ASE Project Report - GROUP 2

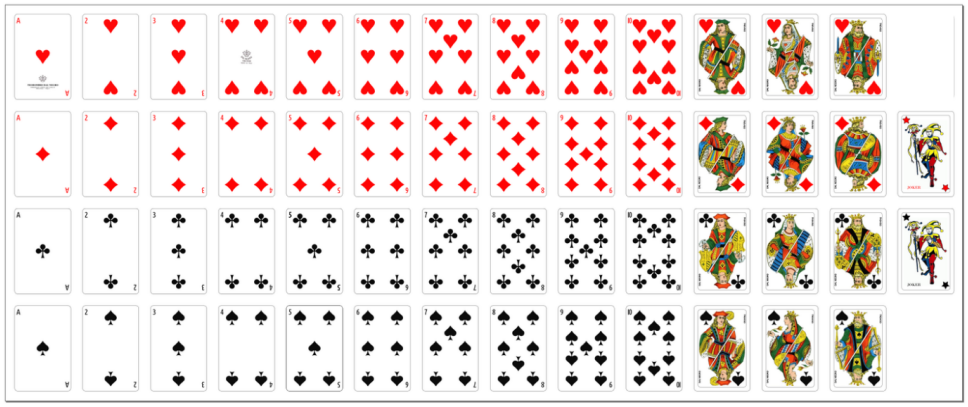
Advanced Software Engineering

First Name	Last Name	Student ID	E-Mail
Filippo	Morelli	608924	f.morelli38@studenti.unipi.it
Federico	Fornaciari	619643	f.fornaciari@studenti.unipi.it
Ashley	Spagnoli	655843	a.spagnoli9@studenti.unipi.it
Marco	Pernisco	683674	m.pernisco@studenti.unipi.it

Contents

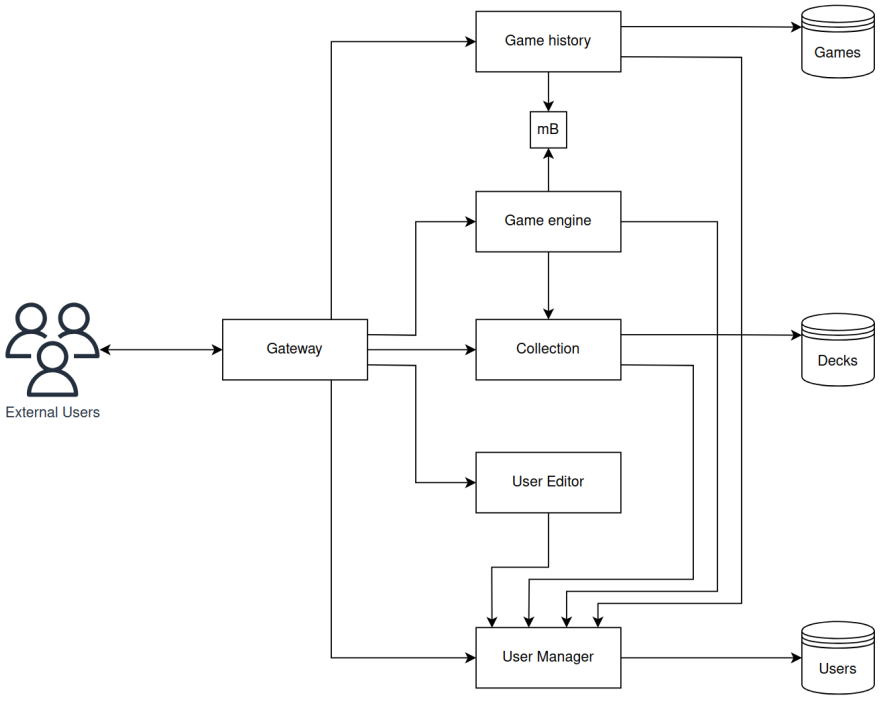
1 Cards Overview

As cards we used the classical 52 + Joker deck.
As anyone could expect, the two features assigned to those cards are the number and the suit, while the Joker is a special card with no features, that beats any other card.



2 System Architecture

Below is the high-level architectural drawing of the system, illustrating the interactions between microservices.



All microservices are containerized using Docker and are written in Python. Some of them use the Flask framework and some FastAPI.

Microservice	Framework	Description
Game history	Flask	Stores and retrieves past game data
Game engine	Flask	Handles the game runtime
Collection	Flask	Handles cards and users decks
User Editor	FastAPI	Manages user profiles and settings
User Manager	FastAPI	Manages user authentication and authorization
Gateway	FastAPI	Routes requests to appropriate microservices

2.1 Design choices

- **Centralized Authorization:** (draft)Every microservice connects to the User Manager to verify the token header (so that i can get dynamic username). User manager logs out users when they change username, this means that if we used decentralized auth, a logged-in user could see its old username using an old token, also, we don't have a way to force the log-out of an user with decentralized
- **User Information Update:** We've separated the

3 User Stories

4 Rules of the Game

4.1 Card Game Rules

4.1.1 Deck Building Constraints

Each player constructs a **personal deck** of exactly **9 cards**. The deck must adhere to the following constraints:

- **Composition:** The deck must contain exactly **1 Joker** and **8 Suited Cards**.
- **Suit Distribution:** You must include exactly **2 cards** from each suit (♡, ♢, ♣, ♠).
- **Cost Limit:** The combined point value of the two cards in any single suit **must not exceed 15**.

Card Point Costs

When calculating your deck limits, use the following costs (Note: Face cards have specific costs despite their combat strength):

Card Type	Ranks	Cost per Card	Example Pair Limit
Numbers	2 – 10	Face Value	$10 + 5 = 15$ ✓
Ace	A	7	$A + 8 = 15$ ✓
Jack	J	11	$J + 4 = 15$ ✓
Queen	Q	12	$Q + 3 = 15$ ✓
King	K	13	$K + 2 = 15$ ✓

4.1.2 Combat Hierarchy

When two cards are played, the winner is decided by the following hierarchy:

1. The Combat Triangle The core mechanics function like Rock-Paper-Scissors:

- **Numbers (2–10)** beat **Aces**.
- **Aces** beat **Face Cards (J, Q, K)**.
- **Face Cards (J, Q, K)** beat **Numbers**.

2. The Joker The **Joker** beats all other cards automatically. (If both players play a Joker, it is a tie).

3. Tie-Breakers If the combat rules above do not determine a clear winner (e.g., Number vs Number, or Face vs Face), compare the specific ranks:

1. **Higher Rank Wins:** (e.g., 9 beats 6, King beats Jack).
2. **Equal Rank → Suit Priority:** If ranks are identical (e.g., ♡9 vs ♢9), check suits:

$$♡ > ♢ > ♣ > ♠$$

3. **Mirror Match:** If players play the *exact same card* (Rank and Suit), both players win the round and gain a point.

4.1.3 Gameplay Flow

1. **Setup:** Both players shuffle their pre-built decks.
2. **Initial Draw:** Each player draws **3 cards** to form their starting hand.
3. **The Round:**
   - **Draw Phase:** At the start of every turn (including Turn 1), players draw **1 card**.
   - **Battle Phase:** Both players play one card **face down**, then reveal simultaneously.
   - **Scoring:** Determine the winner based on the Combat Hierarchy. The winner earns **1 point**.
4. **Victory:** The game ends immediately when a player reaches **5 points**.

4.1.4 Example Match Log

Alice's Deck: ♡A, ♡7, ♢A, ♢7, ♣K, ♣2, ♠K, ♠2, Joker.
Bob's Deck: ♡2, ♡3, ♢2, ♢3, ♣3, ♣4, ♠2, ♠3, Joker.

Turn	Alice	Bob	Result Reasoning	Score (A-B)
1	♡A	♣K	**Ace beats Face** (Special Rule)	1 – 0
2	♢A	♢3	**Number beats Ace** (Special Rule)	1 – 1
3	♣K	♣3	**Face beats Number** (Special Rule)	2 – 1
4	♠K	♠2	**Face beats Number** (Special Rule)	3 – 1
5	♡7	♣4	Both Numbers: $7 > 4$	3 – 2
6	♢7	♠3	Both Numbers: $7 > 3$	3 – 3
7	♣2	Joker	**Joker beats Everything**	3 – 4
8	Joker	♡2	**Joker beats Everything**	4 – 4
9	♡7	♠3	Both Numbers: $7 > 3$	**5 – 4**

11 Build and Run Instructions

This project is a multi-service card game platform designed for two players. It features authentication, deck building, a match-simulation engine, and history tracking. All services are containerized and orchestrated via Docker Compose.

11.1 Prerequisites

Before starting, ensure your environment meets the following requirements:

- **Docker & Docker Compose** (Required for orchestration).
- **Python 3.10+** (Optional, only required for local non-containerized development).

11.2 Quick Start Guide

1. **Clone the repository**

   ```
   git clone https://github.com/ashleyspagnoli/ASE_project.git
   cd ASE_project/src
   ```

2. **Build and launch services** Run the following command to build the images and start the containers:

   ```
   docker compose up --build
   ```

3. **Verify Service Status** Once the containers are running, the architecture exposes the following endpoints:

Service Name	Responsibility	Local URL
User Manager	Authentication & JWT	`https://localhost:5004`
Collection	Deck Management	`http://localhost:5003`
Game Engine	Core Logic & Matchmaking	`http://localhost:5001`
Game History	Match Logging	`http://localhost:5002`

11.3 Development & Testing

11.3.1 Environment Configuration

Each microservice is configured via environment variables. For specific configuration keys, refer to the `Dockerfile` and `requirements.txt` located in each service's directory.

11.3.2 Testing the Workflow

A Postman collection is provided for end-to-end testing. Import `game_workflow.postman_collection.json` into Postman to simulate a full lifecycle:

- User Registration and Login (Token generation).
- Deck creation and validation.
- Matchmaking and gameplay simulation.

11.3.3 Key API Endpoints

Authentication

- `/users/register`: user registration [POST]
- `/users/login`: user login [POST]
- `/users/validate-token`: internal JWT token validation [GET]

Deck Management

- `/collection/cards`: get the collection of cards [GET]
- `/collection/decks`: create a new deck [POST]

Game Engine

- `/game/connect`: connect to start playing the game [POST]
- `/game/matchmake`: manual request to start the match [POST] (TO REMOVE)
- `/game/play/{game_id}`: play a card [POST]
- `/game/state/{game_id}`: get the state of the game [GET]

Game History

- `/leaderboard`: get the whole leaderboard [GET]
- `/matches`: get the history of your played matches [GET]
- `/addmatch`: memorize a new match [POST]

11.4 Maintenance

To stop the application and remove containers/networks, run:

```
docker compose down
```