Servicio de almacenamiento de objetos

Revisión 1

1. Introducción

Se deberá implementar una API REST que permita la gestión básica de blobs o paquetes de bytes. El lenguaje de implementación es libre, aunque se recomienda utilizar Python 3. Además, deberá crearse una librería y un cliente en Python 3 para poder 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 los datos que se requieran para mantener la coherencia en el 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 información general de cada blob almacenado: el identificador del mismo, la ubicación en el sistema de ficheros local y dos listas "*readable_by*" y "*writable_by*", que contendrán los usuarios que, respectivamente, pueden leer y escribir en dicho directorio.

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 nuevo fichero.
- Cambiar el blob de un fichero.
- Borrar un blob.
- Añadir permisos de lectura de un blob a un usuario.
- Quitar permisos de lectura de un blob a un usuario.
- Añadir permisos de escritura de un blob a un usuario.
- Quitar permisos de escritura de un blob a un usuario.

Finalmente se creará una API REST que utilizará dicha clase para implementar su funcionalidad. Los métodos y recursos que la API debe exponer son los siguientes:

- /v1/blob/<blob_id> (GET): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista "readable_by". En ese caso se devolverá un código 200 y el blob. En caso contrario, se devolverá el error 401. Si el <blob_id> no existe, un 404.
- /v1/blob/<blob_id> (PUT): Se acepta la petición si incluye la cabecera de administrador o un user token válido. Se devolverá un 200 y se aceptará (y creará) el blob. Si no se incluyen cabeceras administrativas ni de autenticación de usuario o son inválidas, se devolverá el error 401.
- /v1/blob/<blob_id> (POST): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista "writable_by". En ese caso se devolverá un código 200 y se reemplazará el blob. En caso contrario, se devolverá el error 401. Si no existe el blob, un error 404.

- /v1/blob/<blob_id> (DELETE): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista “writable_by”. En ese caso se devolverá un código 204 y se eliminará el blob. Si no existen las cabeceras o no son válidas, se devolverá el error 401. Si no existe el blob con dicho identificador, un error 404.
- /v1/blob/<blob_id>/writable_by/<user> (PUT): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista “writable_by”. Si no se incluyen cabeceras o no son válidas, se devolverá error 401. Si no existe el blob con ese identificador, un 404. Si el usuario ya está en la lista, se devolverá un 204, en caso contrario, se añadirá y se devolverá un 200.
- /v1/blob/<blob_id>/writable_by/<user> (DELETE): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista “writable_by”. En ese caso se devolverá un código 204. Si no se incluyen cabeceras o no son válidas, se devolverá error 401. Si el usuario no estaba en la lista “writable_by” se devolverá un 404, también se devolverá ese código si no existe un blob con ese identificador.
- /v1/blob/<blob_id>/readable_by/<user> (PUT): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista “writable_by”. Si no se incluyen cabeceras o no son válidas, se devolverá error 401. Si no existe el blob con ese identificador, un 404. Si el usuario ya está en la lista, se devolverá un 204, en caso contrario, se añadirá y se devolverá un 200.
- /v1/blob/<blob_id>/readable_by/<user> (DELETE): Se acepta la petición si incluye la cabecera de administrador o un user token cuyo dueño esté en la lista “writable_by”. En ese caso se devolverá un código 204. Si no se incluyen cabeceras o no son válidas, se devolverá error 401. Si el usuario no estaba en la lista “writable_by” se devolverá un 404, también se devolverá ese código si no existe un blob con ese identificador.

2.1 Cabeceras

Las siguientes cabeceras tienen que ser reconocidas por el servicio:

- admin-token: <token>
- user-token: <token>

3. El servidor

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

- “-a <token>” o “--admin <token>”: establece un token de administración. Si no se establece, se generará uno automáticamente y se mostrará por salida estándar.
- “-p <puerto> o “--port” <puerto>”: establece un puerto de escucha, si no se establece por defecto será el 3002.
- “-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.
- “-s <directorio>” o “--storage <directorio>”: establece la ruta donde se almacenan los blobs, por defecto se creará el subdirectorio “storage” dentro del *current working directory*.

Requiere un argumento obligatorio, la URL de la API del servicio de autenticación. Antes de iniciar el servidor, verificará que el token administrativo establecido es válido mediante la URL. En caso de token inválido, el servició cancelará la ejecución y saldrá con un código de error 1.

4. 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 una cobertura de al menos un 75%.
- El código del servidor y la API deberá estar cubierto en al menos un 50%.