Práctica 3

Los objetivos de esta práctica son los siguientes:

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

Enunciado

Vamos a crear un prototipo de base de datos como servicio. Empezaremos con algo totalmente funcional y que se puede utilizar para almacenar documentos en formato JSON. La base de datos se podrá utilizar como una API RESTful y permitirá diferentes cuentas de usuarios.

Para este prototipo, el programa servidor se asumirá en el endpoint `https://myserver.local:5000` donde `myserver.local` resolverá a `127.0.0.1`.

Cuenta de usuarios

Cada usuario tendrá su espacio donde podrá alojar sus documentos JSON. Dicho espacio será accesible a través de `/<usuario>` y los documentos se podrán crear y recuperar a través de `/<usuario>/<id>` donde `<id>` es una cadena que identifica al documento dentro de la cuenta de usuario. Por ejemplo: `https://myserver.local:5000/pepe/receta-arroz-con-leche`, accede al documento con el identificador `receta-arroz-con-leche` del usuario `pepe`.

Los documentos se almacenan como archivos en el propio servidor. Siguiendo el ejemplo anterior, tendríamos un archivo para dicho documento en `<ruta_raiz>/pepe/receta-arroz-con-leche.json` siendo `<ruta_raiz>` configurable.

Autenticación de usuarios

Los usuarios del servicio tendrán un nombre de usuario (`username`) y una contraseña asociada (`password`). Para darse de alta, los usuarios utilizar `/signup`.

Para poder utilizar el servicio, los usuarios necesitan un *token* temporal que el propio servidor emite al utilizar `/signup` o `/login`. Este token expira a los 5 minutos de su creación, así que los usuarios deberán volver a llamar a `/login` para poder obtener un nuevo token. Todas las operaciones que el usuario ejecute mientras el token no expire serán permitidas.

API

Los objetos que se deben implementar, junto con sus verbos, vienen explicados a continuación. Los verbos HTTP no especificados en cada objeto significa que no son aceptados por el propio objeto y no se debe permitir su llamada.

`/version`

- `GET`: devuelve la versión del programa.

`/signup`

- `POST`: crear un usuario nuevo. Necesita los siguientes argumentos:
  - `username`: el nombre de usuario.
  - `password`: la contraseña de usuario.

  Si todo va bien, devuelve el token de acceso para el usuario con el formato:

  ```
  {"access_token": "123456789"}
  ```

`/login`

- `POST`: autenticar un usuario existente. Necesita los siguientes argumentos:
  - `username`: el nombre de usuario.
  - `password`: la contraseña de usuario.

  Si todo va bien, devuelve el token de acceso para el usuario con el formato:

  ```
  {"access_token": "123456789"}
  ```

`/<string:username>/<string:doc_id>`

Todas las operaciones requieren autenticación mediante la **cabecera HTTP** `Authorization` cuyo valor debe tener el formato: `token <token-del-usuario>`.

- `GET`: obtiene el contenido del documento `doc_id` del usuario `username`.

  Si todo va bien, devuelve el contenido íntegro del documento en formato JSON.

- `POST`: crea un nuevo documento con identificador `doc_id` en el usuario `username`. Necesita los siguientes argumentos:
  - `doc_content`: el contenido del documento a crear en formato JSON.

  Si todo va bien, devuelve el número de bytes escritos en disco con el formato:

  ```
  {"size": <total_bytes>}
  ```

- `PUT`: actualiza el contenido del documento `doc_id` del usuario `username`. Necesita los siguientes argumentos:
  - `doc_content`: nuevo contenido del documento en formato JSON.

  Si todo va bien, devuelve el número de bytes escritos en disco con el formato:

  ```
  {"size": <total_bytes>}
  ```

- `DELETE`: borra el documento `doc_id` del usuario `username`.

  Si todo va bien, devuelve una respuesta vacía:

  ```
  {}
  ```

`/<string:username>/_all_docs`

Todas las operaciones requieren autenticación mediante la cabecera `Authorization` cuyo valor debe tener el formato: `token <token-del-usuario>`.

- `GET`: obtiene el contenido de todos los documentos del usuario `username`.

  Si todo va bien, un objeto en formato JSON con la siguiente estructura:

  ```
  {
    "id1": {
      ...contenido de id1...
    },
    "id2": {
      ...contenido de id2...
    },
    ...
  }
  ```

¿Qué se pide?

1. Implementar el servicio en Golang. Durante las sesiones de prácticas se propondrá diferentes frameworks con los que se puede implementar.
   - Para cada argumento de entrada, se debe especificar y comprobar convenientemente su validez y aceptación. Es importante estar seguro que los valores dados por el usuario son **correctos y seguros**.
   - Para cada situación de error, se debe devolver un código de error HTTP que sea apropiado para el error en cuestión. Por ejemplo, si un documento no existe, el código `404` es apropiado, mientras que el código `403` no lo es.
   - Se recomienda utilizar un mecanismo de autenticación de usuario visto en clase (como ficheros `shadow`).
   - No es necesario que los tokens expedidos estén disponibles incluso en caso de reiniciar el servidor. Pueden residir en memoria para este prototipo.
   - La autenticación con el token debe hacerse en cada una de las peticiones que lo necesiten. La generación del token debe ser lo suficientemente buena como para que no haya colisiones entre nuevos tokens.

2. `README.pdf` explicando brevemente el proyecto, como instalarlo y cómo ejecutarlo. Debe contener las instrucciones necesarias para que cualquier profesional, sin necesidad de conocer cómo está implementado, pueda ejecutarlo localmente sin problemas.

    - En este documento se deben también exponer aspectos que aseguren la disponibilidad del servicio como la limitación de cuota de espacio por usuario, o limitación de peticiones de usuarios por un intervalo de tiempo. Estas medidas no son necesarias implementarlas pero sí documentarlas para trabajo futuro.

Las prácticas entregadas se probarán con `curl` y la librería de Python `request` para su evaluación mediante pruebas automáticas, por lo que es muy importante implementar la API tal y como se especifica. Además, la evaluación intentará aprovechar fallos de seguridad inherentes a la especificación de esta práctica.

¿Qué se valora?

1. La creación de un repositorio en GitHub privado y el desarrollo de la práctica en el mismo.
2. Incluir script que lance y configure el servidor adecuadamente y lance pruebas automáticas.
3. Chequear y comprobar parámetros de entrada convenientemente y anticipar posibles fallos de seguridad que puedan aprovechar los usuarios.