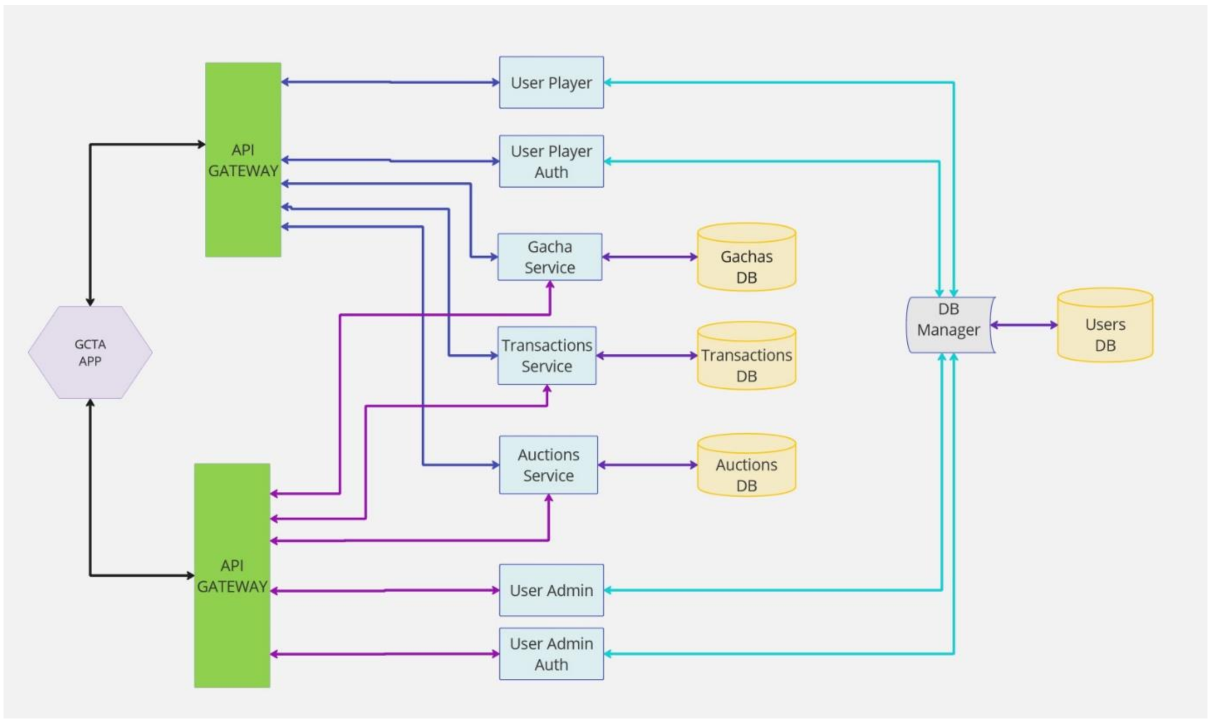
Contents

Introduction

The GCTA project is a backend system for a gacha game. It includes various microservices for user management, authentication, transactions, auctions, and gacha item management. The project is built using Flask and Docker, and it follows a microservices architecture.

Services Architecture

The diagram below shows an overview of the service architecture and how they communicate with each other. It also shows how they access data from the database.



API Gateways

The API gateways handle routing requests to the appropriate backend services. The are core to inter-service communicate. The **gateways_user** routes requests that are related a user who is a player functionalities while the **gateway_admin** routes requests related to admin functionalities. The instructions for running the project are in the **readme** file.

User Services

Listed below are services that are responsible for user management.

- **user_admin**: Admin service for managing administrative. This should only be accessible if the user logged in is an admin.
- **user_admin_auth**: This service handles the admin authentication functionalities like register, login, logout and other authentication services.
- **user_player**:This services manages the users who is only a player functionalities.
- **user_player_auth**: Handles player authentication.

Functional Services

The Microservices listed below are responsible for core game functionalities:

- **transaction**: This service handles all transactions including real money top up to the player account, transactions from bids and auctions for the players.
- **auction**: This service handles the all the auction activities and bids by the players.
- **gacha**: This service handles the gacha related functionalities such as rolling a gacha and viewing players gachas
- **db-manager:** The db-manager manages access to the database for the players and the admins

Communication

API Gateways:

The **gateway_user** communicates with:

- user_player, user_player_auth, transaction, auction, gacha

The **gateway_admin** communicates with:

- user_admin, user_admin_auth, transaction, auction, gacha

The **services communication** is as below:

- **user_admin** and **user_admin_auth** communicate with db-manager to access the admin related users database.
- **user_player** and **user_player_auth** communicate with db-manager to access the player related users database.
- **transaction**, **auction**, and **gacha** services interact with their respective databases.

Network

All services are connected through a shared **bridge network** to ensure seamless communication.

Configuration

Routing rules for the API gateways are defined in **nginx.conf** files. These configurations ensure requests are directed to the appropriate microservices.

API Documentation

The API documentation is available in the **openapi.yaml** and its graphical version in the **openapi.pdf** file. It includes detailed information about all available endpoints, request parameters, and responses. These files are included in the docs folder in the repository.

Testing

Unit Testing

During the whole development cycle of the project, tests have been done using postman to test individual APIs as well as the flow of functionality. The tests included both positive and negative tests.

In addition to running tests on postman, the tests have also been added to the continuous integration pipeline on github. Using **github actions** the tests run every time there is a github push to the main branch. The directory of the test is included in the github repository.

Security

API Segmentation

Admin and user APIs are segmented using separate Docker networks. The admin endpoints are accessed using the port 8081 while those of the user are accessed using the port 8080.

Authentication & Authorization

For authentication, JWT is used to manage user sessions.