CSC 667/867
Internet Application Design and Development
San Francisco State University
Summer 2019

TEAM - 04

Members/Role:

Adam Tremarche: Team Lead/ Front End Developer
Aashutosh Bajgain: Front End Developer
Gem Angelo Lagman: Back End Lead
Jesus Garnica: Front End Lead
Juhi Kushwah: Front End Developer

Github Repository Link:- https://github.com/csc667/csc667-su19-Team04

Project Introduction

G4Chess

Our Term Project for CSC 667 is a chess web app which allows two users to create and play simple one-on-one rounds of the classic board game.

Visitors create an account, login, and then from there can either create a game where they will play as white or view the list of already available games. The list contains both games our visitor has already begun above, and games of players looking to find an opponent which our visitor can add to their available games. Once our visitor chooses a game by setting it as their active game, they can then travel to it and begin play.

Our chess gameplay is handled by the javascript package Chess.js, and our game board display is handled by Chessboard.jsx.

Software Stack Used

AWS Ubuntu Server	*cloud server*	*ver. 18.04*
Gatsby	*React framework*	*ver. 2.13.52*
Node.js	*Javascript runtime*	*ver. 10.16*
Express	*Node framework*	*ver. 4.17.1*
MongoDB	*database*	*ver. 4.0.10*
Socket.io	*real-time interaction*	*ver. 2.2.2*
Axios	*route handling API*	*ver. 0.19.0*

Tools Used For Communication

Discord	CSC 667 TEAM MEMBERS

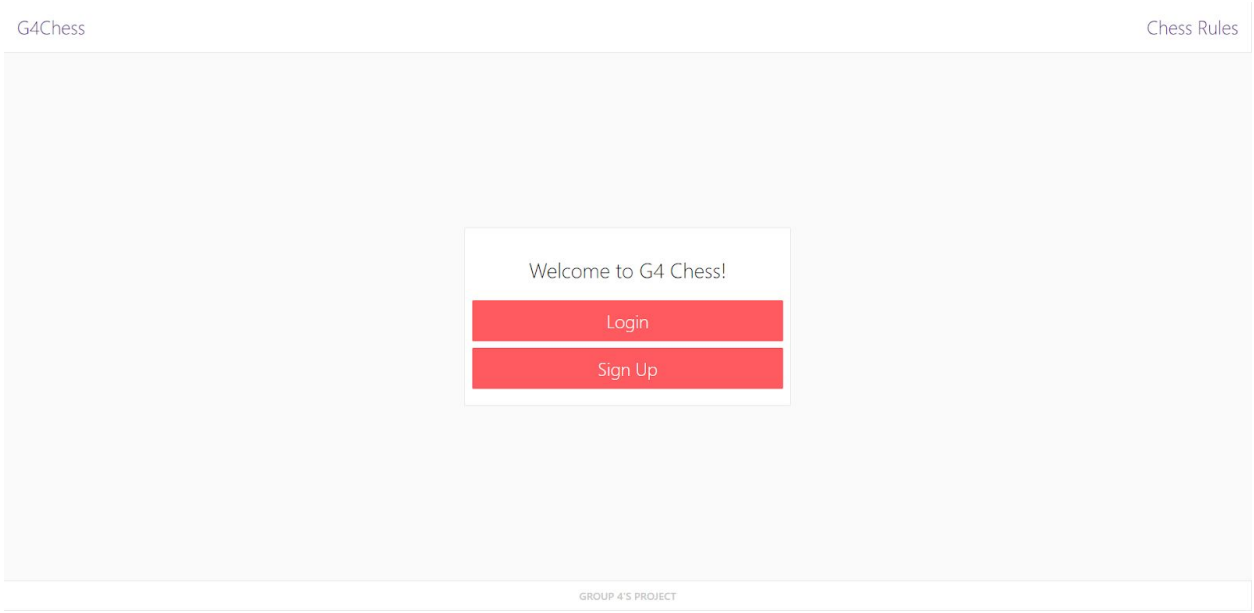
Tools Used for Task management

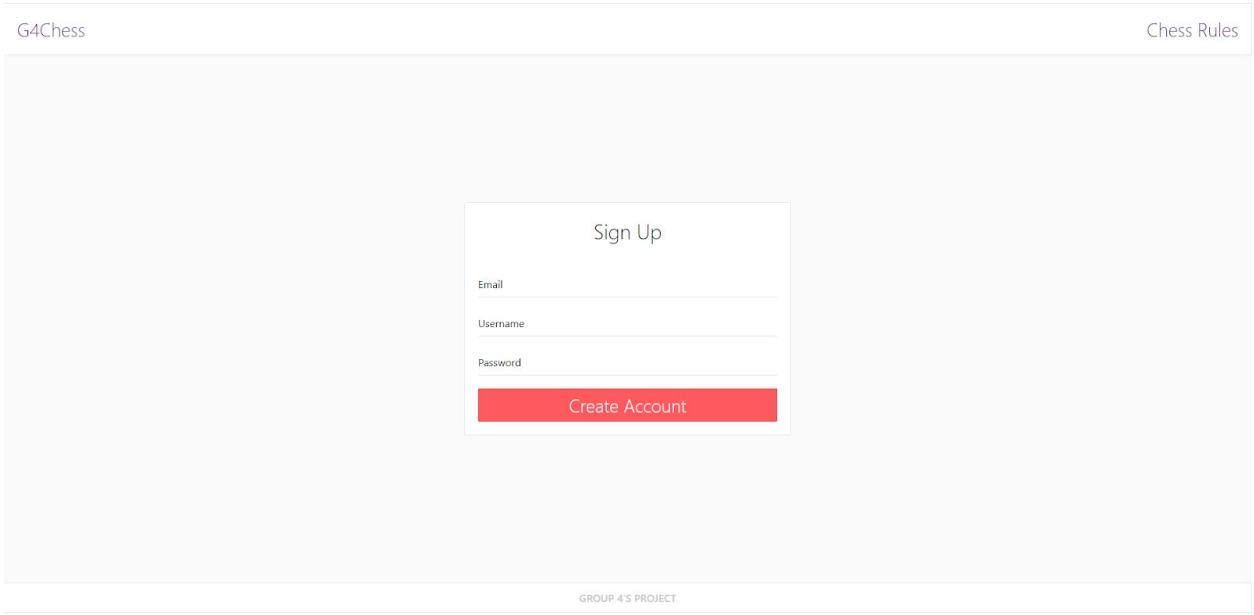
Google Docs	*used for collaborating on milestone documents*
GitHub Issues	*used for assigning tasks at hand*

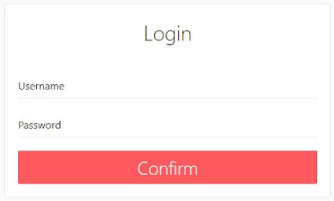
Build Instructions

1. Download the files from our GitHub repository here.
2. Using the console, run “npm install” in both the /Frontend and /backend directories.
3. Inside the /backend directory, run “node index.js” to execute the backend.
4. Inside the /Frontend directory, first run “gatsby build” to compile the static html, and then “gatsby serve” to execute the frontend.
5. To use a more developer friendly interface, use “gatsby develop” to execute a debugger and auto-compile along with executing the server.

Application Storyboard







GO HOME

Chess Rules

Setting up the board:

The board should be set up with the white square in the nearest row on the right, "white on the right". If this isn't done the king and queen will be mixed up. Shake hands across the board before the game starts. White always moves first.

Ranks and files:

Going from left to right, the vertical rows on the board, called files, are labeled a through h. The horizontal rows, called ranks, are numbered 1 to 8. The 1 is white's side of the board; 8 is black's side. This system can be used to show what square a piece is on in a way like the game Battleship. When the board is set up the square a1 will be on the white player's left side.

G4Chess Log Out Chess Rules

Join Game

Your Current Games

Player1	Player2	Set As Active
dood	None	Set As Active
old	dood	Set As Active

Refresh List Go to Active Game

Open Games

Opponent	Game Type	
old	Public	Join

G4Chess Log Out Chess Rules

GROUP 4'S PROJECT

List of API Routes and their Description

➔ //newUser
//this route posts a new user to the database
//request url -> newUser
//response -> {}
app.post('/newUser', (req, res) => {
…
});

➔ //login
//this route checks login info
//request url -> login
//response -> {user: username, password}
app.get('/login', (req, res) => {
…
});

➔ //logout
//this route logs the user out of their session
//request url -> logout
//response -> {}
app.get('/logout, (req,res) => {
…
});

➔ //createGame
//this route creates a new instance of a chess game
//request url -> createGame
//response -> {}
app.post('/createGame', (req, res) => {
…
});

➔ //joinGame
//this route adds a player to a game
//request url -> joinGame

```
//response -> {}
app.put('/joinGame', (req, res) => {
        …
});

➔ //openGames
//this route gets all the open games that the user can join
//request url -> /:username/openGames
//response -> { games[]}
app.get('/:username/openGames', (req, res) => {
        …
});

➔ //currentGames
//this route gets all the games that the user has joined and not finished
//request url -> /:username/currentGames
//response -> { games[]}
app.get('/:username/currentGames', (req, res) => {
        …
});

➔ //getGame
//this route gets the specific game given by the gameID
//request url -> getGame
//response -> { game }
app.get('/getGame', (req, res) => {
        …
});

➔ //updateFen
//this route updates the game's FEN position for the board
//request url -> updateFen
//response -> {}
app.put('/updateFen', (req, res) => {
        …
});
```

➔ //updateResult
//this route updates the game's result and marks the game as finished
//request url -> updateResult
//response -> {}
app.put('/updateResult', (req, res) => {
…
});

Team Member Contributions

Adam Tremarche: Team Lead, Front End Developer

- ➔ Handled initial server setup and ongoing server maintenance
- ➔ Wrote code handling populating available game lists
- ➔ Wrote code handling joining and resuming games
- ➔ Wrote code handling updating player moves into game instances
- ➔ Wrote code handling finished games
- ➔ Assisted integration of Chess.js and Chessboard.jsx
- ➔ Contributed in Milestone Document authorship

Aashutosh Bajgain: Front End Developer

- ➔ Handle setup for Discord, Sourcetree, Github and other documents related collaboration.
- ➔ Wrote code for implementing various buttons and link within the pages.
- ➔ Wrote code for Chess Rule games.
- ➔ Adjusted styling of various pages using various react components.
- ➔ Worked with front end lead to make sure the User interface looks visually appealing and interactive.
- ➔ Contributed in Milestone Documentations authorship.

Gem Angelo Lagman: Back End Lead

- ➔ Handled setup and connection for the MongoDB database.
- ➔ Wrote code for User database model and related express routes.
- ➔ Wrote code for UserSession database model and related express routes.
- ➔ Wrote code for Game database model and related express routes.
- ➔ Wrote code for Socket.io implementation on back end.
- ➔ Assisted in Milestone Document writeup.

Jesus Garnica: Front End Lead

- ➔ Did the initial setup for GatsbyJS
- ➔ Added Flag Notifications to the login and sign up.
- ➔ Created the Navbar and footer
- ➔ Wrote the ContentContainer used across the website to create a uniform look across the pages.
- ➔ Created a server side rendering friendly version of Chessboard JSX
- ➔ Created the animated inputs for the forms.
- ➔ Allowed the website to save whether or not the user is signed in.
- ➔ Did the initial setup using Axios so we could use it across the site.
- ➔ Made the Board mobile friendly.
- ➔ Made the home page mobile friendly.
- ➔ Did the final styling on the game selection screen.
- ➔ Setup PM2 for gatsbyJs
- ➔ Setup Gatsby JS to live on the server.
- ➔ Fixed SocketIo issues on the front end to respond to events.
- ➔ Added move checking for dropping and clicking on chess moves.
- ➔ Setup the flags to let the user know when it is their turn.

Juhi Kushwah: Front End Developer

- ➔ Assisted in writing components and pages.
- ➔ Worked on use cases to build a login system for MERN stack.
- ➔ Worked on game logic.
- ➔ Wrote code for integration of Chess.js and Chessboard.jsx.
- ➔ Assisted in document writeup for Milestones.

Project Reflection

This team began without much experience in web development. Aside from a couple of us who had already taken CSC 648, our skill levels ranged from having some specialized experience in one aspect of development to complete beginners.

Initially we had the idealistic notion that maybe we wouldn't specialize ourselves into a hierarchy of backend and frontend developers so that we might all gain some level of understanding of each aspect of the development process. This proved to be untenable and as the project began to take shape members naturally came to gravitate towards specialization as workload and aptitude dictated, but I believe this initial impulse, to learn and understand each piece of the machine we collectively embraced early on, helped set a tone of communication and teamwork which carried through with us from beginning to end.

Everyone one of you showed dedication, responsibility, and a willingness to communicate and for that you have my sincere thanks. We may not have delivered every feature, but we did what we could with the time we had, and I am proud of our simple, but elegant little chess website.

Adam Tremarche, Team 04 Leader

Conclusion

Our project was generally a success, but with a few promised features missing.

We were able to deliver a live, chess web application, which stores and retrieves information from our database to facilitate real-time chess games between any number of concurrent pairs of players. Our site has registration, and account authentication as per the instructions of the assignment. And, our site features a minimal, but pleasing design, which works on both desktop and mobile.

Currently, we are missing the chatroom functionality of the game. As time pressure became more of an issue we deprioritized this feature as advised, and unfortunately we needed all the time we had left in order to make our core game features work completely.