Sandpiles

Project for the course *Applicazioni e Servizi Web*

Andrea Biagini - 0001145679 andrea.biagini5@studio.unibo.it

Filippo Gurioli - 0001146182 filippo.gurioli@studio.unibo.it

Leonardo Randacio - 0001125080 leonardo.randacio@studio.unibo.it

Introduction

Sandpiles is a project designed to emulate the functionality of online board game platforms, providing an interactive and competitive environment for players of all skill levels to play *Sandpiles*. The platform offers real-time matchmaking, AI-driven opponent, statistics review and match history review.

The *Sandpiles* game is based on the Abelian sandpiles mathematical model. It sees two players face each other head to head in a strategic game with time constrains.

Developed with a focus on usability, performance, and scalability, *Sandpiles* aims to demonstrate best practices in web application development while delivering an engaging and accessible game experience.

Requirements

During project analysis the following requirements have been identified.

Functional Requirements

User Functional Requirements The user can:

1. User authentication
    1. Register a new account on the website
    2. Login to the website using an existing account
    3. Logout from the website
2. Profile
    1. View profile statistics
    2. View profile game history
        1. Review old games move by move
    3. Customize profile settings
        1. Load a custom profile picture
        2. Set the website theme
        3. Change profile name
        4. Change profile password
        5. Customize the game board style choosing from some given styling options
3. Game

    1. As a logged in user
        1. Play a ranked game against other users of similar rating
        2. Play an unranked game against an AI opponent
    2. As a guest user
        1. Play an unranked game against an AI opponent
    3. Gameplay
        1. Add a grain to a controller pile
        2. View opponents moves
        3. View personal and opponent clock time remaining
        4. Resign the current game
        5. Disconnect from the current game
4. View the website leaderboard
5. View a tutorial on *Sandpiles* game rules
6. Developers
    1. Use the public API to access data about players and games, in an authenticated manner
    2. Find the public API documentation on the *Sandpiles* website

System Functional Requirements

- usare jwt?
- gestione dati sensibili (hashing locale, salt in db, conferma e-mail)

1. Security
    1. Password hashing client side
    2. Matching double password input at registration time
    3. Matching double password input at password change time
    4. Hashed password storage with salting server side
    5. Profile confirmation with e-mail
2. User session handling
3. Badge system handling to encourage user retention

Non-Functional Requirements

1. Responsive UI: The website interface adapts to the user's device to allow identical use across devices
2. AI opponent responds to the user's moves in under a second
3. Intuitive UI design allows users to interact with the website intuitively

Implementation Requirements

1. MEVN
    1. MongoDB database technology
    2. Express.js backend technology
    3. Vue.js frontend technology
    4. Node.js runtime environment
2. Logical programming language for game AI

Design

The following design has been developed starting from the user stories specified here. From the user stories we also have derived the domain model (which established an ubiquitous jargon) and the pages flow diagram as shown below.

```
flowchart TD
    A@{ shape: circle, label: "Start" } -->|auto login| Play(Play)
    A@{ shape: circle, label: "Start" } --> Landing(Landing)
    Landing <--> Login(Login)
    Landing <--> Register(Register)
    Landing <--> Tutorial(Tutorial)
    Landing -->|play bot| Match
    Register -->|registration completed| Login
    Login --> Play
    Play <--> Tutorial(Tutorial)
    Play <-->|play ranked| MatchMaking(Match Making)
    Play -->|play bot| Match
    Play <--> Profile(Profile)
    Tutorial -->|play ranked| MatchMaking
    Tutorial -->|play bot| Match
    MatchMaking -->|opponent found| Match(Match)
    Match -->|quit| Play
    Match -->|game over| GameEnd("Game End (stats)")
    GameEnd -->|play again| Match
    GameEnd --> Play
    GameEnd --> Leaderboard
    GameEnd --> Profile
    Leaderboard <--> Profile
    Profile <--> MatchHistory(Match History)
    MatchHistory <--> Replay(Replay)
```

Based on the pages flow a project mockup has been developed also taking into account user experience and adopting a user centered approach.

Requirements suggested an hexagonal client server architecture as explained in the dedicated sub-section.

Domain Model

Domain: Sandpiles game

Contexts:

- Account Management
- Game Experience
  - PVP
  - PVE
  - Matchmaking

- Game History

Glossary:

User: a human interacting with the application through the UI

Player: a User interacting with the sandpiles game.

Login: access the application with a given account.

Register: the creation of a new account.

Game Board: square field divided into square Cells

Cell: a tile of the Game Board.

Pile: a pawn on a Cell owned by a Player, with a number of Grains.

Grain: fundamental unit of a Pile.

Owner of a Pile: the player who owns the pile.

Match: a sequence of Turns on the Game Board ending when the Win Condition is satisfied for one player.

Win Condition: the Game Board state where all Piles are owned by one player or the opponent clock's time is finished.

Turn: a player move and its eventual collapses.

Move: interaction by which a player increases the number of grains in an owned pile by 1.

Collapse: the event of a pile reaching 4 grains. This event will:

- remove the current pile
- let the adjacent cells be conquered by the owner of the collapsing pile

Conquer: the event of a cell where:

- if empty, a pile with 1 grain is placed
- if not empty, the pile is incremented by 1 and, if the owner is different, the ownership is changed to the conqueror (i.e. the player who made the move)

Clock: time counter that tracks how many seconds each player has left to play to game

Glicko rating system: method used to calculate and update player ratings in competitive games and sports (link).

Glicko Ranking: ranking rappresenting the strenght of a given player.

Matchmaking: the algorithm responsible for selecting players to face each other among the players available.

Context map The following context map arises from the previous description.

```
graph
   subgraph gameExperience["Game experience"]
      subgraph matchContext["Match context"]
         player1("Player")
         GameBoard
         Cell
         Pile
         Grain
         PileOwner
         Match
         WinCondition
         Move
         Turn
         Collapse
         Conquer
         Clock
         glickoRanking1("Glicko Ranking")
      end

      subgraph matchMaking["Matchmaking"]
         glickoRanking2("Glicko Ranking") <--> glickoRanking1
         player2("Player") <--> player1
      end
   end

   subgraph accountManagement["Account management"]
      User <--> player1
      User <--> player2
      Login
      Register
   end
```

Mockup

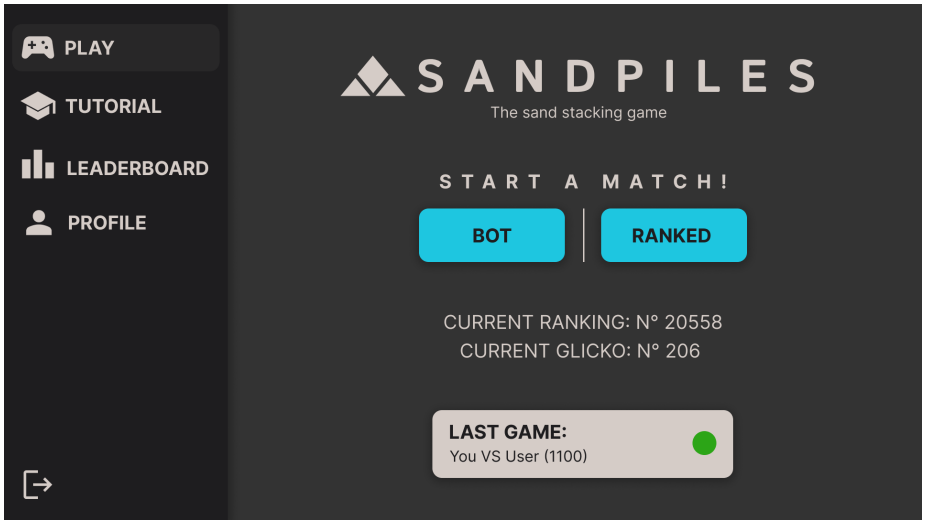

Figure 1: Dashboard

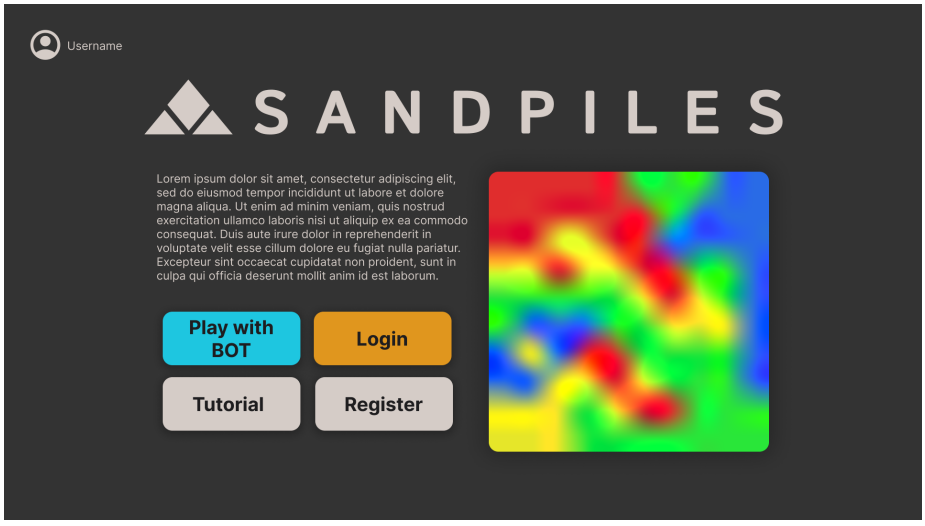

Figure 2: Landing

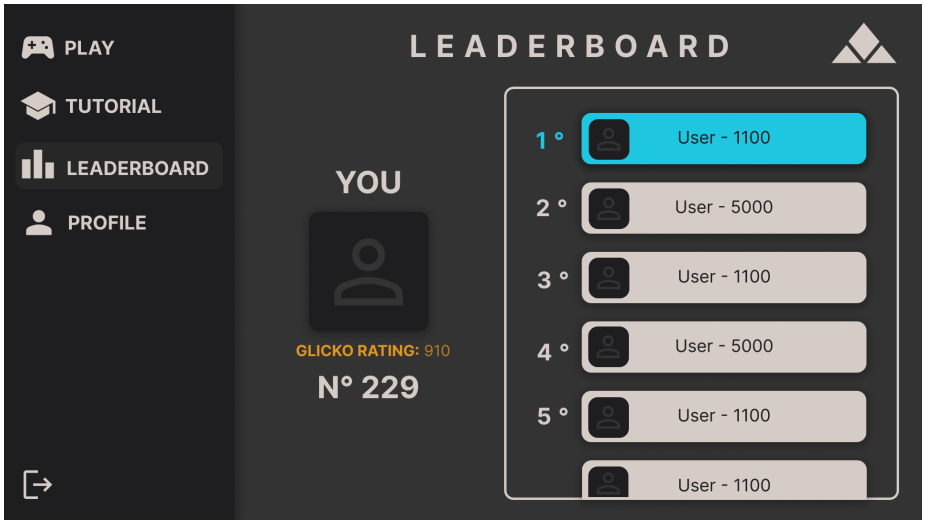

Figure 3: Leaderboard

Figure 4: Login

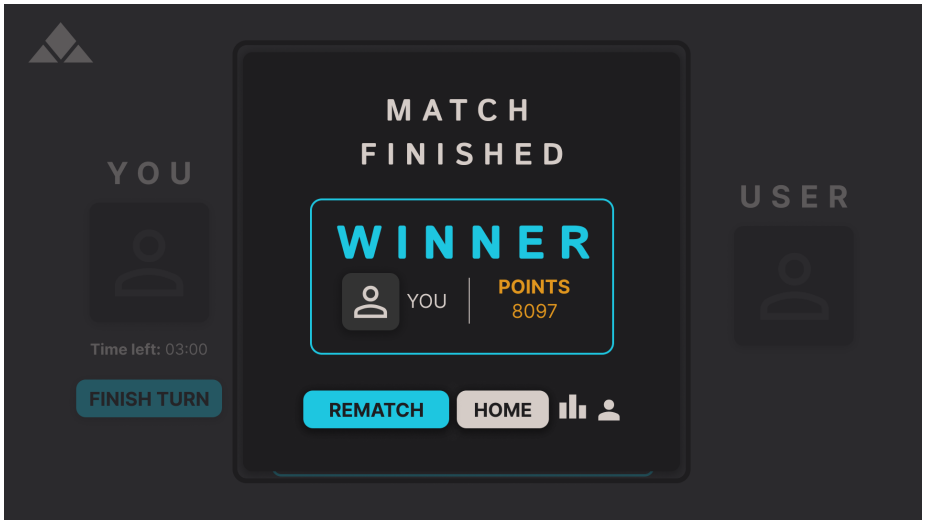

Figure 5: Match end

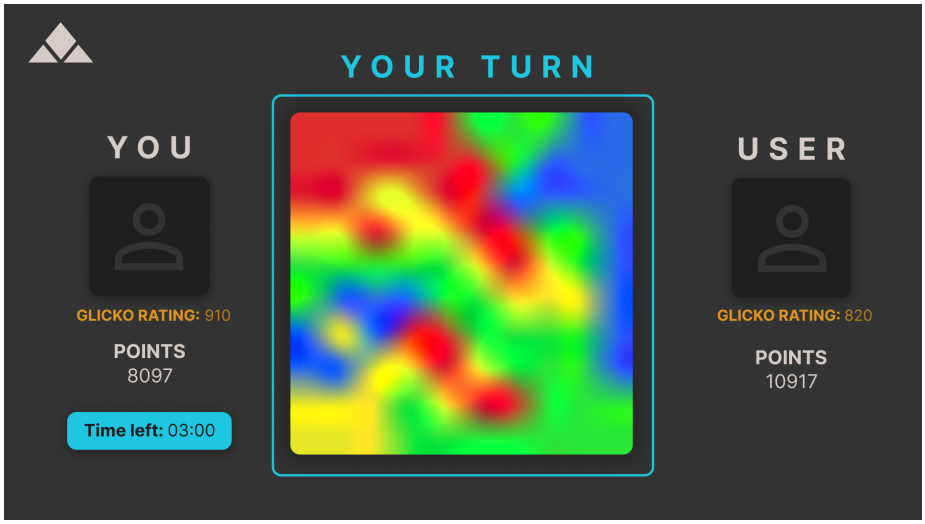

Figure 6: Match

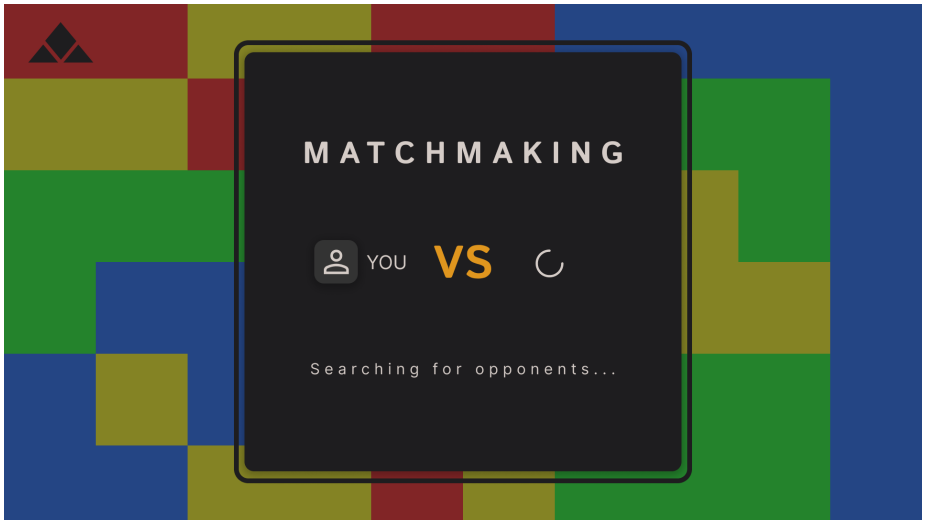

Figure 7: Matchmaking

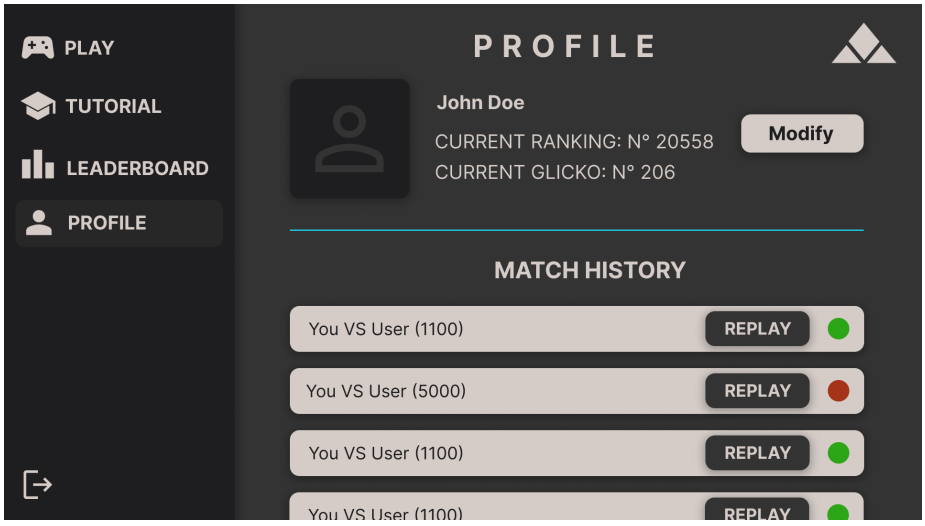

Figure 8: Profile and Match History

Figure 9: Register

The following are the responsive versions of the mockup pages.

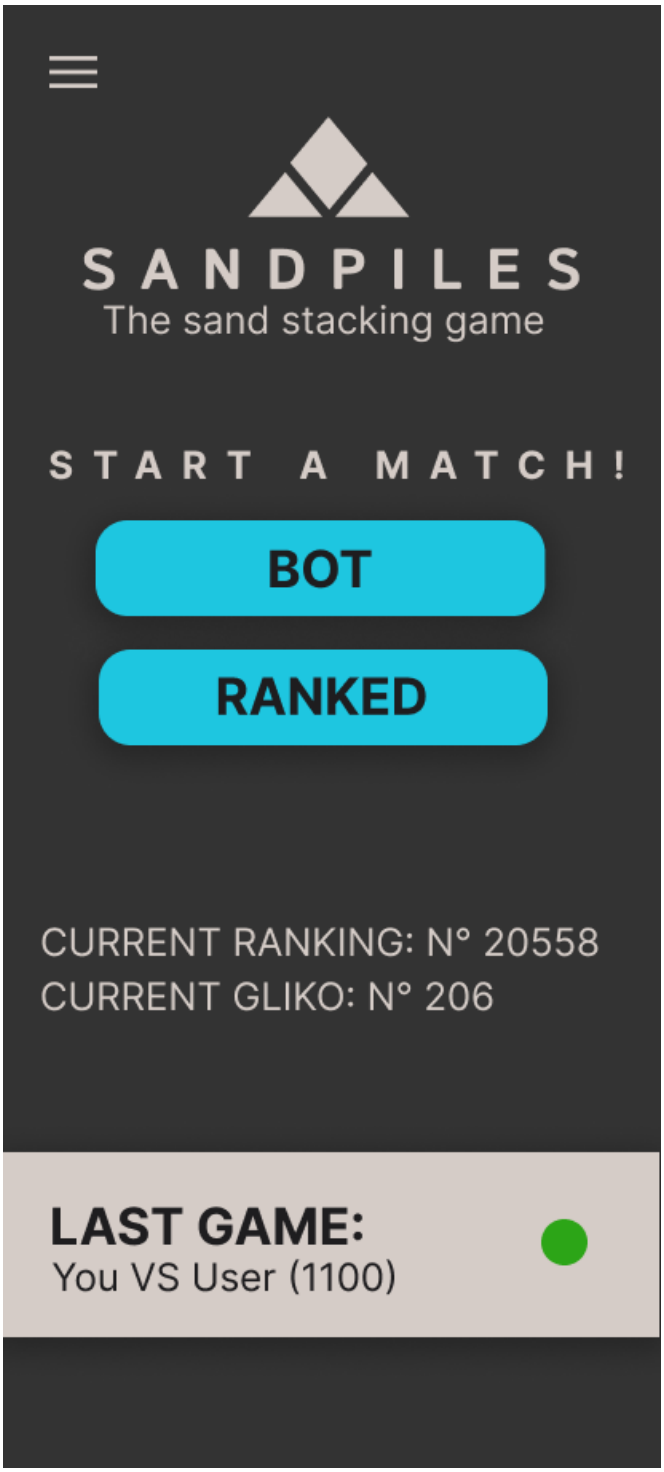

Figure 10: Dashboard Responsive

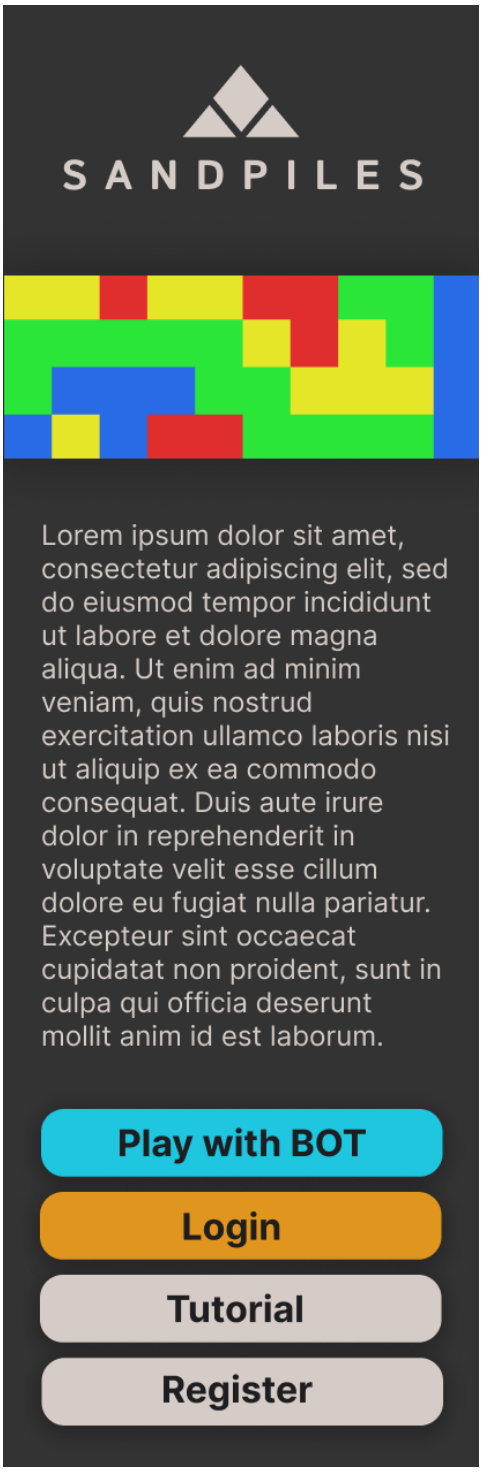

Figure 11: Landing Responsive

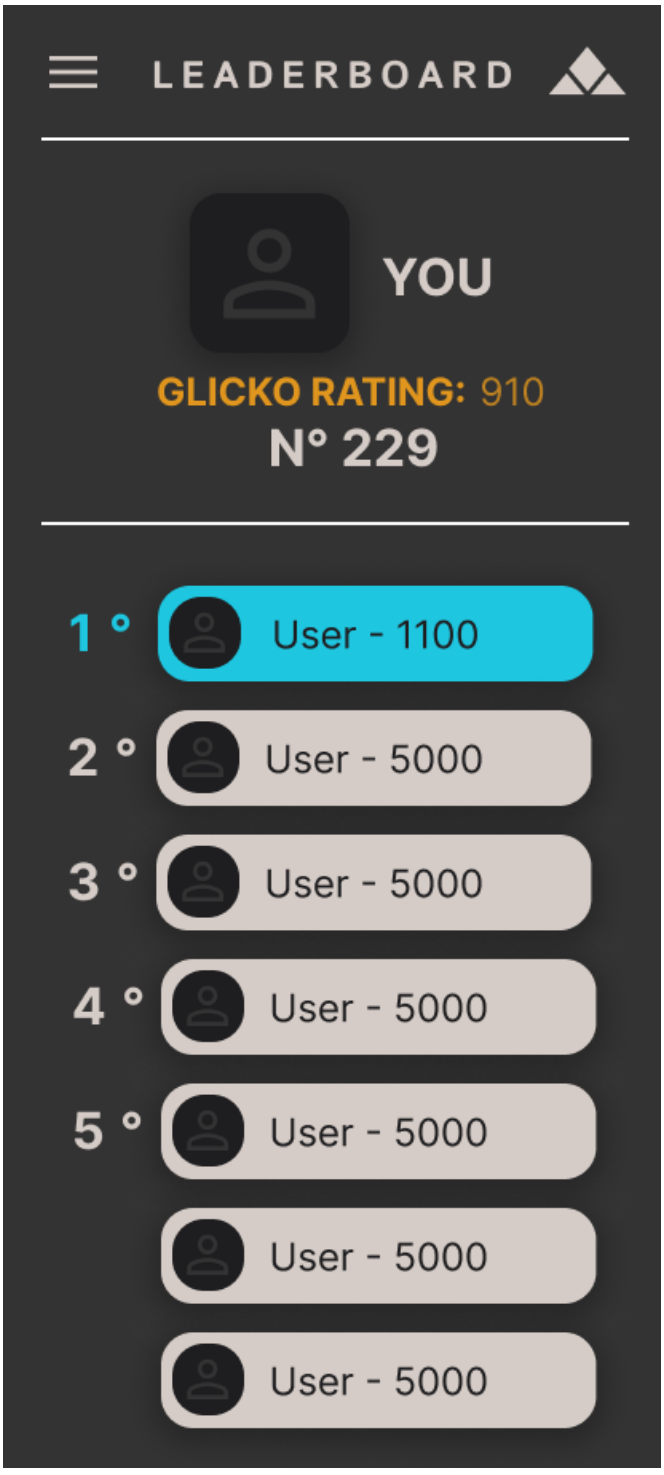

Figure 12: Leaderboard Responsive

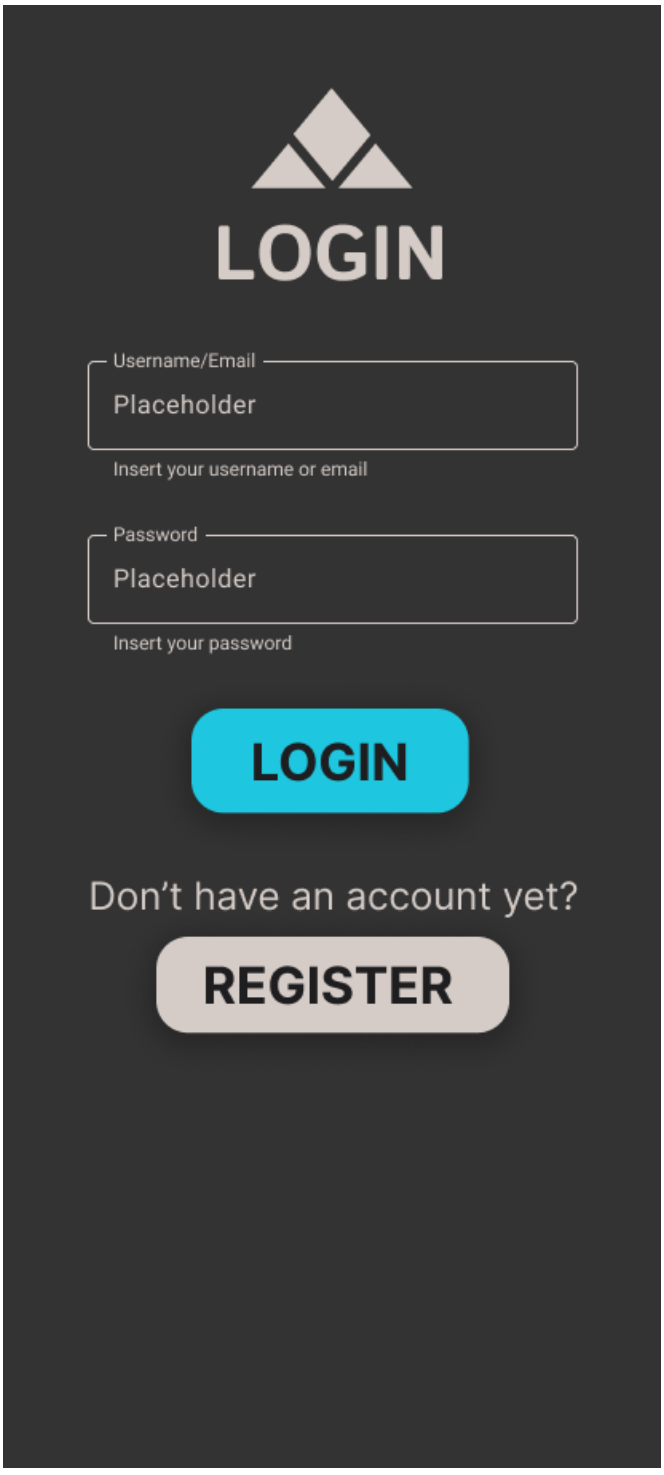

Figure 13: Login Responsive

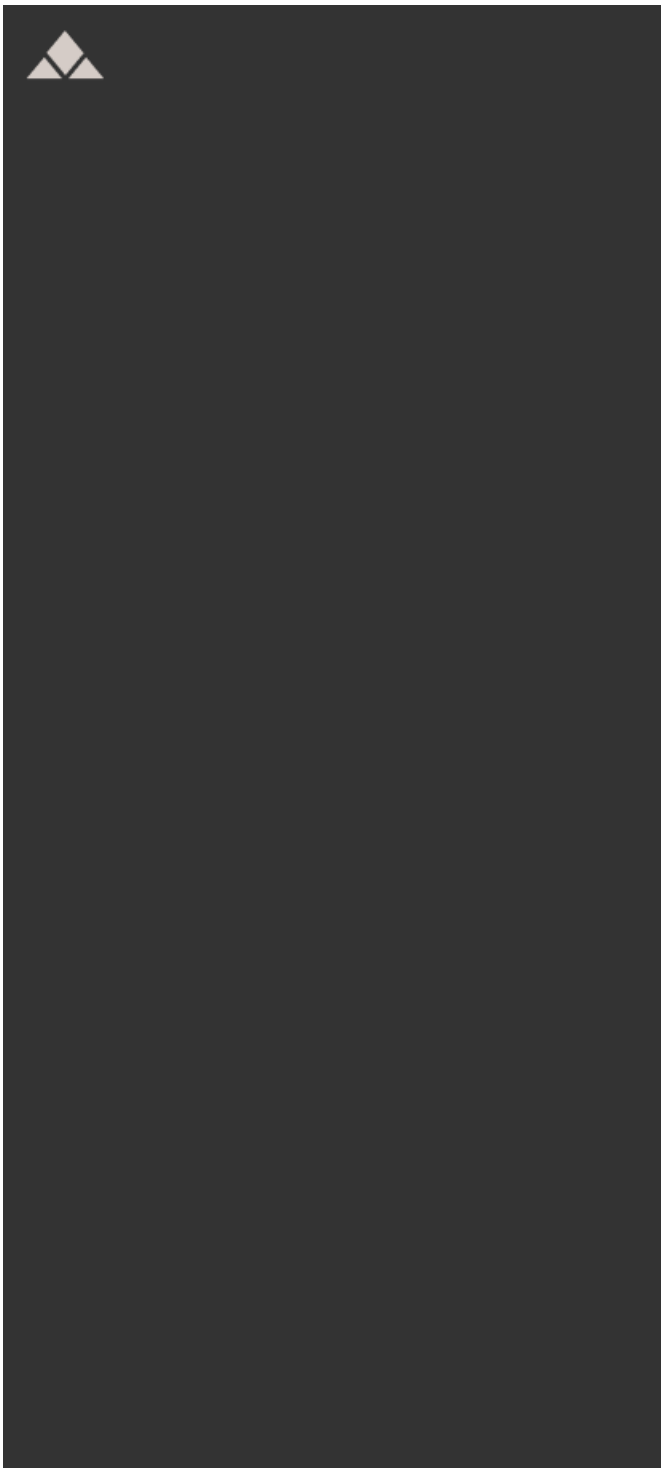

Figure 14: Match end Responsive

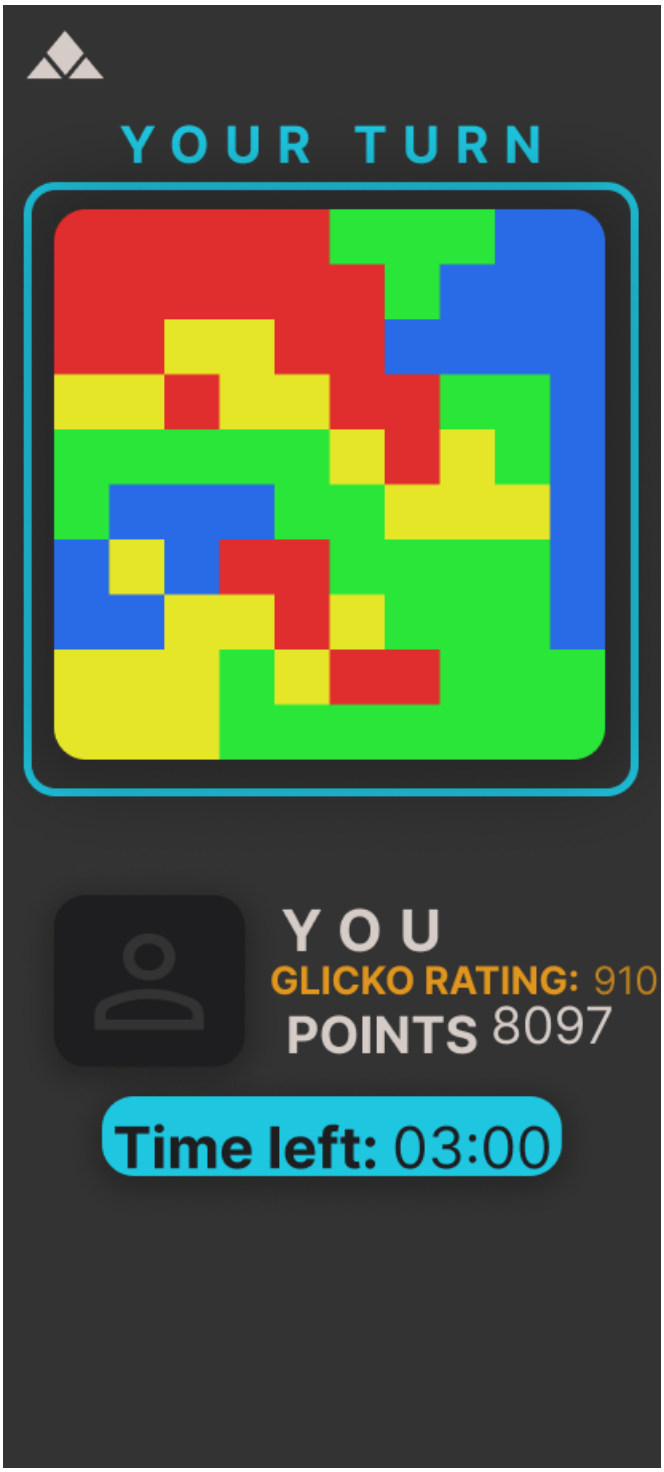

Figure 15: Match Responsive

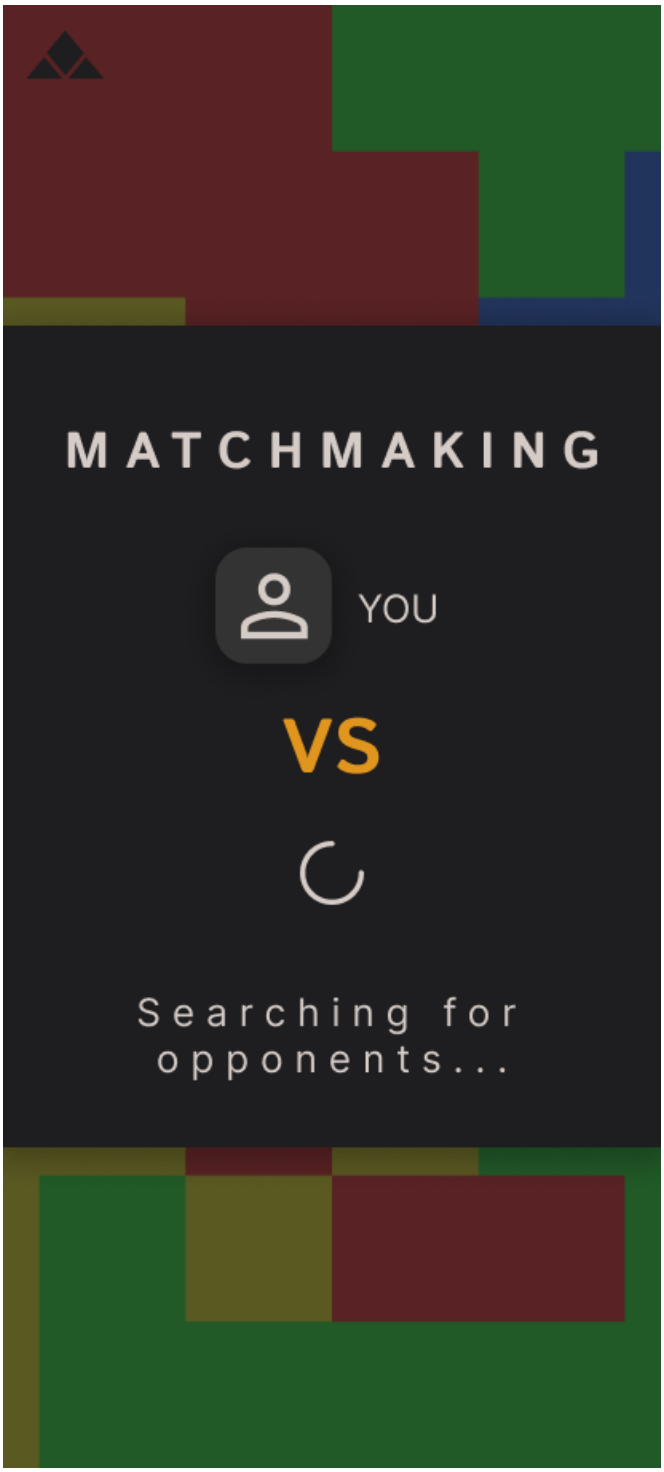

Figure 16: Matchmaking Responsive

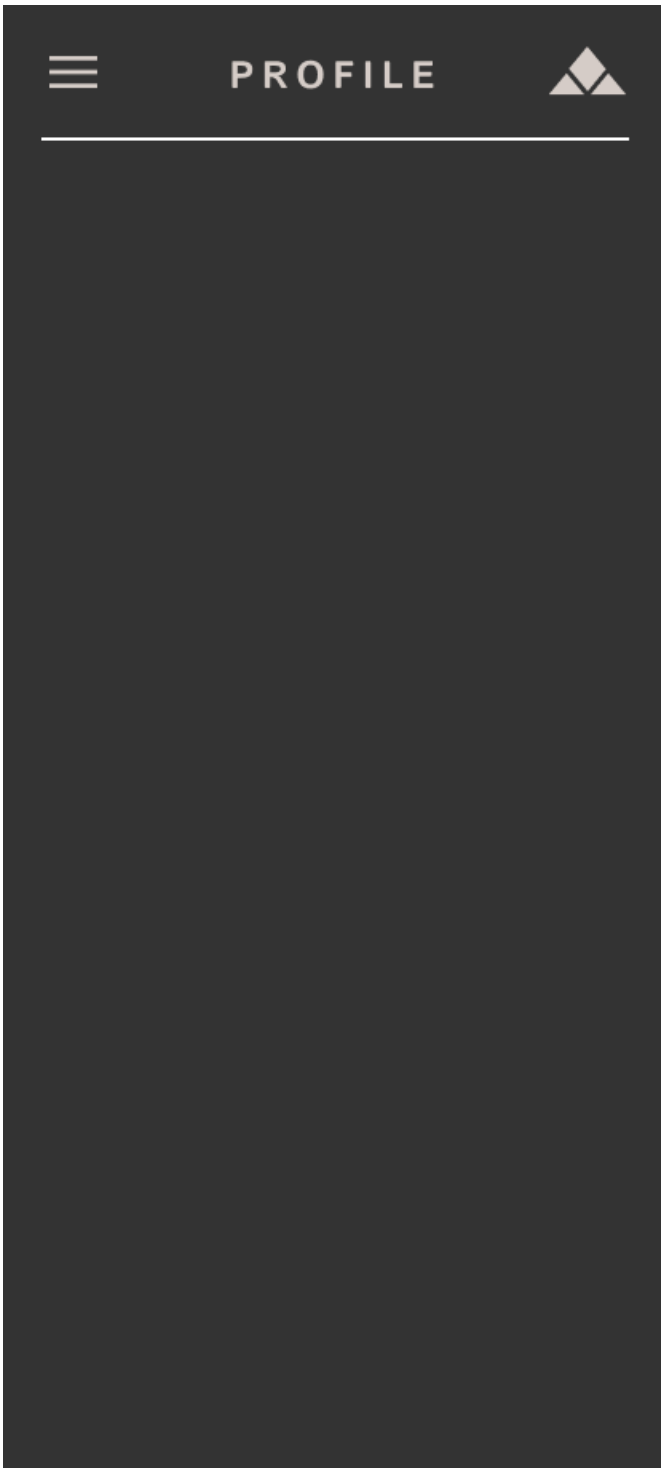

Figure 17: Profile and Match History Responsive

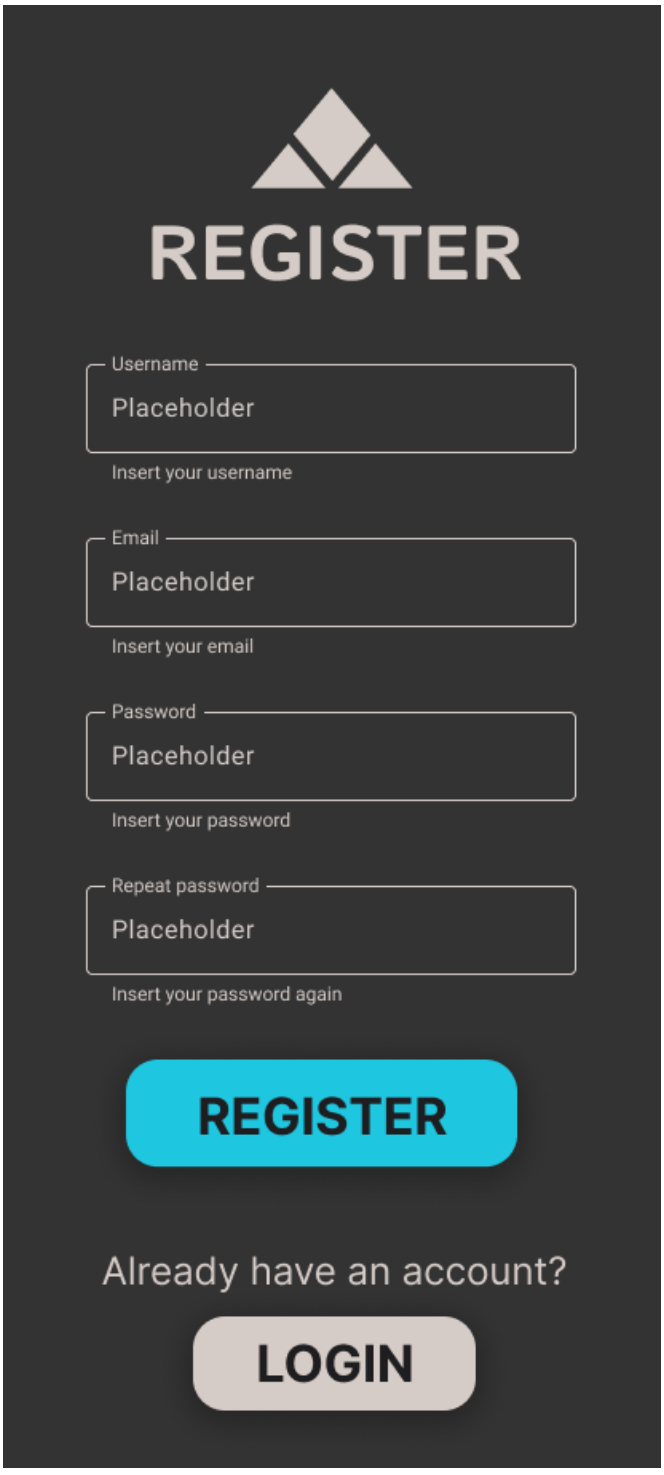

Figure 18: Register Responsive

Architecture

Frontend The frontend uses a component-based architecture.

The main components map one-to-one the pages listed in the mockup.

Also some components are present in all pages, such as the header and the footer.

```
classDiagram
      class Header
%% TODO ADD UML COMPONENT DIAGRAM
```

Components diagram

Backend The backend uses an hexagonal architecture, leveraging Domain Driven Design (DDD) principles.

Detailed Design

Game State Data Representation A given match can be represented uniquely by:

- A starting board
- A list of moves

The starting board can be represented as a matrix of dimensions dxd where every given xi,j is the number of grains in the cell with coordinates i,j.

The list of moves can be represented as a list of tuples (i,j) where the tuple represents the coordinates of the cell where the player has decided to add a grain.

Implementation

Technologies

MEVN

Code

Solo aspetti rilevanti.

Tests

jest and stuff

Deployment

To execute the system via Docker Compose, it is necessary to create the file `src/server/secrets/mongo_root_password.txt`. Additional instructions are provided in the `secrets` directory.

Conclusioni

Conclusioni