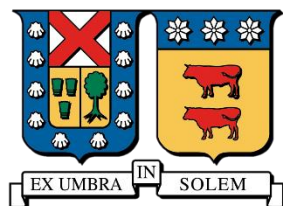


UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO – CHILE



DESARROLLO DE UN PORTAL WEB PARA SUBIR VIDEOS A YOUTUBE

MATÍAS ANDRÉS MÁRQUEZ COSTA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO
CIVIL EN ELECTRÓNICA

Profesor Guía: Agustín González

ENERO 2024

**UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA**

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO – CHILE



DESARROLLO DE UN PORTAL WEB PARA SUBIR VIDEOS A YOUTUBE

MATÍAS ANDRÉS MÁRQUEZ COSTA

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN ELECTRÓNICA**

Profesor Guía: Agustín González

Profesor Correferente: Rudy Malonnek

ENERO 2024

RESUMEN

Matías Márquez

Memoria para optar al título de Ingeniero Civil Electrónico, mención en Computadores

Profesor guía: Agustín González

Enero 2024

Durante el desarrollo de este proyecto, se desarrolla un portal web en donde los usuarios podrán autenticarse con sus cuentas de Microsoft institucionales, y luego subir sus videos a una única cuenta de YouTube. Todo esto dentro de una interfaz que sea sencilla y amigable, con el objetivo de que pueda ser utilizada por no especialistas.

La utilización de un único canal de YouTube para distintos usuarios puede ser problemática por el hecho de tener que manejar una cuenta entre varias personas, lo que implica una única contraseña y un único mail para manejar la cuenta.

Además, subir los videos a YouTube puede ser confuso para aquellos que no están familiarizados con el sitio, por lo que minimizar la cantidad de pasos requeridos y simplificar la información que se les brinda a solamente lo esencial, ayudará a llevar el proceso con más rapidez y facilidad.

Para la creación del sitio web se utilizó Django, un framework para desarrollo de sitios web basado en Python; acompañado de Bootstrap, otro framework que facilita el desarrollo de sitios web, para crear una interfaz que sea intuitiva y fácil de usar.

Para administrar los videos en YouTube, se utilizó la YouTube Data API v3, y para autenticar a los usuarios a través de sus cuentas de Microsoft, se utilizó Microsoft Entra ID, el servicio de administración de identidades de Microsoft.

En el sitio web final, los usuarios, luego de haber sido invitados a usar la aplicación, pueden autenticarse con éxito a través de sus cuentas institucionales, para luego poder subir sus videos a la cuenta compartida, en la cual cada uno podrá luego, editar o eliminar aquellos videos que él o ella haya subido.

Palabras claves: YouTube API, Microsoft Entra ID, portal web, aplicación web.

ABSTRACT

Matías Márquez

Final project report towards the fulfillment of the Ingeniero Civil Electrónico, minor in Computers degree

Advisor: Agustín González

January 2024

Through the development of this project, a web portal is developed in which the users will be able to authenticate with their Microsoft work or school accounts, and then upload their videos to a single YouTube account. All of this within an interface that is simple and friendly, with the objective of being easy to use.

The use of a single YouTube channel for various users can be problematic because of having to manage an account between different people, which implies a single password and a single email to manage the account.

In addition, uploading videos to YouTube might be confusing to those that are not familiarized with the site, so minimizing the number of required steps and simplifying the information provided to only the essentials will help make the process faster and easier.

Django, a Python based framework for website development, was used to create the website, along with Bootstrap, another framework that facilitates website development, to create an interface that is intuitive and easy to use.

To manage YouTube videos, the YouTube Data API v3 was used; and to authenticate users through their Microsoft accounts, Microsoft Entra ID, Microsoft's identity management service, was used.

On the final website, users, after having been invited to use the application, can successfully authenticate through their institutional accounts, and then upload their videos to the shared account, where each user can then edit or delete the videos he/she has uploaded.

ÍNDICE

RESUMEN.....	ii
ABSTRACT	iii
ÍNDICE	iv
1. INTRODUCCIÓN.....	1
2. ADMINISTRACIÓN DE CANALES DE YOUTUBE A TRAVÉS DE VARIOS USUARIOS.....	3
3. REQUERIMIENTOS.....	4
3.1. Sitio web	4
3.2. Método para subir y editar información de videos a YouTube.....	4
3.2.1. YouTube Data API v3.....	4
3.2.2. Herramienta de testeo para navegadores.....	5
3.3. Autenticación.....	6
3.3.1. Autenticación de Django.....	6
3.3.2. Autenticación de Microsoft.....	6
3.4. Registro de actividades.....	7
4. DISEÑO E IMPLEMENTACIÓN.....	7
4.1. Desarrollo del sitio web.....	8
4.2. Desarrollo del código para subir el video a YouTube.....	10
4.3. Desarrollo del sistema de autenticación.....	17
4.4. Desarrollo del código para editar información o eliminar videos.....	23
4.5. Registro de actividades.....	25
5. RESULTADOS	27
6. CONCLUSIONES Y TRABAJO FUTURO.....	28
6.1. Conclusiones.....	28
6.2. Trabajo futuro.....	28
REFERENCIAS	29
ANEXO	30

1. INTRODUCCIÓN

Este proyecto surge como respuesta a la problemática identificada en la compartición de una única cuenta de YouTube entre varios profesores (interacción ilustrada en la Figura 1Figura 1), en donde ellos puedan almacenar sus clases grabadas. La dificultad radica en gestionar los accesos, contraseñas y autenticación para cada persona, además de, en algunos casos, no estar familiarizados con la forma en que se cargan videos. Esta situación motivó la iniciativa de crear una solución que simplificara y centralizara el proceso de carga de videos en YouTube a través de una única cuenta, abordando de manera efectiva los inconvenientes mencionados. Para este fin, se desarrollará un portal web que pueda resolver estos problemas y actúe como intermediario entre el usuario y YouTube, como se muestra en la Figura 2Error! Reference source not found..

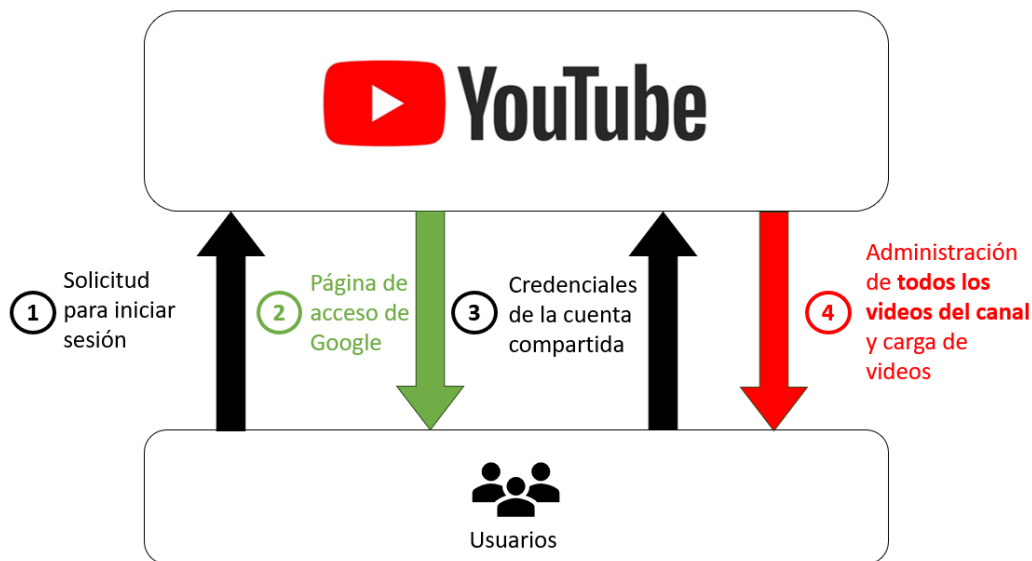


Figura 1. Interacción de los usuarios al utilizar una única cuenta de YouTube.

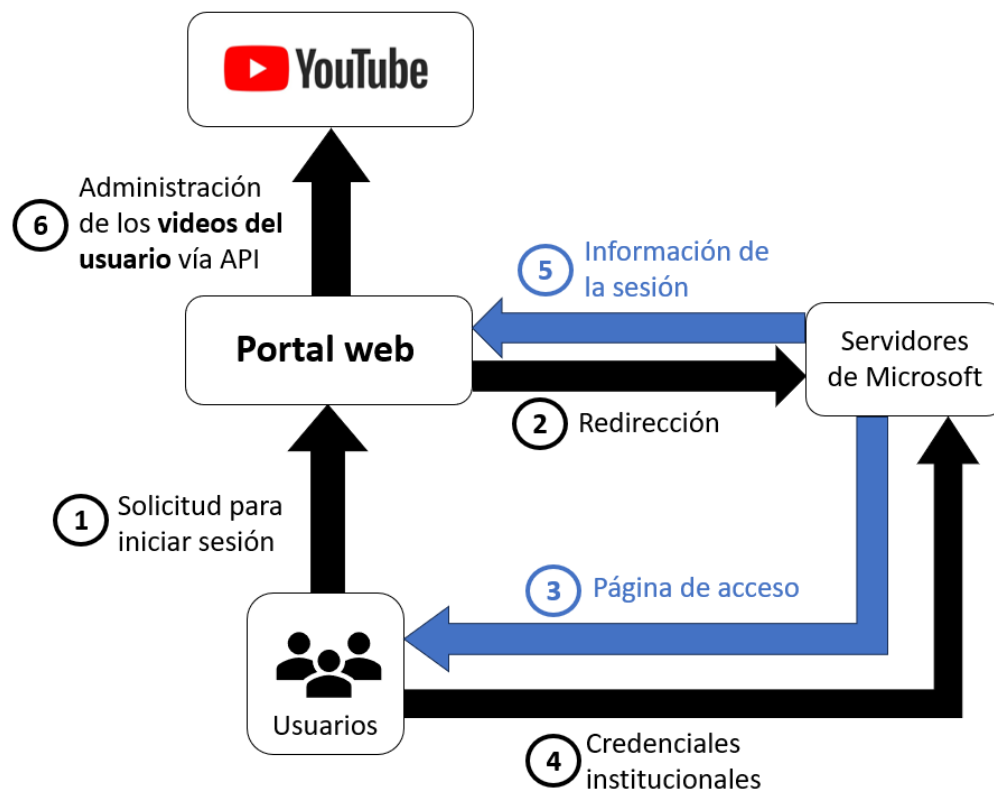


Figura 2. Interacción de los usuarios a través del portal web

La descripción de este trabajo se dividirá en los siguientes puntos:

- Desarrollo del sitio web
- Desarrollo del código para subir el video a YouTube
- Desarrollo del sistema de autenticación
- Desarrollo del código para editar información o eliminar videos
- Registro de actividades

Además de simplificar la carga de videos, el proyecto ofrece a los usuarios la capacidad de gestionar sus contenidos de manera eficiente. Por lo que, dentro del sitio, cada persona tiene la facultad de editar o eliminar únicamente los videos que ha subido.

Para el desarrollo de este proyecto se utilizó el framework Django (creación del sitio), la API de YouTube (carga de videos), y el sistema de autenticación de Microsoft, todo programado en Python.

Se espera que con la debida verificación esta pueda ser una herramienta que ayude a las personas que comparten un canal de YouTube a administrar sus videos de mejor manera.

2. ADMINISTRACIÓN DE CANALES DE YOUTUBE A TRAVÉS DE VARIOS USUARIOS

En la actualidad, la gestión de un único canal de YouTube entre varios usuarios que requieren su acceso para gestionar su contenido puede ser complicada, dado que todos los usuarios comparten la misma contraseña, por lo que, en caso de que alguien la olvide, se tendrá que cambiar para todos; y en caso de ocupar autenticación de 2 pasos (que es una opción de seguridad recomendada), se va a requerir que todos necesiten un único dispositivo para poder ingresar.

Compartir una cuenta entre varias personas puede dificultar además, la tarea de administrar los videos que cada uno sube o quiere ordenar, ya que al querer buscar los videos que el usuario quiere editar, aparecen todos los videos del canal.

En algunos casos, cuando se necesita administrar el contenido de varios usuarios en un canal de YouTube, se puede tener un equipo o persona a cargo de esta labor. En cuyo caso, desde el punto de vista de YouTube es un único usuario y desde el punto de vista de la organización una única persona responsable de los contenidos.

El fin de este proyecto es que un único canal de YouTube que se compartirá sea manejado por los mismos usuarios, a través de un portal web, en el cual cada usuario deba autenticarse de manera individual a través de sus cuentas institucionales. La idea de este portal es que tenga una interfaz más simple para subir y administrar videos, y dar permiso al usuario de subir y administrar sus videos, teniendo cada usuario influencia solamente sobre su contenido.

Esto resuelve los problemas de:

- Compartir las credenciales del canal de YouTube. En este caso ningún usuario las poseerá
- Necesitar un administrador del canal. Al ser cada usuario responsable de su contenido, podrá añadir, editar, o eliminar su material directamente desde el portal.
- Simplificar los procesos de subir, editar o eliminar videos. Estos procesos pueden ser complicados para aquellos usuarios que no manejen la plataforma de YouTube, por lo que hacer una interfaz con las opciones principales simplifica los procesos y la cantidad de pasos que requieren.

Este proyecto puede resultar beneficioso para aquellos profesores o personas que no están tan familiarizados con la plataforma de YouTube, y también para organizaciones o instituciones que deseen tener un canal compartido de YouTube en donde una variedad de usuarios tenga que subir su contenido.

3. REQUERIMIENTOS

Para desarrollar el proyecto es necesario definir la estructura de la solución, y cada uno de los módulos que compondrán la solución. Estos módulos son:

- Desarrollo del sitio web
- Desarrollo del código para subir, editar la información o eliminar videos de YouTube
- Desarrollo del sistema de autenticación
- Registro de actividades

Aunque en esta sección se han unido la subida, edición de información y eliminación de videos, ya que comparten el mismo requerimiento para su ejecución.

3.1. Sitio web

La elección del framework a utilizar para desarrollar el sitio sólo tiene relevancia para el desarrollo, ya que hay una multitud de opciones disponibles, y en lo único que se distinguen es en el lenguaje de programación a utilizar y en algunas funcionalidades que incluyen, pero desde el punto de vista del usuario, el resultado final será el mismo independientemente del lenguaje escogido.

En este caso se utilizará Django ([Django, s.f.](#)), ya que Python es el lenguaje de programación con el que se tiene más experiencia. Django es un framework de desarrollo web basado en Python, el cual busca facilitar el desarrollo de aplicaciones web. La elección de Django como herramienta permitirá crear eficientemente un sitio web que posibilite una implementación rápida y efectiva. Además, incluye una base de datos en la cual se guardará la información de cada uno de los videos subidos por los usuarios.

3.2. Método para subir y editar información de videos a YouTube

Para la carga de videos a YouTube, realmente solo se tienen 2 opciones: utilizar la API oficial de Google, o alguna herramienta que pueda simular a un humano subiendo los videos. A continuación se exploran ambas opciones.

3.2.1. YouTube Data API v3

YouTube Data API v3 es la última versión de la API de YouTube. Esta permite obtener toda clase de datos de canales, como sus suscriptores, videos, listas de reproducción, duraciones, me gusta, comentarios, etc.

La versión normal de esta API otorga al usuario 10.000 créditos por día, los cuales se utilizan para las distintas funcionalidades que proporciona la API ([Google for Developers, 2023](#)). Esto limita la cantidad de videos a subir hasta un máximo de 6 por

día (cada uno gasta 1.600 créditos), por lo que se puede volver fácilmente una limitante para la aplicación. Para poder aumentar la cantidad de créditos diarios, es necesario llenar un formulario ([Google Support, s.f.](#)) en el cual se debe indicar la organización que lo utilizará, y el uso que se le dará entre otros.

Además, YouTube no permite videos con una duración sobre los 15 minutos, a menos de que la cuenta esté verificada ([YouTube Help, s.f.](#)). Para esto se necesita asignar un número celular a la cuenta, y una vez se confirme vía un código, la cuenta quedará verificada, lo que permitirá subir videos de hasta 256 GB o 12 horas de largo, lo que se cumpla primero.

Es relevante notar, que las aplicaciones creadas en Google Cloud para todo tipo de usuario inicialmente están en modo de prueba, por lo cual el token que se otorgue tendrá una duración de 7 días, no extendible. Para extender la duración de los tokens, es necesario verificar la aplicación. Esto se hace pasando por un proceso de verificación, el cual requiere explicar la aplicación, establecer un dominio, crear una política de privacidad, hacer un video instructivo, etc.

Por otro lado, también se puede crear la aplicación para usos internos dentro de una organización de Google Cloud y solo para usuarios del Google Workspace. En este caso no es necesario verificar la aplicación, ya que las restricciones mencionadas anteriormente no aplican.

Además, la API permite crear listas de reproducción (playlists) y añadirles videos, por lo que, al especificar a qué asignatura y semestre corresponde el video, se podría crear una lista de reproducción en caso de que no exista y añadirle el video. En caso de que ya exista, solo se añade el nuevo video a la lista que corresponda.

Para poder hacer uso de la API, se debe crear un proyecto en la nube de Google y agregar la API al proyecto. Esta API tendrá una llave de API, la cual tendrá que ir en cada solicitud que se haga. Además, la cuenta que haga uso de la API debe autorizar en el navegador el uso de la aplicación y el alcance de las acciones que pueda realizar.

La idea es que el usuario, una vez autenticado, se comunique con el portal web y este, a través de la API de YouTube, suba y pueda modificar la información de sus videos, y no modifique nada directamente desde la cuenta de YouTube.

3.2.2. Herramienta de testeo para navegadores

Una alternativa a ocupar la API de YouTube sería una herramienta de testeo para navegadores, como lo es Selenium. Estas herramientas permiten que el código interactúe con el navegador de manera automatizada, simulando acciones realizadas por un usuario convencional, como llenar formularios, hacer clic en botones y más. Esto se logra identificando a los elementos HTML relevantes del sitio web, ya sea por id, clase, nombre, etc., y luego realizando acciones sobre ellos.

Este método permite subir videos de la misma forma que un usuario lo haría interactuando con la página de YouTube, pero requiere identificar el paso a paso cómo subir el video y todos los elementos HTML necesarios para navegar el sitio.

Subir videos de esta forma no está limitado por cuotas como lo es con el uso de la API, por lo que en primera instancia se podrían subir más de 6 videos, aunque el límite no es claro, se estima que se pueden subir entre 15 a 100 videos al día dependiendo del canal.

Por otro lado, en caso de que cambien algunos elementos en el sitio web o los pasos a seguir para subir el video, el código podría dejar de funcionar debido a referencias obsoletas o falta de pasos para completar las instrucciones. Lo mismo aplica para editar un video, eliminarlo, o cualquier actividad que se quiera realizar dentro del sitio de YouTube.

La incertidumbre de los cambios hace esta opción menos viable a largo plazo, ya que puede requerir mantención sin previo aviso y dejar la aplicación sin funcionar hasta que se repare el código. Esta razón es la principal por la cual se escoge la API de YouTube para trabajar en este proyecto.

3.3. Autenticación

Para la autenticación de usuarios en el sitio se tienen en cuenta dos alternativas principales.

3.3.1. Autenticación de Django

La primera opción sería bajo un sistema de registro y acceso dentro del mismo portal, en donde se tendría que autorizar o invitar a cada usuario a crear su cuenta dentro del sitio. Para esto el usuario tendrá que crear un nombre de usuario (u ocupar su correo) y una contraseña para poder ingresar al sitio. Esta alternativa es ofrecida por Django y es la más simple, pero hará que el usuario deba tener un par de credenciales más que manejar y/o aprender, y no tiene medidas de seguridad como la verificación de 2 pasos.

3.3.2. Autenticación de Microsoft

Microsoft Entra ID (antes Azure Active Directory) es el servicio de administración de identidades de Microsoft. Se utiliza para ayudar a los usuarios a iniciar sesión y acceder a aplicaciones y/o recursos.

Al utilizar esta opción, los profesores, luego de ser invitados o añadidos a la aplicación, tendrán la oportunidad de iniciar sesión a través de sus cuentas institucionales, utilizando las mismas credenciales que usan para acceder a los sitios de la universidad.

Dadas las ventajas de comodidad, seguridad, y facilidad de adopción por parte de los usuarios, se escoge la autenticación de Microsoft como la opción a implementar en este proyecto.

3.4. Registro de actividades

El mantener registro de las actividades de los usuarios en el sitio es importante para poder identificar las acciones que se han realizado o errores que han sucedido.

Al igual que la elección de framework para trabajar en el diseño del sitio web, formas de crear registros hay muchas, y todas llevan a los mismos documentos. Por lo que en este caso se trabaja con la herramienta “logging” ([Logging facility for Python, \(s.f.\)](#)) de Python por ser el lenguaje escogido para trabajar.

4. DISEÑO E IMPLEMENTACIÓN

A continuación se muestra cómo se implementan cada uno de los módulos a utilizar durante el desarrollo del proyecto. Para comenzar se muestra un diagrama que indica qué funciones cumplen cada uno.

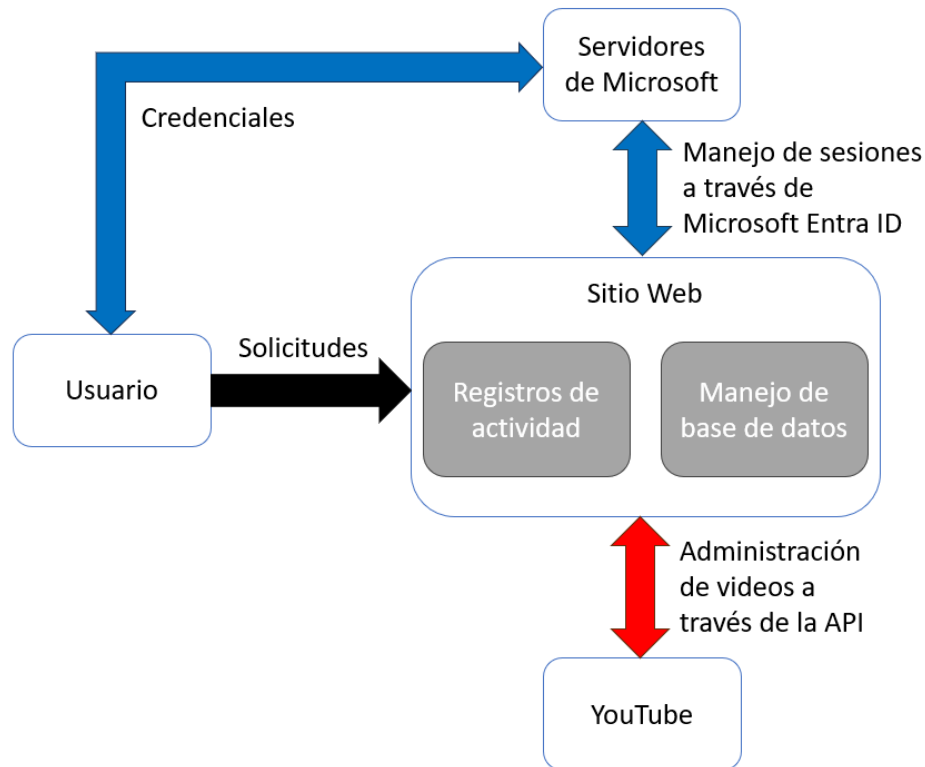


Figura 3. Diagrama de cómo se comunicarán las distintas partes para hacer funcionar el sitio web.

En donde el sitio web se comunicará con los servidores de autenticación de Microsoft para el manejo de sesiones dentro del sitio, una vez redirigido al servidor de Microsoft, el usuario deberá ingresar sus credenciales para iniciar sesión, y luego los servidores de Microsoft le darán el objeto de sesión al sitio web.

Por otro lado, la administración de los videos se hará a través de la API de YouTube, por lo que cualquier petición del usuario para subir, editar los datos, o eliminar videos, se convertirá en una petición a la API para hacer dichos cambios en el canal de YouTube.

A su vez, el sitio web mantendrá registros de las acciones relevantes que realicen los usuarios dentro del sitio, además de manejar la base de datos para que se mantenga actualizada con la información de los videos de cada usuario.

4.1. Desarrollo del sitio web

Para el desarrollo del sitio web, lo primero que se hará es instalar las librerías necesarias para el desarrollo del proyecto, contenidas dentro de requirements.txt (ver el Anexo 1). Ahí se tendrán las librerías:

- django: framework web que permite el diseño de sitios de manera sencilla.
- django-extensions: extensiones varias para el framework Django. Se ocupa de manera exclusiva por su funcionalidad runserver_plus, que permite correr el sitio en HTTPS.
- werkzeug: librería de utilidades necesaria para correr runserver_plus.
- pyopenssl: permite crear certificados para correr el sitio en HTTPS.
- django-cleanup: elimina los archivos asociados a un registro de la base de datos. Como sólo es posible guardar la ubicación de archivos dentro de los registros, esta librería se encarga de eliminar dichos archivos al momento de eliminar una entrada.
- google-api-python-client: permite usar las APIs de Google, en este caso, la de YouTube.
- google-auth-oauthlib: encargada de la autenticación a través de OAuth.
- msal: Microsoft Authentication Library. Permite la autenticación a través de los servidores de Microsoft.
- ms-identity-web: es un wrapper de MSAL, lo cual permite darle una implementación más simple a la librería.

Luego se debe crear el proyecto de Django, para lo cual se usará el siguiente comando en el directorio deseado:

```
> django-admin.exe startproject proyecto_titulo
```

Dentro del proyecto, se añadirá una aplicación al proyecto, en la cual se crearán los sitios a los que accederán los usuarios:

```
> cd proyecto_titulo
```

```
> python manage.py startapp main_app
```

Una vez hecho esto, se debe añadir la aplicación a la lista de aplicaciones del proyecto, para esto, dentro de settings.py se debe añadir 'main_app' a la lista de

INSTALLED_APPS. Además, se añadirán las URLs de main_app a la lista de URLs principales, insertando la línea:

```
path("", include('main_app.urls')),
```

En el archivo urls.py, dentro de la lista urlpatterns. De esta forma, los sitios creados en el archivo urls.py de main_app se utilizarán para completar la URL base del sitio (e.g. <https://localhost:8000/>).

Dentro del directorio de main_app, se creará el modelo de Video (ver el Anexo 2) en el archivo models.py para poder crear las entradas en la base de datos. Este modelo contendrá la ubicación del archivo (no puede guardar archivos directamente en la base de datos) y los datos del video que se quiera subir, como su título, descripción, dueño (usuario que lo subió), ID de YouTube, y fecha de subida. Notar que para el archivo de video se ocupa una función intermedia, la cual cambiará el nombre del archivo de video por uno propio basado en la fecha en la que se subió para guardarlo.

Para navegar el sitio, se hicieron 4 URLs en el archivo urls.py de main_app (ver el Anexo 3): la pantalla de inicio, mis videos (donde se verán los videos del usuario), editar video (donde se podrá editar cada video según su ID), y subir video, en donde se tendrá el formulario para poder subir el video con su título y descripción. Adicionalmente, habrá una URL para eliminar videos. Todos los sitios en donde se interactúa con los videos estarán bloqueados hasta que el usuario inicie sesión.

Junto con las URLs, se crean las plantillas HTML en las cuales se mostrará el contenido correspondiente para las páginas disponibles. A continuación se explicará cada uno junto con una imagen:

- home.html (ver el Anexo 4) - Página principal del portal, en base a la que se hace el sitio.
- navbar.html (ver el Anexo 5) - Barra de navegación del sitio. No es una página en sí, pero ayuda a organizar el código, va dentro de home.html. Contiene los botones para volver al inicio, ir a subir video, ir a mis videos, y el logo con el nombre del usuario o botón de inicio de sesión, dependiendo si hay un usuario con sesión iniciada o no respectivamente.
- upload_video.html (ver el Anexo 6) - Página donde aparece el formulario para poder subir videos.
- my_videos.html (ver el Anexo 7) - Página en donde se muestran los videos que ha subido el usuario, incluyendo información de estos, y botones con diversas opciones.
- edit_video.html (ver el Anexo 8) - Página donde aparece un formulario para editar el video seleccionado por el usuario.
- success.html (ver el Anexo 9) - Página de éxito al cargar el video. Aquí se le mostrará al usuario el link de su video, un código para insertarlo en HTML, y una miniatura de su video.
- error_page.html (ver el Anexo 10) - Página de error en caso de que algo haya fallado en alguna operación.

4.2. Desarrollo del código para subir el video a YouTube

Para poder subir videos a YouTube a través de la API, lo primero que se necesita es crear un proyecto en Google Cloud (console.cloud.google.com) que represente la aplicación que va a hacer uso de la API. En el panel de APIs y servicios (ver Figura 4), se debe habilitar la YouTube Data API v3 (ver Figura 5), la cual nos permitirá subir, editar y eliminar videos. Esta aplicación será utilizada por el código alojado en el servidor que contenga el sitio web.

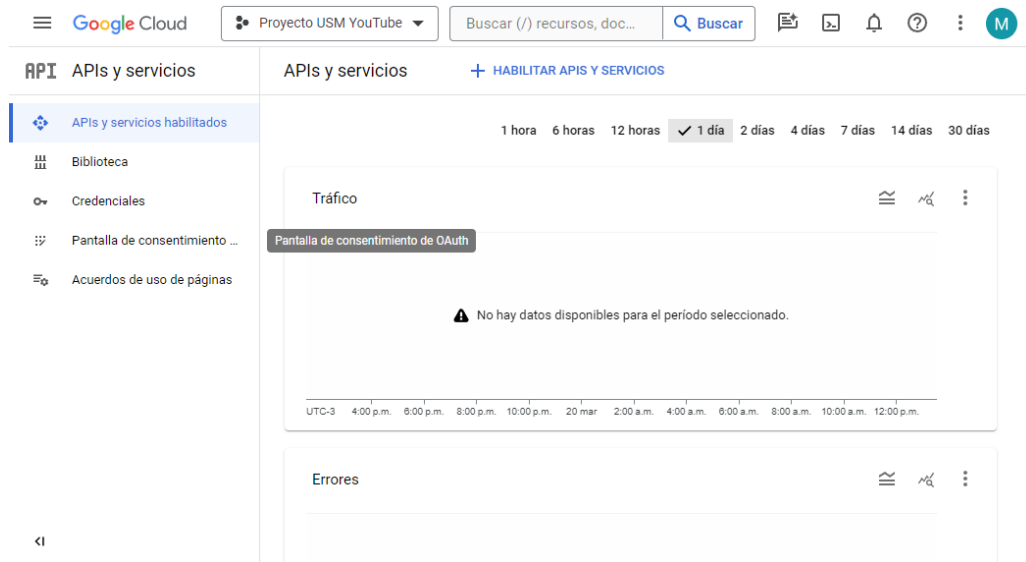


Figura 4. Panel de APIs y servicios de Google Cloud

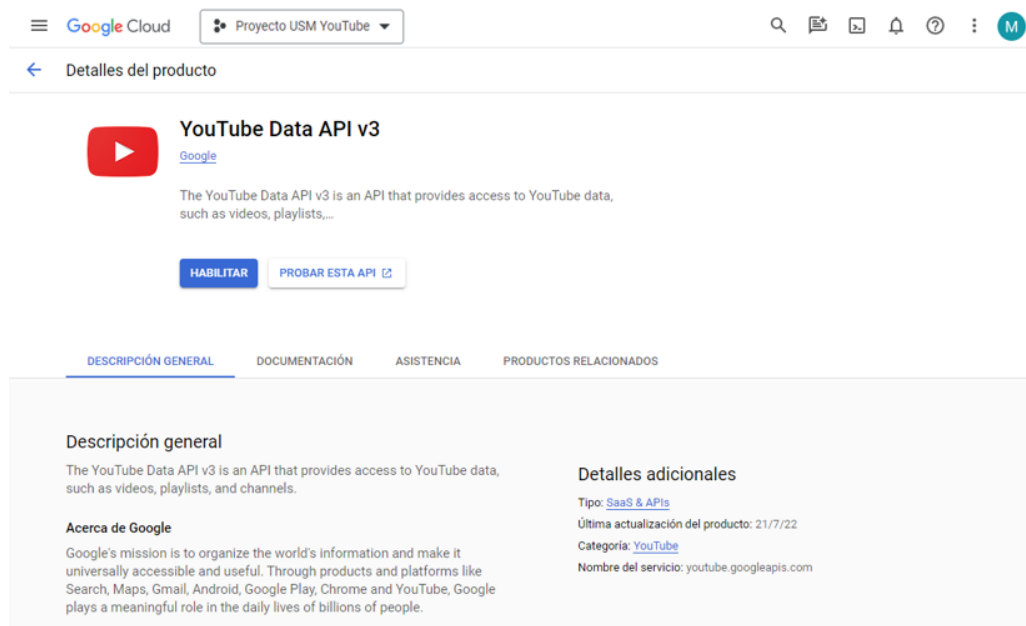


Figura 5. Página para habilitar YouTube Data API v3

Las APIs de Google utilizan el protocolo OAuth 2.0 ([OAuth, s.f.](#)) para su autenticación y autorización, por lo que es necesario para el uso de la aplicación, crear una pantalla de consentimiento de OAuth, la cual solicitará autorización a una cuenta de Google (en este caso, la que poseerá el canal de YouTube) para otorgar los permisos a la API de actuar según los alcances que se le otorguen. En este caso, se creará una pantalla de consentimiento interna para obtener una duración indefinida de los tokens, y los permisos de esta pantalla serían para subir, editar, y eliminar videos de la cuenta de YouTube, los cuales se deben añadir a la pantalla de consentimiento durante la configuración. En este caso serían:

- <https://www.googleapis.com/auth/youtube.upload> - Para subir videos
- <https://www.googleapis.com/auth/youtube.force-ssl> - Para editar y/o eliminar videos

El protocolo OAuth 2.0 se utiliza cuando se le quiere dar a la aplicación acceso limitado a la API en nombre del usuario. Para esto, se le entrega un token a la aplicación, la cual podrá utilizar durante un tiempo definido. La aplicación debe mostrar este token al servidor de recursos (YouTube en este caso) para obtener acceso a los recursos de este.

Una vez creada la pantalla de consentimiento, se puede crear un secreto de cliente, con el cual a través del código se pueden realizar las peticiones a la aplicación.

La primera vez que se intenta ocupar la aplicación, cuando se intente hacer una petición a la API, se abrirá una ventana de autorización de OAuth donde se pedirá seleccionar una cuenta con la cual acceder. El administrador del servidor deberá seleccionar la cuenta a la cual se subirán los videos. Luego de seleccionar la cuenta, aparecerá en la ventana el nombre de la aplicación junto con los permisos que se le dará a la aplicación. Una vez aceptados los permisos de dicha ventana, el servidor de autorización entregará al navegador un código de autorización, el cual luego la aplicación puede intercambiar, junto con el secreto de cliente (secreto generado en la aplicación de Google y presente dentro del código), por el token de acceso. Este flujo se puede visualizar en el diagrama de secuencia de la Figura 6.

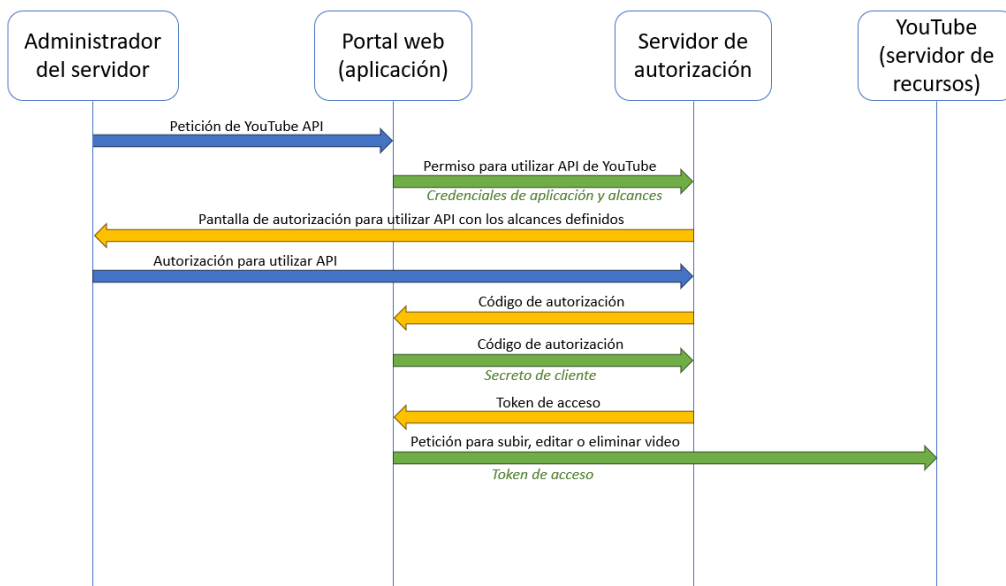


Figura 6. Flujo de información para conseguir la autorización de OAuth

De esta manera, la primera vez que se intente utilizar la aplicación, al no tener un token de acceso, aparecerá la pantalla de la Figura 7.

 Iniciar sesión con Google

Proyecto-YouTube quiere
acceder a tu cuenta de Google

 matias.marquez.14@sansano.usm.cl

Esto permitirá a **Proyecto-YouTube** hacer lo siguiente:

-  Ver, cambiar y eliminar de forma permanente tus vídeos, puntuaciones, comentarios y subtítulos de YouTube 
-  Administrar tus vídeos de YouTube. 

Confirma que confías en Proyecto-YouTube

Puede que estés compartiendo información sensible con este sitio o esta aplicación. Puedes ver o retirar el acceso en cualquier momento en tu [cuenta de Google](#).

Descubre cómo te ayuda Google a [compartir datos de forma segura](#).

Consulta la [Política de Privacidad](#) y los [Términos del Servicio](#) de Proyecto-YouTube.

Cancelar

Permitir

Figura 7. Pantalla de consentimiento de la aplicación de Google Cloud para poder utilizar la API de YouTube

Para subir un video se debe crear un formulario basado en el modelo de Video (ver Anexo 2) que se creó, este formulario se guarda en el archivo forms.py (ver el Anexo 11) dentro del directorio de la aplicación. Por lo que debe pedir título, descripción, y un archivo de video, como se puede ver en la Figura 8.

Subir un video

Título:

Descripción:

Archivo de video:

Seleccionar archivo	Ninguno archivo selec.
---------------------	------------------------

Figura 8. Formulario para subir videos.

Para añadir las funcionalidades de la API se siguen las instrucciones dadas en la página ([Google for Developers, 2023](#)). Primero, se crea un archivo con el código de ejemplo de YouTube para subir videos. Este código contiene 3 funciones clave:

- `get_authenticated_service`: Esta función se encarga de obtener el token que permitirá hacer peticiones autorizadas a la API de YouTube.
- `resumable_upload`: Esta función sube el video en chunks (fragmentos de información), y es capaz de resumir la carga de un video en caso de interrumpirse.
- `initialize_upload`: Es la función que se encarga de iniciar la carga del video, utiliza a `resumable_upload` para tal efecto.

A este código se le hacen algunos cambios para su operación:

- Se actualiza a Python 3:
 - La librería `httplib` se cambia por `http.client`. Esto simplemente requiere reemplazar “`httplib`” por “`http.client`”.
 - Se modifican las funciones `print`, poniendo entre paréntesis el contenido a mostrar.
 - Se modifican los `except`, reemplazando las “,” por “`as`” en las sentencias `except`.

- Se modifica para que funcione como una función con argumentos en vez de con una secuencia de comandos.
- Se actualizan ciertas librerías obsoletas (`apiclient` por `googleapiclient`, y `oauth2client` por `google.auth`).
- Se realizan arreglos menores que no modifican su funcionamiento pero que sirven para un uso más sencillo.

Quedando como se muestra en el Anexo 12.

Para poder correr este código, es necesario crear en su directorio un archivo llamado `client_secrets.json` con la información mostrada en el Anexo 13. Aquí se deben poblar los campos con la información del secreto de cliente creado anteriormente. Por lo que el campo `client_id` será el ID de cliente, y `client_secret` el secreto de cliente. Este archivo, junto con `upload_video.py`, serán guardados en una carpeta llamada `code` dentro de la carpeta del proyecto.

Para subir un video a YouTube, es necesario que el video se encuentre en el computador que lo quiera subir o ejecute el código, por lo que, en este caso, se requiere que el video sea subido al computador que poseerá el sitio antes de ser subido a YouTube.

Al subir el video, su nombre será cambiado por uno dado por la aplicación, el cual se compondrá por la fecha y número de video subido en esa fecha (e.g. 2023-12-20_18-00_001.mp4).

Para subir el video en base al formulario, se creó la función llamada `upload_video` (ver el Anexo 14), la cual tendrá el siguiente comportamiento:

Si el método de la solicitud es POST:

Se crea un objeto de formulario con los datos del video y el archivo adjunto

Si el formulario es válido:

Se crea un diccionario con los datos del formulario y se rellenan otros datos necesarios para subir el video

Se crea un nuevo objeto Video en la base de datos con los datos del formulario, la ID del usuario y la ubicación del archivo de video en el servidor

Se intenta autenticar la aplicación de Google y subir el video a YouTube con los datos del diccionario

Si se sube el video correctamente:

Se guarda el ID del video de YouTube en el objeto Video

Renderizar la página de éxito con el ID del video de YouTube

De lo contrario, si ocurre algún error:

Se elimina el objeto Video creado

Se renderiza una página de error con el mensaje de error

Si el método de la solicitud no es POST:
Se crea el formulario para subir videos vacíos
Se renderiza la página de carga de video con el
formulario

Para finalizar, de haberse subido el video con éxito, se le muestra al usuario la página de la Figura 9.

Video subido con éxito

El video se ha subido con éxito, ahora debe procesarse en los servidores de YouTube. El procesamiento del video puede tardar unos minutos dependiendo del tamaño del archivo.

Link del video: <https://www.youtube.com/watch?v=Zr5lllg8g94>

Para insertar este video en tu sitio web, utiliza el siguiente código HTML:

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/Zr5lllg8g94" title="YouTube video player" frameborder="0" allow="accelerometer; clipboard-write; e
```

Cambiar tamaño del video embebido

Altura del video: 315 pixeles

Ancho del video: 560 pixeles

☒ Mantener relación de aspecto 16:9



Figura 9. Vista de una carga de video a YouTube exitosa

En esta, se le notifica al usuario que el video se ha subido a YouTube, pero requiere un procesamiento adicional en los servidores de YouTube antes de poder reproducirse, lo cual puede tomar algunos minutos dependiendo del tamaño del video. Además, se le entrega el link del video para poder acceder a él o copiarlo si lo desea. Por último, se entrega el código HTML para poder insertar el video en un sitio web si así lo desea, junto con controles para modificar su tamaño y una vista previa del video con las dimensiones seleccionadas.

4.3. Desarrollo del sistema de autenticación

Se decide utilizar Microsoft Entra ID para facilitar el manejo de las cuentas, ya que se utilizan las mismas credenciales que el correo institucional, evitando que el usuario tenga que crear una cuenta adicional para acceder al portal. Al igual que en el caso de la API de YouTube, esta herramienta requiere que se cree una aplicación por la cual se hagan las solicitudes. Esta aplicación es la que permitirá al usuario identificarse con su identidad de Microsoft, y permitirá saber al portal que la persona que está accediendo es quien Microsoft dice que es.

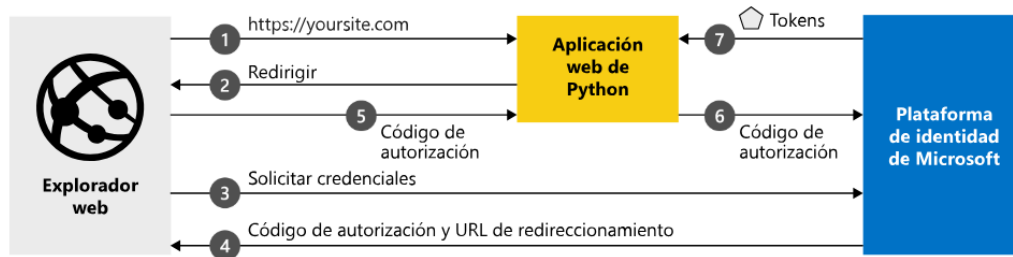


Figura 10. Flujo de información para poder iniciar sesión con los servidores de Microsoft

Para la implementación se utiliza un tutorial de Microsoft ([Microsoft Learn, s.f.](#)), el cual muestra los pasos para aplicar un inicio de sesión único en una aplicación de Django.

La creación de una aplicación de Microsoft Entra ID se hace a través del portal de Azure (portal.azure.com). Aquí se escoge el servicio de Microsoft Entra ID (ver

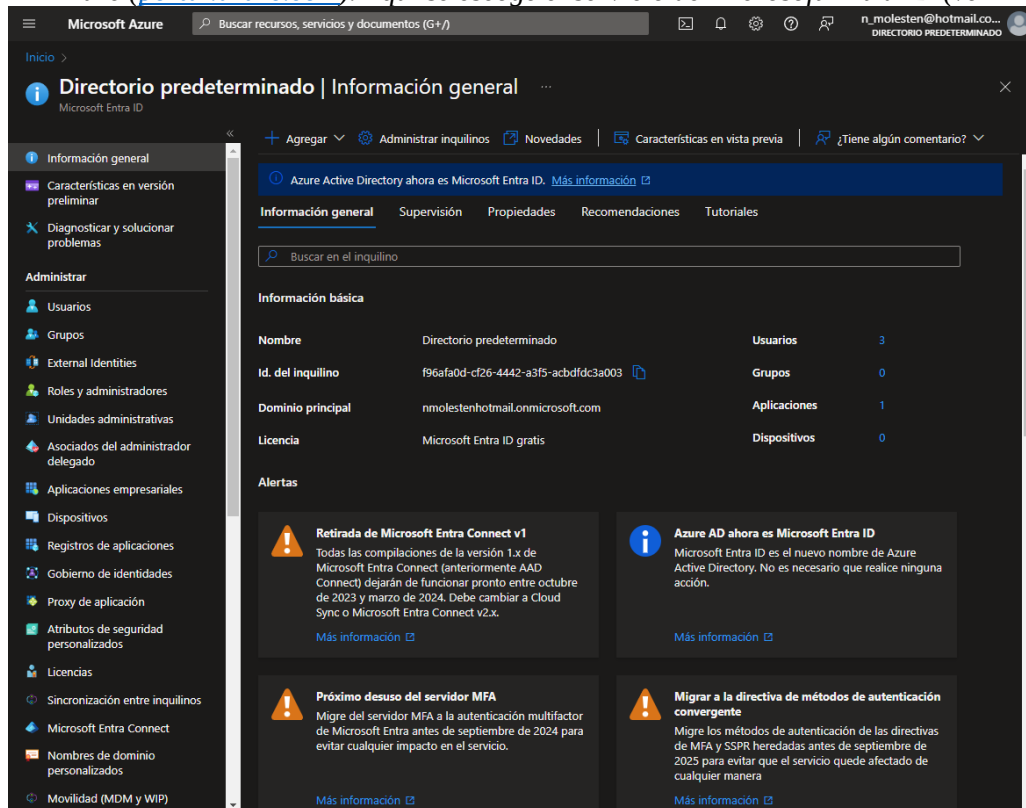


Figura 11), y dentro de este se debe crear un inquilino pulsando el botón “Administrar inquilinos”. Un inquilino representa un departamento dentro de una organización con sus propias aplicaciones y permisos para los usuarios que contenga, por lo que, en este caso, este inquilino se encargará de manejar el acceso a las funciones del portal web.

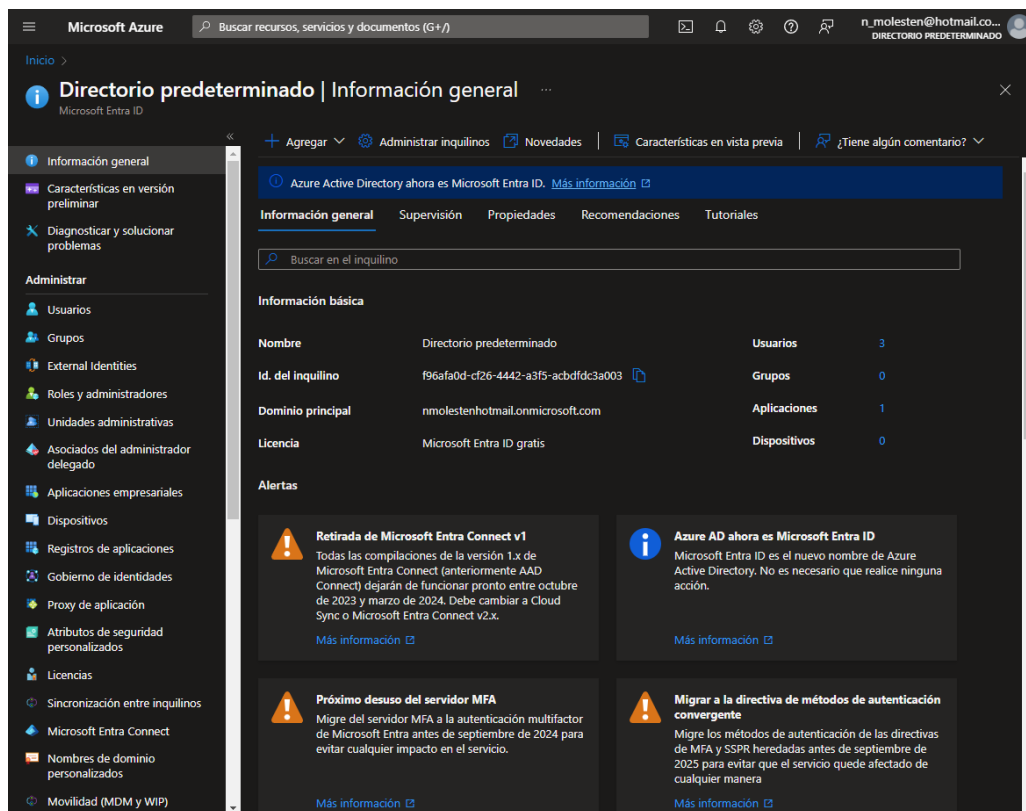


Figura 11. Pantalla principal de Microsoft Entra ID

Una vez dentro del inquilino, se debe ir a Registro de aplicaciones, aquí es donde se creará la aplicación que permitirá el acceso a la cuenta de Microsoft a través del portal con el botón de “Nuevo registro”. En esta pantalla (ver Figura 12) se le debe asignar un nombre a la aplicación, y se deberá escoger qué tipo de cuentas se necesita admitir, por lo que, en este caso, se selecciona “Sólo cuentas de este directorio organizativo”, que permitirá el acceso únicamente de cuentas que pertenezcan a este inquilino, es decir, las que el administrador registre o invite. Opcionalmente, también se pide una URI de redirección, a la cual se redirigirá al navegador una vez iniciada la sesión. En este caso, se elegirá la ruta /auth/redirect como el sitio al cual dirigirse cuando se inicie sesión, ya que esta URL redirige al usuario autenticado a la URL ‘index’ (por defecto en la librería ms_identity_web) dentro de Django, que en este caso es el home.

Registrar una aplicación

* Nombre

Nombre para mostrar accesible por los usuarios de esta aplicación. Se puede cambiar posteriormente.

portal-de-prueba-youtube

Tipos de cuenta compatibles

¿Quién puede usar esta aplicación o acceder a esta API?

☒ Solo cuentas de este directorio organizativo (solo de Directorio predeterminado: inquilino único)

☐ Cuentas en cualquier directorio organizativo (cualquier inquilino de id. de Microsoft Entra - multiinquilino)

☐ Cuentas en cualquier directorio organizacional (cualquier inquilino del id. de Microsoft Entra - multiinquilino) y cuentas personales de Microsoft (por ejemplo, Skype, Xbox)

☐ Solo cuentas personales de Microsoft

[Ayudarme a elegir...](#)

URI de redirección (opcional)

Devolveremos la respuesta de autenticación a esta dirección URI después de autenticar correctamente al usuario. Este dato es opcional y se puede cambiar más tarde, pero se necesita un valor para la mayoría de los escenarios de autenticación.

Web

https://localhost:8000/auth/redirect

Registre una aplicación en la que esté trabajando aquí. Integre aplicaciones de la galería y otras aplicaciones de fuera de la organización agregándolas desde [Aplicaciones empresariales](#).

Al continuar, acepta las directivas de la plataforma Microsoft.

Registrar

Figura 12. Pantalla de nuevo registro de aplicaciones

Una vez dentro de la aplicación, en el panel de autenticación, se requiere establecer una URL de cierre de sesión, a la cual se enviará al usuario una vez que desee terminar su sesión.

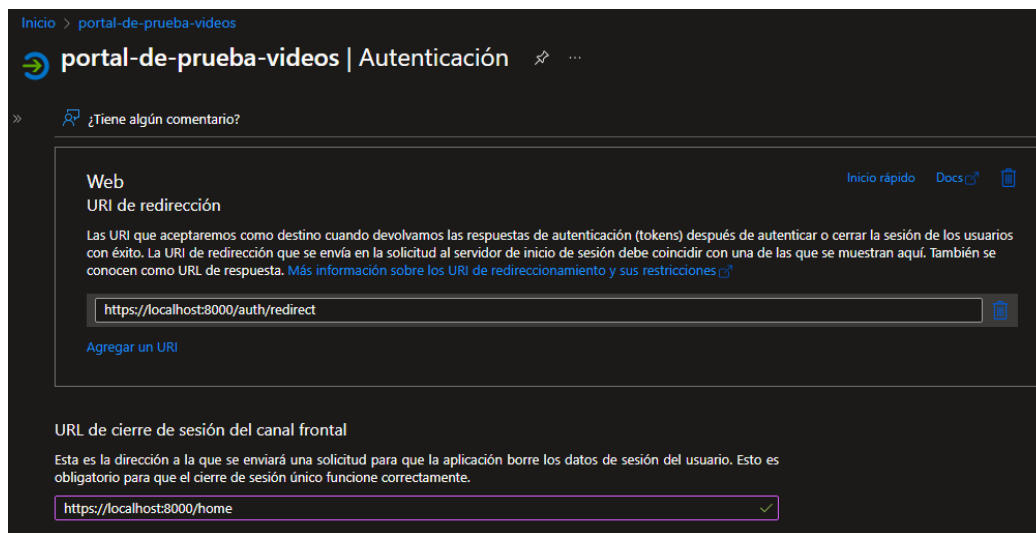


Figura 13. Panel de autenticación de la aplicación

Un detalle a considerar, es que estas aplicaciones solo aceptan direcciones HTTPS para redirigir al usuario, por lo que aquí entran en juego las librerías `django-extensions`, `werkzeug`, y `pyopenssl`, que son las que nos permitirán levantar el sitio en HTTPS vía el comando `runserver_plus`. Para esto, primero hay que agregar `django-extensions` a la lista `INSTALLED_APPS` del archivo `settings.py` del proyecto, luego basta con el comando en consola:

```
>python manage.py runserver_plus --cert-file nombre_certificado.crt
```

Este comando levantará el servidor de Django, a la vez que crea una llave y un certificado autofirmado, que solo sirve para propósitos internos. De esta manera, se accederá al sitio vía HTTPS.

Por defecto, sólo el usuario que crea la aplicación tendrá acceso a esta. Para añadir nuevos usuarios, el administrador de la aplicación debe ir al panel de Usuarios dentro de la administración del inquilino, y añadirlos de manera manual o vía una invitación por correo. Una vez que acepten la invitación los usuarios podrán autenticarse a través de esta aplicación, y por lo tanto, iniciar sesión en el sitio web. La primera vez que lo hagan, se les pedirá permiso para acceder a los datos básicos de su perfil, como se muestra en la Figura 14.



matias.marquez.14@sansano.usm.cl

Permisos solicitados

portal-de-prueba-videos

[sin comprobar](#)

Ni Microsoft ni su organización han publicado esta aplicación.

Esta aplicación quiere:

- ✓ Mantener el acceso a los datos a los que se le ha concedido acceso
- ✓ Ver su perfil básico

La aceptación de estos permisos implica que permite a esta aplicación usar sus datos, tal y como se especifica en las condiciones de servicio y la declaración de privacidad. **El publicador no ha proporcionado los vínculos a las condiciones para que pueda consultarlos.** Puede cambiar dichos permisos en <https://myapps.microsoft.com>. [Mostrar detalles](#)

¿Le parece sospechosa esta aplicación? [Haga clic aquí para informar de ella.](#)

Cancelar

Aceptar

Figura 14. Ventana que pedirá permisos al usuario la primera vez que inicie sesión en el sitio

Una vez creada la aplicación, se requiere crear un secreto de cliente, que es lo que permitirá al sitio web conectarse con la aplicación y poder iniciar sesión a través de las cuentas de Microsoft de forma segura. Por lo que se ingresará al panel de Certificados y secretos, y se creará un nuevo secreto de cliente, cuyo valor debe ser guardado para poder utilizar la aplicación, ya que se mostrará brevemente luego de ser creada.

Dentro de los archivos del sitio web, se debe crear el archivo `aad.config.json` (ver el Anexo 15). Aquí se deben completar algunas casillas para que el sitio se comuniquen con la aplicación. En el campo `client_id` se debe colocar la ID de la aplicación, en

client_credential se debe colocar el secreto de cliente que se creó para la aplicación, y en *authority* se debe colocar la ID del inquilino que hospeda la aplicación.

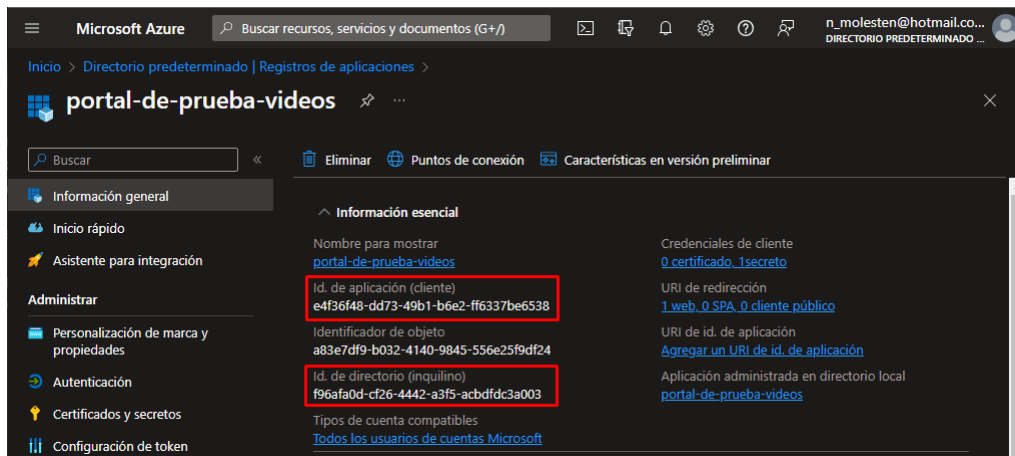


Figura 15. ID de aplicación e ID de inquilino en el portal de Azure

Para hacer funcionar esta librería con el sitio web, es necesario añadir algunas líneas de código al archivo `settings.py` después de la lista `MIDDLEWARE`:

```
from ms_identity_web.configuration import AADConfig
from ms_identity_web import IdentityWebPython
AAD_CONFIG = AADConfig.parse_json(file_path='aad.config.json')
MS_IDENTITY_WEB = IdentityWebPython(AAD_CONFIG)
ERROR_TEMPLATE = 'auth/{}.html'
MIDDLEWARE.append('ms_identity_web.django.middleware.MsalMiddlewar
e')
```

Estas permitirán a la aplicación leer el archivo `aad.config.json`, que contiene los datos para poder utilizar la aplicación de Microsoft Entra ID.

Y a `urls.py` de la aplicación se le agrega lo siguiente para que se agreguen las rutas `/auth/...` incluidas en la librería:

```
from ms_identity_web.django.msal_views_and_urls import MsalViews
from django.conf import settings
msal_urls = MsalViews(settings.MS_IDENTITY_WEB).url_patterns()
```

Además de añadirle a la lista `urlpatterns` la línea para que la aplicación las reconozca:

```
path(f'{settings.AAD_CONFIG.django.auth_endpoints.prefix}/',
include(msal_urls)),
```

Con esto quedan establecidas las URLs de autenticación que ocupa la librería, que utiliza

la ruta /auth (e.g. la ruta /auth/sign_in para iniciar sesión, y la ruta /auth/sign_out para cerrar sesión) para sus funciones.

4.4. Desarrollo del código para editar información o eliminar videos

Para que los usuarios puedan ver los videos que han subido por su parte a YouTube, se crea la página de Mis Videos, cuya vista *my_videos* puede ser leída en el Anexo 14. En esta página, cada usuario podrá ver cuadros con los videos que ha subido, acompañados de su título, descripción, link del video, fecha de subida, y una miniatura del video. Además, esta información estará acompañada de 3 botones: editar video, eliminar video, y código para insertar video; como se muestra a continuación:

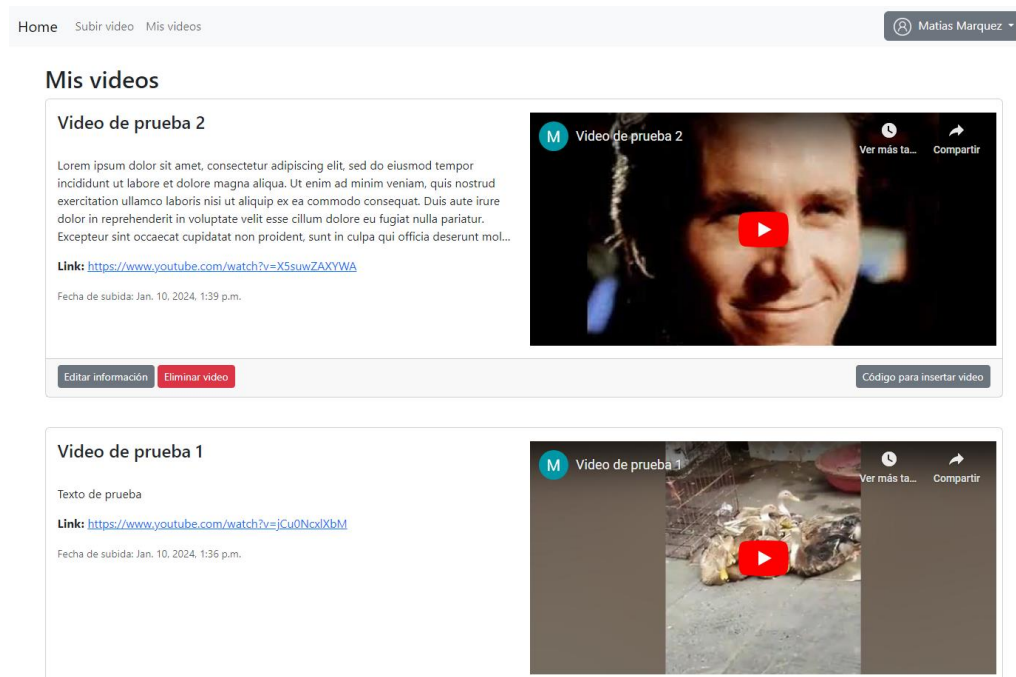


Figura 16. Página de Mis Videos

A continuación se muestra una breve descripción junto a una imagen de lo que muestra cada botón:

- Editar información: envía al usuario a un formulario en donde aparecen el título y la descripción en cuadros de texto editables, este estará controlado por la vista *edit_video* (ver el Anexo 14). Una vez que se editan y se envía el formulario, se actualizan los datos tanto en la base de datos del sitio como en YouTube.

Editar información de video

Título:

Video de prueba 2

Descripción:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Enviar

Figura 17. Formulario de edición de la información de un video

- Eliminar video: abre un cuadro de confirmación para comprobar que no se trata de un accidente, este estará controlado por la vista *delete_video* (ver el Anexo 14). Si el usuario confirma la eliminación, se eliminará el video de la base de datos y de YouTube.

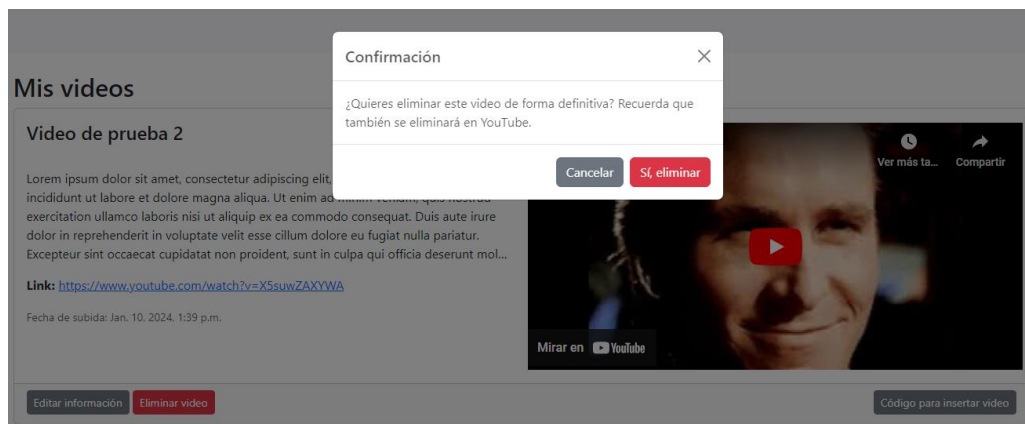


Figura 18. Pantalla de confirmación para eliminar video

- Código para insertar video: hace aparecer el texto HTML necesario para insertar el video en un sitio web debajo del cuadro del video.

Mis videos

Video de prueba 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mol...

Link: <https://www.youtube.com/watch?v=X5suwZAXYWA>

Fecha de subida: Jan. 10, 2024, 1:39 p.m.

[Editar información](#) [Eliminar video](#) [Código para insertar video](#)

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/X5suwZAXYWA" title="YouTube video player" frameborder="0" allow="accelerometer; clipboard-write;
4
```

Figura 19. Botón que muestra el código HTML para insertar el video dentro de una página web

Para la edición de la información de los videos se requiere crear un formulario que contenga los campos de título y descripción ya poblados con la información original. Luego de editar los campos, se envía el formulario para actualizar los datos en la base de datos, y enviar la petición a la API de YouTube para hacer lo mismo con el video.

Para la eliminación de videos simplemente se tiene el botón de “Eliminar video”, el cual al ser apretado contiene un cuadro de confirmación. Si el usuario consiente la operación, se enviará la petición de eliminación a la API de YouTube, y de ser la eliminación exitosa, se eliminará el video de la base de datos.

Para que además de borrarse la entrada del video en la base de datos, se borre el archivo de video también, se debe agregar `django-cleanup` la lista `INSTALLED_APPS`, por lo que luego de esta se debe agregar:

```
INSTALLED_APPS.append('django_cleanup.apps.CleanupConfig')
```

4.5. Registro de actividades

Para obtener un registro de actividades, se recurre al módulo de Python, logging. Este módulo permite crear archivos de registro (.log) en donde se muestren las actividades realizadas por cada usuario a la fecha de su realización.

Para configurar logging dentro de Django, se recurre a la documentación de Django ([Django, s.f.](#)). De esta forma se configurarán 2 loggers: `site-logger`, que hará un registro de las actividades del usuario dentro del sitio, y `MsalViewsLogger`, que es el nombre que se le da al logger dentro de la librería MSAL, por lo que este tendrá un registro de los inicios y términos de sesión. Ambos escribirán en el mismo archivo `main_log.log`, en el formato “[fecha] [nivel]: [mensaje]”. Por lo que habrá que agregar al archivo

settings.py del proyecto lo correspondiente al Anexo 16. Así se tendrá el mensaje que se quiere registrar junto con la fecha de este. (revisar negrita)

Para utilizar el logger creado, se deben añadir las líneas:

```
import logging
logger = logging.getLogger('site-logger')
```

Esto permitirá crear registros usando el logger “site-logger” con la palabra “logger” de la siguiente forma:

```
logger.debug("Ejemplo")
```

Esto creará un log de nivel DEBUG con el mensaje “Ejemplo” en el archivo de registro indicado.

A continuación se muestran los registros que se tendrán en cuenta, cada uno irá en la acción correspondiente, y se muestra un ejemplo de cómo se verán:

- Subir un video: se muestran el nombre de usuario e ID del usuario que sube el video, junto con la ID del video subido. Este registro se creará una vez que se haya terminado de subir el video con éxito.

```
[2023-12-01 11:41:18,667] DEBUG: El usuario Matias Marquez
(60819526-72a4-4f75-9f56-d83ce6d32540) ha subido el video con ID: -
FA49o00ePg.
```

- Editar un video: se muestran el nombre de usuario e ID del usuario que edita el video, junto con la ID del video editado. Este registro se creará una vez que se haya editado el video en YouTube con éxito.

```
[2023-12-01 11:47:15,588] DEBUG: El usuario Matias Marquez
(60819526-72a4-4f75-9f56-d83ce6d32540) ha editado la información
del video con ID: -FA49o00ePg.
```

- Eliminar un video: se muestran el nombre de usuario e ID del usuario que elimina el video, junto con la ID del video eliminado. Este registro se creará una vez que se haya eliminado el video en YouTube con éxito.

```
[2023-12-01 11:47:37,576] DEBUG: El usuario Matias Marquez
(60819526-72a4-4f75-9f56-d83ce6d32540) ha eliminado el video con ID:
-FA49o00ePg.
```

- Inicio de sesión: este logger está controlado por la librería MSAL, y registra el redireccionamiento al sitio de ingreso de Microsoft, y luego de ingresar, la redirección de vuelta al portal. Ambos registros se realizan antes de la redirección.

```
[2023-12-01 11:39:39,695] DEBUG: auth/sign_in: request received.  
will redirect browser to login
```

```
[2023-12-01 11:40:03,332] DEBUG: auth/redirect: request received.  
will process params
```

- Cierre de sesión: este logger está controlado por la librería MSAL, y registra la petición de cierre de sesión junto con el nombre de usuario, y luego la eliminación de la sesión. Ambos registros se realizan antes de realizar la acción que están registrando.

```
[2023-12-01 11:38:14,258] DEBUG: auth/sign_out: signing out  
username: Matias Marquez
```

```
[2023-12-01 11:38:38,864] DEBUG: auth/post_sign_out: clearing  
session for username: Matias Marquez
```

5. RESULTADOS

El sitio web logrado a través de los pasos mencionados para su desarrollo, resulta en un portal que cumple con todos los objetivos propuestos para este proyecto, ya que permite subir los videos a una única cuenta de YouTube de manera simplificada, dentro de una interfaz sencilla, dentro de la cual se puede encontrar el contenido ya subido dentro del portal, junto con herramientas para eliminar o editar la información de dichos videos. Junto con las funcionalidades para administrar los videos, también se muestran el enlace del video, y el código para poder insertarlos dentro de otro sitio web.

La autenticación se logra hacer a través de los servidores de Microsoft, permitiendo que las funciones del portal y su contenido queden restringidas solamente a los usuarios que tengan permitido iniciar sesión en el sitio. Además, cada usuario será responsable exclusivamente por el contenido que ha subido, ya que no le aparecerán videos de otros usuarios.

El archivo de registro permite guardar información sobre las actividades importantes dentro del sitio que realicen los usuarios. Este siempre es un aspecto importante en caso de que se encuentren comportamientos inusuales o maliciosos, ya que permite poder actuar ante dichas conductas para mantener la integridad del canal de YouTube.

6. CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

El desarrollo del portal web destinado a permitir a los profesores cargar videos de manera individual a una cuenta centralizada de YouTube resulta en una solución práctica y eficiente para abordar la necesidad de centralizar y gestionar el contenido de sus videos de manera más efectiva. A lo largo de este proyecto, se ha logrado una serie de hitos significativos que aportan tanto a la comodidad de los profesores como a la administración central de los videos que se quieran subir.

La implementación exitosa de la autenticación mediante cuentas institucionales garantiza un acceso seguro y autorizado al portal, ofreciendo a los profesores una plataforma confiable para administrar sus videos. La interfaz intuitiva y amigable facilita la carga y gestión de los videos,

En conclusión, el presente trabajo proporciona una solución efectiva para la gestión centralizada de videos educativos para una institución educativa.

6.2. Trabajo futuro

Para darle más funcionalidad al sitio, sería deseable que el usuario pudiese crear listas de reproducción que contengan todas las clases de una asignatura dentro de un semestre, de manera que queden agrupados para compartir en un solo enlace. Además, el orden de los videos podría quedar mejor organizado si es que se tienen muchos videos.

Otra medida de organización que es deseable añadir, es agregar paginación a los videos cuando se tengan muchos, junto con una lista de “Ordenar por” para poder disponerlos de forma organizada según sus distintas propiedades. Por último, una casilla para buscar videos según su nombre (título) o asignatura.

En términos de mejorar la experiencia del usuario, sería ideal añadir una barra de carga para ver el progreso del video a medida que es subido a YouTube, esto permite al usuario saber el tiempo aproximado que va a tomar la tarea, y tener conocimiento del porcentaje actual del video que se ha cargado.

Además, se puede añadir un tiempo de inactividad, tras el cual se le cierre la sesión automáticamente, esto como medida de seguridad recomendada. Esto siempre y cuando no esté la carga de un video en curso.

REFERENCIAS

1. Django (s.f.). *The web framework for perfectionists with deadlines* | Django. Recuperado el 18 de Marzo de 2024 de <https://www.djangoproject.com/>
2. Google for Developers (13 de Septiembre de 2023). *YouTube Data API (v3) – Calculadora de cuota* | YouTube Data API | Google for Developers. https://developers.google.com/youtube/v3/determine_quota_cost
3. Google Support (s.f.). *Servicios de APIs de YouTube: formulario de auditoría y de aumento de cuotas - Ayuda de YouTube*. Recuperado el 30 de Junio de 2023 de https://support.google.com/youtube/contact/yt_api_form
4. YouTube Help (s.f.). *Upload videos longer than 15 minutes – Computer – YouTube Help*. Recuperado el 30 de Junio de 2023 de <https://support.google.com/youtube/answer/71673>
5. Logging facility for Python (s.f.). *logging – Logging facility for Python – Python 3.12.2 documentation*. Recuperado el 18 de Marzo de 2024 de <https://docs.python.org/3/library/logging.html>
6. OAuth (s.f.). *OAuth 2.0 – OAuth*. Recuperado el 18 de Marzo de 2024 de <https://oauth.net/2/>
7. Google for Developers (2 de Diciembre de 2023). *Subir un video* | YouTube Data API | Google for Developers. https://developers.google.com/youtube/v3/guides/uploading_a_video
8. Microsoft Learn (s.f.). *Sign in users to a Django web app*. Recuperado el 15 Octubre de 2023 de <https://learn.microsoft.com/en-us/training/modules/msid-django-web-app-sign-in/>
9. Django (s.f.). *Logging* | Django Documentation | Django. Recuperado el 14 de Noviembre de 2023 de <https://docs.djangoproject.com/en/5.0/topics/logging/>

ANEXO

- Anexo 1

Requirements.txt.

```
django == 4.2.7
django-extensions == 3.2.3
werkzeug == 3.0.1
pyopenssl == 23.3.0
django-cleanup == 8.0.0
google-api-python-client == 2.109.0
google-auth-oauthlib == 1.1.0
msal == 1.25.0
ms-identity-web == 0.0.4
```

- Anexo 2

Modelo de video. Incluye los datos que tendrá que dar el usuario: título (obligatorio), descripción (opcional), archivo de video (obligatorio). Junto con los que se llenan de forma automática: cargador del video, ID de YouTube, y fecha de subida.

```
import os
from django.db import models
from django.utils import timezone
import datetime

def video_filename(instance, filename):
    today = datetime.date.today()
    timestamp = timezone.now()
    # Mantiene la extensión del archivo original
    filename_base, filename_ext = os.path.splitext(filename)

    # Calcula la ID del video
    videos_today = Video.objects.filter(date__date=today)
    if videos_today.exists():
        last_video = videos_today.order_by('-id').first()
        new_id = last_video.id + 1
    else:
        new_id = 1

    # Formatea el nombre del archivo con fecha, hora e ID.
    (2023_11_25_16_11-001)
    id = f"{new_id:03d}"
```

```

return
f"videos/{today.year:04d}_{today.month:02d}_{today.day:02d}_{timestamp.hour:02d}_{timestamp.minute:02d}-{id}{filename_ext}"

class Video(models.Model):
    title = models.CharField(max_length=200, blank=False)
    description = models.TextField(blank=True)
    video_file = models.FileField(upload_to=video_filename,
blank=False) # Llama a la función que formatea el nombre
    uploader = models.CharField(max_length=200, null=False)
    youtube_id = models.CharField(max_length=50, null=True)
    date = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return self.title

```

- Anexo 3

Archivo urls.py de main_app, mostrando las URLs pertinentes.

```

from django.urls import path
from . import views
from django.views.generic.base import RedirectView

urlpatterns = [
    path('', RedirectView.as_view(pattern_name='index'),
name='redirect-to-home'),
    path('home', views.home, name='index'),
    path('upload_video', views.upload_video, name='upload-video'),
    path('my_videos', views.my_videos, name='my-videos'),
    path('edit_video/<video_id>', views.edit_video, name='edit-
video'),
    path('delete_video/<video_id>', views.delete_video,
name='delete-video'),
]

```

- Anexo 4

Archivo home.html

```

<!doctype html>
<html lang="en">
<head>

```

```

    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>YouTube Uploader</title>
    <!-- Bootstrap -->
                                                                    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootst
rap.min.css"          rel="stylesheet"          integrity="sha384-
4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGChEKRQN+PtmoHDEXuppvnDJzQIu9"
crossorigin="anonymous">
    <!-- jQuery-->
                                                                    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min
.js"></script>
</head>

<body>
{% include 'main_app/navbar.html' %}

{% if messages %}
    {% for message in messages %}
        <div class="alert alert-warning alert-dismissible fade show"
role="alert">
            {{ message }}
            <button type="button" class="btn-close" data-bs-
dismiss="alert" aria-label="Close"></button>
        </div>
    {% endfor %}
{% endif %}
<br>
<div class="container-lg">
    {% block content %}
        <h2>Home</h2>
        <br>
        <br>
        <!-- Verificador de autenticación de Microsoft -->
        {% if request.identity_context_data.authenticated %}
            Bienvenido, {{ request.identity_context_data.username
}}. <br>
            <a href="{% url 'sign_out' %}">Cerrar sesión</a>
        {% else %}
            No estás logueado.

```

```

        Por favor <a href="{% url 'sign_in' %}">inicia
sesión</a> para acceder a las funcionalidades del sitio.
        {% endif %}
    {% endblock %}

</div>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap
.bundle.min.js"
integrity="sha384-
HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
crossorigin="anonymous"></script>
</body>
</html>

```

- Anexo 5

Archivo navbar.html

```

{% load static %}

<nav class="navbar navbar-expand-lg sticky-top bg-body-tertiary">
    <div class="container-fluid">
        <!-- El botón de Home debe redireccionar a 'index', ya que
ese es el nombre que le da ms_identity_web a la URL de redirección
-->
        <a class="navbar-brand" href="{% url 'index' %}">Home</a>
        <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse"
id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'upload-video'
%}">Subir video</a>
                </li>

                {% if request.identity_context_data.authenticated %}
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'my-videos' %}">Mis
videos</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

```

```

        </li>
        {% endif %}
    </ul>

    <!-- Botón de inicio de sesión -->
    <ul class="navbar-nav me-2 mb-2 mb-lg-0">
        <li class="nav-item">
            {% if request.identity_context_data.authenticated
%}

                <div class="dropdown">
                    <button class="btn btn-secondary dropdown-toggle"
type="button" data-bs-toggle="dropdown" aria-expanded="false">
                        
                        {{ request.identity_context_data.username
}}

                    </button>
                    <ul class="dropdown-menu dropdown-menu-end"
style="padding: 4px 0px; max-width: 100%;">
                        <li><a class="dropdown-item " href="{% url
'sign_out' %}" style="padding: 6px 12px;">
                            
                            Cerrar sesión
                        </a></li>
                    </ul>
                </div>
            {% else %}
                <a class="btn btn-secondary" href="{% url
'sign_in' %}">
                    
                    Iniciar sesión
                </a>
            {% endif %}
        </li>
    </ul>
</div>
</div>
</nav>

```

- Anexo 6

Archivo upload_video.html

```
{% extends 'main_app/home.html' %}

{% block content %}
    {% if request.identity_context_data.authenticated %}
        <h2>Subir un video</h2>
        <div style="max-width: 500px;">
            <form id="upload-form" method='post'
enctype='multipart/form-data'>
                {% csrf_token %}
                {{ form.as_p }}

                <div class="d-flex">
                    <button type='submit' class="btn btn-primary me-3" id="submitBtn" onclick="processing()">Enviar</button>

                    <div style="display: none;" class="align-self-center" id="video_loading_text">
                        <div class="spinner-border spinner-border-sm" role="status"></div> Cargando video...
                    </div>
                </div>

                <button type='button' class="btn btn-primary"
id="uploadingBtn" disabled style="display: none;">
                    <span class="spinner-border spinner-border-sm"
aria-hidden="true"></span>
                    <span role="status">Subiendo...</span>
                </button>
            </form>
        </div>

        <!-- Carga mientras se sube el video -->
        <script>
            function processing(){
                if ($('#upload-form')[0].checkValidity()) {
                    $('#submitBtn').hide()
                    $('#uploadingBtn').show()
                }
            }
        </script>
```

```

    {% else %}
        <h2>Por favor inicie sesión para poder subir un video.</h2>
        <a href="{% url 'sign_in' %}">Iniciar sesión</a>
    {% endif %}
{% endblock %}

```

- Anexo 7

my_videos.html

```

{% extends 'main_app/home.html' %}

{% block content %}
    {% if request.identity_context_data.authenticated %}
        <h2>Mis videos</h2>
        {% if videos_list %}
            {% for video in videos_list %}
                <div class="card mb-3">
                    <div class="row g-0">
                        <!-- Información-->
                        <div class="col-md-6">
                            <div class="card-body">
                                <h4>
                                    {{ video.title }}
                                </h4>
                                <br>
                                <p style="
                                    overflow: hidden;
                                    display: -webkit-box;
                                    -webkit-line-clamp: 5;
                                    -webkit-box-orient: vertical;
                                    height: 100%;">
                                    {{ video.description }}
                                </p>
                                <p><strong>Link:</strong> <a
                                    href="https://www.youtube.com/watch?v={{ video.youtube_id
                                    }}">https://www.youtube.com/watch?v={{ video.youtube_id }}</a></p>
                                <p class="card-text"><small
                                    class="text-body-secondary">Fecha de subida: {{ video.date
                                    }}</small></p>
                            </div>
                        </div>
                    </div>
                </div>
            {% endfor %}
        {% else %}
            <p>No tienes videos.</p>
        {% endif %}
    {% else %}
        <p>Por favor inicie sesión para poder subir un video.</p>
    {% endif %}
{% endblock %}

```

```

        <!-- Video embebido -->
        <div class="col-md-6 d-flex align-items-
center" style="max-height: 350px;">
            <div class="ratio ratio-16x9 mx-2"
style="max-height: 90%;">
                <iframe
src="https://www.youtube.com/embed/{{ video.youtube_id }}?rel=0"
title="YouTube video player" allow="accelerometer; clipboard-
write; encrypted-media; gyroscope; picture-in-picture; web-share"
allowfullscreen></iframe>
            </div>
        </div>
    </div>

    <!-- Botón de edición, eliminación y panel de
confirmación -->
    <div class="card-footer text-body-secondary">
        <div class="d-flex flex-wrap">
            <a class="btn btn-secondary btn-sm me-
1 mb-1" href="{% url 'edit-video' video.youtube_id %}">
                Editar información
            </a>
            <button type="button" class="btn btn-
danger btn-sm me-1 mb-1" data-bs-toggle="modal" data-bs-
target="#deleteConfirmation-{{ video.youtube_id }}">
                Eliminar video
            </button>
            <div class="ms-auto">
                <button class="btn btn-secondary
btn-sm" type="button" data-bs-toggle="collapse" data-bs-
target="#collapseExample-{{ video.youtube_id }}" aria-
expanded="false" aria-controls="collapseExample-{{ forloop.counter
 }}">
                    Código para insertar video
                </button>
            </div>
        </div>
    </div>

    <!-- Panel de confirmación -->
    <div class="modal fade"
id="deleteConfirmation-{{ video.youtube_id }}" tabindex="-1" aria-
labelledby="deleteConfirmation-{{ video.youtube_id }}" aria-
hidden="true">

```

```

        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h1 class="modal-title fs-
5">Confirmación</h1>
                    <button type="button"
class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
                </div>
                <div class="modal-body">
                    ¿Quieres eliminar este
video de forma definitiva? Recuerda que también se eliminará en
YouTube.
                </div>
                <div class="modal-footer">
                    <button type="button"
class="btn btn-secondary" data-bs-
dismiss="modal">Cancelar</button>
                    <a class="btn btn-danger"
href="{% url 'delete-video' video.youtube_id %}" id="deleteBtn-{{
video.youtube_id }}" onclick="startLoading('{{ video.youtube_id
}}')">Sí, eliminar</a>
                    <!-- Botón de cargando -->
                    <button class="btn btn-
danger" type="button" disabled id="deleteLoading-{{
video.youtube_id }}" style="display: none;">
                        <span class="spinner-
border spinner-border-sm" aria-hidden="true"></span>
                        <span
role="status">Eliminando...</span>
                    </button>
                </div>
            </div>
        </div>
    </div>
</div>
<!-- Código para embeber -->
<div class="collapse" id="collapseExample-{{
video.youtube_id }}">
    <div class="card-footer text-body-
secondary">
        <pre style="margin:
0;"><code>&lt;iframe width="560" height="315"

```

```

src="https://www.youtube.com/embed/{{ video.youtube_id }}"
title="YouTube video player" frameborder="0" allow="accelerometer;
clipboard-write; encrypted-media; gyroscope; picture-in-picture;
web-share" allowfullscreen></code></pre>
    </div>
  </div>
</div>
<br>
{% endfor %}
{% else %}
<br>
<h4>Aún no ha subido ningún video.</h4>
{% endif %}
{% else %}
<h2>Por favor inicie sesión para poder ver sus
videos.</h2>
<a href="{% url 'sign_in' %}">Iniciar sesión</a>
{% endif %}

<!-- Carga de la eliminación de video -->
<script>
  function startLoading(videoId) {
    $('#deleteBtn-' + videoId).hide()
    $('#deleteLoading-' + videoId).show()
  }
</script>
{% endblock %}

```

- Anexo 8

edit_video.html

```

{% extends 'main_app/home.html' %}

{% block content %}
  {% if request.identity_context_data.authenticated %}
    <h2>Editar información de video</h2>
    <div style="max-width: 500px;">
      <form id="edit-form" method="post">
        {% csrf_token %}
        {{ form.as_p }}
      </form>
    </div>
  {% else %}
    <h2>Por favor inicie sesión para poder ver sus
    videos.</h2>
    <a href="{% url 'sign_in' %}">Iniciar sesión</a>
  {% endif %}

```

```

        <button type='submit' class="btn btn-primary"
id="submitBtn" onclick="processing()">Enviar</button>
        <button type='button' class="btn btn-primary"
id="uploadingBtn" disabled style="display: none;">
        <span class="spinner-border spinner-border-sm"
aria-hidden="true"></span>
        <span role="status">Editando...</span>
    </button>
    <a class="btn btn-secondary" href="{% url 'my-
videos' %}">Cancelar</a>
</form>
</div>

<!-- Carga mientras se procesa la solicitud -->
<script>
    function processing(){
        if ($('#edit-form')[0].checkValidity()) {
            $('#submitBtn').hide()
            $('#uploadingBtn').show()
        }
    }
</script>
{% else %}
    <h2>Por favor inicie sesión para poder editar un video.</h2>
    <a href="{% url 'sign_in' %}">Iniciar sesión</a>
{% endif %}
{% endblock %}

```

- Anexo 9

success.html

```

{% extends 'main_app/home.html' %}

{% block content %}
    <h2>Video subido con éxito</h2>
    <br>
    <p>El video se ha subido con éxito, ahora debe procesarse en los
servidores de YouTube. El procesamiento del video puede tardar unos
minutos dependiendo del tamaño del archivo.</p>
    Link del video: <a href="https://www.youtube.com/watch?v={{
video_id }}" target="_blank">https://www.youtube.com/watch?v={{
video_id }}</a>
    <br><br>

```

Para insertar este video en tu sitio web, utiliza el siguiente código HTML:

```
<div class="px-2 pt-2 border rounded-2">
    <pre class="mb-2"><code id="embedCode">&lt;iframe
width="560" height="315" src="https://www.youtube.com/embed/{{
video_id }}" title="YouTube video player" frameborder="0"
allow="accelerometer; clipboard-write; encrypted-media; gyroscope;
picture-in-picture; web-share"
allowfullscreen&gt;&lt;/iframe&gt;</code></pre>
</div>
<br>

<!-- Sliders tamaño -->
<div class="d-flex justify-content-start">
    <div class="border rounded-2 p-3" style="width: 50%;">
        <h5>Cambiar tamaño del video embebido</h5>

        <label for="videoHeight" class="form-label mb-0 mt-1">Altura del video: </label> <span id="heightNum">315</span>
        pixeles

        <input type="range" class="form-range" min="200"
max="720" value="315" id="videoHeight">

        <label for="videoWidth" class="form-label mb-0">Ancho
del video: </label> <span id="widthNum">540</span> pixeles
        <input type="range" class="form-range" min="356"
max="1280" value="560" id="videoWidth">

        <div class="form-check">
            <input class="form-check-input" type="checkbox"
value="" id="aspectRatio" checked>
            <label class="form-check-label" for="aspectRatio">
                Mantener relación de aspecto 16:9
            </label>
        </div>
    </div>
</div>
<br>
<!-- Video embebido -->
<div >
    <iframe id="youtubeIFrame" width="560" height="315"
src="https://www.youtube.com/embed/{{ video_id }}"?rel=0">
```

```

title="YouTube video player" frameborder="0" allow="accelerometer;
clipboard-write; encrypted-media; gyroscope; picture-in-picture;
web-share" allowfullscreen></iframe>
</div>
<br>

<!-- Mantener relación de aspecto 16:9 -->
<script>
    var height = $('#videoHeight')
    var width = $('#videoWidth')
    var heightNum = $('#heightNum')
    var widthNum = $('#widthNum')
    var aspectRatioCheckbox = $('#aspectRatio')
    var youtubeIFrame = $('#youtubeIFrame')
    var embedCode = $('#embedCode')

    heightNum.html(height.val())
    widthNum.html(width.val())

    height.on('input', function() {
        heightNum.html($(this).val())
        if (aspectRatioCheckbox.prop('checked')) {
            maintainAspectRatio('height')
        }
        updateIframeSize()
        updateEmbedCode()
    })

    width.on('input', function() {
        widthNum.html($(this).val())
        if (aspectRatioCheckbox.prop('checked')) {
            maintainAspectRatio('width')
        }
        updateIframeSize()
        updateEmbedCode()
    })

    aspectRatioCheckbox.on('change', function() {
        if ($(this).prop('checked')) {
            maintainAspectRatio('height');
        }
        updateIframeSize()
        updateEmbedCode()
    })

```

```

    });

    function updateIframeSize() {
        var newWidth = width.val();
        var newHeight = height.val();
        youtubeIframe.attr('width', newWidth);
        youtubeIframe.attr('height', newHeight);
    }

    function updateEmbedCode() {
        var newWidth = width.val();
        var newHeight = height.val();
        var newCode = '<iframe width="' + newWidth + '"
height="' + newHeight + '" src="https://www.youtube.com/embed/{
video_id }'" title="YouTube video player" frameborder="0"
allow="accelerometer; clipboard-write; encrypted-media; gyroscope;
picture-in-picture; web-share" allowfullscreen></iframe>';
        embedCode.html(newCode)
    }

    function maintainAspectRatio(changedSlider) {
        var aspectRatio = 9 / 16
        var newWidth, newHeight

        if (changedSlider === 'height') {
            newWidth = Math.round(height.val() / aspectRatio);
            width.val(newWidth)
            widthNum.html(newWidth)
        } else if (changedSlider === 'width') {
            newHeight = Math.round(width.val() * aspectRatio);
            height.val(newHeight)
            heightNum.html(newHeight)
        }
    }
}
</script>
{% endblock %}

```

- Anexo 10

error_page.html

```
{% extends 'main_app/home.html' %}

{% block content %}
    <h2>Ha ocurrido un error</h2>
    {{ error }}
{% endblock %}
```

- Anexo 11

Formularios que se asocian al modelo Video, que contiene los campos título, descripción, y archivo de video. VideoForm para subir videos, EditVideoForm para editarlos.

```
from django import forms
from .models import Video

# Formulario para subir video
class VideoForm(forms.ModelForm):
    class Meta:
        model = Video
        fields = ['title', 'description', 'video_file']
        labels = {
            'title': 'Título',
            'description': 'Descripción',
            'video_file': 'Archivo de video',
        }
        widgets = {
            'title': forms.TextInput(attrs={'class': 'form-control'}),
            'description': forms.Textarea(attrs={'class': 'form-control'}),
            'video_file': forms.FileInput(attrs={'class': 'form-control', 'accept': 'video/*'})
        }

# Formulario para editar video
class EditVideoForm(forms.ModelForm):
    class Meta:
        model = Video
        fields = ['title', 'description']
        labels = {
```

```

        'title': 'Título',
        'description': 'Descripción',
    }
    widgets = {
        'title': forms.TextInput(attrs={'class': 'form-control'}),
        'description': forms.Textarea(attrs={'class': 'form-control'}),
    }

```

- Anexo 12

Archivo upload_video.py, que contiene las funciones para poder subir el video a YouTube a través de la API.

```

import http.client
import httplib2
import os
import random
import time

from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
from googleapiclient.http import MediaFileUpload

from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow

# The CLIENT_SECRETS_FILE variable specifies the name of a file
# that contains
# the OAuth 2.0 information for this application, including its
# client_id and
# client_secret. You can acquire an OAuth 2.0 client ID and client
# secret from
# the Google API Console at
# https://console.developers.google.com/.
# Please ensure that you have enabled the YouTube Data API for
# your project.
# For more information about using OAuth2 to access the YouTube
# Data API, see:
# https://developers.google.com/youtube/v3/guides/authentication
# For more information about the client_secrets.json file format,
# see:

```

```

# https://developers.google.com/api-client-
library/python/guide/aaa\_client\_secrets

CLIENT_SECRETS_FILE =
os.path.abspath(os.path.join(os.path.dirname(__file__),
'client_secrets.json'))

# This OAuth 2.0 access scope allows an application to upload
files to the
# authenticated user's YouTube channel, but doesn't allow other
types of access.
YOUTUBE_UPLOAD_SCOPE =
"https://www.googleapis.com/auth/youtube.upload"          # Scope
para subir videos a YouTube
YOUTUBE_ADMIN_SCOPE =
"https://www.googleapis.com/auth/youtube.force-ssl"        # Scope
para editar videos en YouTube
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"

# This variable defines a message to display if the
CLIENT_SECRETS_FILE is missing.
MISSING_CLIENT_SECRETS_MESSAGE = ""
WARNING: Please configure OAuth 2.0

To make this sample run you will need to populate the
client_secrets.json file
found at:

    %s

with information from the API Console
https://console.developers.google.com/

For more information about the client_secrets.json file format,
please visit:
https://developers.google.com/api-client-
library/python/guide/aaa\_client\_secrets
""" % os.path.abspath(os.path.join(os.path.dirname(__file__),
CLIENT_SECRETS_FILE))

VALID_PRIVACY_STATUSES = ("public", "private", "unlisted")

```

```

# Función para obtener el token que permitirá hacer peticiones
autorizadas a la API de YouTube
def get_authenticated_service():
    scopes = [YOUTUBE_ADMIN_SCOPE, YOUTUBE_UPLOAD_SCOPE]
    creds = None
    # The file token.json stores the user's access and refresh
    tokens, and is created
    # automatically when the authorization flow completes for the
    first time.
    if os.path.exists("token.json"):
        creds =
Credentials.from_authorized_user_file("token.json", scopes)

    # If there are no (valid) credentials available, let the user
    log in.
    if not creds or not creds.valid:
        try:
            if creds and creds.expired and creds.refresh_token:
                creds.refresh(Request())
            else:
                flow = InstalledAppFlow.from_client_secrets_file(
                    CLIENT_SECRETS_FILE, scopes
                )
                creds = flow.run_local_server(port=0)

            # Save the credentials for the next run
            with open("token.json", "w") as token:
                token.write(creds.to_json())
        except Exception as e:
            print(f"Ocurrió un error durante la autenticación:
{e}")
            raise e
    try:
        return build(
            YOUTUBE_API_SERVICE_NAME,
            YOUTUBE_API_VERSION,
            credentials=creds,
        )
    except Exception as e:
        print(f"Ocurrió un error creando el servicio de la API de
YouTube: {e}")
        raise e

```

```

# Función para subir el video
    # youtube es un objeto de servicio que se usa para interactuar
con la API
    # data contiene información acerca del video a cargar
def initialize_upload(youtube, data, video_file):
    tags = None
    if data['keywords']:
        tags = data['keywords'].split(",")

    body=dict(
        # Crea
el cuerpo de la petición
        snippet=dict(
            # En
snippet van los campos de title, description, tags, y category
            title=data['title'],
            description=data['description'],
            tags=tags,
            categoryId=data['category']
        ),
        status=dict(
            # En
status está el campo de privacyStatus
            privacyStatus=data['privacy_status']
        )
    )

    # Call the API's videos.insert method to create and upload the
video.
    insert_request = youtube.videos().insert(
        #
youtube.videos().insert() carga el video a YouTube
        part=",".join(body.keys()),
        # El único
parámetro obligatorio. Muestra las llaves de body separados por
comas (str) que serán incluidas
        body=body,
        # El cuerpo de
la petición

    # The chunksize parameter specifies the size of each chunk of
data, in
    # bytes, that will be uploaded at a time. Set a higher value
for
    # reliable connections as fewer chunks lead to faster uploads.
Set a lower
    # value for better recovery on less reliable connections.
    #

```

```

    # Setting "chunksize" equal to -1 in the code below means that
    the entire
    # file will be uploaded in a single HTTP request. (If the
    upload fails,
    # it will still be retried where it left off.) This is usually
    a best
    # practice, but if you're using Python older than 2.6 or if
    you're
    # running on App Engine, you should set the chunksize to
    something like
    # 1024 * 1024 (1 megabyte).

    media_body=MediaFileUpload(video_file, chunksize=-1,
    resumable=True)    # MediaFileUpload maneja el archivo, y lo sube
    a donde ese requiera
    )

    youtube_id = resumable_upload(insert_request)
    return youtube_id

# Maximum number of times to retry before giving up.
MAX_RETRIES = 10

# Always retry when these exceptions are raised.
RETRIABLE_EXCEPTIONS = (httplib2.HttpLib2Error, IOError,
http.client.NotConnected,
    http.client.IncompleteRead,
http.client.ImproperConnectionState,
    http.client.CannotSendRequest, http.client.CannotSendHeader,
    http.client.ResponseNotReady, http.client.BadStatusLine)

# Always retry when an apiclient.errors.HttpError with one of
these status codes is raised.
RETRIABLE_STATUS_CODES = [500, 502, 503, 504]

# This method implements an exponential backoff strategy to resume
a failed upload.
# Toma como argumento la petición youtube.videos().insert()
# Esta función entra en loop hasta que la carga del video se
complete

def resumable_upload(insert_request):

```

```

response = None
error = None
retry = 0
while response is None:
    try:
        print("Uploading file...")
        status, response = insert_request.next_chunk()
        if status:
            print(f"Uploaded {int(status.progress()*100)}%")
        if response is not None:
            if 'id' in response:
                print("Video id '%s' was successfully
uploaded." % response['id'])
                return response['id']
            else:
                exit("The upload failed with an unexpected
response: %s" % response)
        except HttpError as e:
            if e.resp.status in RETRIABLE_STATUS_CODES:
                error = "A retrieable HTTP error %d occurred:\n%s"
% (e.resp.status, e.content)
            else:
                raise
        except RETRIABLE_EXCEPTIONS as e:
            error = "A retrieable error occurred: %s" % e

    if error is not None:
        print(error)
        retry += 1
        if retry > MAX_RETRIES:
            exit("No longer attempting to retry.")

        max_sleep = 2 ** retry
        sleep_seconds = random.random() * max_sleep
        print("Sleeping %f seconds and then retrying..." %
sleep_seconds)
        time.sleep(sleep_seconds)

```

- Anexo 13

Archivo client_secrets.json a llenar.

```
{
  "web": {
    "client_id": "[INSERT CLIENT ID HERE]",
    "client_secret": "[INSERT CLIENT SECRET HERE]",
    "redirect_uris": [],
    "auth_uri": "https://accounts.google.com/o/oauth2/
auth",
    "token_uri": "https://accounts.google.com/o/oauth2/
token"
  }
}
```

- Anexo 14

Archivo views.py de la aplicación main_app, que muestra las funciones que controlan la aplicación.

```
import os
from django.conf import settings
from django.shortcuts import render, redirect
from .forms import VideoForm, EditVideoForm
from .models import Video
from django.contrib.sessions.models import Session
from .code.upload_video import *

import logging
logger = logging.getLogger('site-logger')

def get_user_data(request):
    # Se obtiene la cookie sessionid, que contiene la ID de la
    sesión
    sessionid = request.COOKIES['sessionid']
    session = Session.objects.get(session_key=sessionid)

    # Se obtiene el ID y nombre del usuario logueado
    user_id =
session.get_decoded()['identity_context_data']['_id_token_claims']
['oid']
```

```

        username =
session.get_decoded()['identity_context_data']['_id_token_claims']
['name']
        return {'user_id':user_id, 'username':username}

def home(request):
    return render(request, "main_app/home.html")

def upload_video(request):
    if request.method == 'POST':
        form = VideoForm(request.POST, request.FILES)

        if form.is_valid():
            data=dict(
                title = form.cleaned_data['title'],
                description = form.cleaned_data['description'],
                keywords = "", # form.cleaned_data['keywords'],
                category = "27", # form.cleaned_data['category'],
                privacy_status = "public",
#form.cleaned_data['privacy_status'],
            )

            video_file = request.FILES['video_file']

            user_id = get_user_data(request)['user_id']
            video = Video.objects.create(
                title = data['title'],
                description = data['description'],
                video_file = video_file,      # El nombre es creado
por el modelo
                uploader = user_id
            )
            file_name = os.path.basename(video.video_file.name)
            file_path = os.path.join(settings.MEDIA_ROOT,
'videos', file_name) # Ruta del video a subir

            try:
                youtube = get_authenticated_service()
                youtube_id = initialize_upload(youtube, data,
file_path)

                video.youtube_id = youtube_id      # Se guarda el
ID del video de YouTube en la base de datos

```

```

        video.save()
    except Exception as e:
        video.delete()
        return render(request, 'main_app/error_page.html',
{'error': e})

    username = get_user_data(request)['username']
    logger.info(f"El usuario {username} ({user_id}) ha
subido el video con ID: {youtube_id}.")
    return render(request, 'main_app/success.html',
{'video_id':youtube_id})

    else:
        form = VideoForm()
        return render(request, 'main_app/upload_video.html',
{'form':form})

def my_videos(request):
    user_id = get_user_data(request)['user_id']
    videos_list =
Video.objects.all().filter(uploader=user_id).order_by('-date')
    return render(request, 'main_app/my_videos.html',
{'videos_list':videos_list})

def edit_video(request, video_id):
    user_id = get_user_data(request)['user_id']
    username = get_user_data(request)['username']
    try:
        video = Video.objects.get(youtube_id=video_id)
        if user_id != video.uploader:
            raise Exception
        form = EditVideoForm(request.POST or None, instance=video)
    except:
        return render(request, 'main_app/error_page.html',
{'error':"El video que quiere editar no existe o no es suyo."})

    if form.is_valid():
        try:
            youtube = get_authenticated_service()
            api_request = youtube.videos().update(
                part="snippet",
                body={
                    "id": video_id,

```

```

        "snippet": {
            "title": form.cleaned_data['title'],
            "description":
form.cleaned_data['description'],
            "categoryId": "27",
        },
    },
)
api_request.execute()
form.save()
logger.info(f"El usuario {username} ({user_id}) ha
editado la información del video con ID: {video_id}.")
return redirect('my-videos')
except HttpError as e:
    print("An HTTP error %d occurred:\n%s" %
(e.resp.status, e.content))

    return render(request, 'main_app/edit_video.html',
{'form':form})

def delete_video(request, video_id):
    user_id = get_user_data(request)['user_id']
    username = get_user_data(request)['username']

    try:
        video = Video.objects.get(youtube_id=video_id)
        if user_id != video.uploader:
            raise Exception
    except:
        return render(request, 'main_app/error_page.html',
{'error':"El video que quiere eliminar no existe o no es suyo."})

    try:
        youtube = get_authenticated_service()
        api_request = youtube.videos().delete(id=video_id)
        api_request.execute()
        video.delete()
        print("Video eliminado")
    except HttpError as e:
        print("Un error HTTP %d ha ocurrido:\n%s" %
(e.resp.status, e.content))

```

```

    logger.info(f"El usuario {username} ({user_id}) ha eliminado
el video con ID: {video_id}.")
    return redirect('my-videos')

```

- Anexo 15

Archivo aad.config.json a llenar.

```

{
  "type": {
    "client_type": "CONFIDENTIAL",
    "authority_type": "SINGLE_TENANT",
    "framework": "DJANGO"
  },
  "client": {
    "client_id": "{enter-your-client-id-here}",
    "client_credential": "{enter-your-client-secret-here}",
    "authority": "https://login.microsoftonline.com/{enter-
your-tenant-id-here}"
  },
  "auth_request": {
    "redirect_uri": "null",
    "scopes": [],
    "response_type": "code"
  },
  "flask": null,
  "django": {
    "id_web_configs": "MS_ID_WEB_CONFIGS",
    "auth_endpoints": {
      "prefix": "auth",
      "sign_in": "sign_in",
      "edit_profile": "edit_profile",
      "redirect": "redirect",
      "sign_out": "sign_out",
      "post_sign_out": "post_sign_out"
    }
  }
}

```

- Anexo 16

Configuración de logging para crear el archivo de registros.

```
import os
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters':{
        'main':{
            'format': "[{asctime}] {levelname}: {message}",
            'style': '{'
        }
    },

    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': os.path.join(BASE_DIR, 'logs',
'main_log.log'),
            'formatter': 'main',
            'encoding': 'utf-8',
        },

    },

    'loggers': {
        'site-logger': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
        'MsalViewsLogger':{
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        }
    },
}
```