Práctica 1: Micro-servicios REST API en contenedor

Nombres: Javier Agüero, Luis Núñez

Descripción-guía archivos creados en el desarrollo de la práctica

client.py

Función: **execute(method)**

Esta función envía solicitudes HTTP a la API según el método especificado (POST, GET, DELETE). Imprime la respuesta de la API y el código de estado de la respuesta. Si la respuesta contiene un UID, se guarda el token y el UID para uso posterior.

Operaciones del Cliente:

Creación de un Usuario: Se establece la URL para crear un nuevo usuario, se definen las cabeceras y los datos necesarios (nombre y contraseña) y se llama a la función execute con el método POST.

Login de Usuario: Intenta iniciar sesión con el usuario creado utilizando la URL correspondiente y los datos requeridos, y llama a la función execute.

Cambio de Contraseña de un Usuario: Cambia la contraseña del usuario especificando la contraseña antigua y la nueva, utilizando el método POST.

Listado de Archivos del Usuario: Realiza una solicitud para listar los archivos del usuario autenticado, utilizando el token de autenticación.

Creación de un Archivo: Crea un nuevo archivo especificando el nombre y el contenido para el usuario autenticado.

Lectura del Archivo: Solicita leer el contenido de un archivo existente.

Modificación del Archivo: Modifica el contenido de un archivo existente agregando texto adicional.

Eliminación del Archivo: Elimina un archivo existente.

Eliminación del Usuario: Elimina un usuario especificando su nombre y contraseña.

-El archivo interactúa con la API a través de diversas operaciones como la gestión de usuarios y archivos.

-Además se gestiona el token y UID obtenidos durante el inicio de sesión para autenticar operaciones posteriores.

user.py

Funciones:

generar_token(uid): Esta función genera un token único para un usuario basado en su UID, utilizando el UUID secreto.

user_already_exists(nombre): Verifica si un usuario ya existe en el archivo users.txt buscando su nombre. Si el archivo no existe, se crea.

uid_already_exists(uid): Comprueba si un UID ya está registrado en el archivo users.txt.

Rutas de la API:

@app.route('/home'): Ruta de prueba que devuelve un saludo ("Hello World").

@app.route('/user/create', methods=['PUT', 'POST']): Crea un nuevo usuario. Valida si el nombre de usuario ya existe, genera un UID único y guarda el nombre de usuario, el hash de la contraseña y el UID en users.txt. Crea un directorio para el usuario si no existe.

@app.route('/user/login', methods=['PUT', 'POST']): Permite a un usuario iniciar sesión. Comprueba si las credenciales son correctas y devuelve el UID y un token si lo son.

@app.route('/user/delete', methods=['DELETE']): Elimina un usuario del sistema. Verifica las credenciales y, si son correctas, elimina la entrada en users.txt y su directorio correspondiente en el sistema de archivos.

@app.route('/user/modify', methods=['POST']): Cambia la contraseña de un usuario. Busca el usuario y su contraseña antigua en users.txt, y si se encuentran, actualiza la contraseña a la nueva.

-La API permite la creación, inicio de sesión, modificación y eliminación de usuarios mediante la gestión de un archivo de texto (users.txt).

-Se utiliza hashing para almacenar contraseñas de forma segura y se generan tokens únicos para autenticar sesiones de usuario.

-Los archivos de cada usuario se almacenan en un directorio dedicado, lo que permite organizar los datos del usuario.

file.py

Configuración de la Ruta de Archivos: PATHFILE se establece como la ruta donde se almacenarán los archivos del usuario, especificando que están en la carpeta files.

Funciones:

generar_token(uid): Genera un token único para un usuario basado en su UID utilizando el UUID secreto.

user_already_exists(nombre): Verifica si un usuario existe en el archivo users.txt buscando su nombre.

uid_already_exists(uid): Comprueba si un UID ya está registrado en el archivo users.txt.

Rutas de la API:

@app.route('/file/create', methods=['POST']): Crea un nuevo archivo para un usuario. Verifica que el UID existe y que el token es válido. Si el UID no está registrado, devuelve un error. Si el archivo se crea correctamente, se guarda en la carpeta del usuario.

@app.route('/file/delete', methods=['DELETE']): Elimina un archivo especificado por su nombre para un usuario. Comprueba si el UID y el token son válidos antes de intentar eliminar el archivo. Si el archivo no existe, devuelve un mensaje de error.

@app.route('/file/list', methods=['GET']): Lista todos los archivos pertenecientes a un usuario. Valida el UID y el token, y si son correctos, devuelve una lista de los archivos disponibles en la carpeta del usuario.

@app.route('/file/read', methods=['GET']): Lee el contenido de un archivo especificado por su nombre. Verifica el UID y el token antes de intentar acceder al archivo. Si el archivo no existe, devuelve un mensaje de error.

@app.route('/file/modify', methods=['POST']): Modifica el contenido de un archivo existente. Acepta contenido adicional que se añadirá al archivo. Al igual que en las otras funciones, verifica el UID y el token antes de realizar la modificación.

-Si el script se ejecuta directamente, se inicia el servidor de la aplicación en la dirección 0.0.0.0 en el puerto 5090 con el modo de depuración habilitado.

-La API permite crear, eliminar, listar, leer y modificar archivos para usuarios específicos, organizando los archivos en carpetas por UID.

-Se utilizan tokens para autenticar las acciones de los usuarios y asegurarse de que solo los usuarios autorizados pueden interactuar con sus archivos.

-Cada usuario tiene su propia carpeta para almacenar sus archivos, asegurando que los datos de diferentes usuarios estén separados y organizados.

interfaz.py

Variables Globales: Se declaran varias variables globales que se utilizarán en el programa:

-uid, token, user, pwd para almacenar información del usuario.

-commands contiene la lista de comandos disponibles que el usuario puede introducir.

-response almacenará la respuesta de las solicitudes a la API.

Funciones:

fill_data(coms, command): Prepara la información que se enviará a la API en función del comando ingresado. Configura la url, headers y data necesarias para la solicitud basándose en el tipo de comando (LOGIN, REGISTER, CREATEFILE, etc.).

execute(method): Realiza la solicitud HTTP al servidor. Dependiendo del método especificado, utiliza requests.post, requests.get o requests.delete para enviar la solicitud y devuelve la respuesta.

verify(string, num, filemode): Comprueba si el comando ingresado tiene la cantidad correcta de argumentos y, si es necesario, extrae el contenido adicional para operaciones de archivo.

printv(string): Imprime el mensaje en color verde para resaltar la información importante.

extract_uid(response): Extrae el UID y el token de la respuesta de inicio de sesión. Si la respuesta indica un error, devuelve False, de lo contrario, actualiza las variables globales uid y token.

Bucle Principal

Un bucle while True que permite al usuario ingresar comandos de manera continua.

Se procesa la entrada del usuario:

- **Comando NO VÁLIDO**: Si el comando ingresado no es válido, muestra la lista de comandos disponibles.
- **LOGIN**: Inicia sesión si se proporcionan un nombre de usuario y una contraseña correctos. Si el inicio de sesión es exitoso, se extrae el UID y el token.
- **REGISTER**: Registra un nuevo usuario si se proporcionan el nombre de usuario y la contraseña.
- **MODUSER**: Modifica la contraseña del usuario si se especifican los argumentos correctos.
- **DELETEUSER**: Elimina el usuario si se proporciona el nombre de usuario y la contraseña correctos.
- **CREATEFILE**: Crea un nuevo archivo si el usuario ha iniciado sesión y se proporcionan el nombre del archivo y el contenido.
- **MODFILE**: Modifica un archivo existente, añadiendo contenido al mismo.
- **DELETEFILE**: Elimina un archivo existente si el usuario ha iniciado sesión.
- **READFILE**: Lee el contenido de un archivo y lo muestra en pantalla.
- **LISTFILE**: Lista todos los archivos del usuario actualmente autenticado.
- **LOGOUT**: Cierra la sesión del usuario actual, limpiando las variables relacionadas con la sesión.
- **HELP**: Muestra la lista de comandos disponibles.
- **EXIT**: Sale del programa.

-Proporciona un medio para que los usuarios interactúen con el sistema de gestión de archivos y usuarios a través de comandos escritos.

-Permite a los usuarios iniciar sesión, registrarse, modificar y eliminar sus cuentas.

-Facilita la creación, modificación, eliminación, lectura y listado de archivos asociados a los usuarios.

-Incluye validaciones para asegurar que los comandos se ejecuten correctamente y que los usuarios estén autenticados antes de realizar acciones críticas.

Por último cabe mencionar que también existe un archivo dockerfile, el cual se utiliza para crear una imagen Docker que ejecuta una aplicación en Python, específicamente nuestra API, además de un archivo README.txt que contiene información para probar la práctica.