COMP 2406 Project Report

For Professor Dave McKenny, Fall 2020 semester
By Gordon Tang 101158226

Project of choice: Connect 4 web app

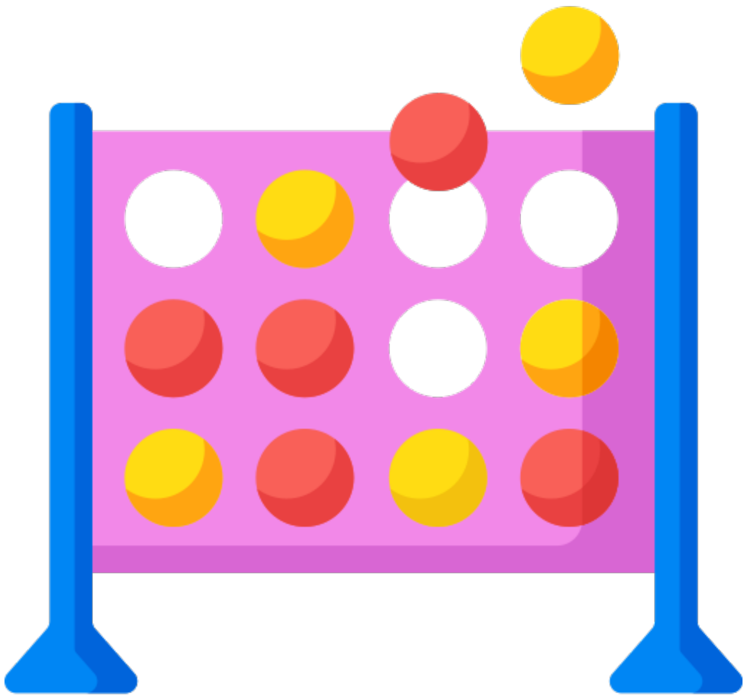


Table of Contents

1. How to access via Open Stack

Instructions on how to open this project on OpenStack.

public IP is: 134.117.130.233
username is: student@134.117.130.233
password is: topsecretpassword

instructions to launch and test the website:
connect to Carleton's VPN, then
in the terminal, type:
ssh student@134.117.130.233
login in with password:
topsecretpassword
(now you have connected to OpenStack on my account)

in the same terminal, type:
cd connect4
node expressServer.js
(this starts my server and in case of failure, go to this terminal,
press control + c to stop all terminal operations, then type
node expressServer.js again)

connect to the ssh tunnel in another terminal, type:
ssh -L 9999:localhost:3000 student@134.117.130.233
login in with password:
topsecretpassword
(this creates the tunnel which connects the internal port 3000 to a public port of 9000)

access the website in a browser window at http://localhost:9999

2. Description of Features

Below is a list of features that have been implemented successfully and what functionality that are not implemented successfully. **Bold** means not implemented, otherwise, the feature has been implemented.

User Functionality

- Search for other users to form a friendship with
- Send a friend request to other users of the application
- See their current friend requests and accept/reject those requests
- **Unable to see if a friend is online or not**
- Navigate to a friend's profile
- Remove a friend
- Update personal privacy setting
- Update profile privacy

- View current active games and navigate to any specific game
- Create a new game with a random person
- View a history of games played, who they played against and the result of the game

Viewing user’s profile

- See summary statistics of another user
- See last 5 games of another user
- See other user’s current active games

Game Functionality

- View current active games
- Continue playing active games
- Easily see who’s turn it is
- Forfeit any game
- Chat with players and observer
- Indicate who is in the game lobby

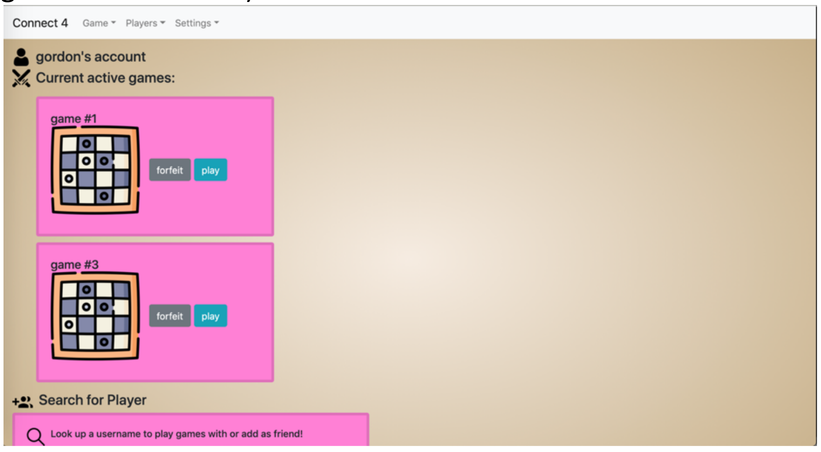
REST API

- /users -> optional name query parameter
- /users/:user -> get information about this public user
- /games -> optional query parameters to get information on all games

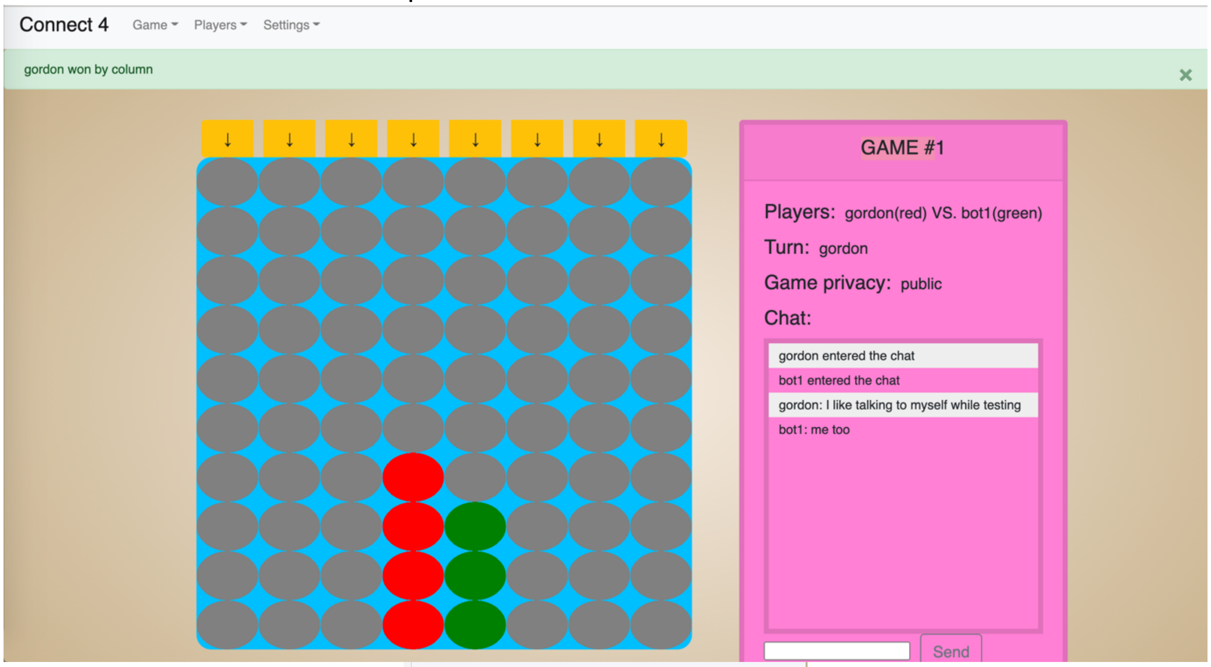
3. Extensions beyond course requirements

Below is a list of features that have been successfully implemented beyond basic requirements.

1. Mobile first principle of Bootstrap’s philosophy
    - Impressive CSS
    - Flexbox for responsiveness
    - Intuitive layout with cards
    - Intuitive icons for quick pattern recognition
    - Appealing colors (I went for a blue vs red theme with an earthy background which is a metaphor for red vs blue, man vs women but at the end, all grounded on earth)



2. Responsive website, mobile and desktop friendly
    - More accessible to users
    - Enhance user experience



gordon won by column

GAME #1

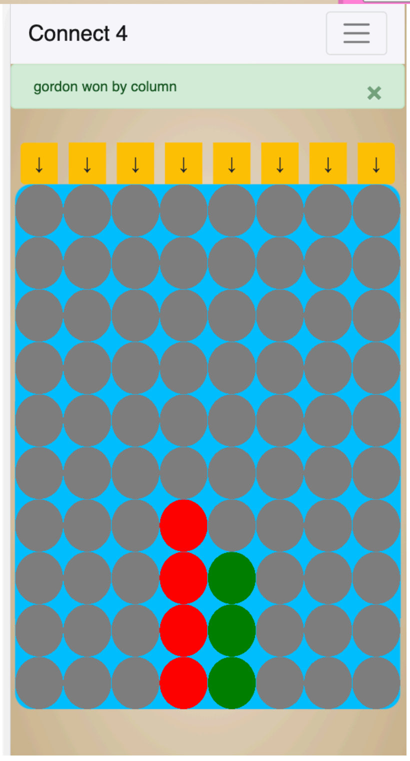
Players: gordon(red) VS. bot1(green)

Turn: gordon

Game privacy: public

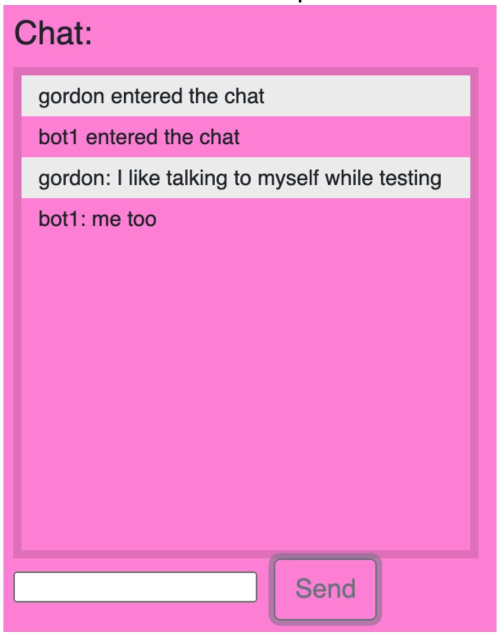
Chat:

gordon entered the chat
bot1 entered the chat
gordon: I like talking to myself while testing
bot1: me too



gordon won by column

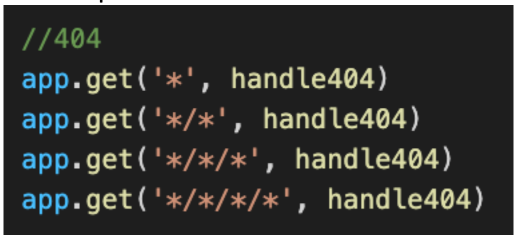
3. Live chat rooms with socket io
    - Faster and more reliable than AJAX updates


Chat:

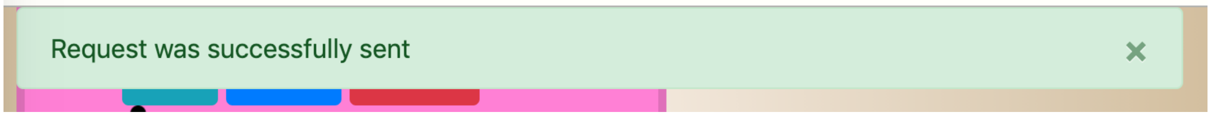
gordon entered the chat
bot1 entered the chat
gordon: I like talking to myself while testing
bot1: me too

Other smaller features which I thought were cool.

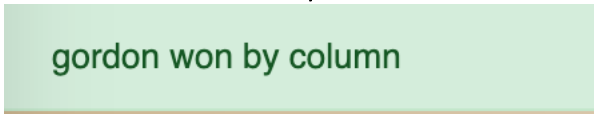
- Exhibit A: Infallible 404 responses


```
//404
app.get('*', handle404)
app.get('*/*', handle404)
app.get('*/*/*', handle404)
app.get('*/*/*/*', handle404)
```

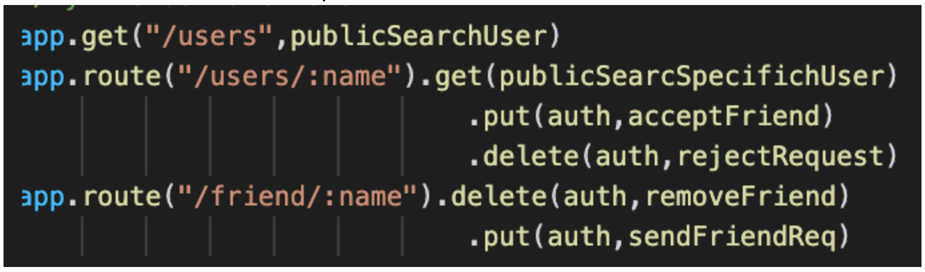
- Exhibit B: Sticky and removable speech bubble from the computer

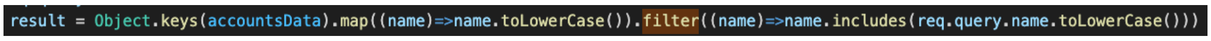

Request was successfully sent ×

- Exhibit C: Why just know who won when you can also know how they won


gordon won by column

- Exhibit D: Clean REST routes, clean filter functions


```
app.get("/users",publicSearchUser)
app.route("/users/:name").get(publicSearcSpecifichUser)
                          .put(auth,acceptFriend)
                          .delete(auth,rejectRequest)
app.route("/friend/:name").delete(auth,removeFriend)
                          .put(auth,sendFriendReq)
```


```
result = Object.keys(accountsData).map((name)=>name.toLowerCase()).filter((name)=>name.includes(req.query.name.toLowerCase()))
```

4. Design decisions to improve the quality of the application

First, I will outline my overall tech stack. Then I will list features, best coding practices or industry standard procedures that enhance the overall robustness, speed or scalability of the application or areas that improve the user or developer experience in the application.

Technologies: HTML, CSS, JavaScript, Bootstrap, NodeJS, Express, Socket IO, Express session

- Bootstrap

Bootstrap is a popular external CSS resource. I picked Bootstrap because there are lots of resources online and is flexible to implement. Their well-designed components are so easy to add that all I had to do was copy and paste. Besides augmenting my html code with their ready-made classes, I used their alert and nav bar feature to enhance the user experience. Nav Bars are especially convenient for mobile users since moving between screen is harder in general. Alerts are useful for displaying server responses on client interactions, thus enhancing the client server interface.

- AJAX

Or more appropriately, the acronym should be changed to AJAJ for Asynchronous JavaScript and JSON. It uses HTML/CSS, DOM, JSON, JavaScript and XMLHttpRequest to asynchronously update parts of the screen instead of reloading the entire screen.

I implemented AJAX requests when updating the following functions of remove friend, add friend, reject request and forfeit game.

- NPM for developer experience

NPM allows developers to easily add external modules and easily re-build projects in other locations. This strategy improves the developer experience as necessary packages are automatically updated to the right version.

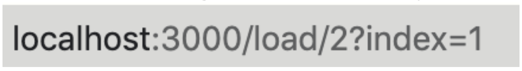
I implemented NPM to include the follow libraries: express, express-session, pug, socket io and UUID.

- REST API

A RESTful design is a web standard that aims to make websites more scalable, fast, extensible, reliable and simple.

1. Separate client and server side
   a. Interaction is through create, read, update, delete operations also known as CRUD operations. Refer to Appendix Picture 1 for my API design.
   b. In my code, the server is responsible for storing users and game data, providing rules of interactions between users and rules of the Connect 4 game. And clients are people or servers who are able to interact with the server to retrieve meaningful responses or data through public and private API routes.

2. Stateless design
    a. The server does not remember the historical requests in the case where users are viewing their past games.
    b. In my code, the game board and information are requested independently which leads to greater scalability.



localhost:3000/load/2?index=1

3. Uniform interface
    a. My system has the following routes: users, game and load which are responsible for most operations a user or server can make. Further specification on the routes of user and load will output different information.
    b. Headers have been specified in my AJAX requests and are one of either GET, POST, PUT or DELETE.
4. Layered System
    a. In my code, there is a distinction between a client and a server.
    b. Further directions can include separating the status quo with load balancers and database.

5. Future directions for this application

Below is a unexhaustive list of next steps to further improve the quality of the application.

- Include databases for more data persistence (ex. Mongo, Mongoose)
    - I have looked into tutorials on Mongo databases. It doesn't look too hard considering the application needs are so simple in this application. However, I have been quite tight on time this semester and debugging, learning, designing, implementing takes up a lot of time.
    - It seems that after creating a database instance, there are CRUD operations on the database for manipulating data.
    - The benefit of using a database is greater data persistence. As of now, my data resets after a server refresh which is annoying since I am limited to the amount of data I create while the server is not down.
    - Another benefit of using databases is that it makes the server faster. This is because less memory is stored on RAM which will be a bottleneck when scaling the app to more users.
- Include an AI agent (ex. minimax algorithm)
    - I have been researching how to code a minimax algorithm. As of now, I have a working code of a connect 4 AI, but I didn't want to copy his code, so I never included it in my project. The concept is very interesting to me.
    - In essence, the AI is based on the min max algorithm. The naive version of this algorithm draws up all final board states by creating an exhaustive decision tree of all outcomes. At every move, the computer searches the best score and makes that move. The best score function is where the creativity happens. You can decide the score be calculated as a function of number of moves * quality of move. Where number of moves is the depth of the tree and quality of move is the number of 2 chains, or 3 chains on the board.

- Include automated and in-depth test cases (enhance developer experience)
    - Automated test cases can be done with Selenium. I have had some experience with Selenium before. It is a python library that automates clicks on the browser.
    - An example test case of Selenium is to test the adding friend feature. What would happen is that the bot logs in and navigates to the friend's page then click on a add friend button then stop the script. A successful outcome would be one that has a new friend in the friend list and one less item in the friend request list.
    - Now that I think about it, I have been playing the role of a bot this whole time. Logging in every time slows down my rate of development so having an automated test script is definitely something I will consider for future projects.

6. Defense of technologies used in this application

NPM modules that were utilized were express, express-session, pug, socket io and UUID.

Express is much better than vanilla Node JS as most repetitive code is abstracted away from the developer, hence allowing the developer to spend his or her energy more towards the design and less towards nitty gritty details.

PUG is a dynamic templating engine similar to JSX which I have some experience in. Dynamic rendering is a MASSIVE upgrade to manually updating DOM elements.

Express session puts a cookie on a user to keep track of authentication his or her account and authorization throughout the application. This is useful since it is often that companies can analyze user behaviors to generate better suggestions for individual users.

Socket IO creates a connection between a server and a client akin to a plug on the wall and the cable between the socket and an electrical appliance. I used Sockets for the chat feature so users can talk to each other in real time and not wait a one second lag between every page update.

UUID is a useful tool when new object identification tags are needed. Different objects requiring different identification seems quite intuitive. I used UUID for creating game ids.

7. Final words on the project

Below is my reflection of this semester long project.

My favorite feature is the chat. I love the chat because for the first time in the whole development process, it felt like users can have a legit platform to creatively express themselves and a place where mistakes are inconsequential. There won't be a compiler yelling at me for forgetting to update a variable name because it doesn't matter what a user type. This contrasts everything else in my app which are 100% hardcoded data that I made or rules that I made or bugs that I created.

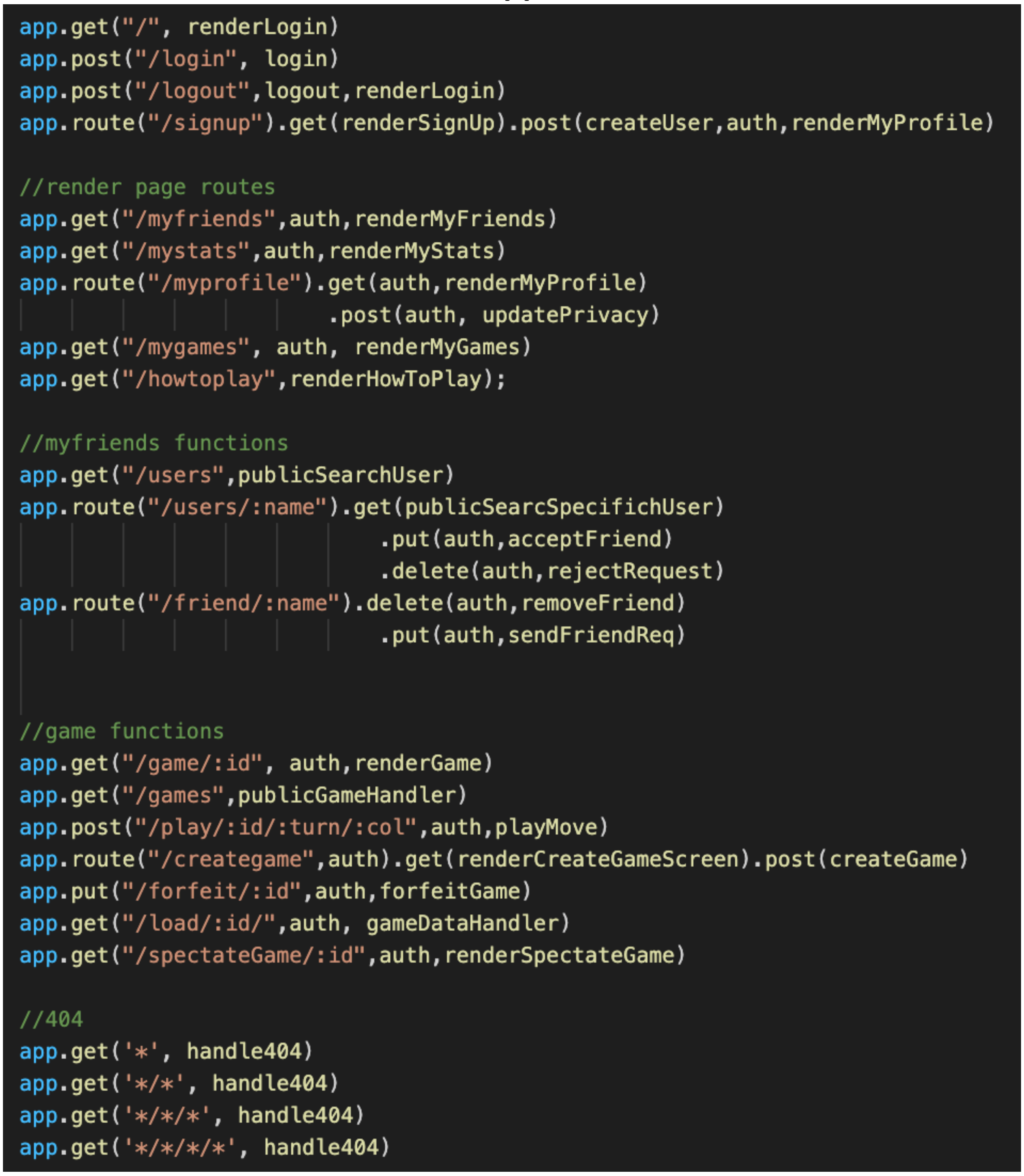
I'll wrap up by reviewing a meme we were introduced to earlier in the semester. It reads "K.I.S.S. – Keep It Simple Stupid. Great advice…Hurts my feelings every time.". Looking back, I would have done a lot of things differently and it is often that a simpler solution can be found but only in hindsight....

K.I.S.S. - Keep It Simple Stupid
Great advice...
Hurts my feelings every time

8. Acknowledgments

- Icons made by Freepik from www.flaticon.com
- Special thanks to Dave, Jaegan the GOAT, Lucas and Bruce the coolest TA's.

9. Appendix



```
app.get("/", renderLogin)
app.post("/login", login)
app.post("/logout",logout,renderLogin)
app.route("/signup").get(renderSignUp).post(createUser,auth,renderMyProfile)

//render page routes
app.get("/myfriends",auth,renderMyFriends)
app.get("/mystats",auth,renderMyStats)
app.route("/myprofile").get(auth,renderMyProfile)
                        .post(auth, updatePrivacy)
app.get("/mygames", auth, renderMyGames)
app.get("/howtoplay",renderHowToPlay);

//myfriends functions
app.get("/users",publicSearchUser)
app.route("/users/:name").get(publicSearcSpecifichUser)
                            .put(auth,acceptFriend)
                            .delete(auth,rejectRequest)
app.route("/friend/:name").delete(auth,removeFriend)
                            .put(auth,sendFriendReq)


//game functions
app.get("/game/:id", auth,renderGame)
app.get("/games",publicGameHandler)
app.post("/play/:id/:turn/:col",auth,playMove)
app.route("/creategame",auth).get(renderCreateGameScreen).post(createGame)
app.put("/forfeit/:id",auth,forfeitGame)
app.get("/load/:id/",auth, gameDataHandler)
app.get("/spectateGame/:id",auth,renderSpectateGame)

//404
app.get('*', handle404)
app.get('*/*', handle404)
app.get('*/*/*', handle404)
app.get('*/*/*/*', handle404)
```

Picture 1