# Advanced Software Engineering
# Final Project Report

## Academic Year 2024/2025

Leonardo Pantani
Riccardo Fantasia
Christian Sabella

University of Pisa

# 1 Gachas Overview

JoJo's Bizarre Adventure is a manga created by Hirohiko Araki, known for its generational story-line that follows the adventures of the Joestar family, each characterised by strategic combat and surreal imagery. One of the distinctive elements of the series are the Stands, physical manifestations of a character's spiritual powers, which are often represented by entities with unique abilities related to their user's personality.

We choose stands (some examples in 1, 2 and 3) from Jojo's Bizzare Adventure saga, because they are described in detail through a set of statistics already provided by the manga, thus allowing a more realistic and easier representation and implementation of the pvp battles. In the 1.0 release of the game we were able to implement 60 stands, each with its own statistics. Stands have a level of rarity based on our considerations of their strength and style. They are divided into four main categories: *common*, *rare*, *epic* and *legendary*. Each character (stand) has 6 different statistics, used in the implemented combat (pvp) functionality: power, speed, durability, precision, range, potential.

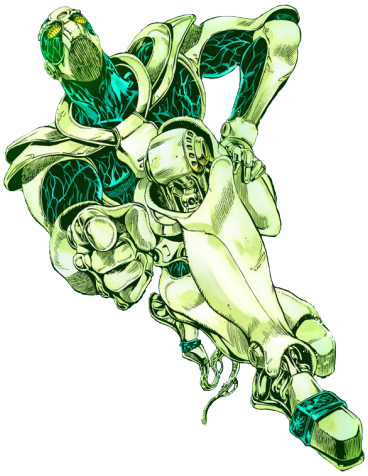

Figure 1: Hierophant Green Stand

Figure 2: Star Platinum Stand

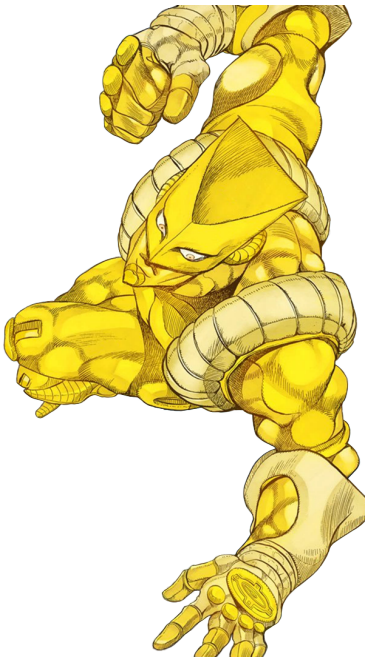

Figure 3: The World Stand

## 1.1 Rarities

A pool is a set from which gacha are drawn, one can imagine it as the abstract representation of a *gashapon* (or *gacha machine*). In our case, the stands are those that can be 'drawn' from a pool. Each stand has a probability of being drawn from a specific pool based on the probability of its rarity. For example, the stand **Made in Heaven** of *legendary* rarity, in the Joestar's Legacy pool has a probability of being drawn of 1% because within that pool the probability of a *legendary* stand being drawn is 5% and there are a total of 5 stands of rarity within it. As there are several pools implemented in the game, it can happen that the same stand can appear inside several pools. Each

Pool may have a different draw probability for each rarity, so the smart player would be better off choosing the Pool with the best draw probability to get the stand he prefers. In the Heaven Pool there are 3 *legendary* stands, but the probability of a *legendary* being drawn is 10%.

# 2 Architecture

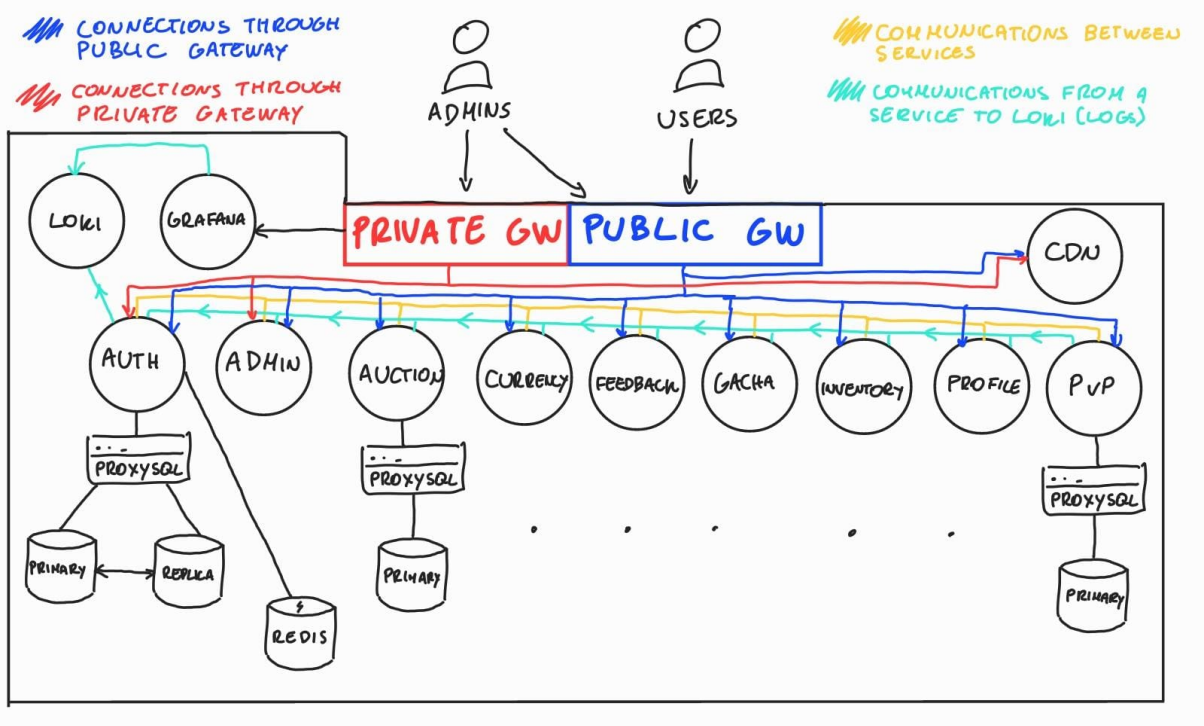


Figure 4: Handwritten System Architecture

The infrastructure shown in 4 uses numerous containers for its proper functioning. To maintain the logical separation of functional requirements, it was decided to divide it into 2 networks: **public** and **private**. The **private** part is only accessible with *admin* access. We will elaborate on this aspect later. As far as containers are concerned, we have decided to define them in this way:

- **services** → perform a particular activity and offer their endpoints externally (accessible via *API Gateway*) or internally (not accessible from the outside, but only from other containers within the Docker network). All were built on the `python:3.13-alpine` image. In alphabetical order:
  - **admin** → accessible from the **API Gateway Private**, allows administrators to perform specific operations on the game. Can create/edit/delete gacha and pools. It can ban and edit profiles, as well as update auctions and manage feedback. It can also view user feedback and service logs.
    - This service communicates with all other containers of type **service**, to perform its tasks correctly.
  - **auction** → manages the auction system, allowing users to create auctions, bid currency, see the status of their auctions and a general list.

    * This service communicates with Inventory to verify that an object is actually owned and to move its ownership. It communicates with Profile to add in-game currency to those who sell a gacha, and remove it from those who buy it. Communicates with Currency to enter transactions into the database.

- **auth** → allows users to log in, register and log out. It also offers various other services including, most importantly, verification of the JWT token for identification and authorisation.
    * This service communicates with Profile to associate the username used for access with the relative uuid, as well as to store some parameters stored in Profile, such as username, currency and PvP score.
- **currency** → allows users to buy in-game currency and view bundles. You cannot convert an arbitrary amount of real money into in-game currency, but you can buy bundles like: `bundle` costs 9.99 EUR and provide 1000 in-game currency. Note that you can use this path to add currency to your mock bank account with: POST https://localhost/currency/buy/add_myself_some_currency.
    * This service communicates with Profile to add currency when a user buys a Bundle by exchanging real money for in-game currency.
- **feedback** → manages the feedback part of the architecture. It gives a feedback list to the admins and enters the feedback into the database on behalf of the users.
    * This service communicates with Profile to obtain the user's username given the uuid.
- **gacha** → handles all the operations around gacha, including the *pull* operation, by which a new gacha is obtained. It allows players to see which gachas exist and what statistics they have.
    * This service communicates with Inventory to filter the list of Gacha with those owned by the user and insert a pulled Gacha into the user's inventory. Communicates with Profile to obtain currency from the uuid, remove in-game currency from players who pull from a Pool.
- **inventory** → lists to players their owned gacha, as well as allowing them to delete and view it in detail. It is used by many internal services.
    * This service communicates with Auction to check whether an item is at auction, before removing it, for example.
- **profile** → allows you to edit certain properties of your profile, such as username and email. Allows you to see information from other players.
    * This service communicates with all services to delete the account. It also communicates with auth to change the user's e-mail and to obtain the user's hashed password (required to approve profile changes).
- **PvP** → implements the PvP system between two users. PvP consists of having one's own team of 7 stands fight with the opponent's team. At the start of the fight a statistic is chosen at random from the stands and each stand is matched with another of the other player at random. Whoever wins gets **pvp score**, which appears in the player profile.

    * This service communicates with Profile to check if a profile exists and to add PvP scores, with Inventory to check if a stand is actually owned by the player. Communicates with Gacha to get the statistics of a stand.

- **databases** → are responsible for storing the persistent data of services. They have been specialised into 3 different types: **proxy**, **primary**, **replica**.
  - **proxy** (`proxysql/proxysql:2.7.1`) → Databases of type **proxy** act as database managers, i.e. they sort requests arriving from the 'auth' service to the 'auth_primary' or 'auth_replica' database. The replication will only receive SELECT requests, while those of other types, i.e. which update the data in some way, are sent to the primary. This division ensures better load management and redundancy.
  - **primary** (`mysql:9.1.0`) → Databases of type **primary** receive requests other than simple SELECT from the proxy and process them. They have two configured accounts (in addition to root): *monitor*, used by the proxy to check the status of the primary server, and *replication* which is used by the replica to copy the updated data and store it locally.
  - **replica** (`mysql:9.1.0`)→ Databases of type **replica** only receive SELECT requests from the proxy; replication is handled internally by MySQL, which uses the *replication* account to access the primary for: checking status and replicating data. Each service (except CDN, Redis and Admin) has at least its own proxy and primary. It has been chosen not to enable replication for all services for performance reasons, even though they are all configured to potentially enable it.

- **Redis** (`redis:7.4.1`) → is used to store the JWT tokens associated with each user at login time. It associates each `UserUUID` with a `JWT Token`. It is used by the **auth** service, which offers multiple endpoints to other services to verify the validity of the token.

- **CDN** (`python:3.13-alpine`) → is a simple Python-Flask server that offers endpoints and stores stand images. Endpoints to add/remove an image check the authenticity of the token by contacting **auth**.

- **API Gateway Public** (`nginx:alpine`) → allows access to the Docker network for all users. It has a `health_check` endpoint to check its health status. It is accessible on port 443 (HTTPS).

- **API Gateway Private** (`nginx:alpine`) → allows access to part of the Docker network for administrators. It has a `health_check` endpoint to check its health status. It is accessible on port 444. It is used to communicate with services: **auth**, **admin** and **graph**.

# 3 User Stories

## 3.1 USER

| **User Story** | **Endpoint** | **Microservice** |
|---|---|---|
| I WANT TO create my game account/profile SO THAT I can participate to the game | /register | Auth, Profile |
| I WANT TO delete my game account/profile SO THAT I can stop participating to the game | /profile/delete | Profile, Auth, Feedback, Currency, Inventory, Auction, PvP |
| I WANT TO modify my account/profile SO THAT I can personalise my account/profile | /profile/edit | Auth, Profile |
| I WANT TO login and logout from the system SO THAT I can access and leave the game | /login | Auth, Profile |
| I WANT TO be safe about my account/profile data SO THAT nobody can enter in my account and steal/modify my info | /login, /register | Auth, Profile |
| I WANT TO see my gacha collection SO THAT I know how many gacha I need to complete the collection | /gacha/list | Gacha, Inventory |
| I WANT TO want to see the info of a gacha of my collection SO THAT I can see all of info of one of my gacha | /gacha/list, /inventory/, /inventory/UUID | Gacha, Inventory |
| I WANT TO see the system gacha collection SO THAT I know what I miss of my collection | /gacha/list | Gacha, Inventory |
| I WANT TO want to see the info of a system gacha SO THAT I can see the info of a gacha I miss | /gacha/UUID | Gacha |
| I WANT TO use in-game currency to roll a gacha SO THAT I can increase my collection | /gacha/pull | Gacha, Currency, Profile, Inventory |
| I WANT TO buy in-game currency SO THAT I can have more chances to win auctions | /currency/buy/ID | Currency, Profile |
| I WANT TO be safe about the in-game currency transactions SO THAT my in-game currency is not wasted or stolen | /login, /register | Auth, Profile |
| I WANT TO see the auction market SO THAT I can evaluate if buy/sell a gacha | /auction/list | Auction, Inventory |
| I WANT TO set an auction for one of my gacha SO THAT I can increase in game currency | /auction/create | Auction, Inventory |
| I WANT TO bid for a gacha from the market SO THAT I can increase my collection | /auction/bid/UUID | Auction, Profile |

| User Story | Endpoint | Microservice |
|---|---|---|
| I WANT TO view my transaction history SO THAT I can track my market movement | /auction/history | Auction |
| I WANT TO receive a gacha when I win an auction SO THAT only I have the gacha a bid for | /auction/status/UUID | Auction, Inventory, Profile, Currency |
| I WANT TO receive in-game currency when someone win my auction SO THAT the gacha sell works as I expect | /auction/status/UUID | Auction, Inventory, Profile, Currency |
| I WANT TO receive my in-game currency back when I lost an auction SO THAT my in-game currency is decreased only when I buy something | /auction/bid/UUID | Auction, Profile |
| I WANT TO that the auctions cannot be tampered SO THAT my in-game currency and collection are safe | (built-in security) | (built-in security) |
| I WANT TO have different level of roll rarity SO THAT increase the probability to have more rare gachas | (built-in) | (built-in) |
| I WANT TO pvp mechanics SO THAT I can use my gachas against other players | /pvp/sendPvPRequest/UUID | PvP, Inventory, Profile |
| I WANT TO notify a problem to the system SO THAT administrators can resolve my problems | /feedback/ | Feedback, Profile |

## 3.2 ADMIN

| User Story | Endpoint | Microservice |
|---|---|---|
| I WANT TO login and logout as admin from the system SO THAT I can access and leave the game | /login, /register (from Private API Gateway) | Auth, Profile |
| I WANT TO check all users accounts/profiles SO THAT I can monitor all the users accounts/profiles | /admin/profile/list | Admin, Profile, Auth |
| I WANT TO check/modify a specific user account/profile SO THAT I can monitor and update a specific user account/profile | /admin/profile/UUID/edit | Admin, Profile, Auth |
| I WANT TO check a specific player currency transaction history SO THAT I can monitor the transactions of a player | /admin/profile/UUID/history | Admin, Profile, Auth, Currency |
| I WANT TO check a specific player market history SO THAT I can monitor the market of a player | /admin/profile/UUID/history | Admin, Profile, Auth, Currency |
| I WANT TO check all the gacha collection SO THAT I can check all the collection | /gacha/list | Admin, Auth, Gacha, Inventory |

| User Story | Endpoint | Microservice |
|---|---|---|
| I WANT TO modify the gacha collection SO THAT I can add/remove gachas | /admin/gacha/create, /admin/gacha/delete, /admin/gacha/update | Admin, Auth, Gacha |
| I WANT TO check a specific gacha SO THAT I can check the status of a gacha | /gacha/UUID | Gacha |
| I WANT TO modify a specific gacha information SO THAT I can modify the status of a gacha | /admin/gacha/update | Admin, Auth, Gacha |
| I WANT TO see the auction market SO THAT I can monitor the auction market | /auction/list | Auction, Inventory |
| I WANT TO see a specific auction SO THAT I can monitor a specific auction of the market | /auction/status/UUID | Auction, Inventory, Profile, Currency |
| I WANT TO modify a specific auction SO THAT I can update the status of a specific auction | /admin/auction/UUID/update | Admin, Auth, Auction |
| I WANT TO see the market history SO THAT I can check the market old auctions | /auction/list | Auction, Inventory |
| I WANT TO check the system logs SO THAT I can monitor the system behaviour | /admin/logs | Admin, Auth, Loki |
| I WANT TO ban a player account SO THAT punish cheating behaviour | /admin/profile/UUID/ban | Admin, Profile, Auth, Feedback, Currency, Inventory, Auction, PvP |

# 4 Market Rules

**Question 1:** What happens to the currency of a player when someone else bids higher?

**Answer:** The player who lost the bid gets their money back.

**Question 2:** What happens if I bid at the last second of the auction?

**Answer:** The auction ends the second later. Nothing else happens.

**Question 3:** Can I bid for an auction in which I am the highest bidder?

**Answer:** No. See 5.

**Question 4:** Can I try to sell an item that is not mine in the market?

**Answer:** No. See 5.

**Question 5:** Can I make multiple auctions for the same item?

**Answer:** No. See 5.

**Question 6:** Is there a minimum bid of in-game currency for an auction?


Figure 5: Illustrative Image: Auction Restriction

**Answer:** Yes, 1 in-game currency more than the previous bid, or the starting price if you are the first bidder.

**Question 7:** What happens if no one bids on my auctioned item?

**Answer:** The auction ends without a winner, and the item stays to your inventory.

**Question 8:** Is there a maximum bid limit for an auction?

**Answer:** No, you can bid as long as you have sufficient in-game currency.

**Question 9:** Can I extend the duration of my auction after it has started?

**Answer:** No, the auction duration is fixed to 10 minutes once the auction starts.

**Question 10:** Can I create a private auction visible only to specific players?

**Answer:** No, all auctions are public and visible to all players.

# 5 Testing

The integration and unit testing phase was conducted using *Postman*. In particular, attention was paid to the expected behaviour for each microservice, including error cases, by addressing requests both to the gateway and to the micro-services themselves, accessing them, in the Unit Testing, directly via an exposed port. Since each service is provided via both internal and external requests, we decided to create ad-hoc collections for each of the two cases; therefore, in the "Docs" folder you will see four types of collections, each with its own environment:

1. Integration Testing (External Requests), Environment: "Integration Testing".

2. Integration Testing (Internal Requests), Environment: "Integration Testing".

3. Unit Testing (External Requests), Environment: "Unit Testing".

4. Unit Testing (Internal Requests), Environment: "Unit Testing".

Each of these covers both success and error cases ;therefore, it was decided to include post-request scripts via *Postman* that test the expected behaviour. See 6 for reference.

For simplicity and greater clarity, given the volume of internal requests, it was decided to include deliberate erroneous fields for each request to test the desired error cases, and consequently, to omit the scripts. Descriptions on how to cause the desired behaviour are included for the user who wants to test internal endpoints.

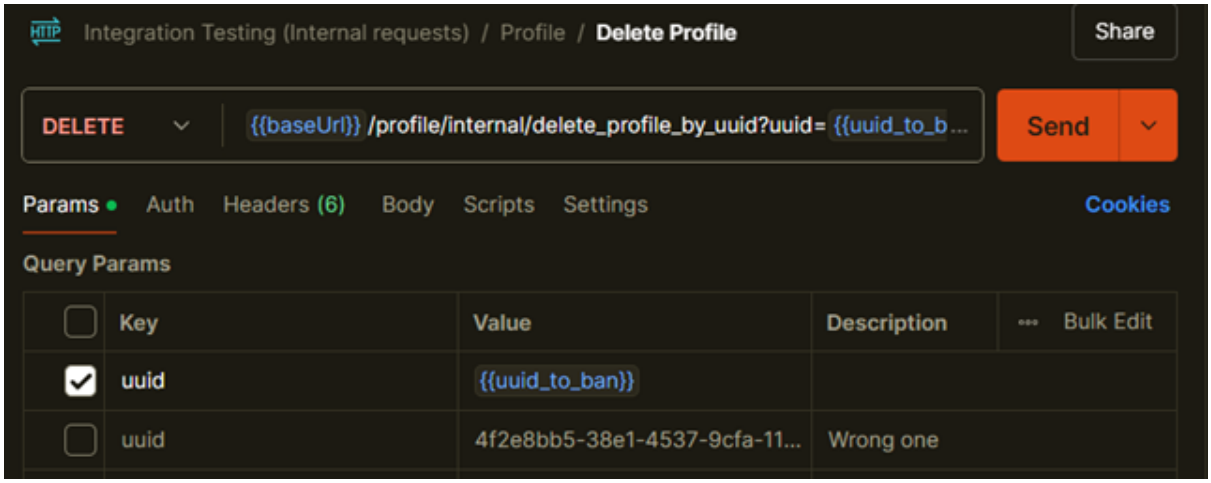


Figure 6: Postman Internal Request Example

Moreover, to properly test both the typical flow of a user and the correctness of actions, a collection of requests mimicking the potential behaviour of a user, involved in auctions and gacha rolls, was created and conveniently used in *GitHub Actions* to check for errors in the build at each push.

Additionally, performance tests were carried out using *Locust*. More specifically these tests were conducted to assess what bottlenecks the software had, the maximum number of users it could handle with an acceptable error rate and that the distribution of the rarity of the gacha pulls overall reflected the probabilities defined for each gacha pool.

As shown, the distributions actually reflect the defined probabilities. To obtain both of these results, we decided to create two files inside the `PERFORMANCE_TESTING` folder, `locustfile.py` and `locustfile_gacha.py`. Respectively used to test player operations involving gachas, and gacha pulls. After careful analysis, the results show that the auctions service soon becomes a bottleneck, and that the greatest number of failures is attributable precisely to the excessive load it has to handle and the resulting timeout errors it displays, along with its open circuit breaker.

This behaviour is expected and is due to the logic of an item's removal from one's inventory within the locust file. In particular, one can see in the code how an extensive analysis of all auctions is made to check whether one's item is included there before removing it, which means that although the number of auctions grows linearly, the number of calls to the endpoint that returns all auctions (/auction/list) grows exponentially, thus causing an understandable failure. Of course, it is expected from a normal user to find the

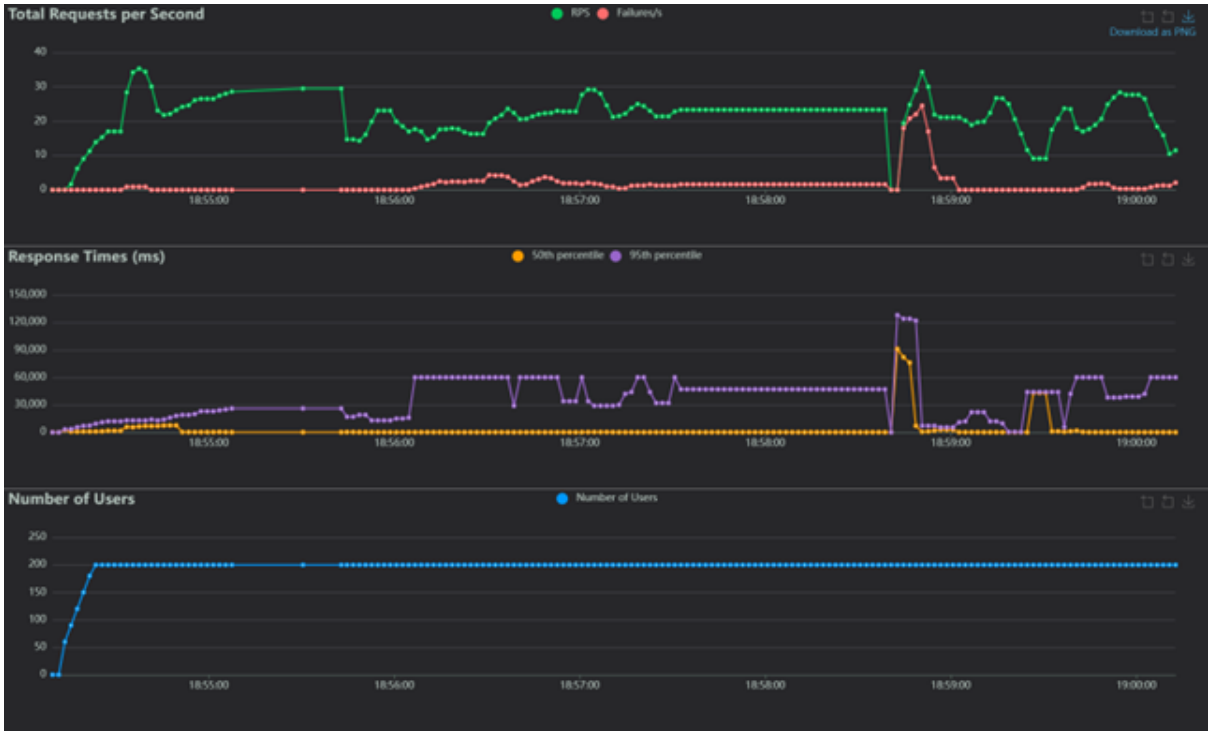


Figure 7: Locust Performance Test Results

relevant auction much sooner, but nevertheless, the software manages to handle **more than 200 users with ease**.

## 5.1 Security - Data

## 5.2 UUID Sanitation

UUIDs are the basis of the architecture of this project. They are used by all the services of the architecture to refer to a specific object. By their nature, they have a low probability of conflict and are difficult to guess, and are an excellent alternative to auto-increment ids. The sanitation of the various input types is handled by the `input_checks.py` file contained in the *common* folder. **sanitize_uuid_input** takes a UUID string as input and tries to convert it to a UUID object from the UUID library to check its validity. If it is a valid UUID, the function returns *true* and the UUID itself. If a valid UUID could not be created from the string, we try removing the - from the string and checking with a RegularExpression that the contents of the string without the - is the contents of an acceptable UUID. If true, we re-format the UUID by inserting the dashes correctly and return *true* and the correct UUID. If RegularExpression fails, sanitation fails and *false* is returned.

## 5.3 Database Encryption

It was decided to encrypt the password using the **bcrypt** library, a modern and secure solution that provides out-of-the-box password encryption. This library guarantees a high level of security but also includes a salting mechanism that effectively protects against dictionary and rainbow table attacks. We have used 12-round **bcrypt** making hashing more robust and resistant to attacks as the computational power increases.

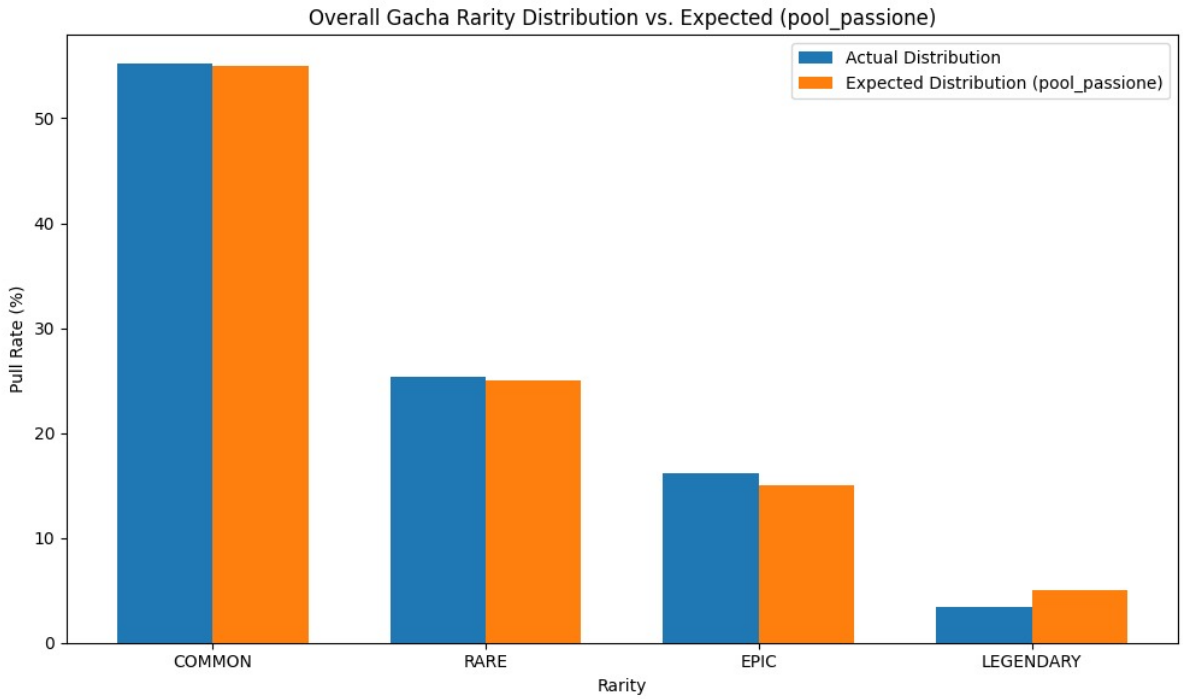


Figure 8: Rarity Distribution Graph: pool_passion

However, we encountered some limitations related to the version of MySQL used in our environment (9.1), which prevented us from enabling at-rest encryption of the database. Despite numerous searches in both official documentation and unofficial sources, we found no clear indications on how to enable at-rest encryption when MySQL is run inside a Docker container. This lack of information, coupled with compatibility constraints with the existing infrastructure and the need to keep the development and production environment stable, forced us to forego this additional security measure. Consequently, we concentrated on implementing the aforementioned encryption with **bcrypt**, to mitigate the risk of sensitive data being exposed.

# 6 Security - Authorisation and Authentication

We opted for the centralised scenario. Each service has stored a Python module `authorization.py` that implements the function **verify_login**. This function has 3 arguments (1 mandatory).

- **auth_header** → the actual HTTP request header (base64 encoded, as per standard)
- **audience_required** → string of your choice between 'public_services' and 'private_services'. Each service will pass to this parameter the level of Access that the user needs to have to access the service they offer. For example, Admin calls this function with the 'private_services' argument so that access will be denied to holders of the JWT token **auth** with the 'audience' field set to something other than 'private_services'.
- **service_type** → this parameter is passed solely for logging any errors caused by

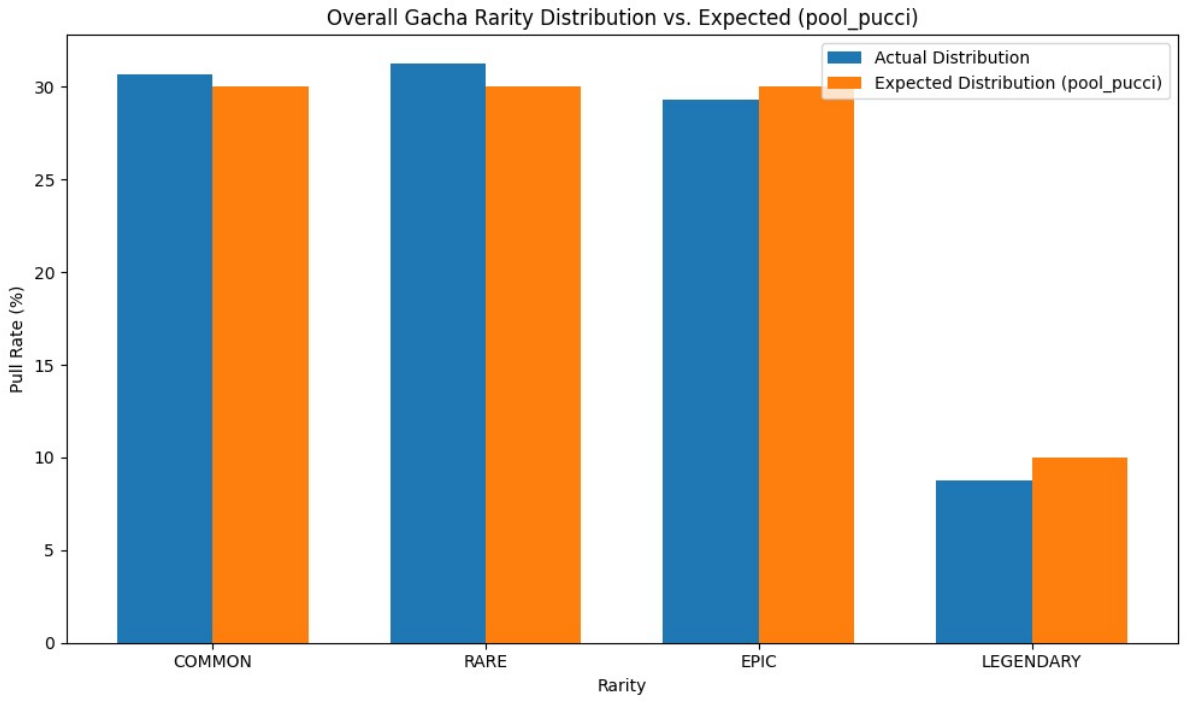


Figure 9: Rarity Distribution Graph: pool_pucci

incorrect parameter passing or for handling expected/unexpected errors during the call to *introspect* of **auth** for actual token verification.

Note that this function is only a utility for calling *introspect* of **auth**, so we can define our scheme as centralised.

The token visible in 10 is generated by the function `complete_access` shared by both the `register` and `login` of the **auth** service. This generates a token and returns it in the header of the client's request response as 'Bearer ¡token JWT¿'. The token is set to expire after 1 day. Before the return, the function stores the token on Redis to be accessed later by associating it with the value of the ¡useruuid¿ key.

When introspect is called, it verifies on Redis, after correctly decoding the token, that the raw token just sent is the same as the one stored on Redis. If this check fails, it means that Redis has been restarted or that the token was generated artificially and not by **auth** and is rejected.

Tokens are invalidated when an admin / user edits some properties in their profile that are also stored in the JWT Token.

## 6.1 Certificates

To enable a secure connection between the various services, we made use of certificates generated via the **OpenSSL** library. We generated a *Fake Root CA* and then created the following service-specific *Certification Authorities*: apigwCA, cdnCA, dbCA, redisCA and serviceCA. For each service, a different certificate was generated via the referring Certification Authority. The certificate of the specific CA is concatenated with that of the Root CA to enable the service to verify validity. All services use the certificates. Redis, Grafana and Loki do not verify certificates because they require that the certificate


Figure 10: Example JWT Token

provided to them is not self-signed, which is impossible in our case. Databases communicate with each other (proxy, primary and replication) in a secure manner. Services communicate with their database with TLS.

# 7 Security – Analyses

Static analysis of vulnerabilities in the code and its relative packages was conducted using Bandit, pip-audit and Github Dependabot. For vulnerabilities in the docker images built by us, we used Docker scout.

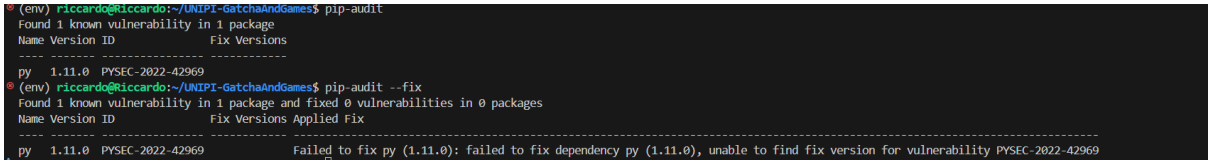


Figure 11: Pip Audit Scan Results

Taking as reference 11, despite trying fixing with the its built-in command, no recommendation were found. No documentation was found reporting this issue, so we considered it as a false alarm.

As reported in 12 while Bandit's indicates absence of a timeout with request, it jokingly reported the example of a call clearly configured with a timeout.

```
--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.8.0/plugins/b113_request_without_timeout.html
   Location: UNIPI-GatchaAndGames/services/profile/openapi_server/controllers/profile_controller.py:281:35
280             def delete_auth_user():
281                 delete_auth_response = requests.delete(
282                     f"{AUTH_SERVICE_URL}/auth/internal/delete_user_by_uuid",
283                     params={"uuid": session.get("uuid"), "session": None},
284                     verify=False,
285                     timeout=current_app.config["requests_timeout"],
286                 )
287                 delete_auth_response.raise_for_status()

--------------------------------------------------
>> Issue: [B501:request_with_no_cert_validation] Call to requests with verify=False disabling SSL certificate checks, security issue.
   Severity: High   Confidence: High
   CWE: CWE-295 (https://cwe.mitre.org/data/definitions/295.html)
   More Info: https://bandit.readthedocs.io/en/1.8.0/plugins/b501_request_with_no_cert_validation.html
   Location: UNIPI-GatchaAndGames/services/profile/openapi_server/controllers/profile_controller.py:312:23
311                 url = "https://service_auth/auth/internal/token/invalidate"
312                 response = requests.delete(url, params=params, verify=False, timeout=current_app.config["requests_timeout"])
313                 response.raise_for_status()
```

Figure 12: Bandit Security Audit Result

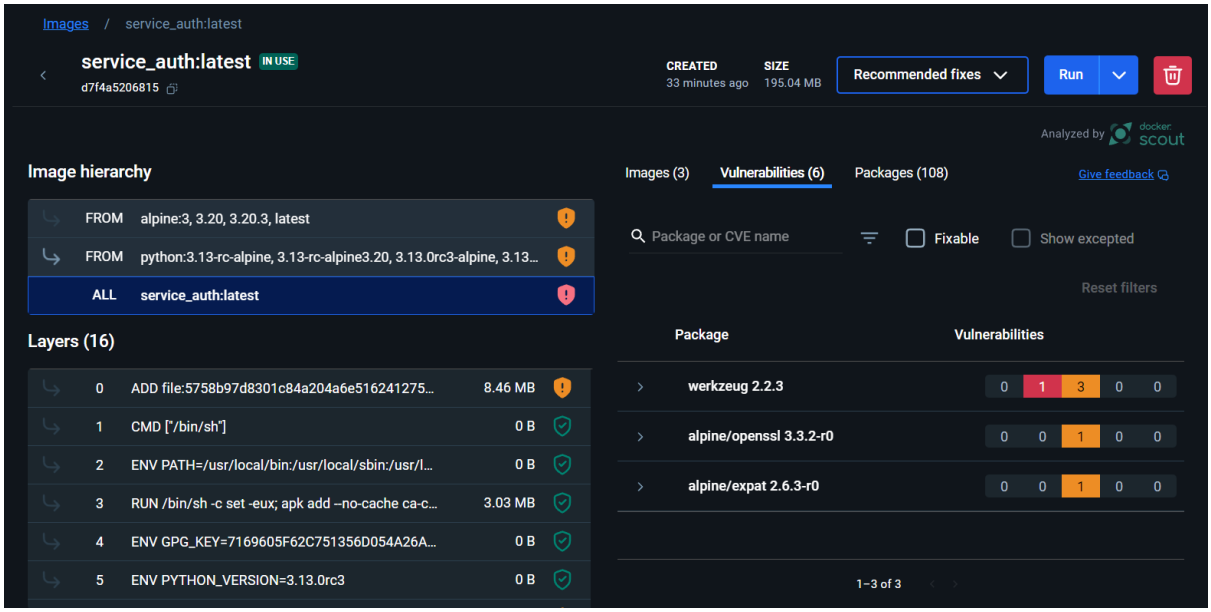


Figure 13: Mitigated Werkzeug Vulnerability

The only critical vulnerability identified in the **werkzeug 2.2.3** (CVE-2024-34069) package cannot be resolved in our infrastructure by simply updating the package. While upgrading werkzeug would address the vulnerability, our **openapi-generator** tool used does not support version 3 of **connexion**. Implementing this upgrade would break our build.

Despite this, disabling for each serving configuration mitigates this vulnerability (like we did). However, **docker scout** is unable to detect this configuration, and continues to report the vulnerability as present.

# 8 Additional Features

- **pvp** → allows teams of 7 stands of two players to play against each other. We decided to implement it to make the Jojo-themed Gacha experience more complete.

It's fun, but took a fair amount of development time.

- **various level of roll rarity** → implemented for greater variety in the gacha pull.
- **replications of DB** → allows better scaling in case of many connected users. It could be extended by increasing the number of replicas.
- **logging** → helped us find internal code issues. Nearly every event is logged, this required some time but was useful. It consists of two extra containers: **Loki** which takes care of storing the logs sent by the other services, offering a specific endpoint. It has an endpoint used by admin to retrieve logs based on certain attributes such as: service name, time, endpoint and severity level. The second container **Grafana** is a graphical interface for the logs, linked to **Loki**. It is accessible here and already has a dashboard set to 'Logging'.
- **further (optional) admin methods** → they have been added both to better test the application and to provide the admin with more functionality to manage the game.