UNIVERSITY OF PISA

MASTER'S DEGREE IN CYBERSECURITY

GACHA LOLLECTION

COURSE SECURE SOFTWARE ENGINEERING

Author(s)
Fabbri Andrea
Lombardi Giacomo
Maldarella Giacomo

Academic year 2024/2025

Contents

CHAPTER *1*

Gachas Overview

1.1 Gacha items as Memes

In our implementation, each gacha item represents an **internet meme** (Figure 1.1).

Figure 1.1: *Examples of gacha items*

Each meme gacha item is characterized by the following information: **ID**, **Name**, **Image**, **Rarity**, and **Description**.
The official in-game currency is called **Memecoin**.

1.2 Rarity classification

Each meme is assigned one of three rarity levels, based on when the meme became famous, with older and more established memes considered rarer:

- **Common** Memes: relatively recent and still widely circulated in popular culture.
- **Rare** Memes: popular some years ago but less commonly seen today.
- **Legendary** Memes: widespread long ago and have achieved iconic status.

CHAPTER 2

Architecture

2.1 Microservices overview

The system is composed of specialized services, each dedicated to a specific functionality:

- **Authentication Service**: responsible for managing user authentication, including account creation, login/logout, token refresh, and account deletion. It ensures secure access by using JWT tokens for authentication.

- **Profile Setting Service**: manages player profiles (creation, modification, and deletion) and their associated data, including gacha collections and currency balances. It allows players to view details or update specific fields such as the profile image or email. It also manages the retrieval of players' gacha collections and the details of specific gacha items within their collection.

- **Gacha System Service**: manages all aspects related to gacha items within the platform. It allows administrators to add, update, delete, and retrieve gacha items from the collection. It is also responsible for extracting random gacha items based on the roll level when requested, ensuring that the probability of obtaining common, rare, or legendary gachas aligns with the specific roll level (standard, medium, or premium).

- **Gacha Roll Service**: enables players to perform a gacha roll at different levels (standard, medium, or premium). The different roll levels have varying probabilities for obtaining gachas of different rarities. When a player rolls, the system subtracts the required amount of currency from its account (communicating with the Payment Service), fetches a random gacha from the Gacha System (interacting with the Gacha System Service), and adds the obtained gacha to the user's profile (through the Profile Setting Service).

- **Auction Market Service**: allows users to manage auctions for gacha items. Sellers can create, modify, and close auctions, while bidders can place bids. The API handles actions like transferring the gacha item to the winner, refunding non-winning bidders, and ensuring that the auction process is properly managed. The system also allows for auction status updates and bid management.
- **Payment Service**: manages player transactions and balances. It enables payments, purchase of in-game currency using real money, and allows users to view their transaction history. In addition, it supports the creation, retrieval, and deletion of players balances.

In addition, there are two gateways responsible for routing requests:

- **Gateway**: serves as the entry point for player operations, directing requests to the appropriate services for player interactions.
- **Admin Gateway**: serves as the entry point for administrator operations, directing requests to the appropriate services for admin interactions.

All the services within the system have been developed using **Python** and, for building the APIs, the `Flask` framework has been used, providing simplicity and flexibility in managing HTTPS requests and responses.

2.2 Microservices Interactions

2.2.1 Authentication Interactions

Authentication interacts with Profile Setting during user signup process, to initialize a new user profile (e.g., username, email, and default profile image), and deletion process, to remove the associated user profile. Authentication connects to Payment during signup, to initialize the user's currency balance, and deletion to remove the user's payment data.

2.2.2 Profile Setting Interactions

Profile Setting interacts with Gacha System to retrieve details about gachas owned by a player, and with Payment to retrieve the user's currency balance.

2.2.3 Gacha System Interactions

Gacha System interacts with Profile Setting when an admin deletes a gacha item to remove the gacha from user profiles.

2.2.4 Gacha Roll Interactions

Gacha Roll interacts with Payment to handle the payment when a new roll is performed, with Gacha System to obtain a gacha, and with Profile Setting to insert it into user's profile.

2.2.5 Auction Market Service

Auction Market is connected with Profile Setting to remove a gacha from user's profile when an auction is created and to add it to the profile of the user who wins the auction. Auction Market interacts with Payment to manage the payment for a bid, refund users who lose an auction, and add currency to the seller.

2.3 Architecture Diagram

The diagram below (Figure 2.1), created with MicroFreshener, visually represents the architecture and its components, including the relationships and interactions among services. MicroFreshener was also utilized to analyze the architecture for potential architectural smells, such as no API gateway, wobbly service interaction, and shared persistency.

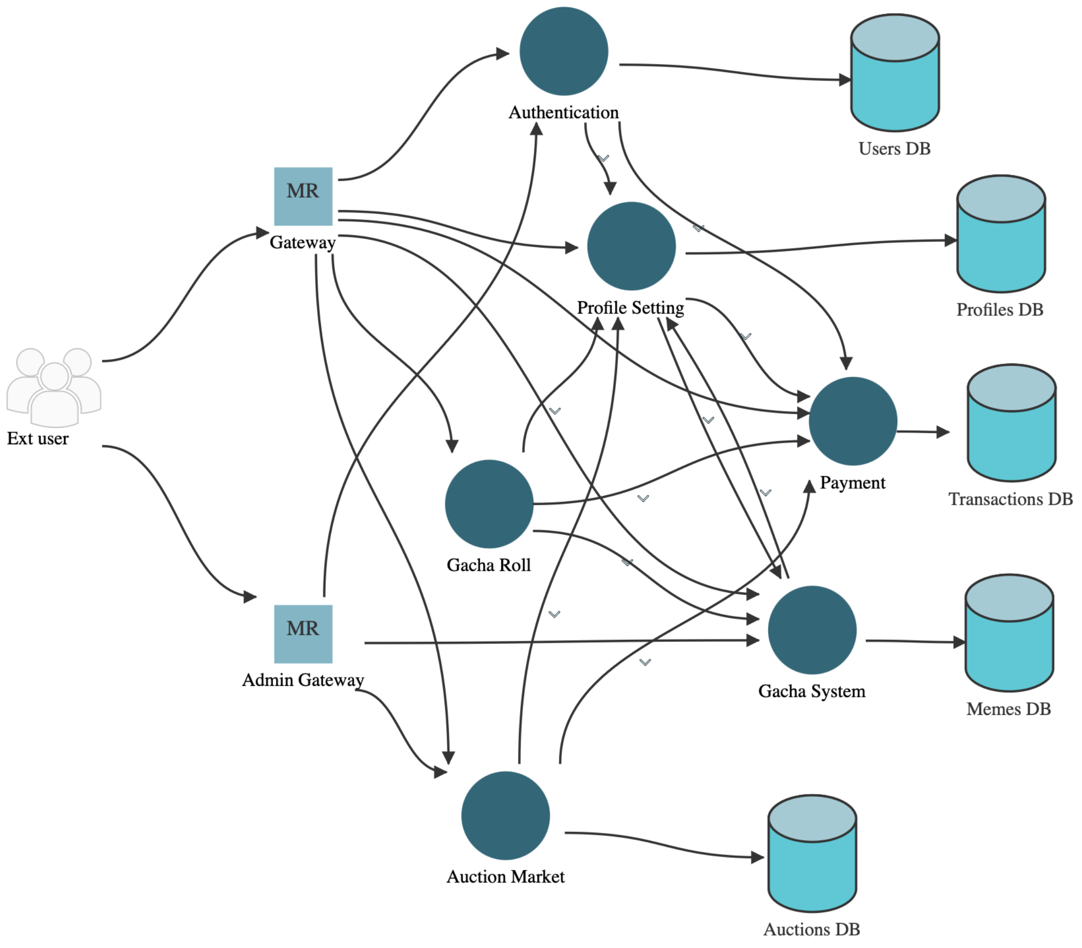


Figure 2.1: *Architecture with MicroFreshner*

CHAPTER 3

User Stories

3.1 Player User Stories

3.1.1 Account

1. *AS A player I WANT TO create my game account/profile SO THAT I can participate in the game*

   - **Endpoint**: `/signup`
   - **Microservices Involved**: Gateway, Authentication, Users DB, Profile Setting, Profiles DB, Payment, Transactions DB

2. *AS A player I WANT TO delete my game account/profile SO THAT I can stop partecipating to the game*

   - **Endpoint**: `/delete`
   - **Microservices Involved**: Gateway, Authentication, Users DB, Profile Setting, Profiles DB, Payment, Transactions DB

3. *AS A player I WANT TO modify my account/profile SO THAT so I can personalize my account/profile*

   - **Endpoint**: `/modify_profile`
   - **Microservices Involved**: Gateway, Profile Setting, Profiles DB

4. *AS A player I WANT TO login and logout from the system SO THAT I can access and leave the game*

   - **Endpoint**: `/login` and `/logout`

- **Microservices Involved**: Gateway, Authentication, Users DB

5. *AS A player I WANT TO be safe about my account/profile data SO THAT nobody can enter in my account and steal/modify my info*

    - **Endpoint**: `/signup` and `/login`
    - **How**: passwords are stored as *Hash(Hash(password) + Salt)* and access token is used for authorization management.

3.1.2 Collection

1. *AS A player I WANT TO see my gacha collection SO THAT I know how many gacha I need to complete the collection*

    - **Endpoint**: `/retrieve_gachacollection`
    - **Microservices Involved**: Gateway, Profile Setting, Profiles DB, Gacha System, Memes DB

2. *AS A player I WANT TO see the info of a gacha of my collection SO THAT I can see all of info of one of my gacha*

    - **Endpoint**: `/info_gachacollection`
    - **Microservices Involved**: Gateway, Profile Setting, Gacha System, Memes DB

3. *AS A player I WANT TO see the system gacha collection SO THAT I know what I miss of my collection*

    - **Endpoint**: `/info_gachacollection` or `/get_gacha_collection`
    - **Microservices Involved**: Gateway, Profile Setting, Gacha System, Memes DB or Gateway, Gacha System, Memes DB

4. *AS A player I WANT TO see the info of a system gacha SO THAT I can see the info of a gacha I miss*

    - **Endpoint**: `/info_gachacollection` or `/get_gacha_collection`
    - **Microservices Involved**: Gateway, Profile Setting, Gacha System, Memes DB or Gateway, Gacha System, Memes DB

3.1.3 Currency

1. *AS A player I WANT TO use in-game currency to roll a gacha SO THAT I can increase my collection*

    - **Endpoint**: `/gacharoll`
    - **Microservices Involved**: Gateway, Payment, Transactions DB, Gacha System, Memes DB, Profile Setting, Profiles DB

2. *AS A player I WANT TO buy in-game currency SO THAT I can have more chances to win auctions*

- **Endpoint**: `/buycurrency`
- **Microservices Involved**: Gateway, Payment, Transactions DB

3. *AS A player I WANT TO be safe about the in-game currency transactions SO THAT my in-game currency is not wasted or stolen*

   - **Endpoint**: `/pay` and `/login`
   - **Microservices Involved**: Payment, Transactions DB and Gateway, Authentication, Users DB

3.1.4 Market

1. *AS A player I WANT TO see the auction market SO THAT I can evaluate if buy/sell a gacha*

   - **Endpoint**: `/see`
   - **Microservices Involved**: Gateway, Auction Market, Auctions DB

2. *AS A player I WANT TO set an auction for one of my gacha SO THAT I can increase in game currency*

   - **Endpoint**: `/create`
   - **Microservices Involved**: Gateway, Auction Market, Auctions DB, Profile Setting, Profiles DB

3. *AS A player I WANT TO bid for a gacha from the market SO THAT I can increase my collection*

   - **Endpoint**: `/bid`
   - **Microservices Involved**: Gateway, Auction Market, Auctions DB, Payment, Transactions DB

4. *AS A player I WANT TO view my transaction history SO THAT I can track my market movement*

   - **Endpoint**: `/viewTrans`
   - **Microservices Involved**: Gateway, Payment, Transactions DB

5. *AS A player I WANT TO receive a gacha when I win an auction SO THAT only I have the gacha a bid for*

   - **Endpoint**: `/gacha_receive`
   - **Microservices Involved**: Auction Market, Auctions DB, Profile Setting, Profiles DB

6. *AS A player I WANT TO receive in-game currency when someone win my auction SO THAT the gacha sell works as I expect*

   - **Endpoint**: `/auction_terminated`

- **Microservices Involved**: Auction Market, Auctions DB, Payment, Transactions DB

7. *AS A player 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*

   - **Endpoint**: `/auction_lost`
   - **Microservices Involved**: Auction Market, Auctions DB, Payment, Transactions DB

8. *AS A player I WANT TO that the auctions cannot be tampered SO THAT my in-game currency and collection are safe*

   - **Endpoint**: `/login`
   - **Microservices Involved**: Gateway, Authentication, Users DB

3.1.5 More features

1. *AS A player I WANT TO have different level of roll rarity SO THAT increase the probability to have more rare gachas*

   - **Endpoint**: `/gacharoll`
   - **Microservices Involved**: Gateway, Payment, Transactions DB, Gacha System, Memes DB, Profile Setting, Profiles DB
   - **Notes**: specify the level of roll in the field `level`.

2. *AS A player I WANT TO a beautiful graphical user interface SO THAT I feel motivated to play everyday*

   - **Files**: `/index.html`, `/styles.css`, `/app.js`

3.2 Admin User Stories

3.2.1 Profiles/Accounts

1. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO signup as admin SO THAT I can administrate the system*

   - **Endpoint**: `/signup`
   - **Microservices Involved**: Admin Gateway, Authentication, Users DB, Profile Setting, Profiles DB, Payment, Transactions DB

2. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO delete my game account/profile SO THAT I can stop managing the system with the current account*

   - **Endpoint**: `/delete`
   - **Microservices Involved**: Admin Gateway, Authentication, Users DB, Profile Setting, Profiles DB, Payment, Transactions DB

3. *AS AN administrator I WANT TO login and logout as admin from the system SO THAT I can access and leave the game*
    - **Endpoint**: `/login` and `/logout`
    - **Microservices Involved**: Admin Gateway, Authentication, Users DB

3.2.2 Gachas

1. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO check all the gacha collection SO THAT I can check all the collection*
    - **Endpoint**: `/get_gacha_collection`
    - **Microservices Involved**: Admin Gateway, Gacha System, Memes DB
2. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO modify the gacha collection SO THAT I can add/remove gachas*
    - **Endpoint**: `/add_gacha` and `/delete_gacha`
    - **Microservices Involved in adding a gacha**: Admin Gateway, Gacha System, Memes DB
    - **Microservices Involved in deleting a gacha**: Admin Gateway, Gacha System, Memes DB, Profile Setting, Profiles DB
3. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO check a specific gacha SO THAT I can check the status of a gacha*
    - **Endpoint**: `/get_gacha_collection`
    - **Microservices Involved**: Admin Gateway, Gacha System, Memes DB
    - **Notes**: Specify the name(s) of the gacha(s) you want to obtain information about
4. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO modify a specific gacha information SO THAT I can modify the status of a gacha*
    - **Endpoint**: `/update_gacha`
    - **Microservices Involved**: Admin Gacha, Gacha System, Memes DB

3.2.3 Market

1. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO see the auction market SO THAT I can monitor the auction market*
    - **Endpoint**: `/see`
    - **Microservices Involved**: Admin Gateway, Auction Market, Auctions DB
2. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO see a specific auction SO THAT I can monitor a specific auction of the market*
    - **Endpoint**: `/see`

- **Microservices Involved**: Admin Gateway, Auction Market, Auctions DB
- **Notes**: specify the id of the auction you want to see

3. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO modify a specific auction SO THAT I can update the status of a specific auction*

   - **Endpoint**: `/modify`
   - **Microservices Involved**: Admin Gateway, Auction Market, Auctions DB

4. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO see the market history SO THAT I can check the market old auctions*

   - **Endpoint**: `/see`
   - **Microservices Involved**: Admin Gateway, Auction Market, Auctions DB
   - **Notes**: you can also specify the status of the actions you want to check (e.g. "active")

3.2.4 More features

1. *[ADDITIONAL FEATURE] AS AN administrator I WANT TO a beautiful graphical user interface SO THAT I feel motivated to manage the system*

   - **Files**: `/admin_client/index.html`, `/admin_client/styles.css`, `/admin_client/app.js`

CHAPTER 4

Market Rules

In the game, players can acquire gachas not only through rolls but also by purchasing and selling items through auctions. Auctions involve real-time bidding, with the highest bidder winning the item at the end of the auction. Sellers can list their items at a desired starting price and players can compete to place the highest bid before the auction time runs out. In this chapter, we will analyze some important details about the auction market, including the rules and mechanics that govern the bidding process.

4.1 Starting an auction

An auction is initiated by the seller, who lists an item for auction. As soon as the auction is created, the gacha is immediately removed from the seller's profile. The seller also sets a base price, which represents the minimum acceptable bid for the auction. The auction has a predefined end time, which cannot be extended once it's set. Once the auction is created, it becomes open for bids from other players.

4.2 Bidding Process and Bid Controls

Players can place a bid only when the auction is open. To place a bid, the amount must be higher than both the current highest bid and the base price set by the seller. When a player places a bid, the amount is immediately deducted from their account and transferred to the system's fund. If the player had previously placed a bid but was outbid by someone else, the system will only deduct the difference between their previous bid and the new higher bid. For instance, if a player initially bid 10 and later bids 30, the system will subtract 20 (the difference) from their account, ensuring that only the new bid amount of 30 is transferred to the system. Additionally, the current highest bidder cannot place another bid, as they are already in the lead, and the auction creator is not permitted to place a bid on their own auction.

4.3 Bidding at the Last Second

Players can place bids up until the final second of the auction, as long as the auction is still open. However, placing a bid at the last second does not extend the auction's duration. The auction will close exactly at the scheduled time, regardless of any bids placed in the final moments.

4.4 What Happens When an Auction Ends

- **No Bids**: if no bids were placed by the end of the auction (i.e., current bid is 0), the gacha item is returned to the seller and no money is exchanged.
- **Bids Were Placed**: the highest bidder wins the auction and receives the gacha item. All other bidders are refunded the amount they bid. The seller receives the winning bid amount from the system.

4.5 Who Can Anticipate the End of an Auction

- **Auction Creator**: the auction creator has the right to close the auction early, but only if no bids have been placed.
- **Admin**: the admin can close the auction whenever he wants.

CHAPTER 5

Testing

Testing is a critical part to ensure the reliability and performance of the system. Our testing strategy involved functional tests (unit and integration tests) to find bugs of individual components and their interactions, as well as performance tests to measure system responsiveness under load. Below, we describe the testing methodologies, tools, and processes employed during development.

5.1 Unit Testing

Unit tests were implemented for each microservice to validate individual functionalities in isolation. To isolate dependencies, all external interactions with services or databases are mocked, simulating real responses.

5.1.1 Key Features of Unit Testing

- **Mocking Dependencies**: each microservice includes an `app_test.py` file that mirrors the `app.py` logic but replaces real interactions with mocked versions. This ensures the tests do not rely on external systems or actual database connections. As an example, in the Gacha Roll service, the following interactions were mocked:
  - **Payment Service**: to simulate payments, a `mock_payment` function was created to return a successful payment response.
  - **Gacha System Service**: the `mock_gachasystem` function simulated the fetching of a gacha based on the specified level.
  - **Profile Setting Service**: the `mock_profile` function mimicked the process of inserting a gacha item into the user's profile.

Here is a snippet of the `app_test.py` file from the Gacha Roll service, illustrating the use of mocks:

```python
def mock_payment(payer, receiver, amount):
    return jsonify({'msg': 'Correctly payed'}), 200

def mock_gachasystem(level):
    gacha_details = {
        "gacha_id": "1",
        "gacha_name": "Trial",
        "description": "",
        "rarity": "standard",
        "img": f"https://localhost:5001/images_gacha/uploads/"
    }
    return gacha_details, 200

def mock_profile(gacha):
    return jsonify({"msg": "gacha correctly received"}), 200
```

Additionally, JWT decoding was mocked.

- **Postman Collection**: each microservice has a dedicated Postman test collection file in its directory.

- **GitHub Actions Workflow**: unit tests are executed in a dedicated unit-tests job in the CI/CD pipeline, with each service tested in isolation.
  The key steps in the workflow are:

  1. Checkout the repository.
  2. Start the service container using docker compose.
  3. Run Newman tests against the service-specific Postman collection.
  4. Stop and remove the service container after tests complete.

5.2 Integration Testing

Integration tests verify the interaction between microservices, ensuring the flow of data and logic is consistent across the system. These tests are designed using Postman, with all test collections and environment files stored in a dedicated `/tests` directory.
The integration-test job in our workflow runs all microservices together using Docker Compose. Newman executes integration tests defined in a shared Postman collection. Write here any particular fact about the testing.

- Services are launched concurrently with Docker Compose.

- Tests ensure interactions between services are functioning correctly.

- The `insecure` flag is used to bypass certificate verification for compatibility during testing.

5.3 Performance Testing

The performance test was conducted using Locust and was designed to simulate the actions of users interacting with the application. These actions are defined as tasks in the `locustfile.py`. The test was designed to run multiple tasks concurrently, simulating the behavior of many users interacting with the system at the same time.
Additionally, the test handles token expiration by requesting a refresh of the access token when it expires. This is necessary because the system assigns an access token to each user, which has a short lifespan of 5 minutes, along with a refresh token, which lasts longer. This approach was adopted because the token management system is distributed: each service can independently verify the validity of the access token it receives using the public key associated to the private key used to sign the token. By using a refresh token, the revocation process is centralized within the authentication service, ensuring that when users log out, their refresh token is invalidated. This eliminates the need for a blacklist of expired access tokens across all services.
[ADDITIONAL FEATURES] The performance test covers all user operations *(Not just endopoints involving gacha operations)*, such as performing a gacha roll, creating an auction, placing bids on active auctions, and more.

CHAPTER 6

Data Security

6.1 Protection of Data in Transit

To ensure the protection of sensitive data during transmission, the system employs HTTPS for all communication between services. This is achieved by utilizing TLS, which encrypts the data exchanged, preventing unauthorized access and tampering during transmission.
The following steps were taken to implement HTTPS for the services:

1. **Creation of TLS Certificates**: TLS certificates were generated for each service using the following OpenSSL command:
   ```
   openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem
   -keyout key.pem -days 365
   ```

2. **Pass Certificates to Each Microservice**: the certificates were added to the Docker Compose configuration under the `secrets` field. This allowed each service to access the certificate and key needed for encrypted communication.

3. **Make Flask Services Access Certificates and Keys**: in the Dockerfile of each service the command to run the Flask application was updated like this:
   ```
   CMD ["flask", "run", "-host=0.0.0.0", "-port=5001",
   "-cert=/app/gateway_cert.pem", "-key=/app/gateway_key.pem"]
   ```

4. **Disable certificate Verification**: the requests to HTTPS have been changed adding `verify=False` field to avoid issues with self-signed certificates during testing.

6.2 Data Sanitization

To ensure the integrity and security of the data being processed, data sanitization was implemented for user inputs, covering both strings and numeric values (integers and floats).

6.2.1 String Sanitization

For string inputs, sanitization was applied to restrict the input to alphanumeric characters, spaces, and some special characters. The following function was used to sanitize string data:

```python
def sanitize_input(input_string):
    if not input_string:
        return input_string
    return re.sub(r"[^\w\s\-]", "", input_string)
```

This sanitization process was applied across various services, including Authentication, Auction Market, Gacha Roll, Gacha System, Payment, and Profile Setting.
Additionally, another version of sanitization was implemented only in the Gacha System and Profile Setting services another:

```python
def sanitize_input_gacha(input_value):
    if isinstance(input_value, str):
        return re.sub(r"[^\w\s\-.]", "", input_value)
    elif isinstance(input_value, list):
        return [sanitize_input_gacha(item) for item in input_value if isinstance(item, str)]
    elif isinstance(input_value, (int, float)):
        return str(input_value)
    else:
        return ""
```

The Authentication service also includes the following email validation function:

```python
def validate_email(email):
    email_regex = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
    return re.match(email_regex, email)
```

6.2.2 Numeric Values Sanitization (Integers and Floats)

For numeric inputs, both integers and floats were sanitized through type validation. In the Auction Market service, for instance, validation was performed to ensure that the `auction_id` provided is either an integer or a float. When the input is received via JSON, the validation is as follows:

```python
if not auction_id:
    return jsonify({"error": "Invalid input: auction_id is required"}), 400
if not isinstance(auction_id, (int, float)):
    return jsonify({"error": "auction id must be int or float"}), 400
```

This ensures that the input is of the correct type, preventing invalid data from being processed.

In cases where the `auction_id` was received through a form submission, it was explicitly cast to an integer, with error handling in place to catch invalid values:

```
    try: # Try to convert auction_id to an integer
        auction_id = int(auction_id)
    except (TypeError, ValueError):
        return jsonify({"error": "auction_id must be an integer"}), 400
```

6.3 Protection of Data at Rest

The only sensitive data stored within the system pertains to user accounts, specifically the credentials required for authentication. To protect this data, a hashed version of the password is stored in the Users DB. The system employs a two-step process for securely storing passwords:

1. **Salting the password**: a unique salt is generated for each user. This salt is combined with the user's password to ensure that identical passwords have different hash values, preventing dictionary and rainbow table attacks.

2. **Hashing the salted password**: the combined salted password is then hashed using a secure hashing algorithm. The result is a hash of the password and salt, which is stored in the database.

Therefore, in the Users DB the credentials are stored in the following way:

- **Username**
- **Hash(Hash(password) + salt)**
- **Salt**

This approach ensures that even if the database is compromised, the actual passwords remain protected.

CHAPTER 7

Access Security - Authorization and Authentication

The system follows a distributed authentication and authorization approach, where each service independently validates the access token received. This allows each service to perform token validation without the need for a centralized validation service, enabling efficient and decentralized token verification.

7.1 Key Steps for Token Validation

1. **Client Login**: the user logs in, and the Authentication service provides three tokens:
    - **ID Token** for authentication (never sent in this application).
    - **Access Token** short-lived token for authorization (sent in every request after login).
    - **Refresh Token** long-lived token used to refresh the access token.
2. **Token Validation**: when the client makes requests to any service, the service verifies the validity of the access token. Each service stores and uses the public key required to validate the access token, while the private key used to sign the tokens is stored only in the Authentication service.
3. **Token Expiration**: when the access token expires, the client can send the refresh token to the Authentication service to request a new access token (using the route `/newToken`).

4. **Logout**: when the user logs out, the Authentication service revokes the refresh token, making it unusable for future requests.

7.2 Key Management

- The **Authentication service** stores both the private key and the public key:
  - The private key is used to sign ID token, access token and refresh tokens.
  - The public key is made available to all services that need to validate the tokens.
- **Other services** only store the public key, which is used to verify the validity of the access token included in requests from the client.

7.3 Tokens Management

Token Type	**Lifetime**	**Purpose**	**Storage**	**Revocation**
ID Token	Long-lived	Used to cache profile information and provide it to a client application	Stored client-side	Valid until expiration
Access Token	Short-lived	Grants access to services; must be included in each request	Stored client-side	Valid until expiration
Refresh Token	Long-lived	Used to obtain a new access token when the access token expires	Stored client-side and server-side	Valid until expiration or Revoked upon logout

In the following sections you can find a more detailed description of the structure of the access and refresh token.

7.4 Access Token Format

The Access Token is a JWT (JSON Web Token) and is structured as follows:

- **Header**:

```
        {
            "alg": "RS256",
            "typ": "JWT"
        }
```

  The header specifies the algorithm (RS256) used to sign the token and the type of token (JWT).

- **Payload**:

```
        {
          "iss": "https://auth_service:5002",
          "sub": "user.username",
          "aud": ["profile_setting", "gachasystem", "
             payment_service", "gacha_roll", "auction_service",
             "auth_service"],
          "iat": "datetime.datetime.now(datetime.timezone.utc)",
          "exp": "datetime.datetime.now(datetime.timezone.utc) +
              datetime.timedelta(minutes=5)",
          "scope": "scope",
          "jti": "jti"
        }
```

7.5 Refresh Token Format

The Refresh Token is also a JWT, but it has a longer expiration time compared to the access token. The refresh token format is as follows:

- **Header**: same as access token.
- **Payload**:

```
        {
          "iss": "https://auth_service:5002",
          "sub": "user.username",
          "aud": "auth_service",
          "iat": "datetime.datetime.now(datetime.timezone.utc)",
          "exp": "datetime.datetime.now(datetime.timezone.utc) +
              datetime.timedelta(hours=1)",  # Expiration (1
             hour)
          "scope": "scope",
          "jti": "refresh_jti"
        }
```

CHAPTER 8

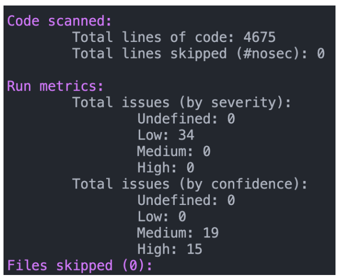
Security Analysis

A multi-faceted approach to Security Analysis was adopted, leveraging both static analysis tools and dependency management solutions to identify and mitigate vulnerabilities in the codebase and its dependencies. Below is an overview of the tools used and their roles in the security analysis

8.1 Bandit

Bandit was used to perform static analysis on the Python code to find common security issues.

- **Command used** (inside project directory): `bandit -r .`
- **Results**: in Figure 8.1 you can see the final results of Bandit. As you can see, it shows 19 medium-confidence issues and 15 high-confidence issues. However, they only pertain to the use of the `random` function and `mock` objects in unit tests.



```
Code scanned:
        Total lines of code: 4675
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 34
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 19
                High: 15
Files skipped (0):
```

Figure 8.1: *Bandit's final table*

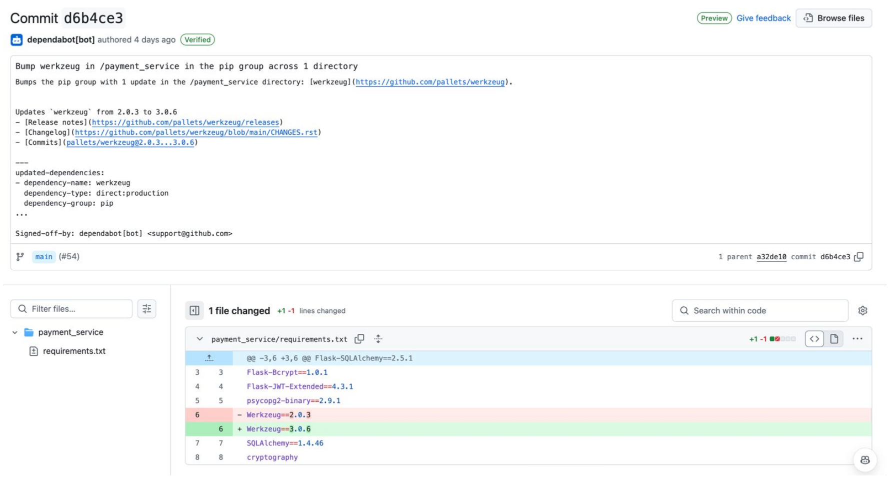
8.2 pip-audit

pip-audit was used to scan all the installed Python packages for vulnerabilities.

- **Command used** (inside project directory): `pip-audit`
- **Results**: same as Dependabot (see next session).

8.3 Dependabot

Dependabot, a tool integrated in GitHub, automatically monitored dependencies for security issues and recommended updates. Pull requests created by Dependabot were reviewed and merged. In Figure 8.2 you can see an example of some problems identified, while Figure 8.3 shows that the repository has no vulnerability alerts anymore.

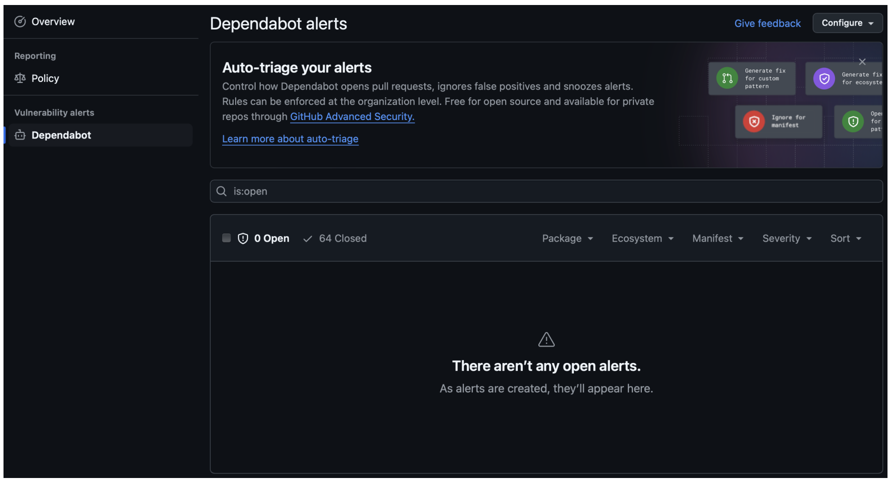


```
Bump werkzeug in /payment_service in the pip group across 1 directory
Bumps the pip group with 1 update in the /payment_service directory: [werkzeug](https://github.com/pallets/werkzeug).


Updates `werkzeug` from 2.0.3 to 3.0.6
- [Release notes](https://github.com/pallets/werkzeug/releases)
- [Changelog](https://github.com/pallets/werkzeug/blob/main/CHANGES.rst)
- [Commits](pallets/werkzeug@2.0.3...3.0.6)

---
updated-dependencies:
- dependency-name: werkzeug
  dependency-type: direct:production
  dependency-group: pip
...

Signed-off-by: dependabot[bot] <support@github.com>
```

main (#54)
1 parent a32de10 commit d6b4ce3
Filter files...
payment_service
requirements.txt
1 file changed +1 -1 lines changed
Search within code
payment_service/requirements.txt
+1 -1

```diff
@@ -3,6 +3,6 @@ Flask-SQLAlchemy==2.5.1
 Flask-Bcrypt==1.0.1
 Flask-JWT-Extended==4.3.1
 psycopg2-binary==2.9.1
- Werkzeug==2.0.3
+ Werkzeug==3.0.6
 SQLAlchemy==1.4.46
 cryptography
```

Figure 8.2: *Dependabot - vulnerability alerts*



Control how Dependabot opens pull requests, ignores false positives and snoozes alerts. Rules can be enforced at the organization level. Free for open source and available for private repos through GitHub Advanced Security.

Learn more about auto-triage
Generate fix for custom pattern
Generate fix for ecosyste
Ignore for manifest
is:open
0 Open
64 Closed
Package
Ecosystem
Manifest
Severity
Sort

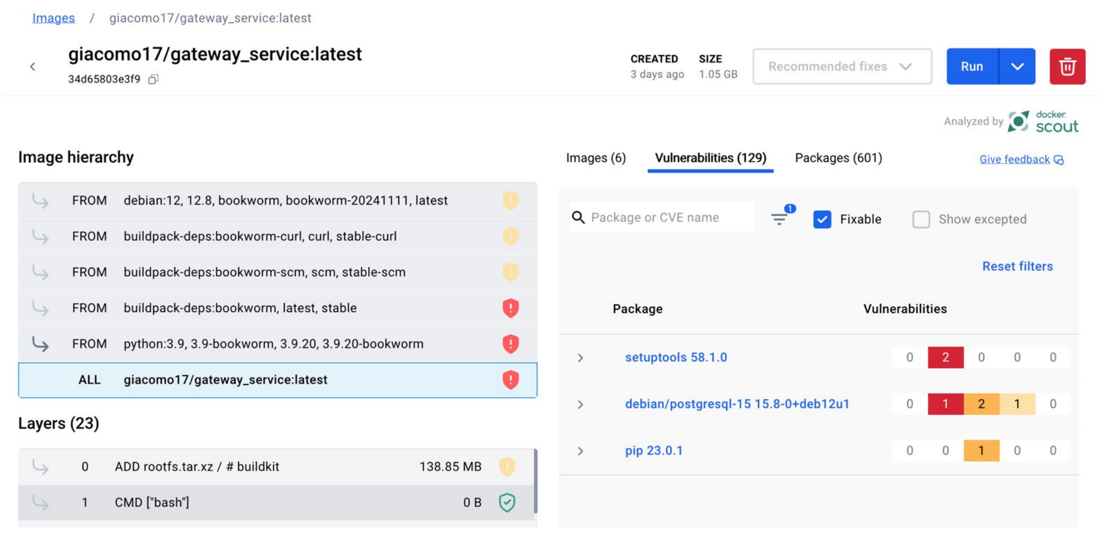
There aren't any open alerts.
As alerts are created, they'll appear here.

Figure 8.3: *Dependabot - No vulnerability alerts*

8.4 Docker Scout

Docker Scout analyzed vulnerabilities in the Docker images developed for the project.

- **Command used**: Docker Scout is integrated into Docker Desktop.
- **Results**: no image analyzed by Docker Scout has now either high or critical vulnerabilities. As an example, Figure 8.4 shows the vulnerabilities highlighted from Docker Scout for the Gateway image and Figure 8.5 demonstrates that those vulnerabilities have been solved.
  Of course, all the images have been analyzed and all the vulnerabilities have been fixed.



Images / giacomo17/gateway_service:latest

giacomo17/gateway_service:latest
34d65803e3f9

CREATED 3 days ago
SIZE 1.05 GB

Analyzed by docker scout

Image hierarchy

FROM	debian:12, 12.8, bookworm, bookworm-20241111, latest
FROM	buildpack-deps:bookworm-curl, curl, stable-curl
FROM	buildpack-deps:bookworm-scm, scm, stable-scm
FROM	buildpack-deps:bookworm, latest, stable
FROM	python:3.9, 3.9-bookworm, 3.9.20, 3.9.20-bookworm
ALL	giacomo17/gateway_service:latest

Layers (23)

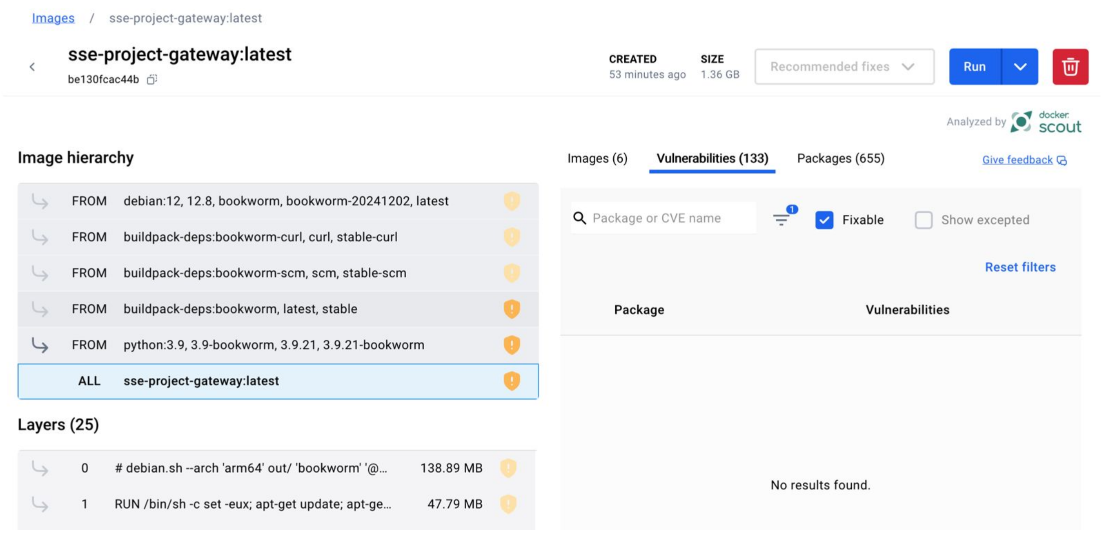
0	ADD rootfs.tar.xz / # buildkit	138.85 MB
1	CMD ["bash"]	0 B

Images (6) Vulnerabilities (129) Packages (601)

Package or CVE name ✓ Fixable ☐ Show excepted Reset filters

Package	Vulnerabilities				
setuptools 58.1.0	0	2	0	0	0
debian/postgresql-15 15.8-0+deb12u1	0	1	2	1	0
pip 23.0.1	0	0	1	0	0

Figure 8.4: *Docker Scout - Gateway Image vulnerabilities*



Images / sse-project-gateway:latest

sse-project-gateway:latest
be130fcac44b

CREATED 53 minutes ago
SIZE 1.36 GB

Analyzed by docker scout

Image hierarchy

FROM	debian:12, 12.8, bookworm, bookworm-20241202, latest
FROM	buildpack-deps:bookworm-curl, curl, stable-curl
FROM	buildpack-deps:bookworm-scm, scm, stable-scm
FROM	buildpack-deps:bookworm, latest, stable
FROM	python:3.9, 3.9-bookworm, 3.9.21, 3.9.21-bookworm
ALL	sse-project-gateway:latest

Layers (25)

0	# debian.sh --arch 'arm64' out/ 'bookworm' '@...	138.89 MB
1	RUN /bin/sh -c set -eux; apt-get update; apt-ge...	47.79 MB

Images (6) Vulnerabilities (133) Packages (655)

Package or CVE name ✓ Fixable ☐ Show excepted Reset filters

Package Vulnerabilities

No results found.

Figure 8.5: *Docker Scout - Gateway Image without vulnerabilities*