**MEMORIA\_API\_VIRTUALDISKSD.**

1. **Justificación.**

VIRTUALDISKSD es un API implementado en Python (2.7) que viene a resolver una situación provocada por la oferta de almacenamiento gratuito en la nube. El hecho de registrarse en varios servicios gratuitos nos aporta una gran capacidad de almacenamiento, pero nos obliga a llevar un control de qué ficheros subimos y dónde los subimos, encontrándonos en ocasiones con situaciones en las que tenemos un servicio prácticamente en desuso y otro al borde del colapso. VIRTUALDISKSD nace con la finalidad de ofrecer un servicio centralizado de almacenamiento en la nube.

1. **Objetivo.**

Centralizar todos los servicios de almacenamiento en la nube del usuario en una única API que los gestione de forma transparente.

1. **Planificación del proyecto.**
   1. **Etapas**

El proyecto se ha dividido en 11 etapas, cuyo diagrama de Gantt se ha añadido como Anexo 1, siendo su descripción la siguiente:

* + - **Brainstorming**: Duración estimada de 2 horas. En horario de clase de la asignatura, todos los miembros del equipo aportarán ideas sobre la finalidad de la API, consensuándose al final varias propuestas para su estudio.
    - **AnálisisPropuestos**: Duración estimada de 1 día. Los miembros del grupo, de forma individual, sopesará los pros y contras de cada propuesta salida de la etapa anterior y las jerarquizará. Al final del día, se consensuará la API a diseñar.
    - **AnálisisServicios**: Duración estimada de 2 días. Una vez consensuada la API a diseñar, cada miembro del grupo de forma individual analizará al menos un servicio disponible en la nube que satisfaga los requisitos para el desarrollo de la misma. Al finalizar dicho periodo, informará al resto del grupo de la información obtenida.
    - **EstudioAnálisisServicios**: Duración estimada de 4 días. Cada miembro del grupo de forma individual, buscará información de las APIS disponibles de los servicios analizados en la anterior etapa. Se prestará especial atención a la información disponible sobre las APIs.
    - **RepartoTareas**: Duración estimada de 1 hora. En horario de clase de la asignatura, con la información obtenida en la etapa anterior, el grupo consensuará las dos APIs que se integrarán. Una vez seleccionadas, los 2 miembros del grupo que analizaron dichas APIs asumirán el rol de “responsable de programación” (cada uno de la que estudió) y los otros dos asumirán el rol de “ayudante de programación”, los cuales serán asociados a un responsable de programación.
    - **ProgramaciónMódulos**: Duración estimada de 4 días. Para cada API seleccionada, el responsable de programación y el ayudante de programación elaborarán las funciones necesarias usando su API, así como las plantillas html vinculadas a dicha API.
    - **ElaboraciónMemoria**: Duración estimada de 4 días. Simultáneamente a la tarea anterior y posteriormente a la misma se deberá ir redactando la memoria del proyecto.
    - **ComprobaciónMódulos**: Duración estimada de 1 h. En horario de clase de la asignatura, con los módulos implementados en la etapa anterior, el grupo verificará el funcionamiento de los dos módulos y repartirá las tareas de integración de los mismos.
    - **IntegraciónMódulosWeb**: Duración estimada de 6 días. En esta etapa se programará la API definitiva y su integración con un servicio web que permita comprobar su funcionamiento. Se designará un “responsable de integración”, quedando el resto del equipo a disposición del mismo para el diseño y programación de las secciones que dicho responsable estime oportuno.
    - **FinalizaciónMemoria**: Duración estimada de 3 días. Simultáneamente a la tarea anterior y posteriormente a la misma se deberá terminar de redactar la memoria en base al trabajo ejecutado.
    - **Entrega**: Duración estimada de 1 hora. En horario de clase de la asignatura, se procederá a la entrega del trabajo al profesor responsable de la asignatura.
  1. **Roles**
     + **Director** **del** **proyecto**: Se encargará de velar por el cumplimiento de la planificación, teniendo el poder de decisión en aquellas situaciones en las que el consenso no sea posible.
     + **Responsable** **de** **la** **memoria**: Se encargará de velar por la correcta elaboración de la memoria, con especial énfasis en la coherencia de la misma, siendo responsabilidad suya la comprobación de la aportación de todos los miembros.
     + **Responsable** **de** **programación**: Se encargará de la correcta programación de un módulo de la API, así como de las plantillas en html relacionadas con dicho módulo.
     + **Ayudante** **de** **programación**: Se encargará de las tareas asignadas por el responsable de programación en el ámbito de la programación del módulo asignado y de las plantillas html asignadas.
     + **Responsable de integración:** Se encargará de la correcta programación de la integración de los módulos y de la página web que permita su comprobación. Repartirá la carga de trabajo según su propio criterio entre los otros tres miembros del grupo.

1. **Características de los Servicios.**

Para alojar la información, se requiere de servicios que admitan el almacenamiento de datos en la nube. Estos servicios serán los utilizados para alojar los datos que los usuarios suban de manera transparente, dando la sensación al usuario de que se encuentra ante un único servicio de almacenamiento.

Para elegir los servicios a investigar, se consensuaron los siguientes requisitos:

1. **Prestigio:** Los usuarios deben de tener previamente cuenta en los servicios de alojamiento, por tanto es imprescindible que sean conocidos para facilitar la adopción.
2. **Capacidad:** Los servicios deben de tener una capacidad considerable, no siendo óptimo un servicio que disponga de una capacidad exponencialmente inferior al resto.
3. **API:** Los servicios deben de disponer de una API para desarrolladores, ya que esta es la esencia de poder trabajar con servicios ajenos de manera sencilla. Además el uso de la API debe de ser intuitivo, sencillo y estar bien documentado.
4. **Autentificación OAuth 2:** Este protocolo de autentificación provee una forma segura de realizar autentificaciones a través de APIs, ya que creas una aplicación en el servicio, generando unas keys únicas y se facilita información de esta, no de la cuenta del usuario, de modo que esta información se envía y se recibe un token como respuesta, si este token es correcto la autentificación es correcta, sino no se permite el acceso.

Con estas características en mente y tras un filtro previo, los servicios de alojamiento elegidos son: Dropbox, Google Drive, Box y Microsoft OneDrive.

1. **Requisitos previos.**

El objetivo de VIRTUALDISKSD no es otro que realizar una capa superior y transparente para los usuarios, donde utilizando una aplicación, se pueda interactuar con varios servicios de almacenamiento en la nube, de modo que será VIRTUALDISKSD el encargado de tratar con las APIs de los servicios para manejar la información sin que el usuario tenga consciencia de que por ejemplo dos archivos pueden estar alojados en diferentes servicios. De esta manera se propusieron los siguientes requisitos para la aplicación:

* **Subir fichero:** El objetivo de este requisito no es otro que poder realizar una subida a un servicio de almacenamiento en la nube de manera transparente para el usuario, de modo que antes de realizar la subida, se compruebe cual es el servicio que tiene mayor capacidad en ese momento y se procederá a realizar la subida a ese servicio, por lo que el usuario en ningún momento elegirá a que servicio realizar la subida.
* **Capacidad restante:** Para elegir a que servicio realizar la subida, se necesita saber cuál es la capacidad de la que dispone el servicio, por lo que es necesaria conocer este dato devolviendo un entero con el que realizar la comparación.
* **Listar ficheros:** Este requisito mostrará por pantalla todos los ficheros almacenados en la nube, de manera que el usuario pueda ver la información previamente subida.
* **Bajar fichero:** Una vez subida la información, es necesario poder tener una manera de descargarla para poder acceder a ella.
* **Eliminar fichero:** Cuando un fichero deja de ser útil, se debe de proceder a su eliminación, por lo que se debe de disponer de un método que elimine un fichero aunque el usuario no sepa donde se encuentra almacenado dicho fichero.
* **Reorganizar ficheros:** Se sumará la capacidad disponible de todos los servicios y para ver si un archivo de gran tamaño que no cabría en ningún de ellos podría llegar a caber haciendo una reorganización de los datos, de manera que si el fichero cupiese se proceda al traslado de datos de un servicio al resto y luego se proceda a la subida al servicio con espacio disponible.
* **Autentificación:** Se necesitará un método transparente, previa disposición de una cuenta en el servicio de almacenamiento, se pueda autentificar el usuario en VIRTUALDISKSD.

1. **Problemas con las APIs (AQUÍ CADA UNO LOS PROBLEMAS CON SU API EN SU APARTADO)**
   1. **Dropbox:** E
   2. **Google Drive:**
   3. **Box:**
   4. **Microsoft OneDrive:**

Ante estos problemas, tras consultar al profesor y viendo que el enunciado del ejercicio tan solo propone el uso de 2 APIs diferentes, se decide centrar los esfuerzos en la utilización de las APIs Dropbox y Google Drive, desechando el uso de Box y Microsoft OneDrive para una futura ampliación.

1. **Implementación de VIRTUALDISKSD.**

Como lenguaje de programación se decide utilizar Python, ya que es un lenguaje sencillo, se ve en la asignatura de Sistemas Distribuidos, tiene compatibilidad con multitud de APIs y además dispone de frameworks que dotan al lenguaje de capacidad web. Como framework para el uso en la web de la aplicación se elige Flask por las recomendaciones habidas en clase y porque Flask es lo suficientemente potente y sencillo a la vez para los requisitos de la aplicación, desechando por el camino otros frameworks como Django que aunque tienen más popularidad, son más complejos a priori.

* **Funciones que provee el API VIRTUALDISKSD**:
  + home(): No recibe ningún parámetro y devuelve el fichero html que será el index de del servicio web de VIRTUALDISKSD.
  + Total\_List(): No recibe ningún parámetro y devuelve un html que será el encargado de mostrar por pantalla la lista con todos los elementos almacenados en la nube.
  + driveConfig(): No recibe ningún parámetro y devuelve un html para autentificarte en el servicio de Google Drive.
  + uploadDriveFile(): No recibe ningún parámetro y devuelve un html que sirve para subir un fichero al servicio Google Drive.
  + driveList(): No recibe ningún parámetro y devuelve un html que contiene una lista con los ficheros alojados únicamente en Google Drive.
* **Persistencia con JSON:**

1. **Instalación y Desplegado.**

El servidor elegido para desplegar la aplicación es cloud9, ya que más que un servidor, es un conjunto de herramientas donde se permite desarrollar de manera conjunta y en tiempo real a varias personas, además de proveer una serie de herramientas como servidor propio, de forma que al trabajar con c9 parece que estamos trabajando en local, pero podemos proveer a un tercero o a nosotros mismos para pruebas de una dirección web desde donde se puede acceder al resultado del proyecto.

Por tanto, para instalar el proyecto, ya sea en cloud9 o en local se necesitaran instalar las siguientes dependencias:

sudo pip install flask

sudo pip install google-api-python-client

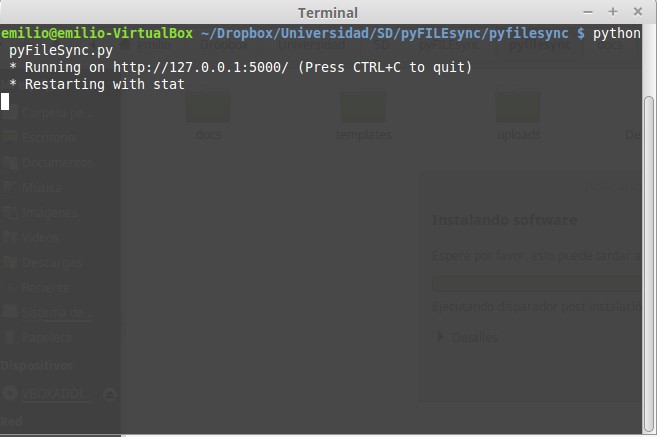
sudo pip install Dropbox

una vez instaladas las dependencias de manera satisfactoria, ya hay que comenzar a diferenciar en la instalación entre el servidor y el modo local:

* **Local:** Para ejecutar la aplicación en modo local, tan solo se deberá de ejecutar el archivo principal de la siguiente manera y aparecerá la siguiente informacion:

Python pyFileSync.py

Y ya estaría corriendo la aplicación lista para abrir un enlace en el navegador:



* **Cloud9:** Al crear el proyecto en c9, debe de elegirse un proyecto de tipo Python/Django, por lo que el proyecto no se encuentra predefinido para trabajar con flask aunque se instalen las dependencias de Flask, así que con esto en mente se tienen que realizar los siguientes cambios:
  + Cambio en el código: La siguiente línea:

if \_\_name\_\_ == "\_\_main\_\_":

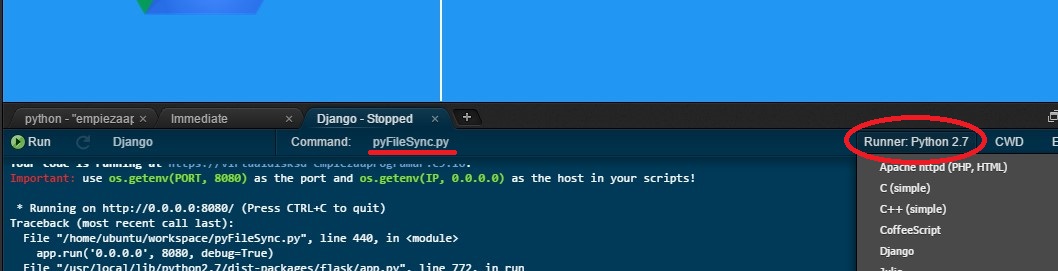
app.run(debug=True)

Debe de ser reemplazada porque no se tiene acceso al PYTHONPATH:

if \_\_name\_\_ == "\_\_main\_\_":

app.run('0.0.0.0', 8080, debug=True)

* + Cambios en el servidor: Para ejecutar Flask en cloud9, debe pulsarse Run Project y ocurrirá un error, entonces se cambian los siguientes parámetros tal como aparece en la imagen:



1. **Uso y Prueba de la aplicación.**
2. **App Movil y ampliaciones futuras.**