Universitat Oberta de Catalunya

uoc.edu

Cloud Docs Signature Platform

Eduardo Jesús Díaz Dioniz

Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones.

Sistemas de autenticación y autorización

Consultor: Juan Carlos Fernández Jara
Profesor: Victor Garcia Font

31/12/2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	Cloud Docs Signature Platform
Nombre del autor:	Eduardo Jesús Díaz Dioniz
Nombre del consultor/a:	Juan Carlos Fernández Jara
Nombre del PRA:	Victor Garcia Font
Fecha de entrega (mm/aaaa):	12/2018
Titulación:	Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área del Trabajo Final:	Sistemas de autenticación y autorización
Idioma del trabajo:	Español
Palabras clave	Identidad Digital Firma centralizada Seguridad Informática

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El contenido de este trabajo es el desarrollo de un aplicativo que permite la firma de documentos en la nube usando plataformas como Drive, Dropbox o TrustedX. Este aplicativo cumple con los estándares de firma de documentos en PDF ISO 32000-1 y además pretende adaptarse al nuevo funcionamiento introducido por el reglamento europeo eIDAS en materia de firma descentralizada en la nube.

La metodología usada ha sido fundamental para el desarrollo del trabajo, ya que se ha diseñado lo más modular posible, permitiéndonos usar distintos tipos de proveedores de almacenamientos y abriéndonos la puerta a desarrollar distintas interfaces o distintos proveedores de firma.

Con este desarrollo se espera poder continuar con el proyecto añadiendo nuevas funcionalidades y nuevos proveedores que amplíen su funcionalidad.

Abstract (in English, 250 words or less):

This work talk about the development of an application that permit the document signing at the cloud using platform such as Drive, Dropbox or TrustedX.
This applications is agree of standards for PDF documents signings (ISO 32000-1) and also try to adapt to the new operation introduced by the European regulation eIDAS in decentralized signature in the cloud.
The methodology used has been fundamental for the development of the work because has been designed as modular as possible allowing us to use different kind of storage providers and opens many doors to develop different interfaces or sign providers.
With this development we hope can continuous with the project adding a new features and new providers that upgrade us functionality

Índice

1. Introducción **7**
1.1 Contexto y justificación del Trabajo 7
1.2 Objetivos del Trabajo 8
1.3 Planificación del Trabajo 9
1.5 Breve sumario de productos obtenidos 11
1.6 Breve descripción de los otros capítulos. 11

2. Estado del arte. **14**
2.1 eIDAS. Sist. europeo de reconocimiento de identidades electrónicas. 14
2.2 oAuth. 15
2.3 Proveedores de almacenamiento 16
2.3 Proveedores de firma 17

3. Diseño. **17**
3.1 Arquitectura. 17
3.2 Enfoque del desarrollo. 18
3.3 Diagramas y comportamiento 20
3.4 Subsistemas. 23

4. Implementación **25**
4.1 Interfaces y navegación. 25
4.2 Carga Asíncrona. 28
4.3 Previsualización de objetos. 29
4.4 Descarga de ficheros. 30
4.5 Subida de identidades. 30
4.6 Gestión de las identidades. 31
4.7. Dropbox y control de las sesiones. (IdP) 33
4.7.1 Planificación. 33
4.7.2 Gestionar identidad Dropbox. 35
4.7.3 Tipo de proveedor de archivos. 37
4.7.4 Previsualización de Dropbox. 38
4.8 Drive (eIdP) 39
4.9 TrustedX (eSigP) 40

5. Conclusiones y líneas futuras. **42**
5.1 Conclusiones. 42
5.1 Trabajos futuros: 42

6. Glosario **44**

7. Bibliografía **45**

8. Anexos **46**

8.1 Anexo 1: Manual de usuario. 46

Agradecimientos

A Juan Carlos Fernández Jara, por el esfuerzo y dedicación, motivándome y asesorandome en el desarrollo de este trabajo.

A Davinia, mi eterna compañera de aventuras. Gracias por la paciencia y el apoyo incondicional.

A mi madre, me has enseñado tanto que no bastan mil hojas de agradecimientos.

Y agradecer también a los profesores de la UOC por su continuo apoyo. Ha sido una experiencia única que ha cambiado mi visión del mundo y la seguridad.

.

1. Introducción

1.1 Contexto y justificación del Trabajo

Actualmente los sistemas de firma electrónica de documentos llevan aparejados problemas técnicos y de seguridad. Ya sea por el tener que disponer de un equipo, un lector de tarjetas criptográficas o disponer del certificado en dispositivos “no seguros” con los riesgos que esto conlleva.

Hace algunos años entró en vigor el reglamento 910/2014 o reglamento eIDAS e introdujo una serie de novedades que nos han permitido avanzar en cuanto a las cuestiones de firma electrónica. En concreto, ha contemplado la posibilidad de disponer de la firma cualificada (firma electrónica reconocida) en la nube, en dispositivos seguros de creación que mantengan la seguridad. Esta novedad nos ha abierto una nueva veda donde desarrollar y es el principal impulso de nuestro TFM, gracias a esto podemos disponer de una firma cualificada y reconocida como la firma manuscrita en un sistema seguro desde el que podamos firmar los documentos que también tenemos en la nube con total confidencialidad sin nisiquiera tener que tener ningún documento o firma físicamente en nuestros dispositivos. Por motivos de simplicidad y recursos del proyecto no usaremos firmas cualificadas, pero será una vía de desarrollo futuro el hacer uso de estas firmas.

El valor principal del proyecto, a parte de ser una plataforma que permita facilitar la firma a los usuarios, es el uso de distintos proveedores. Es por ello que, será vital dada las características del proyecto, no imponer sistemas sino establecer múltiples alternativas que puedan ser configuradas tanto para la firma como para el almacenamiento. A pesar de que debido al alcance del proyecto, no se incluyan alternativas al sistema de firmado o todos los sistemas de almacenamiento deseados, la plataforma irá orientada a permitir integrar todo este tipo de soluciones.

Actualmente existen sistemas de firma tanto públicas como privadas y multitud de plataformas de almacenamiento en la nube. La propuesta es generar un sistema que interconecta estos proveedores y sobre todo que el resultado esté orientado a este nuevo reglamento.

1.2 Objetivos del Trabajo

El principal objetivo de este TFM es una aplicación funcional que permita realizar la firma de documentos usando los documentos almacenados en proveedores externos. A continuación detallamos varios objetivos fundamentales:

- Implantar un sistema de gestión de identidades y autorizaciones mediante oauth que nos permita identificarnos con servicios externos como Drive o Dropbox.

- Implantar un sistema que nos permita consumir los servicios de almacenamiento permitiéndonos descargar y subir ficheros así como navegar por las carpetas como si de un filesystem se tratara.

- Implementar un sistema de gestión de autorizaciones de firmado permitiéndonos tratar con proveedores de firma, en concreto, TrustedX, de modo que nos permita la gestión de las identidades de firma y la realización de la firma de los documentos.

- Implementar un sistema que permita tratar los documentos PDF de modo que otros subsistemas del aplicativo puedan obtener hashes y la composición de pdf firmados.

- Desarrollar una interfaz intuitiva para los usuarios, responsive además de ligera permitiendo su uso desde cualquier dispositivo a través de la plataforma web.

- Desarrollar un sistema que integre los subsistemas anteriores y permita al usuario interactuar con todos ellos.

1.3 Planificación del Trabajo

Para la planificación del trabajo usamos una esquematización simple compuesta principalmente por una primera fase de diseño, una segunda fase de desarrollo y una última fase de testing. En ellas se irá definiendo el proyecto hasta tener un entregable final que será el producto que se presentará para su evaluación.

Tarea	Subtareas	Estimación
1.Planificación	- Diagramas de flujo - Diagrama de casos de uso - Plan de despliegue	4 días

2. Desarrollar una interfaz web	- Navegación por las carpetas:
	  - Interfaz de carpetas
	  - Previsualizador de PDF.
	  - Interfaz de seleccionar firma.
	- Menús y submenús
	  - menú general
	  - cierre de la sesión
	- Ventana autenticación (index)
	- Ventana de proveedor de firma
	  - Listado de firmas
	  - Uploader de firma
	  - Confirmar eliminar firma
		6 días

3. Implementar módulo de autentificación.	- Gestión de la sesión:
	  - con Dropbox.
	  - con Google Drive.
		1 día

4. Implementar un sistema de autorización e identificación.	- Usar módulo de autenticación para obtener las credenciales. - Solicitar al usuario autorización para uso de documentos - Sistema que permita descargar y subir los documentos.	4 días
5. Implementar sistema de gestión de firmas	- Realizar la firma de un hash.	3 días
6. Implementar un gestor de composición de PDF.	- Realizar hash del documento. - Generar firma visual a partir de los datos del proveedor e incrustarlo en el pdf.	4 días

7. Sistema que integre los subsistemas.	- Interconectar todos los sistemas con la interfaz gráfica proveyendo una interfaz unificada.	4 días
8. Despliegue y testing.	- Despliegue final del aplicativo - Testing - Retoques finales.	4 días

Una vez tenemos estas tareas acotadas podemos definir los recursos que tenemos para terminar el proyecto en el tiempo límite. En concreto comenzamos en el inicio de la PEC 3 y para esta obtenemos un entregable de funcionalidades básico que entregar.

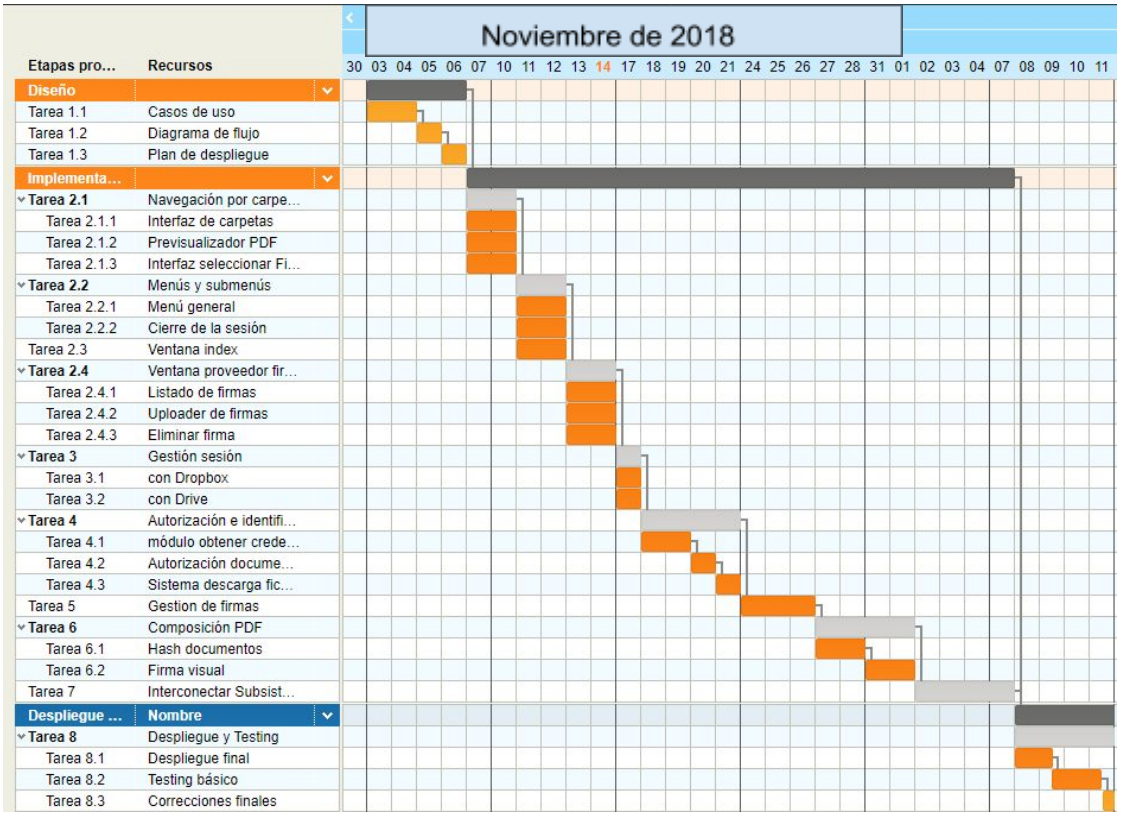

Figura 2. Planificación temporal del trabajo.

1.5 Breve sumario de productos obtenidos

Cloud Docs Sign Platform, tras completar el trabajo obtendremos una plataforma funcional que nos permitirá firmar documentos almacenados en plataformas de almacenamiento como Drive o Dropbox. Este proceso se realizará conectando con proveedores externos por lo que será capaz de tratar los PDF para poder firmarlos, componer la firma y volver a depositarlo en las plataformas de almacenamiento sin la necesidad de disponer de base de datos. Esta plataforma estará contenida en un Docker que tendrá toda la configuración necesaria para desplegar la plataforma.

1.6 Breve descripción de los otros capítulos.

La estructura que continúa la memoria del Trabajo de fin de Máster comienza realizando una pequeña introducción del estado actual, explicando las tecnologías y el sistema de proveedores externos. Posteriormente definimos la fase de diseño donde tenemos la arquitectura, el enfoque que le hemos dado al proyecto y los diagramas necesarios para entender el comportamiento y utilidad que le daremos. Y por último una fase de implementación donde detallamos todos los puntos que hemos ido desarrollando, las dificultades obtenidas y las soluciones.

Finalmente la parte común de bibliografía, conclusiones, líneas de trabajo futura y los anexos que dado su contenido y extensión no es posible su inclusión en el bloque del trabajo.

2. Estado del arte.

Actualmente nos encontramos con un panorama con recientes cambios muy profundos en los que normativas europeas han abierto una nueva vía de desarrollo de las tecnologías más comunes. A partir de este punto es necesario renovar la forma en la que se ofrecen servicios de firma en la nube y su forma de interactuar con las aplicaciones. En este proyecto nos apoyamos en servicios de firma electrónica en la nube gracias a TrustedX para ofrecer interconexión con otros servicios de internet, proporcionando funcionalidades adicionales a nuestra firma.

2.1 eIDAS. Sist. europeo de reconocimiento de identidades electrónicas.

El reglamento 910/2014 del Parlamento Europeo y del Consejo establece las bases en que los estados miembros reconocerán medios de identificación así como las condiciones y normas sobre servicios de confianza para firmas y documentos electrónicos bajo un marco jurídico común.

El proyecto permite el reconocimiento europeo de identidades electrónicas de modo que se acepten los certificados electrónicos de los países de origen en otros estados miembros de la unión europea. Este proyecto se conoce como eIDAS o electronic identification, authentification and trust services.

Uno de los puntos fuertes de esta medida es la posibilidad de la firma centralizada con certificados en la nube. De esta manera se pretende superar los problemas de uso que tienen los certificados comunes ya sea problemas técnicos, como de falta o incompatibilidad de drivers en los equipos así como de no disponer de lector de tarjeta criptográfica e incluso la inseguridad de disponer el certificado en dispositivos que puedan quedar obsoletos.

Gracias a esas medidas podemos disponer los certificados custodiados en la nube con fuertes medidas de seguridad de modo que se resuelven dos problemas graves, la custodia del certificado y los problemas técnicos por parte de los usuarios ya que podemos ofrecerles alternativas más adaptadas para que puedan usarlo sin problemas.

Los datos de creación de firma electrónica con un alto nivel de confianza podrán considerarse como firma electrónica cualificada según los términos del reglamento eIDAS lo cual nos abre un nuevo horizonte de

posibilidades que permita disponer de nuestros certificados desde un sitio seguro e interoperable.

En nuestro proyecto usaremos ese potencial para darle a nuestra plataforma la versatilidad de firma en la nube que nos provee TrustedX abriendo camino hacia la implementación de un sistema completo amparado legalmente bajo este nuevo reglamento.

2.2 oAuth.

oAuth o Open Authorization es un estándar que permite el intercambio de información entre aplicaciones a través de autorizaciones seguras de modo que, podamos pasar información entre varios sitios sin compartir la identidad pudiendo consumir servicios en un sitio con la identificación de otro.

Actualmente este método de intercambio se usa mucho cuando nos identificamos en muchas páginas webs. En concreto podemos ver como en sitios como Spotify nos permite consumir el servicio que ofrecen sin necesidad de identificarnos ya que nos permite identificarnos en facebook. Este sistema se basa en la confianza de los sitios ya que el sitio donde se consumen los servicios debe confiar en que la plataforma donde se proporciona la identidad cumple con un criterio seguro para obtener la identidad y asegurar la veracidad de los datos.

Usaremos esta tecnología en nuestro proyecto para consumir el servicio de los proveedores que disponemos como TrustedX, Drive o Dropbox.

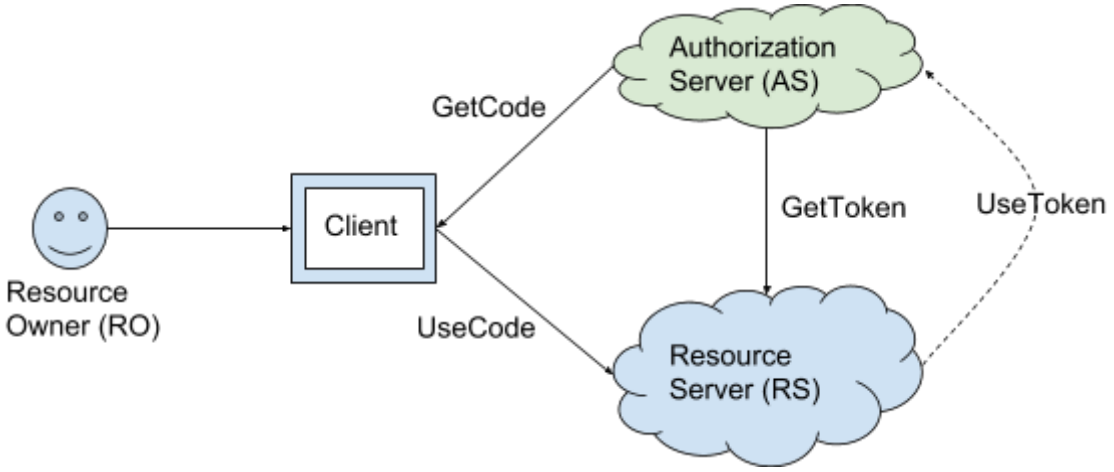

Figura 3. Descripción del oAuth.

2.3 Proveedores de almacenamiento

En nuestro proyecto definimos los proveedores externos como las entidades o plataformas externas a nuestro proyecto, en este caso disponemos de proveedores destinados a la identificación y almacenamiento de documentos. Estas entidades externas permiten por sus características compartir funcionalidades mediante APIs. En concreto, los proveedores de almacenamiento no solo disponen de soporte para guardar los archivos sino que son capaces de poder identificar a los usuarios, por ello, es algo más complejo ya que una entidad externa valida un usuario y sin compartir los datos de este un tercero debe tener la suficiente confianza en el servicio como para dejar usar sus servicios, por lo que, es ahí donde entra la tecnología de oauth, que nos permite dar la confianza necesaria para la identificación y el consumo de recursos de una forma segura.

En concreto los proveedores que disponemos nos permiten hacer peticiones REST para consumir los servicios, previo obtener un token, y devuelven estructuras en formato JSON con el contenido de la respuesta.

HTTP request

```
GET https://www.googleapis.com/drive/v2/apps/appId
```

Figura 4. Petición REST

```
{
  "kind": "drive#app",
  "id": string,
  "name": string,
  "objectType": string,
  "shortDescription": string,
  "longDescription": string,
  "supportsCreate": boolean,
```

Figura 5. Respuesta en formato JSON.

2.3 Proveedores de firma

Además de los proveedores de almacenamiento, en este proyecto contamos con otro tipo de proveedor que es el de firma. Este proveedor se comporta del mismo modo que el de almacenamiento para el intercambio de mensaje pero sus funcionalidades son permitirnos realizar firmas sobre los hash que le pasamos además de gestionar las identidades de firma que disponemos, permitiéndonos almacenarlas y usarlas desde la nube.

Para estas funcionalidades usamos TrustedX, el cual es un completo sistema de firma y nos provee de las funcionalidades necesarias por medio del consumo de una API REST. Este sistema cumple con la normativa eIDAS pudiendo gestionar firmas cualificadas, pero en nuestro proyecto únicamente usaremos firmas electrónicas con la finalidad de simplificar el desarrollo. Este punto es importante ya que intentaremos realizar la menor diferencia posible para que quede como vía de desarrollo futura el terminar de cumplir con los requisitos legales para la firma cualificada.

3. Diseño.

3.1 Arquitectura.

Para la implantación del sistema se escoge realizarlo sobre docker el cual nos provee una forma sencilla de tener desplegado aplicaciones software permitiendo tener toda la configuración en una caja autocontenida que nos permita lanzar el aplicativo en el momento que se requiera. Para el lenguaje a desarrollar escogimos J2EE el cual es un lenguaje versátil basado en Java que nos provee de un aplicativo web con todas las librerías de Java. Esta decisión viene fundada en las librerías seleccionadas para la manipulación de los PDF y de la firma electrónica. Podremos generar un WAR con el aplicativo colocar la configuración necesaria dentro del docker y poder guardarlo y desplegarlo sin problema.

Para el desarrollo nos basamos en un JDK 8 que una de las últimas versiones de Java LTS. Las versiones LTS o las Long-Term Support son versiones que nos ofrecen un soporte mucho más duradero que las otras versiones. Para el servidor que desplegará el WAR usaremos Tomcat 7 la cual es una versión estable con soporte y actualizaciones de seguridad.

En la infraestructura usaremos un modelo IAAS que nos permita centrarnos únicamente en el aplicativo sin la necesidad de trabajar sobre la plataforma. Elegimos Amazon EC2 para crear una imagen de Ubuntu con contenedores Dockers de modo que nos permita lanzar nuestro contenedor con el aplicativo y su configuración así como el servidor de aplicaciones y el JDK.

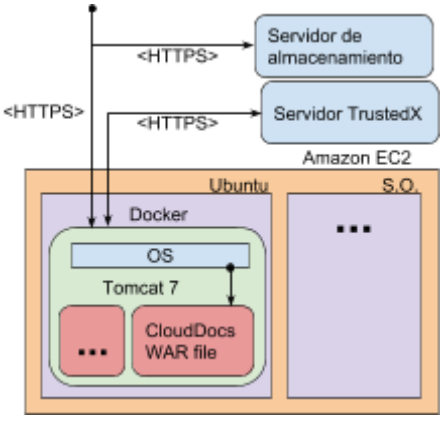

Figura 4. Diagrama de despliegue

3.2 Enfoque del desarrollo.

Existen varios enfoques posibles para el desarrollo del TFM. Inicialmente planteamos una idea de solución unificada donde el subsistema de firma fuera capaz de manipular el pdf así como navegar por las carpetas. Con esta idea, conseguimos poder usar muchas funcionalidades peculiares de cada sistema ya que, en el fondo, los proveedores de firma tienen distintas opciones como por ejemplo tipos de metadatos, etiquetas, etc que nos permite ver como queremos los ficheros cuando naveguemos por nuestro drive. La desventaja de este sistema es que en el fondo tantas peculiaridades hacen incompatible que se implemente de manera unificada más de un proveedor.

Por ello y dado a la necesidad de poder usar diferentes proveedores de almacenamiento planeamos separar en varios subsistemas que sean comunes como por ejemplo, usar únicamente llamadas comunes y básicas como listar los directorios o descargar ficheros, etc. Al usar esta estrategia podemos definir un concepto único de proveedor que se pueda implementar rápidamente y que todo lo que siga esa estructura pueda ser ejecutado por nuestro aplicativo.

Una vez planificamos la forma en la que se tratará la información nos queda el enfoque de cómo se comunicará con el cliente. ¿Escogemos una app, una web, cliente pesado? Para ello nos planteamos que las mejores opciones es un cliente web que nos permita usar el aplicativo en cualquier terminal que permita web de este modo no dependemos únicamente de un terminal móvil o pc.

Dada la estratégia de subsistemas obtenemos el siguiente esquema:

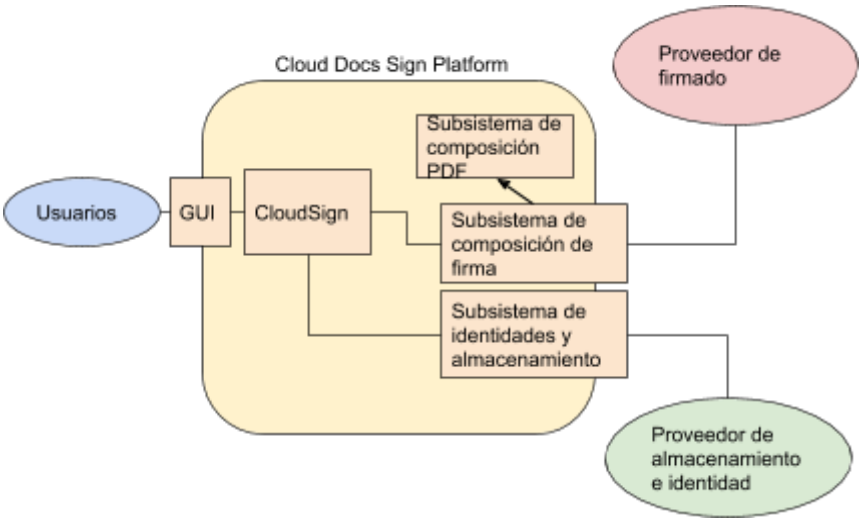

Figura 1. Esquema de subsistemas

El subsistema de composición de PDF: Se encarga de crear los hash para firmar y componer el documento pdf visualmente

El subsistema de composición de firma: Se encarga de interactuar con el proveedor de firma para realizar la firma de un hash y para gestionar las identidades de firma. Además de eso trabajará con el gestor de PDF para poder obtener los hash y pasarle los datos que permitan generar el PDF firmado y con una imagen incrustada.

El subsistema de identidades y almacenamiento: Será el sistema que interactúe con los proveedores de almacenamiento e identidad (Drive y Dropbox) nos acredita la identidad de los usuarios y nos otorgan autorización para el uso de los documentos almacenados.

Y por último CloudSign se encarga de componer lo que le otorgan los tres subsistemas, recuperar los documentos y entregarlos al sistema de firma para que cree la firma y devolver el documento firmado al documento de almacenamiento.

3.3 Diagramas y comportamiento

Comenzamos el proyecto detectando las necesidades que queremos solventar. En concreto, vamos a permitir a los usuarios poder firmar documentos en la nube por lo que es necesario la parte de autentificación del usuario (iniciar y cerrar sesión). También será necesario la gestión básica de las identidades de firma como será la de eliminar y añadir identidad. Continuando el flujo normal de ejecución deberíamos ver las carpetas y poder navegar por ellas hasta encontrar los documentos que queremos. Una vez divisamos el documentos, se deberá poder mostrarnos una previsualización para tener conocimiento del documento que vamos a firmar y permitirnos firmarlo. Como condición extra, para la firma deberemos seleccionar con qué identidad vamos a proceder a la firma. Una vez hemos firmado el documento podremos descargarlos para poder usarlos dando el proceso completo. A partir de todos este flujo de funcionamiento de la aplicación podemos especificar los casos de uso que son necesarios para el aplicativo.

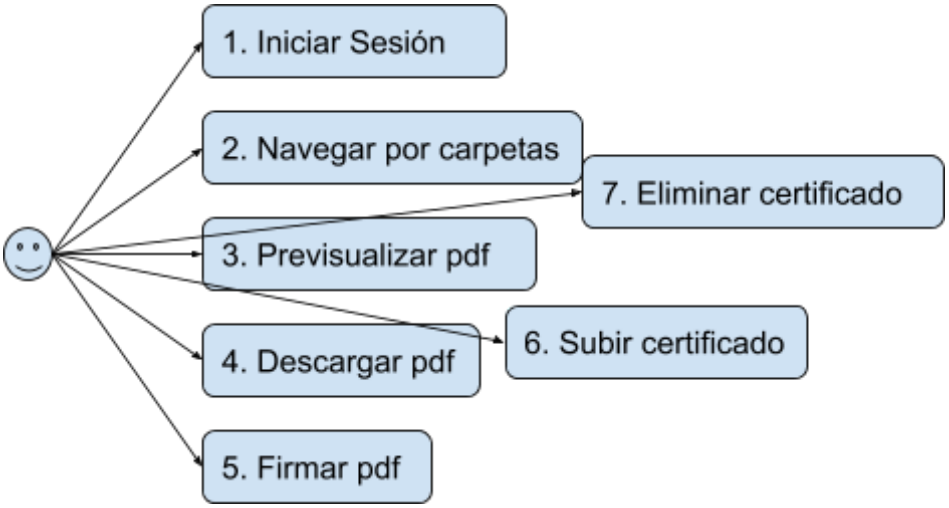

Figura 5. Diagrama casos de uso.

1. Los usuarios se podrán identificar en la plataforma con su cuenta de Drive o Dropbox.

2. Se permite navegar por las carpetas para facilitar la búsqueda de los documentos y permitir la misma organización que los usuarios tengan en sus respectivas cuentas.

3. Para mejorar la selección de los pdf's los usuarios podrán pre-visualizar tanto los pdf que tienen como los firmados de modo que también puedan verificar que se firman visualmente.

4. Los pdf se podrán descargar dándole la mayor funcionalidad al proceso de firma.

5. Como principal funcionalidad los documentos pdf se podrán firmar con los certificados en la nube.

6. Para la firma el usuario podrá subir el certificado a la plataforma de firma TrustedX.

7. Además se les permitirá la gestión de los certificados permitiendo borrarlos.

Es necesario detallar el flujo que se producirá el aplicativo para la obtención de una firma o la comunicación con los proveedores de almacenamiento ya que esta nos marcará las pautas para desarrollar nuestro proyecto y la forma de implantar las tecnologías usadas, sobre todo el consumo de los recursos por medio de oauth.

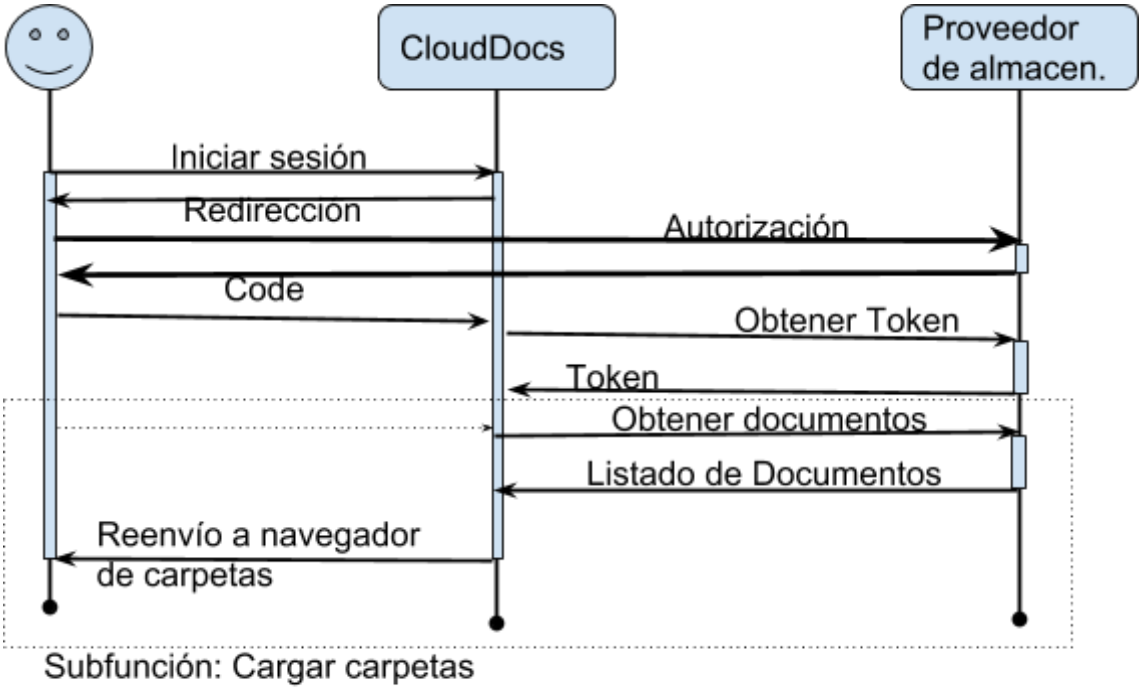

Figura 6. Diagrama de inicio de sesión.

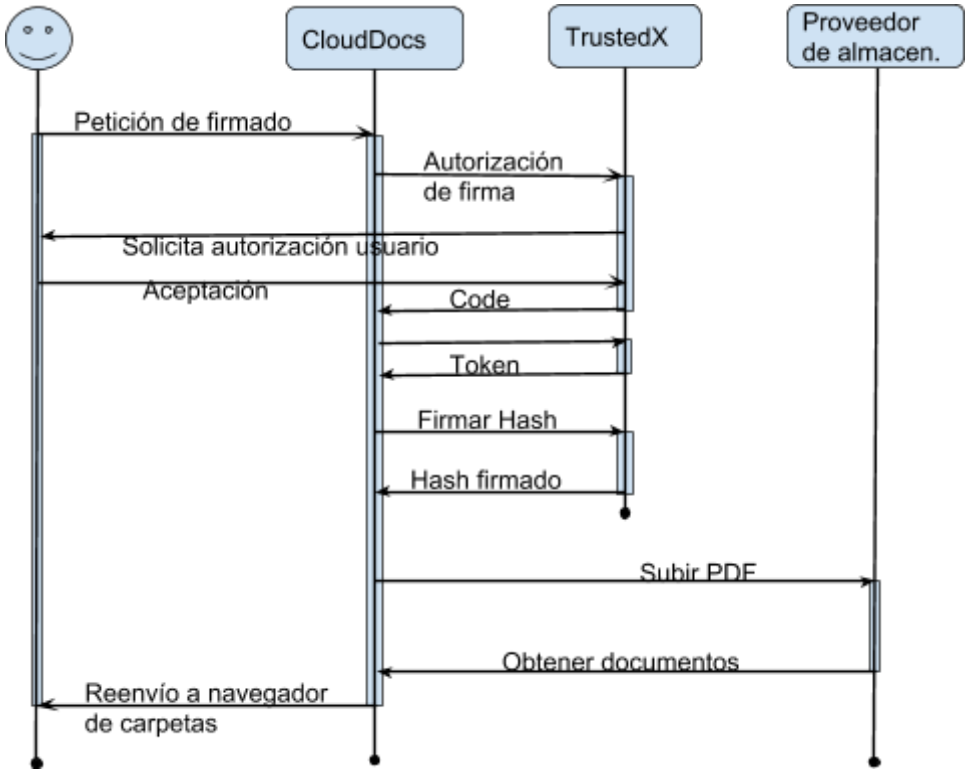

Figura 7. Diagrama de firmado de pdf.

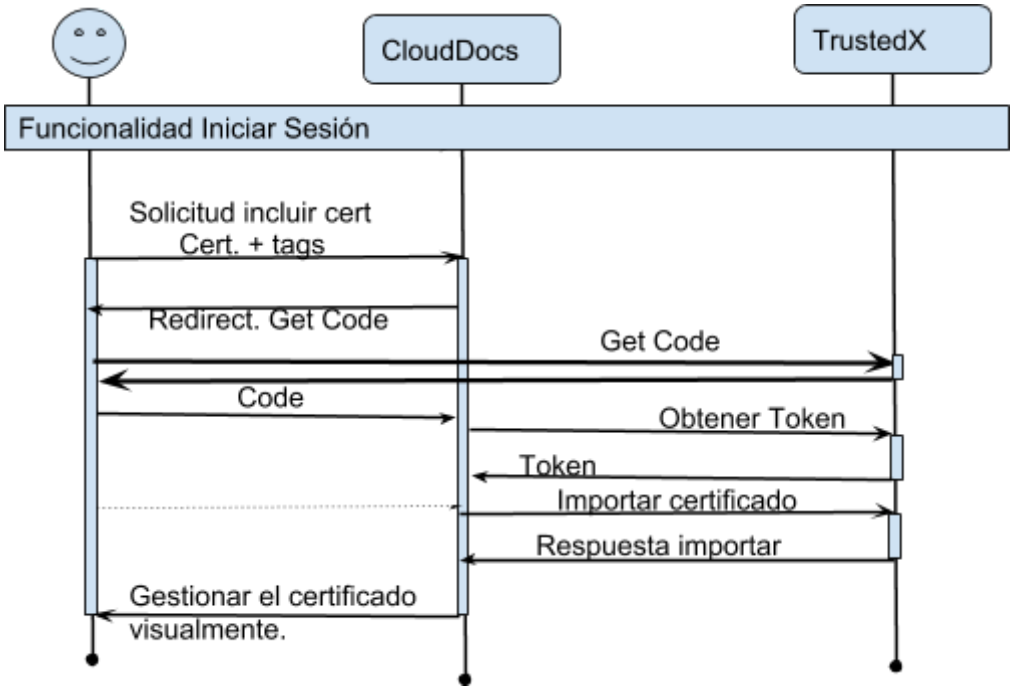

Figura 8. Diagrama de incluir certificado.

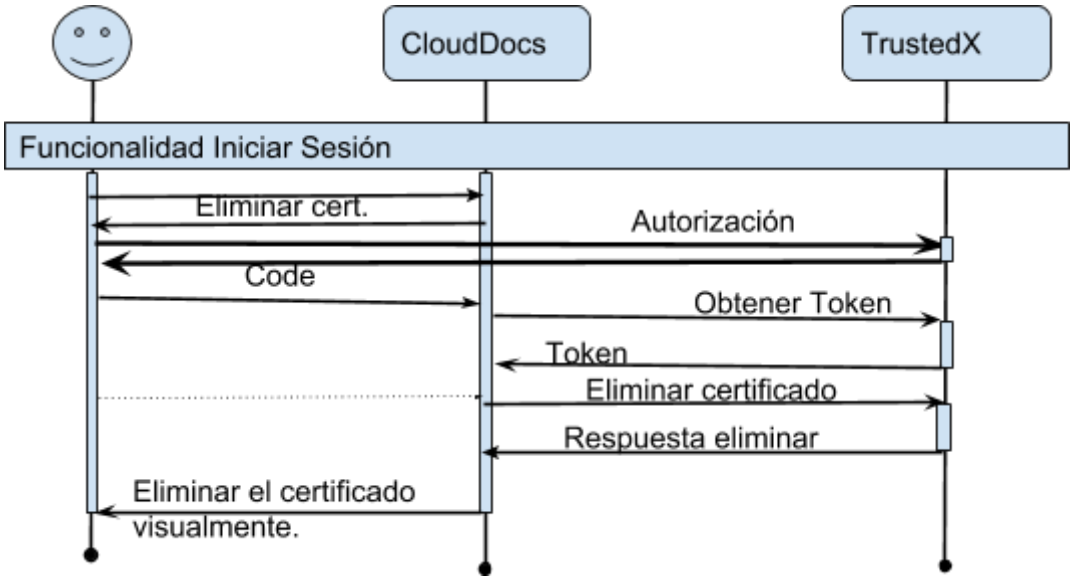

Figura 9. Diagrama de eliminar certificado.

3.4 Subsistemas.

Existen varios subsistemas críticos que son los que se encargan del funcionamiento del aplicativo, a continuación mostramos un esquema completo.

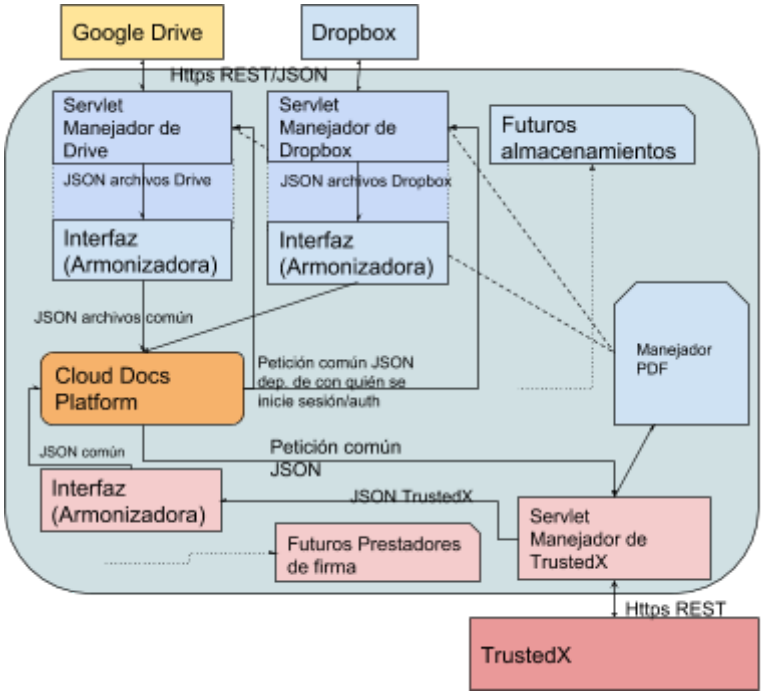

Figura 10. Diagrama de los subsistemas.

Una idea para desarrollo futuros es implementar nuevos proveedores de firma y de almacenamiento por lo que, es vital el diseño de subsistemas independientes que se usen entre sí para poder añadir nuevas funcionalidades sin tener que modificar el esquema principal del aplicativo. Es por esto, que definimos la aplicación en forma de subsistemas que se comunican por medio de JSON en un lenguaje común que nos permite poder añadir un nuevo proveedor. Por ejemplo, podemos ver como Google Drive nos proporciona una respuesta JSON con mucha información pero a su vez es completamente diferente de Dropbox, así que, usaremos una "Interfaz" o una traducción entre lo que nos dan los proveedores a lo que le pasamos a nuestro sistema. Es por esto que hablamos de subsistemas, un proveedor de almacenamiento necesitará una serie de requisitos para que sea funcional en nuestra aplicación y que devuelva una salida concreta.

Al mandar a firmar algún documento se realiza una petición al servlet de firma y esta será una petición, tal como hemos indicado anteriormente, JSON que incluya la forma de obtener el documento, la firma que se desea usar y donde dejar el documento finalizado. Este tipo de peticiones deben ser estándares y deben poderse intercambiar por otro subsistema de firmado que tenga los mismos requisitos de entrada y que devuelva el mismo patrón de firma.

4. Implementación

4.1 Interfaces y navegación.

Comenzamos la implementación desarrollando las interfaces y ventanas que serán la forma de interactuar con la aplicación.
Comenzando diseñando las ventanas de index o la página principal, para esta labor usamos Framework de bootstrap y en este framework existen múltiples plantillas que podemos adaptar a nuestro gusto. En concreto escogemos la base Cover para el diseño de nuestra página de inicio.
(https://getbootstrap.com/docs/4.1/examples/cover/)

La plantilla no contiene todo lo que deseamos pero simplemente es para usar la estructura simple y poder no tener nada más en la página sin que quede mal estéticamente.

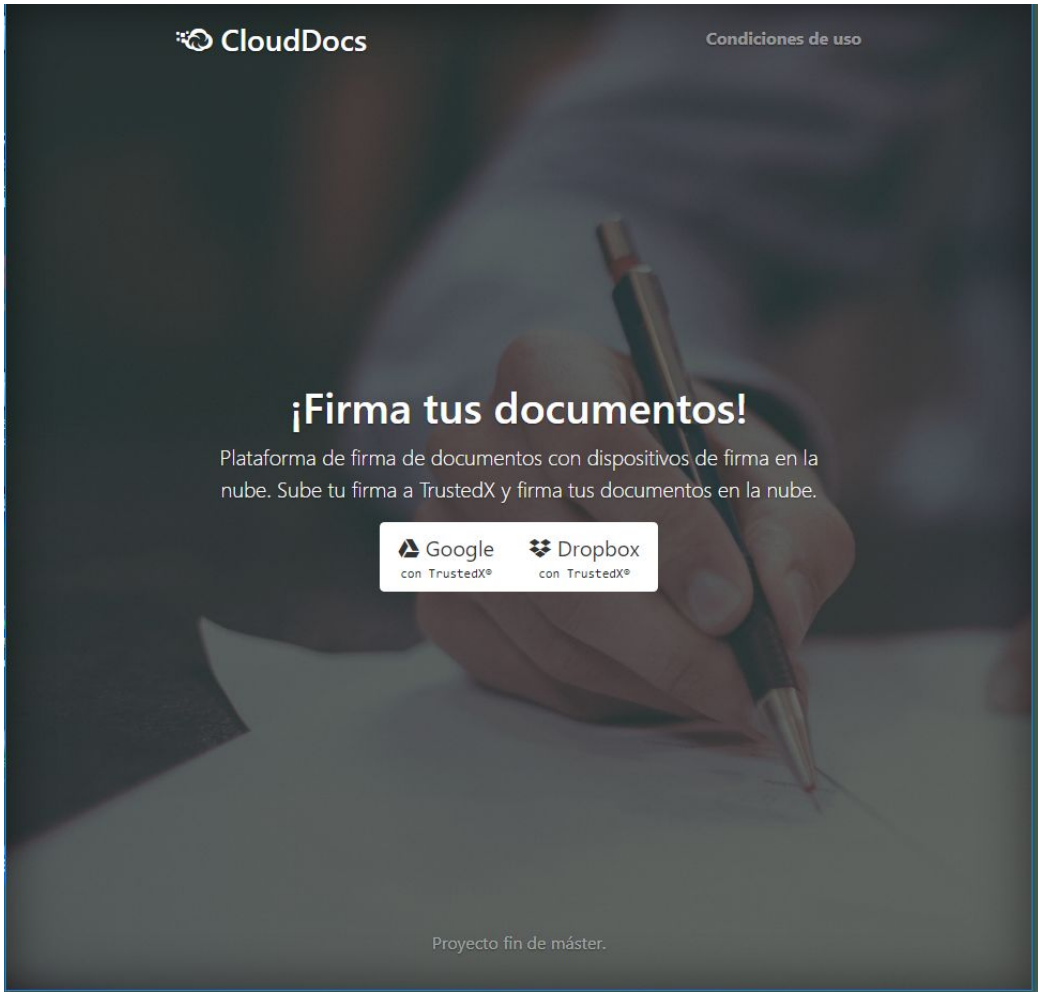

Figura 11. Pantalla de inicio

Posteriormente continuamos desarrollando la pantalla maestra de nuestro proyecto donde se navegará, se previsualiza y mandará a firmar los PDF. Esta página no tiene código embebido en el JSP carga el contenido de forma asíncrona de modo que permite una carga más transitiva sin esperar por los prestadores externos. Este planteamiento nos produce un problema en cuanto a la implementación ya que hay que desarrollar dos sistemas que trabajen de forma autónoma y que se comuniquen por JSON, en este caso nuestro aplicativo y la interfaz de usuario.

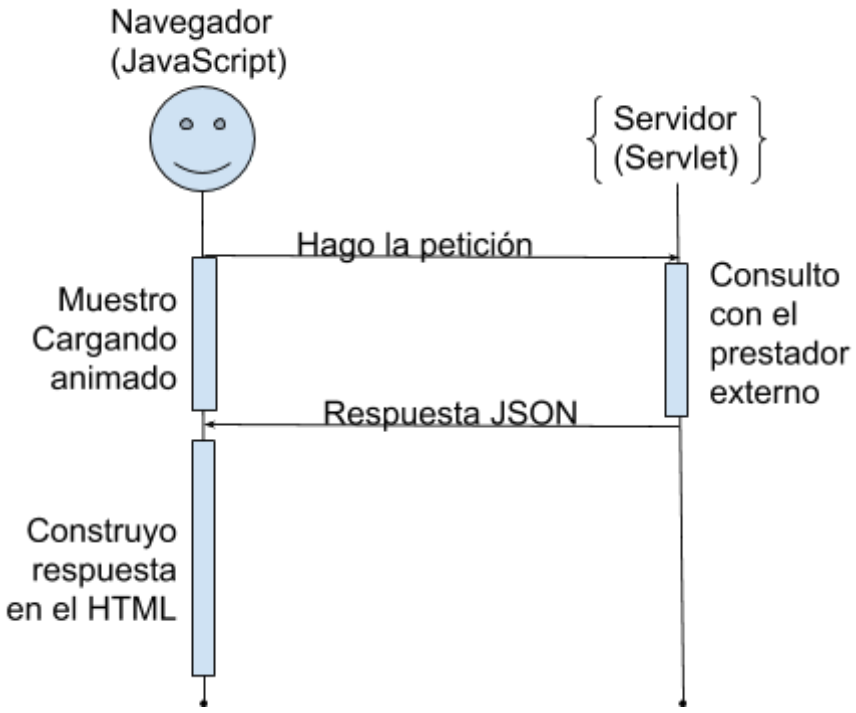

Figura 12. Comportamiento Interfaz cliente.

Para tratar los datos que se intercambian es necesario tratar un estándar tal como hemos comentado en la fase de diseño. Para esto elaboramos la respuesta común que tendrá nuestros servlet de almacenamiento.


```
▼{path: "", parent: [],…}
   ccd: "200"
  ▶data: [{route: "1N3WDiyYAZdruChJ-uCbHbF1PXOQTLUUJ", id: "1N3WDiyYAZdruChJ-uCbHbF1PXOQTLUUJ",…},…]
   msj: "OK"
   origin: "drive"
   parent: []
   path: ""
```


Figura 13. Respuesta JSON de Almacenamiento.

En esa respuesta tenemos el código CCD que es idéntico a la codificación HTTP, en la cual obtenemos código 200 para peticiones correctas, 403 para los forbidden y 400 para las peticiones mal formadas. A continuación le sigue data que contendrá la información que se está pasando en la petición, en este caso es una petición de GET de los directorios por lo que le contenido será la información de directorios que tiene la siguiente estructura:

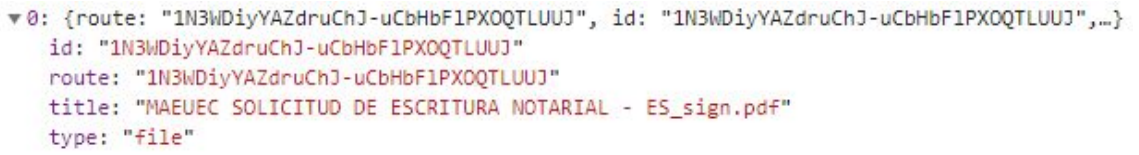

Figura 14. Petición JSON, listado de ficheros.

En este detalle tenemos el identificador del objeto, la ruta donde podemos acceder a él, el título para mostrar y el tipo de objeto que es que puede ser directorio o ficheros en este caso. Para la ruta de acceso cada proveedor puede tener diferentes formas de acceso y es por ello que hay que separarlo del identificador del objeto. En dropbox se navega por directorios como en el filesystem y Drive usa identificadores para cada objetos.

Continuando con los datos de la petición general tenemos un msj que nos de un mensaje legible del estado de la petición como puede ser “OK” o “No se pudo recuperar los ficheros”. Sigue el parent que es para poder recomponer la barra de navegación. A pesar de recuperarlo por primera vez puede tener padres el directorio en el que estamos y es por eso que lo contemplamos ahí. Por ejemplo, tras firmar un fichero se produce un callback que vuelve al mismo directorio, sin esta opción, no podríamos saber si estamos en el raíz o por el contrario estamos en un directorio concreto. Por último el path que se está consultando para los proveedores de almacenamiento que necesiten llevar ese registro (opcional) en el caso de que esté vacío o con root será el directorio raíz del aplicativo.

4.2 Carga Asíncrona.

[Problema]

En concreto, para Drive detectamos tiempos de respuestas muy altos para el aplicativo y esto se debía a que debíamos realizar una segunda consulta que complete los datos que necesitamos.

Name	Status	Type	Initiator	Size	Time
filedrive...	200	xhr	jquery.min.js:2	1.6 KB	3.32 s
filedrive...	200	xhr	jquery.min.js:2	181 B	1.50 s
filedrive...	200	xhr	jquery.min.js:2	845 B	2.60 s

Figura 15. Tiempos de respuesta del aplicativo.

Como se puede ver en la figura anterior los tiempos de respuesta eran en torno a los dos segundos y eso provocaba que la web quedará bloqueada dando un feedback negativo como si la web dejara de funcionar.

Con la carga asíncrona solventamos este problema pudiendo plasmar una carga visual que permite a los usuarios estar tranquilos de que la web se encuentra funcionando.

Figura 16. Muestra de carga asíncrona.

4.3 Previsualización de objetos.

Si el objeto que se consulta en la navegación es un tipo objeto concreto se provoca una preview que nos devuelve una salida estándar. Si el objeto genera una preview la salida será la siguiente.

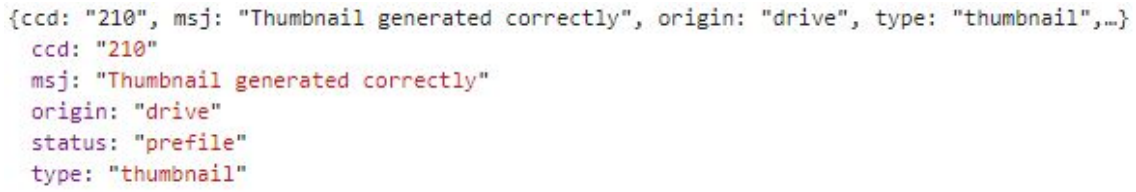

Figura 17. JSON Thumbnail generado.

Y si el objeto no puede generar un thumbnail volcará la siguiente salida.

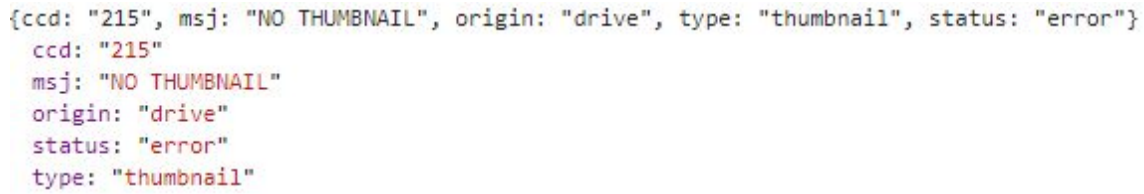

Figura 18. JSON error generando thumbnail.

Cada proveedor de almacenamiento es el que genera el thumbnail por lo que será el que decida lo que puede o no procesar. Por ejemplo, existen varios formatos a parte del pdf que son capaces de generar, como los docx.

Esta respuesta nos indica el estado, cuando es “prefile” significa que dentro del servidor existe una ruta con el código único que se genera cada inicio de sesión y que tras generar el thumbnail existirá una imagen que contiene lo que buscamos.

```
background-image:url("/prefile?1544983469445");
```

4.4 Descarga de ficheros.

Para la descarga de ese fichero usamos un servlet denominado prefile que permite esa descarga protegiéndonos de ataques y controlando que solo es consultable lo generado.

En esta ventana de preview permitimos la descarga del documento y en el caso de que sea pdf permitimos su firma. Para la descarga, usamos el link que nos provee el proveedor de almacenamiento al consultar por el identificador de los ficheros, a la hora de realizar la firma únicamente redirigimos al servlet de firma que se encarga de descargar el fichero.

Una vez tenemos este paso de la interfaz, procedemos a ver las gestiones de identidades de firma. El funcionamiento será similar, de forma asíncrona cargaremos las identidades, listamos las que tenemos y borraremos. Al cargar la página se realiza la siguiente consulta:

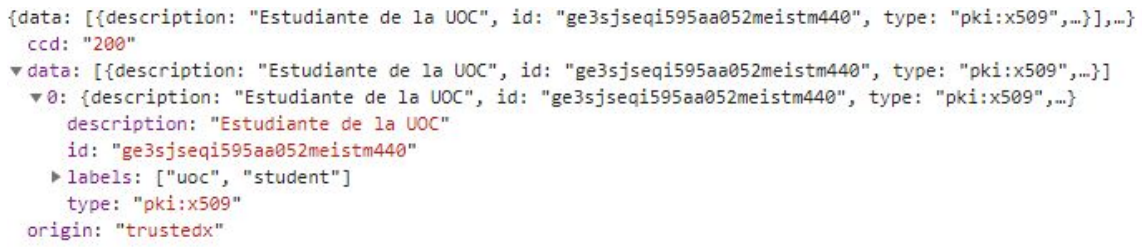

Figura 19. JSON de listado de identidades de firma.

4.5 Subida de identidades.

Para la subida de identidades realizamos una petición post que contiene la siguiente información.

```
pkcs12:
MIIYjg....r8OIj7GQICBAA=
labels:
["uoc","student"]
password:
trustedx
```

Para ello es necesario subir el contenido del fichero pkcs12 por lo que vía javascript debemos leer el contenido y adjuntarlo a la petición de subida, para ello realizamos una carga directa.

```
function fileselect(e) {
...
var files = e.target.files;
...
reader.readAsBinaryString(f);
```

Para hacer el borrado, siguiendo la misma dinámica realizamos una petición a la url pasándole como parámetros las acciones a realizar.

```
https://uoc.safelayer.com:2080/sign?a=del&id=ge3sjseqi595aa052meistm440
```

Y recibimos la respuesta estándar para indicar que el proceso se ha realizado correctamente.

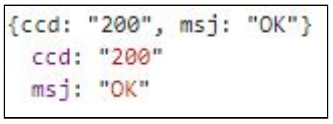

Figura 20. JSON correcto genérico.

4.6 Gestión de las identidades.

Para gestionar donde realizará las peticiones de almacenamiento y firma, el sistema almacenará datos en la sesión sobre el token y variables que nos apuntan al proveedor de almacenamiento que se está usando.

Nuestra interfaz realizará peticiones estándar a los servlet de almacenamiento y de firma por lo que estos deben estar adaptados para poder dar respuesta. A continuación detallamos un listado de los tipos de operaciones disponibles que deben cumplir los proveedores de almacenamiento.

Operación	**URL**
Obtener datos de un fichero o directorio	**[Servlet]**?a=nav&path=**[Ruta]** Petición: **GET** Parámetros: **a = acción a ejecutar.** **path = ruta.**
Descargar un fichero al navegador del cliente.	**[Servlet]**?a=download&path=**[Ruta]** Petición: **GET** Parámetros:

	a = acción a ejecutar. **path = ruta.**
Descargar un fichero a la caché local para operar con él	**[Servlet]**?a=downloadToSign&path=**[Ruta]** Petición: **GET** Parámetros: **a = acción a ejecutar.** **path = ruta.**

Y para los servlets de firma las operaciones necesarias que deben implementar son las siguientes.

Operación	**URL**
Listar las identidades de firma disponibles.	**[Servlet]**?a=list Petición: **GET** Parámetros: sin parámetros.
Obtiene respuesta sobre la eliminación de una identidad de firma.	**[Servlet]**?a=del&id=**[ID]** Petición: **GET** Parámetros: **id = Identidad a eliminar.**
Envía la acción para que se firme un documento predescargado.	**[Servlet]**?a=clientsign&callback=**[callback]**&id=**[ID]**&fileid=**[Ruta]**&filename=**[Nombre]** Petición: **GET** Parámetros: **callback = URL donde retornar al cliente, se usa para volver a una carpeta concreta donde estaba navegando.** **ID = Identidad de firma.** **Ruta = Donde encontrar el fichero** **Nombre = Nombre del fichero generado**

A partir de este punto tenemos diseñado el comportamiento de las interfaces por lo que podemos comenzar a desarrollar los servlet que controlan la lógica de negocio del aplicativo.

4.7. Dropbox y control de las sesiones. (IdP)

4.7.1 Planificación.

Para el desarrollo de los prestadores de almacenamiento y proveedor de identidad es primordial comenzar por generar la aplicación desde la consola que nos ofrezcan, en este caso la de dropbox (https://www.dropbox.com/developers/apps)

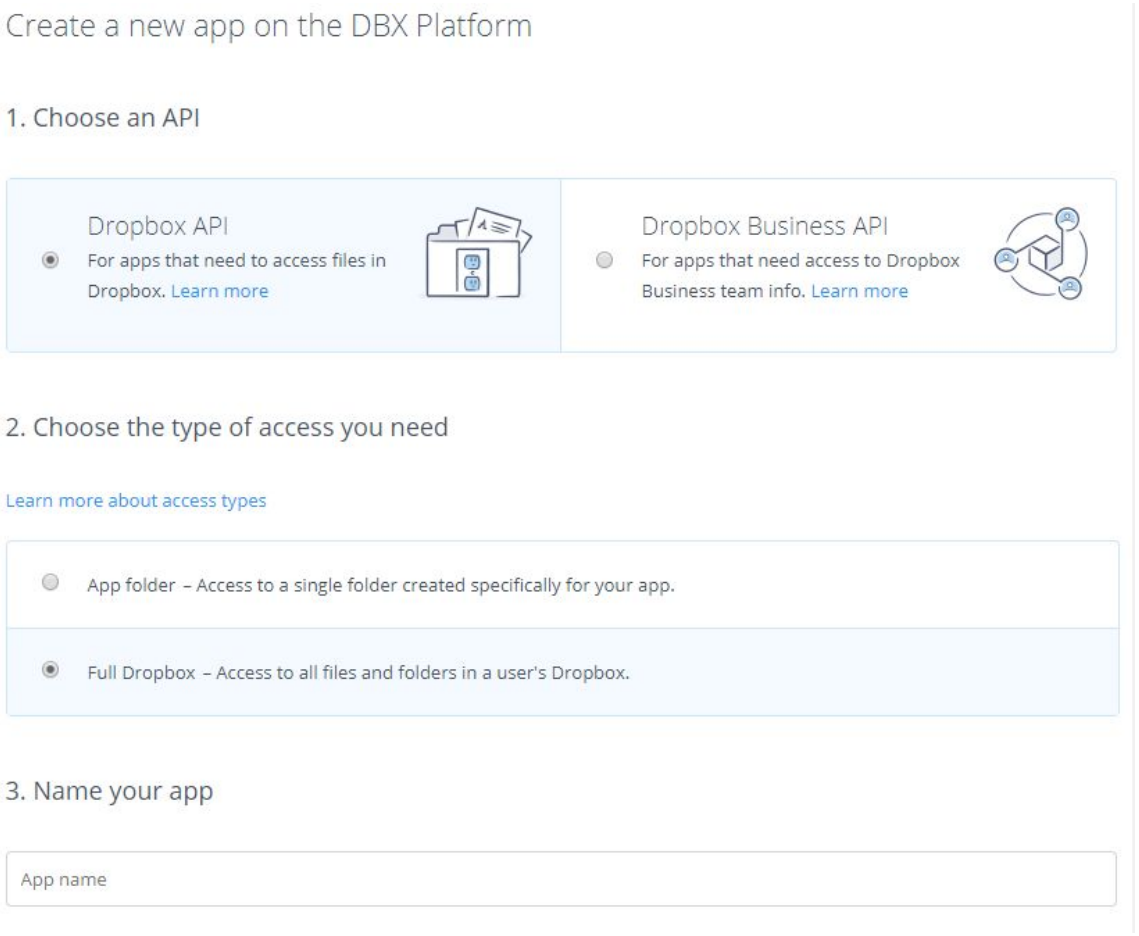

Figura 21. Consola Dropbox para crear aplicación.

Una vez creada nos mostrará una ventana donde tenemos nuestras aplicaciones y podremos realizar las configuraciones básicas, administrar o incluso borrarla.

Este punto es importante para autorizar las url de callback que son a dónde se va a retornar a los usuarios tras validarse en en dropbox. Si este callback no existiera podría existir robo de datos ya que el usuario sale de nuestro aplicativo y es dropbox el que lo dirige de vuelta.

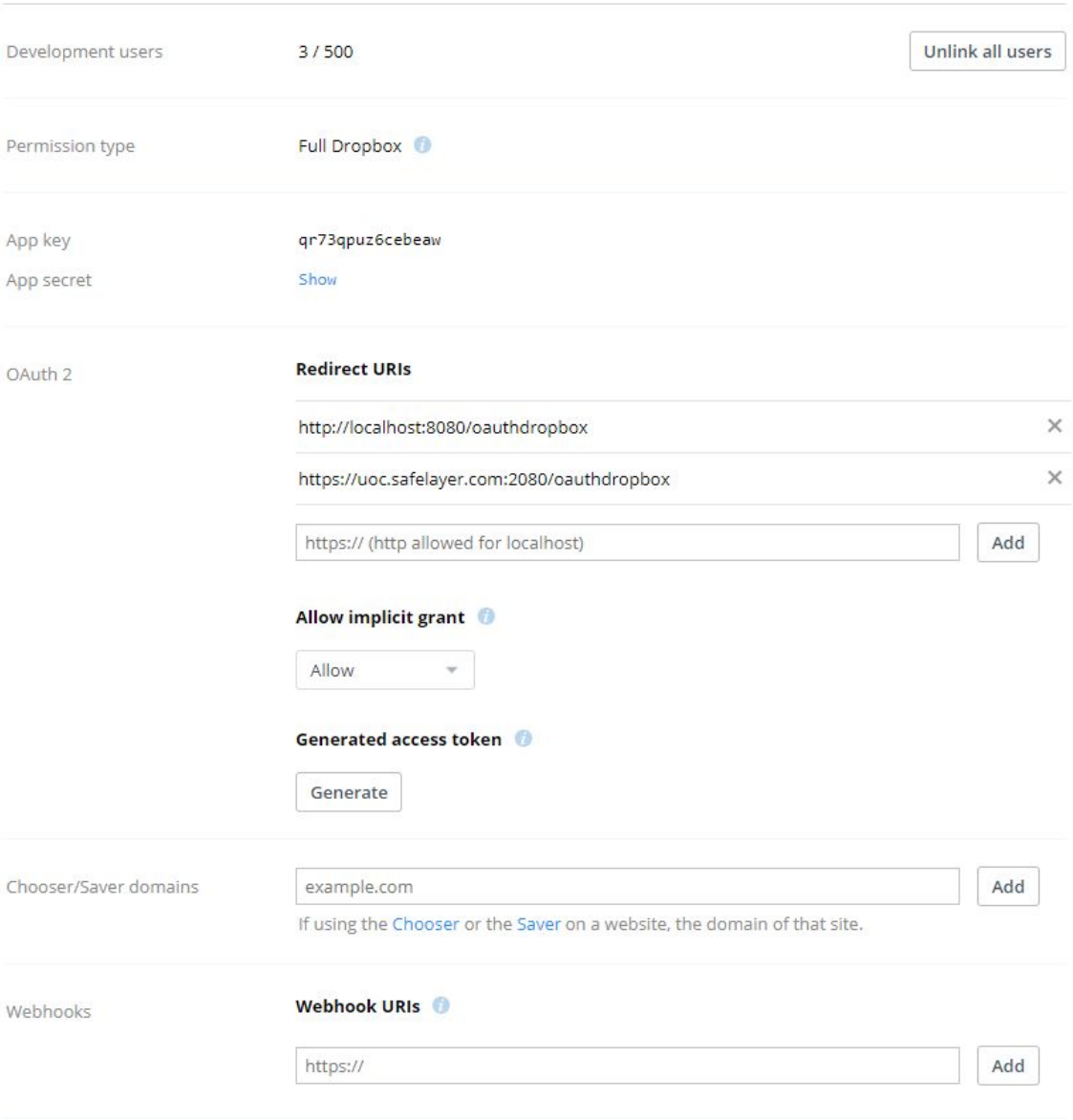

Figura 22. Pantalla de gestión de Dropbox

[Problema]

Es importante a la hora de confeccionar las credenciales, permitir la aplicación a varias personas, por ello hay que indicar el número de desarrolladores para que no de errores de acceso, además para que las credenciales pasen a un estado de producción requiere tener términos y condiciones de uso y ser necesario algo de tiempo hasta que el equipo de Dropbox valide la aplicación. Estos punto hay que tenerlos en cuenta para el desarrollo ya que puede afectar a la planificación temporal y ocasionar inconvenientes.

Una vez tenemos las credenciales del aplicativo es necesario buscar en la API el funcionamiento, para este caso usamos las librerías que recomienda Dropbox.

```
<dependency>
	<groupId>com.dropbox.core</groupId>
	<artifactId>dropbox-core-sdk</artifactId>
</dependency>
```

4.7.2 Gestionar identidad Dropbox.

Una vez tenemos claro el planteamiento, creamos el servlet que nos permitirá gestionar la identidad vía dropbox.

La primera fase es la obtención del token, para ello, si al llegar al servlet disponemos de un token, somos redirigidos a la pantalla inicial en cambio si no disponemos de él comienza el proceso de obtención. En este proceso se manda al usuario a una web de Dropbox a que obtenga un código que nos servirá para acceder al token. El usuario tiene que loguearse y demostrar su identidad, confirmar que nos da permisos para poder acceder a sus datos almacenados y Dropbox lo redirigirá a nuestra plataforma con el token como parámetro, una vez de vuelta, se encuentra en la fase de obtención por lo que es el sistema el que con ese código hacer un requerimiento del token directamente hacia dropbox por lo que obtenemos el token y lo almacenamos para futuras consultas. Aquí hay un ejemplo del código de ese procedimiento:

```
if(request.getParameter("state")==null){
    DbxWebAuth.Request authRequest = DbxWebAuth.newRequestBuilder()
        .withRedirectUri(getRedirectUri(request), getSessionStore(request))
        .build();

    String authorizeUrl = getWebAuth(request).authorize(authRequest);
    response.sendRedirect(authorizeUrl);
}else if(request.getParameter("token")==null){
    try {
        DbxAuthFinish authFinish;
        authFinish = getWebAuth(request).finishFromRedirect(
            getRedirectUri(request),
            getSessionStore(request),
            request.getParameterMap()
        );
        request.getSession(true).setAttribute("token", authFinish.getAccessToken());
        request.getSession(true).setAttribute("tmphash", UUID.randomUUID());
        request.getSession(true).setAttribute("type_store_servlet", "url");
        request.getSession(true).setAttribute("store_servlet", "filedropbox");
```

Figura 23. Procedimiento de obtención del token Dropbox.

En este aplicativo tal como comentamos antes se almacenará el servlet de almacenamiento que se usa por lo que, para incluir uno nuevo solo debemos tener otro servlet disponible y al iniciar sesión almacenar su nombre, a partir de ahí, todas las peticiones se realizarán hacia ese servlet.

```
request.getSession(true).setAttribute("token", authFinish.getAccessToken());
request.getSession(true).setAttribute("tmphash", UUID.randomUUID());
request.getSession(true).setAttribute("type_store_servlet", "url");
request.getSession(true).setAttribute("store servlet", "filedropbox");
```

Figura 24. Datos almacenados en sesión Dropbox.

```
request.getSession(true).setAttribute("token", json.get("access_token"));
request.getSession(true).setAttribute("tmphash", UUID.randomUUID());
request.getSession(true).setAttribute("type_store_servlet", "hash");
request.getSession(true).setAttribute("store_servlet", "filedrive");
```

Figura 25. Datos almacenados en sesión Drive.

Para realizar las peticiones hacia el proveedor de almacenamiento es necesario tener el token que hemos obtenido a través de la autentificación oauth con este se realizarán las peticiones de consulta de los ficheros, los directorios, previsualización, etc. Para la gestión de los documentos que realice el usuario es necesario generar una caché que nos permita manipular los datos. Sobre todo lo usamos para la generación de la preview y la generación del documento PDF firmado ya que debemos tener el documento en local para operar en el e incrustar la firma. Todo esto sería sencillo almacenando variables de usuario o llevando un registro de estos, pero un punto fuerte de este proyecto se basa en no gestionar datos de usuario ni montar bases de datos además de cara a la protección de los datos que se manipulen no se llevará registro de nada. Por este motivo como identificador de usuario en los datos que se traten en la caché se usará un identificador único con cada inicio de sesión de modo que la información queda almacenada en la memoria temporal del sistema operativo y se borrará en cuanto no se esté usando.

4.7.3 Tipo de proveedor de archivos.

Además de todos estos datos es necesario un campo que nos permita discernir entre un proveedor de almacenamiento que trata los ficheros como rutas absolutas al estilo del filesystem o uno que usa identificadores para los archivos.

Por ejemplo:

Fichero	Ruta para acceder a él	Tipo
inscripcion.pdf	/mis documentos/inscripcion.pdf	url
inscripcion.pdf	1N3WDiyYAZdruChJ-uCbHbFlPXOQ	hash

Tras finalizar y almacenar todos estos datos, el sistema que se está encargando de identificar al usuario y obtener un token para poder trabajar ya ha acabado, así que, de forma independiente se redirige al servlet que se encarga de la firma. En este momento las funciones del almacenamiento funcionará de forma independiente por lo que operaría sin problema hasta que se haga uso de la firma. A continuación seguiremos detallando el proceso de desarrollo de Dropbox obviando la implementación del servlet de firma.

Una vez tenemos el acceso al token lo que definimos ahora son las operaciones en el servlet pasadas por parámetros que reconoce la interfaz web. En concreto definimos las operaciones mencionadas en la implementación de la interfaz usando el framework de Dropbox que nos han proveído.

```
if(action != null && action.length()>0 && action.equals("nav")){
    jsonResp = DropboxFileServlet.actionNavigator(request, client, tmp_hash.toStr
}else if(action != null && action.length()>0 && action.equals("downloadToSign")){
    jsonResp = DropboxFileServlet.actionDownloadToLocal(request, client, tmp_hash
}else if(action != null && action.length()>0 && action.equals("download")){
    jsonResp = DropboxFileServlet.actionDownloadFileUrl(request, client);
}else if(action != null && action.length()>0 && action.equals("upload")){
    jsonResp = DropboxFileServlet.actionUploadFile(request, client);
```

Figura 26. Implementación de funcionalidades.

No entraremos en detalle en cuanto a la implementación de las funcionalidades ya que es simplemente consultar la documentación de la API de Dropbox y no es el objetivo final de este TFM. En concreto este Framework no proporcionó ningún problema pero como apunte

característico, las facilidades que otorgó la herramienta a la hora de subir ficheros pasandole un simple FileInput o hasta para la obtención de la previsualización de los documentos.

```
InputStream inputStream = new FileInputStream(new File(ori));
Metadata metadata = client.files().uploadBuilder(dest).withMode(WriteMode.OVERWRITE).uploadAndl
inputStream.close();
resp = metadata.toString();
```

Figura 27. Framework Dropbox subida fichero

```
FileOutputStream outputStream = new FileOutputStream(url);
client.files().getThumbnailBuilder(path).withFormat(ThumbnailFormat.PNG).
outputStream.flush();
outputStream.close();
```

Figura 28. Framework Dropbox subida fichero

4.7.4 Previsualización de Dropbox.

En nuestro sistema, la previsualización de las imágenes está delegado en los proveedores de almacenamiento. Son estos los que con sus medios crean imágenes de los documentos que solicitamos pero a pesar de eso, para continuar con el criterio de servlet y consumir el servicio, no permitiremos que la interfaz de usuario acceda a el directorio donde está los ficheros, así que, lo publicamos a través de este servlet de modo que nos de mayor seguridad. Para ello solo existe la posibilidad de tener una única instancia que la consumiremos por el servlet “prefile”.

```
String mime = cntx.getMimeType(filename);

if (mime == null) {
    response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    return;
}

response.setContentType(mime);
File file = new File(filename);
response.setContentLength((int)file.length());

FileInputStream in = new FileInputStream(file);
OutputStream out = response.getOutputStream();
```

Figura 29. Porción de código Servlet Prefile.

[Problema]

La problemática que nos surge es que como nuestro aplicativo funciona de forma asíncrona, no detecta cambios en la url y por lo tanto la

cachea. Para solucionarlo añadimos un parámetro aleatorio que confunda al navegador y obligue a descargar la thumbnail.

4.8 Drive (eIdP)

Para completar los proveedores de almacenamiento, nos queda la plataforma de google, una vez realizamos los pasos anteriores ya tenemos una plataforma funcional que permite operar con dropbox. Para Google drive definimos las acciones necesarias tal y como hicimos en el otro proveedor de almacenamiento y para poder realizar las peticiones no usamos un framework, realizaremos las peticiones HTTPS tal y como nos muestra en la documentación. Para ello es necesario usar un certificado de google y usar un túnel ssl para la conexión tal como hicimos en TrustedX.

```
SSLContext sslcontext = SSLContexts.custom().l…
```

Una vez tenemos las acciones detalladas en el servlet, simplemente necesitamos llamar al nombre de este servlet. Como ya hemos mencionado, el servlet al que se realizan las peticiones de almacenamiento se almacena en una variable. Añadimos un nuevo botón de identificación al nuevo servlet, y al responder a las llamadas definidas funciona sin ningún inconveniente. A partir de esto necesitamos definir las credenciales de acceso como hicimos en el inicio del proyecto con dropbox. En concreto para Drive las credenciales se administran en la siguiente url: https://console.cloud.google.com

[Problema]

Es necesario contemplar el paso de las credenciales desde la planificación ya que no está funcionando igual a dropbox. Para pedir permisos que vean los documentos de drive es necesario un permiso que requiere validación y en nuestro caso tardó mucho y con múltiples pasos incluyendo grabar demos, etc.

El primer correo lo realicé desde el 24 de noviembre y obtuve la validación el mismo día de la entrega de la PEC.

Figura 32. Validación de Drive

4.9 TrustedX (eSigP)

Tras completar la identificación y el acceso a los datos por parte de Dropbox y Drive, comenzamos a hacer la conexión con TrustedX, nuestro proveedor de firma electrónica. Para ello, hemos establecido que se usará protocolo Https por lo que para establecer la conexión necesitamos el certificado público de TrustedX y lo metemos en un JKS de proveedores de almacenamiento y una vez tenemos esto ya podemos establecer conexiones https

```
try {
    SSLContext sslcontext = SSLContexts.custom().loadTrustMaterial(new File(routejks), p
    SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslcontext, new St
    CloseableHttpClient httpclient = HttpClients.custom().setSSLSocketFactory(sslsf).bui

    this.client = httpclient;
}   catch (NoSuchAlgorithmException ex) {
    Logger.getLogger(TrustedX.class.getName()).log(Level.SEVERE, null, ex);
} catch (KeyStoreException ex) {
```

Figura 30. Objeto de conexión SSL.

Una vez podemos establecer peticiones REST solo nos queda definir las operaciones necesarias tal como hicimos en el servlet de almacenamiento.

Operación	**URL**
Obtener token	https://uoc.safelayer.com:8082/trustedx-authserver/oauth/main/token Petición: **GET** Parámetros: • **grant_type**= permisos • **code**= código entregado al cliente • **redirect_uri**=donde devolver el token
Obtener identidades de firma	https://uoc.safelayer.com:8082/trustedx-resources/esigp/v1/sign_identities Petición: **GET** Parámetros: • **Authorization**= token
Borrar identidades de firma	https://uoc.safelayer.com:8082/trustedx-resources/esigp/v1/sign_identities/**[ID]** Petición: **DELETE** Parámetros: • **ID= identificador a borrar** • **Authorization**= token
Añadir identidades de firma	https://uoc.safelayer.com:8082/trustedx-resources/esigp/v1/sign_identities/server/pki_x509/pkcs12

	Petición: **POST** Parámetros: • **Authorization**= Token • **labels**= etiquetas definidas • **pkcs12**= contenido de la firma • **password**= contraseña del p12
Firmar hash ** CASO ESPECIAL	https://uoc.safelayer.com:8082/trustedx-resources/esigp/v1/signatures/server/raw Petición: **POST** Parámetros: • **Authorization**= Token • **digest_value=** Hash a firmar • **signature_algorithm=** Algoritmo usado • **sign_identity_id=** Identidad de firma

Para todas las identidades de firma se deberá cumplir estos requisitos ya que serán llamados desde la interfaz del mismo modo que los proveedores de almacenamiento.

```
if(action != null && action.length()>0 && action.equals("list")){
    jsonResp = sign.getIdentities(token);
}else if(action != null && action.length()>0 && action.equals("add")){
    String pkcs12 = request.getParameter("pkcs12");
    String labels = request.getParameter("labels");
    String password = request.getParameter("password");
    jsonResp = sign.addIdentity(token,pkcs12,labels, password);
}else if(action != null && action.length()>0 && action.equals("del")){
    String id = request.getParameter("id");
    jsonResp = sign.deleteIdentity(token, id);
}else if(action != null && action.length()>0 && action.equals("signtoken")
    jsonResp = sign.getTokenSign(request.getParameter("code"));
```

Figura 31. Servlet de firma peticiones.

En el caso de la firma detallamos qué es un caso especial ya que usamos las librerías de itext para la firma. Para que esta librería firme es necesario instanciar un tipo de objeto signature por lo que definimos el objeto que al intentar firmar un pdf se obtiene su hash y se manda como petición al servlet de firma y generar el pdf firmado.

```
MakeSignature.signDetached(appearance, dig, ObjSignature, new Certificate[]{cert}, null, null, null, 0, CryptoStandard.CADES);
```

Es vital que el documento firmado cumpla con la especificación de la ISO 32000-1. Con esto, el documento desde cualquier visor pdf podrá ser verificado danto las garantías necesarias.

5. Conclusiones y líneas futuras.

5.1 Conclusiones.

Al finalizar este trabajo hemos visto como el eIDAS nos ha abierto todo un abanico de posibilidades en cuanto a sistemas de firmado, permitiéndonos disponer de él, en la nube, siguiendo el modelo natural de progresión de internet en el que ya no basta con tener los documentos o los certificados en nuestros dispositivos. Además de este punto, es vital la forma de compartir estas funcionalidades adquiridas, gracias a esto, hemos podido completar este trabajo como si un puzle se tratara consumiendo servicios de diferentes proveedores.

Desde el inicio del TFM nos lo planteamos con un breve espacio de tiempo, donde desarrollar un prototipo funcional del proyecto y en este caso, hemos realizado una plataforma muy completa que cumple con todos los casos de uso que nos propusimos. Existen mejoras que a lo largo del desarrollo hemos ido ideando y que expondremos ya que fue materialmente imposible incluirlas.

A lo largo del trabajo se han indicado los problemas que se tuvieron pero la planificación fue bastante acertada aunque dediqué algunas horas más por cada día. El problema más grave fue en la obtención de los permisos de Google Drive, el cual estuvo hasta el último día del proyecto sin estar aceptado. A pesar de eso desarrolle el proyecto a la par que cumplía con los requisitos que me iban solicitando y gracias a que inicie el procedimiento antes del comienzo de la PEC3.

Tras finalizar este TFM, podemos decir que hemos trabajado con varias tecnologías, en temas sobre identidad en internet y firma electrónica y hemos conseguido desarrollar una plataforma totalmente funcional que permite a cualquier usuario firmar sus documentos mediante firmas electrónicas en la nube.

5.1 Trabajos futuros:

Producto más modular: Actualmente, el trabajo está preparado para diseñar fácilmente IdP y SigP pero sería muy ventajoso modelar en forma de librería, para poder implementar rápidamente cualquier proveedor incluyéndose en el proyectos.

Reglamento eIDAS: El proyecto permite el uso de firmas electrónicas y no implica mayor modificación cumplir con el uso de eIDAS. Una vía futura de desarrollo es hacer uso de las funcionalidades de TrustedX que nos permitan cumplir con el reglamento europeo.

Otras interfaces: Gracias a cómo hemos diseñado la parte de la interfaces, es muy sencillo poder diseñar otro tipo de interfaces como aplicaciones móviles, que hagan consumo de los servlet disponibles. Se podrían diseñar en Android o iOS incitando a la liberación de los datos del dispositivo, permitiéndole subir el contenido y sus certificados a la nube.

Estadísticas: Sería interesante realizar estadísticas sobre el uso del aplicativo, a modo de business intelligence, aprovechando los documentos que se suben y el uso que se le de. Así como ofrecer una trazabilidad a los documentos firmados al tener datos de cuando se suben, si se firman o se modifican.

Validar firmas online y otras ventajas de TrustedX: Además de lo propuesto, TrustedX ofrece muchas otras ventajas, sería útil usarlas todas validando los certificados online, usando identificación por medio de Safelayer Mobile ID.

Otros proveedores: Es interesante añadir, a modo de mayor funcionalidad, otro tipos de proveedores de identidad y firma además de separar los proveedores de identidad de los proveedores de almacenamiento, de este modo podríamos aceptar nuevos IdP como Facebook o Twitter y almacenamientos como OneDrive.

6. Glosario

- API: API o La interfaz de programación de aplicaciones es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

- oAuth: oAuth es un estándar abierto que permite flujos simples de autorización para aplicaciones que permite autorización segura de una API para consumir servicios de terceros.

- Firma electrónica: Es un conjunto de datos electrónicos que están asociados a un documento electrónico y identifica al firmante de manera inequívoca, asegurar la integridad del documento firmado y asegura que el documento firmado es exactamente el mismo que el original y que no ha sufrido alteración o manipulación.

- Firma cualificada: Es una firma electrónica avanzada que se crea mediante un dispositivo cualificado de creación de firmas electrónicas y que se basa en un certificado cualificado de firma electrónica.

- Sigp: Es el proveedor de firma electrónica que nos ofrece servicios de firma.

- IdP: Es un proveedor de identidad que nos ofrece servicios de identidad desde terceras plataformas.

7. Bibliografía

1. Reglamento (UE) Nº 910/2014 del Parlamento Europeo y del Consejo, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG

2. S. Pressman, Roger. Ingeniería del Software, Un enfoque práctico. McGrawHill Interamericana de España S.L., 2010.

3. JDK8 Documentation. https://docs.oracle.com/javase/8/docs/

4. Docker Documentation. https://www.docker.com/get-started

5. Tomcat 8 Documentation. http://tomcat.apache.org/tomcat-8.0-doc/

6. RFC 8017. PKCS #1: RSA Cryptography Specifications https://tools.ietf.org/pdf/rfc8017.pdf

7. Dropbox APIv2 https://www.dropbox.com/developers/documentation/java

8. Ley 59/2003, de 19 de diciembre, de firma electrónica https://www.boe.es/boe/dias/2003/12/20/pdfs/A45329-45343.pdf

9. Elder Moraes, Java EE 8 Cookbook. Packt, United Kingdom, Birmingham.

10. David Flanagan, JavaScript: La Guía Definitiva. Anaya Multimedia, O'Reilly

8. Anexos

A continuación, se adjuntan los anexos al Trabajo de Fin de Máster que por motivo de su carácter autocontenido se encuentran separados del documento inicial.

8.1 Anexo 1: Manual de usuario.

Inicio de sesión y navegación

Lo primero que podemos observar entrando a la url https://uoc.safelayer.com:2080/ es la pantalla inicial en la cual podemos decidir qué proveedores usar.

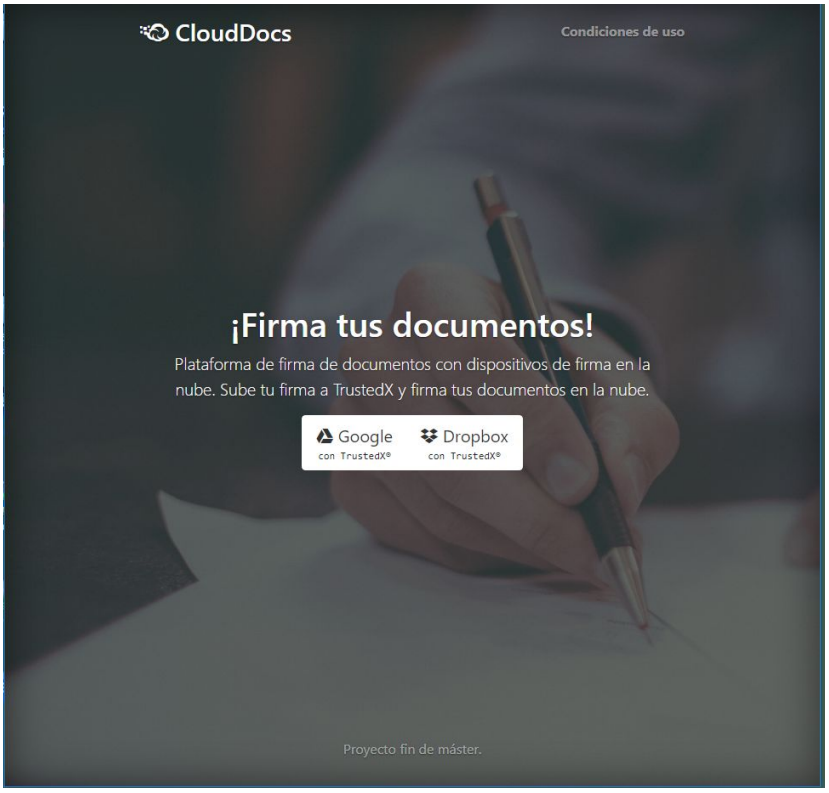

Figura a1. Inicio de sesión.

Podremos usar Google Drive y Dropbox para identificarnos y como proveedores de almacenamiento. Actualmente únicamente se provee de TrustedX como proveedor de firma.

Una vez seleccionamos un proveedor de almacenamiento, nosotros escogemos para la prueba a Google Drive, nos redirigirá a Google para

que nos identifiquemos con la cuenta que deseamos entrar y otorgar permisos a la aplicación.

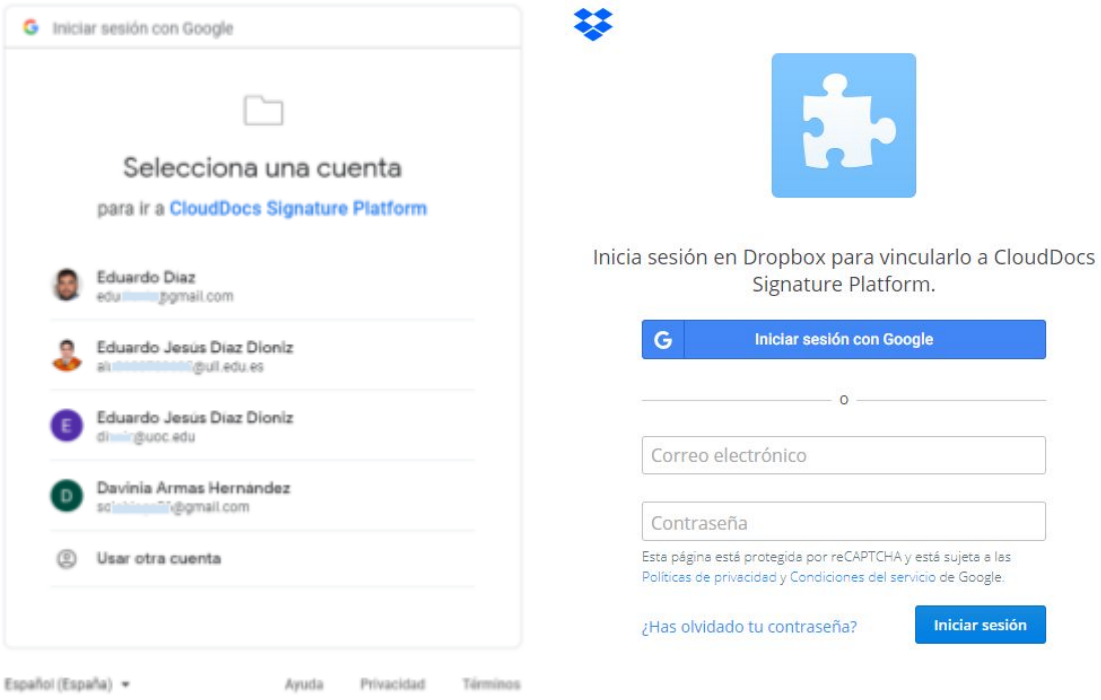

Figura a2. Autentificación Drive y Dropbox.

En la figura anterior podemos ver varias cuentas que ya están iniciadas en el equipo.

A continuación nos redirigirá a el proveedor de firma para obtener autorización para ver y gestionar las identidades de firma, este paso es necesario ya que una vez dado los permisos a la aplicación toda la gestión se encuentra integrada.

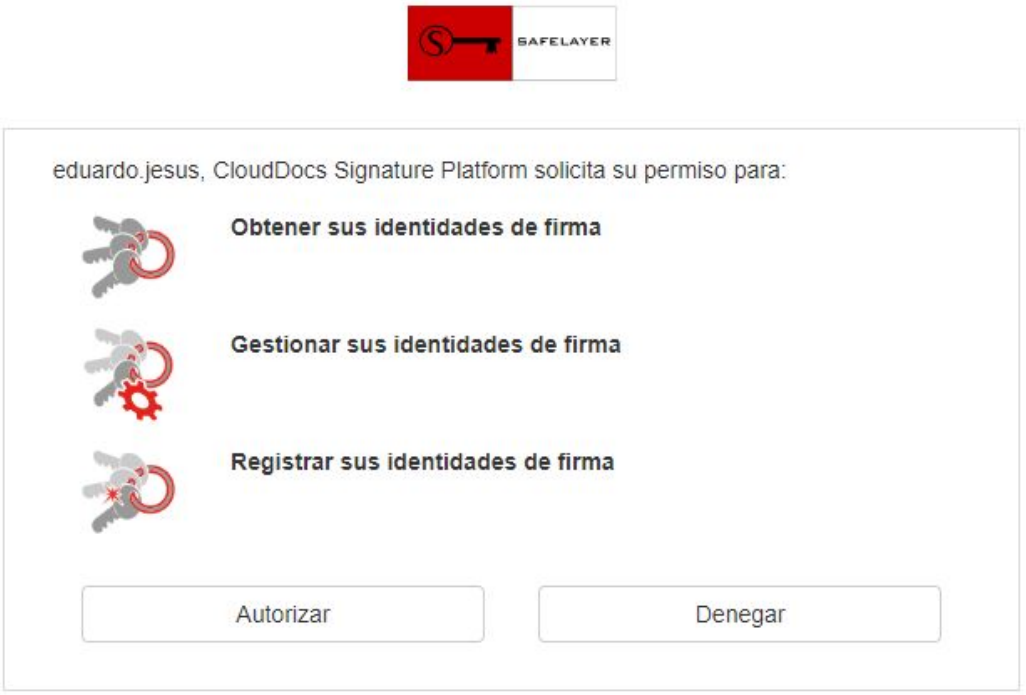

Figura a3. Autorización de token TrustedX.

Una vez otorgamos los permisos para la gestión de la firma ya estamos dentro del aplicativo y podemos navegar por las carpetas que se muestran del proveedor de almacenamiento que hayamos seleccionado.

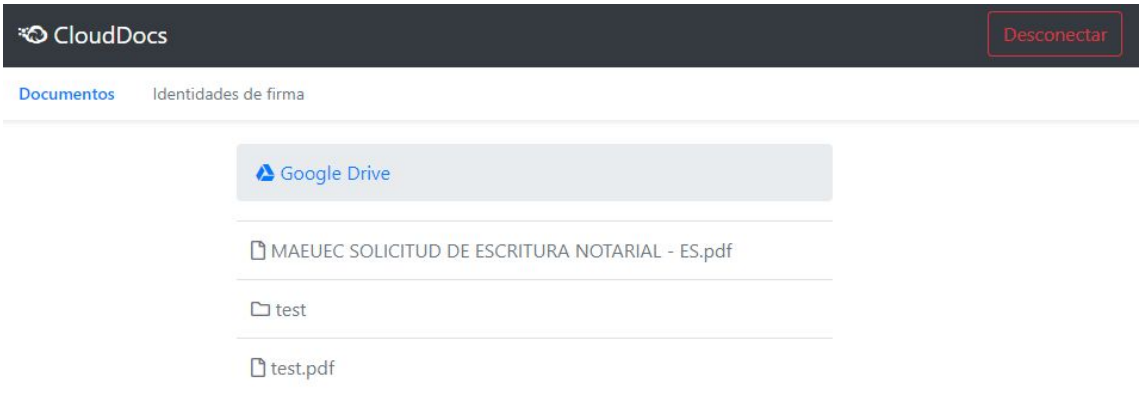

Figura a4. Ficheros y funcionalidad de navegación.

Podemos navegar por ellas y volver atrás con nuestra barra de navegación o previsualizar los documentos que desees.

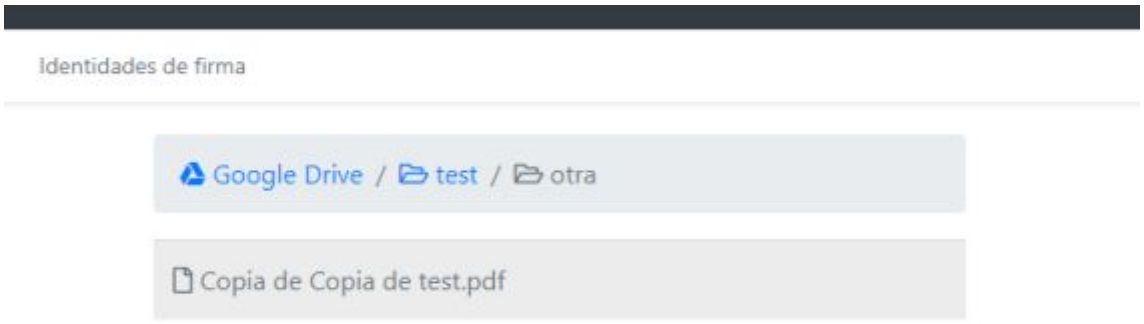

Figura a5. Funcionalidad navegación.

Proceso de firma de un documento.

Una vez abierto un documento, se mostrará una previsualización para entender que estás manipulando y se da la opción de descargarlo o de firmarlo electrónicamente con TrustedX.

Figura a6. Previsualización

Al seleccionar la opción de firma nos muestra la siguiente pantalla que nos indica el documento a firmar y nos pide elegir la identidad de firma que usaremos.

Figura a7. Ventana de firmado.

Tal y como vimos anteriormente, esta vez es necesario una autorización específica por parte del usuario para usar la identidad de firma para crear el PDF firmado.

Figura a8. Autorización para usar identidad de firma.

Una vez se completa la firma podemos ver como en nuestro navegador vemos ya el documento firmado. Si entramos en nuestra cuenta del proveedor de almacenamiento veremos que tenemos el documento firmado ahí.

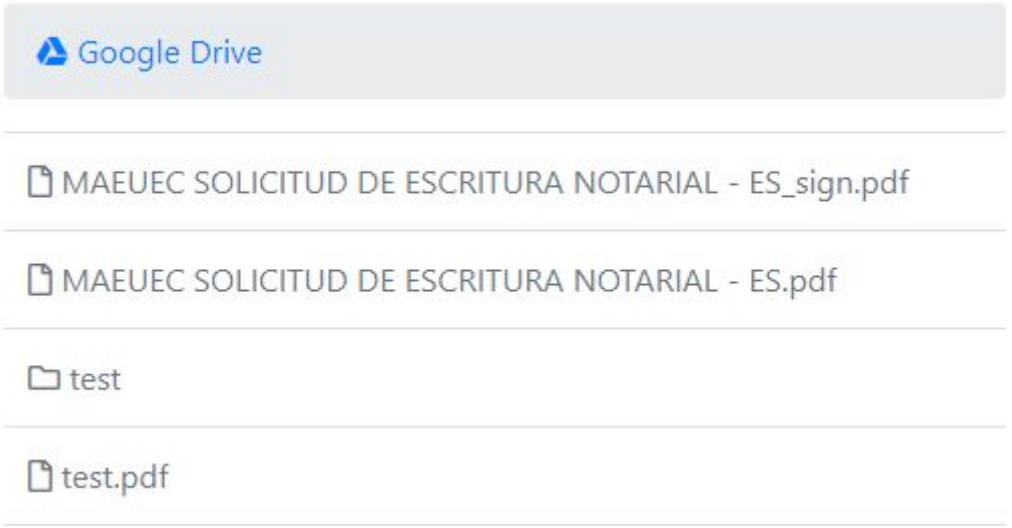

Figura a9. Navegación con documento firmado.

Si descargamos el documento y lo abrimos con un visor de PDF podemos ver la información de los metadatos y la firma correctamente realizada además de visualmente en una esquina del PDF.

Figura a10. Ejemplo de documento firmado.

Identidades de firma.

Por último nos queda las identidades de firma que se encuentran en el apartado “identidades de firma” en la barra de navegación superior.

Figura a11. Barra de navegación.

Una vez seleccionado podemos ver las identidades de firma que tenemos, borrarlas o incluir nuevas.

Figura a12. Identidades de firma

Si seleccionamos nueva identidad de firma nos indicará que seleccionemos un documento `.p12` del equipo y abrirá un cuadro de diálogo para solicitarnos la password del fichero y una vez puesta nos añade la identidad de firma lista para firmar.

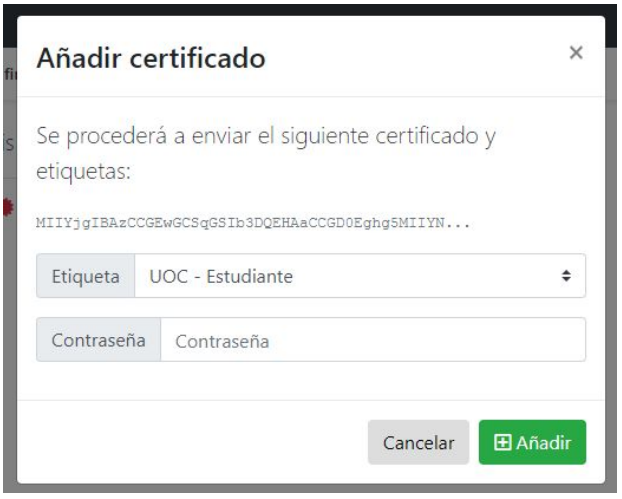

Figura a13. Añadir certificado