



UNIVERSITÀ DI PISA

BeetleQuest

Advanced Software Engineering - Project Delivery

2024/2025

*Cosimo Giraldi
Giacomo Grassi
Michele Ivan Bruna*

Index

1. Introduction	3
2. Gacha Collection	3
3. Architecture	4
3.1. Design Choices	4
3.2. Microservices	5
3.2.1. Microservices connections	6
4. User Stories: Player	6
4.1. Account	7
4.2. Collection	7
4.3. Currency	7
4.4. Market	7
5. User Stories: Admin	8
5.1. Account	8
5.2. Gacha	8
5.3. Market	9
6. Market rules	9
7. Testing	10
8. Security	10
8.1. Data	10
8.2. Authentication and Authorization	10
8.3. Analyses	12
9. Additional features	13

1. Introduction

The goal of this project is to develop a web app and define its architecture for creating a web-based gacha game. So the users will be able to engage in all the standard activities found in a gacha game like: *roll*, *buy coin*, *create auctions*, *bid*.

All these actions will be implemented with **Go** language and through a *microservices* architecture.

2. Gacha Collection

The gachas are fictional creatures inspired by beetles, they are divided into five classes of rarity. Below are a few examples of these imaginative beings.

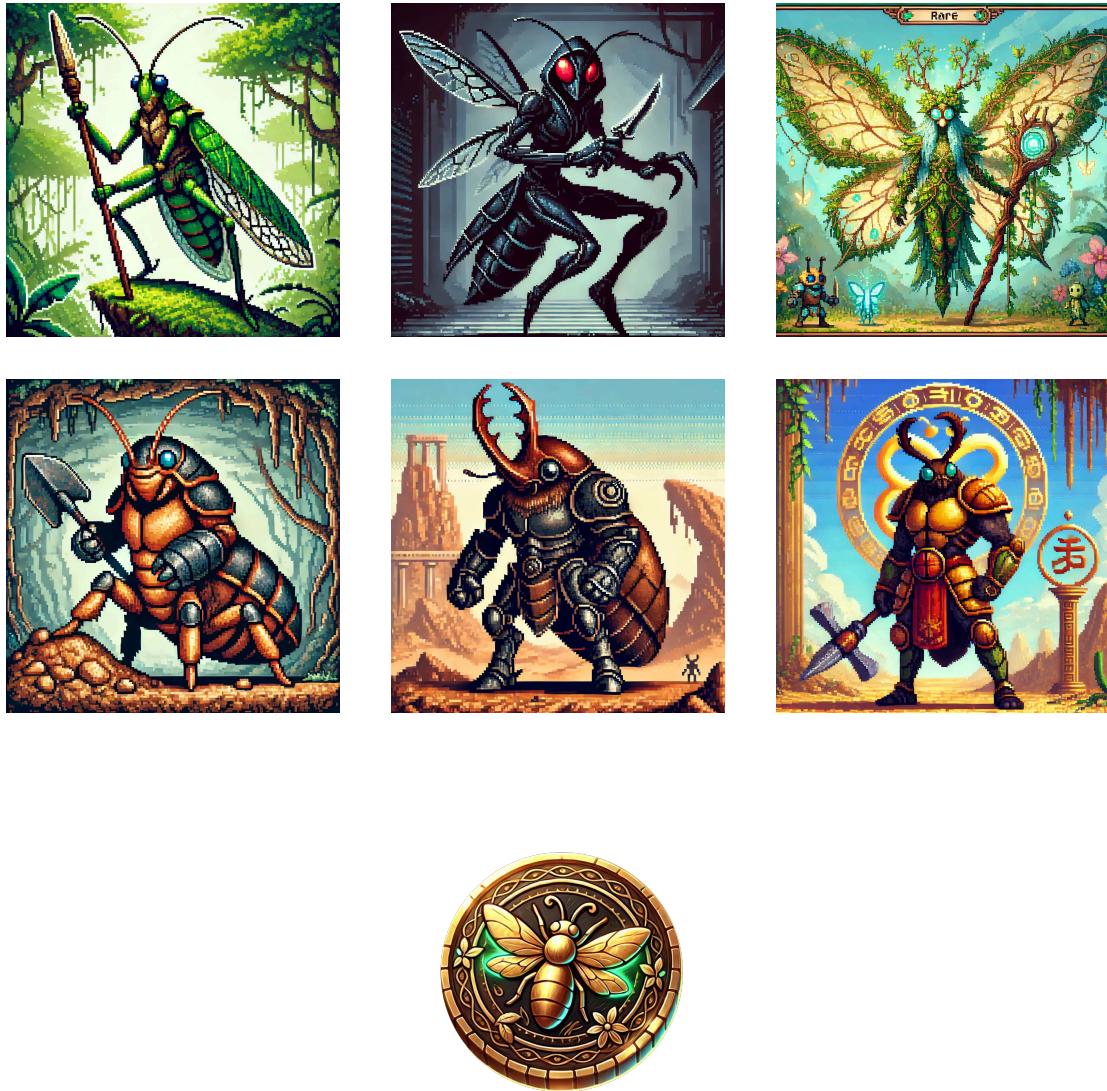


Figure 1: The currency used within the game is called BugsCoins

3. Architecture

The microservices architecture defined for this project is the result of a process of analysis and detection of the smells present in the original monolithic prototype, carried out using MicroFreshner.

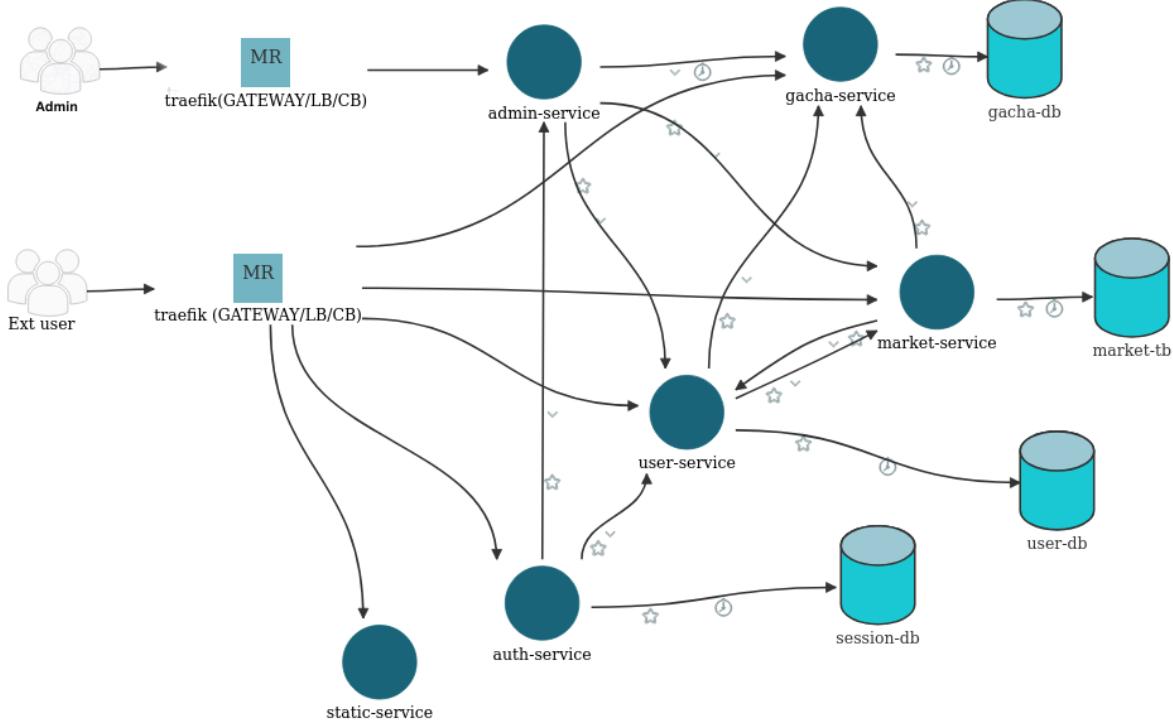


Figure 2: BeetleQuest architecture

3.1. Design Choices

The architectural analysis of our initial system, carried out using MicroFreshner, revealed smell between the microservices. To isolate potential failures and improve the system's resilience, we introduced Circuit Breakers (CBs).

The introduced Circuit Breakers effectively address the issues caused by continuous failures of a microservice, preventing the cascading propagation of errors that could slow down or completely halt the entire system.

To achieve more effective control over the system we have introduced **Timeouts** on database connections. This solution significantly improves resilience and reliability. If a connection or query exceeds the maximum time defined by the timeout, the system considers the operation as failed and immediately activates error-handling mechanisms, ensuring a quick response and preventing bottlenecks or slowdowns. We have also used a reverse proxy called **Traefik**, which acts as an intermediary between external users and the system's internal services. In this architecture, Traefik functions as an access gateway, managing and routing requests to the appropriate microservices, ensuring efficient and centralized traffic handling.

3.2. Microservices

The main idea was to divide a monolithic system into a series of microservices, each of which handles a specific functionality. This fragmentation allows for greater modularity and control of the system. To make the web-application more scalable, the microservices have been designed to be independent and stateless. Microservices that need to store data use their own dedicated database, which they access directly. However, if a service needs to access data managed by another service, it must use the internal API which is only accessible within the internal network.

In the following paragraph we will examine the implemented services, and their functionalities. Each service, except for the *Static service*, has its own PostgreSQL or Redis DB. Furthermore user sessions and market-timed-events, that will be discussed later, are stored in Redis DBs.

Auth

User registration, login and logout are all managed by the Auth service, which also checks the validity of access tokens, allowing authentication and authorization within the application.

User

This service is responsible for managing user's account informations. A user, once logged in, can access its account details, modify them or delete the account itself.

Gacha

The Gacha service manages collections, providing users with a list of available gachas and details about each one, as well as access to inspect the personal inventories of various players.

Market

The Market service allows users to perform actions involving the acquisition of BugsCoins and gachas. It manages auctions lifetime and transactions in the system. Through this service users can obtain gachas by either buying or rolling for a random gacha based on rarity.

Static

This service is responsible for serving the static content of the web-app, like the images, the *css* and the *html* files.

Admin

This service provides the administrator with the necessary tools to manage the system in a controlled manner: allowing operations on users, gacha and market services.

API gateway

There are two reverse proxies (the gateways to access the application) which use different technologies like circuit breakers, load balancers to handle requests. One gateway is exclusive for the admin's operation the other for the users' ones. Reverse proxies are implemented with Traefik.

3.2.1. Microservices connections

Admin-Service ↔ Gacha-Service:

The *admin service* connects with the *gacha service* to manage gacha, such as adding/delete/modify gachas.

Admin-Service ↔ Market-Service:

Admin service interacts with *market service* to regulate or manage the marketplace, including listing auctions, listing transactions, or update/modify auction.

Admin-Service ↔ User-Service:

The *admin service* connects with *user service* to manage user accounts, such as listing users, modifying user profiles, or checking user transaction history and the user auction list.

Auth-Service ↔ User-Service:

The *auth service* relies on *user service* for user data, such as validating credentials.

Gacha-Service ↔ User-Service:

The *user service* connects with the *gacha service* to retrieve owned gachas by an user.

Market-Service ↔ User-Service:

The *market service* connects with the *user service* to manage the user's currency and transactions, such as checking the user's currency and adding currency.

Market-Service ↔ Gacha-Service:

The *market service* connects with the *gacha service* to add a gacha when a user wins an auction or buy a gacha. Also to remove a gacha to the owner of a closing auction.

4. User Stories: Player

Evrey request has to pass through the *gateway*, the *auth-service* and the *session-db*, to check if it is a valid request. So those services are omitted in the list of the microservice(s) involved for the following requests.

4.1. Account

- I want to be able to register to the system, so that I can access the game.
 - ▶ `/auth/register` (*user-service, user-db*)
- I want to be able to delete my account, so that I can remove my information to the game.
 - ▶ `/user/account/{userId}` (*user-service, user-db/gacha-service, gacha-db/ market-service, market-db*)
- I want to be able to modify my account information, so that I can update my profile.
 - ▶ `/user/account/{userId}` (*user-service, user-db*)
- I want to be able to login and logout, so that I can access and leave the game. I want be safe from unauthorized access, so that my account access information is protected.
 - ▶ `/auth/logout, /auth/login` (*auth-service, auth-db, session-db*)

4.2. Collection

- I want to see my gacha collection, so that I can see what I have.
 - ▶ `/gacha/user/{userId}/list` (*gacha-service, gacha-db*)
- I want to see the info of a gacha in my collection, so that I can see the details of a gacha.
 - ▶ `/gacha/{gachaId}/user/{userId}` (*gacha-service, gacha-db*)
- I want to see the system gacha collection, so that I can see what I can get.
 - ▶ `/gacha/list` (*gacha-service, gacha-db*)
- I want to see the info of a gacha in the system collection, so that I can see the details of a gacha.
 - ▶ `/gacha/{gachaId}` (*gacha-service, gacha-db*)

4.3. Currency

- I want to use in-game currency to roll a gacha, so that I can get a random gacha.
 - ▶ `market/gacha/roll` (*user-service, user-db/market-service, market-db*)
- I want to buy in-game currency, so that I can get more gachas.
 - ▶ `/market/bugscoin/buy` (*user-service, user-db/market-service, market-db/user- service, user-db*)
- I want to be safe about the in-game currency transactions, so that my money is protected.
 - ▶ `/auth/logout, /auth/login` (*auth-service, session-db*)

4.4. Market

- I want to see the auction market, so that i can evaluate if buy/sell a gacha.
 - ▶ `/market/auction/list` (*market-service, market-db*)
- I want to set an auction for one of my gacha, so that I can sell it.

- ▶ `/market/auction/` (*user-service, user-db/gacha-service, gacha-db/market-service, market-db, market-timed-events*)
- I want to bid for a gacha from the market, so that I can buy it.
I want to receive a gacha when i win an auction, so that I receive a gacha.
I want to receive in-game currency when someone win my auction, so that I sell work as I expect.
I want to receive my in-game currency back when i lost an auction, so that my in-game currency.
I want to that the auctions cannot be tempered, so that my in-game currency and collection are safe.
- ▶ `/market/auction/{{auctionId}}/bid` (*user-service, user-db/market-service, market-db/gacha-service, gacha-db, market-timed-events*)
- I want to view my transaction history, so that I can track my market movements.
 - ▶ `/internal/market/get_transaction_history` (*user-service, user-db/market-service, market-db*)

5. User Stories: Admin

All the following endpoints requests involve the *admin-service*.

5.1. Account

- I want to login and logout as admin from the system, so that I can access and leave the game.
 - ▶ `/auth/admin/login,/auth/logout` (*auth-service, auth-db, session-db*)
- I want to check all users account/profile, so that I can monitor all the users accounts/profiles.
 - ▶ `/admin/user/get_all` (*user-service, user-db*)
- I want to check a specific user account/profile, so that I can monitor user account/profile.
I want to modify a specific user account/profile, so that I can update a specific user account/profile.
 - ▶ `/admin/user/{{userId}}` (*user-service, user-db*)
- I want to check a specific player currency transaction history, so that I can monitor the transactions of a player.
 - ▶ `/admin/user/{{userId}}/transaction_history` (*user-service, user-db/market-service, market-db*)
- I want to check a specific player market history, so that I can monitor the market of a player.
 - ▶ `/admin/user/{{userId}}/auction/get_all` (*user-service, user-db/market-service, market-db*)

5.2. Gacha

- I want to check all the gacha collection, so that I can check all the collection.

- ▶ `/admin/gacha/get_all` (*gacha-service,gacha-db*)
- I want to modify the gacha collection, so that I can add gachas.
 - ▶ `/admin/gacha/add` (*gacha-service,gacha-db*)
- I want to modify the gacha collection, so that I can delete gachas.
- I want to check a specific gacha, so that I can check the status of a gacha.
- I want to modify a specific gacha information, so that I can modify the status of a gacha.
 - ▶ `/admin/gacha/{{gachaId}}` (*gacha-service,gacha-db*)

5.3. Market

- I want to see the auction market, so that I can monitor the auction market.
 - ▶ `/admin/market/auction/get_all` (*market-service,market-db*)
- I want to see a specific auction, so that I can monitor a specific auction of the market.
- I want to modify a specific auction, so that I can update the status of a specific auction.
 - ▶ `/admin/market/auction/{{auction_id}}` (*market-service,market-db/gacha-service,gacha-db*)
- I want to see the market history, so that I can check the market old auctions.
 - ▶ `/admin/market/transaction_history` (*market-service,market-db*)

6. Market rules

The market service has been implemented with the following rules in mind:

- The user has the permission to create and delete it's own auctions but can not bid to them.
- The owner of an auction can delete it only when no bids have been made and the auction is open for less than 1/3 of the total time.
- When a user places a higher bid than the previous one nothing appends till the end of the auction, when all the losing bids will be refounded.
- If someone places a bid at the very last second of the auction, they will win the gacha as the last valid bidder.
- It's also possible to bid on an auction where you are already the highest bidder. The user cannot place a bid if they do not have the required amount of coins to bid.
- As the owner, you can delete the auction at any time before it expires, but you need to confirm the action by entering your password.
- The maximum duration of an auction is 24 hours.
- All bids will be refounded at the end of the auction, except for the highest one.
- All auctions remain visible to users, along with all the auction details. Additionally, all bids made are displayed.
- A user can auction the same gacha multiple times, but only one auction can be active at a time.

- The owner of an auction will get the gacha back if no one bids on it.
- The owner of an auction will get the money when the auction ends.

7. Testing

The tests were conducted using mocks that allowed isolated testing of individual services. These mocks simulate the behavior of external components enabling the verification of each service's functionality without relying on real external resources. Both unit and integration tests were carried out with Postman.

A performance testing tool, Locust, is used to perform load simulations and analysis of the service responses in various scenarios, ensuring an accurate assessment of the performance and robustness of each component.

Locust is also used to calculate the probability distribution of each rarity class of gacha, based on the number of rolls made by the users. Each locust run can add/remove gachas so the distribution can change between locust runs and does not present a fixed value.

8. Security

8.1. Data

All input data which goes into dbs is automatically sanitized thanks to GORM, a GO library used to communicate with databases, which will automatically escape arguments.

In the application, the databases are implemented using PostgreSQL or Redis. For PostgreSQL, Transparent Data Encryption (TDE) is used. TDE is a technology that protects sensitive data by encrypting the database files at rest. It ensures that data stored on disk is encrypted, making it inaccessible to unauthorized users or applications, while it automatically encrypts the data before it is written to disk and decrypts it when it is read.

On the other hand Redis data is not encrypted. This decision is mainly driven by its architecture as an in-memory database, which means that the data is not stored persistently on disk.

All connections between databases/services and services use mutual TLS (mTLS), ensuring secure communication and authentication between the involved parties .

8.2. Authentication and Authorization

The application is equipped with a centralised authentication and authorization management system. A middleware in the gateway delegates authentication to the *auth service*. Which will answer with a 2XX code if the access token is valid, otherwise the original request is rejected.

A user has to perform the following requests, irrelevant headers are omitted:

1. *Login*: provide user credentials and authenticate himself.

```
POST /api/v1/auth/login HTTP/1.1
Content-Type: application/json
Host: localhost
{
    "username": "admin",
    "password": "admin"
}

# Response
HTTP/2.0 302 Found
location: /api/v1/auth/authorizePage
set-cookie:
    go_session_id=ZGQxNmE50WUt0GJjYy00YjYxLWEwMTktNGQ1YjdjYzAxZ
    TNm.cfb4dfbd6ddf1da42c5cd21eaf5aad54d06ad6e; Path=/; Expires=Fri, 13 Dec 2024
    14:09:21 GMT; Max-Age=604800; HttpOnly; Secure
```

2. *Authorize*: authorize a client to access specific resource server.

```
GET /oauth/authorize?response_type=code&client_id=beetle-quest
&redirect_uri=https%3A%2F%2Flocalhost%2Fapi%2Fv1%2Fauth%2FtokenPage
&state=1234zyx&code_challenge=Fel21eLqcCtfPR-4P01pZh8wOHWOrnU2s1jrKj1_dbQ
&code_challenge_method=S256 HTTP/1.1
Cookie:
    go_session_id=ZGQxNmE50WUt0GJjYy00YjYxLWEwMTktNGQ1YjdjYz
    AxZTNm.cfb4dfbd6ddf1da42c5cd21eaf5aad54d06ad6e; Path=/; Expires=Fri, 13 Dec 2024
    14:09:21 GMT; Max-Age=604800; HttpOnly; Secure
Host: localhost

#Response
HTTP/2.0 302 Found
content-length: 0
date: Fri, 06 Dec 2024 14:13:02 GMT
location: https://localhost/api/v1/auth/tokenPage?code=Y2VKYMZH0GITOTNLZS0ZZGYZLWE
0MZITMZA1NGE5NGNKODA4&state=1234zyx
```

3. *Token*: exchange authorize code to retrieve access and id tokens.

```
POST /oauth/token HTTP/1.1
Content-Type: multipart/form-data; boundary=91c9b6f32ab6bf35-4fd95e306a9da8de-
ae6d91617e6618a3-c2427f089c8f8027
Cookie:
    go_session_id=ZGQxNmE50WUt0GJjYy00YjYxLWEwMTktNGQ1YjdjYzAxZTNm.cfb4dfbd6ddf1da42c5cd21eaf5aad54d06ad6e; Path=/; Expires=Fri, 13 Dec 2024 14:09:21
    GMT; Max-Age=604800; HttpOnly; Secure
Host: localhost

--91c9b6f32ab6bf35-4fd95e306a9da8de-ae6d91617e6618a3-c2427f089c8f8027
Content-Disposition: form-data; name="grant_type"

authorization_code
```

```

--91c9b6f32ab6bf35-4fd95e306a9da8de-ae6d91617e6618a3-c2427f089c8f8027
Content-Disposition: form-data; name="code"

Y2VKYMZHOGITOTNLZS0ZZGYZLWE0MZITMZA1NGE5NGNK0DA4
--91c9b6f32ab6bf35-4fd95e306a9da8de-ae6d91617e6618a3-c2427f089c8f8027
Content-Disposition: form-data; name="redirect_uri"

https://localhost/api/v1/auth/tokenPage
--91c9b6f32ab6bf35-4fd95e306a9da8de-ae6d91617e6618a3-c2427f089c8f8027
Content-Disposition: form-data; name="client_id"

beetle-quest
--91c9b6f32ab6bf35-4fd95e306a9da8de-ae6d91617e6618a3-c2427f089c8f8027
Content-Disposition: form-data; name="code_verifier"

Jso64mDhrRrtEZ5huMPut6la0aXoy2kevDpmUkqwJq4
--91c9b6f32ab6bf35-4fd95e306a9da8de-ae6d91617e6618a3-c2427f089c8f8027--

# Response
HTTP/2.0 200 OK
content-type: application/json; charset=UTF-8
{
    "access_token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJiZWV0bGU0bGUtcXVlc3QiLCJleHAiOjE3MzM1MDE3MjEsImhdCI6MTczMzQ5NDUyMSwiaXNzIjoiYmVldGx1LXF1ZXN0IiwbmJmIjoxNzMzNDk0NTIxLCJzdWIi0iIwOTA4N2Y0NS01MjA5LTR1ZmEtODViZC03NjE1NjJhNmRmNTMiLCJzY29wZSI6IiJ9.HRJMvO-DvRHEFYBMM6XEozlL5m8xn4JEuBeN1SU7-M5I0k4ySr8KDwP05o7e4f1SHChnRXH0h_X5PFLHN34xxVg",
    "expires_in": 7200,
    "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJiZWV0bGUtcXVlc3QiLCJleHAiOjE3MzM00TgxMjEsImhdCI6MTczMzQ5NDUyMSwiaXNzIjoiQmVldGx1IFF1ZXN0IiwbmJmIjoxNzMzNDk0NTIxLCJzdWIi0iIwOTA4N2Y0NS01MjA5LTR1ZmEtODViZC03NjE1NjJhNmRmNTMiFQ._RsinFKR9pnxNIJ8AMBD6o8dGIdY_wGkm4-PmvCyWn0",
    "refresh_token": "MTLHOTI2ZJKTMJY3NI01ZWYWLTK0YTETNGQ2NDGZYMIZM2JM",
    "token_type": "Bearer"
}

```

8.3. Analyses

For the static code analysis, `govulncheck` was used, which identifies vulnerabilities in Go dependencies by checking against the Go vulnerability database. Install `govulncheck` and then execute `govulncheck ./...` in the `beetle-quest/` directory, to obtain the following output.

```
== Symbol Results ==
```

```
No vulnerabilities found.
```

```
Your code is affected by 0 vulnerabilities.
```

```
This scan also found 0 vulnerabilities in packages you import and 1  
vulnerability in modules you require, but your code doesn't appear to call these  
vulnerabilities.
```

```
Use '-show verbose' for more details.
```

Listing 1: Govulncheck analyses output report

Meanwhile, for the analysis of Docker images, `trivy` was employed. In addition to analyzing images for CVEs using commands like `docker scan`, it also allows for the examination of Go binaries for vulnerable dependencies, misconfigurations, and potential leaks of secrets.

The scan results can be obtained executing `./scan-images.sh`, which is placed in `beetle-quest/tests/`, the output will be found inside `trivy_scan_results/` in the same folder. For the sake of space, we will only report the results of the summary of the scan on *admin service*, the other results are similar as all services images are based on `debian 12.8`.

```
beetle-quest-admin-service:latest (debian 12.8)
=====
Total: 7 (UNKNOWN: 0, LOW: 7, MEDIUM: 0, HIGH: 0, CRITICAL: 0)

Library: libc6
Vulnerabilities: CVE-2010-4756, CVE-2018-20796, CVE-2019-1010022, CVE-2019-1010023,
CVE-2019-1010024, CVE-2019-1010025, CVE-2019-9192
Severity: LOW
Status: affected
Installed Version: 2.36-9+deb12u9
Fixed Version: N/A
```

Listing 2: Trivy summary on the *admin service*

9. Additional features

The final application also incorporate several additional features to enhance its functionality and user experience.

From the security point of view a shared Certificate Authority (CA), public and private key, has been used in conjunction with mutual TLS (mTLS) between microservices, which will ensure secure communication between clients and servers by requiring both parties to authenticate each other.

The OAuth 2.0 protocol is implemented following the RFC 7636 Authorization Code Grant with PKCE standard, instead of the Password Grant one that nowadays is deprecated and it's use its discouraged¹.

Refresh tokens are also implemeted, which allows the client to obtain a new valid access token without the need to redo the full authorization procedure.

A simple web GUI has been developed to improve usability, providing only users (not admins) with an intuitive interface. Furthermore, a “buy gacha” feature has been be introduced, enabling users to directly acquire gachas.

¹The latest OAuth 2.0 Security Best Current Practice disallows the password grant entirely.
(<https://oauth.net/2/grant-types/password/>)