Proyecto 1 entrega 1 - Arquitectura, conclusiones y consideraciones

Integrantes:

- Daniel Santamaría Álvarez - 201720222
- Nestor Gonzalez – 201912670
- Álvaro Plata – 20
- Rafael Humberto Rodriguez Rodriguez - 202214371

El proyecto fue desarrollado mediante un API en Flask y se realizaron pruebas de los endpoints implementados mediante postman. El código del proyecto se encuentra en el siguiente repositorio público: https://github.com/NestorGon/SWNube.

En el backend la información se almacenó en una base de datos local SQLite:

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///files.sqlite'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Por otro lado, para el procesamiento asíncrono se utilizó redis como motor de base de datos en memoria en conjunto con celery que permite enlistar actividades:

```
celery = Celery('tasks', broker='redis://redis-container:6379/0')
```

Además, se dockerizó la aplicación mediante dos contenedores que contienen el servidor de redis y el API desplegado y se conectan entre ellos mediante una red común. Las instrucciones para correr el proyecto en docker se encuentran en el README del proyecto en el repositorio.

Respecto al API, hay dos partes importantes de la aplicación: los endpoints y las tareas asíncronas. Los endpoints se definieron de acuerdo con las especificaciones de la guía:

```
api.add_resource(SignUp, '/auth/signup')
api.add_resource(LogIn, '/auth/login')
api.add_resource(Tasks, '/tasks')
api.add_resource(Task2, '/tasks/<int:task_id>')
api.add_resource(Download, '/download/<int:task_id>')
api.add_resource(BackgroundTask, '/tasks/<int:task_id>/processed')
```

Para la parte de autenticación, se implementaron dos endpoints bajo el prefijo /auth uno para crear usuarios (signup) y otro para iniciar sesión (login). La autenticación fue implementada en la ruta flaskr > endpoints > auth.py, donde se segmentan dos requests de tipo POST que reciben los datos del usuario. En el caso del inicio de sesión, se proporciona un token json (JWT) en la respuesta para identificar al usuario en el uso de otros endpoints de la aplicación.

Respecto al procesamiento de tareas de conversión, la actividad principal se realiza mediante la clase Tasks, en la ruta flaskr > endpoints > conversion.py, que contiene un POST para crear una tarea de conversión y un GET que obtiene todas las tareas asociadas a un usuario. En el caso de crear una nueva tarea, una vez se ha almacenado el archivo enviado por el usuario, se inicia el procesamiento asíncrono mediante celery y redis:

```
start_conversion.delay(task.id,in_route,out_route,in_ext,out_ext)
```

En esta tarea asíncrona se usa la librería pydub para convertir de un formato a otro y al finalizar se realiza una petición al API para actualizar la tarea almacenada en la base de datos como procesada. El endpoint del API que responde a esta solicitud corresponde a la clase BackgroundTask con un único endpoint que recibe el id de la tarea por parámetro para modificar la base de datos.

Para obtener una tarea específica es necesario que el usuario se encuentre autenticado, luego filtramos entre las tareas que el usuario tenga y seleccionamos la tarea a la que le corresponde el id que se provee.

Para el endpoint que se encarga de obtener el archivo original se requiere que el usuario se haya autenticado y se provee el id de la tarea que se busca.

Para poder actualizar el formato de conversión de un archivo el usuario debe estar logueado y proveer el id de la tarea a la que se le va a actualizar el formato.

El endpoint de delete se encarga de eliminar la carpeta correspondiente a la tare y los archivos que se encuentren guardados en la misma.

Para probar los endpoints se realizaron 3 colecciones de postman:

Signup collection: https://documenter.getpostman.com/view/14951808/2s847CxFc6

Login collection: https://documenter.getpostman.com/view/14951808/2s847CxFJX

User tasks: https://documenter.getpostman.com/view/19274258/2s84LPvWs9

Escenario y Pruebas de Estrés API REST y Batch

1. **¿Cuál es su entorno de prueba?**

   El entorno en el que se realizó la configuración para las pruebas consiste en una Máquina Virtual con sistema operativo Ubuntu 18.04 LTS, 1GB de memoria RAM y 10 GB de almacenamiento, ejecutándose sobre un equipo Host con sistema operativo MacOS 12.5 con 16GB de RAM.

   Se realizaron múltiples intentos de las pruebas de carga conectándose desde equipos externos (El equipo Host, una segunda máquina virtual, desde el equipo Host hacia un computador con Ubuntu 18.04 en la misma red), pero debido a complicaciones en la comunicación entre dichos equipos no fue posible conectarlos para realizar las peticiones HTTP hacia el API de la aplicación.

2. **¿Cuáles son los criterios de aceptación?**

a. Tras explorar los límites estándar de latencia entre aplicaciones y servicios comerciales modernos, se determinó que el límite de latencia aceptable percibido por el usuario es de 150 ms.

b. En términos de rendimiento, considerando que se espera que el uso de la aplicación sea local y que al ser lanzada recientemente no se espera una adopción inmediata por parte de todos los potenciales usuarios, se definió que se espera que la aplicación y la infraestructura puedan soportar un máximo de 750 usuarios concurrentes. Se espera que esta cifra aumente al mejorar las prácticas de arquitectura (Desacoplamiento) y la infraestructura al hacer un redespliegue con mejores capacidades de Hardware en próximas iteraciones.

c. Respecto a la utilización de recursos, se definió que el uso de la CPU y memoria RAM no deben superar el 70%, en cuyo caso se debe generar una notificación a los administradores.

d. Al realizar la posterior migración de Infraestructura On-premises hacia un ambiente Cloud, se realizarán pruebas de configuración de la infraestructura para determinar cuál es la combinación ideal de recursos para correr esta carga de trabajo específica.

3. ¿Cuáles son los escenarios de prueba?

a. **Escenarios clave:** Un escenario clave es un pico en el número de usuarios y peticiones hacia la aplicación, es decir, cuando se cuente con 750