Advanced Software Engineering Project

Team: Lady Gatcha

Emanuele Buonaccorsi
Francesco Camaccioli
Matteo Giannini
Leonardo Manneschi

Table of Contents

Gachas Description

Given the name of the group, the gachas we chose to include in the project are Lady Gaga vinyl records. We divided these gachas in four level of rarity given by the value of the discs on Discogs. Higher value discs are assigned a higher rarity.

Common Gachas

ARTPOP – 2019 Reissue

Chromatica – Picture Disc

The Fame – 2008 Original Pressing

Chromatica – Silver Vinyl

Joanne – 2016 Original Pressing

Rare Gachas

Bad Romance – 7” Single

Telephone – 7” Single

Alejandro – 7” Single

The Fame Monster – 2024 Picture Disc

Epic Gachas

Joanne – Fluorescent Pink Vinyl

Chromatica – 2021 RSD Exclusive Yellow Vinyl

Born This Way – Red Vinyl

Legendary Gachas

Born This Way – Box Set, Numbered, 9 x 12” Picture Discs

The Fame + The Fame Monster – Box Set:

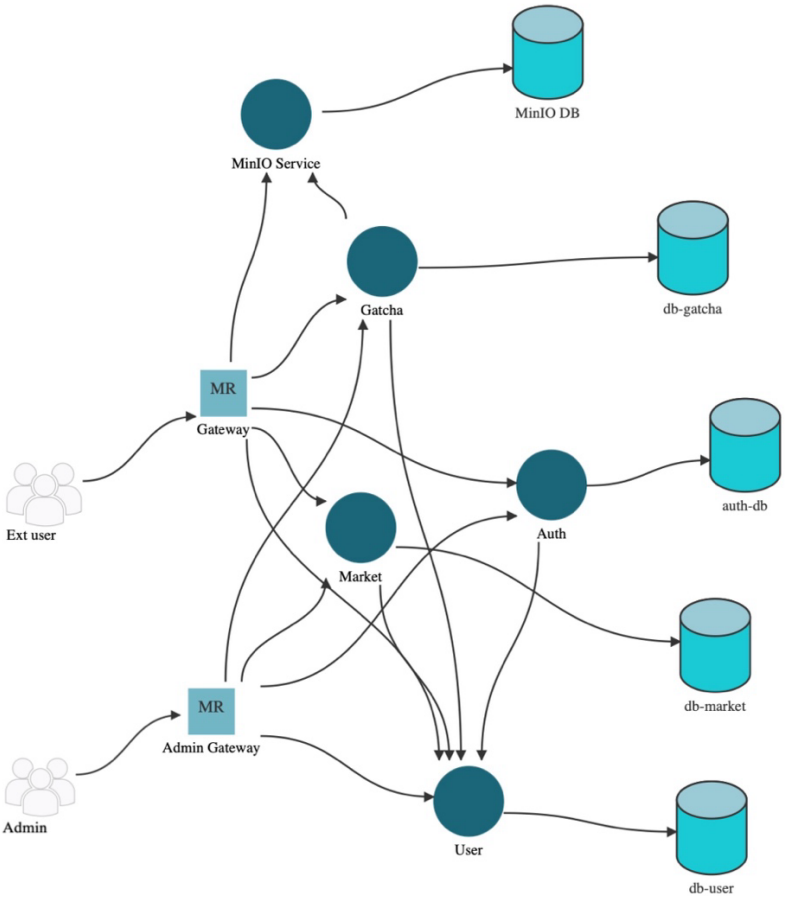
- The Fame Monster: Silver Vinyl
- The Fame: Coke Bottle Clear Vinyl

Architecture

This section describes the architecture of our application and includes some details about the microservices involved and the connections between them.

Architecture Scheme

The scheme of the architecture was realized with µFreshener and provides an overall view of the architecture.



Description of the entities

Microservices

All the microservices were developed using Python

- **Gatcha:** Manages a “gacha” system for a game, allowing administrators to manage gacha items and players to view all the items or to perform a “roll” to receive a random gacha.
- **Market:** Allows the creation and management of auctions, letting users place bids and query active auctions. It integrates with external services for user balances and item transfers, using scheduled tasks to handle auction finalization.
- **Auth:** Handles user authentication and authorization, including registration, login, token generation, and validation. It provides role-based access control, supports token revocation, and integrates with external services for user management.

- **User:** Manages user accounts, balances, and gacha collections, integrating authentication, database operations, and inter-service communication. It supports user initialization, balance updates, transaction tracking and collection management.
- **MinIO (Additional feature):** Microservice used to store and retrieve the images associated to the gachas with an accessible GUI.

Gateways

- **Gateway:** This Gateway forwards requests coming from external users to the specific microservices according to a predefined whitelist, ensuring only approved endpoints are accessible.
- **Admin Gateway:** This Gateway is used to forward the admin requests to the admin-only accessible endpoint. It ensures secure routing by enforcing role-based access control (@role_required('adminUser')) to sensitive routes.

Databases

All the database in the architecture except for the MinIO DB have been implemented with MongoDB and each database is relative to each microservice.

Microservices Connections

- **Gatcha -> User:** The interaction between the gatcha microservice and the user microservice happens when the */roll* endpoint is invoked by the user. The roll endpoint invokes two endpoints of the user microservice:
    i. */decrease_balance:* When the user rolls a gacha, a fixed amount of GagaBucks is subtracted from the user's current balance. This allows also to check wether the user possesses the sufficient amount of GagaBucks to perform a roll.
    ii. */add_gatcha:* If the user has the sufficient amount of GagaBucks, after a roll it will receive a gacha from the gatcha microservice. This gacha is put into the user collection by this endpoint.
- **Auth -> User:** The interaction between the auth microservice and the auth microservice happens within two auth endpoints:
    i. */register:* this endpoint calls the user microservice endpoint */init-user* to insert a new user in the user DB by providing the UserID of the newly registered user.
    ii. */delete_user:* this endpoint call the user microservice endpoint */delete_user* to remove the corresponding user from the user DB. This operation causes the deletion of the corresponding user gacha collection.
- **Market -> User:** The market microservice interacts with the user microservice in multiple endpoints:
    i. */add-auction* interacts with the user endpoint */remove-gatcha* to remove the gacha that the user listed for auction from the user gacha collection.
    ii. */refund* is used when another user bids a higher amount of GagaBucks for the auction with respect to the previous auction winner. This endpoint interacts with the user endpoint */increse_balance* to refund the previous winner with the previously bade amount.

iii. */bid* is the market endpoint used by a user to bid a certain amount of GagaBucks. This endpoint interacts with the user endpoint */decrease_balance* to subtract the bade amount from the user balance

iv. */finalize_auction* is the market endpoint used to finalize and auction and to reward the winner with the the bade gacha. To do so, this endpoint calls the user endpoint */add_user.* This endpoint also uses the user */increase_balance* endpoint to add the final bid amount to the bid winner balance.

- **User -> Gatcha:** The user endpoint */collection/<gatcha_ID>* interacts with the gatcha mictoservice endpoint */gatchas/<gatcha_ID>* to obtain detailed information about the specified gacha.
- **Gatcha -> MinIO:** MinIO is a bucket where we can put files. When an admin uploads a new gatcha it sends also a *.png* image along the request. This image is uploaded into the bucket stored into the MinIO DB.

User Stories

The following section describes how user stories are implemented and in particular it describes the endpoint used to access that specific user story and how to access them using the user accessible endpoints.

External Users User Stories

Create my game account/profile so that I can participate in the game

Endpoint: /auth/register
Microservices Involved: Gateway, Auth Service, User Service

Delete my game account/profile so that I can stop participating in the game

Endpoint: /auth/delete_user
Microservices Involved: Gateway, Auth Service, User Service

Modify my account/profile so that I can personalize my account/profile

Endpoint: /auth/editinfo
Microservices Involved: Gateway, Auth Service, User Service

Login and logout from the system so that I can access and leave the game

Endpoints: /auth/login, /auth/tokens/revoke
Microservices Involved: Gateway, Auth Service

Be safe about my account/profile data so that nobody can enter in my account and steal/modify my info

Endpoints: All Authentication Endpoints (Security Measures)
Microservices Involved: Gateway, Auth Service

See my gacha collection so that I know how many gacha I need to complete the collection

Endpoint: /user/collection
Microservices Involved: Gateway, User Service

See the info of a gacha of my collection so that I can see all of info of one of my gacha

Endpoint: /gatcha/collection/{gatcha_ID}
Microservices Involved: Gateway, User Service

See the system gacha collection so that I know what I miss of my collection

Endpoint: /gatcha/gatchas
Microservices Involved: Gateway, Gatcha Service

See the info of a system gacha so that I can see the info of a gacha I miss

Endpoint: /gatcha/gatchas/{gatcha_ID}
Microservices Involved: Gateway, Gatcha Service

Use in-game currency to roll a gacha so that I can increase my collection

Endpoint: /gatcha/roll
Microservices Involved: Gateway, Gatcha Service, User Service

Buy in-game currency so that I can have more chances to win auctions

Endpoint: /user/increase_balance
Microservices Involved: Gateway, User Service

Be safe about in-game currency transactions so that my in-game currency is not wasted or stolen

Endpoints: All Currency Transaction Endpoints (Security Measures)
Microservices Involved: Gateway, User Service, Auth Service

See the auction market so that I can evaluate if buy/sell a gacha

Endpoint: /market/auctions
Microservices Involved: Gateway, Market Service

Set an auction for one of my gacha so that I can increase in-game currency

Endpoint: /market/add-auction
Microservices Involved: Gateway, Market Service, User Service

Bid for a gacha from the market so that I can increase my collection

Endpoint: /market/bid
Microservices Involved: Gateway, Market Service, User Service

View my transaction history so that I can track my market movement

Endpoint: /user/transactions
Microservices Involved: Gateway, User Service

Receive a gacha when I win an auction so that only I have the gacha I bid for

Process: Auction Scheduler finalizes auction when its time expires
Microservices Involved: Market Service, User Service

Receive in-game currency when someone wins my auction so that the gacha sell works as I expect

Process: Auction Scheduler finalizes auction when its time expires
Microservices Involved: Market Service, User Service

Receive my in-game currency back when I lose an auction so that my in-game currency is decreased only when I buy something

Process: Refund endpoint call as soon as a higher bid is placed
Microservices Involved: Market Service, User Service

Ensure that the auctions cannot be tampered so that my in-game currency and collection are safe

Endpoints: All Market Endpoints (Security Measures)
Microservices Involved: Gateway, Auth Service, Market Service

Admin User Stories

All the endpoints used by the admins must be accessed via the admin gateway using its address and port.

Login and logout as admin from the system so that I can access and leave the game

Endpoint(s): /auth/login, /auth/logout
Microservices Involved: Admin Gateway, Auth Service

Check all the gacha collection so that I can check all the collection

Endpoint: /gatcha/gatchas
Microservices Involved: Admin Gateway, Gatcha Service

Modify the gacha collection so that I can add/remove gachas

Endpoints: /gatcha/gatchas (POST for adding), /gatcha/gatchas/{gatcha_ID} (PUT for updating, DELETE for removing)
Microservices Involved: Admin Gateway, Gatcha Service

Check a specific gacha so that I can check the status of a gacha

Endpoint: /gatcha/gatchas/{gatcha_ID}
Microservices Involved: Admin Gateway, Gatcha Service

Modify a specific gacha information so that I can modify the status of a gacha

Endpoint: /gatcha/gatchas/{gatcha_ID}
Microservices Involved: Admin Gateway, Gatcha Service

Market Rules

Auction Mechanics

Auction Creation

- **Listing an Item:** Players can create an auction to sell a Gacha item from their collection selecting a starting price for the bade item.
- **Auction Duration:** Each auction runs for a fixed duration of 10 minutes from the time of creation.
- **Item Transfer:** Upon auction creation, the Gacha item is removed from the seller's collection and is gave back to the user when the auction concludes.

Bidding Process

- **Placing Bids:** Players can place bids on active auctions as long as their bid is higher than the current highest bid.
- **Immediate Balance Decrease:** When a bid is placed, the bid amount is immediately subtracted from the bidder's balance.
- **Outbidding:** If another player places a higher bid, the previous highest bidder's amount is automatically refunded.

Auction Finalization

- **Automatic Closing:** At the end of the auction duration, the system's BackgroundScheduler finalizes the auction.
- **Winner Determination:** The highest bidder at the time of auction closing wins the item.
- **Item Delivery:** The Gacha item is transferred to the winner's collection.
- **Seller Payment:** The final bid amount is credited to the seller's balance.
- **No Bids Received:** If no bids are placed, the item is returned to the seller's collection without any fees.

Design Decisions

Immediate Currency Handling

- **Rationale:** Deducting the bid amount immediately ensures that bidders have sufficient funds to bid in the current auction and prevents overbidding.
- **Automatic Refunds:** When a bidder is outbid, the bid amount is promptly refunded to their balance, allowing them to use the currency elsewhere.

Fixed Auction Duration

- **Consistency:** Auctions have a fixed duration to provide a consistent bidding environment.
- **No Extensions** The auction duration remains fixed; last-second bids do not extend the auction time. Bidding at the last second is risky as you may not have time to react if you're outbid simultaneously.
- **Strategic Bidding:** Players are encouraged to bid their maximum value early or monitor auctions closely.

Bid Restrictions

- **No Self-Outbidding:** Prevents players from artificially inflating auction prices or reserving items.
- **Encourages Competition:** Players must wait to be outbid before placing a higher bid, fostering competitive bidding.

Auctions ending without any bids

- **Item Return:** The Gacha item is returned to the seller's collection automatically.
- **No Penalties:** There are no fees or penalties for auctions that end without bids.
- **Option to Relist:** Sellers may choose to relist the item for auction if desired.

Security and Fair Play

- **Authentication Required:** All market actions require players to be authenticated to ensure secure transactions.
- **Role Enforcement:** Access to certain market functions is restricted based on user roles (e.g., admin privileges).
- **Data Validation:** Inputs are validated to prevent invalid data and ensure the integrity of the auction process.
- **Logging and Monitoring:** Activities are logged for transparency and to aid in resolving disputes or issues.

Integration with Other Services

User Service

- **Balance Management:** Handles the deduction and refund of in-game currency during bidding.
- **Item Transfers:** Manages the addition and removal of Gacha items from player collections.

Authentication Service

- **User Verification:** Utilizes JWT tokens to authenticate players and retrieve user IDs.
- **Access Control:** Ensures players have the necessary permissions to perform market actions.

Testing

Security – Data

Input Sanitization

Sanitized Inputs and Their Usage

Service Names and Paths

- **Description:** Inputs related to the routing of requests through the API Gateway, specifically the service names and endpoints extracted from URLs.
- **Examples of Inputs:**
    - **Service Names:** “auth”, “user”, “gatcha”, “market”.
    - **Endpoints:** “login”, “register”`, “getAll”, “roll”, “add-auction”.
- **Usage:** These inputs are used by the Gateway microservice to forward requests to the correct microservices. Sanitization ensures that only valid service names and paths are accepted, preventing unauthorized access to internal services or execution of unintended routes.

User Credentials and Personal Data

- **Description:** Data provided by users during registration, authentication, and profile updates, such as usernames, passwords, and email addresses.
- **Examples of Inputs:**
    - **Usernames:** “maria_di_123”, “giovanni.rossi”, “user@example.com”.
    - **Passwords:** “P@ssw0rd!”, “Secure*Pass123”.
    - **Email Addresses:** “user@example.com”.
- **Usage:** These inputs are used by the Auth microservice for creating new accounts, authenticating users, and managing user profiles. Sanitization prevents injection attacks and ensures data integrity by validating formats and encoding or escaping special characters.

Query Parameters and Request Data

- **Description:** Parameters passed via URLs and data included in HTTP requests.
- **Usage:** These inputs are used by various microservices to process user requests, such as searching for items, submitting bids, or updating user balances. Sanitization ensures that inputs conform to expected formats, preventing injection of malicious content or execution of unintended commands.

Methods of Sanitization Implemented

Input Validation Using Regular Expressions

- **Purpose:** Ensures that inputs match the expected patterns, containing only allowed characters and conforming to required formats.
- **Implementation:** Before processing, inputs like service names, usernames, and paths are checked against regular expressions to validate their structure.

Endpoint Whitelisting

- **Purpose:** Restricts access to only predefined and authorized endpoints. Only some endpoints are made available to external users, whereas the others are accessible only to other microservices.
- **Implementation:** The Gateway microservice maintains a whitelist of allowed endpoints, specifying permissible service names, HTTP methods, and paths. Any

request not matching the whitelist is rejected, preventing unauthorized access or potential exploitation of unsecured routes.

Sanitization of User Inputs

- **Purpose:** Prevents malicious data from being processed or stored, protecting against injection attacks and data corruption.
- **Implementation:** User-provided data is sanitized by removing or escaping special characters and validating data types. For instance, passwords are checked for strength and complexity, and usernames are verified to contain only allowed characters.

Security – Authorization and Authentication

Authentication Scenario Selection

In our microservices project we opted for a centralized authentication scenario. This approach centralizes authentication by delegating token validation to a dedicated Authentication Service, in our case the Auth microservice. All other microservices rely on this service for verifying access tokens and retrieving authentication information.

Basic Steps to Validate a Token

1. Client Authentication Request: The client sends authentication credentials (e.g., username and password) to the Authentication Service.
2. Token Issuance: The Authentication Service verifies the credentials. Upon successful authentication, it generates an Access Token signed with a private key. The token contains user identity and permissions encoded as claims.
3. Client Accesses Microservices: The client includes the Access Token in the Authorization header of requests to other microservices.
4. Microservice Receives Request: The microservice receives the request with the Access Token. Instead of validating the token itself, it forwards the token to the Authentication Service for validation.
5. Token Validation by Authentication Service: The Authentication Service verifies the token's signature and expiration. Checks the token claims to ensure the user has the necessary permissions.
6. Response to Microservice: If the token is valid, the Authentication Service returns the user information or a success status. If invalid, it returns an authentication error.
7. Microservice Processes Request: Upon successful validation, the microservice processes the request using the user information. Sends the appropriate response back to the client.

Key Management Summary

The key used for the generation of the JWT Token is managed with the Docker Compose Secret Manager.

JWT Token Payload

The Access Token issued by the Authentication Service is a JSON Web Token (JWT) containing the following fields:

```
"sub": user["userID"], # JWT Subject: the user's ID
"role": user["role"], # User role: Can be User or Admin
"iat": datetime.now(), # Timestamp of JWT Issuance
"exp": datetime.now() + timedelta(minutes=TOKEN_EXPIRATION_MINUTES),
      # JWT Expiration Timestamp
```

```
"iss": "https://auth.ladygatcha.com", # JWT Issuer example
"jti": str(uuid.uuid4()) # JWT ID: univocal identifier for the token
```

Token Revocation

When a user logs out from the system, the corresponding token is revoked by putting it into a Redis DB containing the list of all revoked tokens. Every time a user wants to perform an authenticated operation, this list is checked for the presence of the token and if it's present the operation is aborted.

Security – Analyses

Additional Features