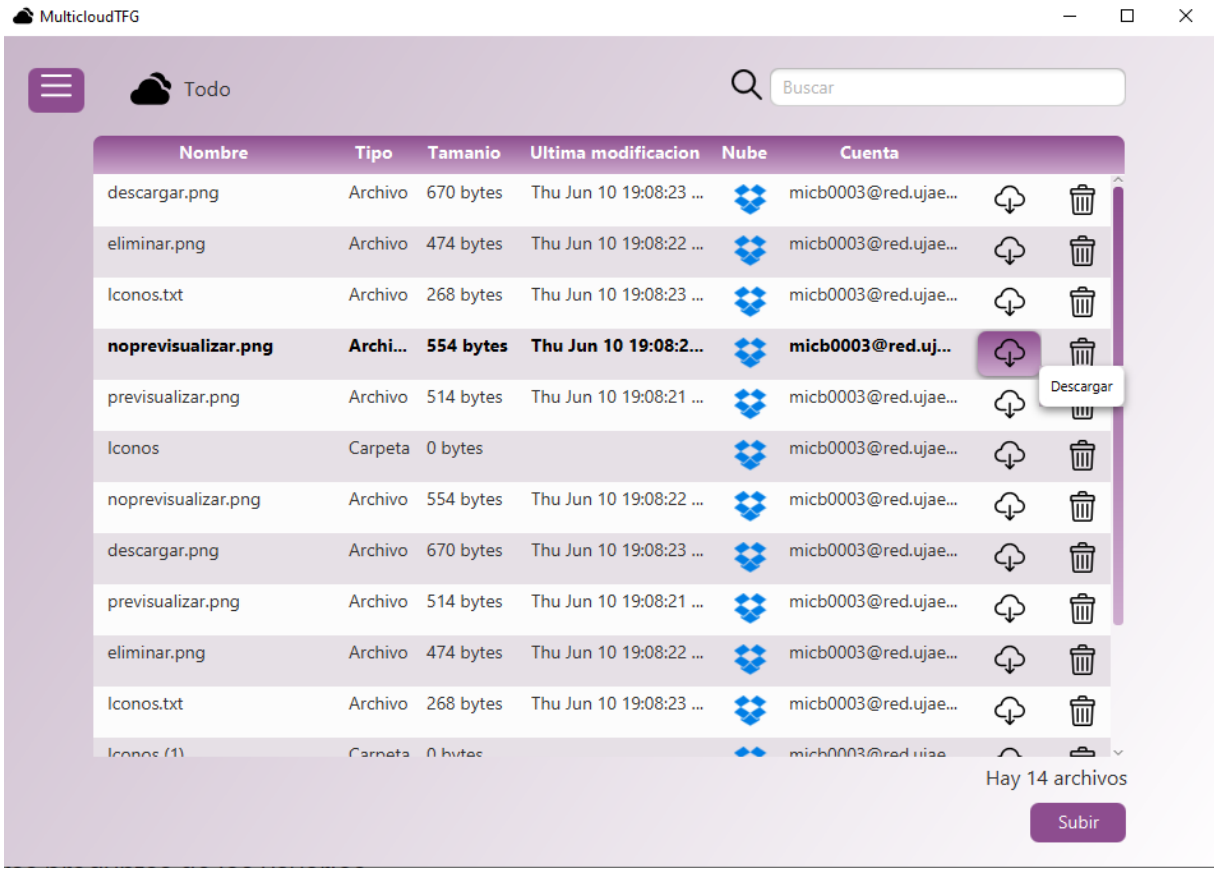

Ilustración 36: ventana principal aplicación finalizada

[28] https://paletton.com/#uid=12A0u0kllllaFw0g0qFqFg0w0aF

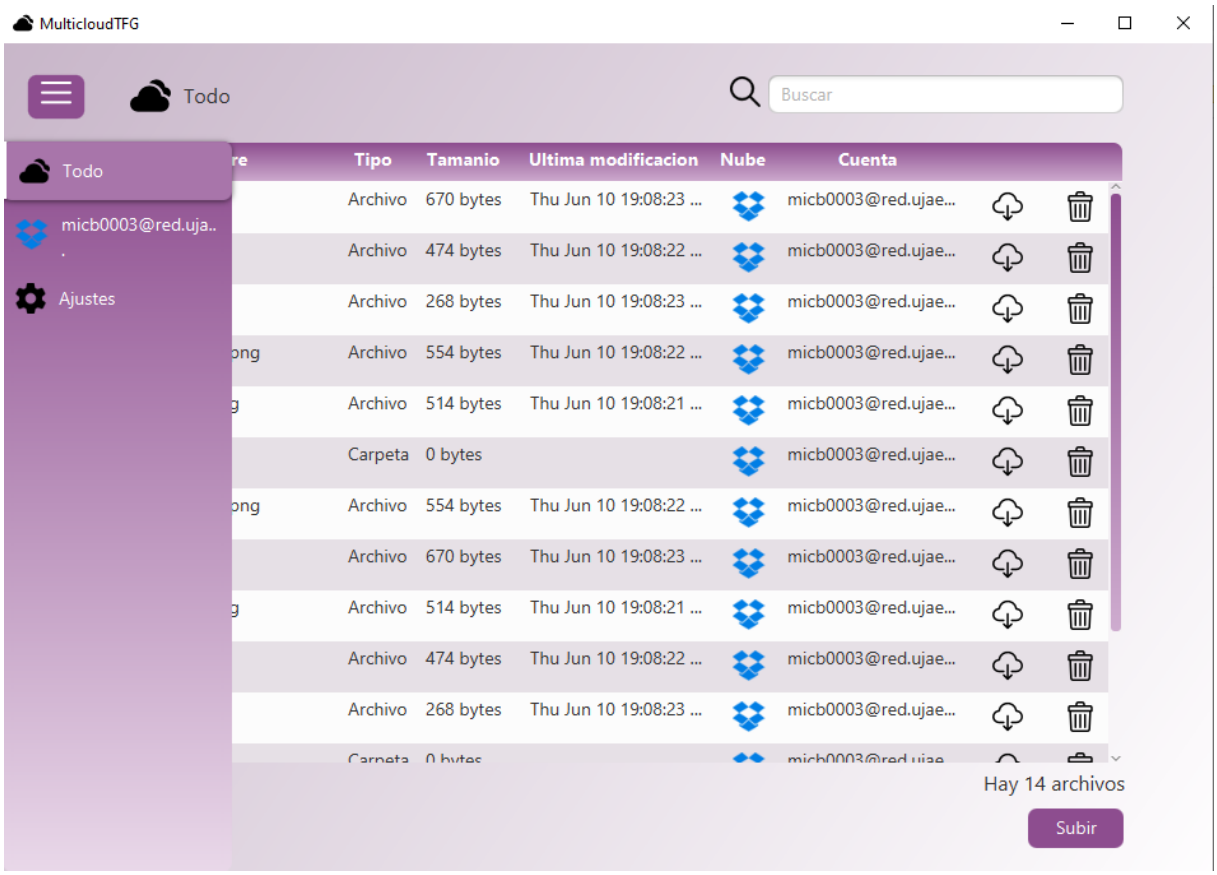

Ilustración 37: ventana principal aplicación finalizada con menú desplegado

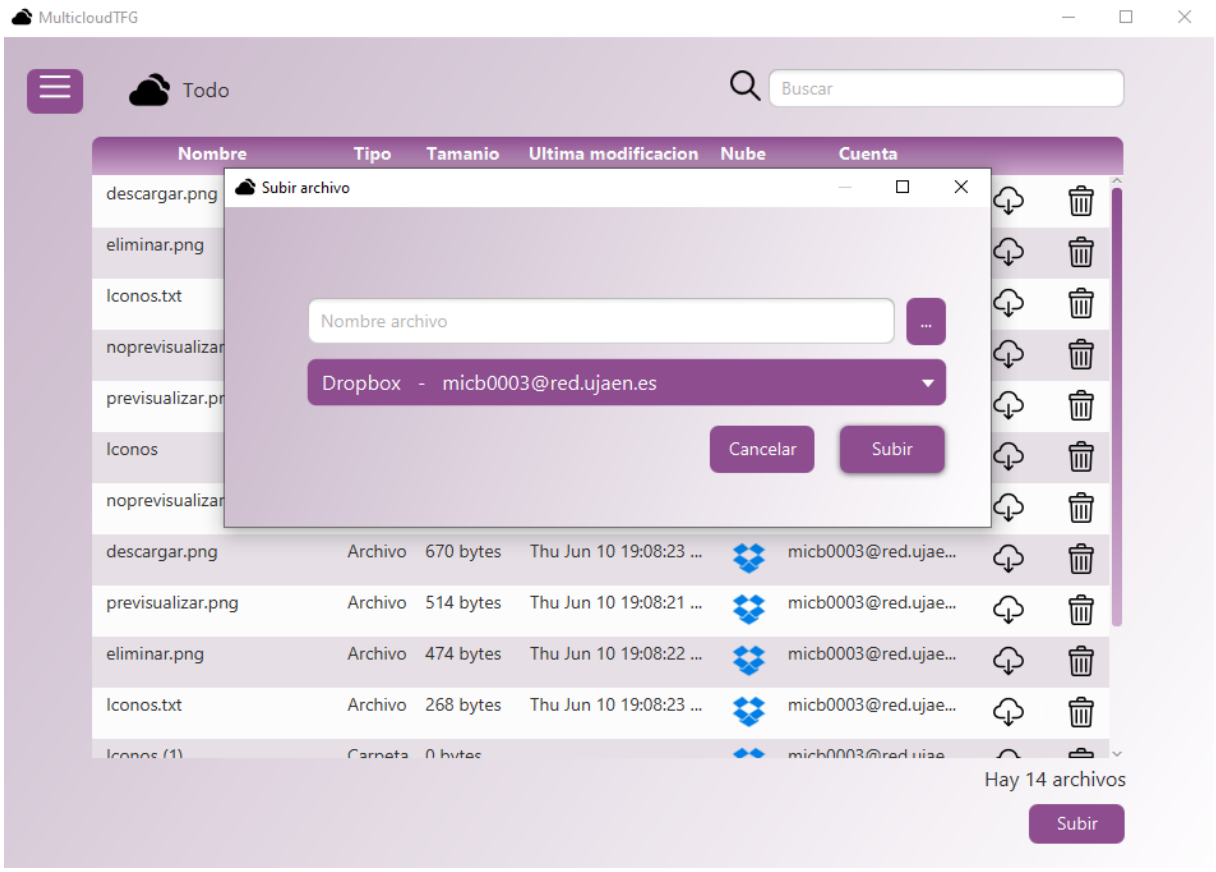

Ilustración 38: subir archivos aplicación finalizada

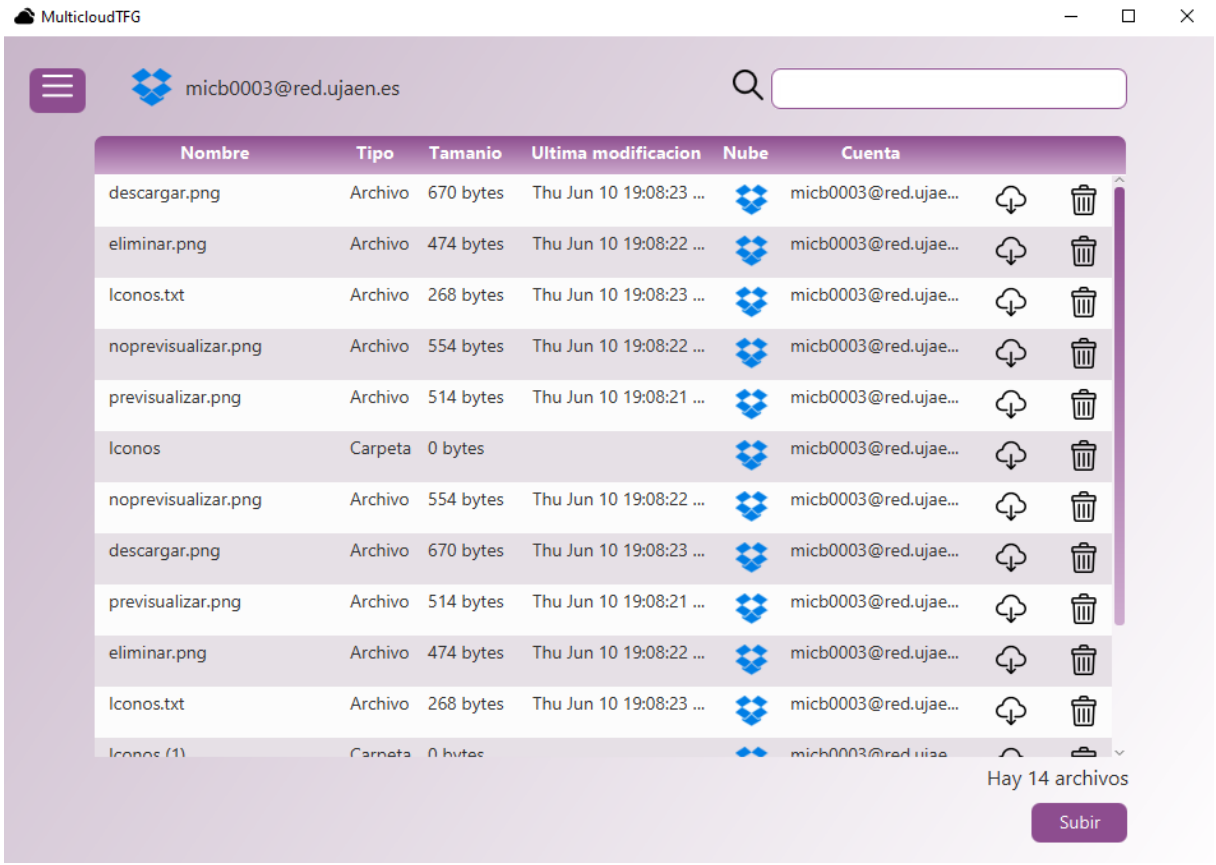

Ilustración 39: vista de archivos individuales aplicación finalizada

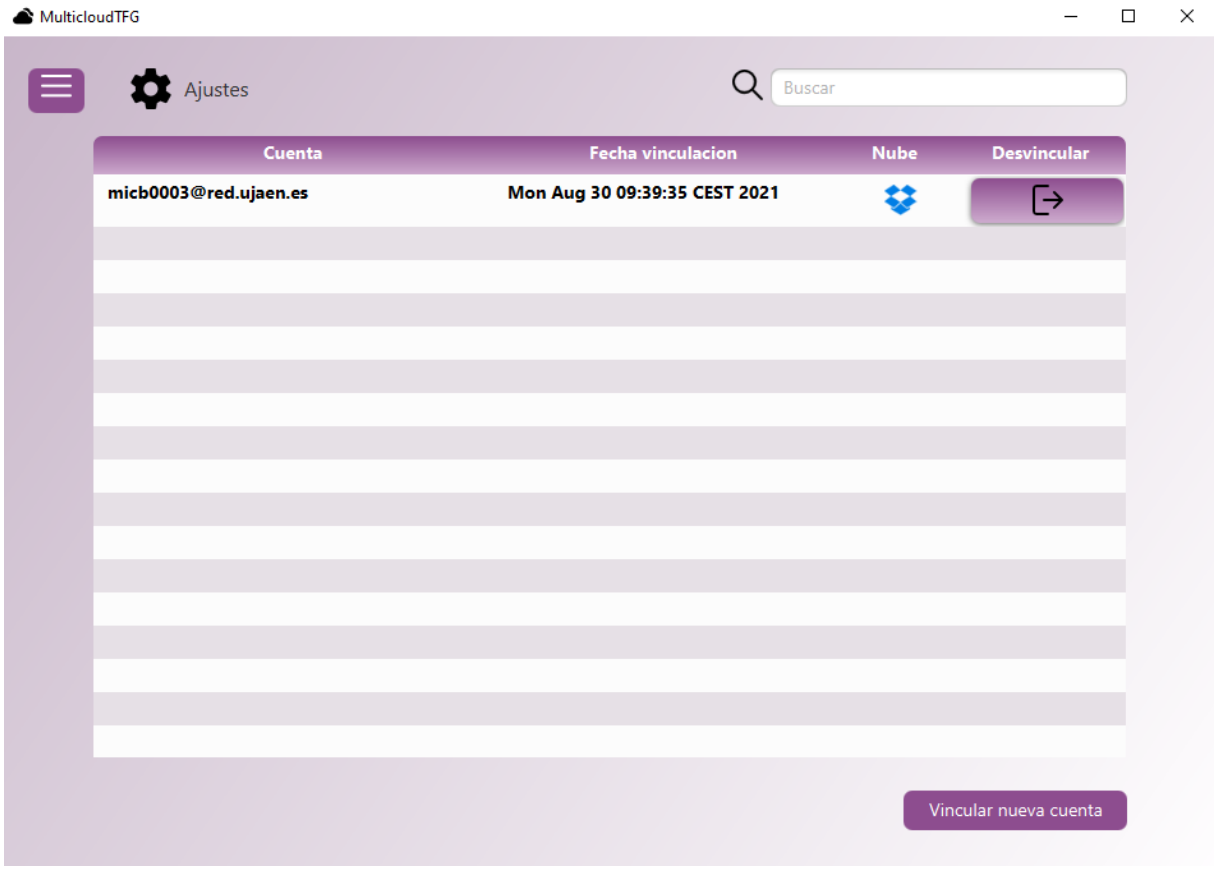

Ilustración 40: vista de ajustes aplicación finalizada

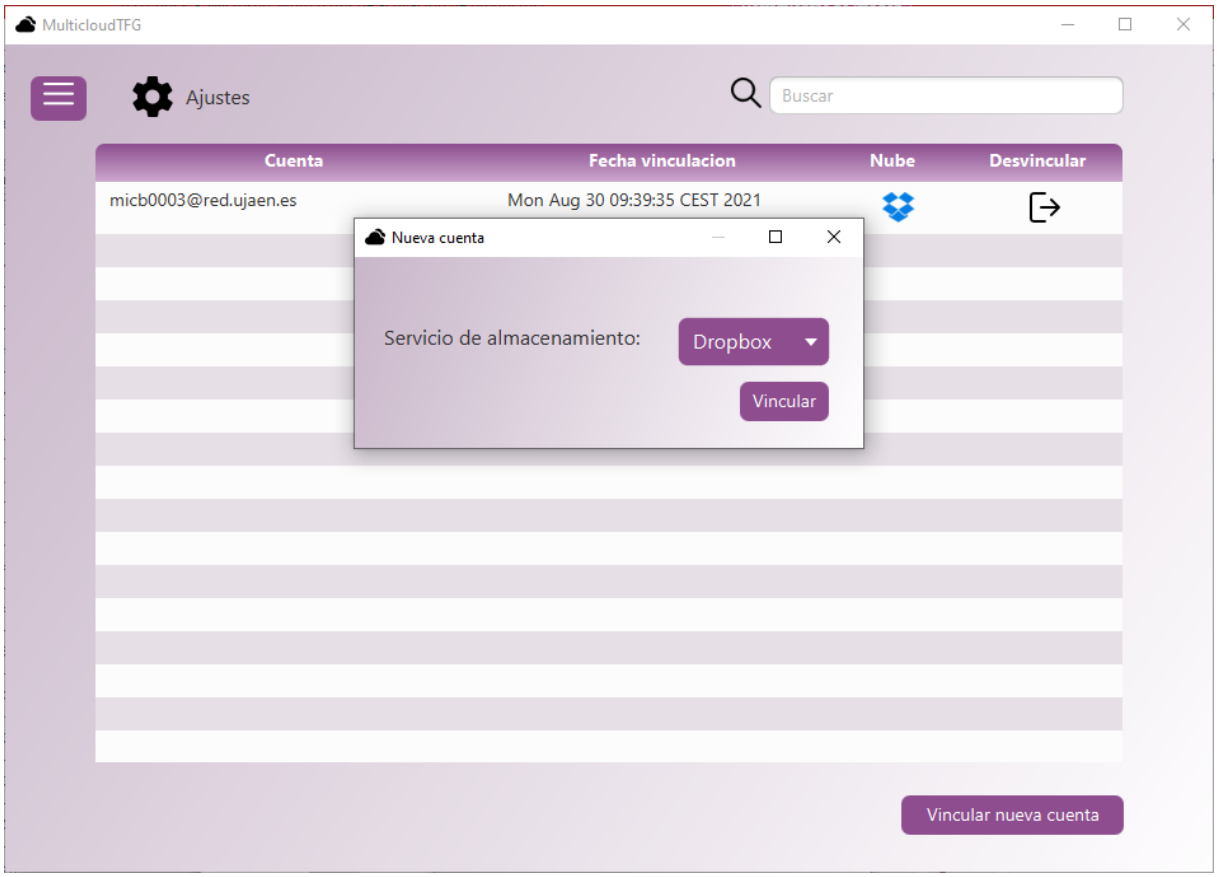

Ilustración 41: nueva cuenta aplicación finalizada

3.7 Pruebas finales

Al tratarse de un proyecto desarrollado mediante una metodología ágil basada en iteraciones, ha sido tras la finalización de cada una de estas cuando se han ido realizando las pruebas pertinentes para la comprobación de la calidad del software así como del cumplimiento de todos los criterios de aceptación definidos inicialmente en el apartado 2.1. Tal y como se puede observar en la Tabla 19, una vez acabada la cuarta iteración han sido verificados todos los criterios de aceptación por lo que la aplicación puede darse por terminada correcta y satisfactoriamente.

4 CONCLUSIONES Y TRABAJOS FUTUROS

Tras concluir con el desarrollo de este proyecto tal y como se había previsto en la planificación temporal, se plantean una serie de mejoras que podrían implementarse en un futuro al continuar avanzando con el desarrollo de la aplicación.

Una de estas mejoras sería la implementación de paginación en el listado de archivos, mostrando por ejemplo de 50 en 50 archivos, de manera que disminuya el tiempo de respuesta al listar los datos al usuario, además de eliminar la posibilidad de colapso en la tabla de datos al no cargar todos los datos de golpe.

Además, la aplicación ha sido diseñada y desarrollada para permitir la inclusión de más proveedores de almacenamiento en la nube, siguiendo las instrucciones indicadas en el apartado 5.4. De esta manera, ésta sería otra mejora a tener en cuenta, añadiendo proveedores como Box, OneDrive o MediaFire.

Puesto que durante el uso de la aplicación se muestran mensajes informativos al usuario que durante un tiempo prolongado puede llegar a resultar molesto, se propone como trabajo futuro la incorporación de la posibilidad de eliminar de manera permanente este tipo de mensajes, dando está opción al usuario mediante un check y guardando está opción para el resto de ocasiones en que se abra la aplicación, mediante algún fichero de configuración.

Otra mejora que es viable como trabajo futuro, es la inclusión de una opción que permita al usuario cambiar los colores de la interfaz gráfica entre una serie de predefinidos.

También se ha pensado en incluir un botón en todas las vistas que listen archivos para actualizar las tablas de datos correspondientes, para que, en caso de que el usuario haya realizado modificaciones entre sus archivos mientras la aplicación está en uso y desea actualizarlos de nuevo, pueda llevar a cabo esta tarea.

Además de las funciones ya establecidas para cada uno de los archivos, también podría incorporarse un botón más en la tabla que permita la previsualización de los mismos sin necesidad de descargarlos.

Finalmente, podrían incluirse unos mensajes de confirmación al eliminar cuentas o archivos, preguntando al usuario si realmente está seguro antes de realizar la acción.

Como conclusiones obtenidas una vez que se ha completado con éxito el proyecto objetivo de este documento, puedo decir que he adquirido un mayor conocimiento en metodología ágil al trabajar con ella en un trabajo real en el que he tenido que marcar desde un inicio, con las historias de usuario y los criterios de aceptación, las bases de esta metodología de desarrollo de software, siguiendo con el desarrollo de las distintas iteraciones y las verificaciones oportunas a la

finalización de cada una de estas. En este proyecto no solo he adquirido un mayor conocimiento y destreza con la metodología ágil sino que también he avanzado con la utilización de APIs externas, la inclusión de estas en un proyecto y su correcto uso. También me he adentrado más en profundidad en el mundo de la nube, cómo este funciona (lo cual me ha resultado bastante interesante) y la evolución que está teniendo. Como última conclusión, decir que, el proceso de documentación de un proyecto que hasta ahora no había llevado a cabo de una forma tan profunda, me ha servido para adquirir conocimientos sobre cómo se realiza una documentación de un proyecto informático y lo más importante a tener en cuenta siempre que esto se lleve a cabo.

5 APÉNDICES

Como información adicional que puede resultar útil para una mayor comprensión del proyecto se incluyen en este apartado como distintas subsecciones, la guía original del TFG, un manual de instalación del programa y un manual de usuario del mismo, además de una guía para el desarrollador para la inclusión de nuevos proveedores de almacenamiento en la nube con los que pueda trabajar la aplicación.

5.1 Guía original del Trabajo Fin de Título

Título TFG: Desarrollo de un prototipo de agregador de almacenamiento en la nube
Código: 20/21-2711
Tutor TFG: Ángel Luis García Fernández
Idioma: Castellano
Modalidad: Proyecto de ingeniería | Tipo: TFG específico
Número máximo de estudiantes: 1

Descripción corta del TFG

El objetivo del TFG es realizar el análisis, diseño e implementación de una aplicación de escritorio multiplataforma que permita gestionar de forma transparente varias cuentas de proveedores de almacenamiento en la nube.

Conocimientos/requisitos previos recomendados

Tener conocimientos medios/avanzados de Java

Objetivos del TFG

- Poner en práctica todas las capacidades adquiridas durante el periodo de formación del estudiante.
- Desarrollar una aplicación con utilidad real.
- Aumentar los conocimientos del estudiante sobre aplicaciones que hacen uso intensivo de almacenamiento remoto, con todas las posibles casuísticas que puedan aparecer.

Metodología a desarrollar

- Estudio de las distintas APIs de programación que ofrecen las webs que proveen de almacenamiento remoto.
- Análisis de requerimientos.
- Diseño de la aplicación.
- Implementación.
- Pruebas.
- Durante todo el proceso se redactará la documentación asociada, así como un manual de usuario.

Documentos y formatos de entrega (en formato electrónico)

- El código fuente de la aplicación.
- Una versión ejecutable (incluyendo si fuera necesario un instalador).
- La memoria del TFG, incluyendo el manual de usuario.

5.2 Instalación del sistema

Los pasos necesarios para instalar la aplicación objeto de este proyecto en cualquier equipo (al tratarse de una aplicación multiplataforma) son los siguientes:

1. Instalar Java 11 o superior.
2. Descomprimir el archivo .zip que contiene el ejecutable y las credenciales necesarias.
3. Ejecutar el archivo .jar (desde consola o haciendo doble click) y se abrirá la aplicación

Opcionalmente, se puede crear un acceso directo en el escritorio para acceder de manera más rápida y sencilla.

5.3 Manual de usuario

Para un correcto uso de la aplicación, se muestra a continuación una serie de indicaciones sobre el uso de cada uno de los elementos que esta contiene.

En la Ilustración 42 se observan los componentes de la ventana principal cuya funcionalidad es la siguiente:

1. Botón de menú hamburguesa que nos permite desplegar o esconder el menú según la posición en la que se encuentre (si el menú está cerrado, entonces al pulsar se abrirá y al revés).
2. Texto con icono que nos informa de la opción del menú en la que nos encontramos (ya sea *Todo* donde se encuentran todas las cuentas unificadas, de alguna cuenta individual o en *Ajustes*).
3. Barra de búsqueda que nos permite filtrar los datos de las diferentes tablas por cualquiera de los campos que estas contengan entre sus columnas. Al escribir o borrar letra a letra se va realizando el filtrado, independientemente de si se escribe en mayúscula o minúscula.
4. Botón que permite la descarga de un archivo de la tabla de archivos. Aparecerá siempre que sea posible realizar esta acción sobre el fichero concreto.
5. Botón que permite el borrado de un archivo de la tabla de archivos (y por consecuente de la nube). Aparecerá siempre que sea posible realizar esta acción sobre el fichero concreto.

6. Texto informativo sobre el número de archivos totales que existen en la tabla de archivos que se encuentre seleccionada (ya sea de todos los archivos unificados o de los de una sola cuenta).
7. Botón que abre una ventana modal para la subida de archivos.

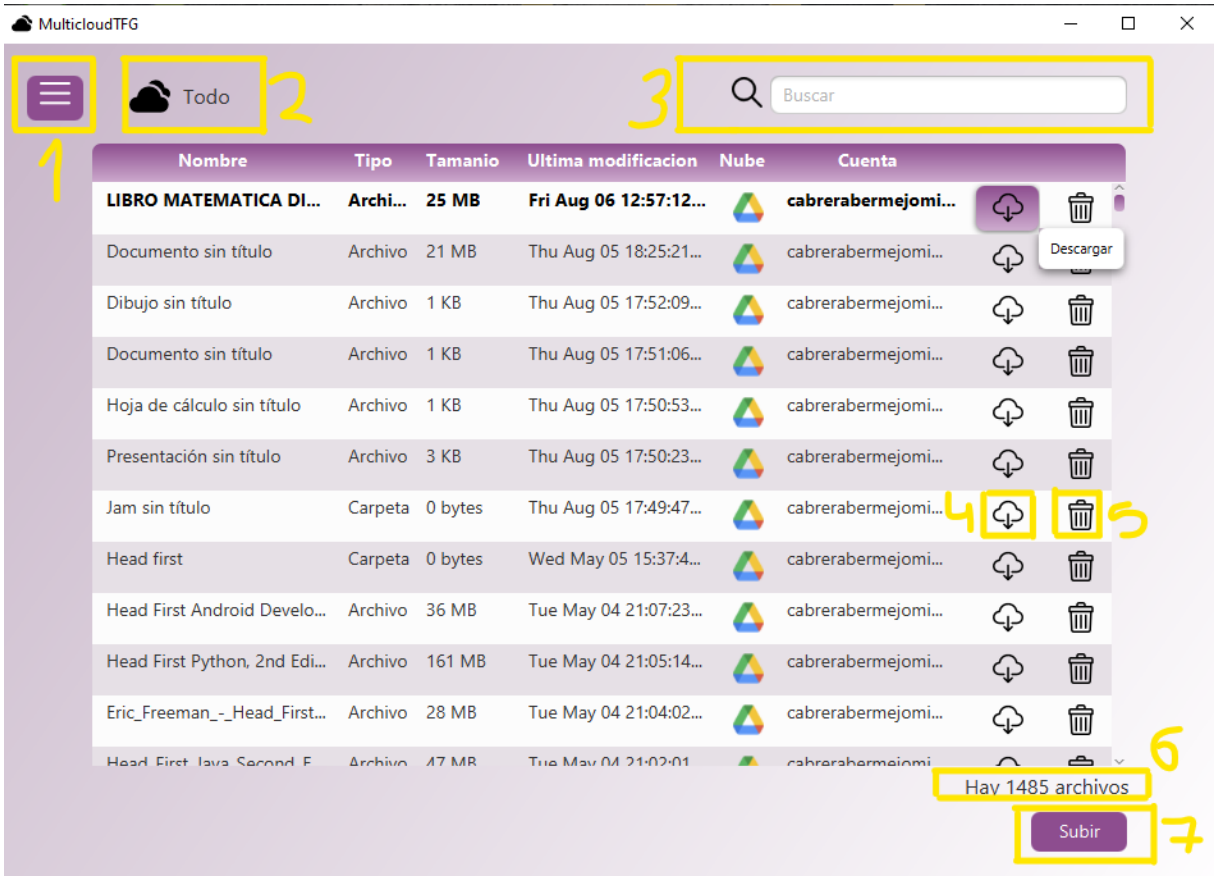

Ilustración 42: manual de usuario ventana principal

La ventana modal que se abre al pulsar el elemento 7, se muestra en la Ilustración 43, junto a los siguientes elementos de la interfaz de usuario:

8. Campo de texto sobre el que no se puede escribir, que muestra al usuario los archivos seleccionados del equipo para subir a la nube.
9. Botón que permite acceder a los archivos locales del equipo para seleccionar uno o múltiples archivos que se desean subir a la nube.
10. Selector donde se selecciona de entre todas las cuentas vinculadas a la aplicación, aquella a la que se desean subir los archivos elegidos,

mostrando el proveedor de almacenamiento en la nube al que pertenece la cuenta y el correo electrónico correspondiente.

11. Botón para cancelar la operación de subida de archivos que cierra la ventana sin llevar a cabo la operación.
12. Botón para realizar la subida de archivos a la cuenta seleccionada, cerrando además la ventana modal.

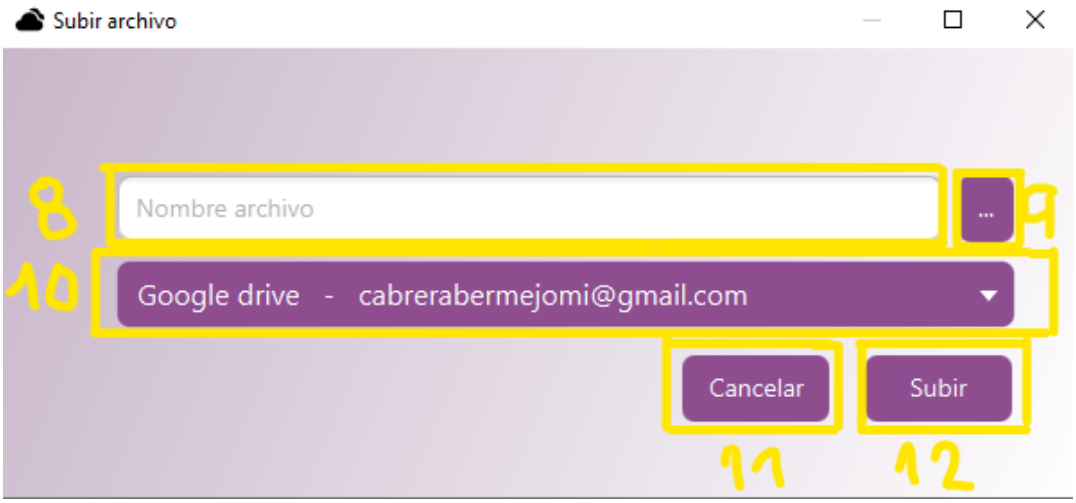

Ilustración 43: manual de usuario subir archivo

La vista de *Ajustes* se muestra en la Ilustración 44. En ella, aparecen elementos que ya se han explicado anteriormente y además otros que se exponen a continuación.

13. Botón para desvincular cada una de las cuentas mostradas en la tabla de cuentas. Aparecerá un botón por cada fila.
14. Botón que abre la ventana modal que permite la vinculación de cuentas nuevas a la aplicación.

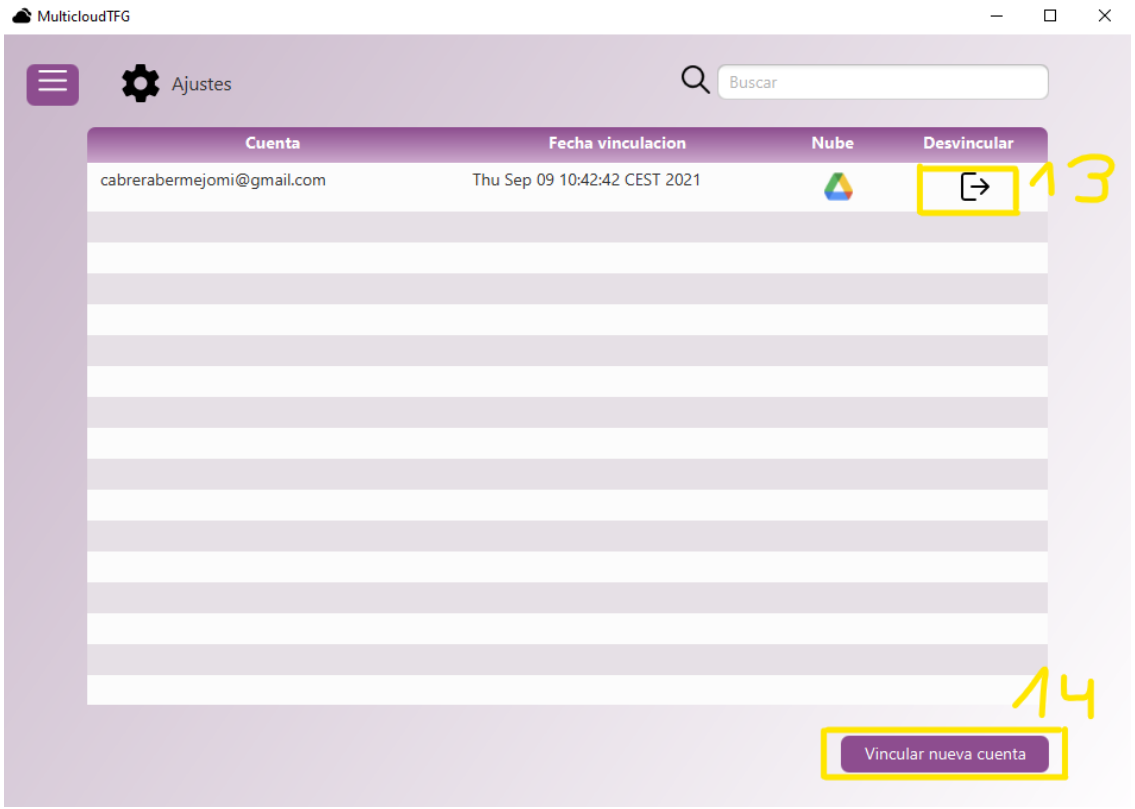

Ilustración 44: manual de usuario ajustes

La ventana modal para la vinculación de una nueva cuenta la podemos observar en la Ilustración 44, junto a los elementos propios de esta.

15. Selector que nos permite elegir con qué proveedor de almacenamiento en la nube de los soportados por la aplicación queremos vincular una nueva cuenta.

16. Botón que comienza el proceso para enlazar una nueva cuenta. Actualmente la aplicación trabaja con dos proveedores: Google Drive y Dropbox. En caso de haber seleccionado Google Drive, se abre el navegador para iniciar sesión con la cuenta correspondiente y otorgar los permisos. Sin embargo, si se elige Dropbox, se abre una ventana que informa al usuario de los pasos a seguir ya que la API de Dropbox trabaja de manera distinta y es necesario que el usuario intervenga.

Ilustración 45: manual de usuario nueva cuenta

La ventana que informa al usuario de los pasos a seguir para enlazar una cuenta de Dropbox se muestran en la Ilustración 46 junto a los elementos que esta contiene.

17. Campo de texto donde hay que introducir el código proporcionado por Dropbox a través de la web (tras y como se muestra en el texto que indica los pasos a seguir).
18. Botón que abre el navegador para la obtención del código que hay que introducir en el campo de texto.
19. Botón para finalizar el proceso de vinculación de una cuenta con Dropbox gracias al código introducido previamente en el campo de texto.

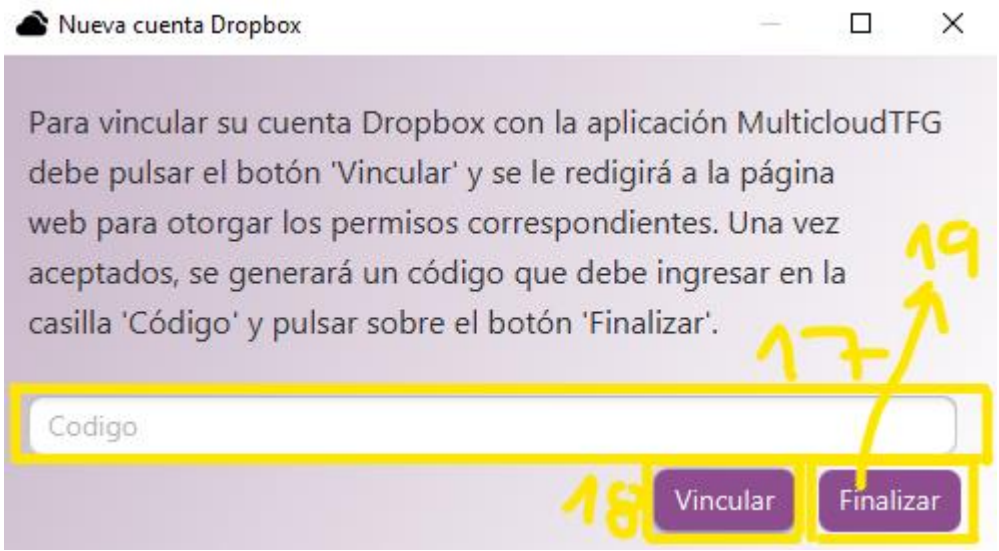

Ilustración 46: manual usuario nueva cuenta Dropbox

5.4 Manual de desarrollador

En este apartado se detalla como incluir un nuevo proveedor de almacenamiento en la nube en la aplicación, aumentando así la funcionalidad de la misma.

Este proceso requiere la incorporación del nuevo proveedor a la clase de enumeración *Plataforma* (dentro de la clase Constantes) tal y como está hecho para Google Drive y Dropbox.

Una vez hecho esto, se crea una nueva clase dentro del paquete *Modelos* con nombre similar al resto de clases encargadas de la funcionalidad de los distintos proveedores (por ejemplo, *AlmacenamientoBox*, *AlmacenamientoOneDrive* etc.) y se hace que esta clase implemente la interfaz *AlmacenamientoNube* con todos los métodos abstractos que esta contiene. En cada uno de estos métodos se hace uso de la API del proveedor para implementar la funcionalidad correspondiente.

Para que aparezca una imagen con el logo del proveedor indicando a cual pertenece cada cuenta en todas las tablas que contiene la aplicación, se debe incluir un logo del proveedor que se desea añadir a la aplicación en la carpeta */src/main/resources/imágenes* en tamaño 32x32 píxeles.

Finalmente, se incluye como caso en los switch que utilicen esta clase de enumeración, diferenciando de esta manera cuándo coger cada una de las distintas clases que implementan la interfaz.

Dado que la aplicación está desarrollada para utilizar tantos proveedores como existan sin necesidad de ningún otro cambio, el añadir uno nuevo es una tarea muy sencilla para la que solo hay que seguir los pasos indicados en este apartado.

6 DEFINICIONES Y ABREVIATURAS

En este apartado se muestran listadas las definiciones y abreviaturas utilizadas a lo largo de este trabajo, incluyendo su significado.

- IaaS (Infraestructure as a Service), permite acceso a una infraestructura hardware proporcionada por un proveedor y que es compartida entre los distintos usuarios que acceden a ella.

- PaaS (Platform as a Service), proporciona una plataforma para el desarrollo de software, mantenimiento o gestión del mismo de manera virtual y sin complejidades en las capas de infraestructura o redes al no trabajar con ellas.
- SaaS (Software as a Service), orientado más al usuario final, ya que permite el acceso a software alojado en la nube, ignorando tareas de mantenimiento y/o desarrollo.
- APIs (Interfaz de Programación de Aplicaciones), conjunto de métodos, funciones y subrutinas que ofrece una biblioteca y permite ser utilizada por otro software como capa de abstracción.
- RESTful API, estilo arquitectónico que utiliza peticiones HTTP para el acceso a datos.
- SDK (Kit de Desarrollo de Software), conjunto de herramientas que permiten al desarrollador crear una aplicación para un sistema concreto.
- AWT (Abstract Window Toolkit), biblioteca proporcionada por Java para la creación de interfaces de usuario, con multitud de herramientas.
- Arquitectura o patrón MVC (Modelo-Vista-Controlador), separa los datos de la lógica de la aplicación y de la representación que esta tiene en las vistas.
- CSS (Hoja de estilos en cascada), lenguaje de diseño gráfico utilizado para la definición y creación de estilos para los elementos de una interfaz gráfica.
- Ficheros FXML (Lenguaje de declaraciones basado en XML), lenguaje para declaraciones de interfaces basado en XML.
- IDE (Entorno Integrado de Desarrollo), aplicación informática que proporciona todos los servicios necesarios para que un desarrollador pueda crear un software de manera cómoda y sencilla.
- LTS (Long Term Support), término informático para nombrar aquellas versiones que tendrán soporte durante un periodo de tiempo más largo de lo normal.

- GUI (Interfaz Gráfica de Usuario), programa que realiza la tarea de interfaz de usuario mediante elementos gráficos que representan la información y las acciones que se pueden llevar a cabo.

7 BIBLIOGRAFÍA

Al tratarse de un proyecto informático en cuyo trascurso se ha utilizado como principal fuente de información la documentación de las APIs utilizadas, se han ido incluyendo los enlaces a las páginas más importantes como notas al pie de página a lo largo de todo este documento. Por ello, a continuación se presentan únicamente dos libros, uno relativo a las metodologías ágiles y otro relacionado con todo tipo de diagramas UML.

Andrew Stellman, J. G. (2017). *Head First Agile: A Brain-Friendly Guide to Agile Principles, Ideas, and Real-World Practices.* O'Reilly Media, Inc, USA.

Kim Hamilton, R. M. (2006). *Learning UML 2.0: A Pragmatic Introduction to UML.* O'Reilly Media;.