Lab 8: APIs with Express

This week you will create a **basic back-end for your FilmLibrary**. To do so, you will use the Express framework. The back-end must implement a series of **APIs** to support the main features of the React web-based FilmLibrary you have developed so far: **create**, **read**, **update**, and **delete** the films. The data will be persistently stored in an SQLite database.

1. API design

Design a set of APIs to support the main features of your web-based FilmLibrary. Data *passed to* or *received from* the API should be in **JSON format**. The APIs should allow the application to:

- **Retrieve** the list of all the available films.
- **Retrieve** a list of all the films that fulfill a given filter (i.e., the same filters described so far).
- **Retrieve** a film, given its "id".
- **Create** a new film, by providing all relevant information – except the "id" that will be automatically assigned by the back-end.
- **Update** an existing film, by providing all the relevant information, i.e., all the properties except the "id" will overwrite the current properties of the existing film. The "id" will not change after the update.
- **Update the rating** of a specific film.
- **Mark** an existing film as favorite/unfavorite.
- **Delete** an existing film, given its "id".

List the designed APIs, together with a short description of the parameters and the exchanged entities, in a readme file. Be sure to identify which are the collections and elements you are representing. You might want to follow this structure for reporting each API:

```
[HTTP Method] [URL, optionally with parameter(s)]
      [One-line about what this API is doing]
      [Sample request, with body (if any)]
      [Sample response, with body (if any)]
      [Error response(s), if any]
```

2. API implementation

Implement the designed HTTP APIs with Express. The films must be **stored persistently** in an SQLite database. To this end, you can use the provided database "films.db" (see hints below). The database contains two tables, "users" and "films", with each user that may have one or more films. The film entries have all the fields already described so far. During this week, consider only the information of the "films" table. In other words, when you are retrieving films, you should not consider the value of the "user" column. When you are creating new films, you should assign all of them to the same user (e.g., user with id=1). The information about users will be used during the last week when you will implement the login functionality.

3. API testing

Test the realized API with the **REST Client extension** for Visual Studio Code. To this end, you will have to write an API.http file according to the following syntax:

```
[HTTP Method] [URL, optionally with parameter(s)] HTTP/1.1
content-type: application/json (if needed)

[Sample request, with body (if any)]
```

Hints:

1. The file “`films.db`” is included in the repository available on GitHub: https://github.com/polito-WA1-AW1-2023/lab08-express
2. As you saw in the lectures, you can connect to an SQLite database using the following module: **sqlite3** (https://www.npmjs.com/package/sqlite3) – the basic library
3. To browse the content of the database, you can use one of the two following approaches:
   a. Download the Visual Studio Code *SQLite extension* (you can search for it in VSCode extension hub or browsing the following link): https://marketplace.visualstudio.com/items?itemName=alexcvzz.vscode-sqlite
   b. Download the application *DB Browser for SQLite*: https://sqlitebrowser.org/dl/
4. Create a back-up copy of the database before testing your APIs.
5. Visual Studio Code **REST Client** extension is available at the following link: https://marketplace.visualstudio.com/items?itemName=humao.rest-client