Actividad 4: Desarrollo API

PROFESOR
Ricardo Vílchez García

Introducción

En esta actividad vamos a aglutinar todo lo aprendido en la semana de práctica desarrollando un API REST clave-valor con FastAPI y Redis como base de datos en un sistema multi-contenedor. La aplicación permitirá recomendar películas a usuarios registrados. Las películas se filtrarán por calificación de edad. El principal dato que exponemos es una lista de películas ordenadas por un valor cocinado, en este caso simulado con un dataset de Rotten Tomatoes.

Cada componente, que describimos a continuación, deberá ser documentado con el mayor rigor y detalle posible.

BBDD Redis

Rotten Tomatoes es una conocida web de críticas cinéfilas. Vamos a utilizar un dataset público de sus valoraciones (fichero adjunto rotten_tomatoes_movies.csv) para cargar una base de datos Redis con películas que serán ofrecidas mediante un API REST.

Este fichero simula en Ingeniería del dato los procesos de enriquecimiento del dato previos que finalizan en la asignación de un peso ("tomatometer_rating"), el valor que queremos comunicar a otras aplicaciones.

Los campos relevantes del fichero son:

- "movie_title": Título de la película.
- "content_rating": Calificación de edad que usa el Motion Picture Association film rating system. Sus valores son:
    - G - General Audiences: Apto para todos los públicos.
    - PG - Parental Guidance Suggested: Algunos contenidos pueden no ser apropiados para niños.
    - PG-13: Parents Strongly Cautioned: No recomendado para menores de 13 años.
    - R - Restricted: No recomendado para menores de 17 años.
    - NC-17: Solo para mayores de 17 años.

    - NR - Not Rated: Sin clasificar.

- "genres": Géneros. Valores separados por comas. Ejemplos: "Drama", "Comedy, Horror".
- "tomatometer_status": Agrupación de las valoraciones de Rotten Tomatoes. Valores: "Rotten", "Fresh", "Certified-Fresh".
- "tomatometer_rating": Valoración de Rotten Tomatoes de la película. Valor entero de 0 a 100.
- "original_release_date": Año de estreno.

Aclaraciones:

- El fichero contiene un total de 17.712 películas. De ellas, solo nos interesa la lista de películas con mejor valoración ("tomatometer_status" = "Certified-Fresh") y con calificación de edad informada ("content_rating" distinto de "NR"). Esta lista son 2.559 películas.
- Las películas deben almacenarse en Redis usando las operaciones SET/GET para registros string (a no ser que se elija otro mecanismo de los sugeridos en la sección “Evaluación – Extras”).
    - La elección de la clave y valor string (CSV, JSON, etc.) queda a elección del alumno (no es necesario almacenar cada película en un registro diferente).
    - Dependiendo de la elección de la clave, será necesario un mayor o menor número de registros en Redis.
- Se deben cargar en Redis cada vez que se arranque el contenedor FastAPI usando Lifespans events. Si se usa un volumen para persistir la bbdd en disco, hay que ejecutar FLUSHALL antes de cargar nada para realizar el borrado de todas las claves.

Aplicación FastAPI

API REST con 4 métodos HTTP agrupados en un router con tag “best-movies” y prefix "act4".

Método 1: POST /act4/register

Permite el registro de nuevos usuarios en una base de datos en memoria con un diccionario de usuarios.

- Protegido mediante API Key pasado por parámetro (detalle en apartado Securización).

- Recibe como parámetros obligatorios tipo string:
    1. “username”: Nombre de usuario. Tamaño mínimo 4 caracteres, la primera letra debe ser mayúscula.
    2. “password”: Contraseña. Longitud mínima 8 caracteres.
    3. “content_rating”: Puede tomar los valores G, PG, PG-13, R, NC-17.

- La contraseña se almacena hasheada usando el algoritmo “bcrypt”. Documentación.

- Las credenciales se almacenan en un diccionario en memoria con estructura a elección del alumno. Por ejemplo, la clave puede ser “username” y el valor, otro diccionario con “username”, la contraseña hasheada y el “content_rating” del usuario:

```
{
  “Rober”: {
     “username: “Rober”,
     “hashed_password”: “$2b$12$VcB9fTaK6nViJaDuZseaaefqjHU58OFGVXXYhxOPNqjA3F.tJtDZO”,
     “content_rating”: “G”
  },
  “Marian”: {
     “username: “Marian”,
     “hashed_password”: “$2b$12$sbxu21/V135gaLJT2i.YKOap6KvuR0DYwCwWjr.8XqK9rs5FIX99u”,
     “content_rating”: “R”
  }
}
```

- Respuestas:
    1. HTTP 204: OK.

2. HTTP 400: Si el usuario ya está registrado.
3. HTTP 401: Unauthorized: API Key no válida.

Método 2: POST /act4/token

Solicitud de token de acceso usando OAuth 2.0 con mecanismo de "password", tal y como vimos en la teoría:

- Sin securización.

- La entrada debe ser un parámetro de tipo form_data: Annotated[OAuth2PasswordRequestForm, Depends()]. Documentación.

- Se deben validar las credenciales de usuario contra las almacenadas en el diccionario en hash. Documentación.

- Tras una autenticación satisfactoria, se debe generar un token de acceso JWT con los claims:
  - "sub": nombre de usuario (“username”).
  - "exp": expiración del token: 15 minutos.
  - “cr”: content rating con el que el usuario se registró.

- El algoritmo de firma del token debe ser HMAC-512 (HS512) con un secreto hasheado. El secreto se puede generar con “openssl rand -hex 32”. Documentación.

- Respuestas:
  1. HTTP 200 OK application-json con el token en formato:

{“access_token”:
“eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJSaWNrIiwiZXhwIjoxNzI5MjU3NTMwLCJjciI6IkcifQ.mQeTuKErGQqtQacskJ4UynWCyru9iM3ZmtxTg5J8NzFGMFhioPJ2zVGhhdRpf6R2NlDVzAKSgNVvu7AyfD9_Tg”,
“token_type”: “bearer”}

En https://jwt.io/:

2. HTTP 401 Unauthorized: Usuario no registrado o contraseña errónea.

Método 3: GET /act4/movies-by-content-rating

Devuelve una lista de las 10 películas mejor valoradas filtrando por la calificación de edad (content_rating) del cliente.

- Protegido con OAuth 2.0: Sólo los usuarios portadores de un token JWT válido podrán obtener el listado de películas.
- El valor del claim “cr” (content-rating) del token JWT validado se utilizará para filtrar las películas.
- Cada resultado debe incluir los campos "movie_title", "original_release_date", "genres", "content_rating" y "tomatometer_rating".
- La lista debe estar ordenada descendentemente por el campo "tomatometer_rating".
- Cada petición debe acceder a la base de datos Redis para obtener las películas.
- Respuestas:
    1. HTTP 200 OK con la lista de películas.
    2. HTTP 401 Unauthorized con el header “WWW-Authenticate” valor ”Bearer”.

Método 4: GET /act4/key-list-size

- Devuelve el número de claves en Redis como entero.

- Protegido mediante API Key pasado por parámetro (detalle en apartado Securización).

- Respuestas:
    1. HTTP 200 OK con el número de claves.
    2. HTTP 401 Unauthorized: API Key no válida.

Securización

Documentación sobre el mecanismo APIKeyQuery e inyección dependencias por operación:

- https://fastapi.tiangolo.com/reference/security/#fastapi.security.APIKeyQuery
- https://fastapi.tiangolo.com/tutorial/dependencies/dependencies-in-path-operation-decorators/

El valor de API_KEY se pasará a la aplicación como variable de entorno mediante configuración en el fichero Docker Compose:

```
environment:
  - API_KEY={value}
```

Hay varias formas de leer una variable de entorno en Python. Recomiendo para FastAPI:

- https://fastapi.tiangolo.com/advanced/settings/#types-and-validation

Notas generales

- El formato de las respuestas de error debe ser coherente en toda la aplicación, siguiendo un modelo simple JSON con un campo “detail” string describiendo el error.
- La excepción HTTPException de FastAPI implementa este formato, y permite añadirle headers. Ejemplo:

```python
raise HTTPException(
    status_code=status.HTTP_401_UNAUTHORIZED,
    detail="Not enough permissions",
    headers={"WWW-Authenticate": authenticate_value},
```

Evaluación

- Básico: 60%.
    1. Funcionalidad 4 métodos: 40%.
    2. Calidad documentación en documento referencia: 20%.
- Extra: 40%:
    1. Almacenar las credenciales de usuario en una base de datos distinta de Redis: 20%
        - Sugerencia: MongoDB, Postgres.
    2. Almacenar las películas en Redis utilizando un mecanismo distinto a SET/GET: 20%
        - Utilizando búsquedas del módulo RedisSearch de Redis OM (Object Mapping). Si se elige esta opción, es necesario modificar la imagen de redis a "redis/redis-stack:latest", que es la que incluye el módulo RedisSearch, entre otros.
        - Operaciones sobre JSON.

Si se decide realizar algún extra, por favor detallad la solución elegida en la entrega.

Entrega

- El alumno deberá entregar un fichero docker-compose.yml con la aplicación multi-contenedor implementada. Como mínimo, deberá contar con dos contenedores, el del código de la aplicación FastAPI y el de Redis.
- La imagen con el código FastAPI debe residir en el repositorio DockerHub del alumno y llamarse “eoi-fastapi-act4”.
- La imagen de Redis deberá ser "redis:latest" o “redis/redis-stack:latest”.
- Se podrá desplegar con Docker Compose mediante "docker-compose up -d"
- El endpoint de SwaggerUI debe ser http://localhost:4242/docs

Ejemplo docker-compose.yml:

```
services:
  act4:
    image: {repositorio_dockerhub_alumno}/eoi-fastapi-act4
    container_name: act4
    ports:
      - "4242:4242"
    environment:
      - API_KEY={value}
    redis:
      image: "redis:latest"
      container_name: redis
```