Maraffa Online

(v. .1.0)

Daniel Capannini
daniel.capannini@studio.unibo.it

September 9, 2024

The project involves the creation of an online multiplayer version of the card game Maraffa. The games will consist of 4 players who will compete 2vs2. The cards to play are 40, divided into 4 suits (cups, coins, clubs, swords).

1 Goal/s of the project

1.1 Q/a:

- *Is registration required to play?* No, just enter a nickname of your choice.
- *How many players are there?* 4 players, teams must consist of 2 players each.
- *How do I join a game?* You can create lobbies or join an existing lobby that is not full. When a lobby is full, you can start the game.
- *What happens if a player disconnects?* When a user disconnects even by closing the window without actually disconnecting, he is automatically deleted making his username available again and if he is in a game the other players will be brought back to the lobby removing the user who disconnected from it

1.2 regole

A game ends when a team gets at least 41 points, for each round 11 points are awarded, there is the possibility that 3 additional points are awarded to a team if it plays a Maraffa, that is, the player who chooses the trump suit has the ace, the 2 and the 3 of that suit and starts by playing the ace as the first card. For each round a player chooses the trump that is worth for that round and plays the first card, the players who follow are forced to play the same suit if possible. The value of the cards is as follows:

- 4-5-6-7 $\rightarrow$ 0 point
- knave-horse-king-2-3 $\rightarrow$ 1/3 point
- 1 $\rightarrow$ 1 point

The order of taking cards of the same suit is:
3-2-1-king-horse-knave-7-6-5-4
Once the match is over, players are returned to the lobby.

1.3 Usage scenarios

Below is the use case diagram that represents the possible interactions that the user can have fig. 1.

- any user can decide to log out.
- An unregistered user can enter his nickname to register.
- A registered user can either create or join an existing, non-full lobby.
- A registered user who is inside a lobby can change teams, leave the lobby or, if the lobby is full, start the match.

- A registered user who is in a match can play a card or set a suit when it is his or her turn or abandon the match at any time.

2 Background and link to the theory

In this project, we use several technologies:

- GSON: used for serializing and deserializing Java objects in JSON format so they can be communicated between the client and the server [2].
- REST Api: Utilized for facilitating communication between the game's front end and back end. REST APIs provide a flexible, easily accessible method to handle various game actions and data exchanges.
- javalin: easy-to-use framework for implementing web servers [1].

3 Requirements Analysis

3.1 Functional requirements

The system will operate on a client-server architecture which should provide a highly available system, capable of handling a large amount of user requests and, at the same time, maintaining the data in a consistent state, so that each client has the same information available (which is crucial during the game). All architectural requirements must not interfere with the user's interaction with the system (i.e., interactions with the system must be as flawless as if it were a simple single client-server architecture).

3.2 Not functional requirements

The game must also aim for high ease of use for users, for this reason there is a graphical interface which must be responsive and above all intuitive for users.

Disconnection Management: The system should handle user disconnections robustly. If a player suddenly disconnects during a match, the system should kick them out of the match and ensure that all players are returned to the lobby.

3.3 Top-down analysis

In order to satisfy the highlighted requirements, a Client/Server architecture was chosen, where the server in particular is implemented as a Web Service, equipped with a ReST API and using HTTP as a communication protocol. In order to try to get as close as possible to the objective of having a high scalability of the system, it was decided to implement the Web Service with the help of the Java Javalin framework, as it is easy to use, provides excellent performance and is very flexible.

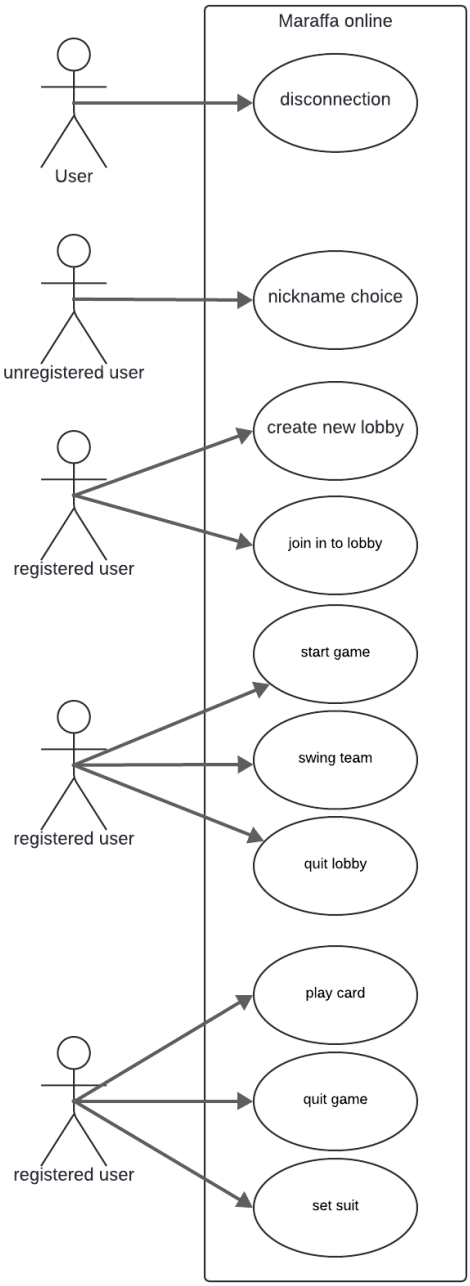

Figure 1: Use case diagram representing the interaction between user and Maraffa online

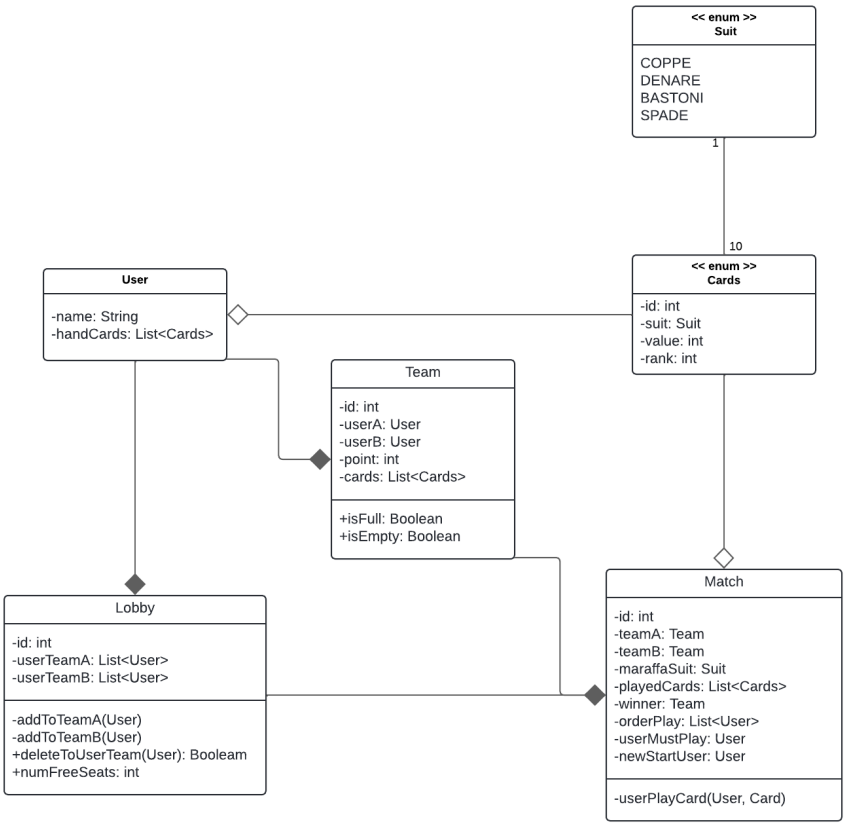

Figure 2: Model domain

4 Design

4.1 Structure

In fig. 2 the Model class diagram is shown, without the presence of *getter/setter*.

Suits The entity simply represents the concept of a card suit. It is implemented as an Enum, containing the four suits: *sticks, cups, coins, swords.*

Cards The entity represents the concept of cards, this is also implemented through an Enum, containing the 40 cards present, which have the following fields:

- *id* identifier.
- *siut* of belonging.
- *rank* which indicates its value to calculate the highest card.

- *value* which indicates the point value the card has.

User the entity represents the concept of user/player, formed by:

- *name* identifier.
- *handCards* which is the list of cards he has in his hand.

Lobby which represents the group of players who want to play, is made up of:

- *id* identifier.
- *userTeamA* which is the list of players on the teamA.
- *userTeamB* which is the list of players on the teamB.

Team used during the Match is made up of: an identifying *id*, *userA* *userB* which are the team members, *points* accumulated during the Match and *cards* which are the cards taken.

Match mainly contains *id* identifier and *teamA*, *teamB*, the main method of the class is *UserPlayCard(User, Cards)* which performs all the necessary checks and actions when a player plays a card.

4.2 behavior

In fig. 3 the State Diagram is shown from the perspective of a user, from the moment he starts the application to the moment a match is initiated. The application can terminate at any time when the user decides to close it.

Initially the user starts the application and enters a nickname that is not already in use. At this point the Home is shown where the list of all the lobbies is visible and selecting one will join it. Otherwise it creates a new one. In any case the Lobby screen will be shown with the lobby details (which can be full or not). When a lobby becomes full, any user can start the match. In fig. 4 the State Diagram of the logic of a match is shown.

4.3 interaction

To exchange information between client and server, it was decided to write messages using the Json format, through the serialization/deserialization of objects, with the help of the Gson library [2]. In fig. 5 the Sequence Diagram is shown for a typical sequence of information exchange via HTTP requests between client and server. The example is the creation of a new lobby by a client 1 and the connection of a client 2 to the newly created lobby and the abandonment of the lobby by client 1.

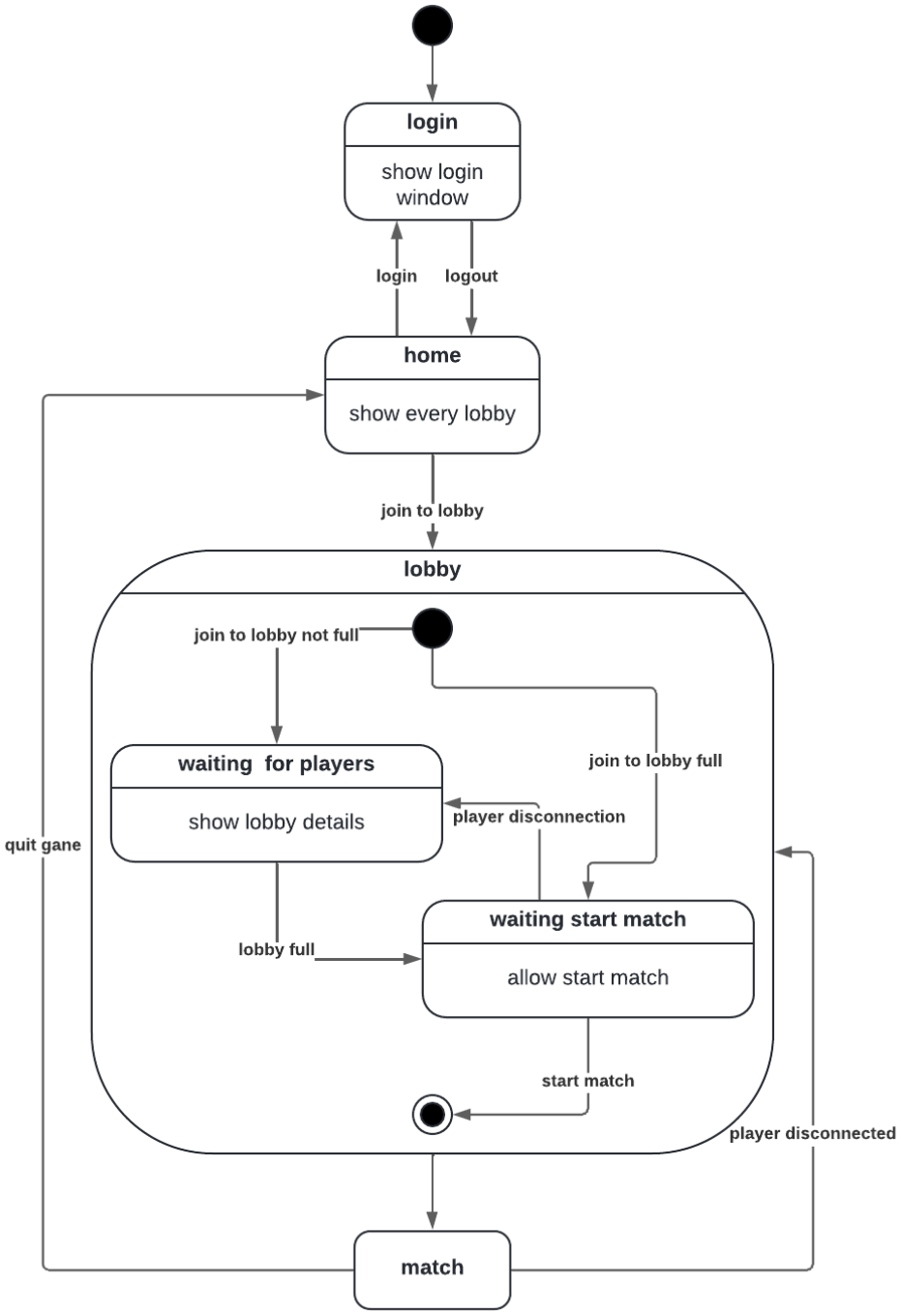

Figure 3: state diagram Maraffa online

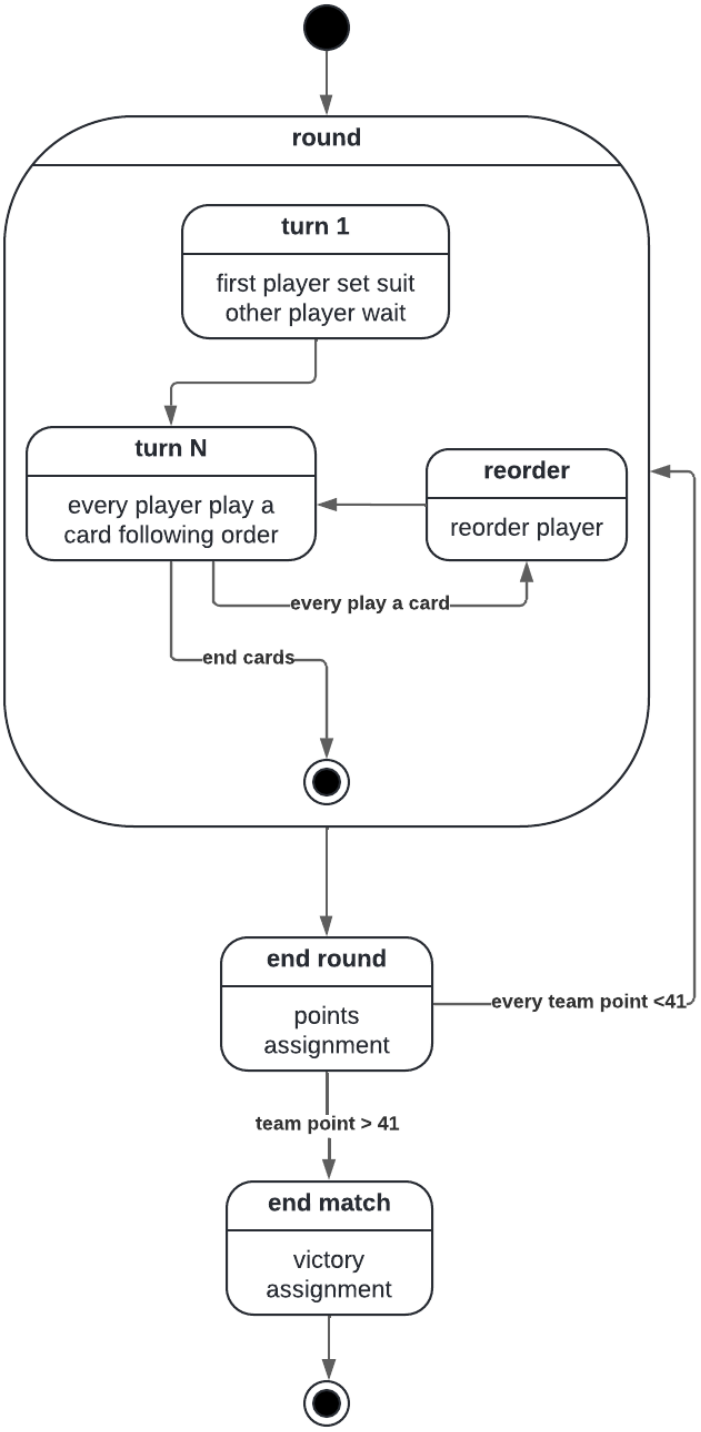

Figure 4: state diagram Match

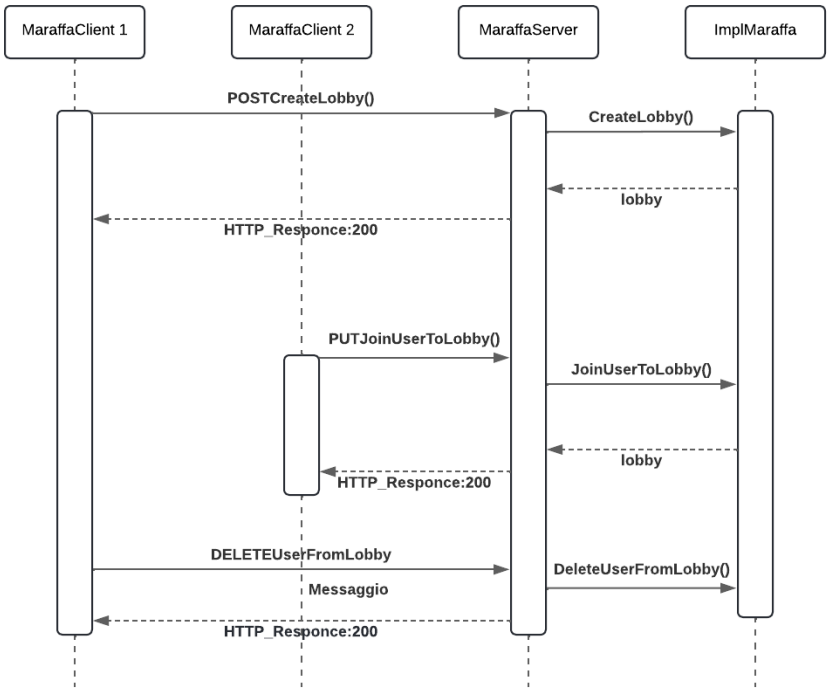

Figure 5: diagramma delle sequenze di una lobby

To facilitate understanding of the scheme, it is necessary to underline that, at the code level, a Maraffa interface has been created that is implemented in two distinct classes: ImplMaraffa server side (where all the active entities of the system are updated, namely User, Lobby and Match) and ClientMaraffa client side (where each method corresponds to an HTTP request to be sent to the server, which will call a method within LocalMaraffa).

The sequence of actions is analogous for any other type of operation involving the exchange of information between client and server, even for the User and Match entities, and to avoid repetitiveness they are not reported.

5 Implementation

The entire system consists of a multi-module Java project, divided as follows:

- forAll
- client
- server
- test

5.1 forAll

This module contains all the elements of the Model (Suit, Cards, User, Lobby, Team, Match) and the prenetation part, serializer/deserializer in JSON of the Java objects through the Gson framework [2], these parts are common to the whole project.

5.2 Client

The client implementation follows the MVC (Model View Controller) pattern. In particular, the view package provides a graphical interface (using the JavaFX [5] framework) with which the user can interact. The Model, as already mentioned, is contained in the common module. The controller package, instead, constantly sends and receives Model updates from the server by sending HTTP requests and applies the changes to the view.

5.3 Server

The server was implemented using Javalin [1]. In particular, it is a ReST-full web-service, equipped with Api. It acts on three main resources: User, Lobby, team and Match.

For each resource there is an associated resourceController that receives requests from clients and directs them to the corresponding resourceApi. The latter acts on a local instance of the game database (ImplMaraffa), implemented as a singleton.

HTTP requests to the server mainly refer to the following URLs:

- */user*
- */lobbies*
- */teams*
- */match*

Such requests can be of type GET, POST, PUT or DELETE, with the addition (if necessary) of parameters.

All routes recorded in the web server have been documented with the help of Swagger [7] (fig. 6, fig. 7) and can be consulted in the path *"http://host/port/doc/ui"*.

6 Validation

It was necessary to conduct a variety of tests with varying degrees of detail and emphasis on various areas to confirm that the software developed was accurate. The following subsection contains an account of the parts that were tested using autonomous testing. Additionally, manual testing was done, in particular to evaluate the GUI and for all match.

The tests were developed using JUnit 5 [6]. In fig. 1 all the tests that were generated are shown, in particular the model and the connection between the client-server were tested.

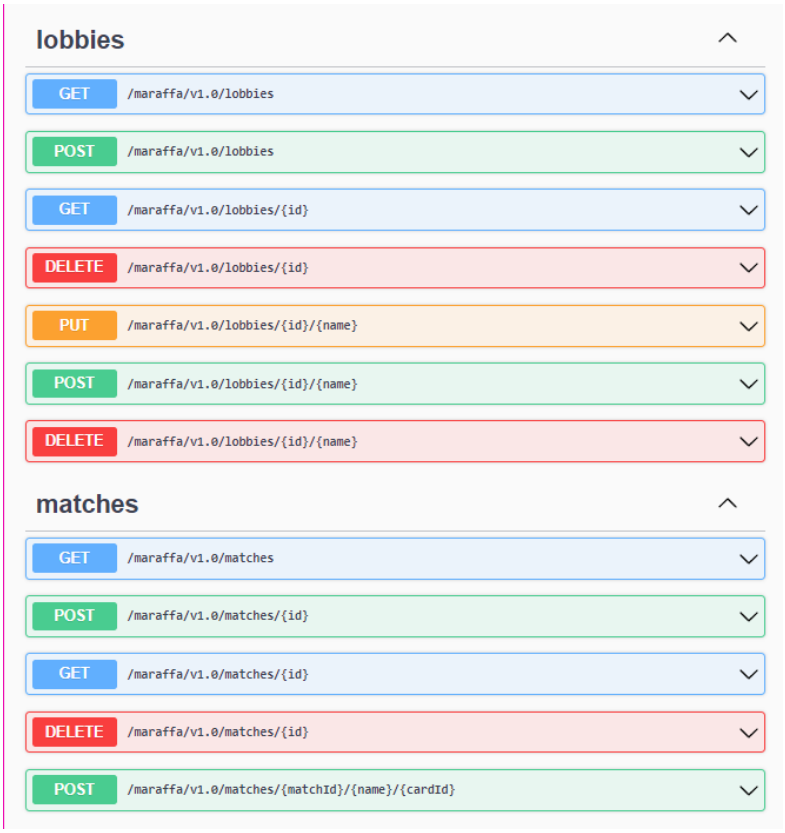

Figure 6: Swagger API docs

Figure 7: Swagger API docs

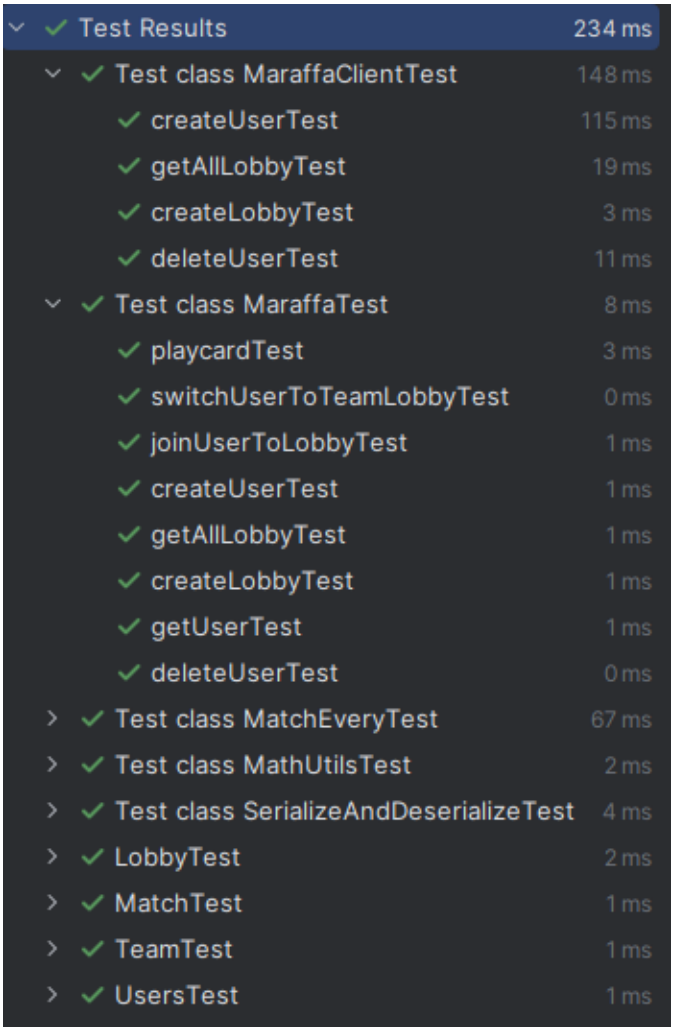

Figure 8: JUnit test

To honor the practice of "Continuous Integration", the CI/CD Pipelines of GitLab [3] were exploited. In detail, at each push of a commit on GitLab, a new pipeline is created following the instructions in the file *.gitlab-ci.yml* listing 1. In each pipeline, the project is initially built and subsequently all the tests present in it are executed.

```
image: gradle:latest

build:
  stage: build
  script:
    - cd maraffa-online
    - gradle classes testClasses

test:
  stage: test
  script:
    - cd maraffa-online
    - gradle test

  artifacts:
    when: always
    reports:
      junit: '**/build/test-results/test/**/TEST-*.xml'
```

Listing 1: .gitlab-ci.yml

Manual tests on the functionality of the GUI and extensive tests on the flow of the game were performed.

7 Deployment Instructions

The choice for the deployment fell on Gradle [4]. To be able to play Maraffa Online it is sufficient that the machine is equipped with a JVM (version 17). Once this simple requirement is satisfied, the server can be launched using the following command (after having positioned itself from the terminal inside the main folder of the repository): *./maraffa-online/gradlew server:run* This command will launch a default Maraffa Online Web Service instance at *http://localhost:666*. You can specify a different port as an argument to the previous command. To launch the game client, use the following command:

./maraffa-online/gradlew client:run

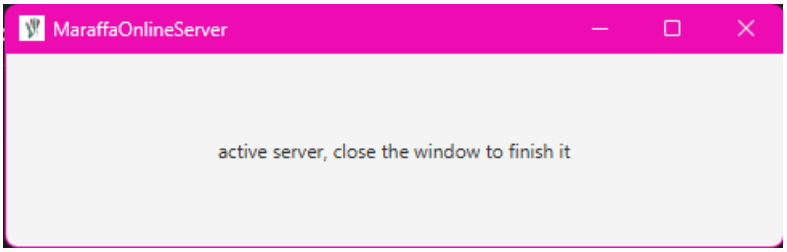

Figure 9: Server View

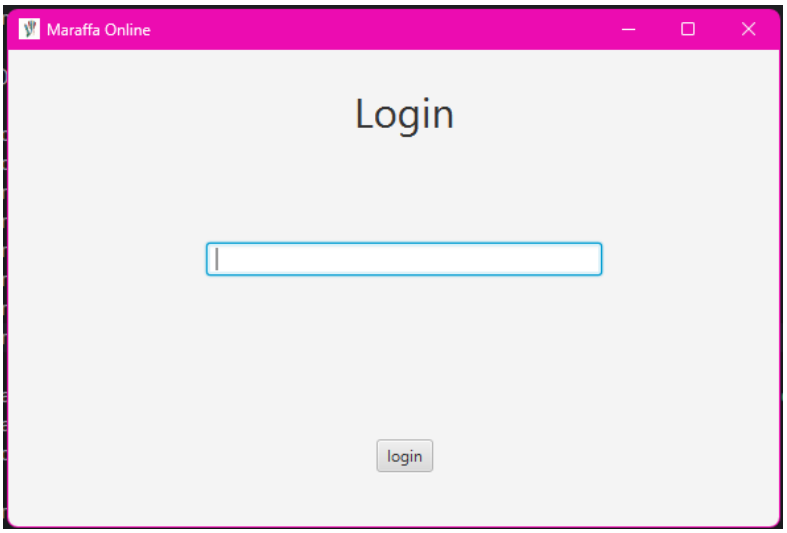

Figure 10: Login View

8 Usage Examples

First you need to start the server, when it is started as in fig. 9 it is listening and to terminate it you just need to close the window.

at this point you can start the client, in fig. 10, fig. 11 and fig. 12 the login view, where the user chooses his nickname, and home view, where the user can enter the lobbies of other players that are not complete or create a new lobby or return to the login view.

Nella fig. 13 a lobby view is shown, where you can quit, change team if possible or start the game when the lobby is full.

In fig. 14 the match screen is shown, in which the teams' score can be seen in the top left, but the current trump. in the top center it is shown whether it is your turn or not. in the center the played cards are displayed, in the lower part of the screen is the player's hand.

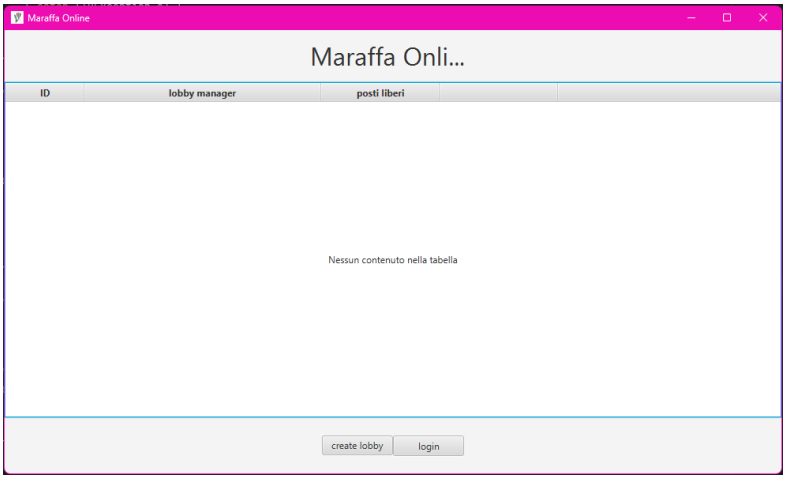

Figure 11: Home View

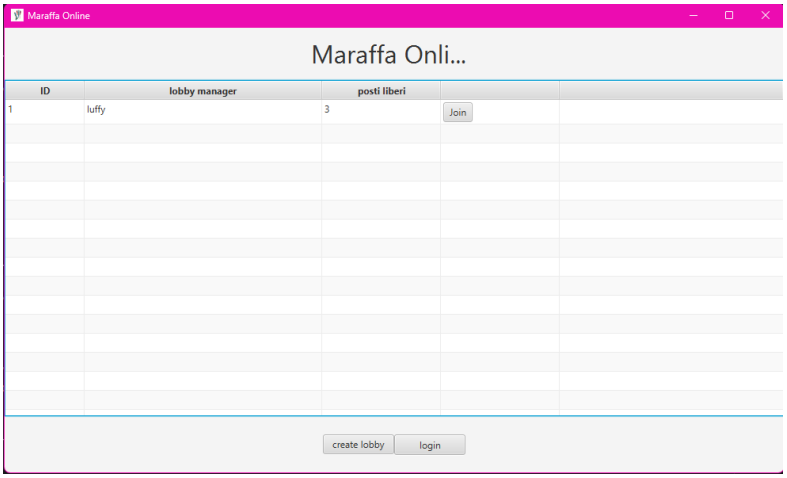

Figure 12: Home View with lobby

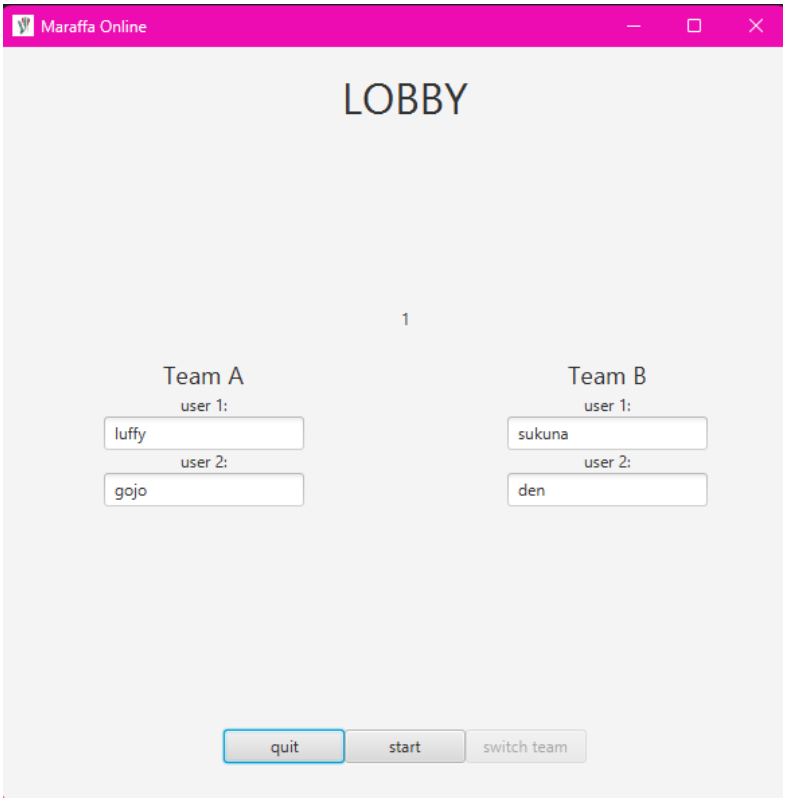

Figure 13: Lobby View

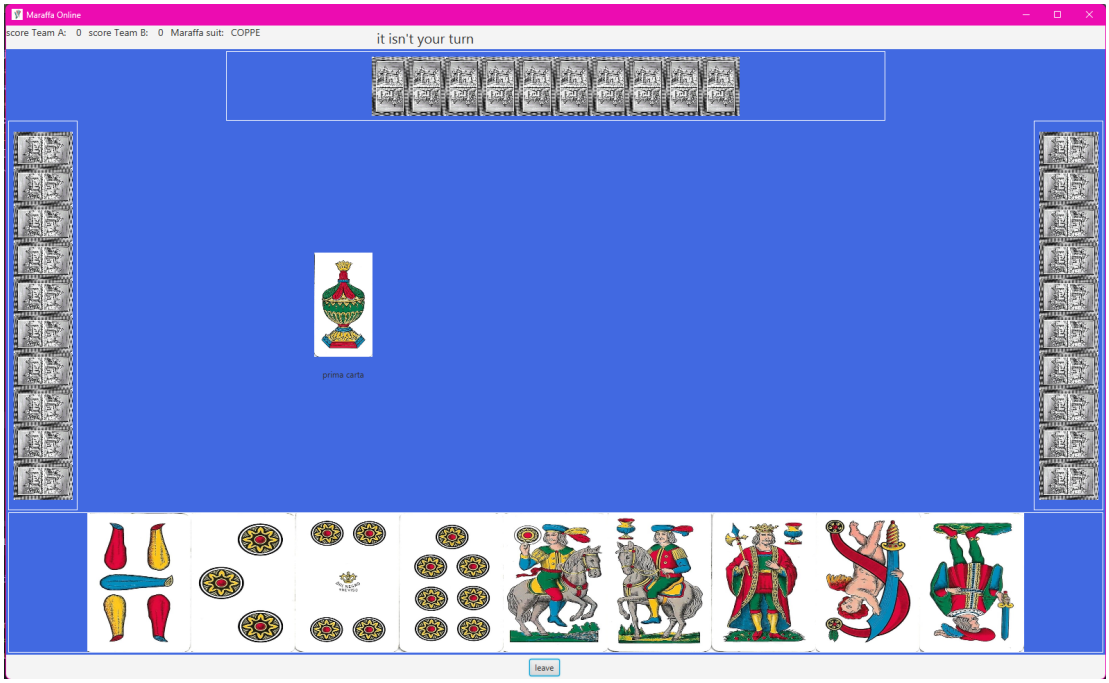

Figure 14: Match View

9 Conclusions

I can consider myself satisfied with the result obtained, all the pre-established points have been reached. The project is an online version of the Maraffa game that respects all its rules, is scalable and easy to use and deploy.

9.1 Future Works

Future improvements to the project could be:

- the improvement of the graphical interface, making it more graphically refined.
- the adoption of a database so as to be able to keep track of players over time, also adding statistics on their game.
- with the adoption of a database we could also move to a more advanced authentication including username and password.

9.2 What did we learned

The work experience has allowed me to deeply immerse myself in the world of Distributed Systems, acquiring a detailed understanding of the logic, problems, as well as the advantages and disadvantages related to this field. I have also learned the crucial importance

of considering *Fault Tolerance* already starting from the application design phase, an aspect that is often overlooked and postponed to the last stages of implementation, with the risk of generating significant technical debt.

Finally, this project gave me the opportunity to delve deeper into several concepts related to web technologies, in particular regarding the creation of a Web Service and its REST APIs.

References

[1] javalin, URL: `https://javalin.io/`

[2] Gson, URL: `https://github.com/google/gson`

[3] CI/CD Pipelines GitLab, URL: `https://docs.gitlab.com/ee/ci/pipelines/`

[4] Gradle, URL: `https://gradle.org/`

[5] JavaFX, URL: `https://openjfx.io/`

[6] JUnit 5, URL: `https://junit.org/junit5/`

[7] Swagger. URL: `https://swagger.io/`