Universitat Oberta de Catalunya

uoc.edu

S3 Cliente Web

Jesús Calvo López
Grado de Ingeniería Informática
Java EE

Antoni Oller Arcas
Santi Caballe Llobet

19 de junio de 2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	*S3 Cliente Web*
Nombre del autor:	*Jesús Calvo López*
Nombre del consultor:	*Antoni Oller Arcas*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega:	*06/2023*
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Java EE*
Idioma del trabajo:	*Español*
Palabras clave:	*S3 Angular SpringBoot*

Resumen del Trabajo (máximo 250 palabras):

Este Trabajo Fin de Grado implementa un cliente web para la gestión de almacenes de objetos en la nube. En concreto los que son compatibles con el API S3 de Amazon. También se permite gestionar conexiones a dichos repositorios, que pueden ser suministrados por distintos proveedores, tales como Amazon AWS, DigitalOcean, Cloudian, NetApp, Akamai, DreamHost , MinIO, Wasabi y Vultr.

Actualmente existen productos con esta funcionalidad, pero normalmente son aplicaciones de escritorio, como por ejemplo Filezilla Pro, S3 Browser o WinSCP. Amazon también proporciona la herramienta AWS CLI, pero es una interfaz de línea de comandos. La aplicación web implementada en este trabajo puede ser interesante para una persona o empresa que trabaje con múltiples plataformas y utilice frecuentemente estos servicios, ya que le permitirá gestionar todos sus repositorios independientemente del proveedor.

Para el desarrollo se ha optado por una metodología ágil, definiendo al principio un mapa de historias de usuario y dividiendo las tareas en varias iteraciones o *sprints*. La aplicación web está implementada con Angular 15 que se comunica con un API REST basada en Spring Boot. Los datos se persisten en una base de datos relacional PostgreSQL.

En el Trabajo se observa el ciclo completo de desarrollo de un proyecto software, utilizando metodologías y tecnologías actuales. El resultado es una aplicación sencilla y con un conjunto de funcionalidades mínimo, aunque el proyecto podría evolucionar fácilmente en caso de continuar su desarrollo.

Abstract (in English, 250 words or less):

This Final Degree Project implements a web client for the management of object stores in the cloud. Specifically, those that are compatible with Amazon's S3 API. It also allows to manage connections to these repositories, which can be provided by different suppliers, such as Amazon AWS, DigitalOcean, Cloudian, NetApp, Akamai, DreamHost, MinIO, Wasabi and Vultr.

There are currently products with this functionality, but they are usually desktop applications, such as Filezilla Pro, S3 Browser or WinSCP. Amazon also provides the AWS CLI tool, but it is a command line interface. The web application implemented in this work may be interesting for a person or company that works with multiple platforms and frequently uses these services, as it will allow them to manage all their repositories independently of the provider.

An agile methodology has been chosen for the development, defining a user story map at the beginning, and dividing the tasks into several iterations or sprints. The web application is implemented with Angular 15 that communicates with a REST API based on Spring Boot. The data is persisted in a relational PostgreSQL database.

The Degree Project observes the complete development cycle of a software project, using current methodologies and technologies. The result is a simple application with a minimum set of functionalities, although the project could easily evolve if it is developed further.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del trabajo

Para este Trabajo Fin de Grado (TFG) se pretende implementar un cliente web para la subida de ficheros a un repositorio en la nube compatible con el API de Amazon S3 (1) (Amazon Simple Storage Service).

Dicho API es un estándar de facto para el almacenamiento de objetos en la nube (*object storage*) tal y como se explica en el artículo de Julia Palmer publicado por Gartner en 2022 titulado "*Magic Quadrant for Distributed File Systems and Object Storage*" (2). En este artículo se analizan detalladamente los distintos proveedores de estos servicios, pero tiene un apartado de visión general del mercado (*market overview*) en el que explica que los mercados para sistemas de ficheros distribuidos y almacenamiento de objetos se han fusionado, y precisamente la estandarización del API S3 de Amazon es uno de los factores que ha contribuido a ello.

El proyecto consistiría en una aplicación web que permitiría gestionar estos repositorios o espacios de almacenamientos en la nube, o *buckets* en la terminología de Amazon. Dichos espacios de almacenamiento podrían ser suministrados por distintos proveedores, tales como Amazon AWS (3), DigitalOcean (4), Cloudian (5), NetApp (6), Akamai (7) (recientemente fusionado con Linode), DreamHost (8), MinIO (9), Wasabi (10) y Vultr (11).

Existen productos comerciales con esta funcionalidad, pero normalmente son aplicaciones de escritorio, como por ejemplo Filezilla Pro (12) o S3 Browser (13). Como cliente web, normalmente cada plataforma en la nube tiene su aplicación que permite realizar la gestión del contenido, pero son exclusivas para su plataforma. Por ejemplo, Amazon tiene su aplicación web para subir objetos a un *bucket*, y MinIO tiene las suyas. Para una persona o empresa que trabaje con múltiples plataformas y utilice frecuentemente estos servicios, puede ser interesante utilizar una única aplicación, desde donde gestionar todos sus repositorios independientemente del proveedor.

Amazon también proporciona la herramienta AWS CLI (14), que permite realizar la gestión de contenidos conectándose a distintos proveedores. Sin embargo, tal y como se indica en su nombre, se trata de una interfaz de línea de comandos (CLI, *Command Line Interface*). Esta herramienta es muy potente, especialmente para administradores de sistema que necesiten realizar tareas de mantenimiento. Sin embargo, es poco intuitiva para un usuario ocasional, que únicamente necesite subir de vez en cuando algún contenido, o comprobar el contenido existente.

1.2. Objetivos del Trabajo

El objetivo principal de la aplicación es poder gestionar el contenido de dichos espacios de almacenamiento, es decir:

- Subir ficheros individualmente o en lote, mediante la selección de ficheros individuales o de directorios.
    - Sería interesante que se implementase *resumable upload*, por si se interrumpe la subida de un fichero pesado, de varias gigas, poder continuar más adelante sin tener que comenzar desde cero.

- Descargar ficheros individualmente
- Eliminar ficheros.

Además, cada usuario podrá gestionar y configurar una o varias conexiones a distintos repositorios (URL, credenciales, etc.) Las credenciales almacenadas, por motivos de seguridad, deberán estar encriptadas.

De igual modo se pretende que la aplicación sea fácil de utilizar y que no requiera de instalación en el cliente.

1.3. Enfoque y método seguido

Se han evaluado varias herramientas alternativas que hay en el mercado. En concreto se han evaluado S3 Browser (13), Filezilla Pro (12) y WinSCP (15) que son aplicaciones de escritorio con interfaz gráfica de usuario, y también se ha evaluado Amazon AWS CLI (14), que es una herramienta con interfaz por línea de comandos.

Todas las aplicaciones requieren instalación en el cliente, lo cual puede ser un problema en algunos entornos. Por ejemplo, en entornos corporativos, la instalación de aplicaciones puede estar restringida por motivos de seguridad.

Por otro lado, tal y como se ha indicado anteriormente, la interfaz de línea de comandos puede ser muy potente para usuarios administradores de sistema, ya que permite su uso desde scripts, sin embargo, para la mayor parte de usuarios suele ser intimidante, y requiere una curva mayor de aprendizaje.

Es por ello que para el enfoque a seguir, se ha optado por realizar una aplicación web. Dicha aplicación debería poder tener todas las ventajas de las aplicaciones del escritorio desde el punto de vista de la interfaz de usuario.

Dicha aplicación web interactuará con servicios que proporcionarán el contenido de los distintos proveedores. Es decir, tendremos una capa de servicios que actuará como un concentrador o hub de servicios ofrecidos por distintos proveedores de almacenamiento en la nube.

Por tanto, la estrategia a seguir será crear un producto nuevo, basado en una aplicación web y una serie de servicios web, que proporcionará una alternativa interesante a los productos existentes.

Durante el análisis se definirá el *User Story Mapping* (16) y para representar el tablero se utilizará la herramienta Miro (17). Para el prototipo se consideró utilizar Figma (18), aunque finalmente se han realizado los bosquejos (sketches) en la misma herramienta que el tablero, Miro.

El desarrollo será iterativo y se dividirá el tiempo en tres periodos o *sprints* iguales en tamaño, entregando al final de cada periodo la funcionalidad finalizada.

1.4. Planificación del Trabajo

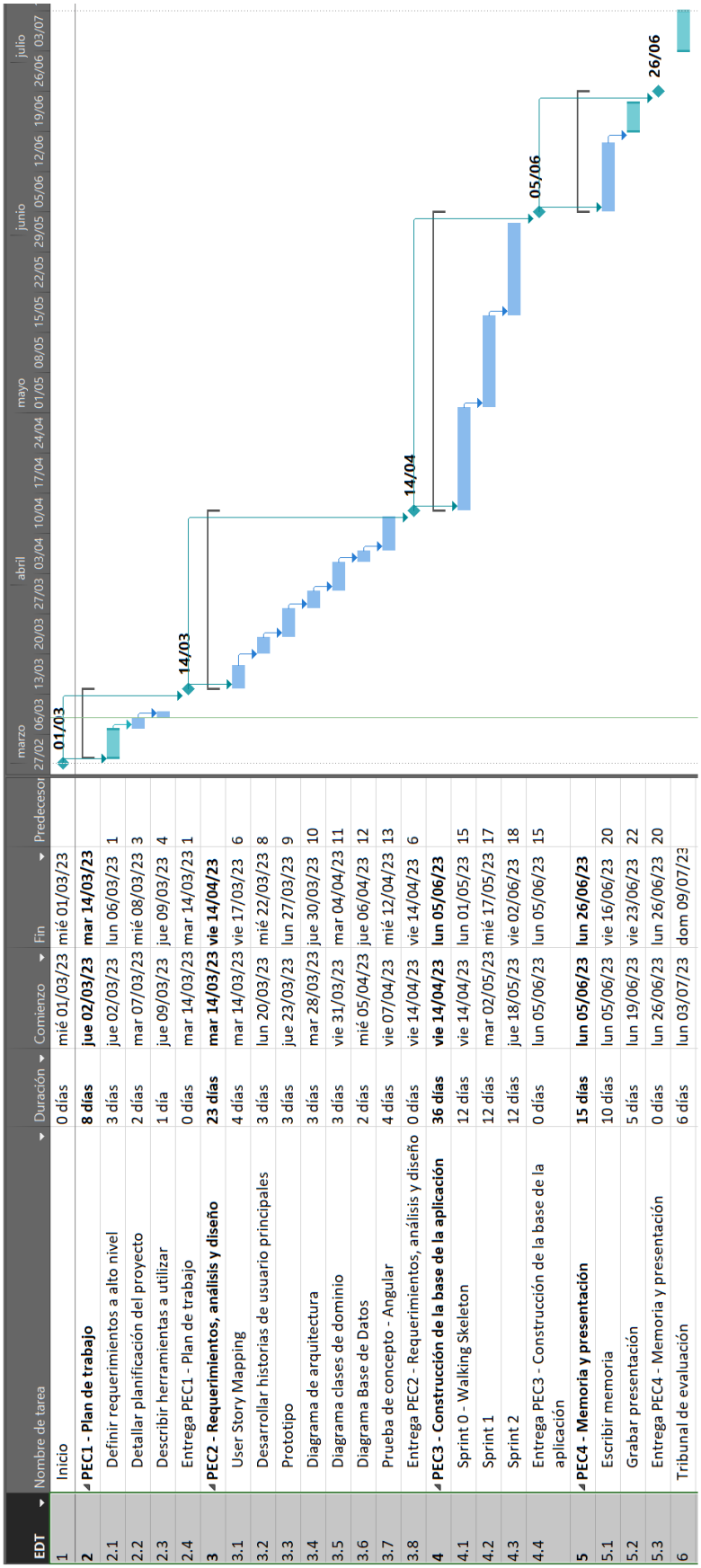
A continuación, se incluye una planificación detallada del trabajo, que consta de la lista de tareas y de un diagrama de Gantt.

Lista de tareas

EDT	Nombre de tarea	Duración	Comienzo	Fin
1	Inicio	0 días	mié 01/03/23	mié 01/03/23
2	**PEC1 - Plan de trabajo**	**8 días**	**jue 02/03/23**	**mar 14/03/23**
2.1	Definir requerimientos a alto nivel	3 días	jue 02/03/23	lun 06/03/23
2.2	Detallar planificación del proyecto	2 días	mar 07/03/23	mié 08/03/23
2.3	Describir herramientas a utilizar	1 día	jue 09/03/23	jue 09/03/23
2.4	Entrega PEC1 - Plan de trabajo	0 días	mar 14/03/23	mar 14/03/23
3	**PEC2 - Requerimientos, análisis y diseño**	**23 días**	**mar 14/03/23**	**vie 14/04/23**
3.1	User Story Mapping	4 días	mar 14/03/23	vie 17/03/23
3.2	Desarrollar historias de usuario principales	3 días	lun 20/03/23	mié 22/03/23
3.3	Prototipo	3 días	jue 23/03/23	lun 27/03/23
3.4	Diagrama de arquitectura	3 días	mar 28/03/23	jue 30/03/23
3.5	Diagrama clases de dominio	3 días	vie 31/03/23	mar 04/04/23
3.6	Diagrama Base de Datos	2 días	mié 05/04/23	jue 06/04/23
3.7	Prueba de concepto - Angular	4 días	vie 07/04/23	mié 12/04/23
3.8	Entrega PEC2 - Requerimientos, análisis y diseño	0 días	vie 14/04/23	vie 14/04/23
4	**PEC3 - Construcción de la base de la aplicación**	**36 días**	**vie 14/04/23**	**lun 05/06/23**
4.1	Sprint 0 - Walking Skeleton	12 días	vie 14/04/23	lun 01/05/23
4.2	Sprint 1	12 días	mar 02/05/23	mié 17/05/23
4.3	Sprint 2	12 días	jue 18/05/23	vie 02/06/23
4.4	Entrega PEC3 - Construcción de la base de la aplicación	0 días	lun 05/06/23	lun 05/06/23
5	**PEC4 - Memoria y presentación**	**15 días**	**lun 05/06/23**	**lun 26/06/23**
5.1	Escribir memoria	10 días	lun 05/06/23	vie 16/06/23

5.2	Grabar presentación	5 días	lun 19/06/23	vie 23/06/23
5.3	Entrega PEC4 - Memoria y presentación	0 días	lun 26/06/23	lun 26/06/23
6	Tribunal de evaluación	6 días	lun 03/07/23	dom 09/07/23

Diagrama de Gantt



EDT	Nombre de tarea	Duración	Comienzo	Fin	Predecesor
1	Inicio	0 días	mié 01/03/23	mié 01/03/23	
2	**PEC1 - Plan de trabajo**	**8 días**	**jue 02/03/23**	**mar 14/03/23**	
2.1	Definir requerimientos a alto nivel	3 días	jue 02/03/23	lun 06/03/23	1
2.2	Detallar planificación del proyecto	2 días	mar 07/03/23	mié 08/03/23	3
2.3	Describir herramientas a utilizar	1 día	jue 09/03/23	jue 09/03/23	4
2.4	Entrega PEC1 - Plan de trabajo	0 días	mar 14/03/23	mar 14/03/23	1
3	**PEC2 - Requerimientos, análisis y diseño**	**23 días**	**mar 14/03/23**	**vie 14/04/23**	
3.1	User Story Mapping	4 días	mar 14/03/23	vie 17/03/23	6
3.2	Desarrollar historias de usuario principales	3 días	lun 20/03/23	mié 22/03/23	8
3.3	Prototipo	3 días	jue 23/03/23	lun 27/03/23	9
3.4	Diagrama de arquitectura	3 días	mar 28/03/23	jue 30/03/23	10
3.5	Diagrama clases de dominio	3 días	vie 31/03/23	mar 04/04/23	11
3.6	Diagrama Base de Datos	2 días	mié 05/04/23	jue 06/04/23	12
3.7	Prueba de concepto - Angular	4 días	vie 07/04/23	mié 12/04/23	13
3.8	Entrega PEC2 - Requerimientos, análisis y diseño	0 días	vie 14/04/23	vie 14/04/23	6
4	**PEC3 - Construcción de la base de la aplicación**	**36 días**	**vie 14/04/23**	**lun 05/06/23**	
4.1	Sprint 0 - Walking Skeleton	12 días	vie 14/04/23	lun 01/05/23	15
4.2	Sprint 1	12 días	mar 02/05/23	mié 17/05/23	17
4.3	Sprint 2	12 días	jue 18/05/23	vie 02/06/23	18
4.4	Entrega PEC3 - Construcción de la base de la aplicación	0 días	lun 05/06/23	lun 05/06/23	15
5	**PEC4 - Memoria y presentación**	**15 días**	**lun 05/06/23**	**lun 26/06/23**	
5.1	Escribir memoria	10 días	lun 05/06/23	vie 16/06/23	20
5.2	Grabar presentación	5 días	lun 19/06/23	vie 23/06/23	22
5.3	Entrega PEC4 - Memoria y presentación	0 días	lun 26/06/23	lun 26/06/23	20
6	Tribunal de evaluación	6 días	lun 03/07/23	dom 09/07/23	

Ilustración 1 - Diagrama de Gantt

1.5. Breve sumario de productos obtenidos

Como resultado de este trabajo se han obtenido los siguientes productos:

- Esta memoria, que incluye entre otras cosas la planificación, el análisis y el diseño de la aplicación.
- Una presentación en video donde explica el desarrollo del proyecto y se demuestra el funcionamiento de la aplicación.
- El código fuente de la aplicación y los servicios web, que está disponible en dos repositorios públicos de GitLab (19).

1.6. Breve descripción de los otros capítulos de la memoria

En los próximos capítulos encontraremos lo siguiente:

- En el capítulo 2 describiremos el análisis del proyecto. Se ha incluido un mapa de historias de usuario, así como la descripción detallada de las historias incluidas y un prototipo de la aplicación.
- En el capítulo 3 describiremos el diseño y la arquitectura de la aplicación. Se descompone la aplicación en módulos y se detallan las principales decisiones tomadas.
- En el capítulo 4 describiremos las tecnologías usadas y los aspectos más relevantes de la implementación, así como cambios sobre el diseño inicial.
- Finalmente, en el capítulo 5 se incluirán las conclusiones del trabajo, incluyendo lecciones aprendidas, objetivos cumplidos y posibles desarrollos futuros.

Como anexos se incluyen también los manuales de instalación y de uso del proyecto.

2. Análisis

A continuación, se muestra una imagen del mapa de historias de usuario (16) del proyecto:

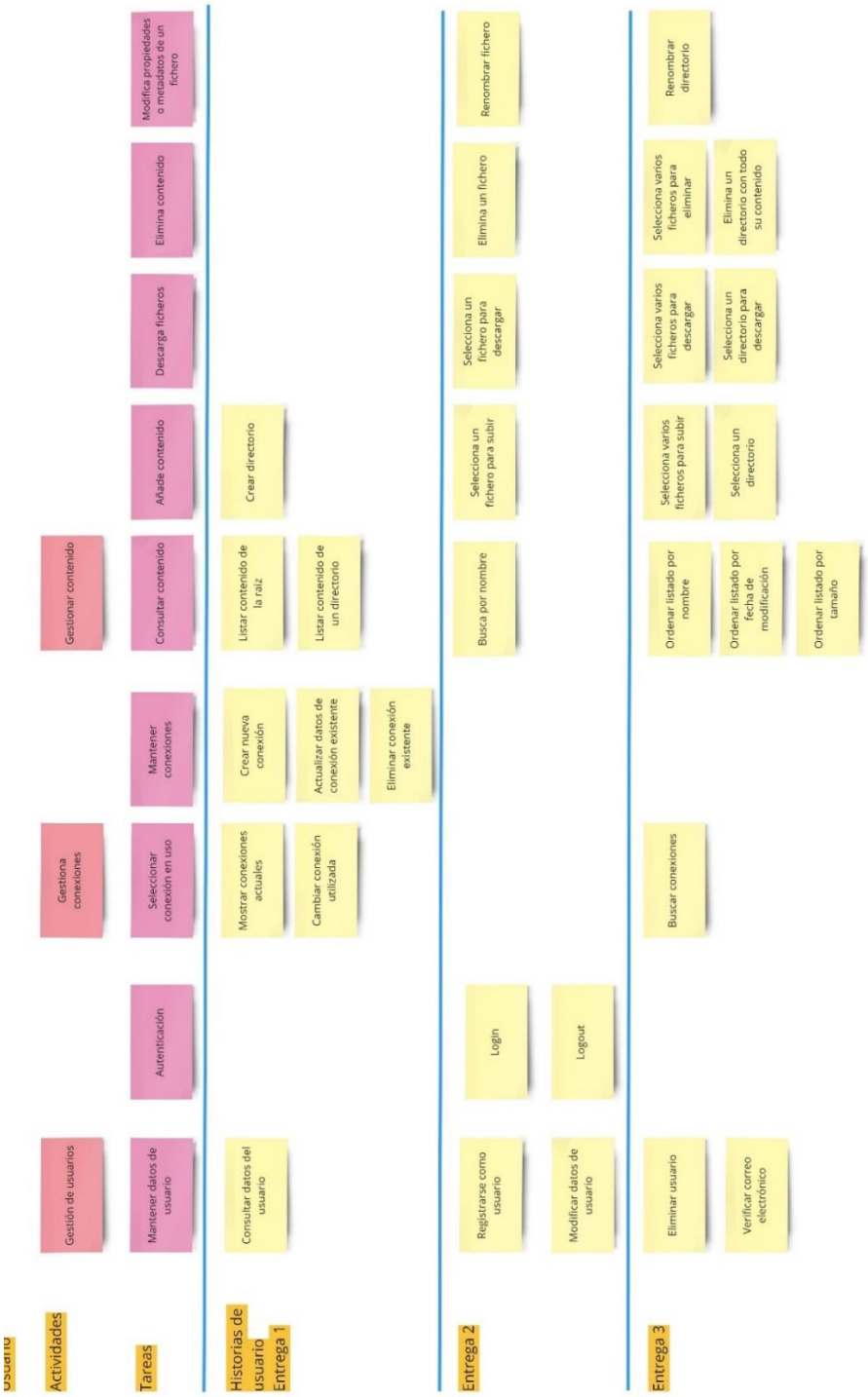


Ilustración 2 - Mapa de historias de usuario

2.1. Descripción de las historias de usuario

En los siguientes subapartados se describen una por una las historias de usuario presentadas en el mapa anterior. Por cada historia de usuario se incluye siempre una descripción y las pruebas de aceptación. En algunas de ellas se ha incluido además el *mockup* o bosquejo del interfaz de usuario.

Listar contenido de la raiz

Como usuario quiero poder consultar el contenido de un repositorio para saber cuál es su contenido

Pruebas de aceptación

Suponiendo que hay contenido en el repositorio, cuando el usuario selecciona la conexión al repositorio, o selecciona el directorio raiz del mismo, entonces el se buscan ficheros y directorios en la raiz de dicho repositorio y se muestran al usuario.

Suponiendo que el repositorio está vacio, cuando el usuario selecciona la conexión al repositorio o lel directorio raíz del mismo, entonces se le muestra una lista vacia y un mensaje indicando que el directorio está vacio o no hay contenido.

Mockup

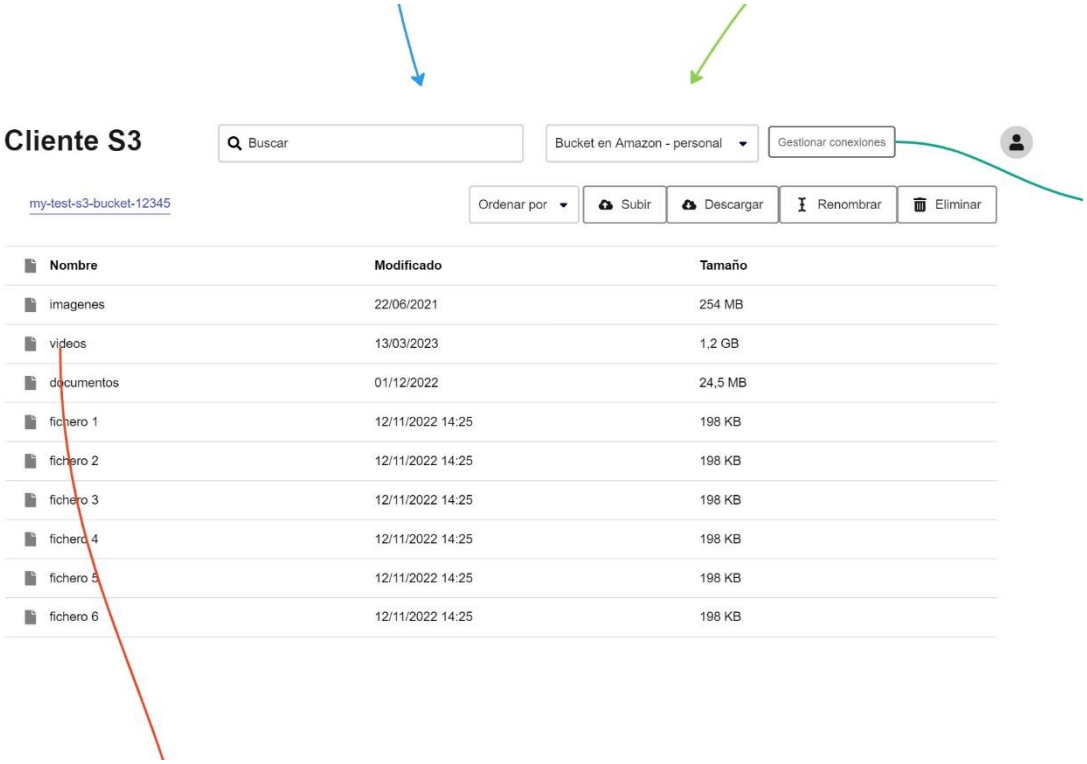


Ilustración 3 - Listado de directorio raíz

Listar contenido de un directorio

Como usuario quiero poder listar el contenido de cualquier directorio dentro de un repositorio para poder gestionar su contenido.

Pruebas de aceptación

Suponiendo que hay contenido en el directorio, cuando el usuario selecciona el directorio del cual quiere ver el contenido, entonces se buscan ficheros y directorios en dicho directorio y se muestran al usuario.

Suponiendo que el directorio está vacio, cuando el usuario selecciona el directorio, entonces se le muestra una lista vacia y un mensaje indicando que el directorio está vacio o no hay contenido.

Mockup

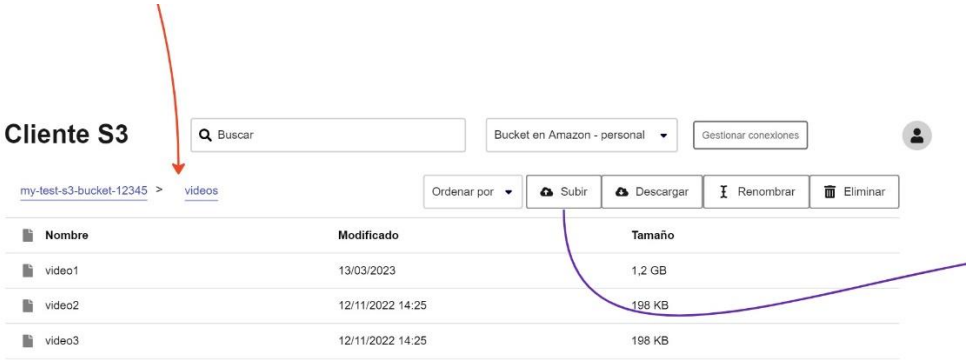


Ilustración 4 - Listado de subdirectorio

Busca fichero por nombre

Como usuario quiero poder buscar ficheros por nombre dentro de un repositorio para poder localizarlo y acceder a su contenido rápidamente sin necesidad de navegar por toda la estructura de directorios.

Pruebas de aceptación

Suponiendo que hay contenido en el repositorio, y que tenemos al menos dos ficheros que se llaman fichero1 y fichero2. Cuando el usuario indica que quiere buscar ficheros con el nombre "fichero", entonces se buscan los ficheros y directorios que contengan ese texto en el nombre y se devuelve una lista que contendrá al menos fichero1 y fichero2 (cualquier otro nombre incluido en la lista debe incluir el texto "fichero" en su nombre).

Suponiendo que hay contenido en el repositorio, y que tenemos al menos dos ficheros que se llaman fichero1 y fichero2. Cuando el usuario indica que quiere buscar ficheros con el nombre "directorio", entonces al buscar los ficheros o directorios que incluyan este texto se devolverá una lista vacia, o si se encuentra algo, no estará fichero1 o fichero2 en la lista.

Ordenar listado por nombre

Como usuario quiero poder ordenar el contenido de un directorio en el repositorio por nombre para poder encontrar o identificar más fácilmente su contenido

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio con contenido, es decir tiene ficheros y/o directorios, por ejemplo fichero1 y fichero2. Cuando el usuario indica que quiere ordenar la lista por nombre, entonces se mostrará el mismo contenido mostrado anteriormente pero ordenado por nombre, es decir en el ejemplo anterior se mostraría primero fichero1 y luego fichero2.

Ordenar listado por fecha de modificación

Como usuario quiero poder ordenar el contenido de un directorio en el repositorio por la fecha de modificación para poder localizar e identficar más fácilmente el contenido.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio con contenido, es decir tiene ficheros y/o directorios, por ejemplo fichero1 y fichero2, habiendo sido modificado fichero1 antes que fichero2. Cuando el usuario indica que quiere ordenar la lista por fecha de modificación, entonces se mostrará el mismo contenido mostrado anteriormente pero ordenado por fecha de modificación, es decir en el ejemplo anterior se mostraría primero fichero1 y luego fichero2.

Ordenar listado por tamaño

Como usuario quiero poder ordenar el contenido de un directorio en el repositorio por el tamaño del fichero o de su contenido para poderidentificar más fácilmente el contenido.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio con contenido, es decir tiene ficheros y/o directorios, por ejemplo fichero1 y fichero2, siendo fichero1 menor en tamaño que fichero2. Cuando el usuario indica que quiere ordenar la lista por tamaño, entonces se mostrará el mismo contenido mostrado anteriormente pero ordenado por tamaño, es decir en el ejemplo anterior se mostraría primero fichero1 y luego fichero2.

Crear directorio

Como usuario quiero poder crear un directorio en el repositorio para poder organizar mejor el contenido.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio en el actual repositorio en el que está conectado, y que no existe aún un directorio con el nombre que él quiere utlizar, por ejemplo directorio1. Cuando el usuario indica que quiere crear el directorio llamado "directorio1", se comprueba que dicho directorio no existe y se crea. Además el listado de contenido del directorio actual incluiye el nuevo directorio creado llamado "directorio1"

Suponiendo que el usuario ha seleccionado un directorio en el actual repositorio en el que está conectado, y que ya existe un directorio con el nombre del directorio que quería crear, por ejemplo "directorio1". Cuando el usuario indica que quire crear el "directorio1" se le indica que dicho directorio ya existe, y se le pide que elija un nombre distinto o que cancele la acción.

Selecciona un fichero para subir

Como usuario quiero poder subir un fichero al repositorio para almacenarlo

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio y que no contiene ningún fichero con el mismo nombre del fichero que se pretende subir. Cuando se indica que se quiere subir un fichero y se selecciona del equipo local un fichero, entonces el fichero se sube al repositorio al directorio seleccionado actualmente y se muestra en la lista de elementos contenidos dentro del directorio.

Suponiendo que ya existe un fichero con el mismo nombre que queremos subir. Cuando el usuario selecciona el fichero para subir, entonces el sistema le indica que ya existe un fichero con dicho nombre y que si quiere actualizar su contenido. Entonces el usuario indica que quiere actualizar y el fichero se sube al repositorio reemplazando el contenido que había anteriormente. Es decir, si había un fichero llamado fichero1 de un 1MB y subimos un fichero llamado fichero1 de 2MB, entonces al final en el repositorio tendremos el fichero llamado "fichero1" de 2MB.

En el mismo supuesto anterior, cuando el usuario no quiere actualizar el fichero existente en el repositorio, entonces no se realiza ningún cambio. En el ejemplo anterior, el directorio contendría aún el fichero llamado "fichero1" de 1MB.

Riesgos y suposiciones

El SDK de Amazon tiene funciones para subir ficheros desde el sistema de ficheros local, pero no estoy seguro si será posible subirlo al vuelo desde el API REST.

Estrategias de mitigación

Estudiar el API y realizar una prueba de concepto

Mockup

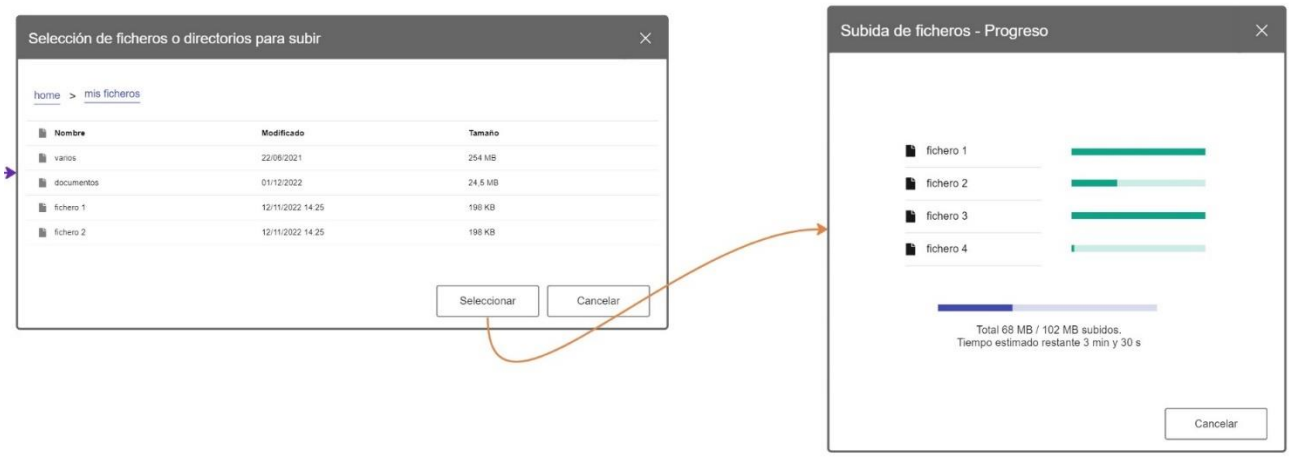


Ilustración 5 - Subida de ficheros

Selecciona varios ficheros para subir

Como usuario quiero poder seleccionar y subir varios ficheros en un lote a la vez de manera desatendida para poder ahorrar tiempo y no tener que ir uno a uno.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que no contiene ficheros con los mismos nombres que los que pretende subir. Cuando el usuario selecciona varios ficheros para subir, los ficheros se suben al servidor y al finalizar se muestran en el directorio.

Selecciona un directorio para subir

Como usuario quiero poder seleccionar y subir un directorio de mi equipo local al repositorio, incluyendo todo su contenido (ficheros y subdirectorios recursivamente), para poder ahorrar tiempo y no tener que subir uno a uno.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que no contiene ningún otro directorio con el mismo nombre que pretendemos subir, y dicho directorio que queremos subir contiene a su vez ficheros. Cuando el usuario indica que quiere subir el directorio, entonces el directorio se crea en el repositorio y se le añaden todos los ficheros que incluía. Al finalizar se puede comprobar que en el nuevo directorio está incluido el mismo contenido que teníamos en nuestro equipo local.

Selecciona un fichero para descargar

Como usuario quiero poder seleccionar un fichero del repositorio y descargarlo a mi equipo local para poder abrirlo o manipularlo.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que contiene al menos un fichero. Cuando el usuario indica que quiere descargar dicho fichero el fichero

se descarga y el usuario puede abrir el fichero en local comprobando que el contenido es correcto.

Selecciona varios ficheros para descargar

Como usuario quiero poder seleccionar varios ficheros y descargarlos al equipo local en un lote de manera desatendida para poder ahorrar tiempo al no tener que está haciéndolo uno a uno.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que contiene varios ficheros. Cuando el usuario selecciona varios ficheros e indica que quiere descargarlos entonces el sistema los descarga comprimidos en un fichero zip con el nombre del directorio donde están ubicados (o el nombre del repositorio si están en directorio raiz).

Selecciona un directorio para descargar

Como usuario quiero poder seleccionar y descargar un directorio, incluyendo todo su contenido, a mi equipo local, para poder consultarlo, manipularlo o modificarlo si es necesario.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio con contenido, es decir, al menos dos o tres ficheros. Cuando el usuario indica que quiere descargar el directorio se descarga un fichero zip con el nombre del directorio que contiene lo que contenía el directorio en el repositorio.

Elimina un fichero

Como usuario quiero poder seleccionar un fichero en el repositorio y eliminarlo de allí si no quiero conservarlo más en el repositorio.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio con contenido. Cuando el usuario indica que quiere eliminar un fichero se le pide que confirme y después se elimina el fichero. Al mostrar el contenido del directorio el fichero eliminado ya no aparece.

Selecciona varios ficheros para eliminar

Como usuario quiero poder seleccionar varios ficheros y eliminarlos del repositorio a la vez para poder ahorrar tiempo.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que contiene varios ficheros. Cuando el usuario indica que quiere eliminar varios ficheros seleccionados y lo confirma, entonces los ficheros son eliminados del repositorio. Al mostrar el contenido del directorio los ficheros eliminados no tienen que mostrarse.

Elimina un directorio con todo su contenido

Como usuario quiero poder eliminar un directorio del repositorio, incluyendo todo su contenido (ficheros y otros directorios), si ya no lo necesito.

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que contiene al menos un directorio con contenido (otros ficheros). Cuando el usuario indica que quiere eliminar dicho directorio y confirma que eliminirá también su contenido, entonces se elimina el directorio y todos los ficheros y directorios incluidos dentro. Al mostrar de nuevo el contenido el directorio eliminado no aparecerá en la lista.

Renombrar fichero

Como usuario quiero poder modificar el nombre de un fichero almacenado en el repositorio para poder eliminar ambiguedades o simplemente para organizar mejor el contenido del mismo

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que contiene al menos un fichero y que no existe otro fichero con el mismo nombre que queremos utilizar. Cuando el usuario indica que quiere renombrar el fichero he indica el nombre, se comprueba que dicho nombre no existe y se procede a renombrar el fichero en el repositorio. Al mostrar el contenido del directorio el fichero aparecerá con el nuevo nombre.

Renombrar directorio

Como usuario quiero poder modificar el nombre de un directorio almacenado en el repositorio para poder organizar mejor el contenido del mismo

Pruebas de aceptación

Suponiendo que el usuario ha seleccionado un directorio que contiene al menos un directorio que a su vez contiene otros ficheros, y que no existe otro directorio con el mismo nombre que queremos utilizar. Cuando el usuario indica que quiere renombrar el directorio he indica el nuevo nombre deseado, se comprueba que dicho nombre no existe y se procede a renombrar el directorio en el repositorio, así como el path en todos los ficheros incluidos dentro de dicho directorio. Al mostrar el contenido del directorio el directorio aparecerá con el nuevo nombre.

Mostrar conexiones actuales

Como usuario quiero poder consultar las conexiones con repositorios de las que dispongo para poder elegir la que quiero usar o saber si tengo que eliminar o modificar alguna de ellas o añadir una nueva.

Pruebas de aceptación

Suponiendo que no existe ninguna conexión creada entonces se mostrará una lista vacía y aparecerá un mensaje indicando que no hay conexiones.

Suponiendo que existe al menos una conexión, cuando el usuario accede a la lista de conexiones puede ver una lista con el nombre y la descripción de las conexiones existentes.

Mockup

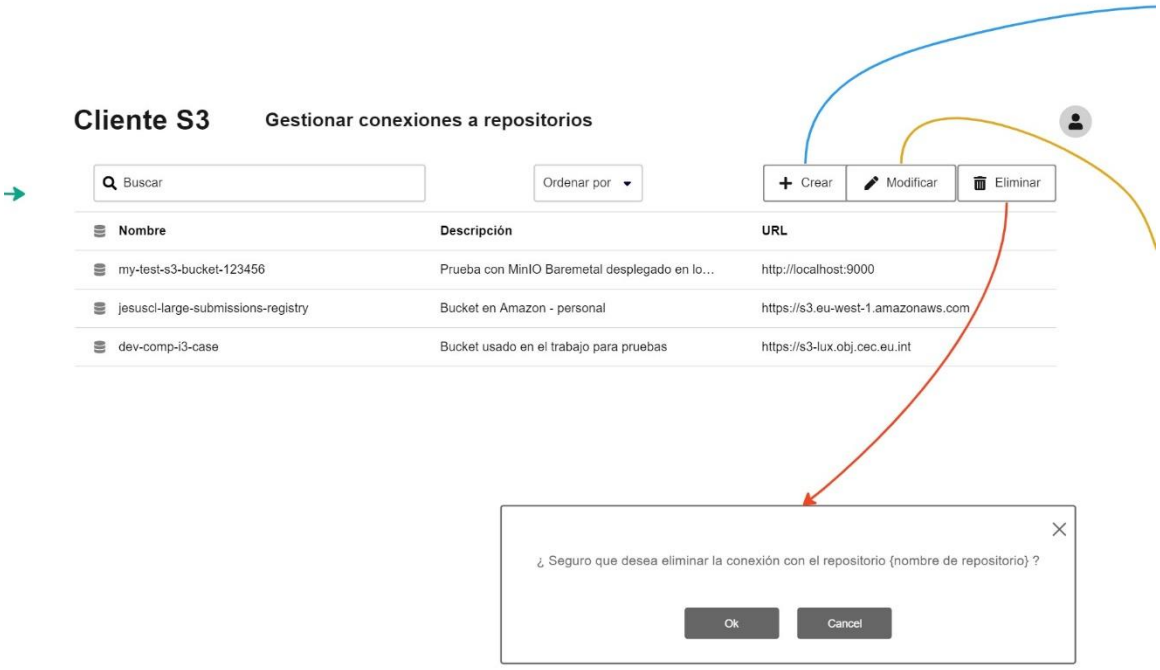


Ilustración 6 - Listado de conexiones

Cambiar conexión utilizada

Como usuario quiero poder cambiar la conexión que estoy utilizando para poder gestionar el contenido de otro repositorio

Pruebas de aceptación

Suponiendo que tenemos al menos una conexión, y los datos de conexión son correctos, cuando el usuario la selecciona para utilizarla, entonces se conecta al repositorio seleccionado y se muestra el contenido de su directorio raíz.

Suponiendo que tenemos una o varias conexiones a repositorios, y el usuario selecciona una conexión con datos erroneos, entonces se le muestra un mensaje con el error obtenido al intentar conectarse y se pregunta al usuario si desea editar los datos de conexión.

Riesgos y suposiciones

Se asume que es posible proveer los datos de conexión dinámicamente mediante la implementación de un proveedor de conexiones.

Estrategias de mitigación

Implementar una prueba de concepto que conecte primero con un repositorio y luego conecte con otro.

Buscar conexiones

Como usuario quiero poder buscar conexiones a repositorios si tengo muchas.

Pruebas de aceptación

Suponiendo que tenemos una o varias conexiones a repositorios. Cuando el usuario busca una conexión por texto se le muestra en una lista las conexiones que incluyen dicho texto en el nombre o en la descripción de la conexión.

Crear nueva conexión

Como usuario quiero poder crear una nueva conexión a un repositorio para poder almacenar nuevo contenido

Pruebas de aceptación

Cuando el usuario introduce los datos de nombre, descripción, URL del endpoint, ID y contraseña de la cuenta, entonces los datos se guardan en el sistema y al mostrarse la lista de conexiones se puede ver el nombre y la descripción de la nueva conexión introducida.

Mockup

Ilustración 7 - Nueva conexión

Actualizar datos de conexión existente

Como usuario quiero poder actualizar los datos de una conexión existente para poder modificar cualquiera de sus parámetros si fuese necesario.

Pruebas de aceptación

Suponiendo que tenemos al menos una conexión a un repositorio. Cuando el usuario modifica los datos de la conexión, al consultar de nuevo los datos aparecen los datos cambiados.

Eliminar conexión existente

Como usuario quiero poder eliminar una conexión a un repositorio si ya no voy a ustilizarla más.

Pruebas de aceptación

Suponiendo que tenemos al menos una conexión a un repositorio. Cuando el usuario solicita eliminar una conexión y confirma que realmente quiere eliminarla, entonces la conexión se elimina y ya no aparece en la lista de la conexiones disponibles.

Registrarse como usuario

Como usuario quiero poder registrarme en la aplicación para poder acceder a ella y poder recuperar los datos asociados a mi cuenta, tales como las conexiones a repositorios S3 que me defina.

Pruebas de aceptación

Suponiendo que el usuario no está registrado, cuando el usuario solicita registrarse e introduce los siguientes datos: nombre completo, dirección de correo electrónico y contraseña que desea utilizar. Entonces dichos datos se almacenan y posteriormente podrán consultarse.

Suponiendo que anteriormente otro usuario hubiese utilizado la misma cuenta de correo electrónio, entonces cuando el usuario introduce los datos el sistema le devuelve un error indicando que ya se ha registrado un usuario anteriormente utilizando dicha cuenta de correo electrónico.

Mockup

Ilustración 8 - Registro de usuario

Consultar datos de usuario

Como usuario registrado quiero poder consultar los datos asociado a mi cuenta para poder comprobar que son correctos.

Pruebas de aceptación

Suponiendo que el usuario está registrado y ya se ha autenticado en el sistema, cuando el usuario solicita ver los datos de usuario se le muestra su nombre completo y su dirección de correo electrónico.

Modificar datos de usuario

Como usuario registrado quiero poder modificar los datos asociados a mi cuenta para poder corregirlos si son incorectos

Pruebas de aceptación

Suponiendo que el usuario está registrado y ya se ha autenticado en el sistema, cuando el usuario solicita modificar el nombre o su dirección de correo electrónico entonces los datos se actualizan en la base de datos y posteriormente pueden ser consultados.

Mockup

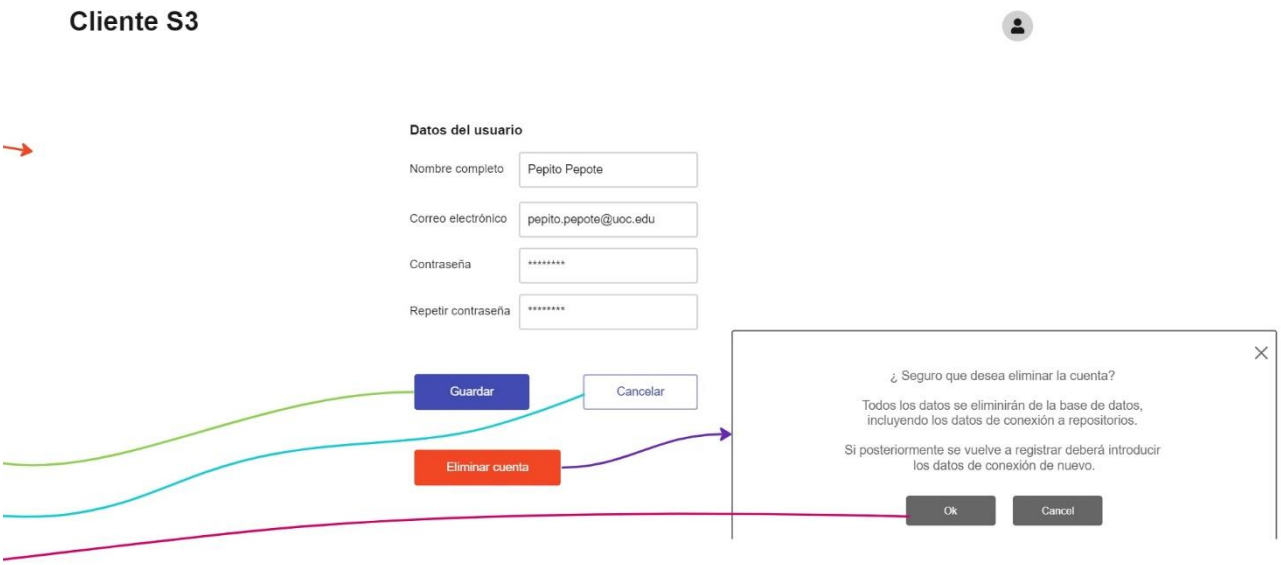


Ilustración 9 - Modificar datos de usuario

Eliminar usuario

Como usuario registrado quiero poder eliminar mi usuario y todos los datos asociados al mismo, tales como conexiones a repostorios S3, para que no quede rastro de mis datos en la aplicación.

Pruebas de aceptación

Suponiendo que el usuario está registrado y ya se ha autenticado en el sistema, cuando solicita eliminar su cuenta de usuario se le pide confirmación y se le advierte de que esta acción es irreversible. Entonces el usuario lo confirma y se eliminan de la base de datos todas las conexiones asociadas a dicho usuario así como los datos del propio usuario. Además se hace logout del usuario y si intenta volver a acceder no podrá porque su usuario ya no existe.

Verificar correo electrónico

El correo del usuario debe ser verificado para poder comprobar que es el suyo y que no ha introducido uno inexistente o de otra persona por error o de manera malintencionada.

Pruebas de aceptación

Suponiendo que el usuario se ha registrado entonces el sistema automáticamente envía un correo electrónico al usuario con una información o un enlace para que el usuario la introduzca en el sistema o se indique al sistema que efectivamente esa es su cuenta de correo electrónico. Cuando se indica al sistema la información enviada por correo electrónico, entonces se marca dicha cuenta de correo electrónico como verificada.

Riesgos y suposiciones

Hay que interactuar con un servidor SMTP y realizar pruebas con correo electrónico puede ser lento y tedioso, además de correr riesgo de enviar SPAM.

Estrategia de mitigación

Ver si se puede utilizar alguna herramienta especial para simular el envío de correos electrónicos.

Login

Como usuario quiero poder hacer login en el sistema para poder acceder a mis datos y poder utilizar la aplicación

Pruebas de aceptación

Suponiendo que el usuario no se ha autenticado aún en el sistema, cuando introduce su usuario y contraseña correctos el sistema lo comprueba y le permite acceder a la aplicación.

Cuando introduce los datos incorrectos el sistema se lo indica y no le permite acceder a los datos.

Riesgos y suposiciones

Para que la autenticación sea segura, la implementación puede ser muy compleja y costosa (en tiempo). Para el TFG habría que tener cuidado de no dedicar demasiado esfuerzo en las tareas de seguridad y gestión de usuario, ya que aunque son fundamentales para un producto final, no es el objetivo principal de este proyecto.

Estrategia de mitigación

Ver si podemos autenticar con cuentas de terceros como Google. Realizar una prueba de concepto.

Mockup

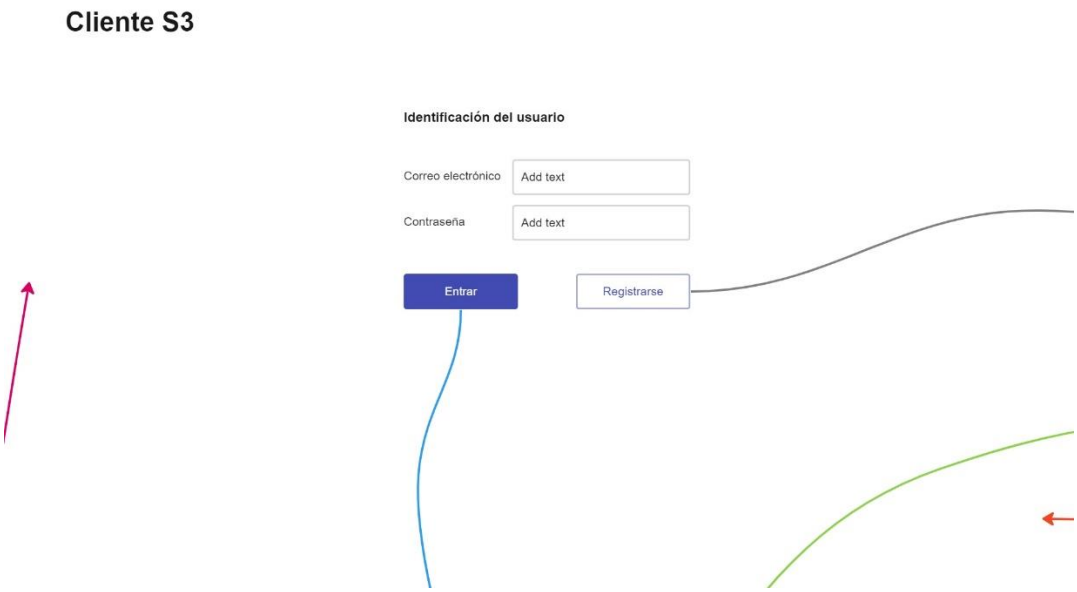


Ilustración 10 - Identificación del usuario (login)

Logout

Como usuario quiero poder hacer logout del sistema para por ejemplo poder identificarme con otra cuenta de usuario o simplemente finalizar la sesión actual.

Pruebas de aceptación

Suponiendo que el usuario está autenticado cuando solicita que quiere hacer logout, se finaliza la sesión de dicho usuario en el sistema y ya no podrá acceder a sus datos hasta que no vuelva a hacer login.

2.2. Prototipo

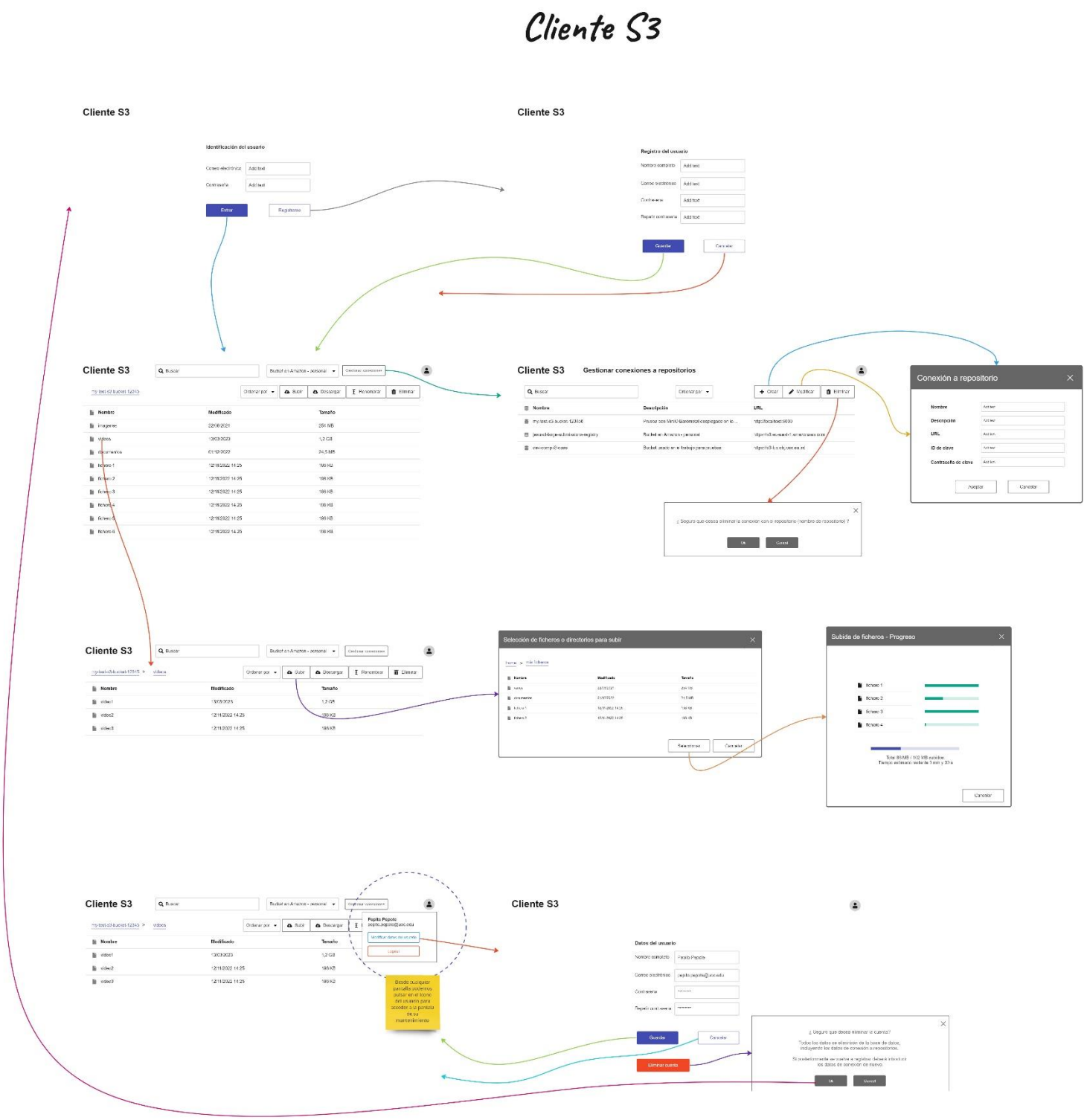


Ilustración 11 - Prototipo completo

3. Arquitectura

A continuación, se presentará la arquitectura del sistema. Para la elaboración de los diagramas se ha seguido el modelo C4 propuesto por Simon Brown (20). Este modelo consiste en varios niveles de abstracción, en concreto cuatro, divididos en contexto, contenedores, componentes y código. Normalmente los dos primeros niveles son necesarios, y el tercero y el cuarto se pueden utilizar ocasionalmente para profundizar en ciertos componentes. Para este trabajo se han completado los diagramas de contexto, de contenedores y el de componentes para uno de ellos.

3.1. Requerimientos de calidad

Los objetivos principales de calidad de este sistema serían:

- Operatividad o facilidad de uso: ya existe una herramienta muy completa que actúa como cliente de repositorios S3, y es Amazon AWS CLI. El problema es que se trata de un interfaz por la línea de comandos, y por tanto su uso es más complicado si no se tiene cierta familiaridad con este tipo de herramientas. Nuestra herramienta tiene que ser comprensible o fácil de aprender, así como sencilla de utilizar y atractiva para los usuarios.
- La capacidad de mantenimiento: además de Amazon AWS CLI, existen otras herramientas que permiten la gestión de repositorios S3 de manera visual, tales como Filezilla Pro o WinSCP. Sin embargo, estas son herramientas que se instalan en la máquina del usuario y normalmente requieren un mantenimiento no sólo para su instalación inicial sino para posteriores actualizaciones.
- Idoneidad funcional: El sistema tiene que proveer funcionalidades básicas que son implícitas de este tipo de herramientas.
- Seguridad: El sistema tiene que proteger los datos de acceso a los repositorios, ya que si estos son expuestos, se podría acceder al contenido que los usuarios tienen en dichos repositorios.

3.2. Restricciones

La principal restricción con la que cuenta este proyecto es el tiempo limitado del que se dispone, ya que hay que cumplir con los plazos de entrega de la universidad. Es por ello que posiblemente no se implementen todas las funcionalidades deseadas, sino únicamente las que se consideren imprescindibles.

Otra restricción es el presupuesto disponible. Idealmente el proyecto debería poder desplegarse en alguna plataforma en la nube, tal como Amazon AWS (3) o Microsoft Azure (21). Sin embargo, hay que tener cuidado con el coste de estas plataformas, ya que en mi caso particular agoté el crédito gratuito de Azure utilizando la cuenta de la UOC y recuerdo que tuve algunos problemas en otras asignaturas con la cuenta de Amazon AWS.

3.3. Contexto

A continuación, se muestra en un diagrama de contexto cuáles son los límites de nuestro sistema. Es decir, incluimos aquí usuarios y otros sistemas externos con los que se interactuará.

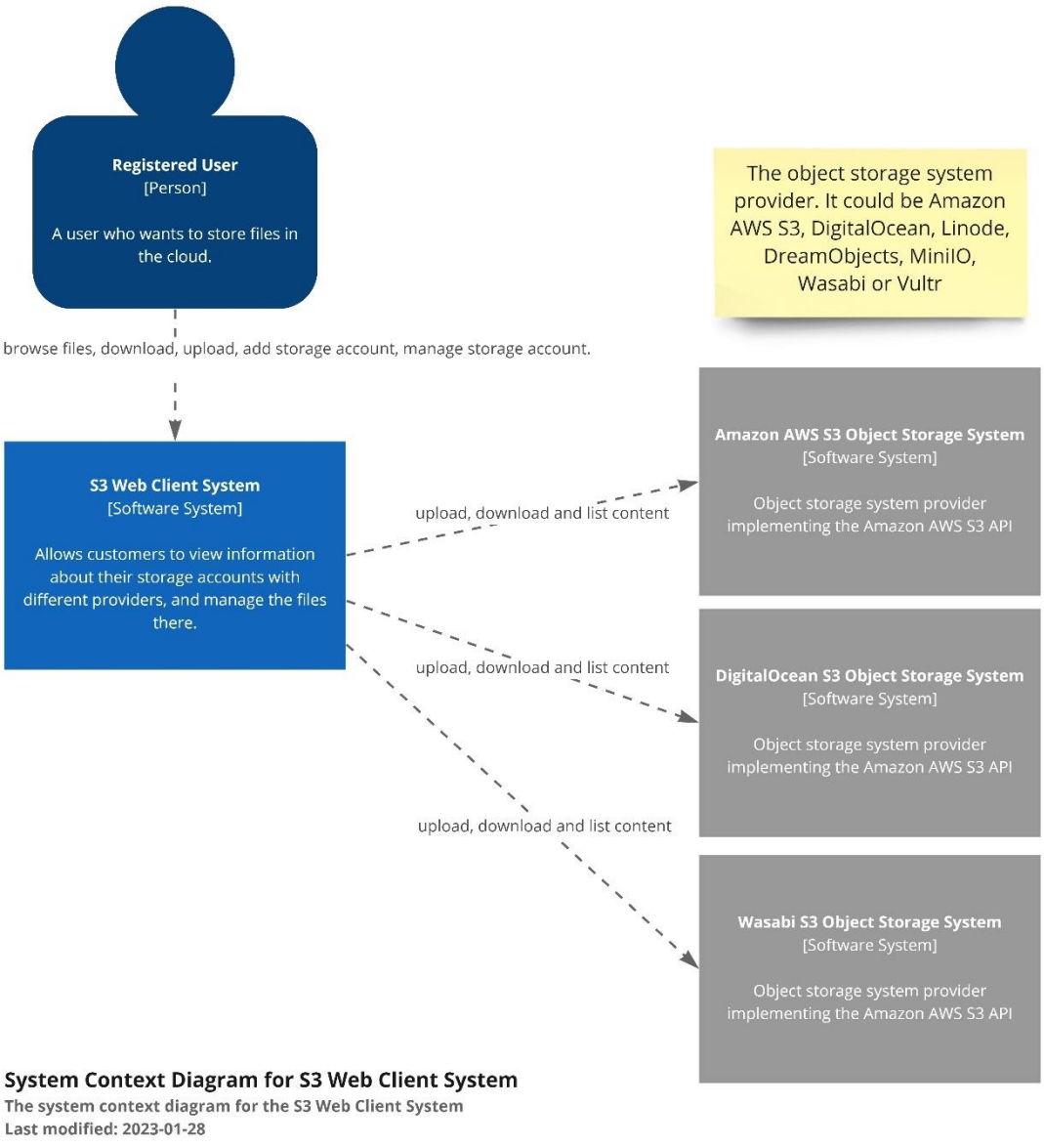


Ilustración 12 - Arquitectura, diagrama de contexto (C4)

3.4. Decisiones

A continuación, se describen las principales decisiones tomadas:

1. Para cumplir los requerimientos de facilidad de uso se ha decidido desarrollar una interfaz gráfica de usuario.
2. Para cumplir el requerimiento de facilidad de mantenimiento se va a desarrollar una aplicación web.
3. Se ha decidido seguir una arquitectura de microservicios, aunque para el desarrollo de este proyecto intentaremos limitar el número de microservicios al mínimo posible para no aumentar demasiado la complejidad.
4. En el *backend* se va a utilizar Java 17 (22) y el framework Spring Boot 3 (23).
5. En el *frontend* se va a desarrollar una aplicación SPA (Single Page Application) mediante el *framework* Angular (la versión 15 es la más reciente en este momento). Para los componentes gráficos se utilizarán los de Material Design.
6. Para cumplir con el requerimiento de la seguridad intentaremos delegar el servicio de autenticación en terceros, tales como el servicio de autenticación de Google. De este modo esperamos poder desarrollar una primera versión funcional sin necesidad de implementar este apartado (para más información, ver la tercera prueba de concepto incluida en el Apéndice en este mismo documento).

3.5. Descomposición del sistema en bloques o módulos

A continuación, se muestra en más detalle cómo se ha organizado el sistema en distintos módulos.

Diagrama de contenedores

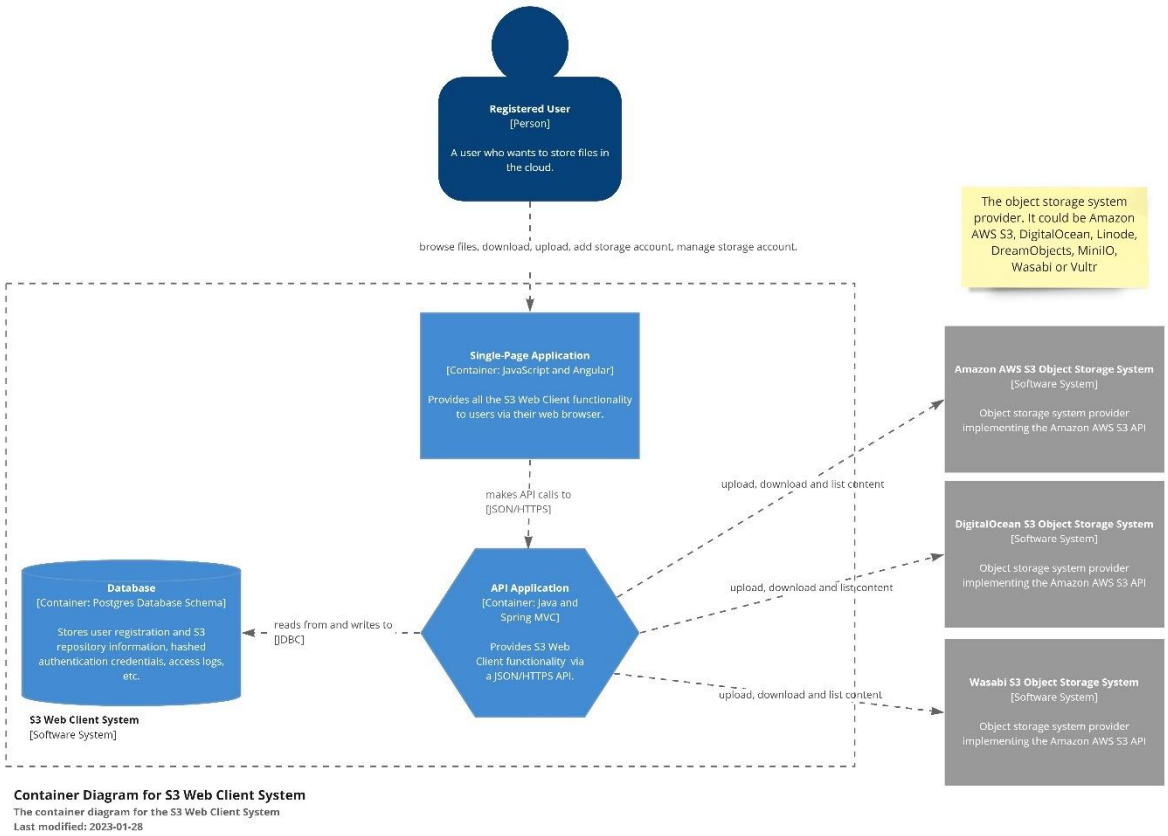


Ilustración 13 - Arquitectura, diagrama de contenedores (C4)

En este diagrama podemos observar que nuestro sistema consistirá principalmente de una aplicación web que interactuará con un microservicio que constará básicamente de un API REST. Este API persistirá datos en una base de datos relacional PostgreSQL (24) y a su vez interactuará con los repositorios S3 (sistemas externos).

Diagrama de componentes – S3 API

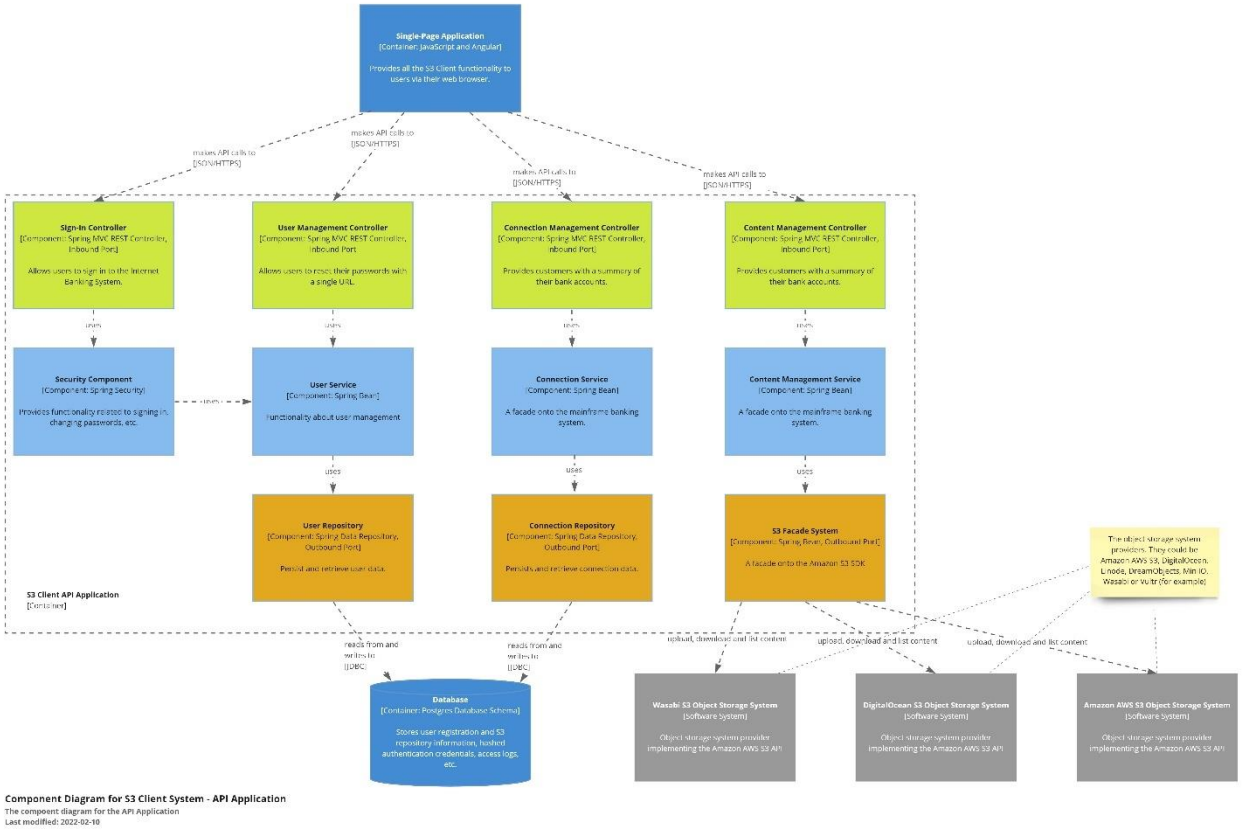


Ilustración 14 - Arquitectura, diagrama de componentes del contenedor S3 API (C4)

Este diagrama describa la estructura interna del *backend*. Internamente se utiliza una arquitectura hexagonal, la cual aísla muy bien las interacciones con los sistemas externos. La idea principal es que las clases de dominio interactúan con interfaces y únicamente las implementaciones de dichos interfaces conocen de estos sistemas externos. Entre otras ventajas, esto facilita mucho las pruebas de los servicios ya que es muy fácil sustituir los servicios externos por *Mocks*.

3.6. Diagrama de clases de dominio

En el siguiente diagrama de clases UML, se pueden observar las principales clases del dominio

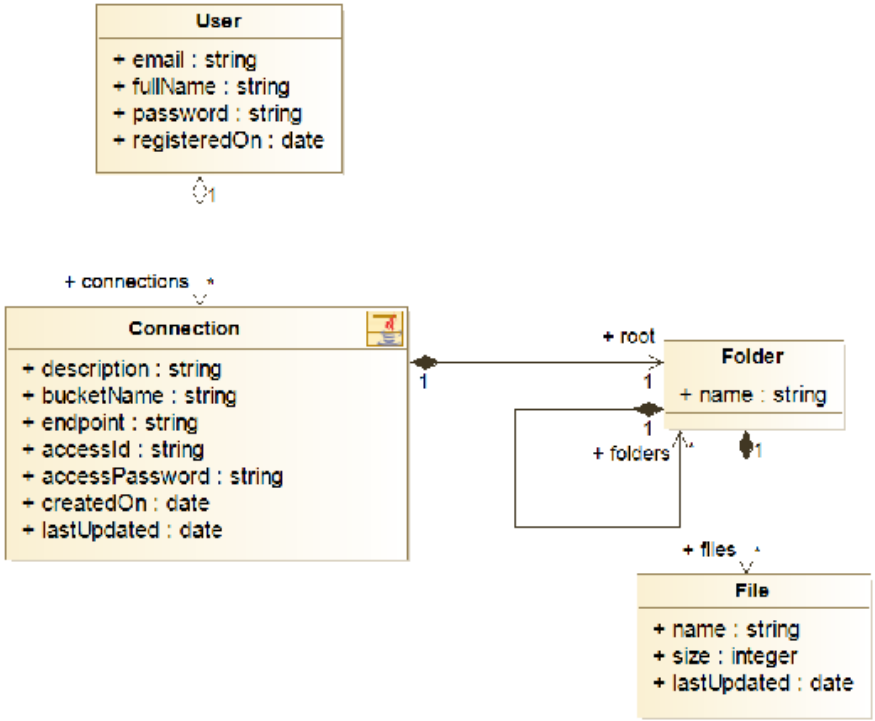


Ilustración 15 - Diagrama de clases del dominio

3.7. Diagrama de base de datos

El modelo de base de datos a priori es muy sencillo, ya que únicamente se van a persistir los datos relacionados con los usuarios y la configuración de acceso a los repositorios de S3. Toda la información relacionada con ficheros y directorios se almacenará en el propio repositorio S3.

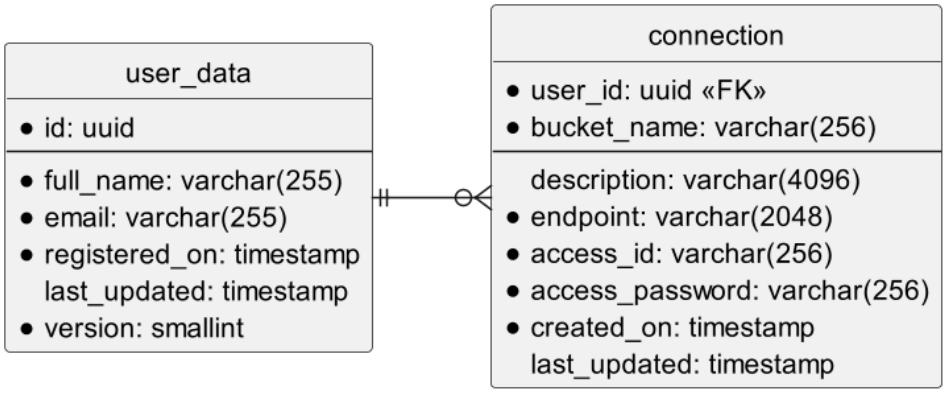


Ilustración 16 - Diagrama de base de datos

4. Implementación

En este apartado vamos a describir aspectos claves de la implementación, tales como tecnologías utilizadas durante el desarrollo.

4.1. Entorno de desarrollo

Para el desarrollo del proyecto se han utilizado principalmente los siguientes entornos de desarrollo integrado (IDEs) y herramientas:

- IntelliJ IDEA Ultimate (25)– Para el desarrollo del *backend* en Java. Este IDE también incluye cliente de base de datos.

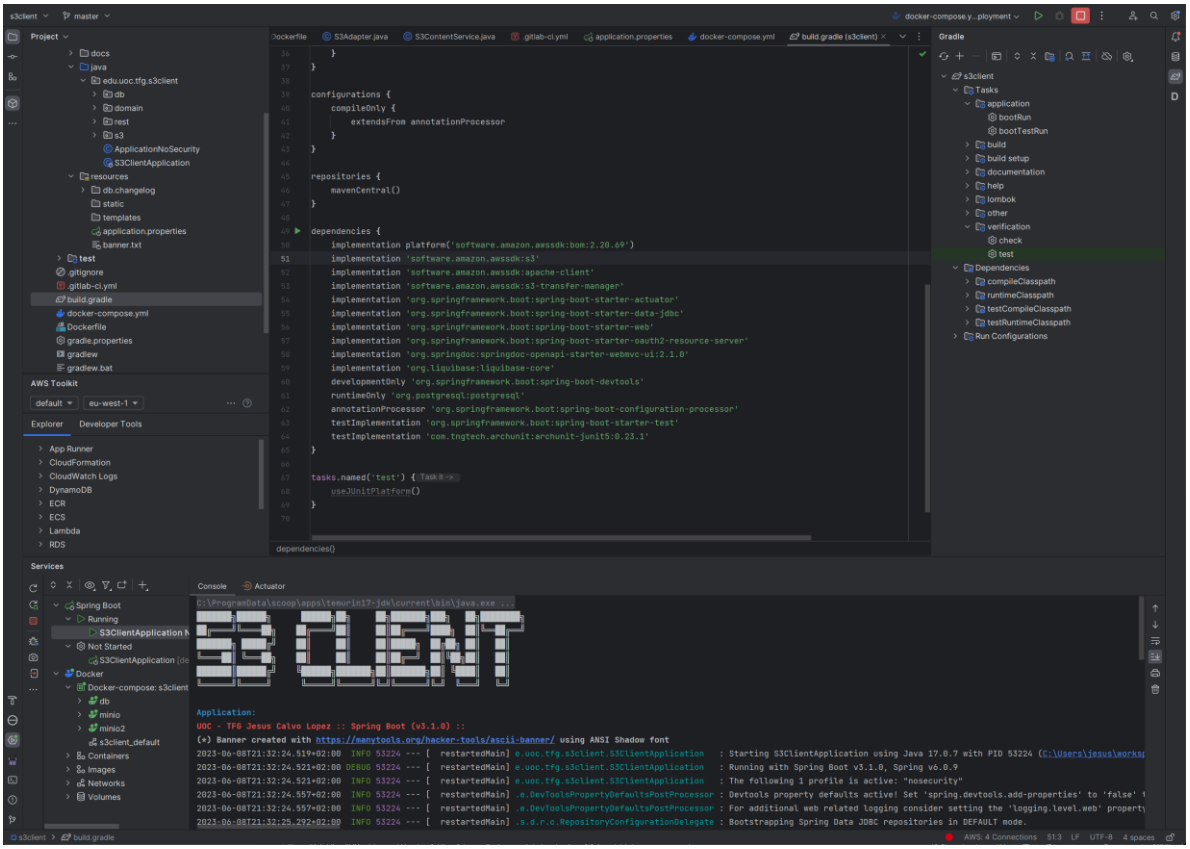


Ilustración 17- IntelliJ Idea

- Visual Studio Code (26) – Para el desarrollo del *frontend*

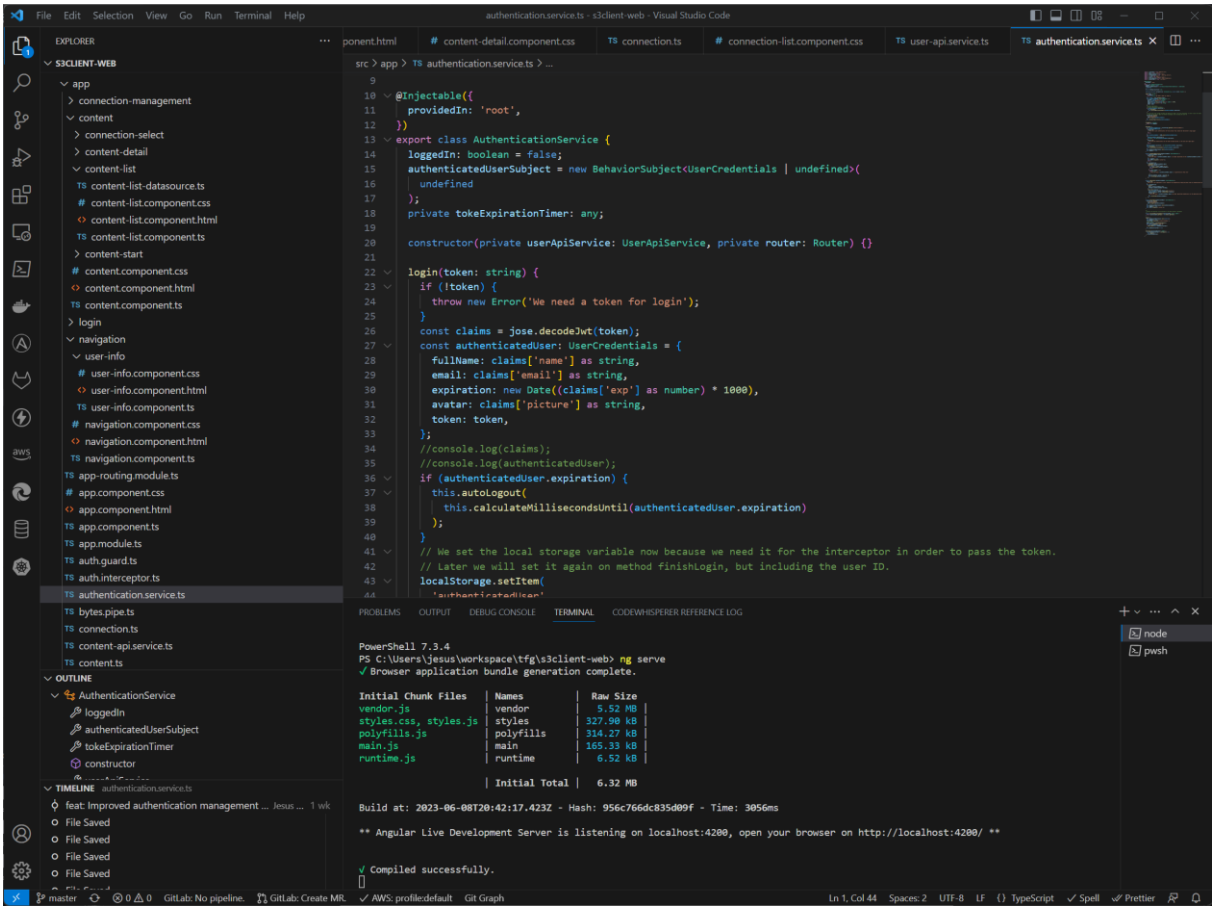


Ilustración 18 - Visual Studio Code

- Postman (27) – Para las pruebas del *backend*

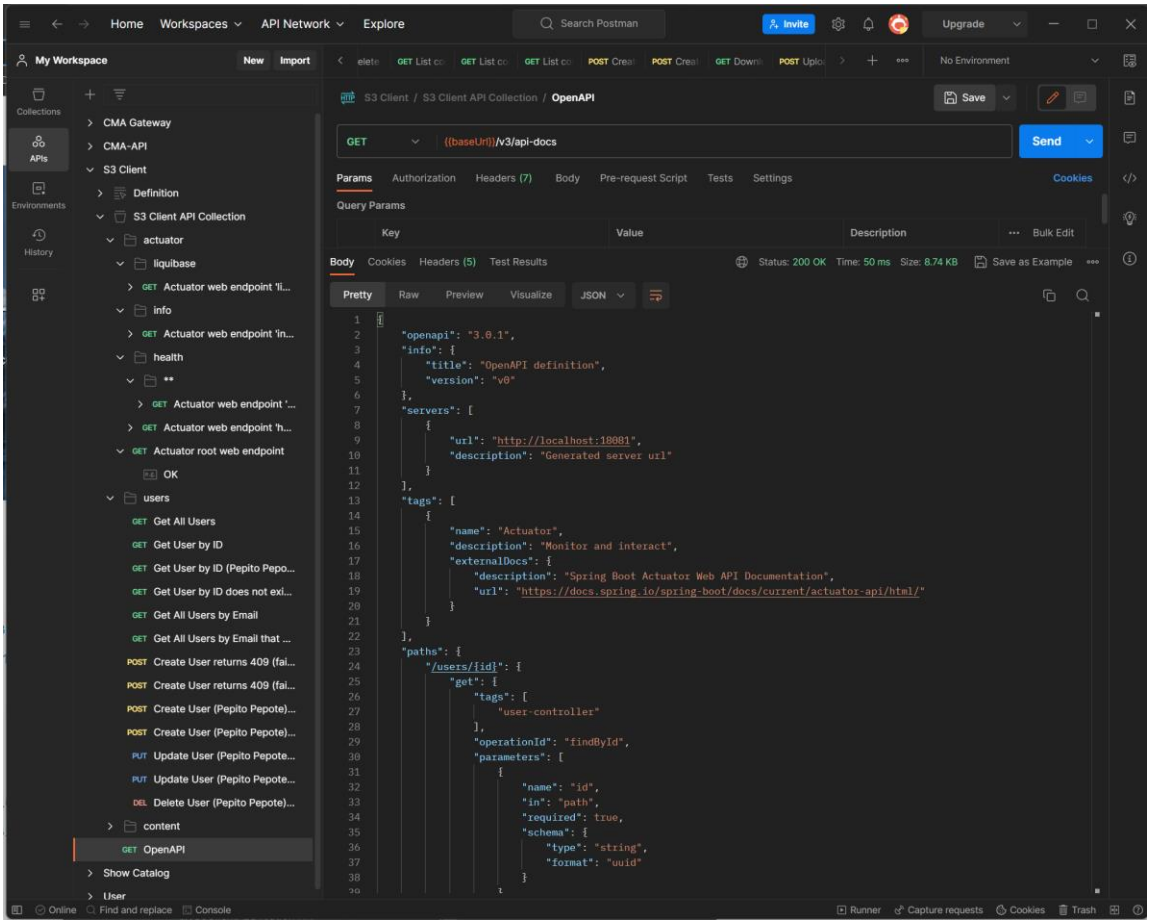


Ilustración 19 - Postman

- Docker Compose (28) – Para arrancar las instancias de Postgresql (24) y MinIO (9) que se han usado para el desarrollo en local.

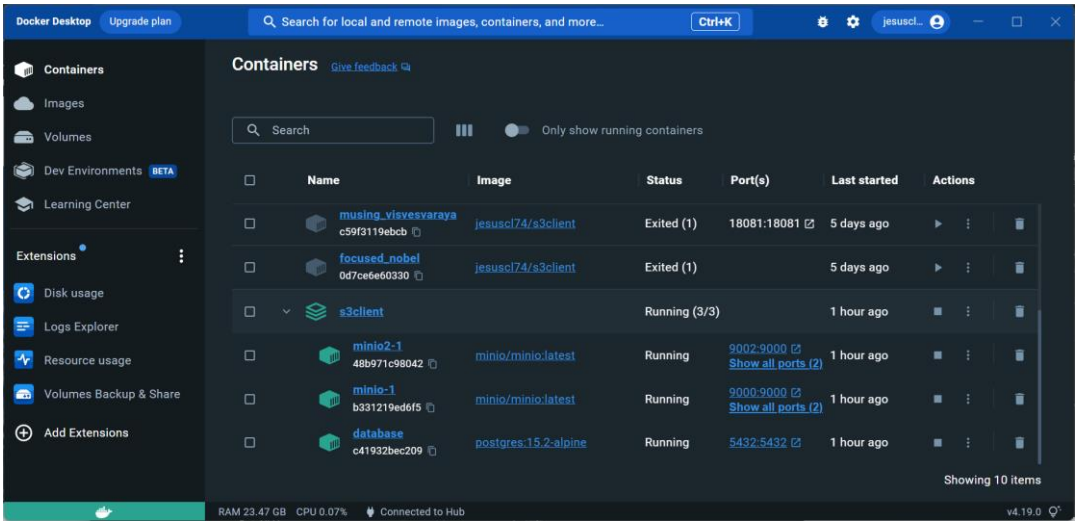


Ilustración 20 - Docker Desktop

- Java JDK versión 17 (última versión LTS) (22)
- NodeJS 18 (última versión LTS) (29)

- Gradle (30) para la construcción del proyecto del *backend*
- Git (31) para el control de versiones.

4.2. Código fuente

El repositorio de código Git (31) está en GitLab (19). Se puede acceder a los proyectos mediante la siguiente URL: https://gitlab.com/jesuscl74-tfg

Este es un grupo de proyectos que incluye dos repositorios, uno para el *backend* y otro para el *frontend*.

Backend

El proyecto del *backend* se llama **s3client** y su código fuente puede obtenerse aquí:
https://gitlab.com/jesuscl74-tfg/s3client

Este es un proyecto Java. El framework principal es Spring Boot (23) version 3.1.0. Para los servicios REST se utiliza Spring Web MVC. Se ha utilizado la librería springdoc-openapi (32) para generar la documentación de los servicios web a partir del código fuente. Esto, además de crear la especificación OpenAPI (33) de los servicios, que es un estándar para la especificación de servicios REST, nos permitiría utilizar Swagger UI (34) para explorar y probar el API, aunque no está disponible actualmente. Como se indicó anteriormente, para las pruebas se ha usado Postman.

Para interactuar con la base de datos PostgreSQL (24) se ha utilizado Spring Data JDBC (35) y se han usado Java *records* como objetos para la transferencia de datos *(DTO – Data Transfer Object* en inglés). Para el mantenimiento de la base de datos, se ha integrado la librería Liquibase (36) con Sprin Boot. Esto permite ejecutar scripts de actualización de la base de datos en el arranque de la aplicación.

Respecto a la seguridad, se ha utilizado Spring Security. En este caso, al tener la autenticación con Google, hemos tenido que configurar nuestra aplicación como un servidor de recursos OAuth2.

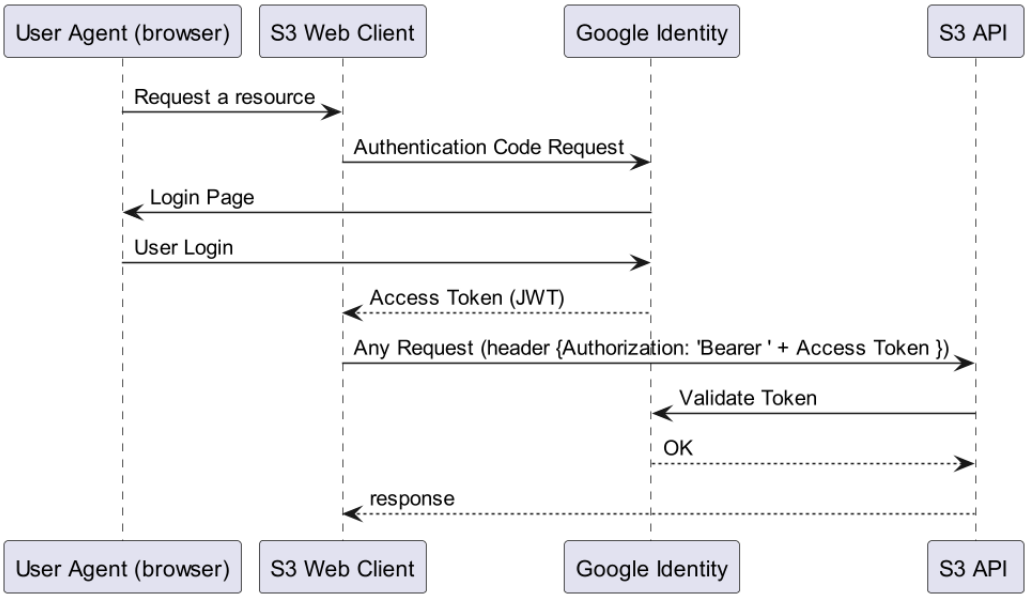


Ilustración 21- Diagrama de secuencia OAuth2 servidor de recursos

Finalmente, para interactuar con los repositorios S3 se utilizan las librerías de Amazon AWS SDK. Inicialmente se utilizó con esta librería un cliente HTTP de Amazon llamado CRT (37), que en principio tiene un rendimiento muy alto porque está implementado nativamente. Sin embargo, se descartó finalmente porque al crear la imagen de Docker usando como base Alpine, se vio que algunas funcionalidades de la aplicación desarrollada tales como la descarga de ficheros, estaba fallando.

El motivo es que actualmente hay un problema con esta librería en la distribución de Linux Alpine (38). En este caso se optó por usar un cliente HTTP alternativo para poder mantener el tamaño de la imagen Docker reducida. La distribución Alpine es popular en contenedores porque su tamaño es muy pequeño comparado con el resto y esto, entre otras cosas, permite reducir el coste de los despliegues en la nube.

Frontend

El proyecto del *frontend* se llama **s3client-web** y su código fuente puede obtenerse aquí: https://gitlab.com/jesuscl74-tfg/s3client-web

El proyecto está desarrollado principalmente en el lenguaje TypeScript (39) y el *framework* principal es Angular (40) versión 15.2 y he utilizado los componentes Angular Material (41) versión 15.2.

Se utiliza la librería de Google GSI client, para integrar la autenticación con Google y también utilizamos la librería jose (42), que es una librería que entre otras cosas da soporte sobre JWT (JSON Web Tokens) (43). Tal y como se puede ver en la Ilustración 21- Diagrama de secuencia OAuth2 servidor de recursos, el token que obtenemos de Google es un JWT, y utilizando JOSE podemos obtener de este token información básica del usuario, tal como su nombre completo y dirección de correo de electrónico.

A continuación vemos la decodificación de un token devuelto por Google. Esta representación la he obtenido desde la página web jwt.io:

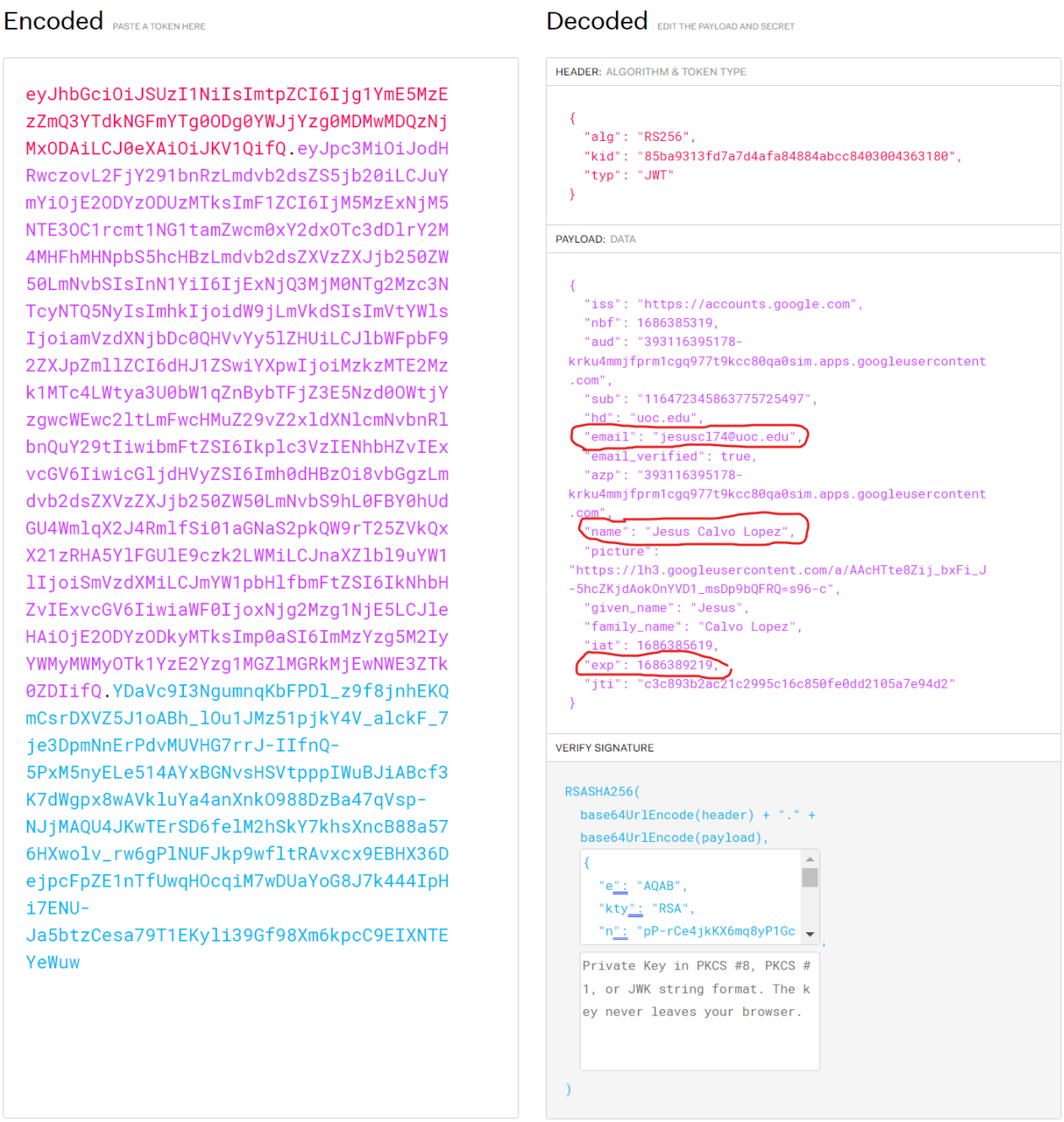


Ilustración 22 - Descodificación del token JWT devuelto por el servicio de autenticación de Google

Como podemos observar, en la ilustración, también se ha resaltado el campo **exp**. Este campo nos indica cuándo va a expirar el token, y lo utilizamos para implementar la funcionalidad de *auto-logout*, o desconectar la sesión del usuario de manera automática. Idealmente se debería intentar prolongar la sesión, por ejemplo, obteniendo un nuevo token, pero no fue posible por falta de tiempo. Sería una funcionalidad interesante para añadir en el futuro.

Otro aspecto interesante de la implementación de la aplicación web es el uso las rutas de Angular, que permite definir la navegación de la aplicación a partir de la URL y viceversa. Esto ha permitido definir rutas para el contenido basándonos en parámetros incluidos en la URL, tales como por ejemplo el nombre del *bucket* al que queremos conectar y el *path* que queremos explorar dentro de ese *bucket*.

Así pues, en la aplicación se han definido dos URL estáticas: */login* y */connections* que nos redirigen a sus respectivos componentes. Sin embargo, para el contenido tenemos partes de la URL dinámicas, es decir que varían en función de algunos parámetros. En nuestro caso los parámetros son el nombre del *bucket* y el *path* dentro del mismo. Así pues, la URL para el contenido sería:

/content/{nombre del bucket};path={path codificado para la URL}

Por ejemplo, a continuación vemos cómo dentro de la aplicación nos hemos posicionado dentro de un subdirectorio y cómo esto se refleja tanto en el interfaz de usuario como en la URL del navegador:

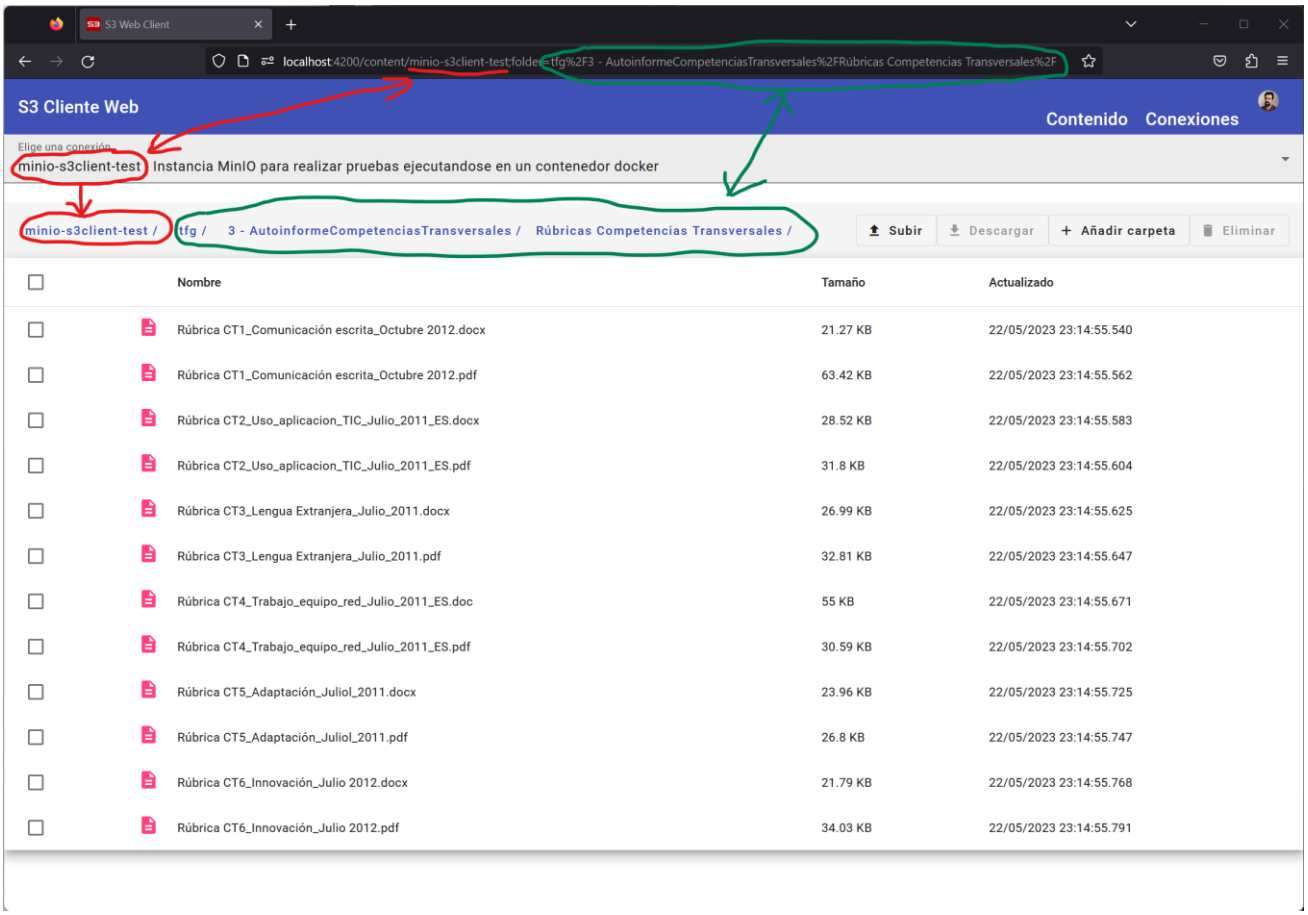


Ilustración 23 - Navegación dentro del contenido de un directorio

Sin embargo, también es posible navegar desde la URL. Es decir, si tuviésemos guardado el enlace e intentamos acceder directamente desde la URL del navegador, se cargará la aplicación de nuevo, lo cual no es óptimo para una aplicación SPA, pero al cargar la aplicación, gracias a las rutas definidas, podemos determinar qué componentes del interfaz de usuario mostrar, y al iniciar dichos componentes, obtener los parámetros que necesitamos para obtener el contenido.

Esta posibilidad de recargar la página completa, también nos ha obligado a implementar la funcionalidad de inicio de sesión automático o *auto-login*. La necesidad surge al añadir la seguridad. Básicamente todos los componentes de la aplicación, excepto el de inicio de sesión o *login*, requieren que el usuario esté autenticado. De lo contrario, el usuario es redirigido al componente de inicio de sesión. Pues bien, si el usuario ya ha iniciado sesión y recarga la página por el motivo que sea, si no implementamos este mecanismo de inicio de sesión automático lo más probable es que forcemos al usuario a identificarse de nuevo.

Esto ocurría al menos en las versiones iniciales de la aplicación, ya que normalmente se almacenaba la información del usuario autenticado como variables o propiedades en el servicio de autenticación. Este servicio está definido como un *Singleton,* y es accedido por casi todos los componentes, por lo que a priori no es una mala estrategia para almacenar esta información. Sin embargo, estos datos se mantienen en la memoria usada por la aplicación, y esta se pierde al recargar la página, ya que se reinicia la aplicación.

Así pues, se optó por implementar esta funcionalidad de inicio de sesión automático, que básicamente consiste en serializar la información del usuario autenticado, incluido el token JWT, y guardarla en el almacenamiento local del navegador (*local storage*) cuando se inicia sesión. Posteriormente, si se recarga la página, y por tanto toda la aplicación, intentamos iniciar la sesión automáticamente con los datos recuperados del almacén local.

4.3. Seguimiento del proyecto y principales cambios realizados sobre el diseño original

Durante el desarrollo del proyecto, al final de cada entrega, se fue revisando el progreso con respecto a la planificación inicial. Ya en la primera entrega se apreció un retraso debido principalmente a las tareas de programación del interfaz web. La causa es que, aunque ya estaba familiarizado con las tecnologías utilizadas en el servidor, no era así con Angular y su ecosistema que eran completamente nuevos para mí, por lo que su aprendizaje fue una de las cosas que más me costaron.

Este retraso provocó que algunas de las tareas previstas no fueran desarrolladas, tales como las relacionadas con la subida o descarga de ficheros por lotes, ya sea mediante selección múltiple o mediante selección de directorios.

Igualmente, esto provocó el mayor cambio producido sobre el diseño original, que está relacionado con la seguridad y la autenticación o identificación de los usuarios. Finalmente, en lugar de implementar la típica pantalla de *login* y de registro de usuarios, almacenando los datos en la base de datos, se ha optado por utilizar la autenticación con Google.

Aprovechando que las cuentas de correo de la UOC son identificadores válidos en Google, se ha utilizado su API para identificar al usuario y registrarlo en nuestra base de datos. Esto nos evita entre otras cosas la necesidad de gestionar y almacenar contraseñas en nuestro sistema.

De este modo, el proceso de identificación o conexión del usuario en la aplicación es distinto del inicialmente planteado. Por ejemplo, el registro es automático, la primera vez que un usuario se identifica, por lo que no es necesario solicitar en

un formulario sus datos, ya que se pueden obtener directamente de los datos del perfil de Google.

También se añaden aspectos a tener en cuenta que inicialmente no estaban previstos, debido al desconocimiento de la arquitectura de la aplicación web, tales como la necesidad de hacer auto-login en caso de que el usuario refresque la página desde el navegador.

5. Conclusiones

En este apartado, describiré las conclusiones a las que he llegado tras finalizar el trabajo.

5.1. Lecciones aprendidas

Durante este trabajo se han aprendido o consolidado muchos de los conocimientos adquiridos durante la carrera, destacando especialmente los correspondientes a las siguientes asignaturas: Gestión de proyectos, Ingeniería de requisitos, Interacción persona ordenador, Ingeniería del software de componentes y sistemas distribuidos, y Proyecto de desarrollo del software.

Sobre el uso de la metodología ágil he aprendido que lo más importante en el análisis inicial no es la precisión, sino es tener una idea clara de cuál es nuestro objetivo y adónde queremos llegar, aún sin conocer del todo el cómo hacerlo. Por ello me ha gustado mucho la idea de los mapas de historias de usuario, ya que permiten visualizar muy bien lo que hay que hacer y cómo encaja en el conjunto de la aplicación.

En cuanto a las tecnologías utilizadas, destacaría el aprendizaje de Angular y TypeScript para el desarrollo del *frontend.* He de decir que ya tenía experiencia previa en el desarrollo de aplicaciones web. Conocía HTML, CSS y Javascript, aunque la última vez que lo utilicé fue en el 2015. He de decir que me ha sorprendido gratamente cómo ha evolucionado este ecosistema en todos estos años. Sin embargo, para ponerme al día he tenido que hacer uso intensivo de los recursos proporcionados por la universidad, en especial de los libros y cursos del sitio web de O'Reilly (44). Me gustaría destacar un curso sobre Angular que me ha ayudado mucho y que se llama "Angular - The Complete Guide [2023 Edition]", de Maximiliam Schwarzmüller (45). Es un curso audiovisual bastante completo.

Respecto a la parte del servidor me ha resultado más familiar, ya que habitualmente trabajo en proyectos con Java y Spring Boot, aunque de igual modo he intentado salir de mi zona de confort utilizando versiones nuevas, tales como la versión 3.1 de Spring Boot y la versión 17 de Java.

5.2. Objetivos logrados

En general, a nivel funcional y de usabilidad, creo que la aplicación cumple los objetivos mínimos planteados. Aunque hay algunas funcionalidades que estaban incluidas en el análisis que no se han desarrollado, tales como la subida o descarga de varios ficheros a la vez, o el filtrado y ordenamiento de la lista de contenido por nombre, pienso que estas son funcionalidades necesarias en un producto final, pero no son imprescindibles para este proyecto.

Sin embargo, pienso que se echa en falta una mejor gestión de errores en la aplicación web. Se han realizado validaciones en los formularios y se muestra el error al usuario cuando el dato introducido no es válido. El problema es que básicamente si hay un error en el servidor, por ejemplo, si el servicio no está disponible, o los parámetros de conexión no son correctos, el usuario no recibe ningún mensaje. Accediendo a la consola del navegador se pueden ver los errores, pero esta gestión de errores no es admisible en un producto final.

Por otro lado, estoy muy contento de haber podido incluir un mínimo de seguridad en el proyecto. Para este tipo de aplicaciones la seguridad es imprescindible, ya que hay que asegurar que podemos identificar a los usuarios y que sólo tengan acceso a los repositorios que ellos mismos han configurado en la aplicación.

5.3. Seguimiento de la planificación

El seguimiento de la planificación ha sido imprescindible para poder seguir enfocado en los problemas importantes. Como se ha explicado en apartados anteriores, prácticamente he tenido que aprender desde cero el uso de las tecnologías de desarrollo web. Esto me ha llevado mucho tiempo y me ha impedido desarrollar todas las funcionalidades que me hubiera gustado.

Sin embargo, el enfoque ágil me ha permitido detectar este problema a tiempo y realizar algunas correcciones, tales como sustituir el registro y autenticación propia de usuarios por la autenticación de Google. En este caso, lo importante era añadir una seguridad mínima y que pudiésemos identificar a los usuarios y esto se ha conseguido.

5.4. Evoluciones posibles de la aplicación

En el futuro, como continuación de este trabajo, además de realizar las tareas mencionadas anteriormente, se podrían desarrollar más las configuraciones de conexión a los repositorios. Actualmente contienen datos muy básicos y es posible que tuviésemos dificultades o no fuese posible conectar con determinados proveedores. Desarrollando la misma idea, quizás incluso se podrían añadir nuevos módulos o servicios para interactuar con otros repositorios que no utilicen S3, o incluso por qué no, servidores FTP o SFTP.

Otra línea a seguir sería añadir más formas de autenticarse para no depender completamente de Google, y quizás implementar finalmente el registro de usuarios.

Por último, podría ser interesante poder gestionar grupos de usuarios que tuvieran acceso a repositorios compartidos. Esto sería interesante para empresas o corporaciones. En este caso, podría distinguirse un tipo de rol administrador, que gestionase las conexiones a repositorios y los usuarios que tienen acceso a los mismos, y otro rol que sólo pudiera gestionar contenido.

6. Glosario

API: API es el acrónimo de Application Programming Interface, que en español significa Interfaz de Programación de Aplicaciones. Una API es una pieza de código que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades.

bucket: Un bucket en la terminología de S3, es un contenedor de objetos.

JWT: JSON Web Token (abreviado JWT) es un estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o claims en inglés.

OAuth2: Open Authorization es un estándar abierto que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Se trata de un protocolo propuesto por Blaine Cook y Chris Messina, que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web.

Object: Un object u objeto en la terminología de S3 es un fichero junto con los metadatos que lo definan.

S3: Simple Storage Service es el nombre de un servicio de almacenamiento de objetos de Amazon Web Services que ofrece escalabilidad, disponibilidad, seguridad y rendimiento.

SPA: Una single-page application (SPA), o aplicación de página única, es una aplicación web ó un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios, como si fuera una aplicación de escritorio. En un SPA todos los códigos de HTML, JavaScript, y CSS se cargan una sola vez1, los recursos necesarios se cargan dinámicamente cuando lo requiera la página, normalmente como respuesta a las acciones del usuario.

7. Bibliografía

1. **Amazon Simple Storage Service Documentation. [En línea] [Citado el: 20 de abril de 2023.] https://docs.aws.amazon.com/s3/index.html.**

2. Julia Palmer, Jerry Rozeman, Chandra Mukhyala, Jeff Vogel. Magic Quadrant for Distributed File Systems and Object Storage. [En línea] 19 de octubre de 2022. [Citado el: 12 de junio de 2023.] https://www.gartner.com/document/4020259?ref=solrAll&refval=369616557.

3. AWS. [En línea] [Citado el: 20 de abril de 2023.] https://aws.amazon.com/es/.

4. DigitalOcean. [En línea] [Citado el: 20 de abril de 2023.] https://www.digitalocean.com/.

5. Cloudian - HyperStore Object Storage. [En línea] [Citado el: 20 de abril de 2023.] https://cloudian.com/products/hyperstore/.

6. NetApp - StorageGRID. [En línea] [Citado el: 20 de abril de 2023.] https://www.netapp.com/es/data-storage/storagegrid/.

7. Akamai. [En línea] [Citado el: 20 de abril de 2023.] https://www.linode.com/.

8. DreamHost - Secure Cloud Storage Hosting. [En línea] [Citado el: 20 de abril de 2023.] https://www.dreamhost.com/cloud/storage/.

9. MinIO. [En línea] [Citado el: 20 de abril de 2023.] https://min.io/.

10. Wasabi. [En línea] [Citado el: 20 de abril de 2023.] https://wasabi.com/es/.

11. Vultr - Object Storage. [En línea] [Citado el: 20 de abril de 2023.] https://www.vultr.com/products/object-storage/.

12. Filezilla Pro. [En línea] [Citado el: 20 de abril de 2023.] https://filezilla-project.org/filezilla_pro.php.

13. S3 Browser. [En línea] [Citado el: 20 de abril de 2023.] https://s3browser.com/.

14. AWS CLI. [En línea] [Citado el: 20 de abril de 2023.] https://aws.amazon.com/es/cli/.

15. WinSCP. [En línea] [Citado el: 11 de junio de 2023.] https://winscp.net/eng/download.php.

16. Patton, Jeff y Economy, Peter. *User Story Mapping.* s.l. : O'Reilly, 2014. ISBN 978-1-491-90490-9.

17. Miro. [En línea] [Citado el: 5 de mayo de 2023.] https://miro.com.

18. Figma. [En línea] [Citado el: 20 de abril de 2023.] https://www.figma.com/.

19. GitLab. [En línea] [Citado el: 3 de junio de 2023.] https://about.gitlab.com/.

20. Brown, Simon. [En línea] [Citado el: 6 de abril de 2023.] https://c4model.com/.

21. Microsoft Azure. [En línea] [Citado el: 14 de abril de 2023.] https://portal.azure.com.

22. OpenJDK - JDK 17. [En línea] [Citado el: 3 de junio de 2023.] https://openjdk.org/projects/jdk/17/.

23. Spring Boot. [En línea] [Citado el: 3 de junio de 2023.] https://spring.io/projects/spring-boot.

24. PostgreSQL. [En línea] [Citado el: 5 de mayo de 2023.] https://www.postgresql.org/.

25. IntelliJ IDEA. [En línea] JetBrains. [Citado el: 3 de junio de 2023.] https://www.jetbrains.com/idea/.

26. Microsoft. Visual Studio Code. [En línea] [Citado el: 3 de junio de 2023.] https://code.visualstudio.com/.

27. Postman. [En línea] [Citado el: 3 de junio de 2023.] https://www.postman.com/.

28. Docker Compose. [En línea] [Citado el: 20 de abril de 2023.] https://docs.docker.com/compose/.

29. NodeJS release 18. [En línea] [Citado el: 3 de junio de 2023.] https://nodejs.org/en/blog/announcements/v18-release-announce.

30. Gradle. [En línea] [Citado el: 3 de junio de 2023.] https://gradle.org/.

31. Git. [En línea] [Citado el: 3 de junio de 2023.] https://git-scm.com/.

32. springdoc-openapi. [En línea] [Citado el: 8 de junio de 2023.] https://springdoc.org/.

33. OpenAPI. [En línea] [Citado el: 8 de junio de 2023.] https://www.openapis.org/.

34. Swagger UI. [En línea] [Citado el: 8 de junio de 2023.] https://swagger.io/tools/swagger-ui/.

35. Spring Data JDBC. [En línea] [Citado el: 8 de junio de 2023.] https://spring.io/projects/spring-data-jdbc.

36. Liquibase. [En línea] [Citado el: 8 de junio de 2023.] https://www.liquibase.org/.

37. AWS CRT HTTP Client. [En línea] [Citado el: 8 de junio de 2023.] https://aws.amazon.com/es/blogs/developer/announcing-availability-of-the-aws-crt-http-client-in-the-aws-sdk-for-java-2-x/.

38. Alpine Linux. [En línea] [Citado el: 8 de junio de 2023.] https://www.alpinelinux.org/.

39. TypeScript. [En línea] [Citado el: 3 de junio de 2023.] https://www.typescriptlang.org/.

40. Angular. [En línea] [Citado el: 2 de junio de 2023.] https://angular.io/.

41. Angular Material. [En línea] [Citado el: 3 de junio de 2023.] https://material.angular.io/.

42. jose - JavaScript module for JSON Object Signing and Encryption. [En línea] [Citado el: 10 de junio de 2023.] https://github.com/panva/jose.

43. JWT - JSON Web Tokens. [En línea] [Citado el: 10 de junio de 2023.] https://jwt.io.

44. O'Reilly Learning. [En línea] [Citado el: 19 de junio de 2023.] https://learning.oreilly.com/home/.

45. Schwarzmuller, Maximiliam. *Angular - The Complete Guide.* s.l. : Packt Publishing, 2023.

46. Angular. [En línea] https://angular.io/docs.

47. AWS SDK para Java. [En línea] [Citado el: 20 de abril de 2023.] https://aws.amazon.com/es/sdk-for-java/.

48. Docker. [En línea] [Citado el: 20 de abril de 2023.] https://www.docker.com/.

49. Guides | Angular Material. [En línea] [Citado el: 15 de mayo de 2023.] https://material.angular.io/guides.

Anexo I – Manual de instalación

Requisitos

Para la instalación y ejecución del proyecto necesitamos tener instalado lo siguiente:

- Docker Compose – Normalmente en Sistemas Operativos Windows y Mac OS podemos tenerlo al instalar Docker Desktop. Para este trabajo la versión que tengo actualmente instalada es la 4.19.0.
- Opcionalmente necesitaremos Git – En mi caso tengo instalada la versión 2.41. No es absolutamente necesarios si nos descargamos directamente el fichero *docker-compose.yml* desde la página web de GitLab

Procedimiento

A continuación, se enumeran los pasos a seguir para poder ejecutar el proyecto:

1. Obtener el fichero *docker-compose.yml* que contiene la configuración con todas las dependencias necesarias. Este fichero se encuentra en el proyecto utilizado para el *frontend*, es decir, en s3client-web. Para obtener dicho fichero podemos hacerlo de dos formas:
    a. Utilizar Git para obtener todo el código fuente del proyecto: git clone https://gitlab.com/jesuscl74-tfg/s3client-web.git
    b. Directamente acceder al fichero desde la web de GitLab y descargarlo: https://gitlab.com/jesuscl74-tfg/s3client-web/-/blob/master/docker-compose.yml
2. Desde la línea de comandos, debemos movernos al directorio donde está el fichero, que será el directorio raíz del proyecto si hemos optado por la opción (a) de clonar el proyecto mediante git, o el directorio donde hayamos descargado el fichero. Por ejemplo, desde mi PC he ejecutado lo siguiente: cd .\workspace\tfg\s3client-web\
3. Ejecutar el siguiente comando: docker compose up -d
Veremos algo parecido a lo siguiente:

```
PS C:\Users\jesus\workspace\tfg\s3client-web> docker compose up -d
[+] Running 6/6
 ✓ Network s3client-web_default     Created
 ✓ Container s3client-web-minio2-1  Started
 ✓ Container database               Started
 ✓ Container s3client-web-minio-1   Started
 ✓ Container s3client-api           Started
 ✓ Container s3client-web           Started
PS C:\Users\jesus\workspace\tfg\s3client-web>
```

Ilustración 24 - Uso de docker compose

4. Una vez arrancados los contenedores Docker, ya podemos acceder a la aplicación mediante la siguiente URL: http://localhost:18080

5. Finalmente, después de utilizar la aplicación, si queremos parar los contenedores, deberemos ejecutar lo siguiente desde el mismo directorio en el que estábamos

situados:
docker compose down

Configuración inicial de los repositorios para las pruebas

Para poder utilizar la aplicación debemos tener acceso a algún repositorio S3. Para facilitar las pruebas, en el fichero docker-compose mencionado en el apartado anterior se incluyen dos contenedores de MinIO, que es un servidor que entre otras cosas implementa el interfaz S3 de Amazon para gestionar contenido.

Dichos contenedores incluyen además una aplicación web de administración que podremos utilizar para crear *buckets* y cargar contenido inicial, ya que la primera vez que arranquemos la aplicación, estará todo vacío. Así pues, tenemos que con los contenedores funcionando podemos acceder a las siguientes URLs para gestionar los servidores MinIO:

- Consola de administración del servidor 1 (minio): http://localhost:9001
- Consola de administración del servidor 2 (minio2): http://localhost:9003

Los credenciales para acceder son las mismas en ambos servidores, siendo el usuario/contraseña: minioadmin/minioadmin

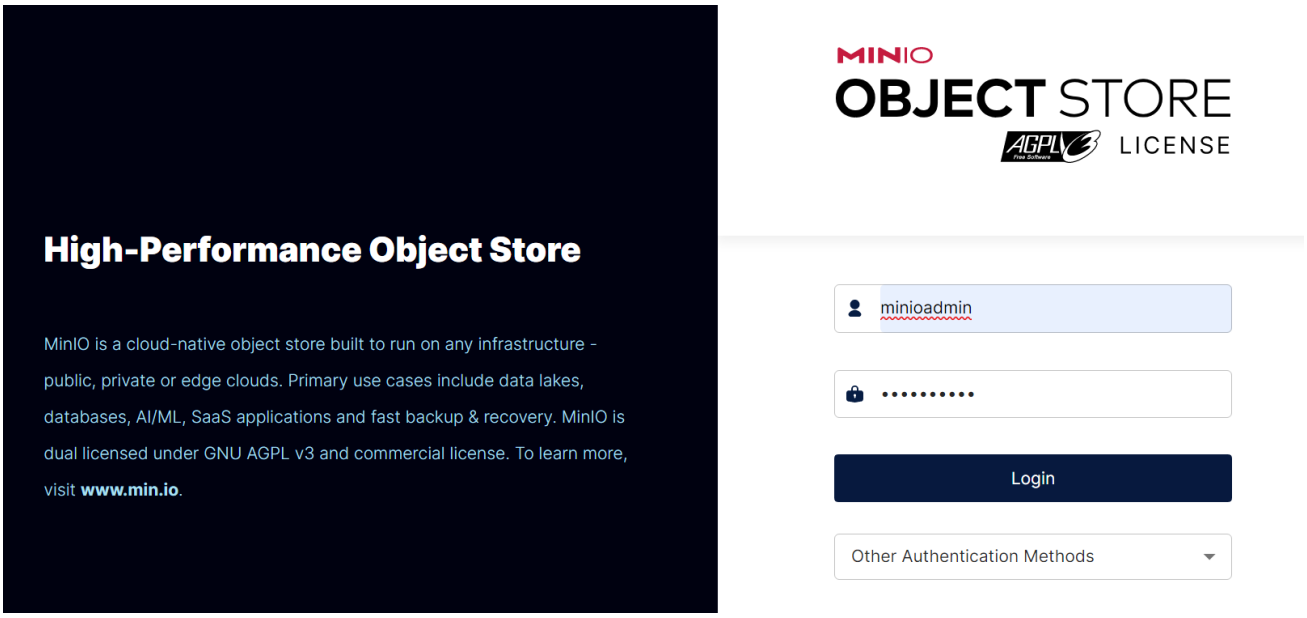


Ilustración 25 - Inicio de sesión en la consola de MinIO

Al introducir los credenciales y pulsar en el botón Login, la aplicación nos redirige al Object Browser, pero la primera vez estará vacío, ya que no tenemos *buckets* ni objetos, y se nos sugerirá crear un *bucket* para empezar:

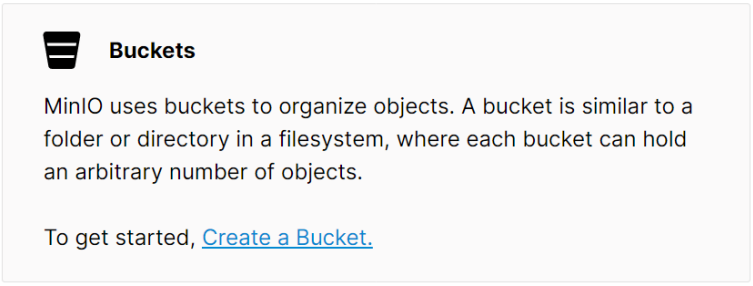


Ilustración 26 - Consola de MinIO mostrando que no hay buckets definidos

Si seguimos el enlace, podemos crear un *bucket*, que básicamente es una unidad de organización de contenido. Sobre dicha unidad podemos definir una serie de políticas o aspectos que se aplicarán sobre los objetos añadidos. Estas políticas se refieren a: seguridad, versionado, bloqueo, cuota y retención. Para nuestras pruebas podemos dejar las opciones por defecto y simplemente asignar un nombre. Este nombre es importante, porque es el que tenemos que referenciar desde la aplicación s3client. Por ejemplo, vamos a llamar al *bucket* **unidad-secundaria.**

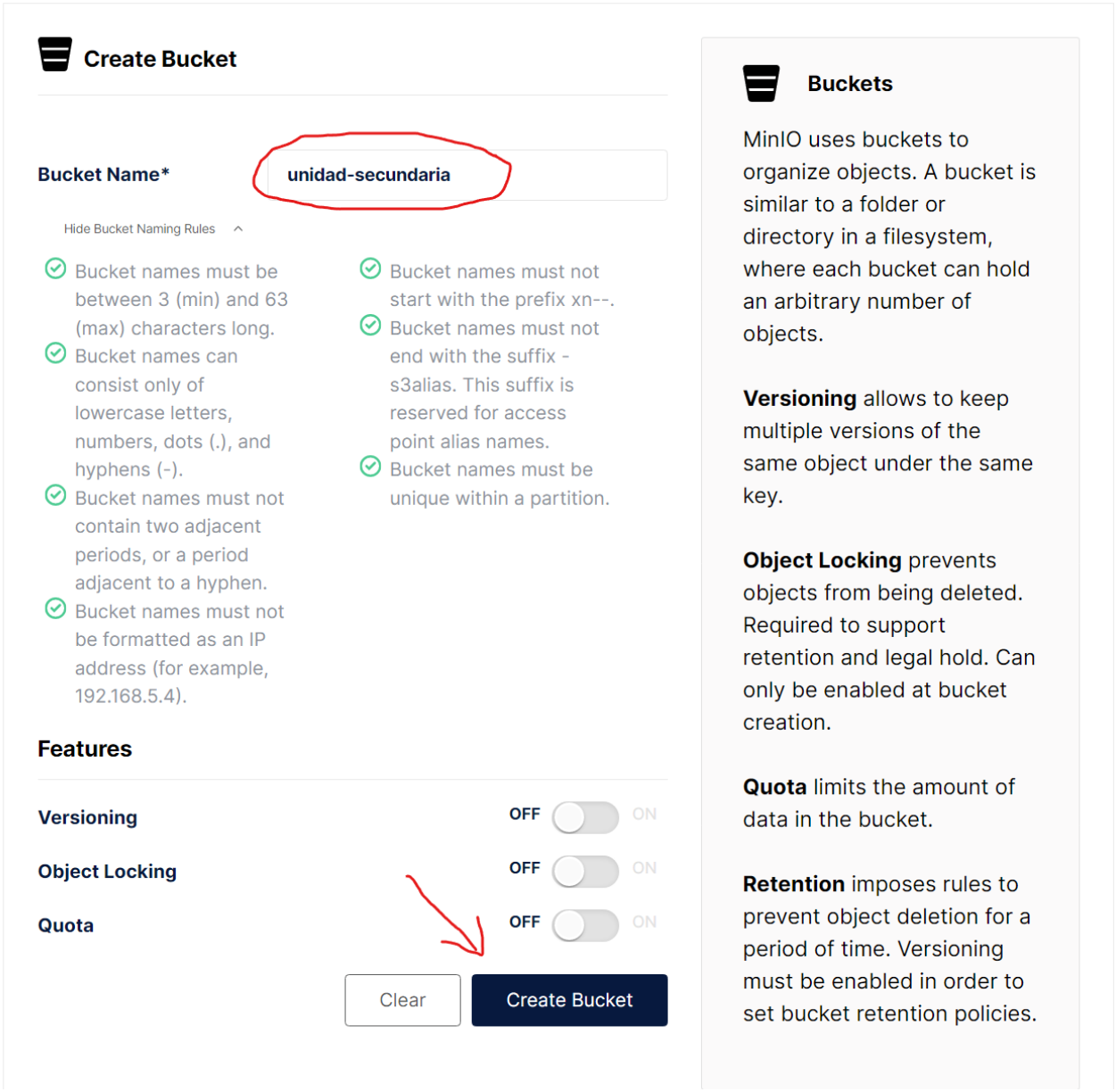


Ilustración 27 - Consola de MinIO, formulariode creación de bucket

Una vez creado el bucket, podríamos ir a la aplicación s3client y empezar a utilizarla. Sin embargo, aprovechando que estamos aquí podemos añadir contenido y así comprobar luego desde la aplicación que tenemos acceso al mismo. Para ello, tenemos que ir a la opción del menú de la izquierda “Object Browser”:

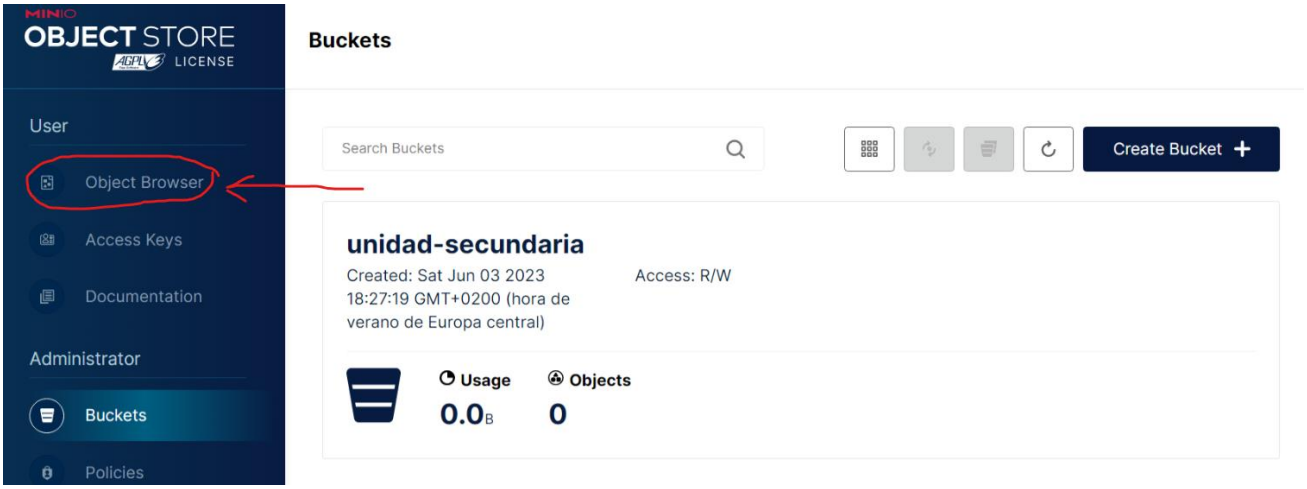


Ilustración 28 - Consola de MinIO, acceso al Object Browser

Aquí veremos que nos aparece una lista con los buckets, en nuestro caso, sólo aparecerá el que acabamos de crear.

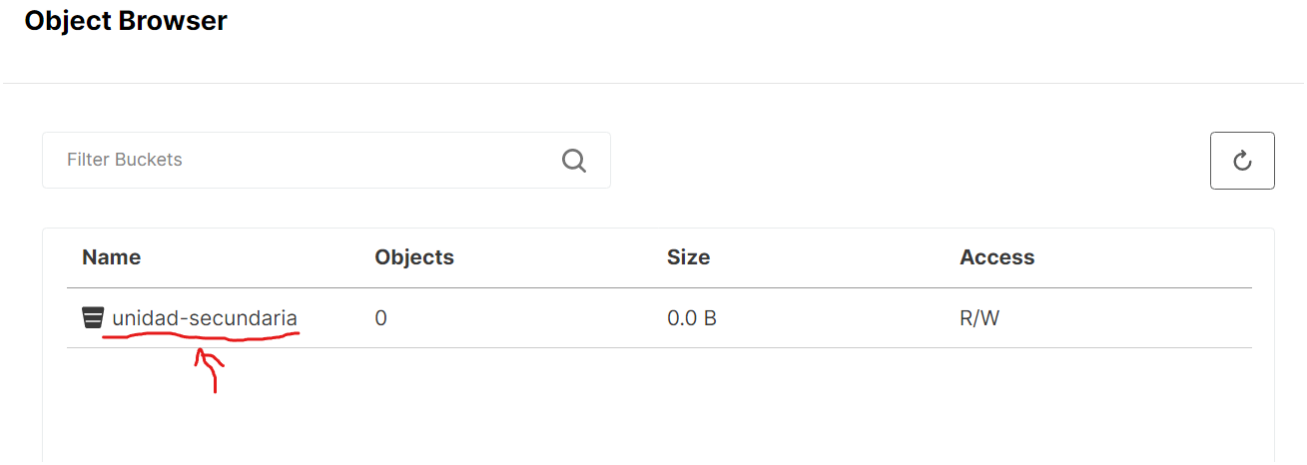


Ilustración 29 - Consola de MinIO, navegación dentro del Object Browser

Si hacemos click en el nombre del *bucket*, nos aparecerá la siguiente pantalla, desde la cual podremos subir algunos ficheros. Para ello tenemos varias opciones: las mostradas en el menú desplegable del botón “Upload” que nos permiten seleccionar ficheros o directorios, o también podemos desde el explorador de archivos de nuestro sistema operativo arrastrar y soltar. Esto es lo que he hecho finalmente, arrastrando un directorio donde tengo documentación relacionada con el TFG.

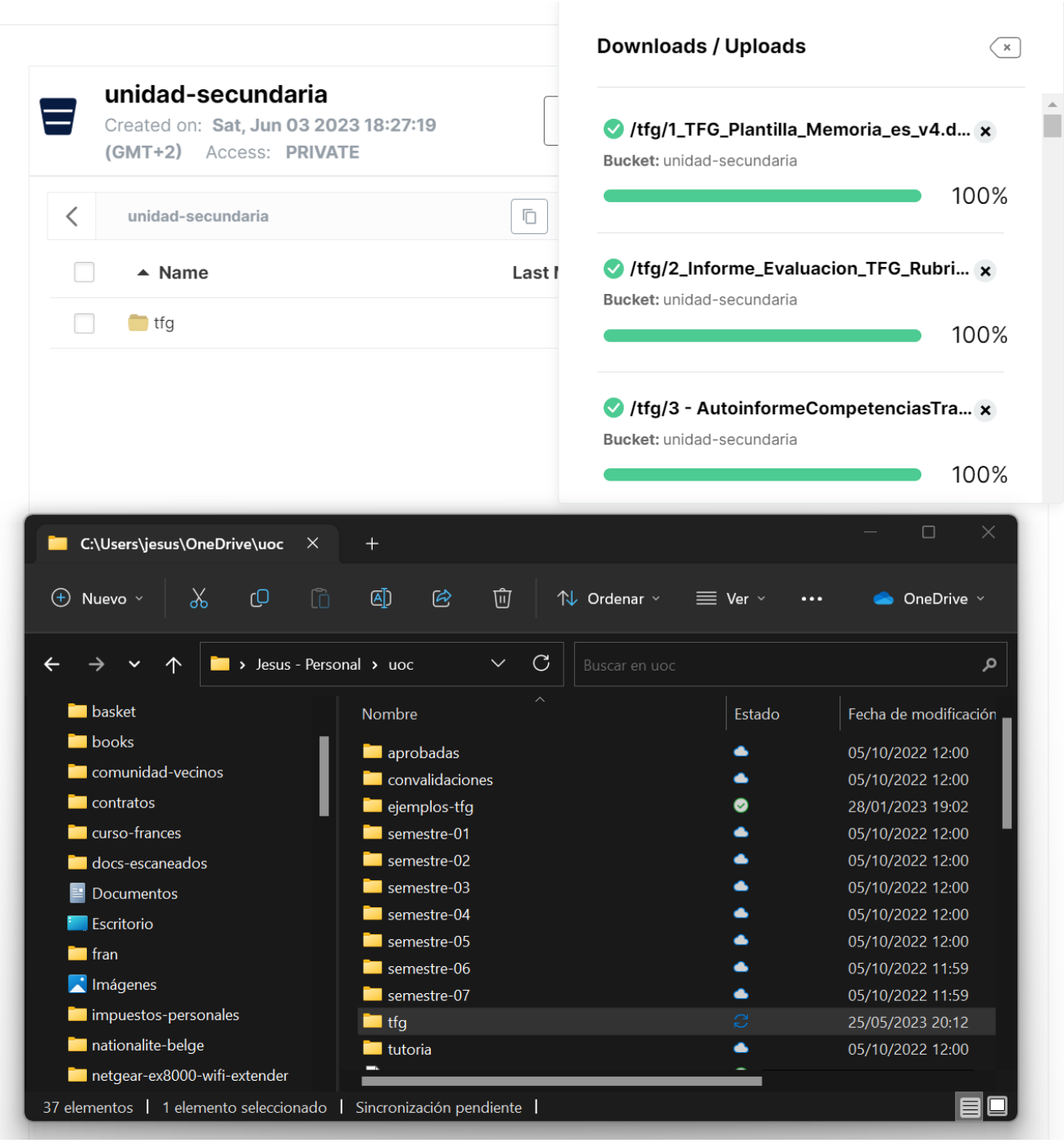


Ilustración 30 - Consola de MinIO, subida de ficheros arrastrando un directorio del explorador de ficheros al Object Browser

Ahora si que podemos entrar a la aplicación y configurarla para ver el contenido que acabamos de añadir.

Anexo II – Guía de uso

Una vez realizados los pasos descritos en el Anexo I, podemos entrar en la aplicación accediendo a la siguiente URL:

http://localhost:18080

Donde veremos la página de identificación o *login*.

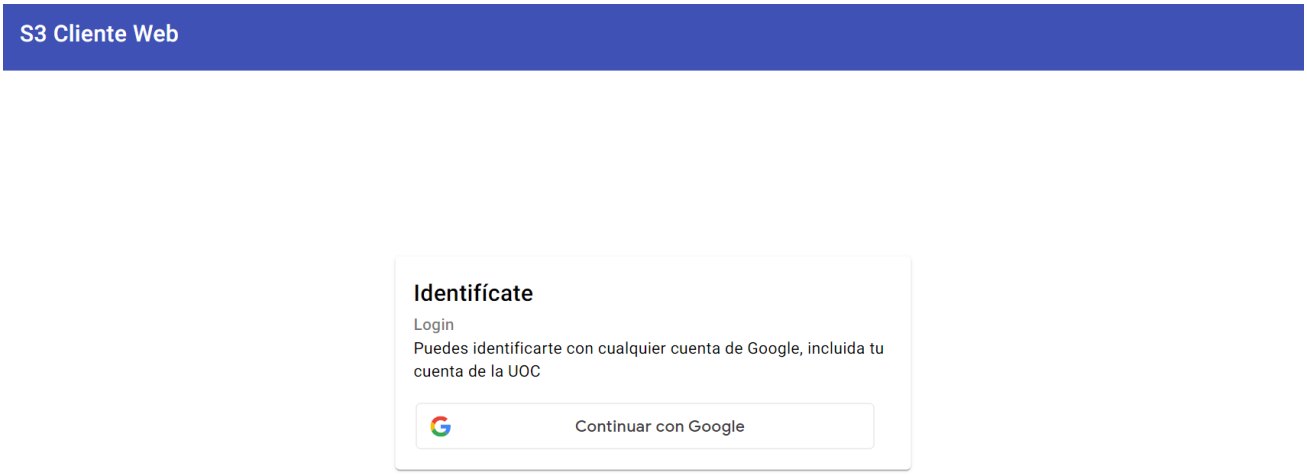


Ilustración 31 - S3 Cliente Web, inicio de sesión con Google

Como vemos, la única opción actualmente es identificarse con Google. Si pulsamos el botón, la primera vez nos pedirá consentimiento para poder acceder a los datos del perfil de la cuenta de Google elegida. Esto es un componente de Google y todo el flujo de inicio de sesión es controlado desde dicho componente. En mi caso aparece el siguiente diálogo:

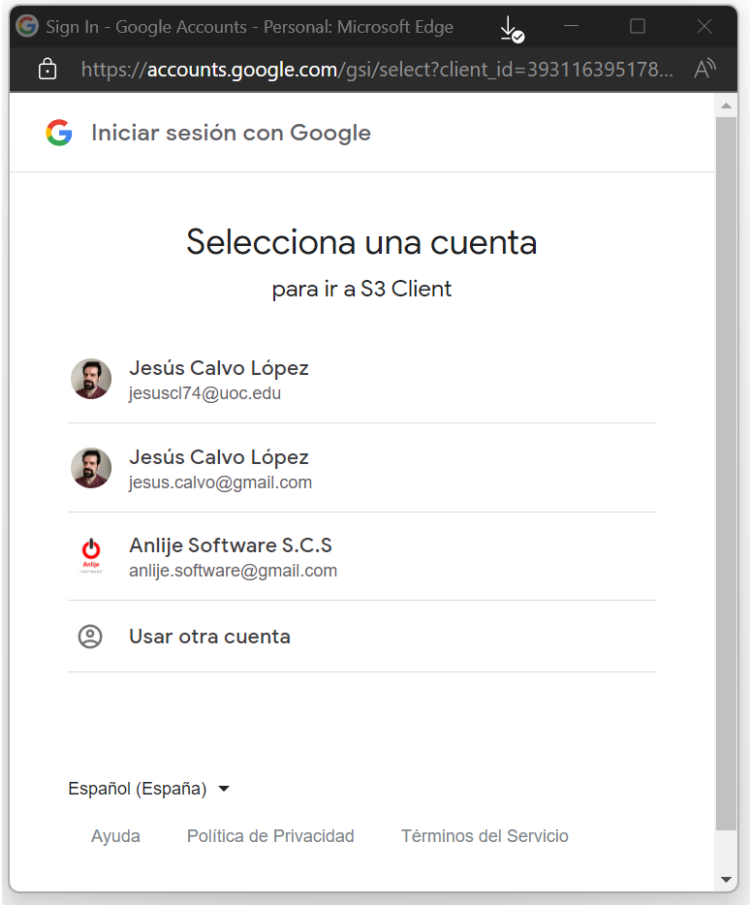


Ilustración 32 - Pantalla de inicio de sesión de Google

Si seleccionamos la cuenta de la UOC y continuamos el proceso de identificación accederemos a la aplicación. Una vez autenticados, en el menú superior nos aparecen dos opciones: Contenido y Conexiones. También vemos la información del usuario si hacemos *click* en la foto del perfil de la cuenta.

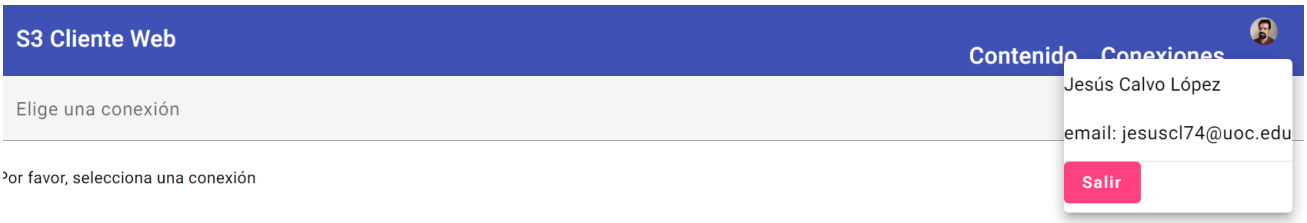


Ilustración 33 - S3 Cliente Web, información del usuario autenticado

Deberíamos configurar la conexión al bucket que hemos creado anteriormente. Para ello pulsaremos la opción de Conexiones y a continuación en Nueva conexión. En el formulario introduciremos los siguientes datos:

- *Bucket*: unidad-secundaria

- Descripción: Conexión de prueba
- *Endpoint*: http://minio2:9002 (el *bucket* se creó en el segundo servidor de MinIO)
- ID de acceso: minioadmin
- Contraseña de acceso: minioadmin

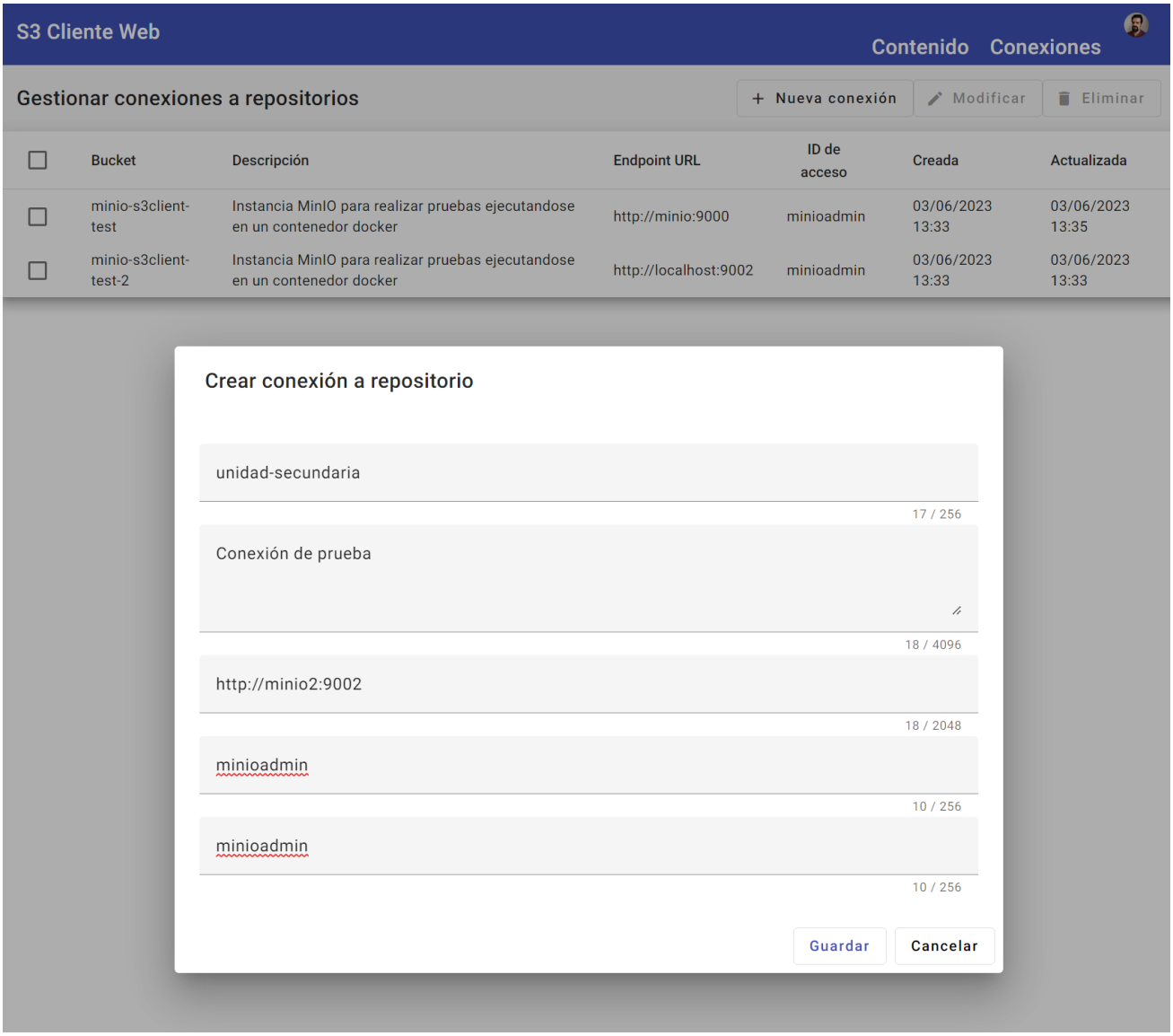


Ilustración 34 - S3 Cliente Web, creación de conexión a repositorio

Cuando pulsamos en Guardar se añade la conexión a la lista de conexiones del usuario. Ahora podemos ir a Contenido y seleccionar la nueva conexión.

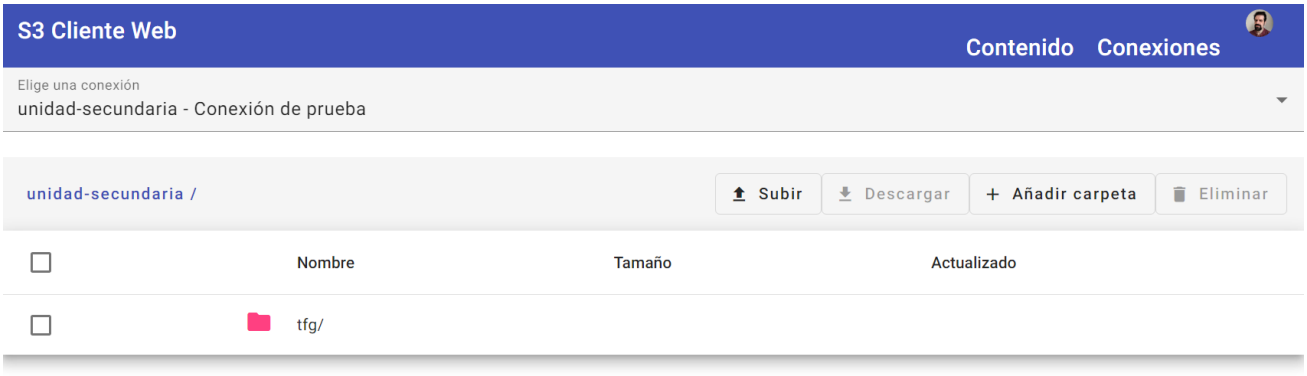


Ilustración 35 - S3 Cliente Web, selección de conexión a repositorio

Ahora podemos navegar por los directorios haciendo doble *click* en su nombre o en el icono de carpeta. También podemos navegar a un directorio padre haciendo *click* en los enlaces de la barra de navegación, siendo el nombre del *bucket* el directorio raíz:

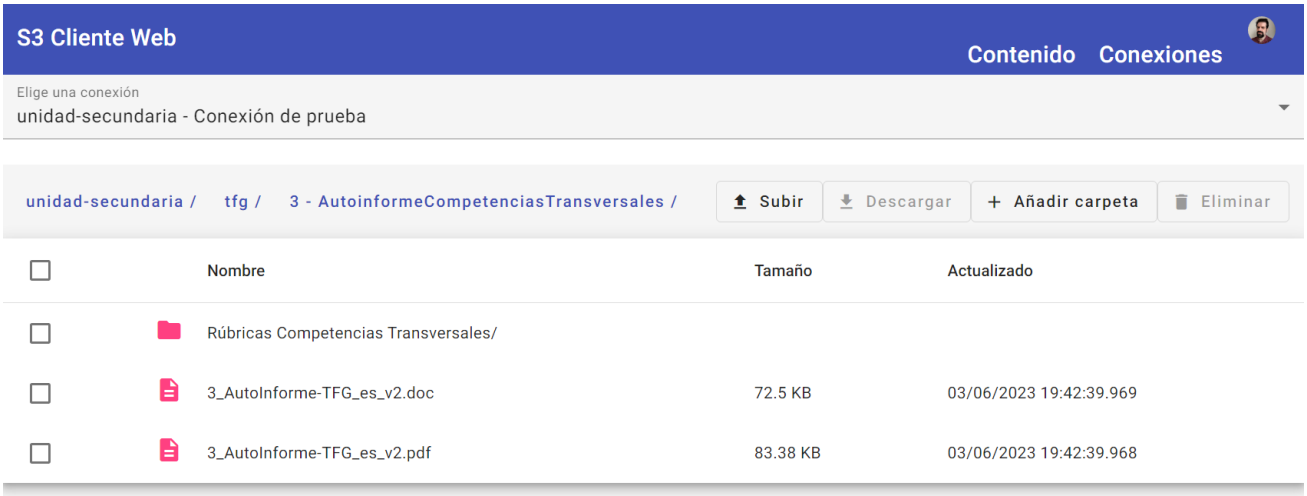


Ilustración 36, S3 Cliente Web, navegación por contenido de repositorio

Podemos descargar un fichero (no he implementado descarga múltiple o descarga de directorios), subir un fichero, crear una carpeta y eliminar un fichero.

Anexo III – Autenticación con Google

Para poder autenticarnos con el API de Google desde la aplicación, se han seguido unos pasos como registrar la aplicación en Google y proveer una serie de datos.

Básicamente tenemos que acceder a la aplicación Google Cloud Console en la siguiente URL:

https://console.cloud.google.com/

Desde allí, una vez que nos hemos identificado, registraremos nuestra aplicación y principalmente configuraremos dos aspectos: Credenciales y Consentimiento.

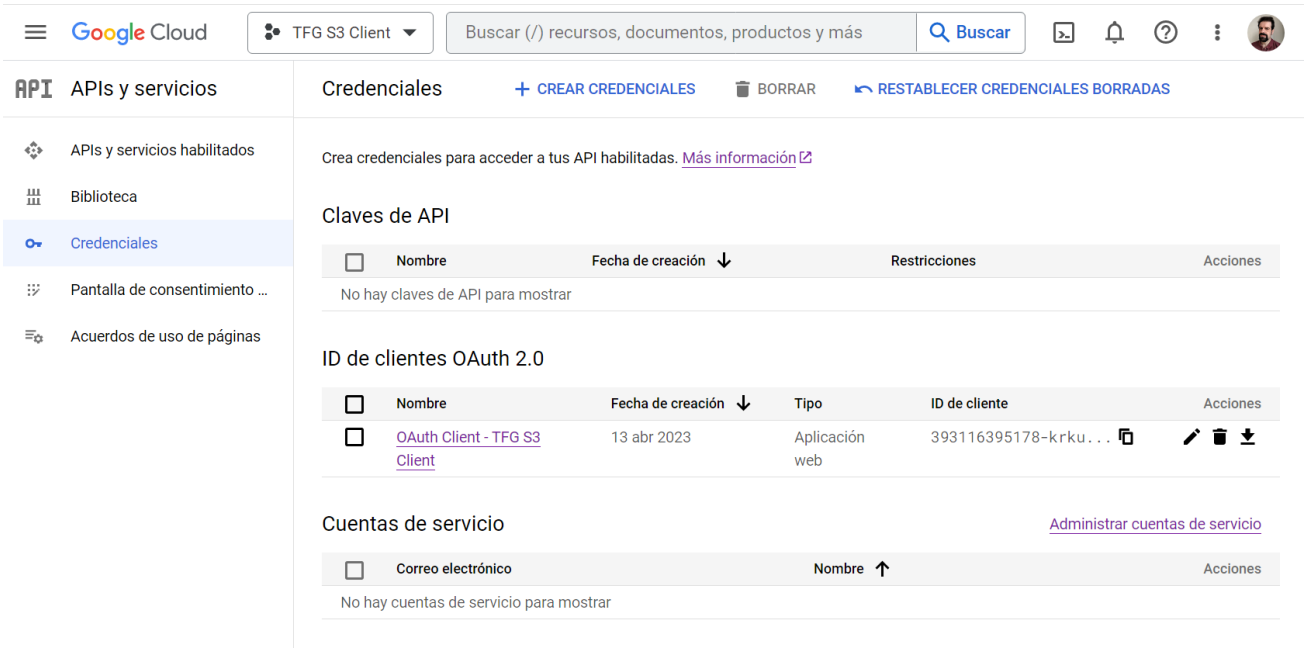


Ilustración 37 - Google Cloud Console

Credenciales

Aquí tenemos que crear un Identificador OAuth de nuestra aplicación.

Consentimiento

Desde aquí podemos configurar tipos de usuario, en nuestro caso hemos seleccionado usuario externo, para poder probar con usuarios que no sean de la UOC. También se puede añadir información sobre la aplicación, que se le mostrará al usuario durante el proceso de login con Google. Y lo más importante, los permisos, en este caso queremos añadir acceso a: *openid*, *email* y *profile*:

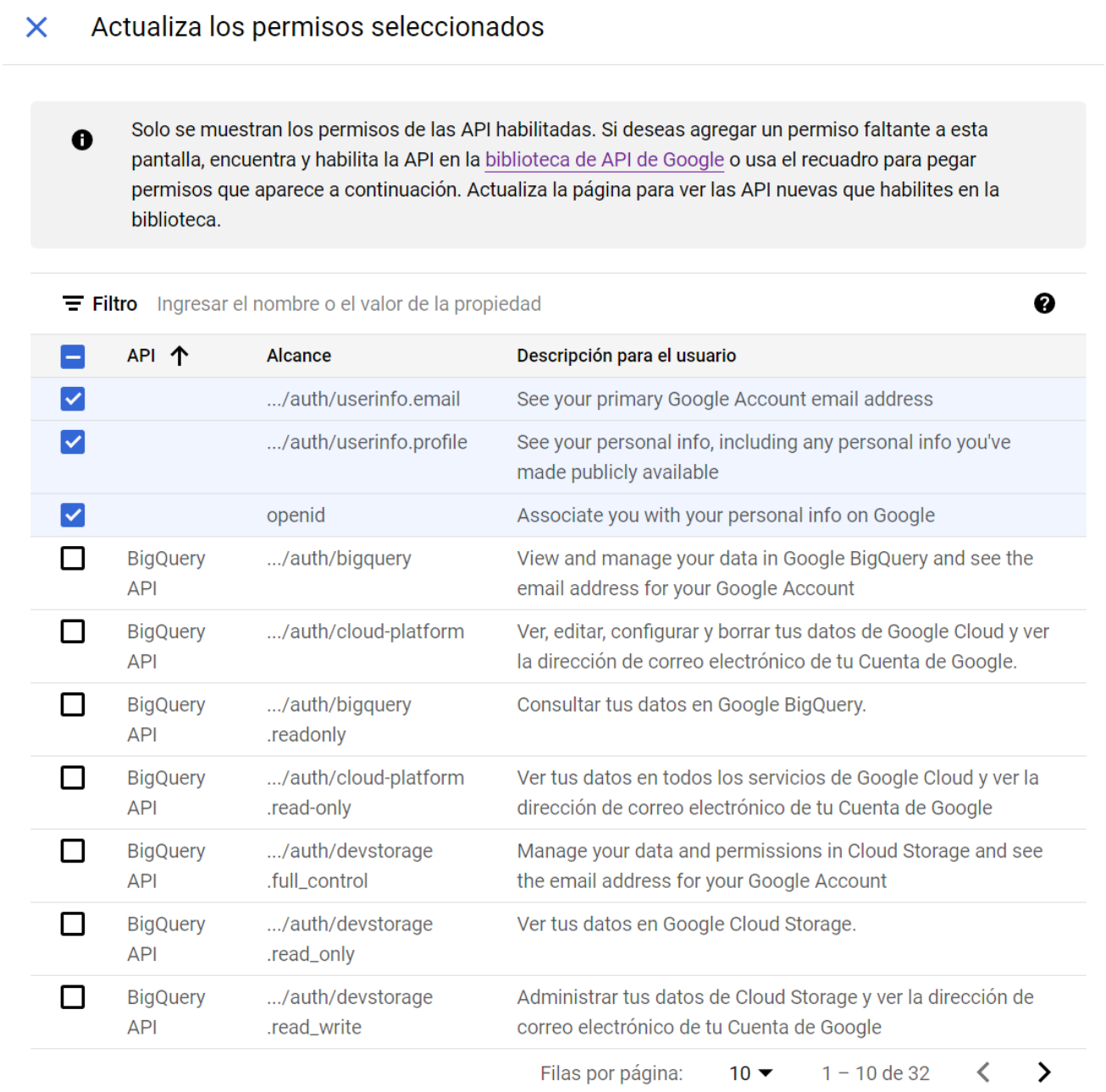


Ilustración 38 - Google Cloud Console, permisos habilitados para nuestra aplicación

Esto entre otras cosas nos permite obtener información básica del usuario, que podremos aprovechar, por ejemplo, para registrar al usuario automáticamente.