COMP 2406

Final Submission Report

Connect4 Web Server

Name: Cyril Patrick Lynch

Student Number: 101169963

Date: December 6, 2020

This server is based on the specifications of the Connect4 term project as given by Professor Dave McKenney in the COMP2406-F20-Connect4-Project pdf.

Report Details:

1. Logging into OpenStack and starting the server	
a.	Instance Information: • public IP = 134.117.131.8 • instance name = PatrickLynch • instance username =student • instance password = studentmarimba • instance Private IP = 192.168.58.112
b.	Everything should be installed, and no other software should be needed
c.	There are no initialization steps for this, as I have yet to add a database. Also, the server should be running, but if it is not, then go to part d for directions.
d.	To run the server: • start a vpn connection with Carleton • Open an instance of a cmd shell and log onto the OpenStack server: • ssh student@134.117.131.8 • Type in password: • studentmarimba • Go to the directory with the server files in it: • /home/student/WebServer/connect4App • Start the server via: • node app.js • Open another instance of a cmd shell and create a ssh tunnel in it via: • ssh -L 9999:localhost:3000 student@134.117.131.8 • Open a client (i.e. browser) and log on to an account. The user with the most data is: • Username: Pat • Password: 1234 . • The log in pages can be found at: • http://localhost:9999/ or • http://localhost:9999 or • http://localhost:9999/login • Other User information can be found in the console logs during the Startup of the server (it might be worth it to shut it down and restart the server if you want to take a detailed look at the user and game objects currently on the server) • Additionally, there are game console logs which I tried to make fairly readable • All of the printing functions are found at the bottom of model.js under the comment ‘testing’ • Logout when done
e.	Everything should be installed, and no other software should be needed

Brief overview of important files and folders	
app.js	This is the main server, which contains all of the routes that a client can pull from. This server only interacts with the Client, resources in the public folder, and the model.js file. It does not interact directly with the databasecontroller.js file.
model.js	This file contains almost all of the business logic and helps maintain data integrity, as well as prevent poor queries and interactions with the databasecontroller.js file that would otherwise cause server errors
database-controller.js	This file contains some business logic but is mostly occupying the space of a database. This is the file that would interact with the database, preventing poor queries and making sure basic rules of information storage on the database aren't broken (ex. visibility can only be set to 'pub', 'pri', 'fri', or 'del'). Most of the querying work is done in this file as well, as that would typically be a job for the database.
public/views	All of the structure for the website. This folder contains all of the pug files as well as a folder separating the pug files used via the "Include" keyword
public/css	Contains a single site wide css file. Every web page on the site uses this file.
public/js	Contains all of the JavaScript files used on the client side of the website. Most of the files are named according to what pug file uses them.

2. Summary of implemented and unimplemented functionality	
Not implemented	The only functionality that I did not implement is connecting the server to a database. I was hoping that by separating all the getters, setters, and query functions and by putting them into the databasecontroller.js file that I would only have to refactor this single file in order to implement the database. However, as I would also have needed to rework each function in the app.js file (the server) to make them asynchronous, I decided that I did not have enough time to implement it, and instead prioritized implementing the rest of the specifications for this project.
Implementations under the User Accounts specifications	
1. Searching and befriending users	After logging in, use the top left dropdown menu and click "Search for People" to go to a search page. By using the buttons on the right, you can: • Send a friend request • Delete a sent friend request • Decline a received friend request • Accept a friend request • Unfriend a user

	The buttons update the user profile accordingly, and upon activating one, the page will dynamically update based on the selection. To become friends with a user, one must send a request to the desired user, and that user must accept the request. The “Search for People” function only shows the first 10 users applicable to your search and excludes the requesting user
2. Sending friend requests to users	• Hover over the drop-down menu icon on the top left side of the screen • Click on “Search for People” • You will then see a list of available users or a search bar in which you can search for usernames • When you have located the user you want to befriend, click a button next to their name labelled “send friend request”
3. Receiving and accepting or rejecting friend requests	• Hover over the drop-down menu icon on the top left side of the screen • Click on “Search for People” • You will then see a list of available users or a search bar in which you can search for usernames • You can scroll through this list of users and buttons to the left of each user’s name will dictate if you have friend requests to accept or reject. You can do so from this page. OR • After logging in, go the aforementioned drop-down menu (top left) and click on the “Profile” link. Then, click on the number in parentheses and it will direct you to your friend request page. The number in parentheses is the number of current pending friend requests. • Alternatively, at the top of the Home page and the Profile page is the username with ‘(friend Notifications)’ beside it. just click on notifications and it will bring you to your notification’s page
4. Check for friends logged in	After logging in, go to your profile. Then, click on the “Go To Your Friends Page” link which will take you to your friends page (should look similar to the Search page). Then in the “Settings” drop-down menu (top right), click on the “Toggle Show Only Users Logged On” button which once toggled will only show logged on users.
5. Navigate to a friend’s profile	If you type /profile/username into the URL search bar, then you will get the profile of whatever username matches the user. Alternatively, you can go to the profile of any user that is on the Search page, Notification page, or Friends page. You can also go to a user’s profile by following the links on the profile page under Game History, or via the colored link above any game. If a user’s profile is marked as: • pub = Public

	◦ Any one can view the full profile (but are not allowed to edit it and can't see this person's friends unless it is their own profile). • fri = Friends Only ◦ Only the user's friends are allowed to see the full profile. Other users will see a smaller profile that does not include contact information either. • pri = Private ◦ No one except the user themselves can see their full profile ◦ Everyone else only sees the smaller profile that doesn't include contact information, friend information, or the edit drop down setting menu. • del = Delete ◦ Not implemented yet, but you will be able to delete users by setting the privacy level to del.
6. Remove a user from friend list	As mentioned prior, we can remove friends either via the Search Page or via the "Go To Your Friends Page" link on your profile. After you can see the searchable list, just click the appropriate unfriends button to remove the friend connection in both this user, and the user's friend.
7. Changing profile settings	You can change the profile settings by going to your profile and clicking the "Edit Profile" option in the top right drop-down settings menu. After going to the "Edit Profile" form/page adjust settings as needed and hit save, and it will take you back to newly updated profile page.
8. View current active games	Go to the home page (you should be redirected here after logging in), and then you should be able to see all of your current ongoing games here. You can navigate through your current games by clicking on the "next" and "prev" buttons, and the "Next Active" button will let you cycle through the games that are currently waiting for your move. You can search through all of the games on the server (past and present) by clicking on the "search games" option in the "Settings" drop-down menu of either the Home page or the Search page. Additionally, from this page you have two toggle options from the settings drop-down menu: 1. Toggle between displaying or hiding active games 2. Toggle between displaying or hiding completed games
9. Create a new random game	Go to the Home page and click on the "Start New Game With Random User" from the settings drop-down menu (top right). Then fill out the form, and when another user selects a game with the same privacy settings then a new game will be visible to you when you go to your home page (and if you go through your games search page too).

10. View history of played games	You can go to the Games Search page by going through the settings drop-down menu on either the Home Page or the Search page. Then you can search through the games (you will only have access to games that either you played in, or with settings making them visible to you such as games set to “public”, or “friends only” games belonging to users that you are friends with). When searching for games you can either search via a player’s name or by a game id.
Implementations under the Viewing a User Profile specification	
1. Summary of statistics	You can see this on the profile page of any user.
2. Summary of last 5 games	You can see this under the user’s profile under the Games History section.
3. See other user’s active games	This functionality is in the Games Search page as mentioned previously (num 10. in the prior section).
Implementations under the Playing Games specification	
The users can return to a game at any time, and each user can cycle through their current active games. Once a game is finished, the game doesn’t instantly leave the screen, but rather stays until the page is refreshed. Games can be forfeited, and the forfeiting user will lose that game (their score will be affected appropriately). In the case that a full turn has not yet occurred, the players can forfeit this game without affecting their score. The other player still wins the game, but it doesn’t affect their scores. There is a chat box next to the game updates at the same time as the game.	
Implementations under the REST API specification	
I implemented all of the REST API necessary for the specification of the project: **Get/users** includes the name(string) query parameter **GET/users/:user** includes the parameter user(string) **GET/games** includes the player(string), active(boolean), detail(full:summary)	

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

There are a few things I added which are not list in the specification of this project. These were:

- There is a custom 404 page. A custom 404 page is nicer than getting the same old 404 white page while testing out the server.

- I think that the overall look of the webpage is not too bad considering this is my first attempt at it. I find that the webpage itself feels reactive, and not overly busy. I think achieved a uniformity of design across the site, for the most part.
- I added a sanitize() function on the server that takes an array of user objects or a single user object and returns a sanitized version of the original object that does not include sensitive information of the user. These objects are then sent to other users when they need them (sanitized user objects no longer include passwords, and contact information).
- In the model.js I wrote a lot of error codes that are only printed out on the server level that return certain values that correspond to particular errors. For example, if the user that is requesting certain information is not a valid user, then often -1 is returned to the calling function.
- I did not compile the pug files as is customary to reduce latency when rendering pug files. Instead of using const pug = require('pug') in my server, I used app.set('view engine', 'pug') and it automatically caches rendered pug files after they are rendered with the res.render() method (i.e. cache = true by default in express's embedded version of pug) . I believe that this should mitigate the time penalty involve with compiling pug files into HTML strings.
- I added a FAQ page as I felt that was a nice to have on a site
- The site as a whole resizes when the user resizes the browser. The game page resizes well, and all of the mouse events based on x and y position inputs readjusts accordingly. The only time when HTML elements will start to overlap is when the user is using developer's tools in Chrome.

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.

In the model.js file I put a lot of effort into making it act as a layer of protection against data corruption, poor queries on the 'database', and improper requests. For the most part, on the server side there are error codes corresponding to improper requests that are logged out. For instance, if we are trying to access a user but misspelled the username, then the server should respond saying that a user with that username does not exist, rather than crashing. This definitely helps a lot in the development and testing stages of the server. Because of this, I think I did a fairly good job with making my server robust.

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.

There are a lot of different parts of the program that I would like to change if given more time, and even just with a bit more forethought.

- I would prefer to make more of my pug files and js files more generalized so that they could work across multiple pages. I ended up doing this with the Search page, friends page, and friend request page in that I they all use the same js file. However, there are other pages that I could do this same thing to, but did not. This would reduce code duplication and code size, making the server more maintainable and more efficient.
- Adding MongoDB and using Mongoose would be one big thing that I would have liked to change given more time. In this regard I would first have to make all of the functions asynchronous in the server. Next, I would revamp databasecontroller.js so that is actually interacted with Mongo. Then, I would add express sessions to the database, and start creating more user information by using the included usernames.txt and password.txt files and then start testing the efficiency of the server. I know at the moment some of my queries are not particularly quick and are more brute force than anything. I did not spend too much time on many of these queries as I knew that Mongoose would be eventually be replacing all of the querying functions in the databasecontroller.js file.
    - Also, after adding a database I would also have the ability to replace the loggedIn Set() that tracks who is logged in. Instead, I would be pulling information directly from the session information, which as far I can tell, is only possible when we can Store the session data (i.e. we can't pull all of the session data from each user's instance from RAM). This would allow me to reduce the cookie maxAge time as well so that it is more reasonable.
- Adding Nodemon for fast developing is starting to seem like a really great idea
- I have some commented out and some redundant code that I was using throughout various iterations, but as the project grew, I did not delete enough of it. I am worried of deleting too much code right before submitting though, so I deferred this to be done at after submitting when I would have more time to test the code. So sometimes you come upon a TODO that just says 'delete later'. This is what that refers to.
- I really like the 418 "I'm a teapot" HTTP status, and I got rid of most of them in preference of more descriptive codes, but I kept a few that enjoyed.
- For ease of developing, I have not yet added much enforcement onto password lengths. it is a lot easier to quickly type '1234' in the password section of the login page rather than typing 8 or more characters. It is pretty easy to change this, and this is another item I would change after implementing MongoDB and Mongoose onto the server. At that point I would start to use schemas (i.e. Mongoose schema to build a blueprint for what values a user object and game object can take on).
- I wouldn't have the client polling from the server as fast as it is doing right now. The fast polling just works really well for finding issues and developing quickly.

- I don’t like that when adding a game, the player not creating the game doesn’t have a say in joining the game. My way of sidestepping this is that if you forfeit a game before you play that forfeit does not count as a loss.
- Appearances (there are more than this but by the time that I type them all out I could have just started to implement a few):
    - I would add a “game was forfeited” in the Games Search page when the game was forfeited to make that clearer for the user
    - Add pictures, rather than having the picture as a placeholder. This is something that I was going to do after getting the database coded.
    - Remove more of the console.log information on the client side
    - Add some JS to the registration page so that if the passwords are not identical the user gets alerted before sending it to the server and getting an error page.
    - The red and yellow text at the top of each game is hard to read so I would like to add some shadow to it or black outlining to make it easier to read.
    - I do not like how the top notification that link to the user’s friend requests page looks, and would change it so that it is smaller (maybe with brackets with the number of notifications inside the brackets like in the profile), and so that it disappears when there are no notifications.

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

I did not use any particularly interesting modules/frameworks for my project outside of what was mentioned in class. These are what I used:

express session:

A very helpful, well tested, and quick form of authentication that uses cookies to track and authenticate users.

pug:

Instead of using HTML, I ended up using the template engine formally named jade, pug. Most notable, by using pug I can input variables when rendering the page to easily add unique user data to a web page. Additionally, we can make use of mixins, includes, loops, and conditionals by using pug to make the webpages appear dynamic.

model.js:

The Business logic of my server

path:

Used to look up local paths on a server.

express.static:

Very easily and quickly allows the user to server static resources. Much better and less repetitive than using dozens of routes.

express.urlencoded:

It is another express middleware that automatically parses the incoming requests. very similar to body-parser. For my use, it allowed me to use the req.body method when receiving a request.

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

I like that it works! But actually, this is by far the largest project I have accomplished. More seriously though, I am happy that I managed to make the website feel responsive, and dynamic. Additionally, while not perfect, I think that the site has the potential to actually look decent.

Before this term I had not used HTML, CSS, JS, Node.js, or pug, so I am happy that I got to develop so much in those languages. I now understand how much effort goes into designing and creating a website, and that if I were to do it again, I would change so much of how I approached the problem. It is not that I didn't use some good practices this time around, but more so that I had to try and create a website before I can understand how to create a website.