Milestone 4 Team 3

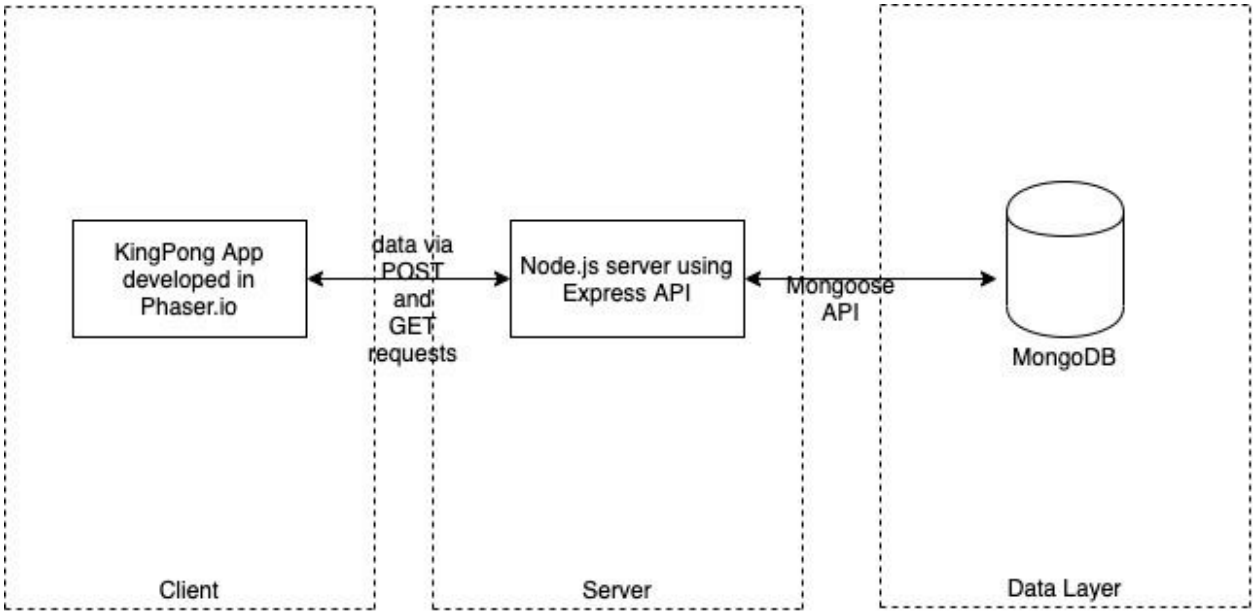
Revised List of Features:

- Paddle interaction via keyboard input -COMPLETE
  This serves as the user's actual interaction with the game.
- Ball interaction with user (including world boundaries and physics of collisions) -COMPLETE
  This is the main game piece that is automated and provides the actual challenge of the game.
- Auto restart based on ball crossing paddle boundaries -COMPLETE
  This keeps the game running automatically when a ball goes out of bounds signifying a lost/won point.
- Score count and display -COMPLETE
  This shows to the player how well they are doing in the game.
- Login Page -COMPLETE
  Allows for the user to create a profile and log in such that their data is associated with their account.
- Node.js local server -COMPLETE
  Serves as the integration layer between the front end and back end.
- Paddle and ball image -COMPLETE
  Actually displays the game pieces to the user.

Features to be Completed (ordered in descending priority):

1. Multiplayer
   This will enable the project to be transformed from the original pong to our idea of battle royale pong. In our research, the actual implementation of this feature encompasses the previously included features of "multiple instances of the game" and "synchronize games".
2. Shrinking/growing n-gon
   This is the mechanism by which the player can view their part in the multiplayer game.
3. Account System
   This is the manner by which a user's high score will be saved, and they will see themselves in the global scoreboard.
4. Global Scoreboard
   This is the display of the overall ranking of all players throughout all played games.
5. Different Color Paddle per Player
   This feature is for the sack of user experience, so that the user can easily differentiate themselves from other players.
6. Waiting Lobby
   This ensures the user understands that the game is working while waiting for enough players.
7. Ball Subtraction/Addition
   This ensures that with more players and therefore a large n-gon, the game doesn't get too easy or boring.

Architecture diagram:



Front End Design:

Login Page:

Registration Page:

Register

Enter a valid email address:

Email Address

☐ Valid email address

Enter a password of at least 8 characters:

Password

☐ Valid password

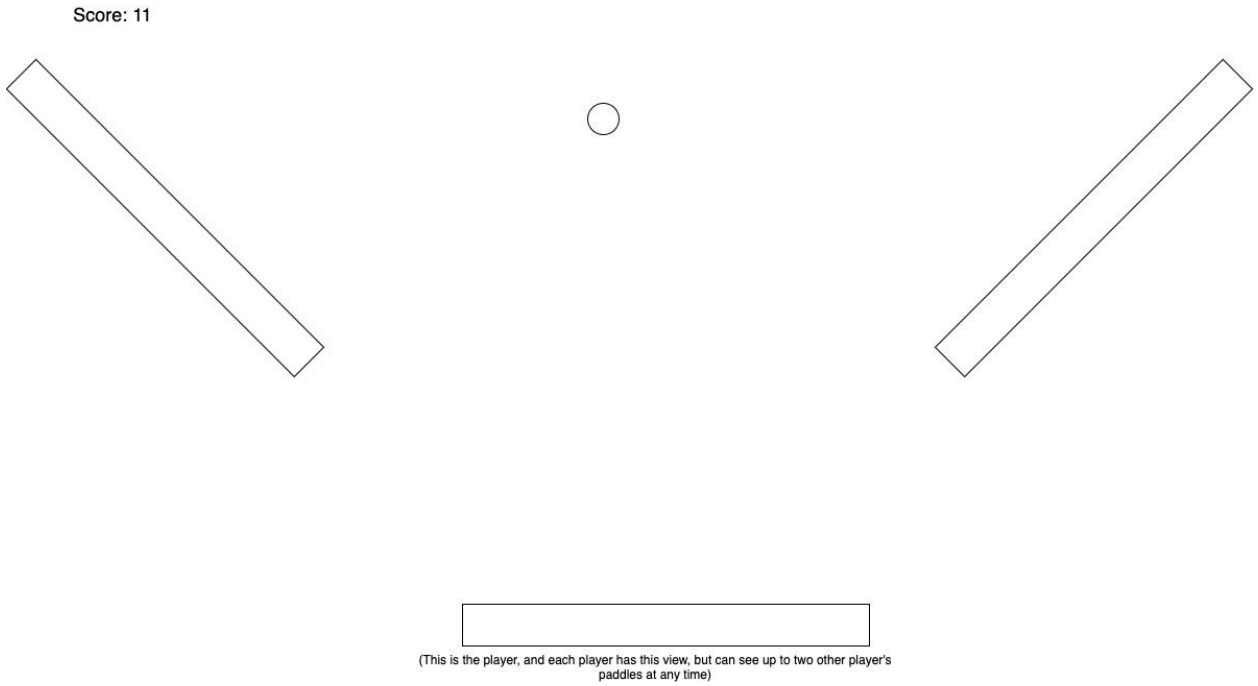
Reenter Password

Password

☐ Same password

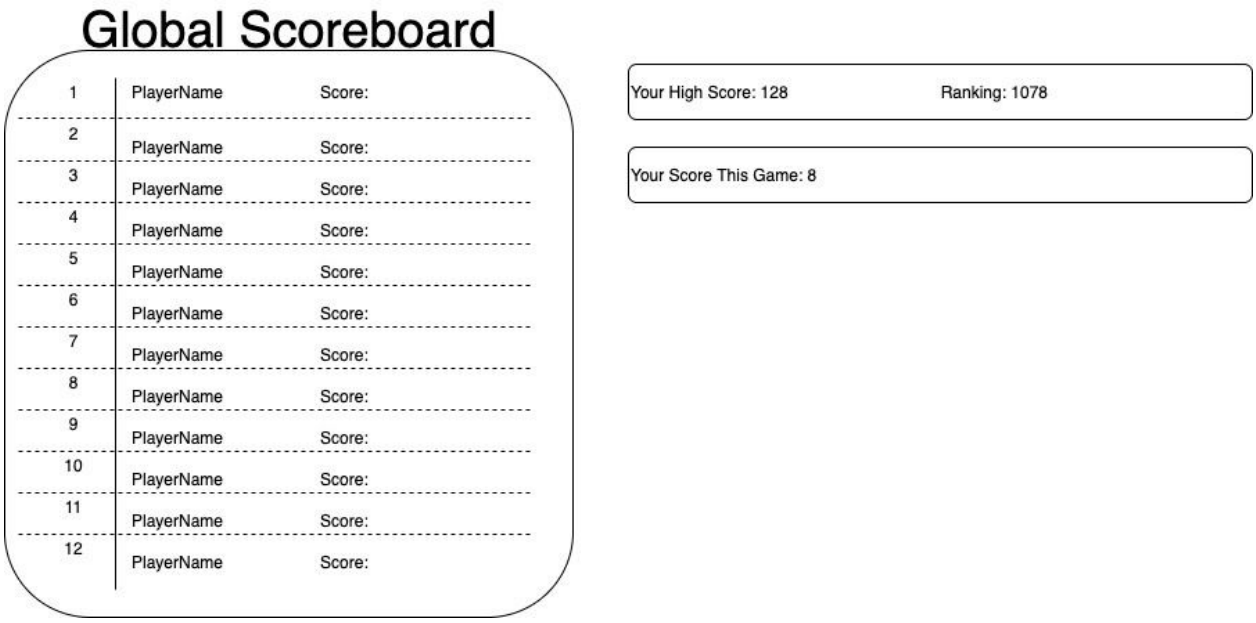
Register

Actual Game Play:



(This is the player, and each player has this view, but can see up to two other player's paddles at any time)

Global Scoreboard:

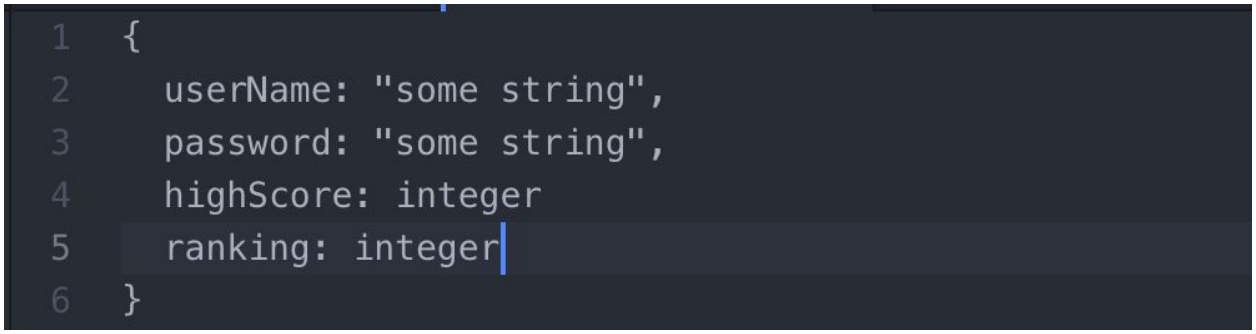


Web Service Design:

Our group is using Phaser, MongoDB, and NodeJS. Phaser is a physics engine that we use to create our game. Here we store images, the game logic and mechanics themselves, and player scores. Phaser also helps us by using built in classes that make our game run smoothly. For example, we are able to use calls that help us signal if an object has crashed or is out of bounds. This helps us with restarting the game and keeping the score up to date. We use NodeJS with the Express API to serve as the integration layer between Phaser (front-end/client) and MongoDB (backend/database). MongoDB stores the information for each player of the userName, password, highScore, and ranking for that particular player. We interface between the integration layer and the backend (MongoDB) with the Mongoose API. In the game we use GET and POST requests for the interaction between the client (using Phaser) and the server (using NodeJS). The data that is being requested is player info such as userName, password, highScore, and ranking. This info is being compared against in the login process and in the ranking for the global scoreboard. This info is being changed after each gameplay by updating highScore and ranking for a particular player. It is also being changed during registration by creating an object for a new player. This information is saved in MongoDB (our backend), with which we interact from the server using the mongoose API by calling the provided functions to update and access the database.

Database Design:

We are using MongoDB which uses NoSQL technology. The data is stored in JSON-like objects that are of this form for each player:



MongoDB is not a relational database. Instead it consists of collections of JSON-like objects, so the database consists of collections of these player objects with the data as shown above. The reasoning for the use of this DBMS is below.

Why we choose MongoDB instead of a relational database: The reasoning for the choice of a non-relational database is that the documentation online for using the phaser game engine was much more significant for MongoDB than relational databases. In fact, a lot of users with phaser.io seemed to struggle with the backend in connection with this game engine; therefore, we decided to go with the design of a database where we could maximize usable online resources should we get stuck. On top of that, given the data we are storing (username, password, and high scores), a relational database is not necessarily needed, because it is simply these three values that can be stored easily in objects (as MongoDB does: https://www.mongodb.com/what-is-mongodb). In terms of additive advantages, MongoDB also provides easy scalability (https://www.mongodb.com/scale/relational-vs-non-relational-database), which is ideal for this project.