University of Pisa

Università di Pisa

Card Game
Project

First Delivery (Optional)

Alexander MITTET 708819
Estelle KEYMEULEN 712904
Katrine MIKALSEN 710649
Tara POPOV 722164

Prof. A. Brocci

Academic year 2025-2026

Contents

1 Project Architecture

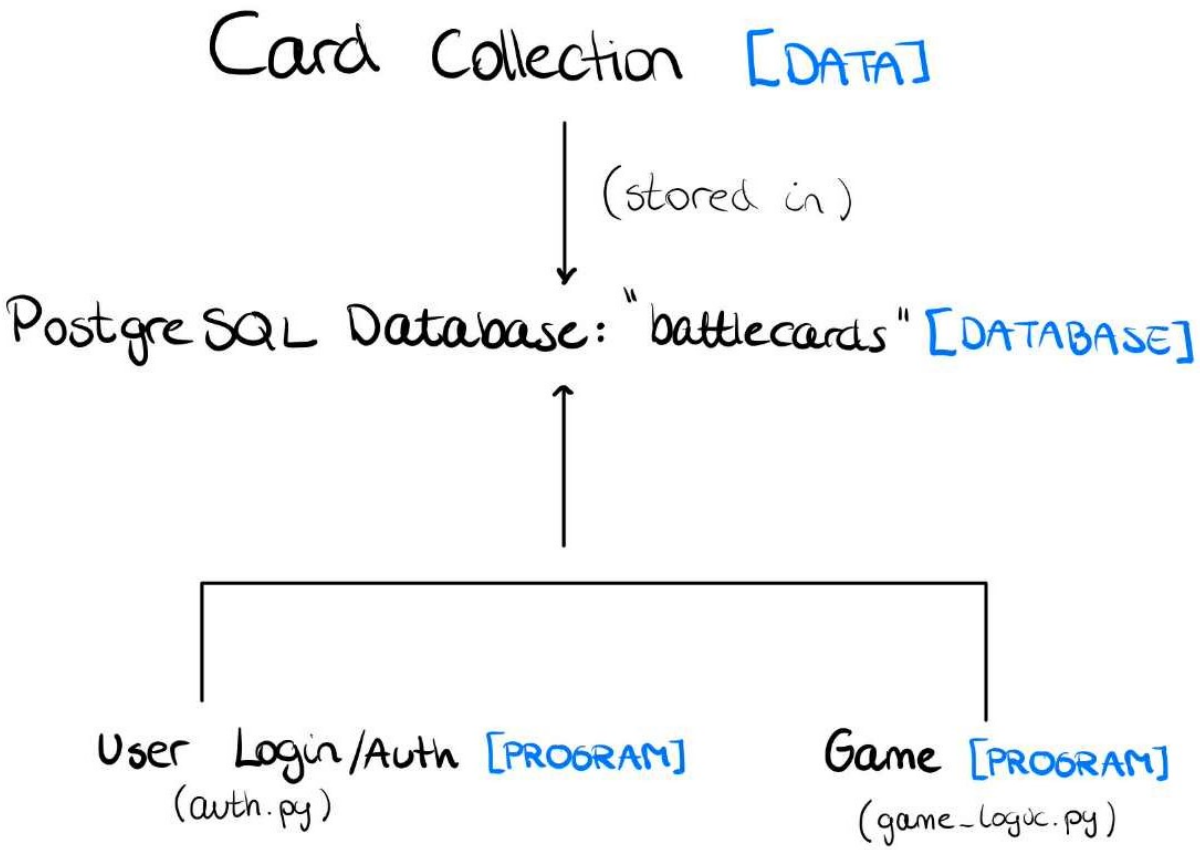

Figure 1: Project architecture

2 Rules of the game

The overall card deck has 39 cards: Rock, Paper, and Scissors suits, each numbered 1 to 13. The game starts with the player viewing all possible cards.

2.1 Overall rules

1. Each player selects 22 cards from their own overall card deck. This becomes the player's deck for the game.
2. At the start of each round, each player chooses a hand of three cards for that round.
3. Both players play one card simultaneously.
4. The winning card gives one point to the winning player. If it's a tie, no one gets a point.

Repeat step 2-4 for all the 7 rounds.

2.2 Ordering rules

1. Rock > Scissors > Paper > Rock ...
2. $1 > 13 > 12 > 11 > 10 > \cdots > 2 > 1$

This means that among the numbers 1–13, the larger number always beats the smaller one, making the relation transitive. The only exception is that 1 beats 13.

PS: it hasn't been implemented like this at this point. At this time being, 13 beats everything.

3 Example Match

Here is an example of a match between Player A and Player B:

Round	Player A's Card	Player B's Card	Winner
1	Rock 7	Scissors 5	Player A
2	Paper 12	Paper 8	Player A
3	Scissors 3	Rock 3	Player B
4	Rock 1	Rock 13	Player A
5	Paper 5	Scissors 6	Player B
6	Scissors 10	Paper 11	Player A
7	Rock 9	Rock 9	Tie

Table 1: Example match showing each round, cards played, and the winner.

In this example, Player A wins 4 rounds, Player B wins 2 rounds, and 1 round ends in a tie. Therefore, Player A wins the match.

3.1 Edge cases

Round	Player A's Card	Player B's Card	Winner
8	Rock 1	Rock 2	Player B
9	Rock 13	Rock 12	Player A
10	Rock 1	Rock 13	Player A

4 How to Run the Project

4.1 Starting and Stopping the System

All microservices (Auth, Cards, Games, Leaderboard, Database, and Nginx gateway) are containerized and managed through Docker Compose.

1. Ensure Docker and Docker Compose are installed.
2. Navigate to the project root directory.
3. Start all services:

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

4. Stop all services:

```
docker compose down
```

5. View logs for a specific service:

```
docker compose logs <service-name>
```

6. Check running services:

```
docker compose ps
```

Once running, all API endpoints are accessible through the Nginx gateway:

```
http://localhost:8080
```

4.2 Creating a Test User

1. Register a new user:

```
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"testuser",
       "email":"test@example.com",
       "password":"password123"}'
```

2. Log in to obtain an access token:

```
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"testuser",
       "password":"password123"}'
```

3. Use the token to call authenticated endpoints:

```
curl -H "Authorization: Bearer <TOKEN>" \
  http://localhost:8080/api/cards
```

4.3 Two–Player Game Setup

Running a game requires two separate terminals.

4.3.1 Player Registration (Once)

```
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","email":"alice@example.com","password":"password123"}'

curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"bob","email":"bob@example.com","password":"password123"}'
```

4.3.2 Terminal 1: Alice

1. Login as Alice and save token:

   ```
   (Bash/Linux)
   ALICE_TOKEN=$(curl -X POST http://localhost:8080/api/auth/login \
   -H "Content-Type: application/json" \
   -d '{"username":"alice","password":"password123"}' | jq -r '.access_token')

   (PowerShell/Windows)
   $ALICE_TOKEN = (curl -X POST http://localhost:8080/api/auth/login \
   -H "Content-Type: application/json" \
   -d '{"username":"alice","password":"password123"}' | ConvertFrom-Json).access_token
   ```

2. Create a game:

   ```
   GAME_ID=$(curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
   -H "Content-Type: application/json" \
   -d '{"player2_name":"bob"}' http://localhost:8080/api/games | jq -r '.game_id')

   PowerShell version:
   $GAME_ID = (curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
   -H "Content-Type: application/json" \
   -d '{"player2_name":"bob"}' http://localhost:8080/api/games | ConvertFrom-Json).game_id
   ```

4.3.3 Terminal 2: Bob

1. Log in as Bob and save token :

   ```
   (Bash/Linux)
   BOB_TOKEN=$(curl -X POST http://localhost:8080/api/auth/login \
   -H "Content-Type: application/json" \
   -d '{"username":"bob","password":"password123"}' | jq -r '.access_token')

   (PowerShell/Windows)
   $BOB_TOKEN = (curl -X POST http://localhost:8080/api/auth/login \
   -H "Content-Type: application/json" \
   -d '{"username":"bob","password":"password123"}' | ConvertFrom-Json).access_token
   ```

2. Set the game ID shared by Alice:

   ```
   $GAME_ID="<insert-game-id>"
   ```

4.4 Playing a Round

Each player performs these steps in their respective terminals.

4.4.1 Step 1: Draw Cards

```
curl -X POST -H "Authorization: Bearer $TOKEN" \
  http://localhost:8080/api/games/$GAME_ID/draw-hand
```

4.4.2 Step 2: View Hand

```
curl -H "Authorization: Bearer $TOKEN" \
  http://localhost:8080/api/games/$GAME_ID/hand
```

4.4.3 Step 3: Play a Card

```
curl -X POST -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"card_index":0}' \
  http://localhost:8080/api/games/$GAME_ID/play-card
```

4.5 Resolving the Round

Once both players have played:

```
curl -X POST -H "Authorization: Bearer $TOKEN" \
  http://localhost:8080/api/games/$GAME_ID/resolve-round
```

4.6 Checking Game State

```
curl -H "Authorization: Bearer $TOKEN" \
  http://localhost:8080/api/games/$GAME_ID
```

4.7 Viewing Leaderboards and Statistics

```
curl -H "Authorization: Bearer $TOKEN" \
  http://localhost:8080/api/leaderboard

curl -H "Authorization: Bearer $TOKEN" \
  http://localhost:8080/api/leaderboard/player/alice
```

5 Precise Instructions to Play a Match (API Requests)

The following steps describe the sequence of API requests needed to play a match in the Battle Card Game. Each request requiring authentication must include the JWT token in the `Authorization: Bearer <token>` header. This workflow assumes two players playing in separate terminals.

1. **Register users (run once per player):**

```
# Register Player 1 (Alice)
curl -X POST http://localhost:5001/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","email":"alice@example.com","password":"password123"}'

# Register Player 2 (Bob)
curl -X POST http://localhost:5001/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"bob","email":"bob@example.com","password":"password123"}'
```

2. **Login and retrieve JWT tokens:**

```
# Alice login
(Linux)
ALICE_TOKEN=$(curl -X POST http://localhost:5001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"password123"}' | jq -r '.access_token')

(PowerShell/Windows)
$ALICE_TOKEN = (curl -X POST http://localhost:8080/api/auth/login \
-H "Content-Type: application/json" \
-d '{"username":"alice","password":"password123"}' | ConvertFrom-Json).access_token

# Bob login
(Linux)
BOB_TOKEN=$(curl -X POST http://localhost:5001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"bob","password":"password123"}' | jq -r '.access_token')

(PowerShell/Windows)
$BOB_TOKEN = (curl -X POST http://localhost:8080/api/auth/login \
-H "Content-Type: application/json" \
-d '{"username":"bob","password":"password123"}' | ConvertFrom-Json).access_token
```

3. **Create a new game (Alice creates, Bob joins using GAME_ID):**

```
# Alice creates a game
(Linux)
GAME_ID=$(curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"player2_name":"bob"}' \
  http://localhost:5003/api/games | jq -r '.game_id')

(PowerShell)
$GAME_ID = (curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
-H "Content-Type: application/json" \
-d '{"player2_name":"bob"}' http://localhost:8080/api/games | ConvertFrom-Json).game_id

# Bob sets the GAME_ID
$GAME_ID="paste-game-id-from-alice"
```

4. **Draw cards to hand:**

```
# Alice draws her hand
curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/draw-hand

# Bob draws his hand
curl -X POST -H "Authorization: Bearer $BOB_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/draw-hand
```

5. **View cards in hand (optional):**

```
# Alice views hand
curl -H "Authorization: Bearer $ALICE_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/hand

# Bob views hand
curl -H "Authorization: Bearer $BOB_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/hand
```

6. **Play a card:**

```
# Alice plays a card (index 0, 1, or 2)
curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"card_index":0}' \
  http://localhost:5003/api/games/$GAME_ID/play-card

# Bob plays a card (index 0, 1, or 2)
curl -X POST -H "Authorization: Bearer $BOB_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"card_index":1}' \
  http://localhost:5003/api/games/$GAME_ID/play-card
```

7. **Resolve the round (automatic if both played, otherwise manual):**

```
curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/resolve-round

# Or from Bob's terminal
curl -X POST -H "Authorization: Bearer $BOB_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/resolve-round
```

8. **Repeat steps 4–6 for all rounds.**
9. **Check game state and scores:**

```
curl -H "Authorization: Bearer $ALICE_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID

curl -H "Authorization: Bearer $BOB_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID
```

10. **Optional: Use tie-breaker rounds if the game ends in a tie:**

```
curl -X POST -H "Authorization: Bearer $ALICE_TOKEN" \
  http://localhost:5003/api/games/$GAME_ID/tie-breaker
```

11. **Optional: View leaderboard and statistics:**

```
curl -H "Authorization: Bearer $ALICE_TOKEN" \
  http://localhost:5004/api/leaderboard

curl -H "Authorization: Bearer $BOB_TOKEN" \
  http://localhost:5004/api/leaderboard/player/alice
curl -H "Authorization: Bearer $BOB_TOKEN" \
  http://localhost:5004/api/leaderboard/player/bob
```

6 Testing the Application

All tests are automated through GitHub Actions CI/CD pipeline and can be run locally via Docker Compose.

7 Pytest Unit Tests (192 Total Tests)

7.1 Overview

Comprehensive Python unit tests using pytest covering all microservice endpoints with both valid and invalid input scenarios.

7.2 Test Distribution

Auth Service (test_auth_service.py) - 45 tests

- Registration (9 tests): success, missing fields, validation rules, duplicate handling
- Login (8 tests): success, wrong credentials, missing data, case sensitivity
- Profile management (11 tests): get/update profile, token validation, password changes
- Token validation (4 tests): valid/invalid/missing/malformed tokens
- Edge cases (5 tests): special characters, concurrent sessions, encoding
- Edge cases (8 tests): boundary conditions, special characters, Unicode handling

Card Service (test_card_service.py) - 29 tests

- Get all cards (3 tests): success, authentication checks
- Get by type (6 tests): rock/paper/scissors, invalid types, case insensitivity
- Get by ID (4 tests): success, not found, token validation, invalid format
- Random deck creation (7 tests): default/custom sizes, boundary validation, randomness
- Statistics (3 tests): success, authentication, percentage calculations
- Card types (3 tests): success, authentication, valid types verification
- Edge cases (3 tests): boundary conditions, format validation

Game Service (test_game_service.py) - 35 tests

- Game creation (5 tests): success, missing player, token validation, empty names
- Game state retrieval (6 tests): both players, not found, authorization, authentication
- Hand management (4 tests): get hand, not found, authorization, authentication
- Draw hand (4 tests): success, not found, authorization, authentication
- Play card (6 tests): success, missing/invalid/negative index, not found, authentication
- Resolve round (4 tests): success, not found, authentication, cards not played
- Edge cases (6 tests): concurrent actions, state validation, authorization

Leaderboard Service (test_leaderboard_service.py) - 32 tests

- Global leaderboard (6 tests): success, limits, max limits, authentication, ranking order
- Player statistics (5 tests): success, nonexistent players, authentication, data structure
- Recent games (5 tests): success, limits, max limits, authentication
- Top players (4 tests): success, authentication, list sizes
- Global statistics (4 tests): success, authentication, data consistency
- Edge cases (8 tests): zero/negative limits, special characters, boundary conditions

7.3 Running Pytest Tests

Prerequisites:

```
# Start Docker services
cd microservices
docker compose up -d --build

# Verify services are healthy (wait 30-60 seconds)
docker compose ps
```

Execute tests:

```
# Run all tests
pytest tests/ -v

# Run specific service tests
pytest tests/test_auth_service.py -v
pytest tests/test_card_service.py -v
pytest tests/test_game_service.py -v
pytest tests/test_leaderboard_service.py -v

# Run with coverage report
pytest tests/ --cov=microservices --cov-report=html
```

8 Postman API Tests (50 Total Tests)

8.1 Overview

Newman-based automated API testing using the `kat_postman_collection.json` collection. Tests validate API contracts, authentication flows, and error handling through the nginx gateway.

8.2 Test Distribution

Auth Service – 12 tests

- Registration endpoints (valid/invalid inputs, duplicate users)
- Login flows (success, wrong credentials, missing fields)
- Profile retrieval and updates
- Token validation and expiration

Card Service – 11 tests

- Get all cards with pagination
- Filter by type (rock/paper/scissors)
- Get individual card by ID
- Random deck generation with size validation
- Statistics and type listing endpoints

Game Service – 20 tests

- Game creation with player validation
- Game state retrieval with authorization
- Hand management (draw, get, play)
- Card playing with validation
- Round resolution logic
- Tie-breaker status and gameplay

- Turn information endpoints

Leaderboard Service – 7 tests

- Global leaderboard with limits
- Player-specific statistics
- Recent games listing
- Top players ranking
- Global game statistics

8.3 Running Postman Tests

Prerequisites:

```
# Ensure Docker services are running
docker compose ps

# Install Newman CLI
npm install -g newman
```

Execute tests:

```
# Run complete test suite
newman run tests/kat_postman_collection.json

# Run with detailed output
newman run tests/kat_postman_collection.json --reporters cli,json

# Run specific folder
newman run tests/kat_postman_collection.json --folder "Auth Service"
```

Import into Postman GUI:

1. Open Postman Desktop
2. Import → `tests/kat_postman_collection.json`
3. All environment variables pre-configured for `http://localhost:8080`
4. Run individual tests or entire collection

9 Locust Performance Tests

9.1 Overview

Load testing using Locust to simulate concurrent users and measure system performance under stress. Tests validate scalability, response times, and failure rates.

9.2 Test Scenarios

AuthServiceUser

- Simulates user registration and login flows
- Tests: register, login, get profile, token validation
- Weight distribution: Login (3x), Profile (2x), Validate (1x)

CardServiceUser

- Simulates browsing card collection
- Tests: get all cards, filter by type, random deck creation, statistics
- Weight distribution: Get all (5x), Get by type (3x), Random deck (4x)

GameServiceUser

- Simulates gameplay sessions
- Tests: create game, get state, draw hand, play cards, resolve rounds
- Weight distribution: Get state (3x), Draw hand (2x), Play card (2x)

LeaderboardServiceUser

- Simulates viewing rankings and statistics
- Tests: leaderboard, player stats, recent games, top players
- Weight distribution: Leaderboard (5x), Recent games (3x), Statistics (2x)

CombinedUser

- Complete end-to-end workflow: register → create deck → play game → check leaderboard
- Weight distribution: Workflow (10x), Leaderboard (5x), Profile (3x)

9.3 Running Locust Tests

Interactive mode (Web UI):

```
# Start Locust web interface
locust -f tests/locustfile.py

# Open http://localhost:8089
# Example: 10 users, spawn rate 2/s, host: http://localhost:8080
```

Headless mode (automated):

```
# 10-second test with 3 concurrent users
locust -f tests/locustfile.py \
  --headless \
  --users 3 \
  --spawn-rate 3 \
  --run-time 10s \
  --host http://localhost:8080

# HTML report
locust -f tests/locustfile.py \
  --headless \
  --users 50 \
  --spawn-rate 5 \
  --run-time 2m \
  --host http://localhost:8080 \
  --html=performance_report.html
```

Test specific services:

```
# Game service only
locust -f tests/locustfile.py --host http://localhost:8080 GameServiceUser

# Combined workflow
locust -f tests/locustfile.py --host http://localhost:8080 CombinedUser
```

9.4 Performance Metrics

- Response times: p50, p95, p99 percentiles
- Request rates: requests per second (RPS)
- Failure rates: percentage of failed requests
- Throughput: concurrent users vs response time

10 CI/CD Integration (GitHub Actions)

10.1 Automated Testing Pipeline

The `tests.yml` workflow executes all three test suites on every push/pull request.

Job 1: Pytest Tests

```
- Build Docker services
- Wait for health checks
- Run pytest with coverage
- Upload coverage reports
```

Job 2: Postman Tests

```
- Build Docker services
- Install Newman CLI
- Run kat_postman_collection.json
- Report test results
```

Job 3: Locust Performance Tests

```
- Build Docker services
- Run 10-second load test (3 users)
- Validate performance benchmarks
- Archive performance reports
```

All jobs run in parallel using the gateway URL `http://localhost:8080` with automated service health checks.

11 Test Execution Summary

Test Type	Tool	Count	Purpose
Unit Tests	pytest	192 tests	Endpoint validation, edge cases
API Tests	Newman/Postman	50 tests	Contract testing, integration
Performance	Locust	5 scenarios	Load testing, scalability
Total		**242 tests**	Comprehensive coverage

12 Quick Start: Running All Tests

```
# 1. Start services
cd microservices
docker compose up -d --build

# 2. Wait for services to be ready (30-60 seconds)
docker compose ps

# 3. Run all tests
pytest tests/ -v                                    # Pytest
newman run tests/kat_postman_collection.json        # Postman
locust -f tests/locustfile.py --headless --users 3 \
  --spawn-rate 3 --run-time 10s \
  --host http://localhost:8080                      # Locust
```

13 Key Testing Achievements

- **192 pytest unit tests** covering all endpoints with valid/invalid scenarios
- **50 Postman API tests** validating contracts through gateway
- **5 Locust performance scenarios** for load testing
- **Automated CI/CD pipeline** with parallel test execution
- **100% endpoint coverage** across all 4 microservices

- **Authentication & authorization testing** on all protected endpoints
- **Edge case validation** including boundaries, special characters, concurrent access