ESCUELA SUSPERIOR DE INFORMÁTICA

SEGURIDAD EN REDES

PRÁCTICA 3

API RESTful

Juan Bautista Castaño García-Cervigón

Alberto Mayorga Naranjo

Arturo Gómez Jiménez

ÍNDICE

I. Explicación de la práctica

Los objetivos de la práctica 3 son:

- Conocer e implementar una API RESTful sencilla.
- Implementar mecanismos de identificación y autenticación de usuarios.
- Implementar mecanismos de confidencialidad utilizando HTTPS.

Por lo que hemos realizado un prototipo de base de datos como servicio en el que se almacenan documentos en formato *JSON*.

La API se puede diferenciar en dos partes: la autenticación de usuarios y los documentos de usuarios.

a. Autenticación de usuarios

En esta parte, los usuarios (con usuario y contraseña), podrán:

- Registrarse en la API RESTful: con */signup*
- Iniciar sesión: con */login*

Al registrarse un usuario por primera vez o al iniciar sesión correctamente, los usuarios obtendrán un token que tendrá caducidad a los cinco minutos de su creación que necesitarán más adelante para poder gestionar sus documentos.
Por otro lado, para obtener otro token, deberán de iniciar sesión de nuevo.

b. Requisitos y almacenamiento de contraseñas

Hemos querido que nuestros usuarios tengan también la certeza de que sus contraseñas sean seguras, por lo que todas las contraseñas deben tener, al menos: una minúscula, una mayúscula, un símbolo y un número. A todo esto, hay que sumarle que las contraseñas deben tener al menos una longitud de ocho.

Por otro lado, las contraseñas son almacenadas en el archivo *users.json*. Para mejorar la seguridad de nuestra API RESTful, no se guardan las contraseñas directamente, si no que al obtener contraseña que pasa el usuario al darse de alta, se genera un *salt* aleatorio. Una vez con este hash, le sumamos la contraseña y realizamos el hash a todo el conjunto.
Finalmente, guardamos la siguiente estructura y se almacena en *users.json*: *hash(contraseña+salt):salt* como en la *imagen 1.*

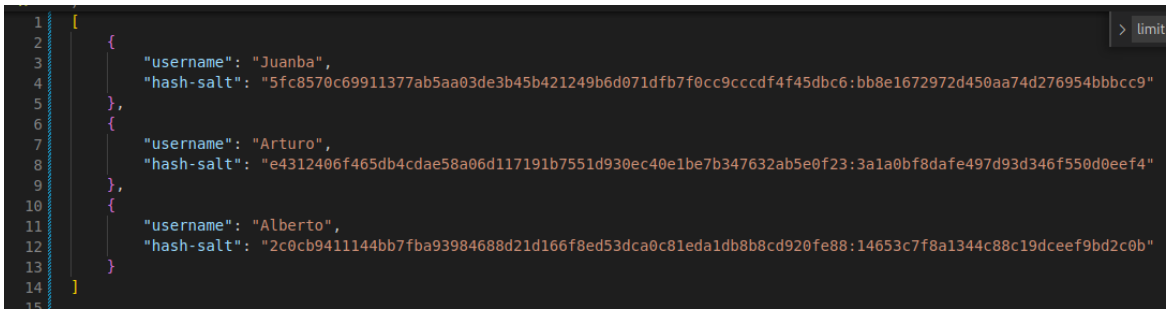

Imagen 1: almacenamiento de contraseñas en users.json

c. Generación y almacenamiento de tokens

La generación del token la hemos realizado con la librería *secrets*, teniendo estos una longitud de 20 bytes y por ello no se repiten.

Los tokens son guardados en el archivo *tokens.json* de manera que cada token generado se guarda asignándolo a su usuario (para que cada token solo sea válido para un usuario) y a los cinco minutos se eliminará.

```
[
    {
        "token_id": "q9CHbwRYY7bPdSoRj5ALNAKaYUA",
        "username": "Arturo"
    }
]
```

Imagen 2: almacenamiento de tokens en tokens.json

d. Documentos de usuarios

En esta otra parte, los usuarios tienen un espacio para guardar sus documentos *JSON*. Para poder gestionar los documentos, será necesario que cada usuario pase su token obtenido anteriormente mediante la cabecera *Authorization* cuyo valor debe de tener el formato: *token <token-del-usuario>.*

Las acciones que puede realizar un usuario son:

- Manipulación de documentos:
  - Crear un nuevo: con POST /user/document
  - Obtener su contenido: con GET /user/document
  - Actualizar su contenido: con PUT /user/document
  - Borrar un documento: con DELETE /user/document
- Listado de documentos:
  - Obtener todos sus documentos

En cuanto a las acciones de manipulación de documentos, en el *GET* y en el *POST* es necesario pasar como argumentos el contenido del documento a crear o a actualizar. Además, este contenido deberá de tener un formato *JSON*.

II. Seguridad implementada

Aunque la práctica no lo pedía, hemos implementado un límite de peticiones (usando *flask-limiter)* para las peticiones de los usuarios en los métodos *signup* y *login*, de manera que cuando un usuario intente darse de alta más de 30 veces por minuto, el servidor devolverá un error *https 429* por realizar demasiadas peticiones.

Si un usuario quiere realizar fuerza bruta a otro usuario, también obtendrá el mismo error al pasar el límite mencionado anteriormente.

```
#/LOGIN
@app.route('/login', methods=['POST'])
@limiter.limit("30 per minute", key_func = lambda : getUsername())
def login():
```

Imagen 3: implementación del límite de peticiones en 30 por minuto

III. Instalación y ejecución

Las peticiones de la *Api RESTful* se implementarán con *curl*. El formato a seguir para pasar el *token* y los parámetros de las peticiones en cada caso, se indican a continuación:

- *Ejemplo de signup:*

```
curl -X POST -d '{"username":"Adrian","password":"Adrian123-"}' https://127.0.0.1:5000/signup
```

Hacer hincapié en que se debe pasar exactamente *username* y *password*. Si no, la API devolverá un error.

- *Ejemplo de POST document:*

```
curl -X POST -H "Authorization:token
q9CHbwRYY7bPdSoRj5ALNAKaYUA " -d
'{"doc_content":{
   "id": 44,
   "game": "Colonos de Catan",
   "author": "Klaus Teuber",
   "ages": "10+"
}}' https://127.0.0.1:5000/Adrian/doc1
```

Hay que tener cuidado con el token, ya que debe tener el formato: "*token <user-token>*" o por el contrario no será aceptado.

La entrega tiene la estructura que se mostrará a continuación:

- Raíz:
  - *Makefile*
  - *README.pdf*
  - *Tokens.json*
  - *Users.json*
- *Src:*
  - *apiRest.py*
  - *http_status_codes.py (definiciones de errores http)*
- *Test:*
  - *Test.sh*
  - *Pass_brute_force.txt*

Para ejecutar la práctica simplemente tenemos que realizar estos pasos:

- Crear un entorno virtual
  - Antes de nada, tenemos que tener instalado python3-venv
    sudo apt install python3-venv
  - Para crear un entorno virtual ejecutamos: *python3 -m venv [directorio]*
  - Activamos nuestro entorno virtual: *source [directorio]/bin/actívate*
  - Para salir de él, solo debemos ejecutar: *deactivate*

- Situados en la carpeta raíz, ejecutar: make *launchServer*
    - Una vez ejecutado este comando, se instalarán las librerías necesarias para ejecutar la API RESTful.
    - También se creará el certificado necesario para lanzarla de forma segura.
        - En el menú emergente, simplemente hay que darle a la tecla *enter* cinco veces hasta que aparezca insertar el *common name,* donde hay que insertar *"127.0.0.1*", como en la *imagen 4*.

```
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:127.0.0.1
Email Address []:
```

Imagen 4: creación de certificados

    - Tras esto, se lanzar a la API y podremos hacer las *request* necesarias.
- Ejecutamos el *test.sh* creado insertando el comando: *make tests*.