Memoria-P1

Ignacio Serena y Marcos Muñoz

1. Introducción

En esta práctica se desarrolló una API basada en `Quart`, un framework asíncrono de Python, para gestionar usuarios y archivos en un servidor. La práctica consistió en implementar y probar funcionalidades como la creación, eliminación y manipulación de usuarios y archivos en un entorno distribuido con Docker. A lo largo del proceso se configuraron varios contenedores Docker, se realizaron pruebas con `curl`, y se creó un script en Python para automatizar las pruebas de las funcionalidades implementadas.

2. Detalle de los pasos seguidos para la obtención de las soluciones

2.1 Configuración del entorno

El entorno de trabajo incluyó la creación de múltiples servicios dentro de contenedores Docker. Se crearon dos servicios principales:

- **API de usuarios (user_api)**: gestiona la creación, registro y eliminación de usuarios.
- **API de archivos (file_api)**: gestiona la creación, listado, muestreo y eliminación de archivos asociados a usuarios.

Ambos servicios fueron configurados usando `Docker Compose`, permitiendo que los contenedores se comuniquen entre sí mediante una red interna.

2.2 Organización de directorios

La distribucion de directorios es la siguiente:

P1/

├── Client/

│ └── client.py
├── File_Services/
│ ├── Files_data/
│ ├── Dockerfile

```
│   ├── file.py
│   └── requirements.txt
├── User_Services/
│   ├── User_data/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── user.py
└── docker-compose.yml
```

Estructura de la información:

- **Usuarios**: Almacenados en un directorio por cada usuario donde hay un json con todos sus datos.
  - Ejemplo: `User_data/Beatriz/`
- **Archivos de usuario**: Cada usuario tiene archivos específicos dentro de un directorio con su uid.
  - Ejemplo: `File_data/Beatriz/file.txt`

2.3 Implementación de funcionalidades principales

- `guardar_usuario_json(name, password, uid, token)`:
  - Guarda la información del usuario (nombre, contraseña, UID, y token) en un archivo JSON dentro de un directorio específico para ese usuario.
- `crear_usuario()`:
  - Ruta API (`/create_user`, método `POST`) que crea un nuevo usuario con un UID y token únicos si el nombre no existe. Guarda la información en un archivo JSON.
- `login()`:
  - Ruta API (`/login`, método `POST`) que autentica al usuario verificando su nombre y contraseña, y devuelve el UID y token si las credenciales son correctas.
- `eliminar_usuario()`:

    - Ruta API (`/eliminar_usuario`, método `DELETE`) que elimina el directorio del usuario y sus archivos, si el usuario existe.

- `verificar_token(authorization, uid)`:
    - Verifica si el token proporcionado es válido comparándolo con el token generado a partir de un UID y un UUID secreto.

- `subir_archivo()`:
    - Ruta API (`/subir`, método `POST`) que permite a un usuario subir un archivo. Verifica el token y guarda el archivo en el directorio del usuario.

- `obtener_archivo()`:
    - Ruta API (`/leer`, método `GET`) que permite a un usuario obtener el contenido de un archivo específico de su directorio, si existe.

- `eliminar_archivo()`:
    - Ruta API (`/eliminar`, método `DELETE`) que elimina un archivo específico en el directorio del usuario, tras verificar el token.

- `listar_archivos()`:
    - Ruta API (`/listar`, método `GET`) que devuelve una lista de todos los archivos almacenados en el directorio del usuario, si el token es válido.

- `eliminar_directorio()`:
    - Ruta API (`/eliminar_dir`, método `DELETE`) que elimina el directorio completo de un usuario, junto con todos sus archivos, tras verificar el token.

2.4 Pruebas automatizadas

Se implementaron pruebas automatizadas en `client.py` usando `requests` para verificar el correcto funcionamiento de las API. Se realizaron las siguientes pruebas:

Pruebas de funcionamiento básico:

- **Prueba de creación de usuario**: Se crea el usuario "Beatriz" con la contraseña "supersecreta1234". Se espera una respuesta exitosa con el mensaje de "Operación exitosa".
- **Prueba de inicio de sesión**: Se inicia sesión con las credenciales del usuario "Beatriz". Se espera una respuesta exitosa con el UID y el token

correspondiente.

- **Prueba de subida de archivos**: Se sube el archivo `fichero_001.txt` con contenido "texto de prueba del fichero" usando el token de autenticación del usuario. Se espera una respuesta exitosa.
- **Prueba de lectura de archivos**: Se solicita leer el archivo `fichero_001.txt`. Se espera una respuesta exitosa y el contenido del archivo.
- **Prueba de listar archivos**: Se lista el contenido del directorio del usuario. Se espera una respuesta exitosa con la lista de archivos almacenados.
- **Prueba de eliminación de archivo**: Se elimina el archivo `fichero_001.txt`. Se espera una respuesta exitosa.

Pruebas de control de errores:

- **Prueba de creación de usuario duplicado**: Se intenta crear nuevamente el usuario "Beatriz". Se espera un error 409 indicando que el usuario ya existe.
- **Prueba de inicio de sesión con credenciales incorrectas**: Se intenta iniciar sesión con el nombre de usuario correcto pero una contraseña incorrecta. Se espera un error 401 de credenciales incorrectas.
- **Prueba de subida de archivo sin autenticación**: Se intenta subir un archivo sin proporcionar el token de autenticación. Se espera un error 403 por falta de autorización.
- **Prueba de lectura de un archivo inexistente**: Se intenta leer un archivo que no existe en el directorio del usuario. Se espera un error 404 indicando que el archivo no fue encontrado.
- **Prueba de eliminación de archivo inexistente**: Se intenta eliminar un archivo que no existe en el sistema. Se espera un error 404 de archivo no encontrado.

Pruebas de eliminación de usuarios:

- **Prueba de eliminación de usuario existente**: Se elimina al usuario "Beatriz". Se espera una respuesta exitosa con el mensaje "Usuario eliminado".
- **Prueba de eliminación del directorio del usuario**: Se eliminan todos los archivos del usuario "Beatriz". Se espera una respuesta exitosa con el

mensaje "Directorio del usuario eliminado con éxito".

- **Prueba de eliminación de usuario inexistente**: Se intenta eliminar nuevamente el usuario "Beatriz", que ya ha sido eliminado. Se espera un error 404 indicando que el usuario no fue encontrado.

3. Evidencias del correcto funcionamiento

La comprobación del correcto funcionamiento de la práctica se realizó en dos etapas:

1ª etapa: Esta primera etapa surgio al terminar la API y por lo tanto, sus funciones. Se hizo una prueba independiente de las funciones por separado para comprobar su correcto funcionamiento de forma independiente.

2ª etapa: Esta segunda etapa de pruebas se realizo al terminar la implementación de docker mediante la ejecución de un script muy completo de prueba (client.py)

4. Guía de ficheros utilizados

Ficheros importantes:

- `Dockerfile`: Contiene la configuración para construir la imagen Docker de los servicios `user_api` y `file_api`.
- `docker-compose.yml`: Orquesta los servicios de `user_api` y `file_api`.
- `client.py`: Contiene las pruebas automatizadas para verificar el correcto funcionamiento de la API.

5. Guía de instalación y ejecución

Para comprobar el funcionamiento de la práctica, se deben seguir los siguientes pasos:

1. Descargue y descomprima el archivo entregado en el directorio deseado
2. Cree e inicie un conjunto de contenedores y otros recursos Docker definidos en un archivo con el comando `sudo docker-compose up`
3. En otra terminal, ejecute el fichero `client.py` y disfrute de la prueba automatizada.

6. Aclaraciones importantes

Para la implementación de las funciones se ha decidido no pasar ningún argumento por la URL y hacerlo en todos los casos en el apartado data de las solicitudes. Hemos seguido esta política debido a que hemos considerado un fallo de seguridad pasar datos importantes y privados, como el UID, contraseña o nombre de usuario, por la URL ya que en un caso real, todos las solicitudes hechas a un servidor se guardan en un registro, y por tanto, son vulnerables.

7. Conclusión

En esta práctica aprendimos a desarrollar micro-sercicios en python que implenten una REST API. También ayudó a que interiorizasemos los conceptos sobre los entornos de python propios (venv), los contenedores y su respectivo compose.

La práctica nos permitió desarrollar una API sencilla y funcional para la gestión de usuarios y archivos, empleando tecnologías como `Quart`, Docker y Python. Además, implementamos pruebas automatizadas para asegurar el correcto funcionamiento de las funcionalidades implementadas.