Michael Quach
101179729
December 6, 2020
COMP 2406

Connect 4 Project Report

1. Provide instructions for the TA describing how to log in to your OpenStack instance and start your server.

The floating IP of the server is 134.117.128.171.
The instance name is MichaelQuach.
The instance username is student, and the password is student.

The server must first be run before connecting to it. To do this, the following must be done:

In the cmd terminal:

```
>ssh student@134.117.128.171
        >enter student as the password here
>cd project
>node server
```

Once this has been completed, the node server will be running. To access the server, in another cmd terminal:

```
>ssh -L 9999:localhost:3000 student@134.117.128.171
        >enter student as the password here
        //in browser on your computer go to
http://localhost:9999/
```

All required modules have already been installed. No initialization scripts are required to be run, but the example initialization data can be viewed in the usersModule.js file, located in the /javascript folder.

2. Provide a summary of what functionality you have implemented successfully and what functionality you have not implemented.

Almost all of the functionality described in the specification has been implemented successfully in the project. To go through in order, with regards to user accounts, the login page provides a way for users to log in to existing accounts and to register a new account. To register a new account, the username provided must be valid (unique), otherwise the registration will not occur. Once logged into an account, the user can log out of the account by clicking the log out button. While in a single browser instance, only a single user can be logged in, the project supports the ability to have a multitude of users to be logged in at once, provided each has their own browser instance.

Once the user is logged in, they are able to make several changes to their account. Firstly, users are able to search for other users to send a friend request. Users can look for other users on the Community page, where they can view a table of users with public access. This table includes a search bar, to narrow the amount of users down according to the given search query. From this table, users can navigate to a specific user’s Profile page, whereupon they can send a friend request to that user. The user can navigate to their own Friends page to view the friend requests that they have sent out and to view their current friend requests sent to them. This is visible in the requests tab on that page. For each request, the user can choose to accept or decline the incoming request, or to cancel their own outgoing request. If the user accepts the incoming request, that friend is added to their friends list (and the requests are deleted from each other’s inboxes, as now they are already friends).

The friends list provides a method for the user to view their friends list. Again, this is visible on the Friends page, on the friends tab. A table of the user’s friends is viewable, along with links to navigate to that friend’s profile page, a link to challenge them to a new game of connect 4 and a status indicator to determine whether they are offline or not. Currently, users are set online once they log in, and offline only once they log out. The project does not yet support setting a user offline if they close the browser, or if their session times out. To remove a friend from the user’s friendslist, the user must first visit their Profile page. A tab displaying “Unfriend” will remove that friend from the user’s friendslist (and the user from the friend’s friendslist in doing so).

On the Index or Dashboard page, the user can access their own user settings. While there is only one setting currently, this provides an area to easily implement more settings in the future if need be (such as if the user wishes to change their password). Here, the user can toggle their privacy setting from Public to Private, saving that setting to their profile. With a private profile setting, the user does not show up in any search results (such as on the community page) unless the searching user has access to their profile (if they are friends). Likewise, with a private profile, only friends of the user can navigate to their profile page.

To view the user’s current games, they can navigate to the My Games page. Here a table of the user’s games (ordered by most recent) is available. For each game, a link to the game, a link to the opponent’s profile page, and the status of the game (which turn it is) is available. Additionally, on the game history tab, a similar table is found containing all previous games that the user has completed, again with links to the game and profile page. Instead of status, the table displays whether the outcome of the game was a win, loss, or draw.

To create a new game, the user can access the New Games page. To create a new match, the user can select an opponent from a drop down menu. The user has the option to select “Open to Public”, where a random opponent can be selected (provided that the opponent has also selected “Open to Public”). The drop down menu is also populated by the user’s friends list, and so they can select any one of their friends to invite to a match. The user must also select from a second drop down menu, indicating whether the match will be able to be spectated by no one (private), friends only, or by anyone with the match link (public). Once both drop down menus have their items selected, the user can send out a game invite.

The game invite system implemented works quite similarly to the friend request system. If both users have sent out game invites of the same spectator setting to each other, then a new game will be created between the two users (and the invites removed). The user can also access their incoming and sent game invites at the invites tab, where they can accept, decline, or cancel any invite. It should be noted that if an invite is declined then the user who sent the invite will have their outgoing invite removed from their outbox as it has now been dealt with.

When viewing a user profile on the Community Profile page, the statistics of that user can be viewed on the statistics tab. Here, this displays that user's wins, losses, draws, total games, and win percentage. On the active games tab, a list of that user's current ongoing games is viewable. Here each game item can be searched for by opponent usernames, and each item displays the opponent's name, status of the game (whose turn it is), and spectate link. If the logged in user has access to the opponent's profile, such as if they are friends, or if the opponent's profile is set to public, the displayed entry's name will be a clickable link to that profile. Otherwise, the name text is not clickable, and the logged in user won't be able to navigate to that profile. Likewise, the spectate link for each game is only available if the game has spectate options such that the logged in user can access it (if it's a public or friends only game). Otherwise, the spectate link will be replaced with text stating that it is private. It should be noted that if the logged in user is actually the displayed opponent of that game, then they will still have access to the game (which should be obvious). In the same vein as the active games tab, the game history tab provides the same features for games that the viewed user has completed. Again, status instead displays the winner or loser of the game, but profile links still remain accessible, and a link to the game's replay is available in place of the spectate link, provided the logged in user has access.

On the MyGame/id page, where id specifies the ID of the particular match being played, users are able to play in a game of connect 4. The game is played completely turn based, and so users can add a move on their turn whenever they want, and can navigate to other games or pages while they wait for their opponent to make a move. Any number of games can be played concurrently, with any number of players. As stated before, the My Games page allows for the user to view what games they are currently playing, and allows for them to easily determine which are on their turn via the status indicator. In the game page, a forfeit button is available to either of the users playing in the match, which when clicked, will end the game immediately and declare the other user the winner of said match.

Replays are also available for completed matches, allowing the user to view the game each move a time. Within a replay, the user can jump forward a move, or jump backward a move. While the user is viewing a replay, the text "Viewing Replay" is displayed as an indicator.

Here, the only functionality from the specification that has not been implemented, is the chat system. I did not have sufficient time to implement any chat system into the game page, and so users and observers are unable to chat with each other.

For the REST API, there are many routes supported. These can be viewed in further depth in each router.js file in the /javascript folder. For the specified REST API, GET /users returns an array of users of the application. This supports the name query parameter, where if included, all users which contain the query parameter (case-insensitive) and set to public and returned in the array. If no parameter is supplied, all users are returned in the array. An example of this API route being used is: http://localhost:3000/users to return all users or http://localhost:3000/users?name=123 to return all users with 123 in their username.

GET /users/:user returns a user object with the given username, if that user exists and is set to public. This user object contains the username, games, played, win rate, and their public game information. http://localhost:3000/users/NoobStomper returns the user with the username NoobStomper.

GET /games returns an array of matching games according to the player, active, and detail query parameter. The player parameter matches games according to either player of the game's usernames. The active parameter filters games that have or have not yet completed. The detail parameter determines whether the array return contains game summary objects or the full game objects with complete detail. An example of this API being used is: http://localhost:3000/games?name=noob&active=true which returns an array of detailed games with the player noob in them, that are still ongoing.

3. Describe any extensions you included beyond the required specification.

Unfortunately, I have not been able to complete any of the described extensions in the specification. There are some areas that I have added extra that weren't in the specification however. The leaderboards tab is one case, in the Community page. It provides a list of players ranked by their win rate, with profile links similar to the regular community tab.

4. Discuss any design decisions you made that you believe increase the overall quality of your system. Some important things to think about in this regard include the scalability, robustness, and user experience.

One important design decision that improved the quality of the system was utilizing one router for each web page. This modular design made it easy to add and improve functions for each webpage, as I would only have to deal with one set of routes at a time. This makes it very easy to expand on the server, as opposed to a central confusing server managing all routes.

Another crucial design decision is that all resources are obtained asynchronously (such as pages or user data like a friends array). This needed for modern web design, as it drastically increases the number of users that can access the server at any given moment (as the asynchronous nature allows the server to continue with other requests while a large one is being made). This also improves user experience, as the user can still interact with the webpage while some items may be loading.

Separating each javascript file for each individual webpage also plays into the idea of modularity, as I can compartmentalize functions specific to a webpage, and work/expand on only those while maintaining organization.

Many of the servers routes require user authorization to ensure that only the logged in user can make those requests. One example is the forfeit PUT request for a game. A PUT request is sent to the server to forfeit a game with the given game ID. To prevent any PUT request from ending a game unfairly (say someone tries to make another user forfeit), the forfeiting player's ID is grabbed from the session cookie, which is only stored upon user authentication. Thus only the logged in player can send a forfeit request for the logged in player's ID, which ensures that the data is robust - forfeits are handled correctly.

I've also made use of error checking to ensure that inputs should be what's expected, and even when an input would be incorrect, the server is able to handle those cases. An example of this is when a winner in connect 4 is declared. The board state is checked on the client side first to ensure that a win should be awarded to the correct player. This win is checked again on the server side, which may seem as redundant, but ensures that errant POST requests with a winner do not yield a change in the game data unless correct.

5. Discuss any improvements to your system that you think could be made to increase its overall quality. This is an opportunity to demonstrate your understanding of course concepts that you feel were not adequately demonstrated in your project implementation.

A large improvement that could be implemented to the system is the introduction of databases. Currently the server stores all its user and game data in RAM, meaning that changes are not saved when the server is shut off. A database like mongoose allows for easy organization of user object data, and perhaps even easier retrieval. For instance, retrieving a field (such as a friends array) from a user object, or creating a game summary object would be easy with mongoose, as it easily allows for data to be mapped to another object. Databases can also filter game and user arrays with ease using built in query parameter functions.

Another improvement to made is using sockets. Currently gameplay requires refreshing the page to view the updated boardstate if an opponent has made a move. Using sockets, the server could broadcast that the opponent has made a move and send this information, and yield an update of the boardstate then. This is more elegant than polling the server repeatedly to check if new information has been passed, as the former only sends information as needed. Using sockets would allow for updating this boardstate in a more streamlined fashion. It could also be implemented for the chat room feature for each game.

Another large improvement that could be made is to the visuals of the connect 4 game itself. The layout of the game page is something that could be worked on more, as I had run out of time when implementing that portion.

6. Identify any modules, frameworks, or other tools that you used and justify their use.

Express is a framework used for the project. Express made for simple and streamlined handling of resources, each requiring specific methods. The framework for an easy method of categorizing different routes, creating routers for individual webpages, and handling each particular HTTP method. With Express, the modularity it provides allows for subsectioning of requests, handling each with greater and greater detail. The middleware chains it allows for also yields reusable code, as some functions would be necessary in multiple routes.

Pug is a template engine used for the project. It allowed for more legible HTML. The modularity of Pug (partial views) also was quite handy for the top and side navigation bars, and thus aided in reusability of code.

7. What do you like most about your project? What would you say is the best feature(s)?

I like the game invite page, as I think its design looks pretty clean. Playing connect 4 is pretty cool too.