Functional requirements for the ASE Lab 24/25 Project

Gacha Game Backend

Introduction

This file defines the project's functional requirements and may be updated if unforeseen situations arise. The service product was introduced in Lab 2 (User Stories), and the project guidelines were outlined in Lab 3 (Project Kickoff). Both files are available on the course website. Some functional requirements relate to topics that will be explained later in the course. Refer to the “Project Take away” slide in each lab for direct links to labs topics and project details.

The final software must implement all published red and yellow user stories (file *US.xlsx*). Green user stories (or any additional features you may add) are optional but will be rewarded with a higher evaluation if implemented effectively. Achieving the maximum grade (30 cum laude) is possible by implementing only the red and yellow user stories. Following best practices for each tool used is highly recommended.

The backend must follow a microservice architecture, deployable and executable with Docker Compose. The microservices must expose a REST API and communicate through it, with persistent backend data.

The code must be hosted on GitHub, and the repository must include instructions for installing and executing the microservice application in a standard “Get Started” section of the *README.md* file. This section should specify commands for installing, building, and running the backend and its main tests. When specifying external dependencies, include the exact version you are using (e.g., in Python, each `requirements.txt` file must specify something like `Flask==3.0.3`).

Documentation delivery

In the GitHub repo you must have a folder called “docs” containing at least:

- final report in PDF format (based on the template we will publish),
- the OpenAPI specification file for the REST API exposed by the backend,
- a JSON file for each microservice representing the Postman collections with unit tests (use a subfolder if they are too many),
- the Locust file for performance tests, and
- a copy of the YAML file(s) specifying GitHub Actions workflow(s).

The microservice architecture must be free of architectural smells detectable by microFreshner, with the exception of the Endpoint-based interaction smell.

A client for interacting with the backend is not mandatory but it can be useful for you to test some flows, e.g. registration&login flow.

Functionalities

- Rolling a gacha: the backend must answer with the image and the gacha information. You can implement it as a flow with different invocations if you write a client.
- In-game currency: we will not be able to cover payment services in the course, the only requirement for it is to expose an endpoint with a request for an increase of currency in the user balance. If you want to implement it as extra feature, Stripe has dockerised version of its client.
- Market: the auction market must have a way to sell gachas via auction and bid to win an auction of other players. In the report you will need to specify the rules you decide for the market and you must provide tests to verify the consistency of these rules.

In general, consider and mitigate potential fraudulent behaviors by players and avoid actions that could lead to an inconsistent balance.

It is not mandatory to log every backend event; the corresponding user story has been downgraded from yellow to green. For debugging, however, it's recommended to use some form of logging.

Testing

Unit tests are required for all microservice endpoints, with each endpoint needing at least one test for correct input (expecting a 200 OK response) and one for incorrect input (expecting an error response). These unit tests must be conducted using Postman, and the collection of requests should be exported as a JSON file.

Performance testing must be conducted using Locust, targeting only the gateway. This test should cover all player endpoints, except for registration and login functionalities. Additionally, the test must verify the rarity distribution of gacha rolls to show that, with a high volume of requests, the distribution remains accurate.

Security

Administrators must have a separate API access from players, running on a different Docker network. All communication must use HTTPS, with each microservice utilizing self-signed certificates. The input received by each microservice must be sanitized, and important data must be stored encrypted in the database(s).

Authentication and authorization processes must comply with OAuth2 and OpenID Connect standards, using JWT tokens. To authenticate a user, credentials must consist of "username" and "password," and these credentials must be securely transmitted, stored, and validated.

The codebase must undergo automatic analysis using free tools for static and dependency analysis available for the programming language in use. Docker images must be free from critical and high-severity vulnerabilities.

Evaluation

Each project group member is responsible for the entire project, meaning that they must understand the backend's design, deployment, execution, and testing, as well as the rationale behind the project's realization. This does not require knowledge of specific implementation details.

The evaluation will be on:

- **How the backend works**. We must be able to install and execute the architecture and perform automatic and manual tests by following the documentation.
- **Quality of the documentation**. Clarity, usefulness and size (not too much, not too few)
- **Design choices**. There is high degree of freedom, make your choices reasonable and sound.
- **Testing**. The tests you provide should be enough to cover the main expected behaviour of the backend.
- **Security**. All the security requested must be implemented, we will try to perform attacks to see if you implemented everything correctly.
- **(Additional features)**. Every additional functionality give a plus if it is reasonable, well-implemented and well-documented.

Miscellaneous

In general, everything compliant with the functional requirements above is accepted. If you are not sure about something about your solution, send an email at alessandro.bocci@unipi.it to expose your concern or to fix an office hour meeting.