CSC667/867 - 01

Internet Application Design & Dev

Aug 16, 2019

Milestone 5

Team 7

Jose Castanon - Team Lead
Jonathan Julian - Git Master
Audrey Wong - Front End Lead
Inez Wibowo - Back End Lead
Tianchen Liu - Front End Dev
Shanna Zhou - Front End Dev

Repository link:

https://github.com/csc667/csc667-su19-Team07

Project Introduction.

Our team built an online real-time two-player online Chess game that includes a lobby with leaderboard, chat feature, creating a new Chess game and rejoining existing games. Our game is played with standard rules implemented in the Chess.js and Chessboard.jsx open source code.

The game is available for logged-in users only. First-time users will be asked to register, if they haven't already, and after logging in, can access any game rooms or lobby page. The lobby acts as a gateway page to access other features like viewing a list of ongoing games, a leaderboard with game standing, up to 10 top players with greater than a 0% win, a chat feature to message other players in the lobby, and create an unlimited number of new games - all on one page.

When a player creates a new game, the game room will display a chess board showing moves in real time via web sockets. The user can interact with the game board by dragging and dropping pieces into the desired valid positions. If the move is invalid, the piece will revert back to its original spot. If the player attempts to drag a piece during the other player's turn or the other player's color, they will not be able to.

There will also be a chat window for the two players, without interference from the lobby's chat. The winner, or loser, standings are tracked once a game is completed and stored in the MySQL database. This database is then used to generate the leaderboard on subsequent game loads. Even when a game is over, users in a game are able to continue playing after the game is completed in the same room.

Software stack used.

– Server: Amazon Web Services Medium

– Web server and version used: Ubuntu Server 18.04

– Database Server used: MySQL

– Server side language used: Javascript

– Framework used: React (front-end), Express & Node (back-end)

– Any additional packages or frameworks used:

Server Side Middleware : Sequelize, Bcrypt, Morgan, Socket.io, Front-End Middleware: Chess.js, Chessboard.jsx, jQuery, Bootstrap, MDBreact, Socket.io-client, Universal Cookies

Tools used for communication.

Slack: Several channels to share status, documentation, smaller tasks and troubleshooting

Weekly meetings in the Library: to collaborate and realign on tasks, troubleshooting, etc.

Google Docs: to create milestone documents

Google Spreadsheet: to keep track of tasks assigned to whom

Tools used for task management.

Tasks were divided between front end and back end. Typically via a list of to-dos on Slack, and a running list on Google Spreadsheet.

Build Instructions for our application.

Running on the Amazon Web Services (AWS) server

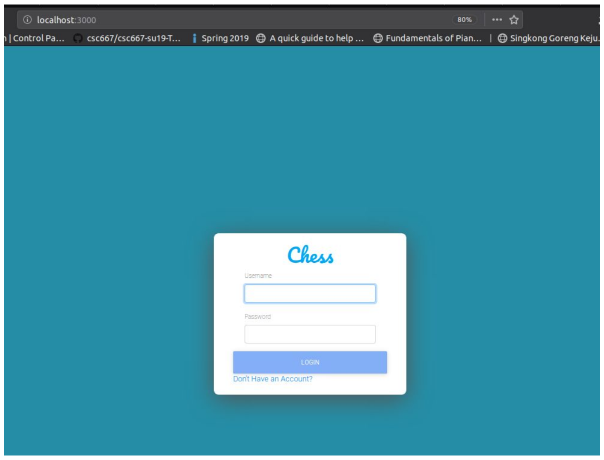
- http://13.58.209.86:3000/

Running on local computer

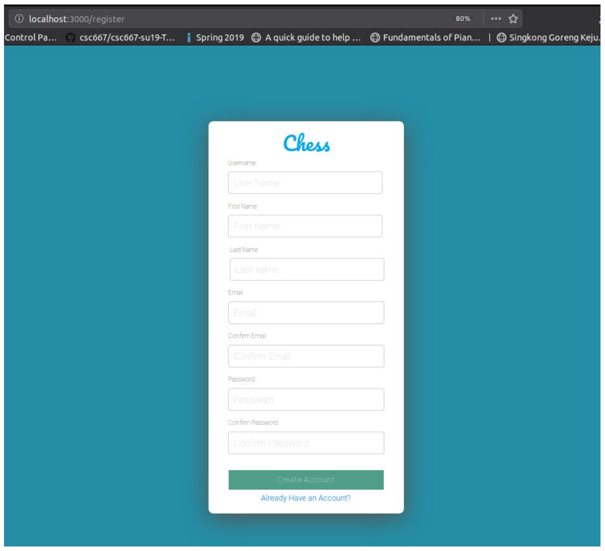
- Before running the server, navigate to front end and back end folders that contains the package.json file
- It may be necessary to install any packages not currently included in the node_modules in the back end folder.
- Enter this command in terminal on each of the folders: npm install
- Then, enter this command in terminal on each of the folders: npm start
- If the application is not running on the server mysql server will need to be set up.

Here are snapshots of pages in our application.

Login
/

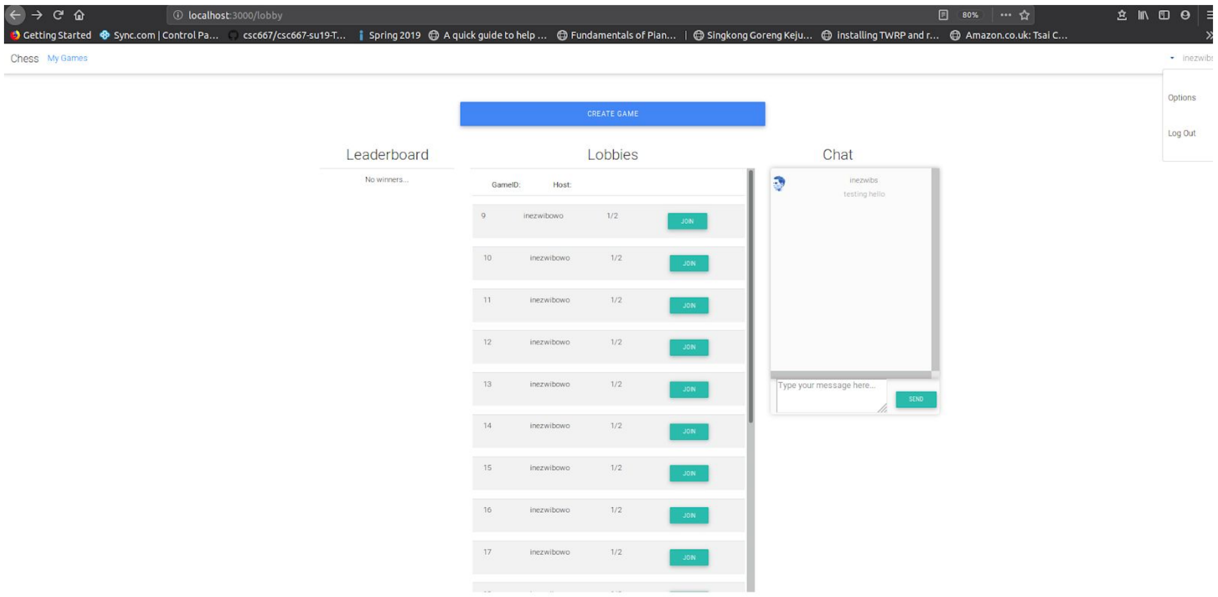


Registration
/register



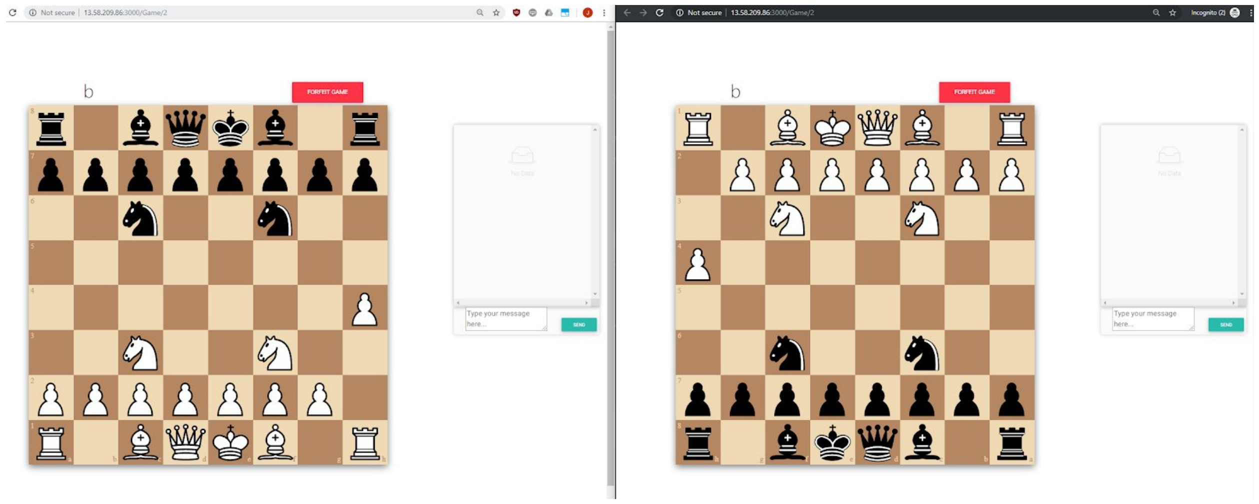
Lobby

/lobby



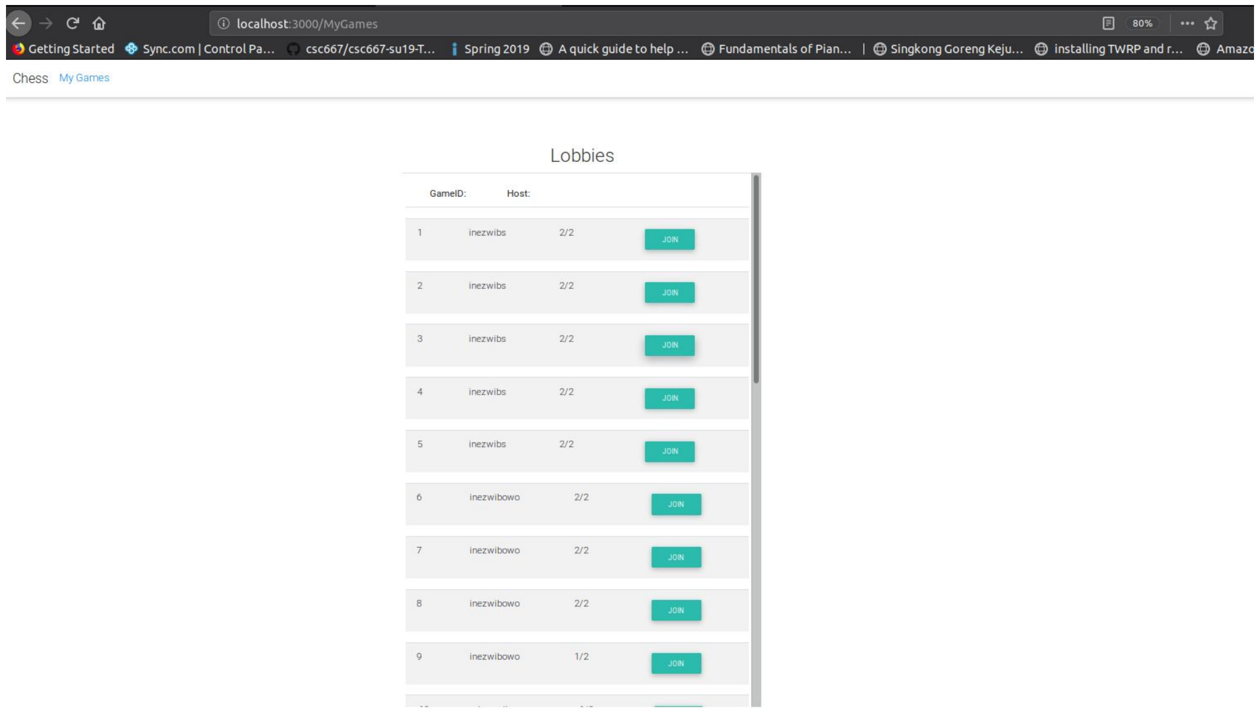
Game

/Game/:gameid/


Not secure | 13.58.209.86:3000/Game/2
b
FORFEIT GAME
No Data
Type your message here...
SEND
Incognito (2)

MyGames, list of all ongoing games for the current user

/MyGames

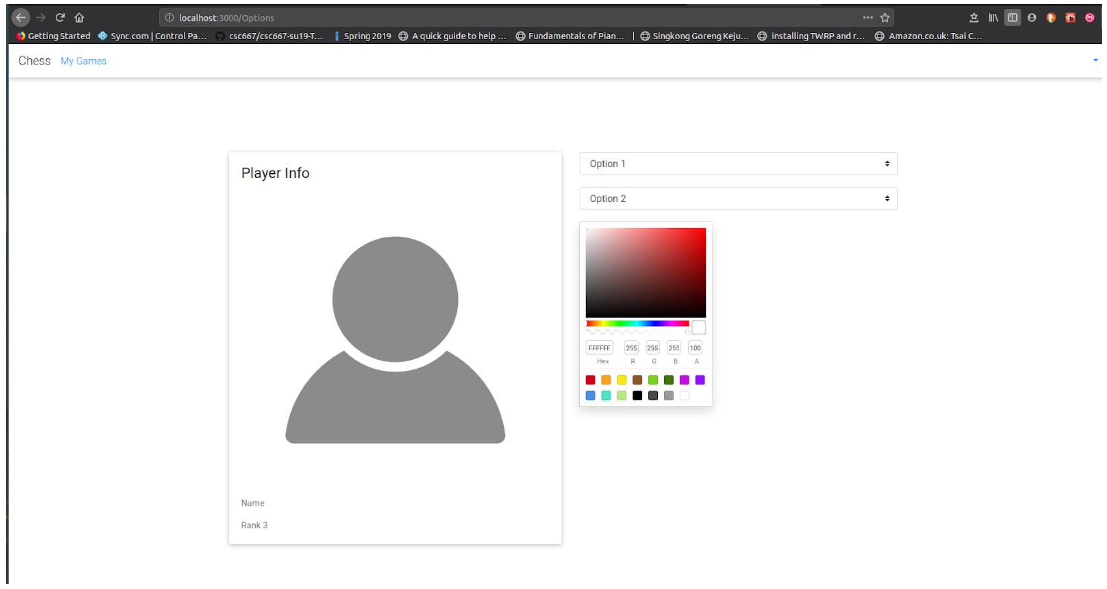


Lobbies

GameID:	Host:		
1	inezwibs	2/2	JOIN
2	inezwibs	2/2	JOIN
3	inezwibs	2/2	JOIN
4	inezwibs	2/2	JOIN
5	inezwibs	2/2	JOIN
6	inezwibowo	2/2	JOIN
7	inezwibowo	2/2	JOIN
8	inezwibowo	2/2	JOIN
9	inezwibowo	1/2	JOIN

Options, profile page allowing users to customize background color

/options



List of API routes and their descriptions.

This list focused on database interactions and functionality of application.
Here is a list of the routes in our routes directory:

App.js

router.get("/")->will call indexrouter which will call index.js

Index.js

url -> localhost/

router.use(“/users”)->Response -- log-in/register screen

router.use(“/game”)->Response -- all game related functions

router.use(“/lobby”)->Response -- functions that need to be shown in the lobby like leaderboard

game-router.js

* request url -> localhost/game

* response -> this is the full list of games playable

* router.post("/newgame")->creating new games, update game state, setting gameid, host

* router.put(":gamesessionid/") -> route to a game session with host player and 2nd player, update 2nd player when joining a game started by host, update game state

*router.post(":gamesessionid/") -> get host, player, and status, and update game stats - wins, losses, win/loss percentage for each - and update the game standings to allow for replay if desired

lobby-router.js
* request url -> localhost/lobby
* response -> this is the lobby/global chat/leaderboard/Game List
* router.get("/getleaderboard") -> get leaderboard with this request

users-router.js
* request url -> localhost/users
* response -> this is the users page
* router.get("login") -> get the login page

*router.post("/games") -> send login info with pre-registered username and password
* router.get("/register") -> display the registration page
* router.post("/register")-> send registration info to the database

Team Member Contributions.

Jose Castanon:

Front-end: Implemented Chess.js and Chessboard.jsx logic in game/board page, and handled back end connection for setting and retrieving game states. Implemented socket functionality for game board. Handled connection of components and back end routes, handled request and responses using React. Set up front end proxy for login, game states, lobbies.

Back-end: Implemented sockets for lobby and game board. Miscellaneous troubleshooting of routes.

Misc: Final application deployment on AWS server. Assigned tasks, kept track of to-do's for milestones. Helped front-end teammates with React troubleshooting and implementation planning.

Audrey Wong:

Front-end: Helped with connection of front and back and created tables to display all games in Lobby. Also connected front and back and created tables to display leaderboard. Worked on overall styling on pages. Created the Login page.

Miscellaneous: Created sketches for website layout and planned original design. Assisted with setting up files in github. Also helped with original layout. Help review and add to milestone documents.

Tianchen Liu:

Front-end: Helped with connection of front and back and created Registration and validation. Worked on overall styling on Login, Registration, Lobby, and helped in sockets.

Back-end: Debugged back-end routing and get function from the SQL DB.

Miscellaneous: Set up server on AWS, set up basic environment for future development.

Inez Wibowo:

Back-end: Created tables in MySQL Workbench, created files in models, helped with creating routers, added Socket.io for chat and joining game

Front-end: Edited Socket.io, chat and joining game

Miscellaneous: M5 documentation, Helped with updating task assignment to-dos list, helped with testing

Jonathan Julian:

Back-end: Designed database architecture, implemented sequelize models, created routes, worked with Socket.io troubleshooting issues on server, initial Express server set up, mySQL server setup/troubleshooting .

Front-end: Troubleshooting Socket.io on server implementation. Assisted Jose with game logic/route interfacing.

Miscellaneous: Managed Git repository, deployed application on to server, hot fixed server issues for both development and production releases, assisted in documentation.

Shanna Zhou:

Front-end: created option page, created client and server socket.io in chat.

Project Reflection.

Chess game required several team members to ramp up quickly and continuously learn new technologies and frameworks at all stages of the project, whether that be React, or Socket.io, Web Sockets, etc.. First challenge in tackling the Chess game is understanding what is being provided by the libraries and what is not. As the development progressed, team members became more comfortable with the code base and settled into respective tasks as agreed upon in team meetings/check-ins.

Meetings/check-ins and recording task assignments became more critical and frequent towards the end of the project.

There were quite a few significant issues which arose during this project that can ultimately be narrowed down to the technology we chose. When putting things in retrospect the software stack we chose was a considerable hindrance to the successful completion of this project. Choosing a technology in which very few members had prior experience with made for an exceptionally difficult time to coordinate the communication between both the front and back end components. This fact coupled with the length of the summer term made this project especially difficult.

After learning some of the intricacies of React and understanding the limits of our production server in terms of processing power really made the deployment of the application problematic. Our choice on the server to run TMUX (terminal multiplexer) also added to this stress. There were numerous times that the server would simply lock up from the lack of memory or we were unable to access the running instance of the application. Upgrading the server was necessary at the end of the semester.

The use of the libraries such as chess.js and chessboard.js assisted us in getting as far as we did. I think that with the extra 6 weeks of the regular semester our team would have been able to complete the project to its entirety per original specs - beyond priority features - and have it all deployed on the server.

Project Conclusion.

A few decisions helped us in this project, such as deciding to use Chess game library, decision to have dedicated front-end and back-end team, and periodic floaters also helped. In retrospect, the Chess game project would have gone smoother with clearer task assignments and more frequent check-ins.

Overall, we were able to deliver most of the features in our priority list: Creating Game, Playing Game, Joining Game, Loading Games, Persistent Game State, Ingame & Lobby Chat, Assignment of Players, and Win/Lose Percentage. The main takeaway from this project underscored the importance of “Scrum” meetings as frequently as possible. Also, this class gave us the experience of working in a startup environment, especially in terms of the timeline, lack of resources (such as server instance processing power), and the overall stress in which a looming deadline provides.