Assignment 1: Web services

Fabian Denoodt, Benjamin Vandersmissen & prof. José Oramas

DS 2024-2025

1 Introduction

This document provides an overview of the requirements and background information for the first practical assignment of the Distributed Systems course. The goal of this assignment is to familiarize yourself with accessing public-facing REST APIs, implementing your own web service via a REST API and consuming the web service in an additional Python script. This assignment is **individual**.

2 Goal

The goal of the assignment is twofold:

- Firstly, you should create a *RESTful* API which by consuming two existing APIs (more on them later) provides a service to the user. Your API will function as a basic recommendation engine for movies based on the data from The Movie Database (TMDB). You will also implement a Python script that consumes this API.
- Secondly, you will provide extensions to this project (related to the contents of this course) based on what you find interesting and would like to learn more about.

3 Base requirements

3.1 Web service

The web service needs to implement at least the following five functionalities:

1. List the first n popular movies, where $1 \leq n \leq 20$ are legal values.
2. Given a movie, return a list of movies that have all its genres in common.
3. Given a movie, return the movies with a similar runtime. (You can assume a similar runtime has a maximum of 10 minutes difference)
4. Given a set of movies, have the ability to generate a bar plot comparing the average score of these movies.
5. Be able to "favorite" and "unfavorite" a movie, and be able to retrieve a list of currently favorite movies.

Importantly, you should choose a good URI design for the web service that incorporates the functionalities **but also adheres to the RESTful principles**. This means that you may need to implement additional endpoints or functionalities not listed above, depending on your URI scheme implementation.

To implement the listed functionalities, your API will need to consume the following two existing APIs:

- TMDB API (`https://developers.themoviedb.org/3`) for movie-related data.
- QuickChart (`https://quickchart.io/`) for consuming any graph-generation-related services.

Additionally, you need to include a manual with your code that details the API endpoints you implemented, along with the arguments they require, their return values, and a brief description. You will do this via Flasgger, a Python package that automatically generates API documentation based on your code. Your API documentation should clearly describe all endpoints, expected inputs, and return values. The following resources may be useful to get started: Implement a Python REST API with Flask & Flasgger - GeeksforGeeks and flasgger/flasgger: Easy OpenAPI specs and Swagger UI for your Flask API.

3.2 Python Script

A non-GUI test script (`consume_api.py`) should consume your API and demonstrate that all five functionalities listed in 3.1 are present. A good testing script clearly indicates which functionality is currently being tested here, e.g., as follows:

```
# ** Functionality 1. List the first n popular movies **
print(‘‘this code tests this functionality...")

# ** Functionality 2. ... **
...
```

4 Extension Requirements

By only satisfying the base requirements, you can obtain a maximum score of 12/20. To obtain the remaining 8 points, you must propose and integrate meaningful extensions into your project. As a rule of thumb, the workload of these extensions should be approximately 40% of the base project. You are flexible in how you extend the project. Possible suggestions (for inspiration) include:

- Improved efficiency: Introduce caching (e.g., using Redis or Flask-Caching) to optimize performance for frequently requested queries.
- Rate limiting: Implement request throttling using Flask-Limiter to prevent excessive API usage.
- Authentication: Support the generation of your own access token & restrict access when no token is provided.
- Frontend integration: Develop a simple frontend using a modern library such as Svelte or React to interact with your API.

The main requirement for this part is that the extensions should contribute to your own learning. Simply implementing another basic functionality (e.g., adding another API endpoint without meaningful complexity) adds little educational value and is therefore not considered a valid extension. The same applies to integrating technologies that have been covered in a previous course. You are now responsible for your own learning; enjoy it and make it worthwhile. You can always ask the teaching assistants during the lab sessions or via email for specific questions on valid extensions.

5 Deliverables

The following are the minimum requirements and deliverables for a passing grade:

- You should include a way for us to easily run your solution via the files `run_api.sh` and `run_script.sh`, which will automatically start your web service and run the Python script, respectively. Ensure that any required information, such as API keys, is injected into the web service when it is launched. One way to achieve this is by passing the keys through command-line arguments, e.g., `python app.py --key XXXX`. For more information, refer to the documentation on argparse: `https://docs.python.org/3/library/argparse.html`.
- Flasgger generates a local website containing the documentation of your API. However, you must also provide a document, `manual.pdf`, which contains a static copy of your documentation (e.g., obtained via "Print to PDF" in your browser). This is necessary in case your solution does not run on our devices, allowing us to still evaluate your work and assign meaningful scores.
- In addition, you should complete a report (maximum 3 pages; see the provided template) that discusses the following aspects:

  Basic requirements

  - Explain how your API follows the RESTful principles.
  - (Optional) Motivate your design decisions. Are there any designs you considered but decided not to implement? Why?
  - Discuss efficiency. How would you improve the performance optimization of your API?
  - Fault tolerance: Can your API handle faulty requests? If so, what kind of errors does it address, and how are they handled?

  Extension requirements (if applicable)

  - Carefully discuss your extensions. Describe what you have added and why. If you implemented additional technologies or algorithms, explain what they do, and how they function. Note: Your extensions will be primarily evaluated based on the report, so ensure that each extension is documented with sufficient depth.
- Ensure that your submitted solution is complete and contains no missing files. If you use additional libraries, provide us an easy way to install them (for Python, use a `requirements.txt` file).
- You must also provide the API keys you used and ensure that there is enough quota available for us to test your solution.

6 Deadline and Submission

The deadline for this assignment is the **15th of April** at **23:59**. Late submissions will automatically have points deducted for tardiness.

Make sure that your submission has the following format and upload it via Blackboard. `DS-Assignment1-`*`Snumber-Lastname`*`.zip`.

Your submission should include the following files:

- run_api.sh
- run_script.sh
- app.py
- consume_api.py
- report.md
- manual.pdf

Hints:

- Your project will be evaluated using Python 3.12.9. Ensure that your code is compatible with this version.
- Before writing any code, carefully study the principles of REST and RESTful APIs, and ensure that your URI scheme satisfies these principles.