Report for FIT3140 Assignment 1: Online Tic Tac Toe

John Goh Rengwu, 27150437, jrgoh1@student.monash.edu

Introduction

This assignment requires us to develop a tic-tac-toe game app, in such that two or more clients can play from browsers and interact in real-time via a server. It is highly recommended to use the socket.io library to enable real-time interactions between the clients. There are a number of requirements that the app is expected to meet, including real-time synchronization of game updates between clients playing the same game, automated player switching between moves, checking and determining the winner, notifying the player of mistakes, and resetting the game from either player at any time. The implementing of these requirements will be discussed later on further sections.

Background, Tools, and Technique

The goal is to create a single-page web application which connects to a server via a socket.io connection. The obvious choice for this would be a node.js server which runs alongside with npm. I chose to develop this application in React.js because it is easy to create well-defined components and cleanly manage the app's centralized state tree. Furthermore, I do not need to rely on jQuery to update DOM elements as React's component re-rendering method upon changing the state tree is almost automatic and pain-free.

To start off, after initializing my npm project, 'socket.io' and 'express' packages are installed and then created 'server.js' on my project root folder. Within this root folder, I created another folder called 'socket-client' and initialized another npm project in that folder, which the client-side code will reside. The directory structure will look like this:

```
react-socket-app
        > node-modules
        package.json
        server.js
        > socket-client
                > node-modules
                > src
                > public
                package.json
```

The reason for the splitting of two separate codebases is to have a clean segregation between server instance and client instances. Although this would eventually require us to run two separate terminal instances of the server and the client respectively as shown below:

```
node server.js              // run server instance
cd socket-client
npm start                   // compile client code and run client instance
```

This is solved later onwards by using the '`concurrently`' npm library, which runs multiple command instances in parallel. Then, in the client-side folder, the create-react-app boilerplate is installed, which is basically a package of everything needed to start developing the app in React. As of now, server-side code will reside in 'server.js' on the root project folder, while client-side code will reside in '`/socket-client/src/`'.

On the server-side, it is pretty straight forward. When the server is run, an express http server is started and will listen on localhost:4001. A socket.io instance is also created to handle oncoming event emits from the client and vice versa.

On the client-side, the HTML file (in 'public' folder) has nothing in its 'body' tag. When the code is compiled, the body will be populated by React components in the entry point, which is in 'socket-client/src/index.js'. Here, 'react-router' library is used to route URL paths to show different components. By default, the root path '/' is mapped to the `<App />` component, which is the component with TicTacToe. All of the client-side TicTacToe code is located in './socket-client/src/App/index.js' which is also accompanied by a CSS file.

Design, Components, and Features of the app

In terms of the app design, the requirements to meet are simple: display a Tic-tac-toe board with clickable cell grids, display the game's status and messages, and making sure clients who are in the same room have their boards synchronized.

Thankfully, using the React framework and libraries such as Bootstrap and Reactstrap, a lot of effort is saved on component design and styling. The landing page of the app is the Tictactoe game itself. All components in the page are wrapped with a page-wrapper div container which gives it 100% viewport height, and then all other components are wrapped with a bootstrap .container class, which provides responsiveness and mobile-support to this Tictactoe app.

I applied mobile-first design principles during the design process: the Tictactoe container component itself has a lean 'card' design which can accommodate most modern mobile screen-sizes. The status bar on top (reactstrap Alert component) clearly shows the current status of the game. The game matrix component is a simple fixed-width (300px) square div container. Using CSS grid, the button components (from reactstrap) are aligned to the grid container with equal fractions of height and width. Below the game matrix, a button group of two buttons are placed: the 'Reset game' button and the 'Taunt opponent' button. Either player can click on these buttons at any time of the game to either reset the board state or taunt the opposing player with a childish insult.

In order to present alert messages to the player, a toast container component (react-toastify) is used to show these messages. For example, if the player clicks on a cell that is already taken, a toast will appear on the bottom-right side of the screen notifying the player that the move was invalid. This toast container is used to show various messages and validation messages and is mobile-friendly too. The previously mentioned 'Taunt opponent' also shows a red toast message on the upper-left corner upon receiving an incoming taunt from the opponent.

At the bottom most of the page, a simple container displays miscellaneous info to the user, such as the room number, the player's sign, and the opponent's id.

No jQuery is used for DOM manipulation and updates, as React automatically re-renders the component upon changing the state. When the server emits an event, for example, updating the board state, the client only needs to change the global app state. React will detect a change in the state tree and re-renders the relevant component accordingly.

The server is able to handle multiple games running concurrently thanks to socket.io's room feature. Each player will be automatically assigned a room upon connection and the game will automatically start when two players are in a room. When a player disconnects, the room will have its board state reset and the remaining player will continue to wait for another player to connect.

Extensibility

Using npm packages and React components, this app is fully extensible and features can be easily added. This app can be extended into a full-fledged React app with the Tictactoe component as just one of the route endpoints. As seen in the client's './src/index.js' file, react-router is used to handle the path routings.

Requirements and Steps to run the app

Upon downloading the app, the node_modules folders are not present and installation is needed. Run a terminal/command line on the project root folder, and then run the following command:

```
cd react-socket-app          // cd to project folder.
npm run first                // npm script install packages for both server and client
```

This will install all packages needed on both server and client side. After this process is finished, we can run both the server and client instances with one single command:

```
npm run start                // npm script that runs both server and client instances
```

Now, the server will listen to the specified port (4001) and webpack will compile and host the client code, listening on a specified port (3000). A browser window should open with the URL 'http://localhost:3000'.

To run two clients at once, open a new tab with the same URL 'http://localhost:3000' or 'http://127.0.0.1:3000' if the former link is not working. The two tabs will now be playing against each other.

However, in the event that the above commands fail, these are the steps to take:

```
cd react-socket-app       // cd to project root directory
npm install               // install server packages
cd socket-client          // cd into client folder
npm install               // install client packages
cd ..                     // cd back into project root directory
node server.js            // run server
```

Open a new terminal here:

```
cd socket-client          // cd to client folder
npm start                 // run client instance
```