BATTLESHIP DEVELOPMENT TASK

Design Document

M Mutahhar Bin Muzaffar
mutahharbinmuzaffar@gmail.com

Contents

1. Introduction:

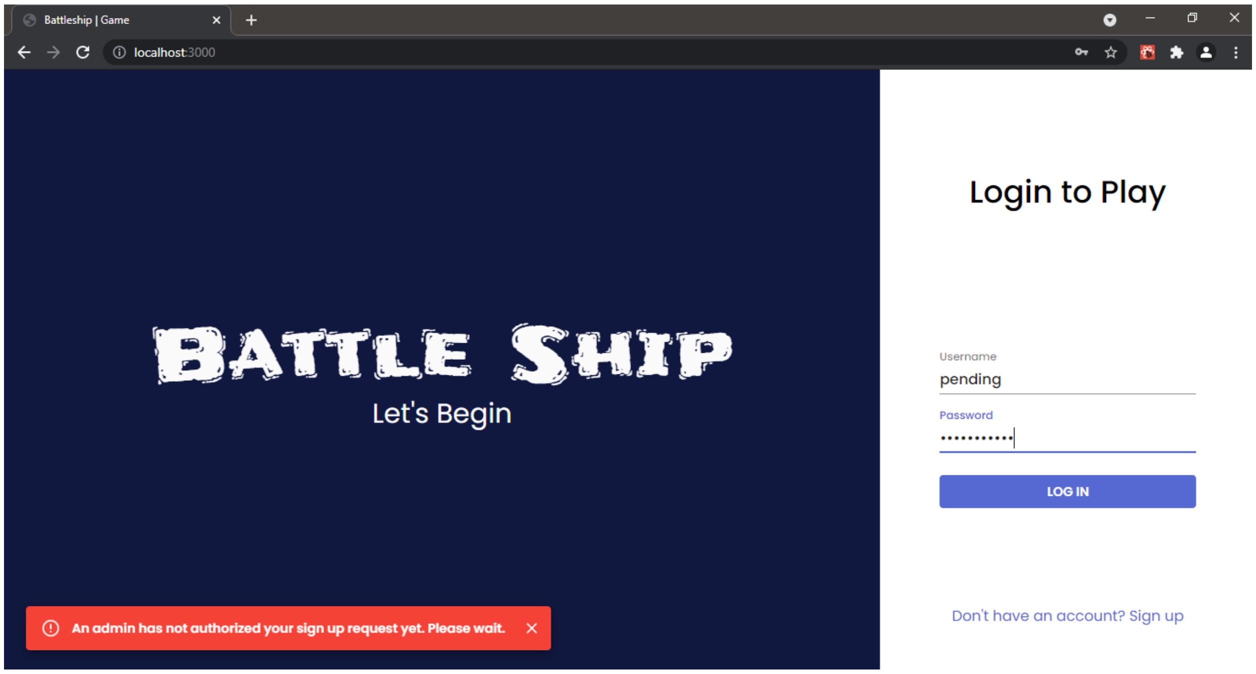
The task at hand was to develop a web application game called Battleship with an admin management system in it. The game uses a grid of 10x10 cells where user place all of his battle ships, usually 5, on the board prior to start the gaming with another player. Each ship consists of different number of cells it can occupy on the board. Upon placing all of his ships, the user begins the game and starts firing at his opponent's grid with the help of cell reference code such as A1, B1, etc. On successful hit, the user hits an enemy ship, and he misses it on an unsuccessful hit. Once the user has hit all of the cell's that are occupied by a ship on the opponent's board, he sinks his opponent's ship and get a point. The game ends once any one of the users has sunk all of his enemy's ships.

The admin management system is supposed to authorized user sign up requests and only after admin's approval the user must be allowed to play the game.

2. Overview:

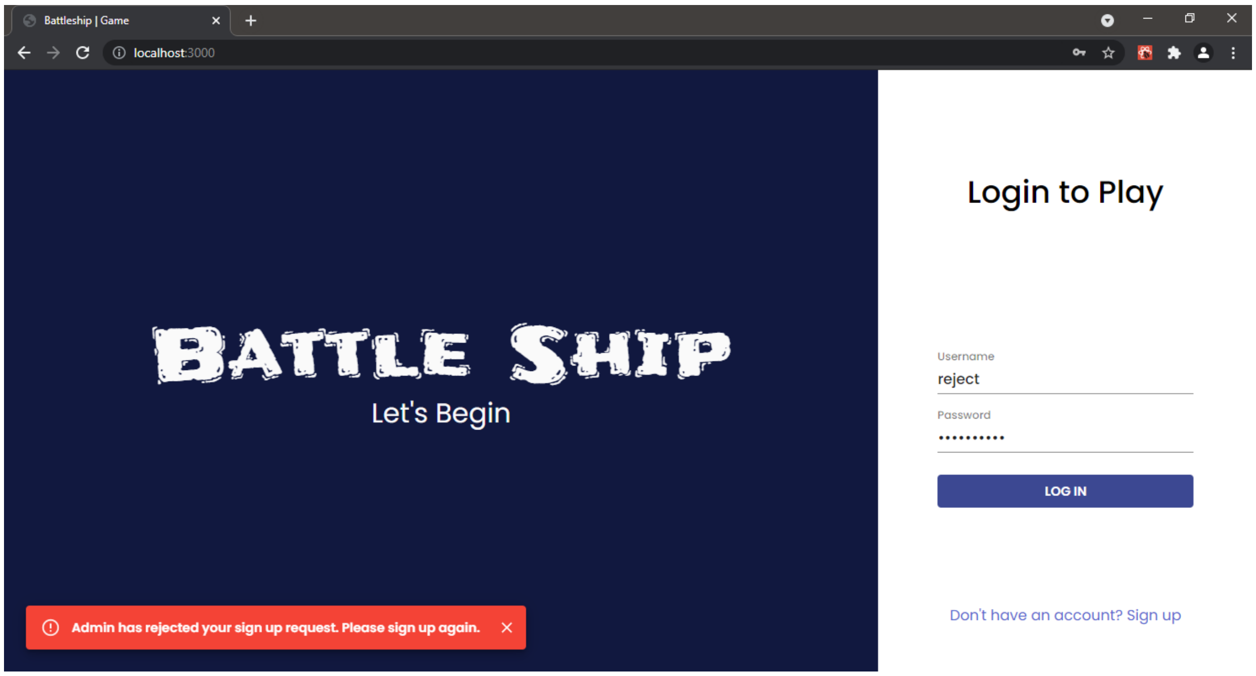
2.1. User Management System:

I have completed the user management system as required by the task. New user's cannot log in unless an admin has approved his sign up request. A user's account can consist of any one of the status such as 'approved', 'pending', and 'rejected'. Only if a user account has the status of approved, he can then proceed to play the game. Appropriate message is shown to the user if his account has the status of either 'pending' or 'rejected' as shown below:



An admin has not authorized your sign up request yet. Please wait.

Username: pending

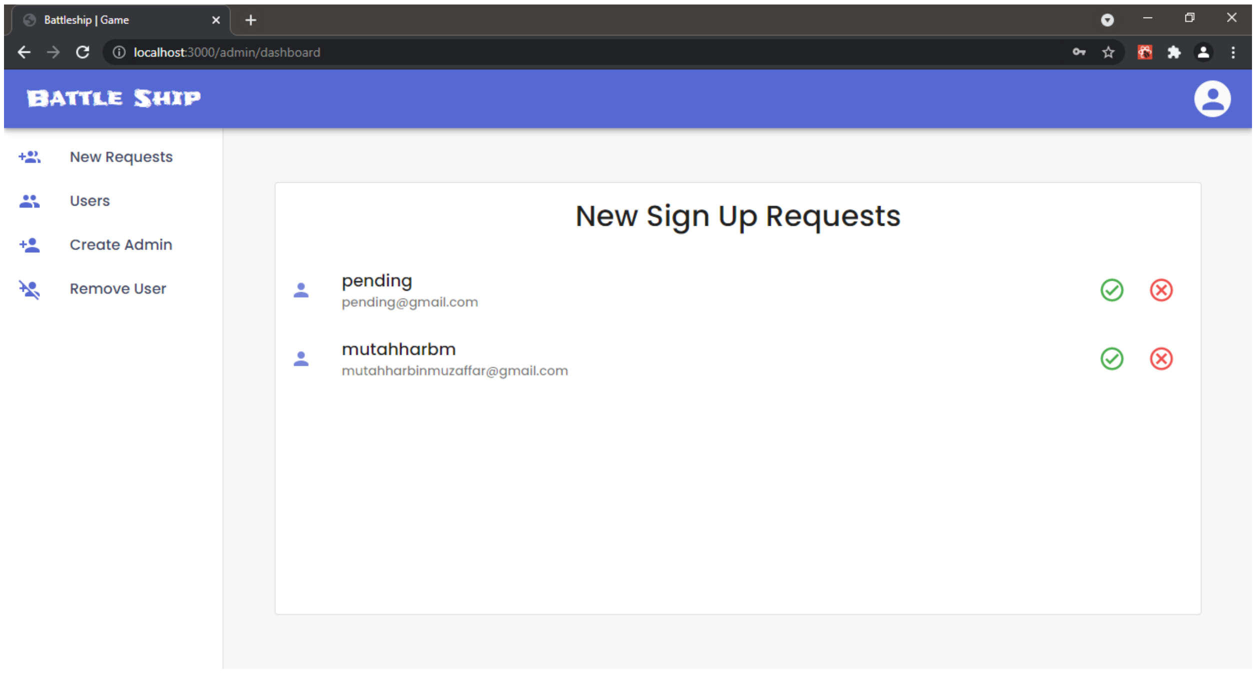


Admin has rejected your sign up request. Please sign up again.

Moreover, to add the more use of database and API, I added some additional features to the user management system. An admin dashboard consists of four screens as shown below:

2.1.1. New Requests:

This screen shows all the pending requests to the admin, including the previous and new ones with the status of 'pending' as shown below:

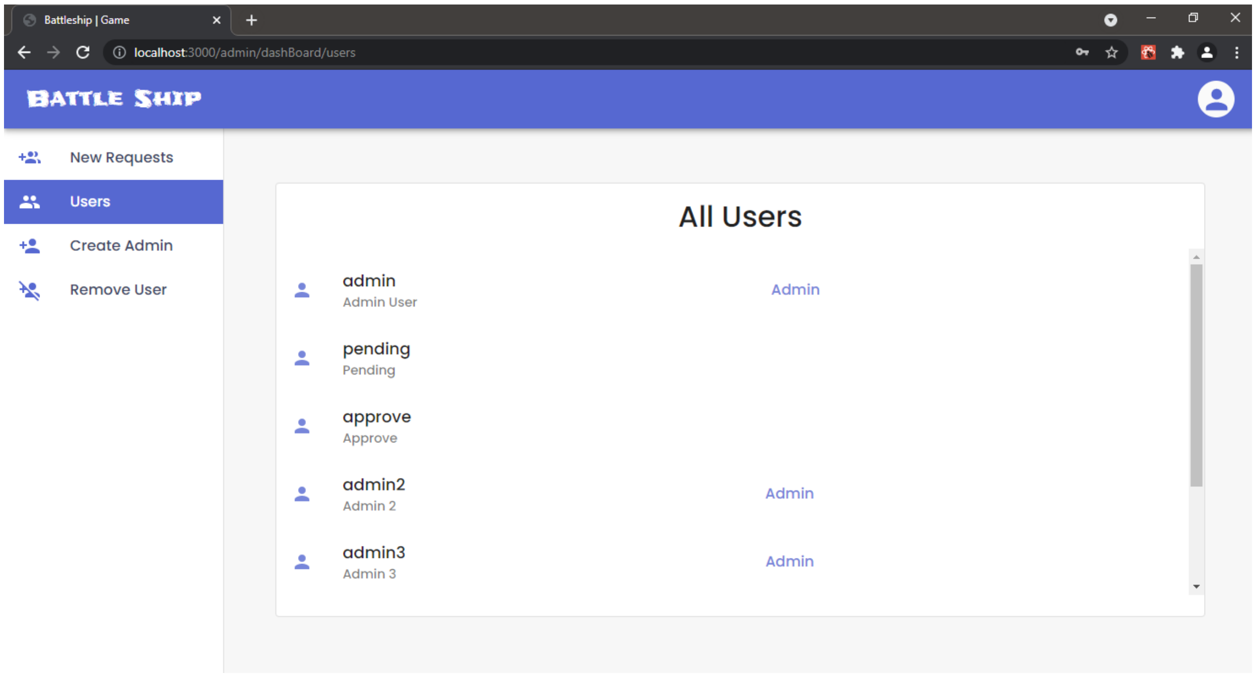


pending
pending@gmail.com

mutahharbm
mutahharbinmuzaffar@gmail.com

2.1.2. Users:

This screen shows all the users that are registered in the system having any status, including the admin users.

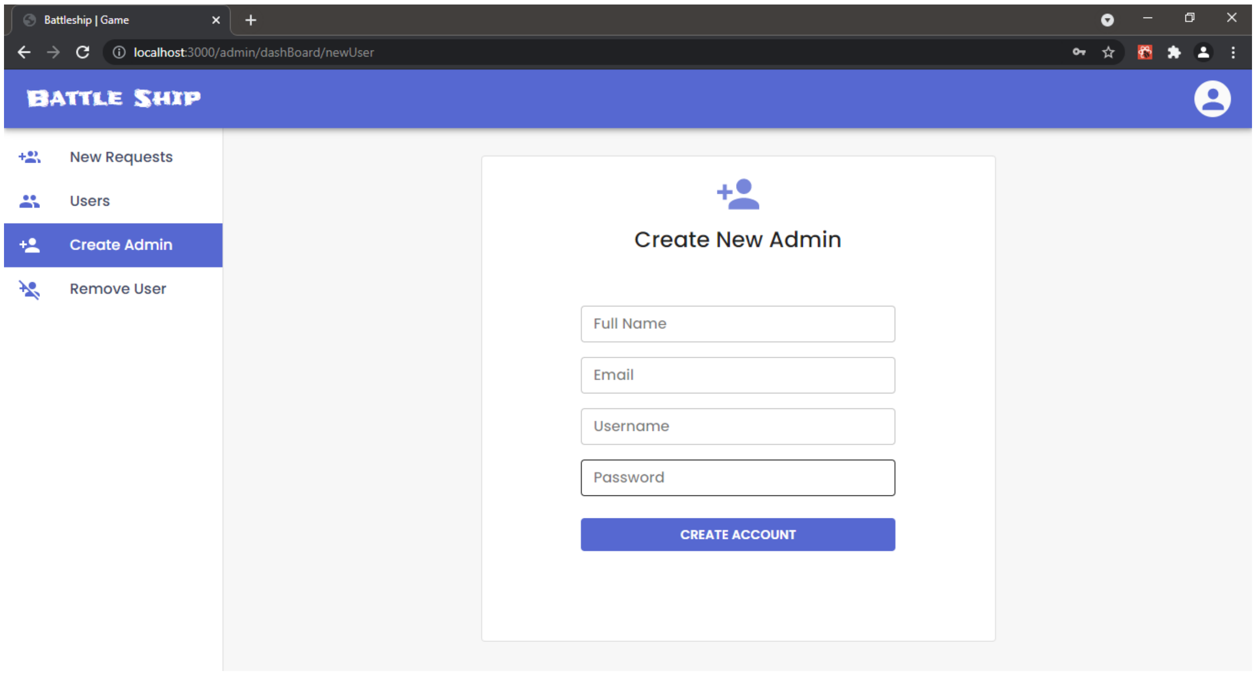


All Users

admin Admin User	Admin
pending Pending	
approve Approve	
admin2 Admin 2	Admin
admin3 Admin 3	Admin

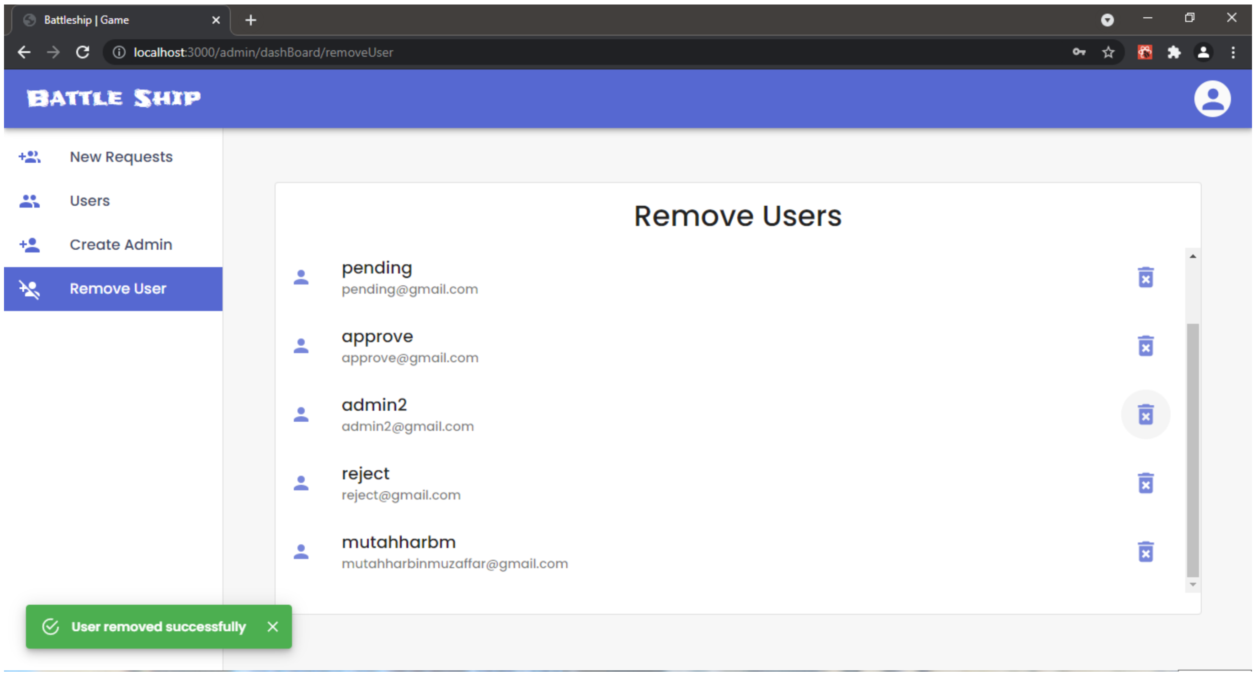
2.1.3. Create Admin:

Since an admin cannot sign up using a player's sign-up form, I have developed this additional screen to allow an existing admin to create an account for new admin.



2.1.4. Remove User:

This screen allows the user to remove and delete any existing user, including the admin users.



Remove Users

pending
pending@gmail.com

approve
approve@gmail.com

admin2
admin2@gmail.com

reject
reject@gmail.com

mutahharbm
mutahharbinmuzaffar@gmail.com

The user management system adheres to RESTful principals and communicates to the backend through appropriate endpoints. All the validation checks and JWT authentication has been implemented in this section and user cannot proceed to any backend endpoint/ frontend route without the authentication token. Private routes have also been implemented on the frontend.

2.2. Battleship Game:

I hadn't heard of this game prior to this task so I also had to learn about this game in detail before beginning development.

2.2.1. Rules:

The task required to involve at least three ships, so I added four ships overall. These ships, with their sizes, are 'carrier (5)', 'battleship (4)', 'cruiser(3)', and 'destroyer(2)'. The user is required to place the ship on the board prior to starting the game. The ships cannot overlap and cannot be placed outside the given constraints. Once all the ships have been placed, only then the user can start the game and being firing at the opponent's grid'. Once the user has hit all the cells that are occupied by the ship, that ship sinks and turns to black. Once all the ships have sunk, the game end. The first player to sink all of opponent's ships wins the game.

To place the ship on board, the user has to click on a ship from the harbor in order to select the ship and then hover over the board wherever he wants to place the selected ship. The user can also right click while hovering to rotate the orientation of the ship.

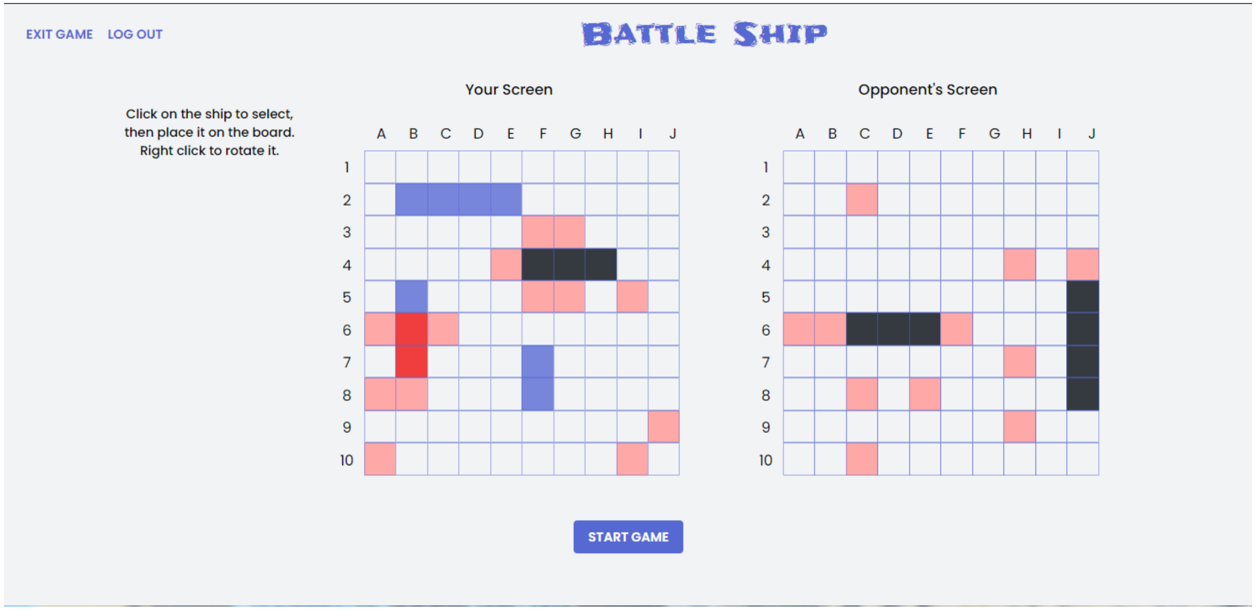
2.2.2. Color Coding:

The following color coding/ CSS classes are given to the board:

- Ship: blue
- Hit: red
- Miss: pink
- Forbidden: light pink
- Sink: Black

2.2.3. Single Player Version:

The single player version of the game involves computer as the second player and is up to mark and fully completed. I developed this system before developing the multiplayer version so that I could learn and test how the game works overall. Since I had to move to the multiplayer version, I didn't incorporate the single player version in the main application system, but all of its code is present on the **frontend** branch of my repository and can be demonstrated separately if required. The screenshots of the single player version are provided below:

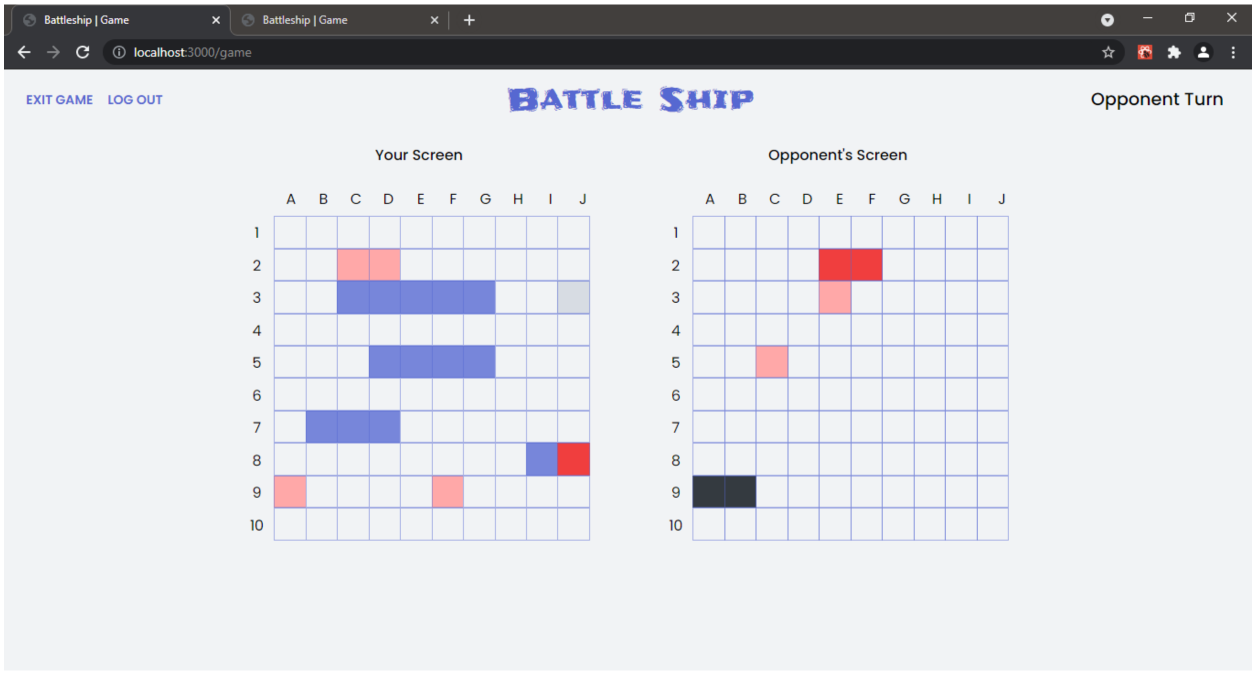


2.2.4. Multiplayer Version:

I have completed the implementation of the multiplayer portion of this game using socket.io. Currently the game only allows two connections simultaneously and shows a message that room is currently full. The game successfully receives the connection and connects two players to begin the game. The player that joins the room first is supposed as player 1 by default. The rules of the game are exactly the same as for multiplayer except that the player now has to wait after placing his ships, for another player to join the room.

The game also displays the message for whose turn it is and successfully transfer the control to that player. It also exchanges ships layouts prior to beginning so that the player's hits and misses can be assessed. The multiplayer version of the game is 99% ready with the only flaw that it does not register the first fire on the opponent's screen. All other fires, including hits , misses, and sinks are registered successfully.

The game also displays a game over message with an indication of whether the player or the opponent has won.



3. Technologies:

Prior to building this game, I had zero knowledge about the languages, libraries, frameworks to be used for the development of the given task so I spent the first two days learning as much as I could before starting the actual development. The technologies I have learned for this task includes:

- Node.js
- Mongoose

- Express
- Typescript
- Node and React with Typescript
- Web sockets and socket.io

I got to learn the first few technologies relatively easily but web sockets and socket.io really gave me a hard time because of my beginner's experience with web sockets.

3.1. Database:

The database used for this task is MongoDB. There wasn't much requirement for the database and there weren't any complex relations therefore, I decided to use the non-relational database and managing this was much easier and quicker which saved me a lot of time. A typical user document in the database has the following fields:

```
{"_id" :{"$oid":"6118caebd5e5225224dfb57d"},
"status":"approved",
"isAdmin":true,
"fullName":"Admin User",
"username":"admin",
"email":"admin@gmail.com",
"password":"$2a$12$3d/vLPMjDZRPPUw99A4we.mdwzqY4V2Y0PbAmDzGYv.nQFK7ACAJS",
"createdAt":{"$date":"2021-08-15T08:06:03.108Z"},
"updatedAt":{"$date":"2021-08-15T08:06:03.108Z"},
"__v":0}
```

I had planned to store the result of the multiplayer game as well but since I ran out of time, I could not implement this feature.

4. Design Decisions:

The design methodology was a hybrid one. Meaning that I used a mixture of Object Oriented, Functional, and Modular approach.

For example, I created object for modals while using mongoose, inherited and extended different interfaces in typescript, split the code/ util functions in different files and used functional programming approach for the socket.io part.

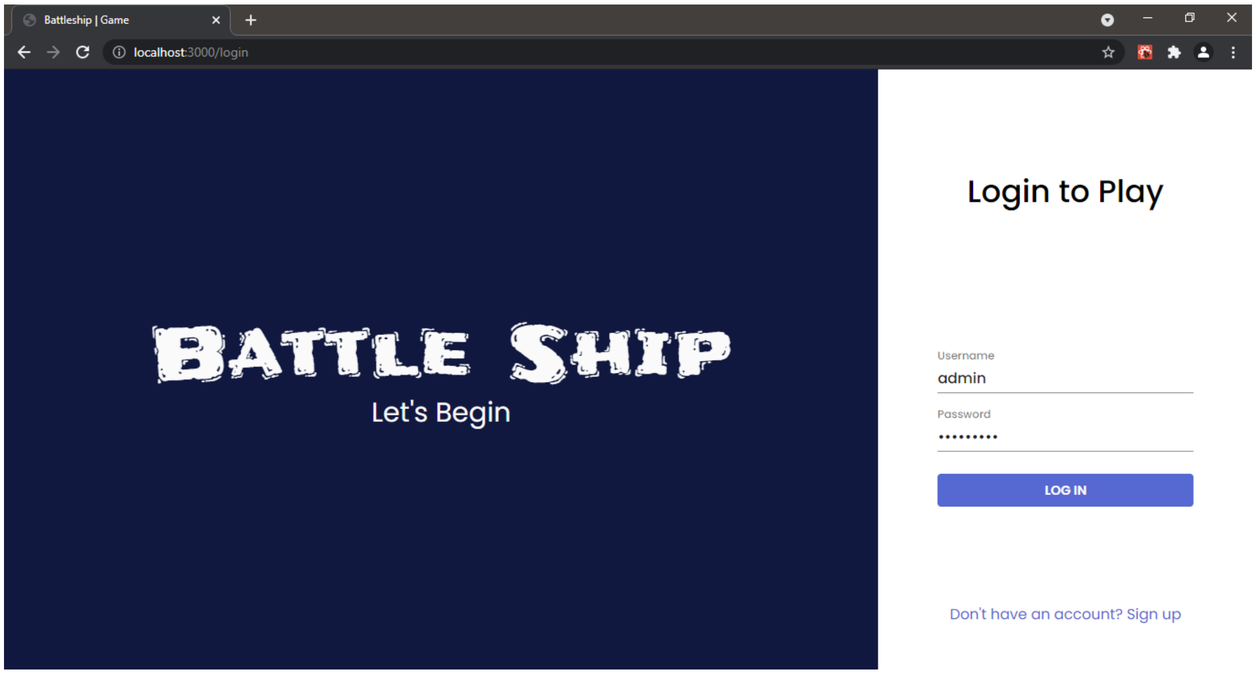
The process methodology was incremental as I developed the project in bits and pieces, tested each step before moving on to the next one and eventually integrated all of the steps.

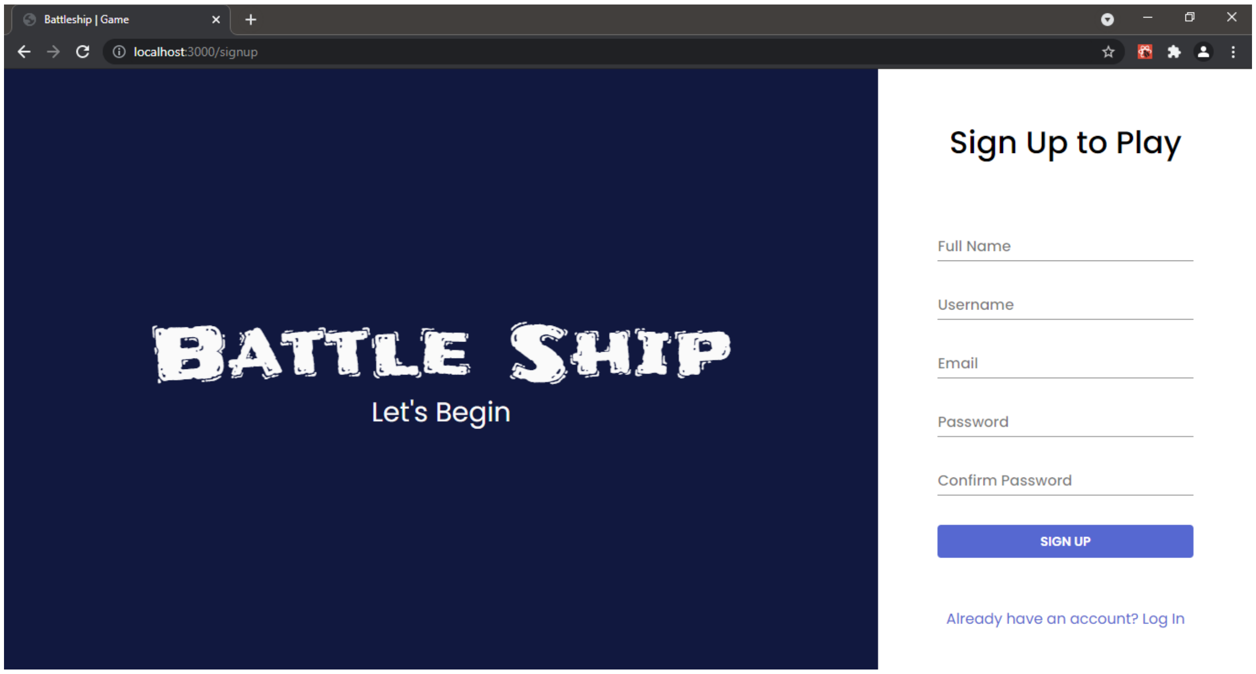
The backend of the system is entirely developed using RESTful API and I have used all the RESTful methods such as GET, POST, PUT, and DELETE

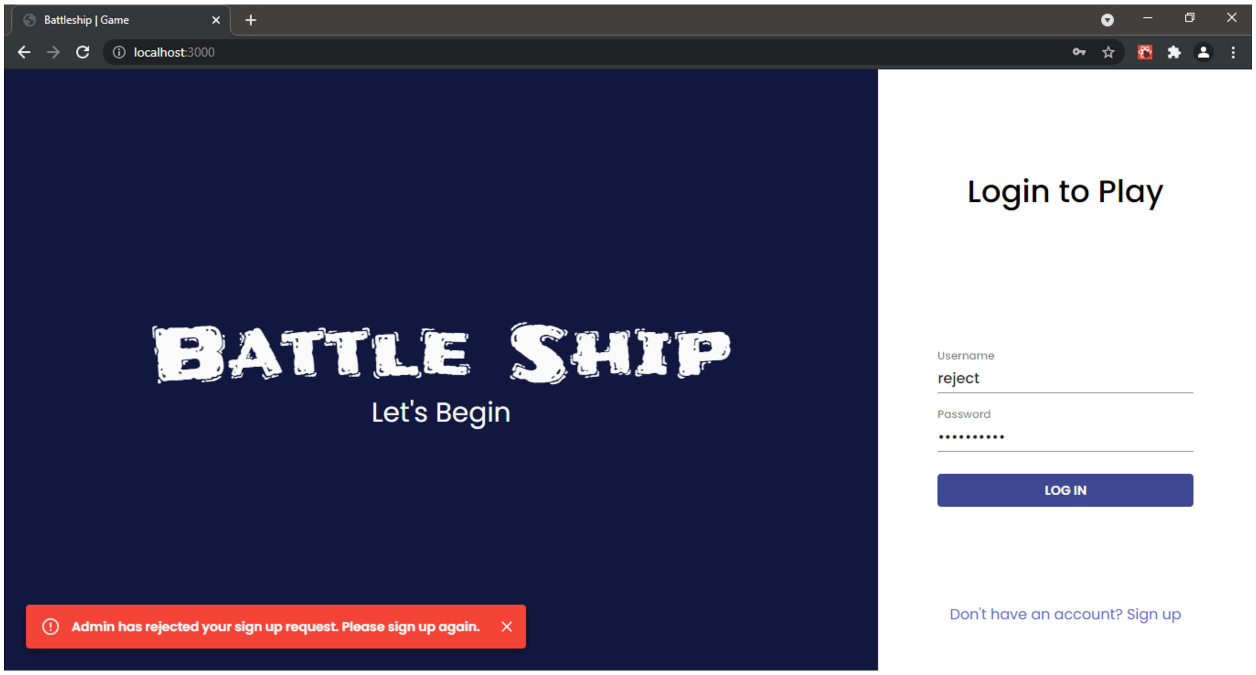
The socket.io part of the system is not developed using REST API as it supports its own protocol of client server connections thorough sockets and ports.

5. Screens:

The section contains all the version of screens of the web application. All of these screenshots will be provided in the **main** branch of the repository along with this document.

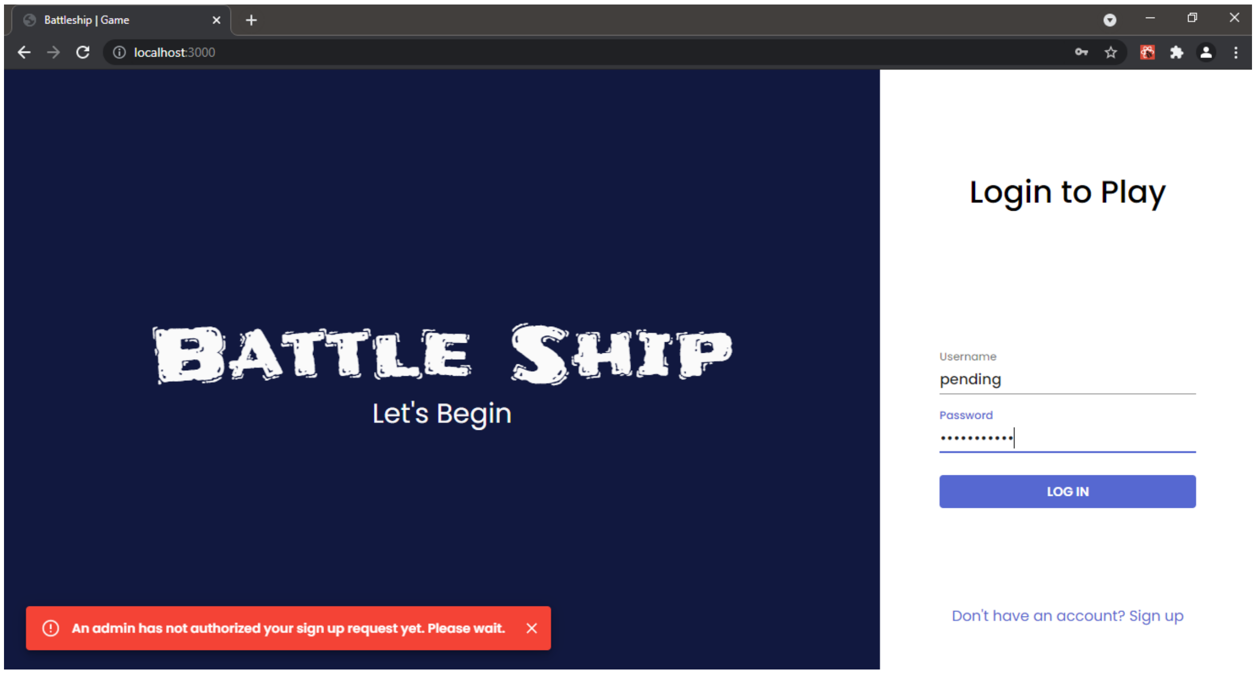




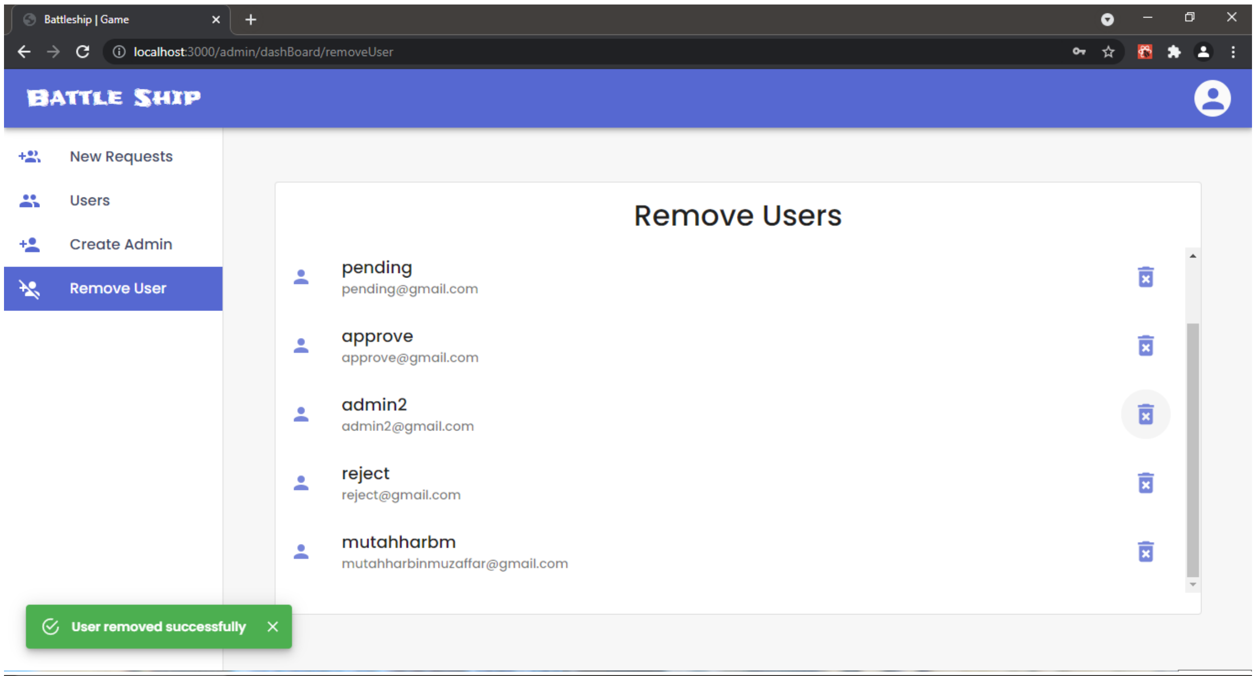


Username
reject
Password
••••••••••

Admin has rejected your sign up request. Please sign up again.



An admin has not authorized your sign up request yet. Please wait.



Battleship | Game

localhost:3000/admin/dashBoard/removeUser

BATTLE SHIP

- New Requests
- Users
- Create Admin
- Remove User

Remove Users

pending
pending@gmail.com

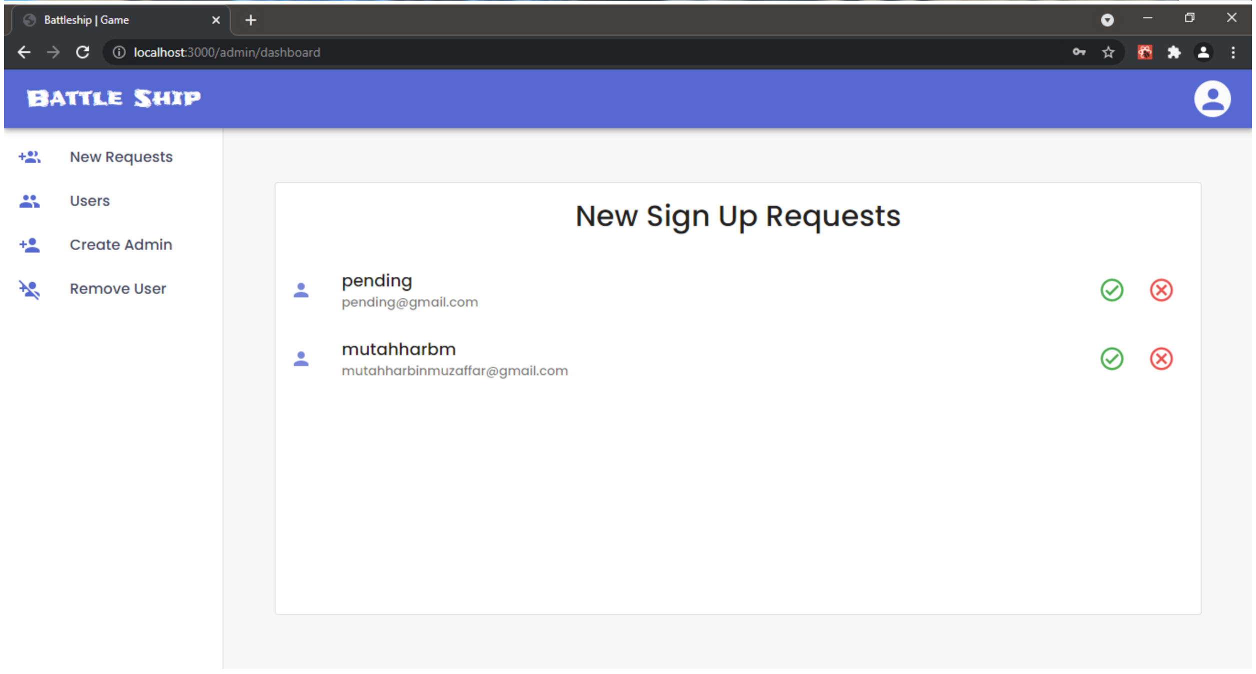
approve
approve@gmail.com

admin2
admin2@gmail.com

reject
reject@gmail.com

mutahharbm
mutahharbinmuzaffar@gmail.com

User removed successfully



Battleship | Game

localhost:3000/admin/dashboard

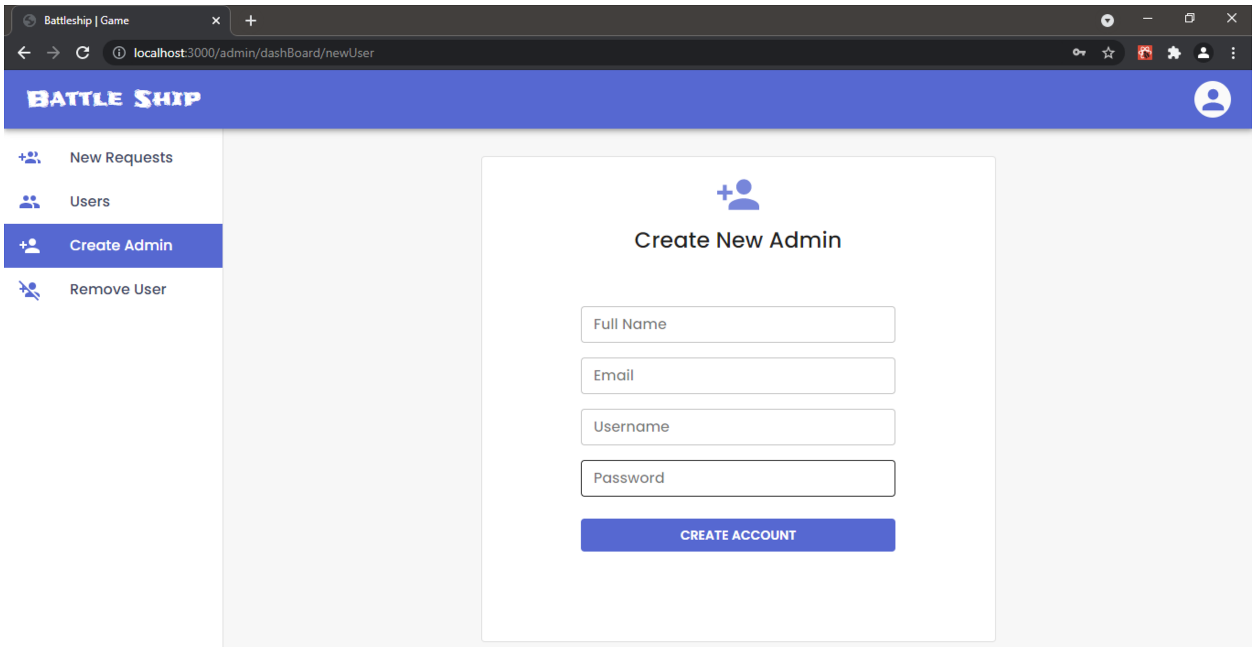
BATTLE SHIP

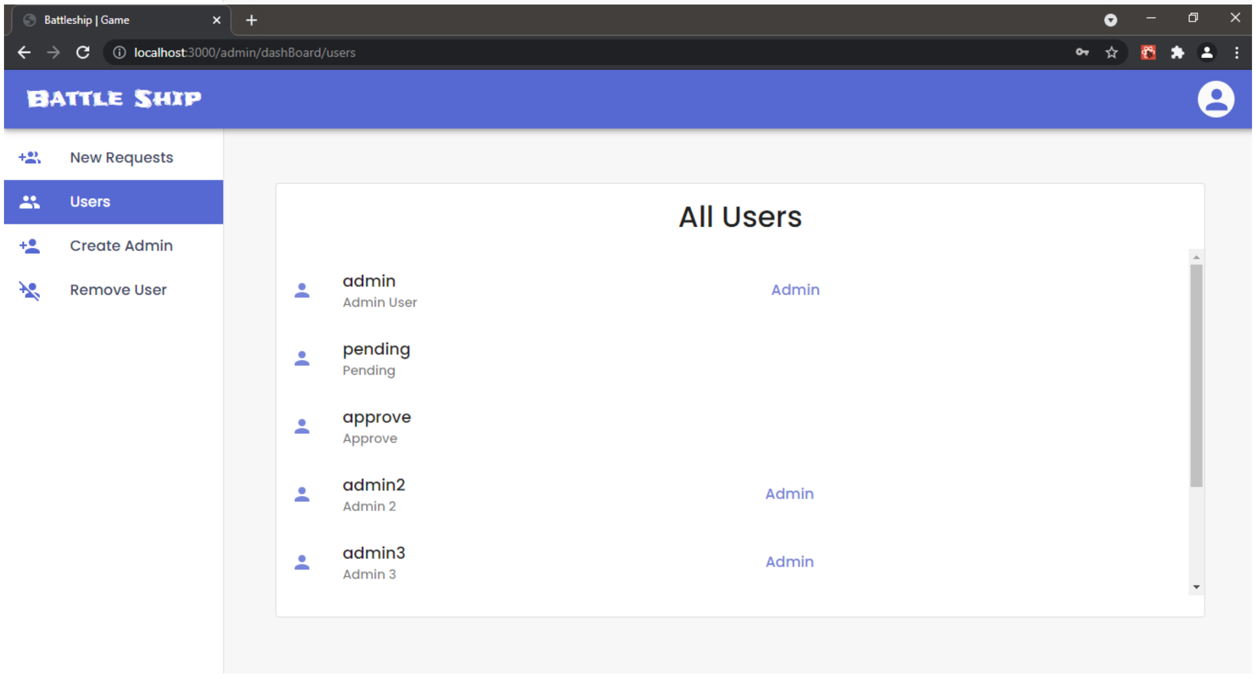
- New Requests
- Users
- Create Admin
- Remove User

New Sign Up Requests

pending
pending@gmail.com

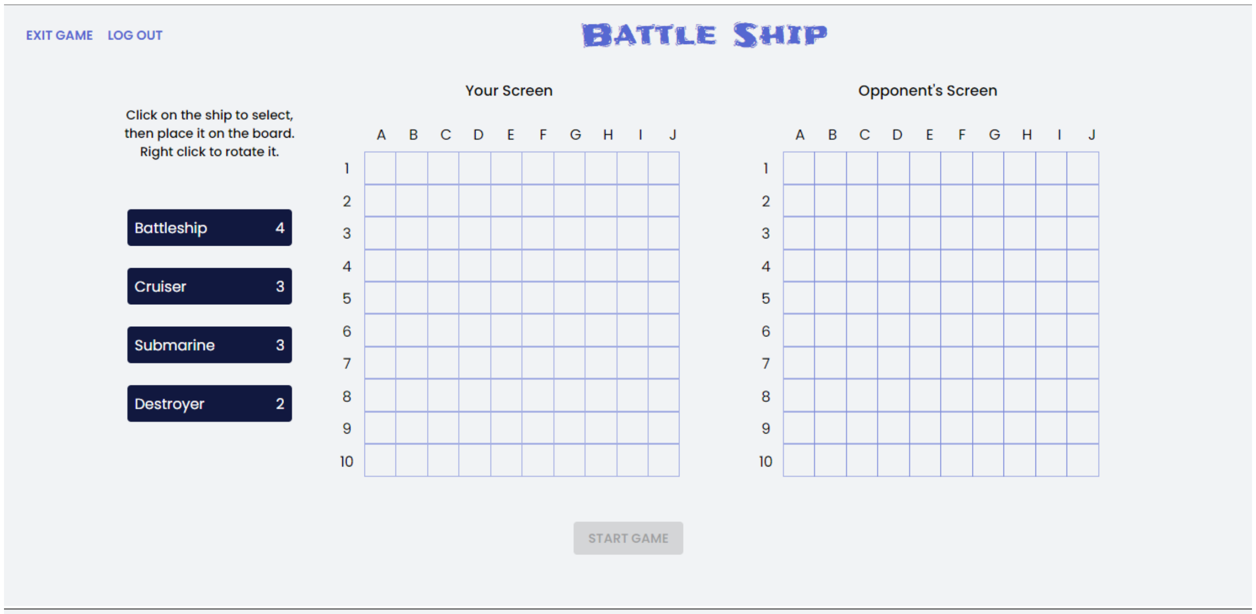
mutahharbm
mutahharbinmuzaffar@gmail.com

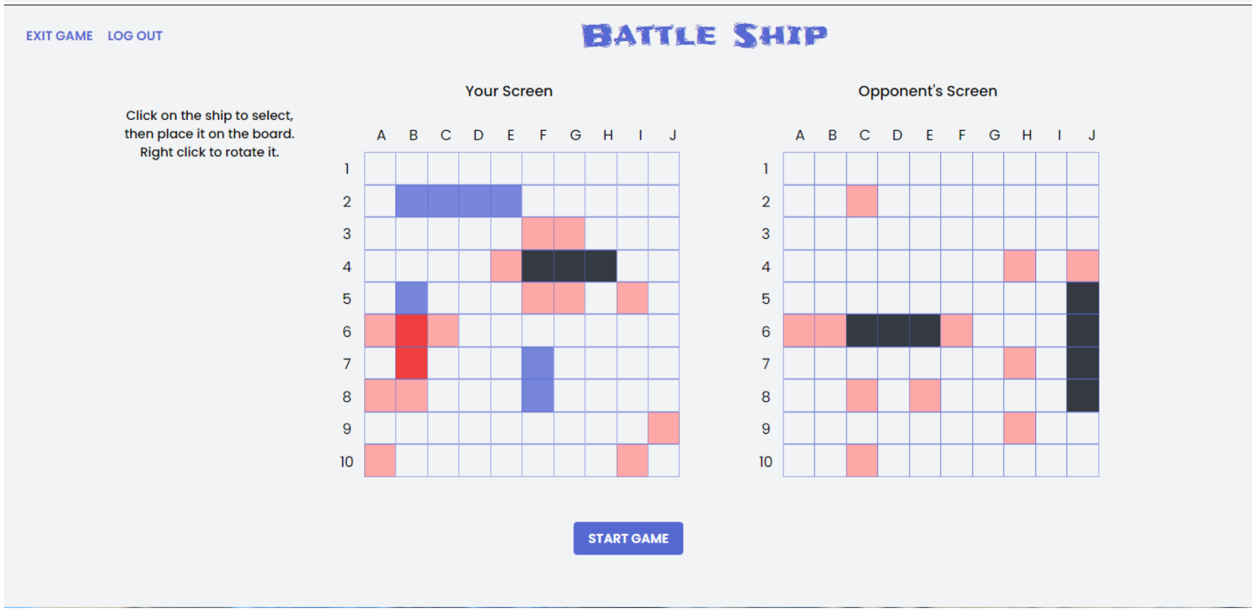


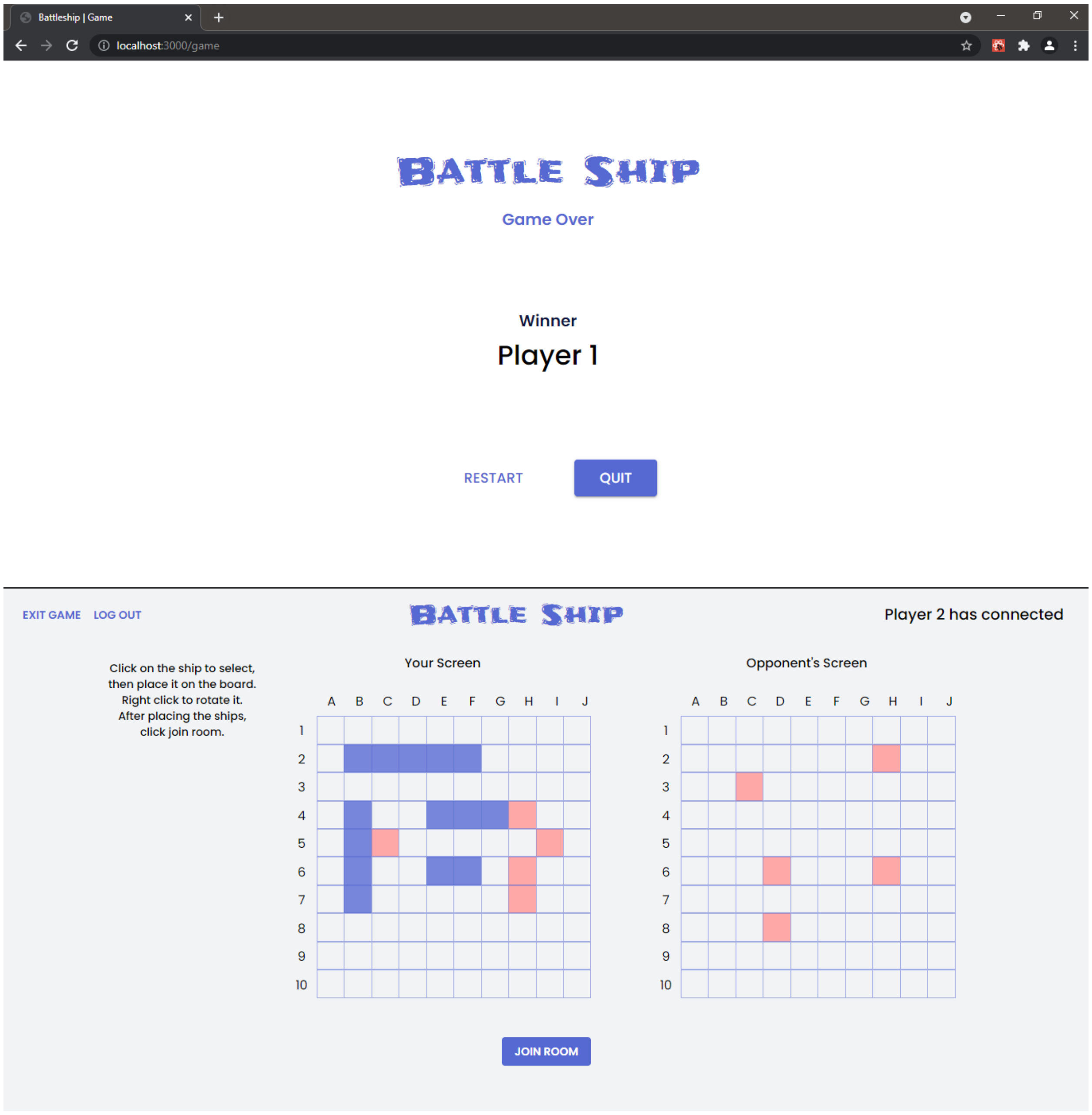


All Users

admin Admin User	Admin
pending Pending	
approve Approve	
admin2 Admin 2	Admin
admin3 Admin 3	Admin


EXIT GAME
LOG OUT
BATTLE SHIP
Your Screen
Opponent's Screen
Click on the ship to select, then place it on the board. Right click to rotate it.
Battleship 4
Cruiser 3
Submarine 3
Destroyer 2
A B C D E F G H I J
1 2 3 4 5 6 7 8 9 10
START GAME


EXIT GAME
LOG OUT
BATTLE SHIP
Your Screen
Opponent's Screen
Click on the ship to select, then place it on the board. Right click to rotate it.
A B C D E F G H I J
1 2 3 4 5 6 7 8 9 10
START GAME



Game Over

Winner

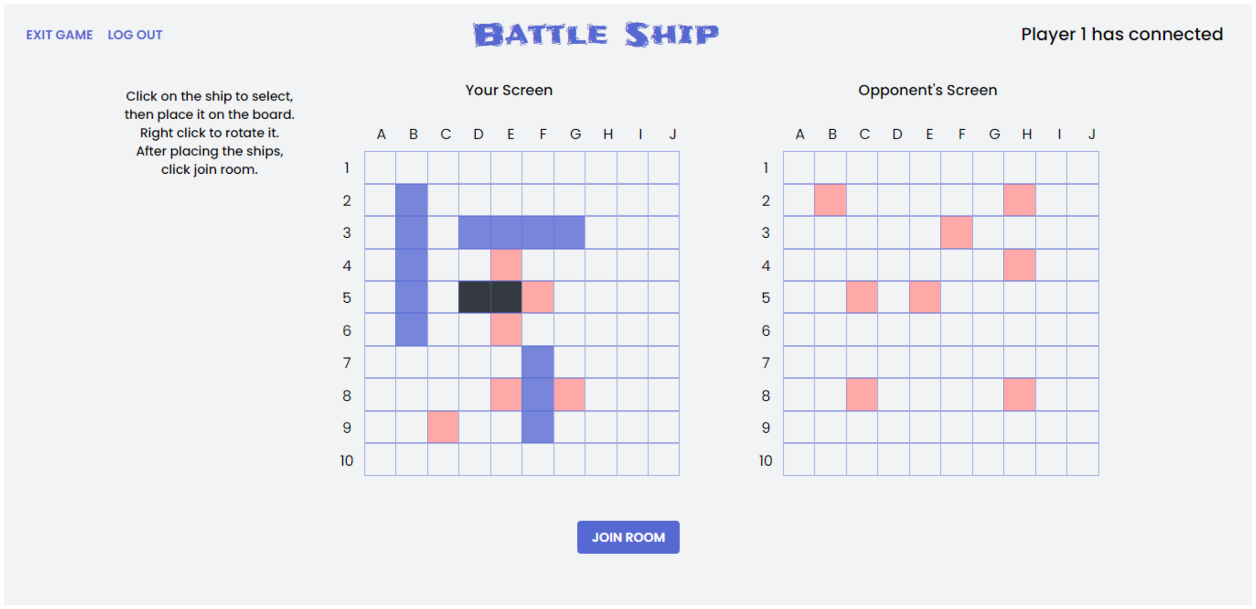
Player 1

Player 2 has connected

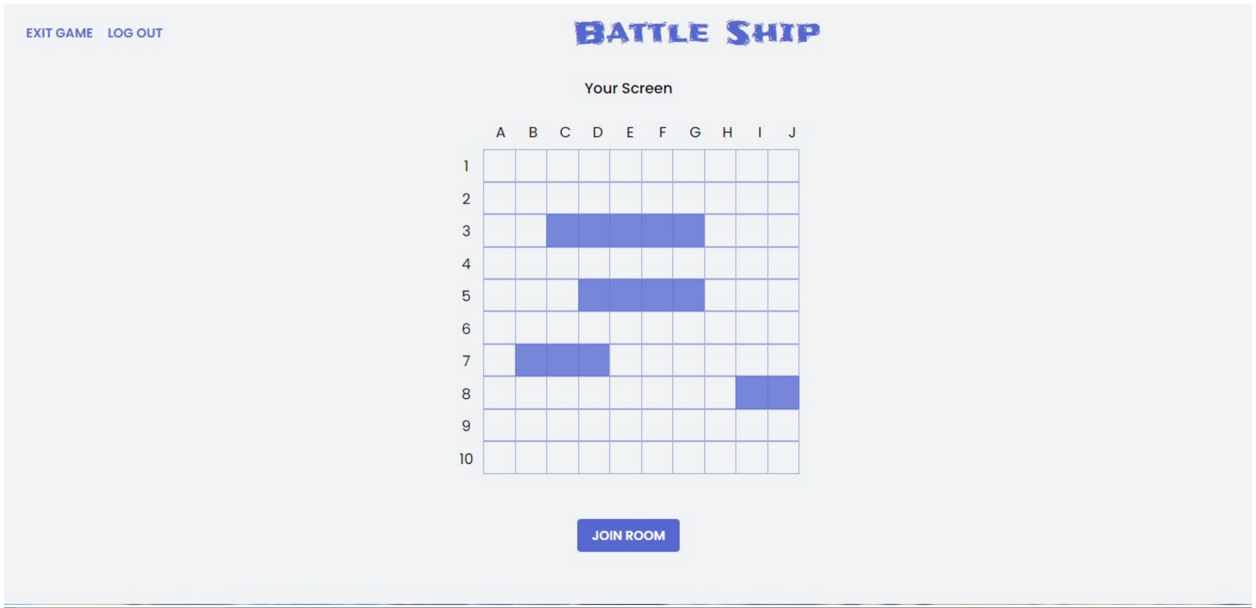
Click on the ship to select, then place it on the board. Right click to rotate it. After placing the ships, click join room.

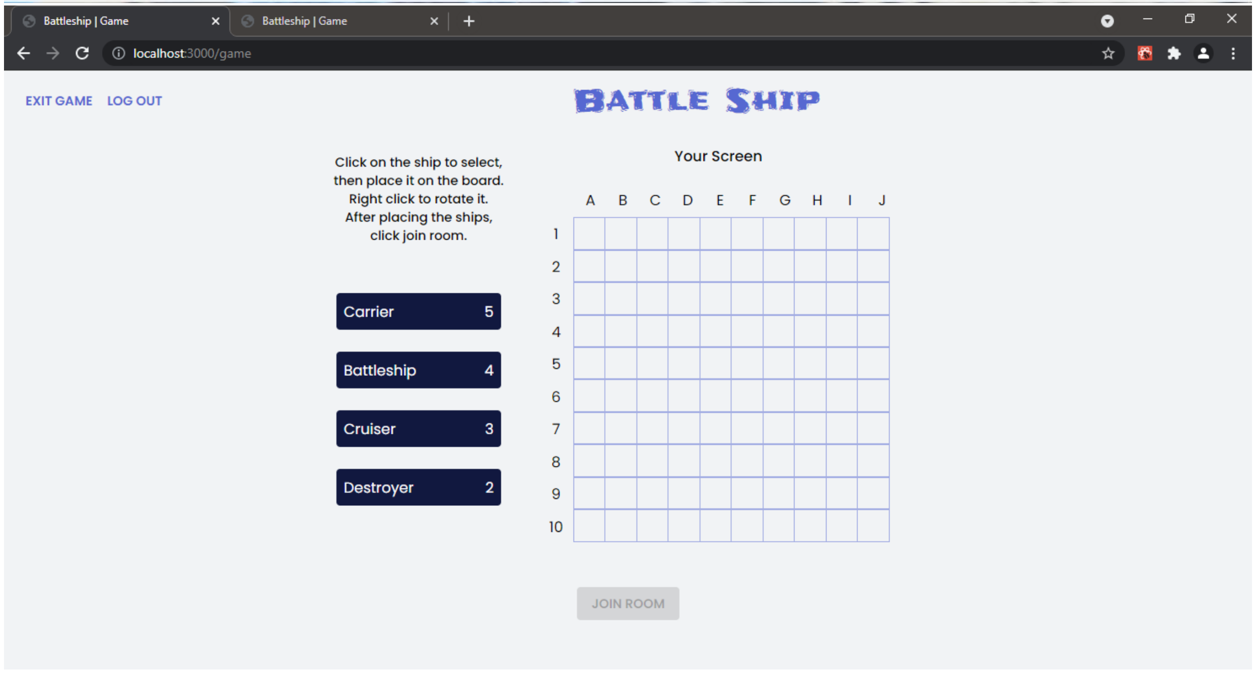
Your Screen

Opponent's Screen

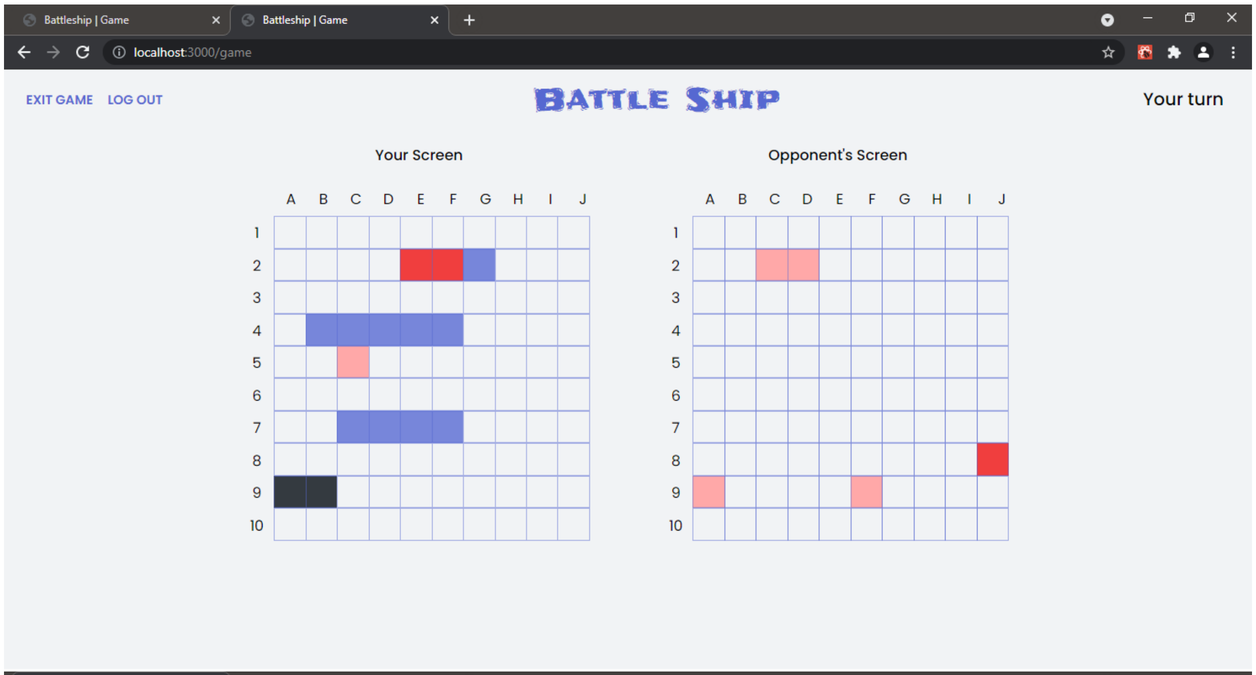


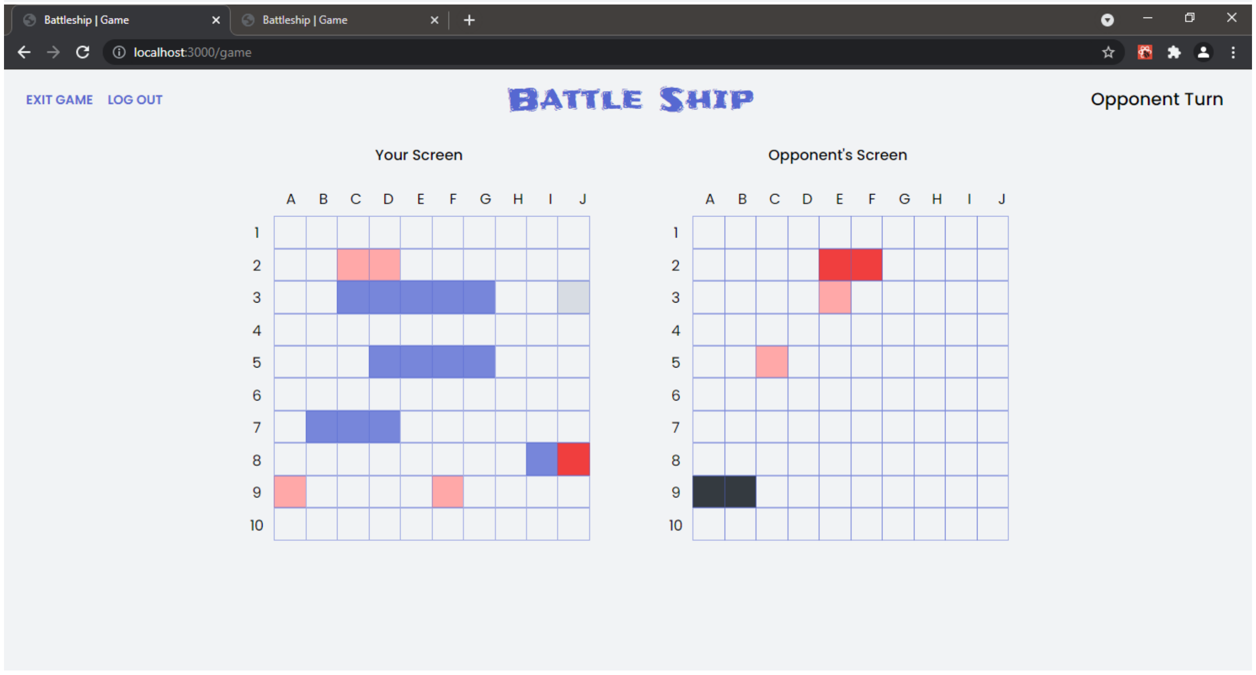
Click on the ship to select, then place it on the board. Right click to rotate it. After placing the ships, click join room.





Click on the ship to select, then place it on the board. Right click to rotate it. After placing the ships, click join room.





6. Limitations:

There are not much or major vulnerabilities, flaws, or limitations to this web application except a few, such as:

- Multiplayer game only supports two users simultaneously.
- The outcome of the game is not stored in the database yet.
- Token is stored in the browser and is not removed if it expired.
- A single user can open multiple instances of the game using the same token.
- Player's usernames are not displayed in the game.

7. Notes on Optional Requirements:

- Unit testing: Non documented and manual unit tests were perform throughout the development process so that each component could be tested for correct and incorrect inputs and outputs.
- API Documentation: I could not implement this due to lack of familiarity with OPEN API etc.
- CI/CD: Again, not familiar with this yet.
- Docker: I know the concept of docker but does not have experience in it and I did not have the time to learn this as well.

8. Github Repository:

The Github repository consists of four branches as explained below:

- Main: This branch consists of all the document material including, screenshots, README.md, and this design document.
- Frontend: This branch consists of all the frontend code for admin and single player game
- Frontend-multiplayer: This branch consists of all the code for admin and multiplayer game
- Backend: This branch contains all the code for the backend development.

9. Conclusion:

On an ending note, I would like to thank X-Grid for providing me the opportunity to develop such an exciting project. Between learning a lot of new things and debugging some things, it was an challenging experience, and I enjoyed every moment of it.