Play Network Take Home Problem Analysis

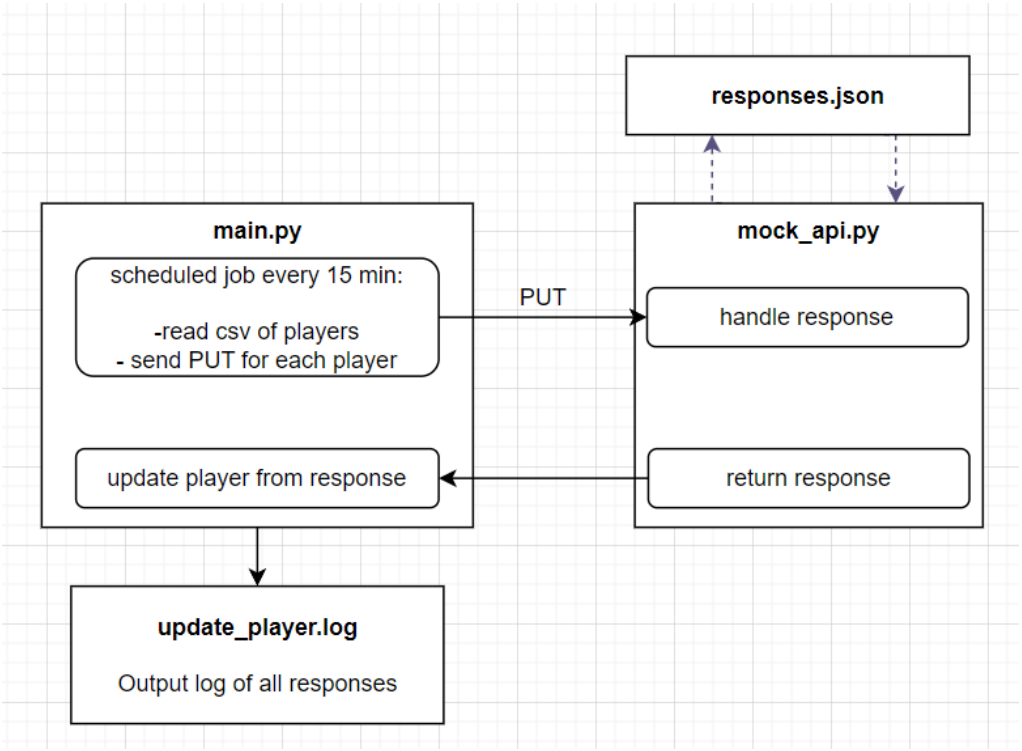
Rayan Isran

October 10, 2024

1 The problem

We have thousands of music players in the wild. Each player queries an API on a server every 15 minutes to check whether a new version is out. If so, the music player updates itself. Provided alongside this document is a tool that automates the update of a thousand such players listed in a .csv file. In this document, I explain 3 things: the structure of the code, assumptions behind technical decisions, and limitations of the tool.

The below diagram summarizes the flow of the code. It runs a scheduled task that reads the *profile* of each music player and sends a PUT request to a mock API, which is a locally running Flask server. This server simulates API responses. Each response is read from *responses.json*. Once a player is handled, the next PUT request is sent until all players are processed.



The project directory contains the following files:

- **Documentation:**
  - `README.md`: Documentation for the user.
  - `DEV-README.md`: Documentation for developers wishing to add to or edit the tool.
- **Python Scripts:**
  - `mock_api.py`: A local Flask server that handles API requests and returns responses based on arbitrary, made-up conditions.

- `main.py`: The main script to run.
- `tests\test_main.py`: Unit tests that run the main script job, and test all possible server responses. Note that it will take approximately 30s to process the .csv on all of Windows, Ubuntu, and MacOS.

- **Other Files:**
  - `test_players.csv`: An input file containing 1,000 generated MAC addresses. This file is regenerated when running the main script.
  - `requirements.txt`: A list of Python packages to install.
  - `keys.conf`: A file containing the client ID and authentication token sent in the PUT request header (note on this file in Section 2).
  - `responses.json`: A list of possible responses and status codes for requests.
  - `ci.yml`: A workflow that setups an environment, runs the API server, and executes all unit tests on Windows, MacOS, and Linux, whenever a change is pushed to the master branch.
  - `main.bat`: An optional script for Windows that runs the server and main script, saving the end user a few keystrokes and clicks.

2 My approach & code style

Python is generally my go-to language for prototyping; it is cross-platform compatible, and to test a simple client-server system is easy enough with Flask. Importantly, it's a language that promotes a procedural programming paradigm, which I generally find easier to both code and understand.

My preferences were to keep the number of code files small, the number of lines in each file relatively small, and thirdly, for a user to understand the code easily after reading the diagram above. Some files were added to enable better readability and scalability, for example, *responses.json*. Although this would add a layer of complexity, a schema could have been used to validate request and response objects. Instead, simple conditions with clear messages have been used as validation checks.

Note: The authentication token and Client ID system is only shown for the purpose of the demo and replication. In reality, *keys.conf* would be kept in a .gitignore file. Users would place their own key and Client ID. On the server side, these variables would ideally be placed in environment variables on the system. A secure library or tool is a better option, but was avoided to keep the codebase simple, especially since the assignment seemed to focus moreso on handling API requests.

3 Limitations & Possible Improvements

To keep the code relatively simple, I used synchronous requests. In reality, it might be more effective to use asynchronous calls, especially as the number of music players will likely increase.

Currently, there is no quick and easy method to check which player updates failed, since the generated .log file is far from pretty. To enhance support and improve the developer experience, it would be beneficial and cool to create a geographical map displaying the locations of all Juke-Boxes with their status reports.

Furthermore, we could implement schema validators and more advanced error handling, such as rate-limiting the API or retrying updates for players that failed on the first attempt.

A CHANGELOG.md file could be set up such that changes are noted with PRs are submitted.