# Lab Project
# Towards the Deadline

Alessandro Bocci

name.surname@unipi.it

Advanced Software Engineering (Lab)

## Content of this document

- Checklist of functional requirements: if you miss a point on this checklist your project is considered failed.
- Structure of the final report.
- Info about the interactive demonstration.

# Checklist of functional requirements

Tests

- ☐ The repository README has instructions on executing the backend and the tests (in isolation, integration and performance).
- ☒ The software is a microservice architecture and not a monolith.
- ☒ With `docker compose up --build` the code runs without crashing.
- ☐ The red and yellow user stories are implemented (players=all; admin=login/logout + gacha management).
- ☐ The report follows the given structure.
- ☐ OpenAPI files for all the microservices and the gateways are in the docs folder.
- ☐ The GitHub Actions workflow file is in the docs folder.
- ☐ Tests in isolation are executable for all the (internal) microservices with players operations and the Postman collections for it are in the docs folder.
- ☒ Integration test is executable for all the players' operations and the Postman collection for it is in the docs folder.
- ☐ Performance test is executable for all the player endpoints involving gacha operations and the locustfile is in the docs folder.
- ☒ All the requests to microservices (internal and gateways) use HTTPS.
- ☒ The only ports bound with the local machine are the gateways (one for players and one for admins) ports.
- ☐ The authentication and authorization server follows the specifications indicated in Lab 11.

## Structure of the final report

Use whatever editor you want (Word, Latex, PowerPoint etc.)

You must write it in English language.

- 1 Page: Names of members and group (if you want also a table of contents).
- 1 Page: Gachas overview (some images, short description, rarities).
- 1-3 Pages: Architecture (details later).
- 1-3 Pages: User Stories (details later).
- 1-2 Pages: Market rules (details later).
- 1-2 Pages: Testing (details later).
- 1-2 Pages: Security – Data (details later).
- 1-3 Pages: Security – Authorization and Authentication (details later).
- 1-3 Pages: Security – Analyses (details later).
- X Pages: Additional Features (details later).

## Structure of the final report

The length of sections is not very strict but try to not exceed the number of pages indicated. If you go too long is a sign that you are writing too much.

Descriptions should be short, give preference to tables or bullet lists.

If you did something special, you have to specify it, otherwise I can miss it.

# Architecture

- Put the image of the architecture (with microFreshener if possible).
- Describe with a table or bullet points all the microservices (for each use two lines of text at most). Put info about what a microservice does, and the language used to develop it (or the third-party software if you used an external image).
- Describe why you connected two microservices (excluding the gateway and the databases).

For example: “Market is connected with Currency because it needs to check the currency of a user and notify the update of it due to the end of an auction or a higher bid”.

## User Stories

For each user story of the player put the endpoint(s) to realize it and the microservice(s) involved.

Use a table or a bullet list.

You can use the index of the shared file to indicate them.

For Example:

“create my game account/profile SO THAT I can participate in the game” has index 4, you can write:

- 4) /register (Gateway, Authserver, DBusers)

or

- 18) /auction/{auc_id}/bid (Gateway, Market, Currency, Gachas)

## Market rules

Give a general description of the decision you took for the market.

Try to answer questions like:

- What happens to the currency of a player when someone else bids higher?
- What happens if I bid at the last second of the auction?
- Can I bid for an auction in which I am the highest bidder?
- …

Basically, specify the rules you could send to players when releasing the game.

# Testing

Write here any particular fact about the testing.

For example:

- I tested in isolation the DBManager_x together with the DB_x
- I used a third-party service that interacts with Service_y and I put them together in the isolation test because mock it is hard for reason z.
- …

## Security – Data

- Select one input that you had to sanitize, describe what it represent, which microservice(t) use it and how you sanitize it.
- List the data you encrypted at rest, describe what they represent, which database stores them and where you en/decrypt them.

## Security – Authentication and Authorization

Describe the scenario you selected (centralized vs distributed) by indicating the basic steps to validate a token and how the keys to sign the token are used and stored.

Try to describe it as schematic as possible (support it with lists, tables or figures)

Put the payload format of your Access Token (bullet list, table or image)

## Security – Analyses

Put the screenshot of:

- The report of the static analysis tool you used (e.g. Bandit’s final table).
- The dashboard of docker scout with your (developed) images, where the vulnerabilities are indicated.

If your language does not have static analysis tools available, specify it here.

Otherwise, put the command(s) you used to reach the results in the screenshot.

Put also the name of the docker hub repository with the images.

Note: there is no need for docker scout for images of third-party software.

# Additional features

Any features not in the functional requirements and not in a red or yellow user story are considered additional features.

Specify your additional features without being too wordy (please be schematic) and answer questions like

- What is this feature?
- Why is it useful?
- How is it implemented?

## Interactive demo

At the interactive demo, all the group members must be present.

I will ask you to connect your device to a projector, and I will ask two things each:

1. One question about the report content.
2. To perform one task on your software.

People failing to answer a question will have a backup question.

People failing to answer backup questions will fail the project.

The language for questions and answers will be English.

# Interactive demo

Some examples of questions:

- Project the architecture and describe it
- What is your authorization scenario? Describe it.
- What is the flow of the endpoint /abc?
- What data did you encrypt?
- How did you test in isolation XY?
- Explain the additional feature XY.

# Interactive demo

Some example of task:

- Build and run the architecture, and perform a request to the endpoint /abc.
- Register a user, do a login and roll for a gacha.
- Launch the microservice xy tests in isolation.
- Create an auction.
- …