Web Applications [2024/2025]

Lab 4: Test APIs with focus on security

In this lab you will make sure that what you created for the **basic back-end for your FilmLibrary** is reasonably robust to malformed/malicious requests. You can also start from the solution provided for the previous lab.

1. Make API code robust and secure

Review the code handling the request to all your APIs. Modify the code to make sure that:

- **SQL injection is always prevented**: for instance, all queries to the database must be parametric queries using the `'?'` placeholder for values coming from the client.
- **All parameters have been formally validated whenever possible**: if a number is expected, this is indeed a valid number, dates are in the expected format, etc. Use a validator such as the `express-validator` package to simplify the checks.
- **Received parameter values are correct according to the application logic**: for instance, if the data contains an external key (e.g., the id of a film), this id exists in the database. Perform the check before carrying out the operation associated with the API, if needed.
- **No race conditions may happen**: make sure that each API, once executed, leaves the database in a consistent state and such consistency does not depend on which APIs are called before or afterwards. (Note: for simplicity, assume that all queries run in the code of a single API are executed in a transaction, even if you do not explicitly write the transaction).
- **No permanent data is stored in global variables in the server**: all information is stored in the database.

2. Test wrong/malicious requests sent to your API

Design a set of requests to be sent to your APIs to the previous statements. For instance, try to send:

- **Wrong IDs**: negative values, values which do not correspond to actual IDs in the database. Check that the APIs behave appropriately, e.g. not performing the requested operation and/or returning an error if appropriate.
- **Invalid data**: for instance, films with invalid values in properties, such as score, use of an invalid filter, …

3. Test your colleague's code (optional)

Give your API server to a colleague and vice versa (via email/telegram/usb stick, …), and test the robustness of the validation by sending malicious requests and checking the behavior.

4. Search API (optional)

Add another API that returns all films whose title contains a given string. Either define an additional, separate route (e.g., /searchFilms) or modify the filter API by supporting an additional query parameter. Beware of avoiding SQL injection.