Indian Institute of Technology,
Bombay

CS 251 Course Project

Git Gud

Battleship

Author:
Arjit Jain
Aditya Vavre
Devki Nandan Malav

November 24, 2018

Contents

1 Introduction

1.1 Overview and Motivation

This report discusses the implementation of the classic arcade game, "Battleship". It is a part of the CS251 Course Group Project. We wanted to explore the Full Stack Development regime and combine modules /frameworks like NodeJS, Express, React, SocketIO with Relational Table Databases like MySQL to create simple yet powerful applications.

1.2 Objective

The main objective was to get a sufficient exposure with Node and Express frameworks. Since socket programming was a completely new paradigm for us, we experimented a lot with sockets. So naturally we were able to enhance our backend programming skills a lot. As for the Frontend part, we experimented with React and HTML5.

2 Features

2.1 Signup and Login

- After browsing to the homepage, the user is shown a login/signup screen.
- Each user is given a username unique to the user.
- The hash of the password is stored in the database instead of the password itself for better security against attacks.
- At each login attempt, the stored hash is compared with the hash of the entered password.
- Bcrypt library was used to achieve this.
- We are also storing session info using sessionStorage, so that even if the user is redirected to some other url or refreshes the tab, the session is still saved

2.2 Pairing

- A Dynamic list of online users is maintained and is visible to all users currently logged in.
- This list is clickable, meaning that each user in this list of online users can be sent an invitation for battle.
- The destined player can choose to accept or deny the request.
- The way we have implemented the requests is such that only the Latest request is visible to the user. This is to avoid dangling requests since players leave the lobby as soon as a battle is paired.

2.3 Placing

- After accepting the request, both the involved users are taken to a url which basically renders the game
- They are shown two grids where they can place their battleships. Placing is done in such a way that it is ensured no two battleships with overlap.
- The players can choose to rotate the battleships as long as it fits the constraints.
- The player who places the ships first gets the first turn in the game.

2.4 Playing

- Playing is implemented in such a way that it is consistent with the rules of classic battleship
- If any part of the ship is sunk, it is visible to both the users.
- Attack on cells already attacked before (sunk or not) are not allowed.
- For each turn, a time limit of 5 seconds is implemented for the players, and the player exceeding the limit loses the game.

3 Technical Implementation Details

- The back-end heavily depends on the use of Sockets. We have used two namespaces in **app.js**, one for handling all the login requests and the other for handling all the dynamic game requests.

- Initially the game was implemented using ReactJS that supported the use of L and T shaped battleships with rotations. But our approach was wrong as we integrated the logic of the game in the frontend itself(React) and hence we failed to build upon the logic, i.e it was difficult to integrate the use of Sockets into this.

- Finally we decided to implement a basic game in JavaScript(Backend) and HTML5(fronted) with the use of Sockets

- The user data is stored in a MySQL database table. The server (app.js) interacts with the MySQL server to handle all the client requests.

- We have used MySQL as our database management system because our data is very well structured. Moreover, MySQL is well-recognized for its high performance, flexibility, reliable data protection, high availability, and management ease.

- We have used socket.io to implement all the bi-directional communications because its compact and easy to use and serves our purpose i.e. realtime, bi-directional communication between web clients and servers.

- Reason for choosing NodeJS was Firsty, NodeJS is completely Asynchronous so it makes the whole process blazingly fast. Also frontend integrations are relatively easy on Node. As for the question of why not Django? NodeJS community is much bigger and active than Django so almost any "issue" will probably have a solution on some forum. Also, since we wanted a deeper insight on basic event driven functions like login/signup we chose Node where we would have to implement each and every thing over Django where all we need to do is import the "right" templates.

4 Contribution

5 References

- http://dwcares.com/2015/10/21/realchess/
- https://stackoverflow.com
- https://socket.io/get-started/chat
- https://github.com/brentvale
- https://www.abeautifulsite.net/hashing-passwords-with-nodejs-and-bcrypt