Servicio de autenticación

Revisión 2

1. Introducción

Se deberá implementar una API REST que permita la gestión básica de usuarios. El lenguaje de implementación es libre, aunque se recomienda utilizar Python 3. Además, deberá crearse una librería de acceso a la API y un cliente que permita utilizar el servicio sin necesidad de conocer la propia API REST.

2. Implementación del servicio

El servicio se apoyará en una base de datos para almacenar todos los usuarios del sistema. El alumno puede utilizar la tecnología de base de datos que desee, aunque se recomienda utilizar alguna de las disponibles en la biblioteca estándar de Python 3, como por ejemplo SQL Lite. En cualquier caso, la base de datos deberá estar almacenada localmente de manera que se pueda indicar mediante un *argumento opcional* la ruta de la misma.

La base de datos almacenará los datos persistentes de un usuario, es decir: nombre de usuario, *hash SHA-256* de la contraseña y la lista de roles asociados al usuario. Se generará un identificador único para indexar a los usuarios: el user_id, de tipo cadena.

Se deberá crear una clase que implemente la persistencia de datos, es decir, que abstraiga de todos los detalles de la base de datos, de manera que tenga métodos similares a:

- Crear un usuario.
- Borrar un usuario.
- Establecer nuevo hash SHA-256 a un usuario.
- Recuperar los roles asociados a un usuario.
- Añadir/eliminar roles asociados a un usuario.
- Indicar si un usuario existe o no.
- Indicar si un hash SHA-256 dado es correcto para un cierto usuario o no.

Además de la capa de persistencia, se creará otra clase que implementará la capa de negocio. Dicha clase se inicializará utilizando una instancia de la clase de persistencia con la que realizará todas las operaciones de acceso a la base de datos. Todas las operaciones necesitarán el nombre de usuario solicitante de la operación. Las acciones que puede realizar un solicitante son:

- Un solicitante con rol “admin” podrá eliminar o crear otros usuarios.
- Un solicitante con rol “admin” podrá cambiar el hash de la contraseña de cualquier usuario.
- Un solicitante con rol “admin” podrá añadir o eliminar roles de otro usuario.
- Un solicitante siempre podrá cambiar su propio hash de la contraseña.
- Un solicitante siempre podrá eliminar a su propio usuario.

La capa de negocio contará con excepciones predefinidas que deberán lanzarse en caso de error, estas excepciones son: UserNotFound(usuario) y Forbidden(usuario). La primera se producirá cuando se intente realizar una operación sobre un usuario que no exista en la persistencia. La segunda cuando el solicitante de una operación no pueda realizar la acción solicitada.

La capa de negocio debe apoyarse en la librería de acceso al servicio de tokens, puesto que la verificación de los tokens de autenticación de las cabeceras es realizada por dicho servicio.

Por último, se escribirá la capa de presentación, que consistirá en la API REST descrita a continuación.

3. API REST

A continuación, se indican los recursos admitidos por la API:

- API_ROOT/user
  - Método PUT
    - Cabeceras obligatorias: AuthToken: <string>
    - Input: {“username”: <string>, “roles”: [<string>], “pass_hash”: <string>}
    - Output:
      - 201 (Created), data: {“user_id”: <string>}
      - 400 (Bad Request)
      - 401 (Unauthorized)
- API_ROOT/user/{user_id}
  - Método GET
    - Cabeceras obligatorias: AuthToken: <string>
    - Output:
      - 200 (OK), data: {“username”: <string>, “roles”: [<string>]}
      - 401 (Unauthorized)
      - 404 (Not Found)
  - Método DELETE
    - Cabeceras obligatorias: AuthToken: <string>
    - Output:
      - 204 (No Content)
      - 401 (Unauthorized)
      - 404 (Not Found)
  - Método PATCH, POST
    - Cabeceras obligatorias: AuthToken: <string>
    - Input: {“username”: <string>, “roles”: [<string>], “pass_hash”: <string>}
    - Output:
      - 200 (OK), data: {“username”: <string>, “roles”: [<string>]}
      - 400 (Bad Request)
      - 401 (Unauthorized)
      - 404 (Not Found)
- API_ROOT/is_authorized/{auth_code}
  - Método GET
    - Output:
      - 204 (No Content)
      - 404 (Not Found)

La capa de presentación debe tener en cuenta los siguientes aspectos:

- El solicitante debe averiguarse utilizando el valor de la cabecera AuthToken. Para ello debe usarse el servicio correspondiente, en caso de no disponer de dicho servicio, podrá utilizarse un mock.

- Una vez obtenido el solicitante, se llamará a la función de la capa de negocio correspondiente.
    - En caso de capturar una excepción se retornará el error 4XX correspondiente.
    - Si la operación no produjo errores, se retornará el estado 2XX indicado.
- El “auth_code” es una cadena formada con la expresión hexadecimal del hash SHA-256 de la cadena resultante de concatenar el nombre de usuario y el pass_hash:
    - usuario = “test”
    - pass_hash = “9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08”
    - auth_code = Sha256(“test9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08”) = **8f286193e023cd52b89af86cc9a5588f329e67c4381ec1eb6cc9c49ced471907**

4. El servidor

El servicio de autenticación se implementará mediante un servidor, que aceptará las siguientes opciones:

- “-p <puerto> o “--port” <puerto>”: establece un puerto de escucha, si no se establece por defecto será el 3001.
- “-l <dirección>” o “--listening <dirección>”: establece una dirección de escucha, por defecto se usará “0.0.0.0”.
- “-d <ruta>” o “--db <ruta>”: establece la ruta de la base de datos, por defecto se usará el *current working directory* o CWD.

5. Pruebas

Se crearán una serie de pruebas unitarias que deberán poder ejecutarse de forma automática dentro del entorno virtual apropiado. Por tanto, el proyecto incluirá un archivo *requirements.txt* con los datos necesarios para crear ese entorno. Las pruebas deberán tener las siguientes características:

- Las clases de la capa de negocio y persistencia deberán probarse con un >70% de cobertura.
- El código del servidor y la capa de presentación en >50%.

Para facilitar la ejecución inicial del servicio, se creará un script (en bash o python) que se llame “bootstrap” y que incializará la base de datos y el usuario inicial, solicitando el password de forma interactiva (“read -s” en bash o “getpass” en python). Si el servicio requiere de algún archivo adicional, ese script debe crearlo. De manera que después de ejecutar el script, el servicio debe estar listo para ejecutarse.