

REPORT

Table of Content

- Participants
- Gachas Overview
- Architecture
- User Stories
- Market Rules
- Testing
- Security
 - Data
 - Authentication and Authorization
 - Analyses

Members

- Luca Cremonese
- Davide di Rocco
- Nicolò Zarulli
- Jacopo Cioni

Gacha Overview



Rarità: Leggendaria
Velocità: 98
Esperienza: 88
Abilità: "Piede di Ferro"
Descrizione: Maestro della velocità pura, domina nelle gare di velocità con un controllo perfetto.



Rarità: Leggendaria
Velocità: 99
Esperienza: 90
Abilità: "Dominatrice dei Rettifili"
Descrizione: Con velocità impressionante nei rettilinei, sfrutta ogni occasione per sorpassare.



Rarità: Leggendaria
Velocità: 89
Esperienza: 90
Abilità: "Dominatrice dei Rettifili"
Descrizione: Con velocità impressionante nei rettilinei, sfrutta ogni occasione per sorpassare.



Rarità: Epica
Velocità: 92
Esperienza: 80
Abilità: "Manovra Fulminea"
Descrizione: Dominatore dei rettilinei, è rinomato per le sue manovre veloci e sicure.



Rarità: Rara
Velocità: 88
Esperienza: 77
Abilità: "Regina della Difesa"
Descrizione: Difende le sue posizioni con tenacia, rendendo ogni sorpasso un'impresa.



Rarità: Leggendaria
Velocità: 96
Esperienza: 90
Abilità: "Fulmine in Partenza"
Descrizione: Maestro delle partenze rapide, guadagna posizioni preziose subito allo start.



Rarità: Epica
Velocità: 92
Esperienza: 75
Abilità: "Maestro dei Circuiti Cittadini"
Descrizione: Pilota esperto nei circuiti cittadini, eccelle nelle curve strette e nei sorpassi ravvicinati.



Rarità: Rara
Velocità: 84
Esperienza: 83
Abilità: "Stratega della Qualifica"
Descrizione: Abile stratega nelle qualifiche, riesce a conquistare posizioni eccellenti in partenza.



Rarità: Rara
Velocità: 85
Esperienza: 86
Abilità: "Esperto delle Curve Lunghe"
Descrizione: Con maestria nelle curve lunghe, sfrutta tutta la potenza della vettura.



Rarità: Comune
Velocità: 80
Esperienza: 70
Abilità: "Veloce in Staccata"
Descrizione: Con una tecnica unica nelle frenate, mantiene alta la velocità in entrata di curva.



Rarità: Comune
Velocità: 78
Esperienza: 72
Abilità: "Veterano della Pioggia"
Descrizione: In condizioni di pioggia, mantiene il controllo come pochi, evitando errori e sbavature.



Rarità: Epica
Velocità: 82
Esperienza: 86
Abilità: "Mente Fredda"
Descrizione: Eccelle sotto pressione, mantenendo la calma anche nelle gare più tese e rischiose.



Rarità: Epica
Velocità: 88
Esperienza: 85
Abilità: "Spirito Combattivo"
Descrizione: Pilota aggressivo ma calcolato, non si arrende mai e lotta fino all'ultimo giro.



Rarità: Leggendaria
Velocità: 95
Esperienza: 85
Abilità: "Il Condottiero"
Descrizione: Conosciuto per il carisma in pista, guida le gare con autorità e intelligenza tattica.



Rarità: Comune
Velocità: 76
Esperienza: 74
Abilità: "Specialista del Gran Premio"
Descrizione: Consistente in ogni Gran Premio, raccoglie punti utili in ogni gara.



Rarità: Rara
Velocità: 85
Esperienza: 80
Abilità: "Sorpasso Letale"
Descrizione: Con una precisione unica, sorpassa gli avversari con audacia e sicurezza.



Rarità: Comune
Velocità: 75
Esperienza: 70
Abilità: "Esperto del Secondo Settore"
Descrizione: Abile nella gestione della parte centrale della pista, ottimizza ogni curva.



Rarità: Comune
Velocità: 74
Esperienza: 75
Abilità: "Consistenza Assoluta"
Descrizione: Pilota costante, sempre in grado di ottenere buoni piazzamenti senza rischiare troppo.

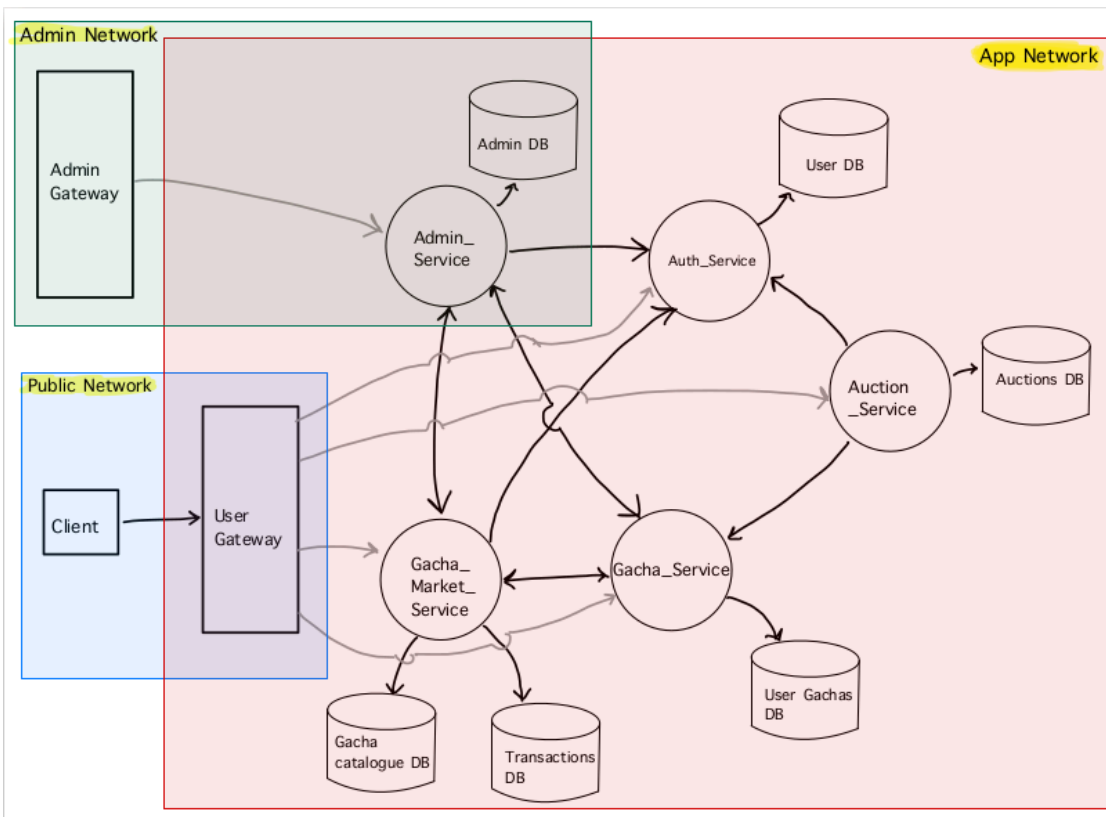


Rarità: Rara
Velocità: 80
Esperienza: 78
Abilità: "Esperto dei Pit Stop"
Descrizione: Riduce i tempi dei pit stop al minimo, ottimizzando le strategie di gara.



Rarità: Comune
Velocità: 78
Esperienza: 70
Abilità: "Accelerazione Istantanea"
Descrizione: Parte in modo fulmineo, guadagnando rapidamente posizioni nelle fasi iniziali.

Architecture



- **Authentication service:** lets user create an account, login, logout. Has endpoints to update the balance of users and to get user information such as ID
- **Gacha Service:** Lets the user see information about his gacha collection, add a gacha to his collection, retrieve the list of missing gachas. Has also an endpoint that gets called by the auction_service to update the new ownership of the gacha won during the auction. An admin is also able to call an endpoint inside this service to update some gacha information(for all users).
- **Gacha Market Service:** The user can see its transactions history, can buy in-game currency, purchase a roll and see the whole list of available gachas.
- **Auction Service:** Lets a user see the active auctions, create an auction and place a bid on an active auction
- **Admin Service:** Lets the admin authenticate, logout and manage user gacha collections.
- **User Nginx Gateway:** public endpoint connection for user actions.
- **Admin Nginx Gateway:** endpoint for admin connection and operations

Microservice	Connections	Details
Authentication service	To Gacha_service to delete user gacha collection when the account is deleted	Flask , Python
Gacha Service	To Gacha_Market_Service to retrieve the whole gacha catalogue. To Admin_Service to verify that the request made to spread the modification to a gacha to other users is made by an actual admin	Flask, Python
Gacha Market Service	To Auth_Service to update the currency when an user bought in-game currency and also to see if the user_id is actually a user. To the Gacha_service in order to add a gacha to a user collection and finally to the Admin_Service	Flask, Python

Microservice	Connections	Details
	to verify it is an actual admin that wants to modify gacha info ONLY in the catalogue	
Auction Service	To Gacha_Service to check if the user actually owns the gacha he is trying to sell and to update the ownership of that gacha when the auction has ended. To the Authentication_Service to check if the user has enough in game currency before bidding, to update the currency when an auction is not won and also when it is won (subtracting money or giving the user money back)	Flask, Python
Admin_Service	To Gacha_Market_Service to update gacha info in the catalogue. To the Authentication Service to obtain info about a user. To the Gacha_Service to spread the modification made to a gacha to all users owning that gacha	Flask, Python
User_Nginx_Gateway	to all the services	Nginx
Admin_Nginx_Gatway	to admin_Service	Nginx

User Stories

- 4. create my game account/profile
- 5. delete my game account/profile
- 6. modify my account/profile
- 7. login and logout from the system
- 8. be safe about my account/profile data
- 9. see my gacha collection
- 10. want to see the info of a gacha of my collection
- 11. see the system gacha collection
- 12. want to see the info of a system gacha
- 13. use in-game currency to roll a gacha
- 14. buy in-game currency
- 15. be safe about the in-game currency transactions
- 16. see the auction market
- 17. set an auction for one of my gacha
- 18. bid for a gacha from the market
- 19. view my transaction history
- 20. receive a gacha when I win an auction
- 21. receive in-game currency when someone win my auction
- 22. receive my in-game currency back when I lost an auction

- **23. that the auctions cannot be tampered**

Paths

- 4 --> [POST] localhost/authentication/account (User_gateway, AUthentication_Service, UsersDB)
- 5 --> [DELETE] localhost/authentication/account (User_gateway, AUthentication_Service, UsersDB)
- 6 --> [PATCH] localhost/authentication/account (User_gateway, AUthentication_Service, UsersDB)
- 7 --> [POST] localhost/authentication/auth // [PATCH] localhost/authentication/logout (User_gateway, AUthentication_Service, UsersDB)
- 8 --> TLS, login with password
- 9 --> [GET] localhost/gacha_service/players/gachas (User_gateway, gacha_service, issuedANDownedDB)
- 10 --> [GET] localhost/gacha_service/players/gachas/ (User_gateway, gacha_service, issuedANDownedDB)
- 11 --> [GET] localhost/market_service/catalog (user_gateway, gacha_market_service)
- 12 --> [GET] localhost/gacha_service/players/gachas/missing (User_gateway, gacha_service, gacha_market_service)
- 13 --> [POST] localhost/market_service/players/gacha/roll (User_gateway, gacha_market_service, AUthentication_Service, gacha_service)
- 14 --> [POST] localhost/market_service/players//currency/buy (User_gateway, gacha_market_service, AUthentication_Service)
- 15 --> TLS
- 16 --> [GET] localhost/auction_service/auctions/active (User_gateway, auction_service, auctions.db)
- 17 --> [POST] localhost/auction_service/players//setAuction (User_gateway, AUthentication_Service, auction_service, auctions.db)
- 18 --> [POST] localhost/auction_service/auctions//bid (User_gateway, AUthentication_Service, auction_service, AUthentication_Service, auctions.db)
- 19 --> [GET] localhost/market_service/players//transactions (User_gateway, AUthentication_Service, gacha_market_service, gacha_market.db)
- 20 --> [PATCH] localhost/gacha_service/players/{auction.current_user_winner_id}/gachas/{auction.gacha_id}/update_owner (auction_service, gacha_service)
- 21 --> [PATCH] localhost/authentication/players/{auction.issuer_id}/currency/update (auction_service, authentication_service)
- 22 --> [PATCH] localhost/authentication/players/{user_id}/currency/update (auction_service, authentication_service)
- 23 --> TLS

Market Rules

Whenever the user wants to auction off a gacha the he owns, he sets the base price and the auction appears to all the users. Whenever a users places a bid higher than the previous one, the previous bidder receives the money back and is able to bid again if he wants. Bids are placeable for the whole duration of the auction. The moment a user bids, funds are withdrawn from his balance. He will receive the funds back only in the moment a higher bid is placed. The highest bidder at the end of the time wins the auction and receives the gacha auctioned off by the original issuer.

In the eventuality the winning bidder places an even higher bid, it is an invalid behaviour. We put in place a control and warning.

Isolation Testing

The mock tests use simplified, in-memory data structures to emulate real-world data, referred to as "mokkate structures." These include:

- **User Data:** Simulated user credentials and session tokens.
- **Gacha Items:** Simplified representations of collectible items.
- **Market Transactions:** Mocked transactions for item trading.
- **Auction Bids:** Sample data for auction simulations.

The results of the mock tests are summarized in the log files located in

`docs/TEST/mock_tests_results`.

- **Running the Services**

The services in the `gacha_mock` folder can be executed using Docker Compose:

```
cd gacha_mock
docker-compose up --build
```

This command builds and starts all services defined in the `docker-compose.yml` file.

- **Running the Tests**

The tests in the `docs/TEST/mock_test_collection` folder can be executed using `newman`:

```
cd docs/TEST/mock_test_collection
newman run Admin_Service_Mock_Tests.json
newman run Authentication_Service_Mock_Tests.json
newman run Gacha_Service_Mock_Tests.json
newman run Gacha_Market_Service_Mock_Tests.json
newman run Auction_Service_Mock_Tests.json
```

Security-Data

One important input that has been sanitized is the **userId** and **gachaId** that is used inside the **/auction_service/players//setAuction**. Inside this endpoint, that is in the auction_service, we call the gacha_service with these parameters in order to verify that the user specified by that **userId** owns the gacha specified by **gachaId**. Since these parameters are used inside **gacha_service/players/{userId}/gachas/{gacha_id}** to build a query to the DB, we sanitized them using this regex.

```
# Validazione del parametro userId come numero intero
if not re.match(r'^\d+$', str(userId)):
    return make_response(jsonify({'message': 'Invalid user ID, must be a positive number'}), 400)

# Validazione del parametro gacha_id come numero intero
if not re.match(r'^\d+$', str(gacha_id)):
    return make_response(jsonify({'message': 'Invalid gacha_id, must be a positive number'}), 400)
```

Moreover, we sanitized the input from the **/authentication/account** function that lets a user create an account. Username and email are also sanitized in the ***/authentication/auth** during login time.

```
# Regex per validare la email
email_pattern = r'^[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$'
if not re.match(email_pattern, data['email']):
    return make_response(jsonify({'message': 'Invalid email format'}), 400)

# Regex per validare lo username
username_pattern = r'^[a-zA-Z0-9_-]{3,20}$'
if not re.match(username_pattern, data['username']):
    return make_response(jsonify({'message': 'Invalid username format. Only alphanumeric characters, underscores, dots, and hyphens are allowed. Length must be between 3 and 20.'}), 400)
```

Data At Rest

The only data at rest that is not left in clear-text is users and admin passwords. Here we use the `hashpw()` python function that implements OpenBSD-style Blowfish password hashing.

```
hashed_password = hashpw(data['password'].encode('utf-8'), gensalt())
```

Security-Authentication and Authorization

We employed a centralized way of JWT verification and validation. If requests are made to a service that is not `Admin_Service` or `Authentication_Service`, the incoming JWT is analyzed to check if it belongs to an Admin or a User and then, in case of a **User Request**, a request is made to **/authentication/validate**

in order to see if the JWT corresponds to that user by checking the users.db table. If the check goes right, it means that the user is currently logged in and a 200 response is sent back to the service originally asked to give access to a resource.

In case of an **Admin-Request**, the `/admin_service/verify_admin` endpoint is asked for validation, granting access to an admin if the actual admin is currently logged into the system.

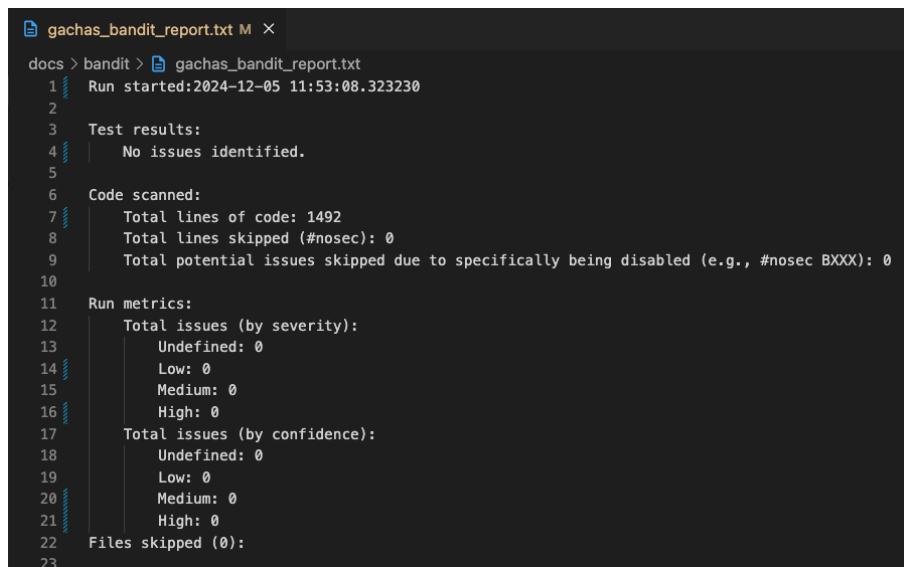
Here is the payload use to generate admin JWT:

```
def generate_jwt(admin):
    payload = {
        'admin_id': admin.id,
        'username': admin.username,
        'exp': datetime.now(timezone.utc) + timedelta(hours=2)  # Scadenza
del token in 2 ore
    }
    token = jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')
    return token
```

In the user case, `admin_id` is changed with `user_id`.

Security-Analyses

Bandit Result



```
gachas_bandit_report.txt M X
docs > bandit > gachas_bandit_report.txt
1 Run started:2024-12-05 11:53:08.323230
2
3 Test results:
4 | No issues identified.
5
6 Code scanned:
7 | Total lines of code: 1492
8 | Total lines skipped (#nosec): 0
9 | Total potential issues skipped due to specifically being disabled (e.g., #nosec BXXX): 0
10
11 Run metrics:
12 | Total issues (by severity):
13 |   Undefined: 0
14 |   Low: 0
15 |   Medium: 0
16 |   High: 0
17 | Total issues (by confidence):
18 |   Undefined: 0
19 |   Low: 0
20 |   Medium: 0
21 |   High: 0
22 Files skipped (0):
23
```

Docker hub Repo: https://hub.docker.com/r/shabby2/ase_project_gacha

Docker Scout Non Fixable Issues

shabby2/ase_project_gacha:latest

MANIFEST DIGEST

sha256:0c27a3adb047f7fa9c72c810ee4c43c889f258fce4ba2a6cda458b1b145a01ef

REGISTRY

Docker Hub

OS/ARCH

amd64

ENVIRONMENTS

-

VULNERABILITIES

000280

MANIFEST DIGEST

sha256:0c27a3...

Policy status (3/7)

Image layers

Exceptions

Learn more about policy

Image hierarchy

FROM debian:83bb9a422a3199968ff75bd730e4d13480dcbd...

FROM oisupport/staging-amd64:3.12-slim, 3.12-slim-bookwo...

ALL shabby2/ase_project_gacha:latest

Layers (17)

0

ADD rootfs.tar.xz / # buildkit

29.13 MB

1

CMD ["bash"]

0 B

2

ENV PATH=/usr/local/bin:/usr/local/s...

0 B

3

ENV LANG=C.UTF-8

0 B

4

RUN /bin/sh -c set -eux; apt-get updat...

3.51 MB

5

ENV GPG_KEY=7169605F62C751356...

0 B

6

ENV PYTHON_VERSION=3.12.7

0 B

7

ENV PYTHON_SHA256=24887b92e2af...

0 B

8

RUN /bin/sh -c set -eux; savedAptMar...

13.63 MB

9

RUN /bin/sh -c set -eux; for src in idle3...

250 B

10

CMD ["python3"]

0 B

11

WORKDIR /app

92 B

12

COPY app.py . # buildkit

4.57 KB

Images (3)

Vulnerabilities (28)

Packages (178)

Give feedback

Package or CVE name

Fixable

Show excepted

Reset filters

Package

Vulnerabilities

debian/glibc 2.36-9+deb12u9

00070

debian/systemd 252.31-1~deb

00040

debian/krb5 1.20.1-2+deb12u2

00030

debian/gcc-12 12.2.0-14

00020

debian/perl 5.36.0-7+deb12u1

00020

debian/coreutils 9.1-1

00010

debian/util-linux 2.38.1-5+deb1

00010

debian/tar 1.34+dfsg-1.2+deb1

00010

debian/libcrypt20 1.10.1-3

00010

1-10 of 15

Docker Scout Fixable issues = none

IMAGE

shabby2/ase_project_gacha:latest

MANIFEST DIGEST

sha256:0c27a3adb047f7fa9c72c810ee4c43c889f258fce4ba2a6cda458b1b145a01ef

REGISTRY

Docker Hub

OS/ARCH

amd64

ENVIRONMENTS

-

VULNERABILITIES

000280

MANIFEST DIGEST

sha256:0c27a3...

Policy status (3/7)

Image layers

Exceptions

Learn more about policy

Image hierarchy

FROM debian:83bb9a422a3199968ff75bd730e4d13480dcbd...

FROM oisupport/staging-amd64:3.12-slim, 3.12-slim-bookwo...

ALL shabby2/ase_project_gacha:latest

Layers (17)

0

ADD rootfs.tar.xz / # buildkit

29.13 MB

1

CMD ["bash"]

0 B

2

ENV PATH=/usr/local/bin:/usr/local/s...

0 B

3

ENV LANG=C.UTF-8

0 B

4

RUN /bin/sh -c set -eux; apt-get updat...

3.51 MB

5

ENV GPG_KEY=7169605F62C751356...

0 B

6

ENV PYTHON_VERSION=3.12.7

0 B

7

ENV PYTHON_SHA256=24887b92e2af...

0 B

8

RUN /bin/sh -c set -eux; savedAptMar...

13.63 MB

9

RUN /bin/sh -c set -eux; for src in idle3...

250 B

10

CMD ["python3"]

0 B

11

WORKDIR /app

92 B

12

COPY app.py . # buildkit

4.57 KB

Images (3)

Vulnerabilities (28)

Packages (178)

Give feedback

Package or CVE name

Fixable

Show excepted

Reset filters

Package

Vulnerabilities

No results found.

0-0 of 0