Gymbuds

Sebastian Pross - CMPS 450
Professor Miller
Spring 2024

Introduction

In simple terms, Gymbuds is a web application that allows users to track their time in the gym more easily. However, as someone who frequents the gym, tracking what you do isn't all that interesting. To make it more interesting and different from any other workout tracker, Gymbuds takes all of your workouts and gives a breakdown of everything you did in the past calendar week. This includes:

1. The amount of total time that they spent in the gym, as well as the total amount of time they spent being active, whether that is playing soccer, basketball, volleyball, cycling, or running.
2. Out of all of the exercises performed throughout the week, which muscle groups were targeted the most. Visualized by their percentages out of the total. For example, a user could log their workouts for the week and find out that 30% of their exercises targeted their chest, 20% their biceps, 15% their quadriceps, and so on. This also includes what sections of their body were trained most, either upper body or lower body, counting exercises that are total body as well. For example 40% total body, 30% upper body, and 30% lower body.
3. And finally, the longest amount of time they spent working out at the gym during that week. For example, a user could've worked out for 30 minutes more one day than all of the others, the recap would point to that time out.

To provide a little bit of background, it is important to pay homage to what gave me the idea to do this project. As many people who listen to music through streaming services such as Spotify know, at the end of the year users receive a recap of their streaming habits. For Spotify users, it is known as "Spotify Wrapped". One of the things that falls short for the yearly recap feature is that many people don't like to wait a whole year to know what their top artists are, or the amount of time they spend streaming music. Some people are a little bit impatient in that aspect, including me. A smaller company came up with a solution to that, allowing users to sign in with their Spotify or Apple Music account and add friends, allowing them to see what their friends are actively listening to, and providing the user with a weekly breakdown of their music streaming habits. That application is called "Airbuds," and immediately skyrocketed in popularity among all of the avid music listeners that I know. From that, stemmed the idea of someone doing the same with the gym. What if I knew exactly what I was doing each work, exactly what muscle groups were being activated, and exactly how much time I was actually spending at the gym? What about my friends? It would serve as a motivator in some ways.

Installation Instructions and Running the Website

As this is indeed a website, with a proper hosting service there isn't anything to download and install if anyone wants to use the service. However, as the project is being submitted without a proper hosting service, everything is done using the localhost function within your browser.

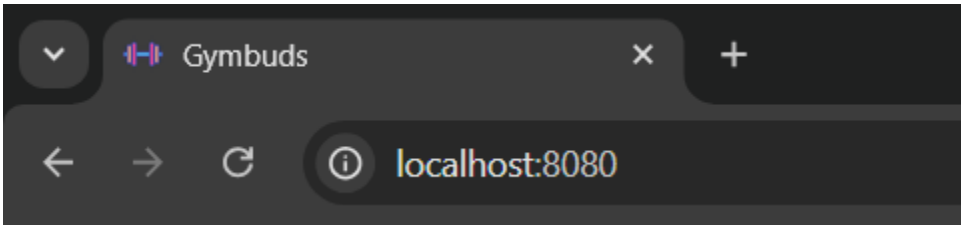
1. After downloading and unzipping the project file, as this project is dependent on Node Package Manager, once opened in your IDE, open the terminal and make sure the terminal is within the overarching project file.
    a. Then type `npm install`
        i.
```
PS C:\Senior-Project> npm install

up to date, audited 239 packages in 520ms

30 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Senior-Project>
```
        ii. If there are any issues with the install, most of the time it can be quickly resolved with `npm audit fix`
    b. To run the project, type `node ./gymbuds.js`
        i.
```
PS C:\Senior-Project> node ./gymbuds.js
Connected to database
Server is running on port 8080
Table created or already exists
Table created or already exists
Table created or already exists
Table created or already exists
Table created or already exists
Table created or already exists
Table created or already exists
Table created or already exists
```
        ii. You will know it is successful if you see the the output above in your terminal. Pay close attention to the "Server is running on port 8080". That is the port you will be running the website on in your local browser.

c. Now, for the website. Go to your local browser and type ‘localhost:8080’ into the URL. Note: the ‘8080’ is the local port the web server is running on.

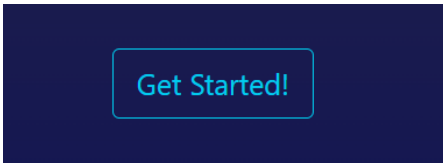
   i. 

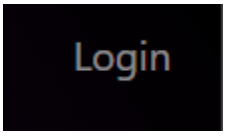
d. The code runs it on that port, but if you want to go into the code and change the port, it is possible. Just change the number in this section of the code in the `gymbuds.js` file.

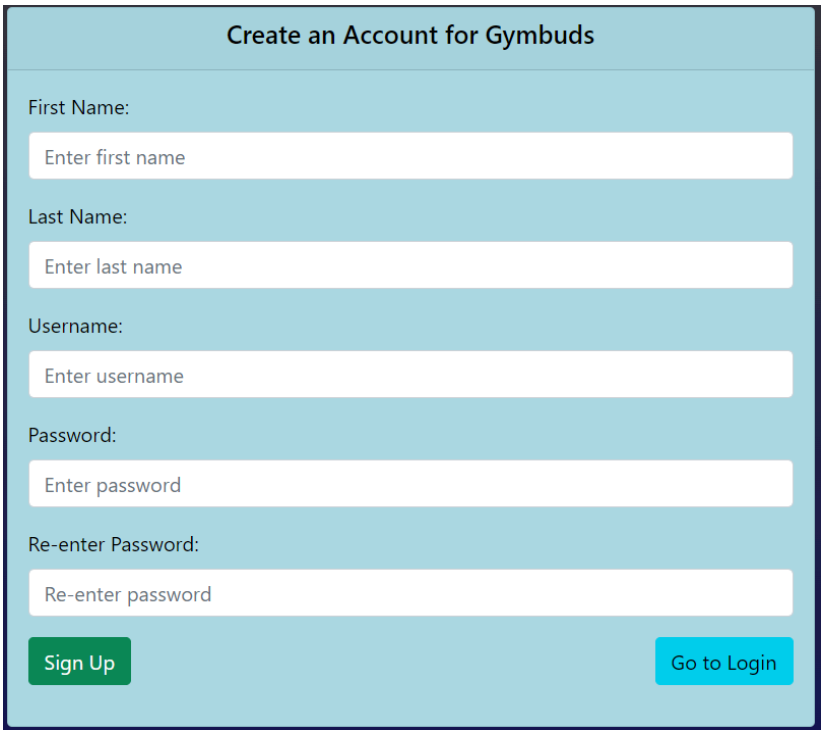
   i.
```
app.listen(8080, () => {
    console.log('Server is running on port 8080')
});
```

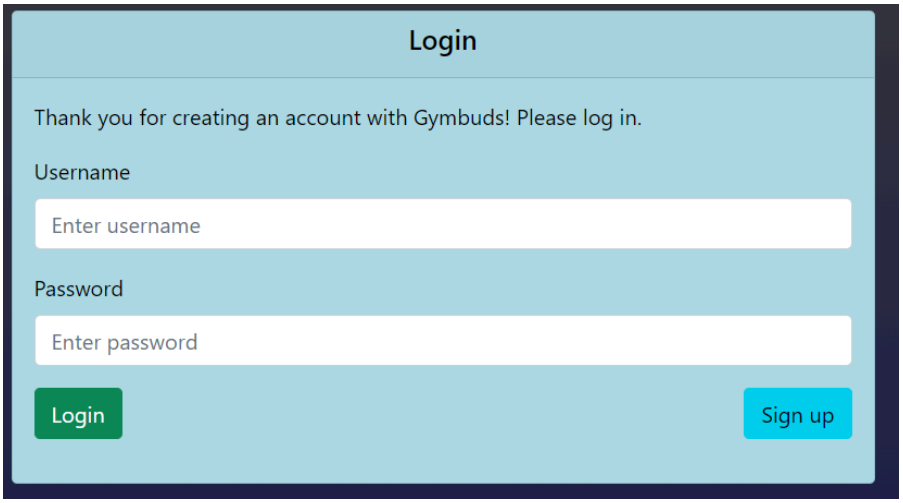
User Guide

Account Creation and Login

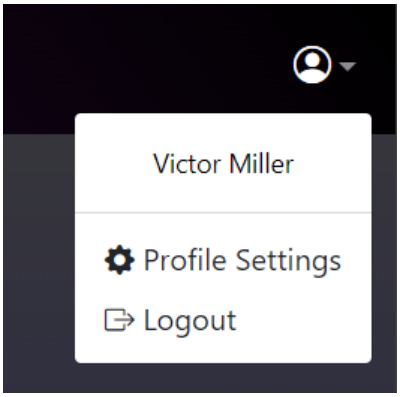
1. Once at the home page, aside from reading the beautifully written description of the website, the user can only access the website if they make an account. On the home page, there are two options to take the user to the login page, either using the navbar to click the 'Login' button on the top right, or clicking on the 'Get Started' button on the bottom of the page.
   a. 

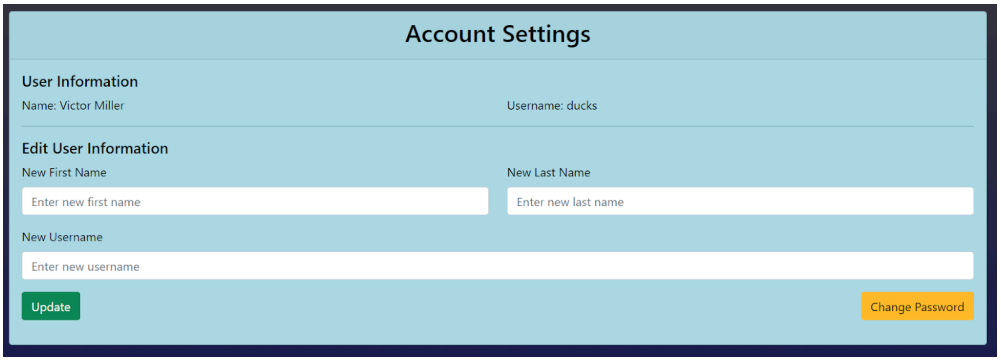
   b. 

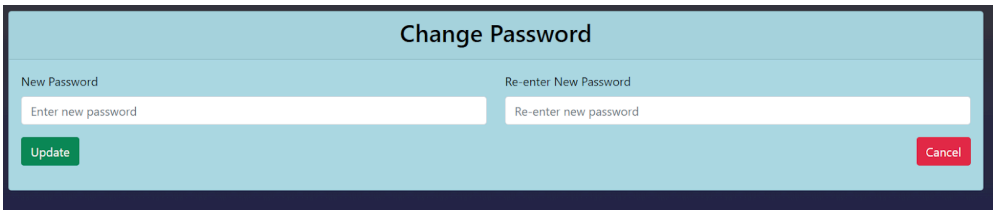
2. From here it is quite, straightforward. Either enter your login details or click on the button to sign up, bringing you to the sign up page.
3. If signing up, you will see this page. Just enter your details and make sure that you choose a strong password. Don't worry about username troubles, if your username is taken, it won't let you make an account under it (first come first serve!).
   a. 

4. Once you successfully create an account, you will be redirected to the login page where you can login with your newly created account.
   a. 

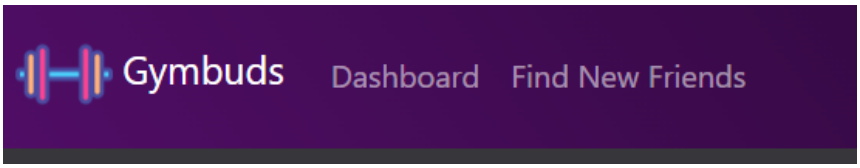
Account Settings

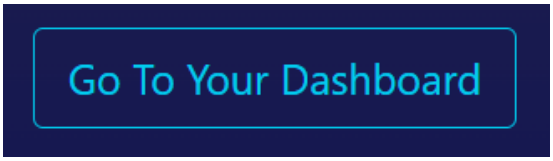
1. In the top right corner of the screen, the very right side of the navigation bar is an icon of a person, a 'profile' icon. Clicking on that icon will open up a small dropdown menu with a link to bring you to your profile settings. There is also a link to click if you want to log out of your account, simply clicking on it will log you out and bring you back to the home page.
   a. 

2. Now, once you are in your profile settings, you can type in new name details or a new username, and successfully update it by clicking on the 'Update' button in the bottom left hand corner.
   a. 

3. Changing your password is a different page, and can be accessed by clicking on the 'Change Password' button on the bottom right hand corner of the account settings menu. Hitting 'Cancel' will bring you back to the account settings page.
   a. 

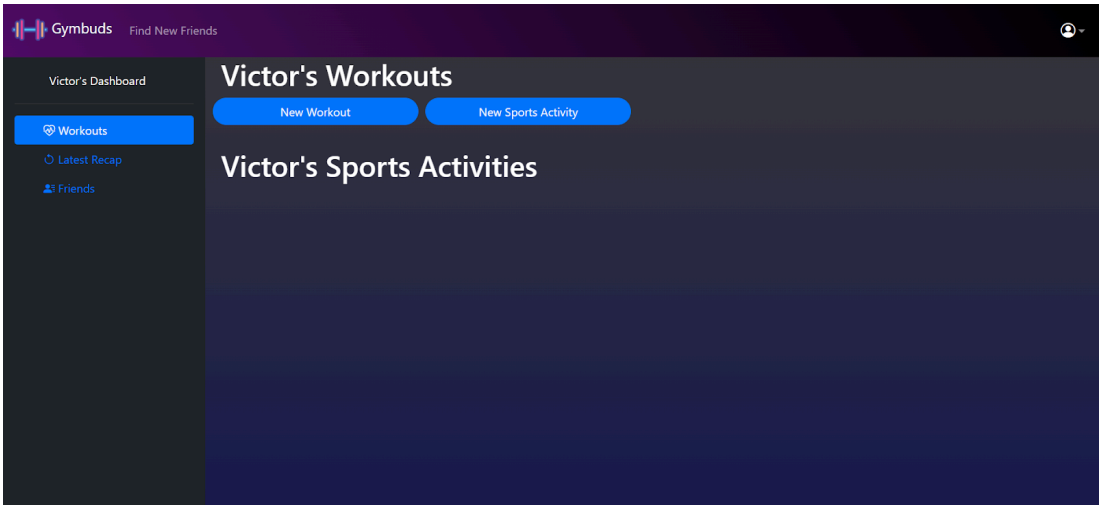
Navigating the Website

1. Now that you have an account, its time to start using the website! First step is to check out your dashboard, either click on the dashboard link in the navigation bar or use the button on the bottom of your page to go to your dashboard.
   a. 

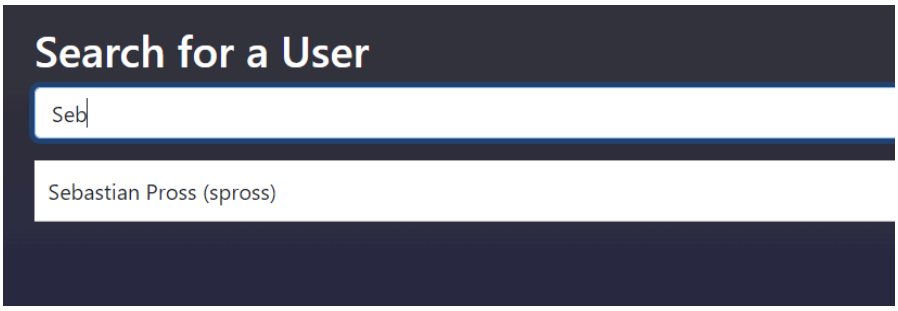
   b. 

2. It is important to note that if you want to **return to the home page** at any time, all you have to do is **click on the logo in** the top left corner of the screen on the navigation bar.
3. At the dashboard, there are a couple of options right now. Since the account is new, there aren't any workouts to edit or display. So that page is blank. On the left, you'll see three options: Your workouts, the recap of this week, and your friends. That is all of your dashboard navigation. Clicking on the "recap" link or "friends" link will be blank right

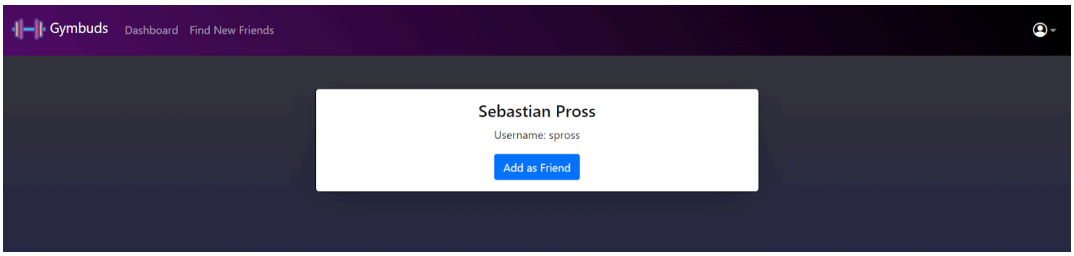
now, as your new account doesn't have any workouts to break down for the week and no friends added yet.

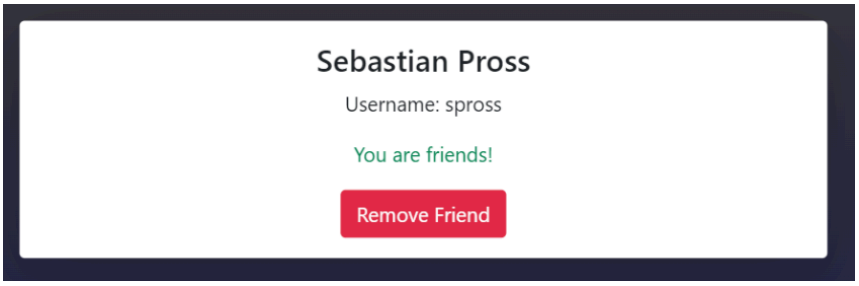
a. 

4. To add friends, all you have to do is click on the 'Find New Friends' link on the navigation bar, redirecting you to a webpage where you can search from all users and add people as a friend. The results will start out as nothing, but as you type a name or username it will fill the results with users with that name or username. And clicking on the name will bring you to their profile. For example:

a. 

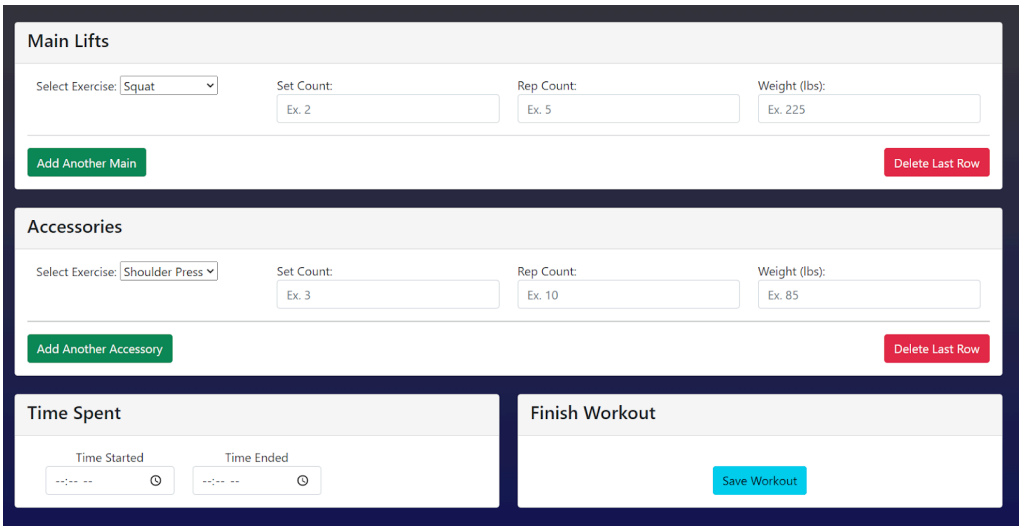
5. Once you click on their name, it will take you to their profile page where you can add them as a friend:

a. 

6. Once they are added, or if they are already added, the card will look like this:
   a. 

Workout Logging

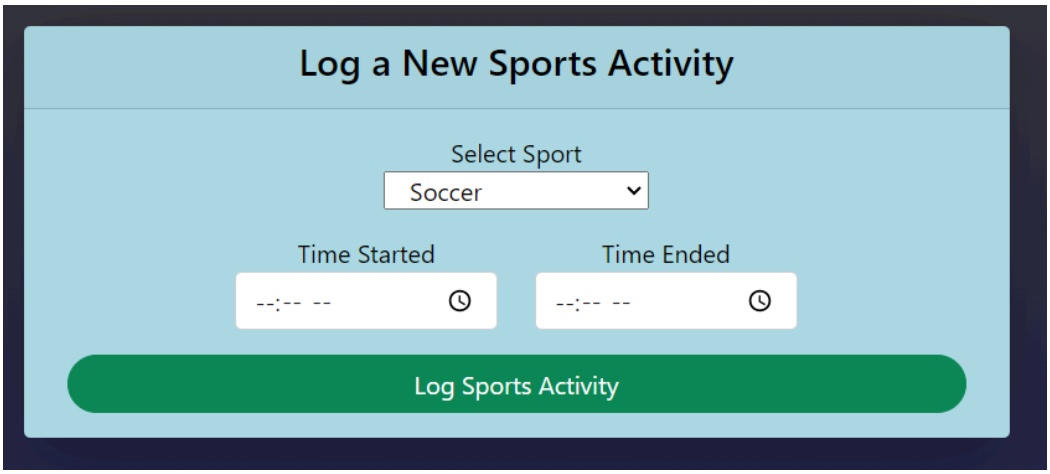
1. From the workouts page in your dashboard, there are two buttons on the top, labeled 'New Workout' and 'New Sports Activity'.
   a. 'New Workout' is for logging any workout you spent at the gym. Exercises and all.
   b. 'New Sports Activity' is for logging any time you spent being active outside of the gym i.e. playing basketball, volleyball, running, or even skiing. Just being active in general. This is just what you did and how long it was for.

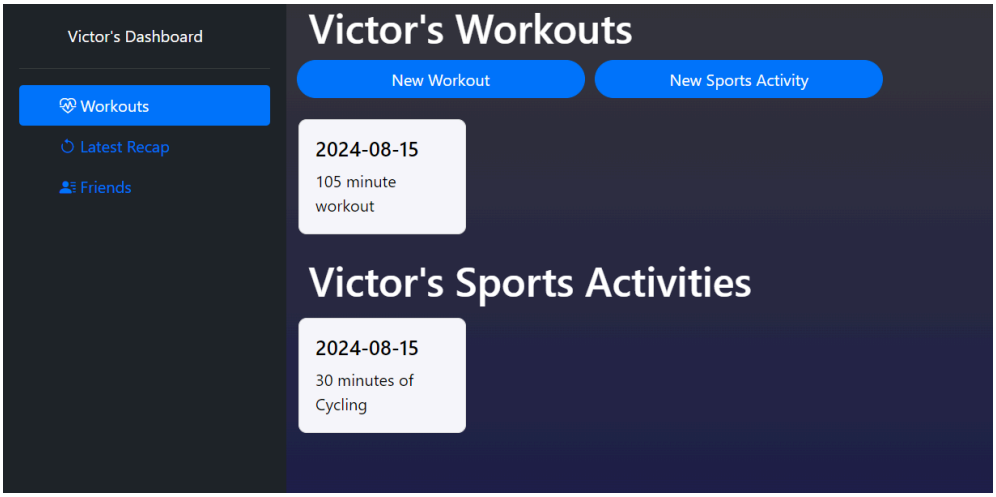
2. **Workouts**
   a. When it comes to logging a workout, it is quite simple. The top section contains where your "Main Lifts" will go. Main lifts are compound movements that use multiple muscle groups and require coordination. Those would be things like squats, benching, weightlifting movements like cleans and snatches, and many others. To choose which, simply click on the dropdown and choose from the many options. To the right of that, you can enter how many sets and reps, as well as the wight for those sets/reps.
   b. In order to add another one, simply press the "Add another main" button, and another blank entry will appear underneath.
   c. 

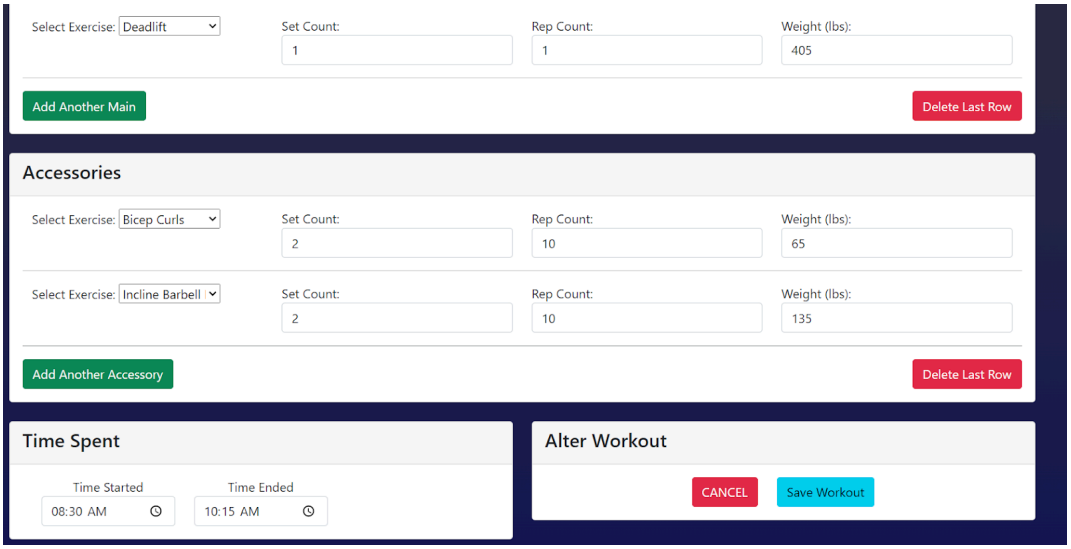
d. The function of accessories is the same, with accessories being exercises implemented as supplemental movements targeting specific muscle groups and helping weak areas, helping the 'main' lift movements. Things such as bicep curls, shoulder press variations, and tricep extensions.
    i. Adding and deleting these movements is the same as the mains.
e. On the bottom left, in an easy to see format, you can choose the time you began the lift and what time you stopped the lift, in order to track how much time you spent.
f. In the bottom right, you can simply press the "Save Workout" button, and it will save it to the database and redirect you to your dashboard.

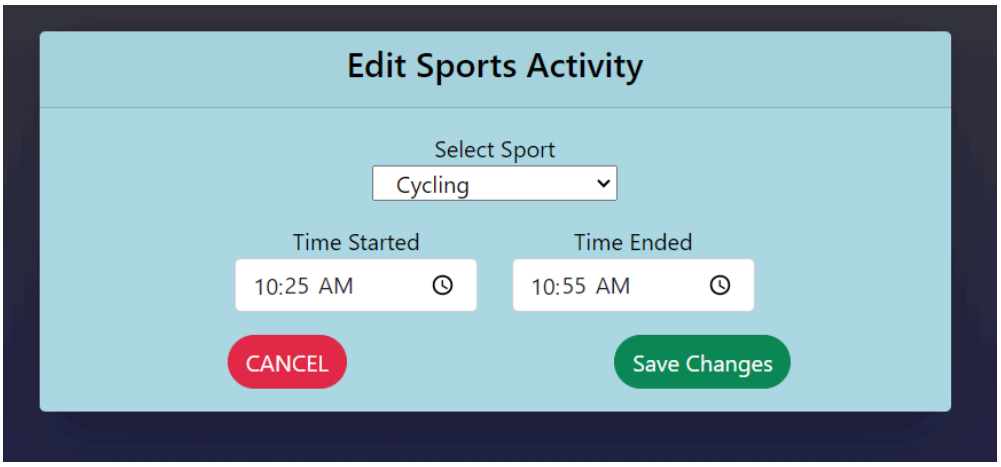
3. Sports Activities

a. These work very similarly to the workout logging, but there is a lot less to it. Simply choose what sport or activity you want to log from the drop down menu, and choose the start and end times in order for the website to calculate how long you spent!

b. 

4. **Editing Workouts/Activity Logs**
   a. Editing workouts and activity logs are quite simple! All you have to do is click on the workout or activity on your dashboard and it will load a page with all of the information already filled out and available to edit.
   b. 

   c. 

   d. 

Design Summary

As stated before and made very clear in the installation instructions and user guide (hopefully), Gymbuds is a web application powered with Express.js. All of the backend code is through javascript, and all of the front-end is generated with Pug and uses Jquery scripts. The database software is SQLite.

The high-level architecture of Gymbuds follows the classic three-tier model, which includes the front-end, back-end, and the database. Each layer has an important responsibility and interacts with the others to be able to support the application.

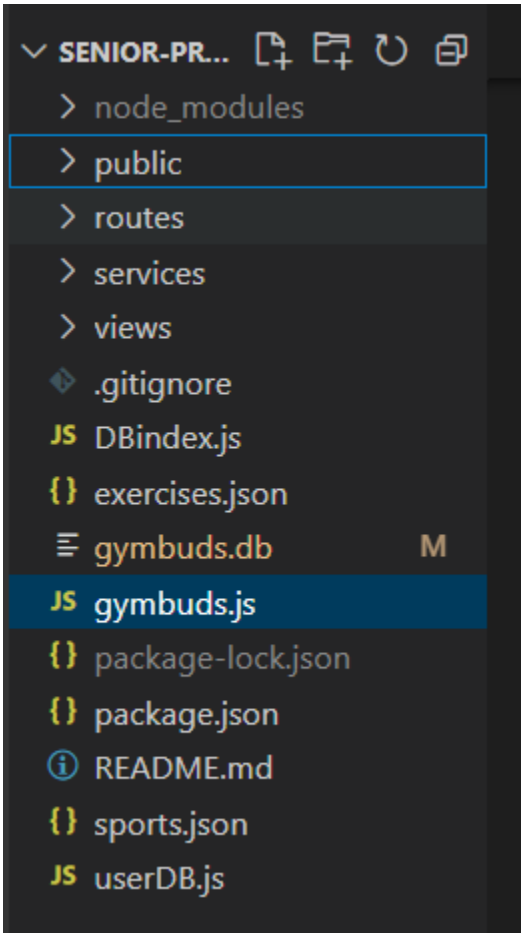
1. Front-End (The Client Side)
    a. The front-end is built using Pug as the templating engine, integrated with Bootstrap for styling and responsive design. It is responsible for rendering the user interface that users interact with, handling user inputs, such as the workout and sports activity logging, and adding or removing friends. These inputs are sent to the back-end via HTTP requests (GET and POST).
    b. When a user submits a workout log, the front-end collects all of the input data and sends it to the back-end via a form submission. The front-end then processes the reponses from the back-end to update the pages and user interface dynamically.
2. Back-End (The Server Side)
    a. The back-end is all handled using Express.js, which is a web application framework for Node.js, and handles the HTTPS requests, manages routes, and contains the 'business logic' of Gymbuds.
    b. In terms of 'business logic' the back-end validates user inputs, does the workout statistic calculations, and manages user authentication. It does all of the checks to make sure the data is correctly processed before it is sent to the front-end or stored/updated in the database.
    c. The back-end communicates directly with the SQLite database. When a user logs a workout, the back-end inserts the data from the workout into the appropriate tables. And when a user looks at the recap page, the back-end queries the database for the relevant data, processes it, and sends it back to the front-end.
3. Database (Data Storage)
    a. The database layer uses SQLite, which is described as a "lightweight, file-based relational database management system." It is simply a quicker and smaller version of MySQL.
    b. The SQLite database (gymbuds.db) stores all of the data for the website that needs to be persistent, which includes the user's information, the workout logs, and the exercises and sports that the users can choose from.
    c. The database is organized into tables that represent the different entities in the application such as 'Users', 'Workouts', 'UserExercises', and 'Friends'.

d. The back-end interacts with the database using SQL queries. For example, when a user logs a workout an INSERT query is executed to add the workout to the 'workouts' table and the exercises to the 'userExercises' table. When a user wants to see their weekly summary, SELECT queries are sent to the database to retrieve the necessary data and information.

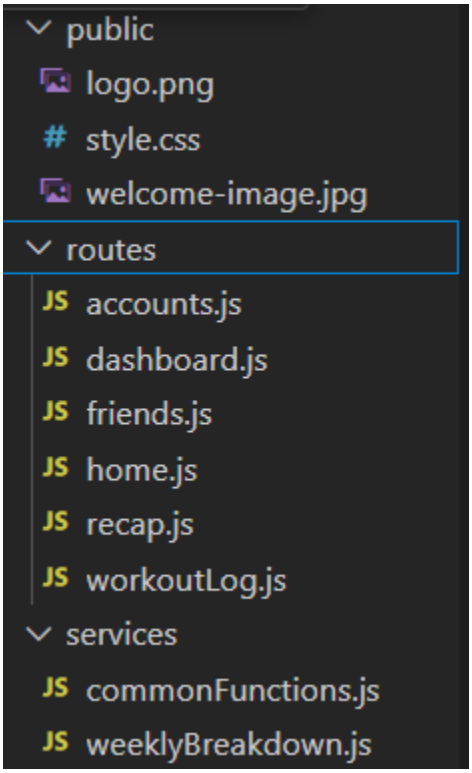
File Structure

Gymbuds is organized into a structured file system that separates the different functionalities of files. Below is the key directories and list of files:

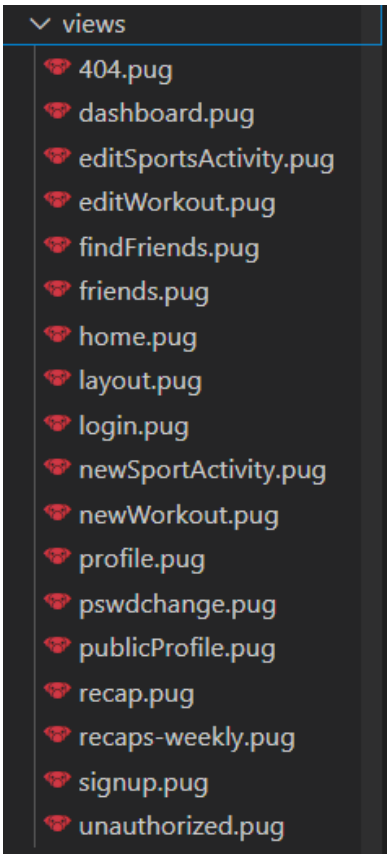
1. The overall folder structure:

a. 

2. Public, Routes, and Services

a. 

3. Views

a. 

The “Public” directory contains any specific CSS styling that is over-riding bootstrap, as well as stores any images that are rendered anywhere on the website.

The “Routes” directory plays a crucial role in organizing the different endpoints and routes of Gymbuds. These routes files determine how the application responds to the client requests for specific URLs. In simple terms, all of these files contain the ‘GET’ and ‘POST’ actions that the application uses to either render webpages or submit input data from webpages. This directory separates the website’s logic from everything else, making it easier to manage and scale. This is what the simplest of the routes files looks like:

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');

/*
    Sebastian Pross - Home

    HOME.JS

    This javascript page holds all of the backend functions pertaining to the home page.

    In order:

        router.get('/')
            --> This function calls the render function for the home page.

*/
//This function checks if the user is logged in (for authorization)
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}

//Home Page rendering
router.get('/', async(req, res) => {
    //Check if the session has a user logged in
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Render the home page with the active user information
    res.render('home', { user: user });
});

module.exports = router;
```

The “Views” directory contains all of the files that the rendering functions call. All of the pug files are the templates that generate the HTML sent to the client’s browser. These templates have a combination of dynamic content based on the data passed from the server (from the ‘res.render’ function shown in the screenshot above) and static content, making the web pages interactive and personalized. Each file is an individual page template the front-end uses in order for a user to access see the information necessary for the webpage.

Database Schema

All of the interactions with the database are through queries, and all of that is generated from the back-end logic. There are many different tables in the SQLite database for Gymbuds. Those being:

1. Exercises
    a. This table includes all of the exercises users can choose from when logging workouts, including information on the muscle groups they target.
2. Friends
    a. This table keeps track of all of the friend additions the users do. Taking the user_id key from the Users table and the friend_id key also from the Users table. Each addition to the table is unique.
        i. For example, when User id 7 adds user id 20 as a friend, it will make a new row with the two IDs paired together as 'user_id: 7' and 'friend_id: 20'.
3. SportActivity
    a. This table keeps track of all of the logged sports activities from users. It stores the date of the activity, the ID of the user whose activity it is logged, the name of the activity, the duration of it, and the start and end times.
4. Sports
    a. This table includes all of the sports the users can choose from when logging sports activities outside of the gym.
5. UserExercises
    a. This table keeps track of all of the exercises from user's workouts. It uses the ID number of the workout the exercise is from in order to keep them tied.
6. Users
    a. The Users table contains the information of all of the users for the website. Including their name, username, and a hashed version of their password.
7. Workouts
    a. This table keeps track of the workouts logged. It stores the date of the workout, the ID of the user whose workout it is, the duration of the workout, and the start/end times.

Data Structures

Gymbuds uses several data structures to manage and process the constantly updating data efficiently. For example, all of the information being passed through the routes to the pug templates, and through all of the database methods are objects or arrays of objects.

1. One example is the "User" object, which stores the information about each user. This is used for managing the user account, page access authorization, and user-specific operations like managing workouts and sports activities, and adding friends.

```
User = {
        id: 1,
        first_name: 'Sebastian'
        last_name: 'Pross'
        username: 'spross'
        Password: '1234'
}
```

Classes

Given that Gymbuds is a web application, there isn't many classes implemented, but the two that are implemented are very important. Those being the 'DataStore' and 'UserDB' classes.

1. **DataStore**
    a. This class is the lowest layer of the application when it comes to database connection. It encapsulates the SQLite database connection details and has all of the methods for generating and executing SQL queries. It is mainly for abstracting the database operations, making sure that all of the data-related functionality is in the same place and easily maintainable.
    b. All of the methods in the class take data and table names as parameters, and maps it to an SQL query. The main reason for this is because there are so many different query operations with different tables, it is a massive waste of time if every single call to the database was individualized from a higher level.
        i. For example: the update() method takes the table name, and the categories to change and updates the desired row in the database, mapping it to a string to pass to the database as a CRUD operation. Without this function, every single time an update call is needed the function calling for an update would have to pass an individualized string for a SQL query.
2. **UserDB**
    a. This class acts as a 'middleman' between the routes (GET and POST) and the 'DataStore' class. Initializing the database connection and containing methods called by the routes to interact with the database. With this as a middle-man it organizes the route logic from the database CRUD operations, making them cleaner and easier to read.
    b. An example of one of the methods would be getUserById(), where the route would call that function, passing in a User ID, and it would call the .read()

function, passing in the ID and the Users table name from ‘DataStore’ which maps the User ID and table name to an SQL query.

3. **How do These Classes Interact? Summary**
    a. ‘UserDB’ initializes the connection by creating an instance of the ‘DataStore’ class. By using the ‘UserDB’ class as an intermediary, the rout handling and data management maintain separation. The routes don’t need to know the specifics of the SQL queries, they simply call methods of the ‘UserDB’ class.

Test Plan and Test Results

The best way to test the program is by actually creating an account and using workout logging functionality. Throughout development, every feature was vigorously tested as it was added, to make sure it was properly working and could interact with the rest of the website smoothly. Oftentimes when a bug was found, it was logged into the console and was easy to find, and most of the time the express framework would log the error.

There were a few bugs found while finishing up the testing. One of the main ones was the recap page crashing the website if there was no recap results for the template to look at, which was quickly fixed. To my knowledge, there are not any large bugs that impact the user’s experience, purely design choices that could be changes.

Summary and Conclusions

Overall, the task of making the website was more complicated than I believed it would be, and I would approach it differently in many ways. In this project, I wanted to build something that someone could use to better understand what they were actually doing and achieving in the gym to a slightly deeper level, in a way to motivate them more or just interest them more in understanding the depths of what people do in the gym. There is still much more that can be done and much that can be refurbished and improved. My overall vision is a lot more ambitious and would be at a much larger scale and higher level.

Overall, the program and website works. The website is organized, and users can easily find other users and log what they do. The recap is detailed and shown well. But in the long term, the website will have trouble scaling, and I do believe that much more could be done in improving the user’s experience and ease of use. Another issue that arose is my skills with front-end development and making the screen reactive and scalable to different screens. Currently not all of the screens are well scaleable to smaller devices, and that is something that would be important with the much larger vision I have with this idea.

Bibliography

1. The inspiration for the project: Airbuds Widget developed by Capp Inc.
    a. They're just a small team of 5!
2. All of the bootstrap documentation and resources located at https://getbootstrap.com/docs/4.1/getting-started/introduction/
3. JavaScript documentation at https://developer.mozilla.org/en-US/
4. Stackoverflow: https://stackoverflow.com/
5. Scott Frees' Web Application Development Class
    a. Many of the programming methods used were learned in his course.

CODE

Accounts.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');


/*
    Sebastian Pross - Accounts

    ACCOUNTS.JS:

        This javascript page holds all of the backend functions pertaining
to anything related to a profile.
        That includes login, signup, profile editing, and password change,
and viewing a person's public profile page, as well as adding/removing
them as friends.

        In order:

            router.get(/logout)
                --> This function is purely for the 'logout' button, and
logs the user out of the session, redirecting them to the home page.


            router.get(/login)
                --> This function calls the render function for the login
page
            router.post(/login)
                --> This function posts the login page, taking the input
information and logging the user in.


            router.get(/signup)
                --> This function calls the render function for the signup
page
            router.post(/signup)
```

```
                --> This function posts the signup page, taking the input
information and creating a new account in the database, issuing an error
                    if the account already exists.


            router.get(/profile)
                --> This function calls the render function for the
profile viewing page
            router.post(/profile)
                --> This function posts the profile page, if the user
inputs a new name or email, and changeds it in the database


            router.get(/pswdchange)
                --> This function calls the render function for the
password changing page
            router.post(/pswdchange)
                --> This function posts the password change page, taking
the user's new password and changing it in the database

            router.get(/profile/username)
                --> This function calls the render function for the public
page to look at other people's profiles
            router.post(/profile/username)
                --> (Finish later) Post for profile


            router.post(/add-friend)
                --> This post function comes from the public profile page,
and calls the database to ADD two users as friends


            router.post(/remove-friend)
                --> This post function comes from the public profile page,
and calls the database to REMOVE two users as friends

*/

//This function checks if the user is logged in, redirecting to the
unauthorized page if they are not
```

```
//This is done for security reasons. If a user is not logged in, we don't
want them to be able to access certain pages
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}

//When a user clicks log out, the user session is abandoned and they are
redirected to the home page
router.get('/logout', async (req, res) => {
    req.session.user = undefined;
    res.redirect('/');
});

//Render the login page
router.get('/login', async (req, res) => {
    res.render('login', { hide_login: true });
});

//Login page functionality
router.post('/login', async (req, res) => {
    const username = req.body.username.trim();
    const p1 = req.body.password.trim();
    const user = await req.db.findUserByUsername(username);

    if (user && bcrypt.compareSync(p1, user.password)){
        req.session.user = user;

        //Note: currently redirects to home, but will instead redirect to
a dashboard
        res.redirect('/');
        return;
    }else{
        res.render('login', { hide_login: true, message: 'Either username
or password is incorrect.' });
        return;
    }
```

```
})

//Render the signup page
router.get('/signup', async (req, res) => {
    res.render('signup', { hide_login: true });
});

//Signup page functionality
router.post('/signup', async (req, res) => {
    //Retreive all the user's input information from the text boxes
    const first = req.body.first;
    const last = req.body.last;
    const username = req.body.username.trim();
    const password1 = req.body.password.trim();
    const password2 = req.body.password2.trim();

    //If the two passwords don't match, warn the user and make them
re-submit
    if (password1 != password2){
        res.render('signup', { hide_login: true, message: 'Passwords do
not match!' });
        return;
    }

    //Check if the user already exists and warn the user if it does
    const user = await req.db.findUserByUsername(username);
    if (user) {
        res.render('signup', { hide_login: true, message: 'This account
already exists!' });
        return;
    }

    //Generate a salt hash for the password for a base level of password
security
    const salt = bcrypt.genSaltSync(10);
    const hash = bcrypt.hashSync(password1, salt);

    //Create the user with the input information
    await req.db.createUser(first, last, username, hash);
```

```
    //This will be passed to the login page for it to know that the user
just created an account and show the correct text
    const fromSignup = true;

    //Redirect the user to the login page in order for them to login
    res.render('login', { fromSignup });
});

//Render the profile settings page
router.get('/profile', logged_in, async (req, res) => {
    //Check if the session has a user logged in
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Render the proile page with the active user information
    res.render('profile', { user: user });
});

//Profile settings page functionality
router.post('/profile', async (req, res) => {
    //Find the current user
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    let newFirst = req.body.first;
    let newLast = req.body.last;
    let newUsername = req.body.username.trim();

    //Check if the user already exists and warn the user if it does
    const userTest = await req.db.findUserByUsername(newUsername);
    if (userTest) {
        res.render('profile', { user: user, hide_login: true, message:
'This username already exists!' });
        return;
    }

    //This block checks to see if any of the post forms were empty, and
fills it with the old info to properly
    // call updateUser with all the information
    if(!newFirst){
```

```
        newFirst = user.first_name;
    }

    if(!newLast){
        newLast = user.last_name;
    }

    if(!newUsername){
        newUsername = user.username;
    }

    //Sends a request to the server to update the user with the new
information
    await req.db.updateUser(userId, newFirst, newLast, newUsername);

    //Gets the new user's information to re-render the page with the
updated information
    const updatedUser = await req.db.findUserById(userId);

    //Re-render the profile page with the updated information
    res.render('profile', { user: updatedUser });
});

//Render the password change page
router.get('/pswdchange', logged_in, async (req, res) => {
    //Check if the session has a user logged in
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Render the proile page with the active user information
    res.render('pswdchange', { user: user });
});

//Password change functionality
router.post('/pswdchange', async (req, res) => {
    //Find the current user
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    const password1 = req.body.password.trim();
```

```
    const password2 = req.body.password2.trim();

    //If the two passwords don't match, warn the user and make them
re-submit
    if (password1 != password2){
        res.render('pswdchange', { user: user, hide_login: true, message:
'Passwords do not match!', success: false });
        return;
    }

    //Generate a salt hash for the password for a base level of password
security
    const salt = bcrypt.genSaltSync(10);
    const hash = bcrypt.hashSync(password1, salt);

    await req.db.updateUserPassword(userId, hash);

    res.render('pswdchange', { user: user, message: "Password successfully
updated!", success: true })
});

//Render the public profile page
router.get('/u/:username', async (req, res) => {
    //Check if the session has a user logged in
    const userId = req.session.user ? req.session.user.id : -1;
    const activeUser = await req.db.findUserById(userId);

    const userPage = await req.db.findUserByUsername(req.params.username);

    let isFriend;

    if (userPage == null) {
        //If there is no user by that username we render the page with
that info to generate a warning
        res.render('publicProfile', { user: activeUser, userPage:
'undefined' });
    } else if(activeUser == null) {
        //Check to see if the user and the profile being looked at are
friends to show the proper page information
        isFriend = await req.db.checkFriendStatus(userId, userPage.id);
```

```
        //If the user isn't logged in we don't pass the user
        res.render('publicProfile', { userPage: userPage });

    } else {
        //Check to see if the user and the profile being looked at are
friends to show the proper page information
        isFriend = await req.db.checkFriendStatus(userId, userPage.id);

        //If there is a user with that username we generate their public
profile information
        res.render('publicProfile', { user: activeUser, userPage:
userPage, isFriend: isFriend });
    }
})

//Post from public profile to add a friend
router.post('/add-friend', async (req, res) => {
    const friend_id = req.body.friend_id;
    const user_id = req.body.user_id;

    await req.db.addFriend(user_id, friend_id);

    res.redirect('back');
})

//Post from public profile to remove a friend
router.post('/remove-friend', async (req, res) => {
    const friend_id = req.body.friend_id;
    const user_id = req.body.user_id;

    await req.db.removeFriend(user_id, friend_id);

    res.redirect('back');
})

module.exports = router;
```

Dashboard.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');

/*
    Sebastian Pross - Dashboard

    DASHBOARD.JS

    This javascript page holds all of the backend functions pertaining to
anything related to the dashboard pages.
        That includes the unauthorized page, and dashboard page.

    In order:

        router.get('/unauthorized')
            --> This function calls the render function for the
unauthorized page.

        router.get('/logout')
            --> This function is purely for the 'logout' button, and logs
the user out of the session, redirecting them to the home page.

        router.get('/dashboard')
            --> This functions calls the render function for the dashboard
page.


*/

//This function checks if the user is logged in (for authorization)
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}
```

```
//Unauthorized Page Render
router.get('/unauthorized', async (req, res) => {
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    res.render('unauthorized');
})

//When a user clicks log out, the user session is abandoned and they are
redirected to the home page
router.get('/logout', async (req, res) => {
    req.session.user = undefined;
    res.redirect('/');
});

//Render the main dashboard page
router.get('/dashboard', logged_in, async (req, res) => {
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //This retreives all of the user's workouts from the database in order
to list them for the user to see them
    let workouts = await req.db.getAllWorkouts(userId);

    //Make sure the workouts are sorted in ascending date.
    workouts.sort((a, b) => {
        const dateA = new Date(a.date);
        const dateB = new Date(b.date);
        return dateA - dateB;
    })

    //This retreives all of the user's logged sports activies from the
database in order to list them for the user to see them
    let sports = await req.db.getAllSportsActivity(userId);

    //Make sure the sports are sorted in ascending date.
    sports.sort((a, b) => {
        const dateA = new Date(a.date);
        const dateB = new Date(b.date);
```

```
        return dateA - dateB;
    })


    res.render('dashboard', { user: user, workouts: workouts, sports:
sports });
});


//Render the weekly recaps dashboard page
router.get('/recaps-weekly', logged_in, async (req, res) => {
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    res.render('recaps-weekly', { user: user })
})

module.exports = router;
```

Friends.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');

/*
    Sebastian Pross - Friends

    FRIENDS.JS

    This javascript page holds all of the backend functions pertaining to
anything related to the friends pages.
        That includes the friends page, as well as the friend finding
page.

    In order:

        router.get('/friends')
            --> This function gathers the necessary information and
                calls the render function for the friends page.

        router.get('/findFriends')
            --> This function gathers the necessary information and
                calls the render function for the findFriends page.

*/

//This function checks if the user is logged in, redirecting to the
unauthorized page if they are not
//This is done for security reasons. If a user is not logged in, we don't
want them to be able to access certain pages
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}
```

```
//Render the friend's recaps dashboard page
router.get('/friends', logged_in, async (req, res) => {
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Get the IDs of all the user's friends
    const friendsIds = await req.db.getAllFriends(userId);

    let friends = [];

    //Loop through the IDs and retrieve each user's information to pass on
to the html.
    for(const user of friendsIds){
        friends.push(await req.db.findUserById(user.friend_id));
    }

    res.render('friends', { user: user, friends: friends });
})


//Render the find new friends page
router.get('/findFriends', logged_in, async (req, res) => {
    const userId = req.session.user ? req.session.user.id: -1;
    const user = await req.db.findUserById(userId);

    //Get all of the users from the database
    const allUsers = await req.db.getAllUsers();

    res.render('findFriends', { user: user, allUsers: allUsers });
})

module.exports = router;
```

Home.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');

/*
    Sebastian Pross - Home

    HOME.JS

    This javascript page holds all of the backend functions pertaining to
the home page.

    In order:

        router.get('/')
            --> This function calls the render function for the home page.

*/

//This function checks if the user is logged in (for authorization)
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}

//Home Page rendering
router.get('/', async(req, res) => {
    //Check if the session has a user logged in
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Render the home page with the active user information
    res.render('home', { user: user });
});

module.exports = router;
```

Recap.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const { formatDate } = require('../services/commonFunctions');
const { calculateWeeklyBreakdown } =
require('../services/weeklyBreakdown'); // Imports the function to
generate the recap

/*
    Sebastian Pross - Recap

    RECAP.JS

    This javascript page holds all of the backend functions pertaining to
the recap page

    In order:
        router.get('/recap')
            --> This function gathers the necessary information and passes
it into the render
                function for the recap page.

*/

//This function checks if the user is logged in (for authorization)
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}

//Render the latest recap dashboard page
router.get('/recap', logged_in, async (req, res) => {
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);
```

```
    //Check which day it is
    const today = new Date();
    const isSunday = today.getDay() === 0;

    //Find the date of the last monday that passed and store it
    let lastMonday = new Date();
    const daysToSubtract = ((today.getDay() + 6) % 7);
    lastMonday.setDate(today.getDate() - daysToSubtract);

    lastMonday = formatDate(lastMonday);
    let formattedToday = formatDate(today);

    //Now we have the dates we are calculating the recap in, we can query
the database for everything within that range
    //Get the workouts for the week
    const week_workouts = await req.db.getAllWorkoutsForWeek(userId,
lastMonday, formattedToday);

    let workout_ids = [];

    //Fill the workout_ids array with all of the workout ids from this
week's workouts
    for(const workout of week_workouts){
        workout_ids.push(workout.id);
    }

    //Get all of the exercises from the workouts for the week
    let week_exercises = []
    for(var i = 0; i < workout_ids.length; i++){
        week_exercises.push(await
req.db.getAllWorkoutExercises(workout_ids[i]));
    }

    //"Flatten" the week_exercises array so its just an array of objects,
not an "array of object arrays"
    week_exercises = week_exercises.flat();

    //Get all of the sport activities for the week
    const week_sportActivities = await req.db.getAllSportsForWeek(userId,
lastMonday, formattedToday);
```

```
    //Get the table of available exercises from the database to pass into
the breakdown function
    // so it can see muscle groups and workout categories
    const exerciseTable = await req.db.getExercises();

    let noWorkouts = false;
    if(week_workouts.length == 0){
        noWorkouts = true;
    }

    //Take all of the information gathered and pass it into the function
to generate the weekly recap
    const recap = calculateWeeklyBreakdown(userId, week_workouts,
week_exercises, week_sportActivities, exerciseTable);

    res.render('recap', { user: user, recap: recap, noWorkouts: noWorkouts
});
})

module.exports = router;
```

WorkoutLog.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const { formatDate, calcDuration } =
require('../services/commonFunctions');

/*
    Sebastian Pross - workoutLog

    WORKOUTLOG.JS

    This javascript page has all backend functions pertaining to workout
logging.
        This includes the workout creation page, workout editing page,
sports activity log page,
        and editting sports activity page.

    In order:

        router.get('/:id/newWorkout')
            --> This function calls the render function for a new workout
page
        router.post('/:id/newWorkout')
            --> This function takes all of the information submitted in
the new workout page and calls functions
                to add a workout to the database

        router.get('/workouts/:id')
            --> This function retrieves all of the workout information and
renders a page to edit the workout
        router.post('/workouts/:id')
            --> This function takes all of the changed information
submitted in the workout editting page
                and calls functions to update the workout.

        router.get('/:id/newSportsActivity')
            --> This function calls the render function for a new sports
activity log page
        router.post('/:id/newSportsActivity')
```

```
            --> This function takes all of the information submitted in
the new activity page and calls functions
                to add the activity to the database

        router.get('/sportsActivities/:id')
            --> This function retrieves all of the specified activity
information and renders a page to edit the activity
        router.post('/sportsActivities/:id')
            --> This function takes all of the changed information
submitted in the sports activity editting
                page and calls function to update the activity entry.

*/

//This function checks if the user is logged in (for authorization)
const logged_in = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/unauthorized');
    }
}

//Unauthorized Page Render
router.get('/unauthorized', async (req, res) => {
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    res.render('unauthorized');
})

//When a user clicks log out, the user session is abandoned and they are
redirected to the home page
router.get('/logout', async (req, res) => {
    req.session.user = undefined;
    res.redirect('/');
});

//Renders a blank workout logging page
router.get('/:id/newWorkout', logged_in, async (req, res) => {
```

```
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Retreive all of the possible exercises from the exercises table
    const exercises = await req.db.getExercises();

    res.render('newWorkout', { user: user, exercises: exercises });
});

//Workout page functionality (saving, etc)
router.post('/:id/newWorkout', async (req, res) => {

    /*
        When a workout is saved/submitted, an ID is created for said
workout, then each exercise is individually saved
        into a table with the workout id, and the exercise id
        (exercise id is linked with the id from the Exercises table)

        That way, if someone wants to re-open a workout, it searches
through the table of workout exercises and finds all with the
corresponding
        workout ID.
    */

    //Get the current date (for the workout entry)
    const currentDate = new Date()

    //Format the date
    const formattedDate = formatDate(currentDate);

    //Get the duration of the workout (for the workout duration entry)
    const startTimeStr = req.body.startTime;
    const endTimeStr = req.body.endTime;

    //Calculate Duration
    const workoutDuration = calcDuration(startTimeStr, endTimeStr);

    //Get the current user id (for the workout entry)
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);
```

```
    //Creates a new workout table entry and returns the workout ID
    const workoutId = await req.db.createWorkout(userId, formattedDate,
startTimeStr, endTimeStr, workoutDuration);


//-----------------------------------------------------------------------
------------------

    //Right here will be a loop to take all of the exercises from the page
and create entries in the "UserExercises" table
    //This is done here specifically at this point in the post function
because the workout id needs to be known already to properly store the
user's exercises.

    //For the mainRowContainer, we first need to get the amount of rows
    const mainRowCount = req.body.mainRowCount;

    //Starts with nothing, then adds one to each, so we begin with the
very first row
    const main_exerciseName = req.body.m_exercise_dropdown;
    const main_exerciseSets = req.body.m_sets;
    const main_exerciseReps = req.body.m_reps;
    const main_exerciseWeight = req.body.m_weight;

    const main_string = "Main";

    //As long as one of the fields are filled, we log the exercise, if
none of them are filled the exercise doesn't get logged. This will be the
same in the loop
    if (main_exerciseSets != '' || main_exerciseReps != '' ||
main_exerciseWeight != ''){
        const firstMain = await req.db.addUserExercise(workoutId,
main_exerciseName, main_string, main_exerciseSets, main_exerciseReps,
main_exerciseWeight);
    }
    //Loop through each main row and add the exercises
    for(var i = 1; i < mainRowCount; i++){
        var loop_main_exerciseName = req.body[`m_exercise_dropdown${i}`];
        var loop_main_exerciseSets = req.body[`m_sets${i}`];
```

```
        var loop_main_exerciseReps = req.body[`m_reps${i}`];
        var loop_main_exerciseWeight = req.body[`m_weight${i}`];

        if (loop_main_exerciseSets != '' || loop_main_exerciseReps != ''
|| loop_main_exerciseWeight != ''){
            const mainExercise = await req.db.addUserExercise(workoutId,
loop_main_exerciseName, main_string, loop_main_exerciseSets,
loop_main_exerciseReps, loop_main_exerciseWeight);
        }
    }


//-----------------------------------------------------------------------
------------------

    //For the accessoryRowContainer, we loop through all the rows
    const accessoryRowCount = req.body.accessoryRowCount;

    //Starts with nothing, then adds one to each, so we begin with the
very first row
    var accessory_exerciseName = req.body.a_exercise_dropdown;
    var accessory_exerciseSets = req.body.a_sets;
    var accessory_exerciseReps = req.body.a_reps;
    var accessory_exerciseWeight = req.body.a_weight;

    const accessory_string = "Accessory";

    //As long as one of the fields are filled, we log the exercise, if
none of them are filled the exercise doesn't get logged. This will be the
same in the loop
    if (accessory_exerciseSets != '' || accessory_exerciseReps != '' ||
accessory_exerciseWeight != ''){
        const firstAccesory = await req.db.addUserExercise(workoutId,
accessory_exerciseName, accessory_string, accessory_exerciseSets,
accessory_exerciseReps, accessory_exerciseWeight);
    }

    //Loop through each accessory row and add the exercises
    for(var i = 1; i < accessoryRowCount; i++){
```

```
        var loop_accessory_exerciseName =
req.body[`a_exercise_dropdown${i}`];
        var loop_accessory_exerciseSets = req.body[`a_sets${i}`];
        var loop_accessory_exerciseReps = req.body[`a_reps${i}`];
        var loop_accessory_exerciseWeight = req.body[`a_weight${i}`];

        if (loop_accessory_exerciseSets != '' ||
loop_accessory_exerciseReps != '' || loop_accessory_exerciseWeight != ''){
            var accesoryExercise = await req.db.addUserExercise(workoutId,
loop_accessory_exerciseName, accessory_string,
loop_accessory_exerciseSets, loop_accessory_exerciseReps,
loop_accessory_exerciseWeight);
        }
    }

    res.redirect('/dashboard');
});

//This renders the workout editting page
router.get('/workouts/:id', logged_in, async (req, res) => {
    //In this router.get we need to do something extra instead of just
rendering something, which is double checking that the user ID matches the
user id that is connected to the workout
    // This is done to ensure that a user can only edit their own
workouts.

    //Get the user id from the database
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    //Find the workout from the database based on the ID in the URL
    const workoutId = req.params.id;
    const workout = await req.db.findWorkoutById(workoutId);

    //Retreive all of the possible exercises from the exercises table
    const exercises = await req.db.getExercises();

    //Store the start and end times to pass into the render call
    const startTime = workout.start_time;
    const endTime = workout.end_time;
```

```
    //Check the user's id matches with the workout's user id to ensure
that the workout they are trying to view is in fact theirs.
    if(workout.user_id == userId){
        //Retreive all of the exercises that have the workout and user id
from userExercises if they match
        const userExercises = await
req.db.getAllWorkoutExercises(workoutId);

        //Variables to store the amount of main and accessories.
        var m_count = 0;
        var a_count = 0;

        //Loop through userExercises to find how many mains and
accessories are in the workout to pass to the page
        for(const exercise of userExercises){
            if(exercise.classification === 'Main'){
                m_count++;
            } else if (exercise.classification === 'Accessory'){
                a_count++;
            }
        }

        //Render the edit page
        res.render('editWorkout', { user: user, workout: workout,
exercises: exercises, userExercises: userExercises, startTime: startTime,
endTime: endTime, m_count: m_count, a_count: a_count });
    } else {
        //Render unauthorized if they don't match
        res.render('unauthorized', { userUnauthorized: true });
    }
});

//Workout editing page functionality/post
router.post('/workouts/:id', async (req, res) => {
    //We don't need the get the current date here because that is already
saved, so it doesn't need to be touched.

    //Get the duration of the workout (for the workout duration entry)
    const startTimeStr = req.body.startTime;
```

```
    const endTimeStr = req.body.endTime;

    //Calculate Duration
    const workoutDuration = calcDuration(startTimeStr, endTimeStr);

    //Get the current user id (for the workout entry)
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    //Now we need to update/change the workout saved by getting the
workout ID from the URL and calling a function to update the workout table
entry
    const workoutId = req.body.workoutId;
    const edittedWorkoutId = await req.db.updateWorkout(workoutId,
startTimeStr, endTimeStr, workoutDuration);

    //The next step is to update all of the exercises that were changed.

    /*
        Strategy: Loop through the rows in the container. If there is an
ID associated with that row, then we update with that ID.
        Else, we create a new entry.

        Check 'm_exercise_id${rowCount}`, and if the value is a number,
that is the exercise ID to edit, and if it is 'null' (meaning it is a new
addition), then we make a new table entry.
    */


//-----------------------------------------------------------------------
-----------------------------------------------------------------------
-------

    //For the mainRowContainer, we first need to get the amount of rows
    const mainRowCount = req.body.mainRowCount;

    //Starts with nothing, then adds one to each, so we begin with the
very first row
    const main_exerciseID = req.body.m_exercise_id;
    const main_exerciseName = req.body.m_exercise_dropdown;
```

```
    const main_exerciseSets = req.body.m_sets;
    const main_exerciseReps = req.body.m_reps;
    const main_exerciseWeight = req.body.m_weight;

    var exerciseID_list = [];

    const main_string = "Main";

    //Check if the first exercise has an ID
    if(main_exerciseID != 'null'){
        exerciseID_list.push(main_exerciseID);
        //If it does, we update the entry in the table

        //As long as one of the fields are filled, we log the exercise, if
none of them are filled the exercise doesn't get logged. This will be the
same in the loop
        if (main_exerciseSets != '' || main_exerciseReps != '' ||
main_exerciseWeight != ''){
            const firstMain = await
req.db.updateUserExercise(main_exerciseID, workoutId, main_exerciseName,
main_string, main_exerciseSets, main_exerciseReps, main_exerciseWeight);
        }

    } else {
        //If it does not, we create a new entry
        if (main_exerciseSets != '' || main_exerciseReps != '' ||
main_exerciseWeight != ''){
            const firstMain = await req.db.addUserExercise(workoutId,
main_exerciseName, main_string, main_exerciseSets, main_exerciseReps,
main_exerciseWeight);
            exerciseID_list.push(firstMain.id);
        }
    }


    //Now the first one is handled, we loop through each main row and
update or add the exercises
    for(var i = 1; i < mainRowCount; i++){
        var loop_main_exerciseID = req.body[`m_exercise_id${i}`];
        var loop_main_exerciseName = req.body[`m_exercise_dropdown${i}`];
```

```
        var loop_main_exerciseSets = req.body[`m_sets${i}`];
        var loop_main_exerciseReps = req.body[`m_reps${i}`];
        var loop_main_exerciseWeight = req.body[`m_weight${i}`];

        if(loop_main_exerciseID != 'null'){
            //Append to the list of exercise IDs (for deletion checking
later)
            exerciseID_list.push(loop_main_exerciseID);

            //If there is a value, we update the entry

            //As long as one of the fields are filled, we log the
exercise, if none of them are filled the exercise doesn't get logged. This
will be the same in the loop
            if (loop_main_exerciseSets != '' || loop_main_exerciseReps !=
'' || loop_main_exerciseWeight != ''){
                const mainExercise = await
req.db.updateUserExercise(loop_main_exerciseID, workoutId,
loop_main_exerciseName, main_string, loop_main_exerciseSets,
loop_main_exerciseReps, loop_main_exerciseWeight);
            }

        } else {
            //If it does not, we create a new entry
            if (loop_main_exerciseSets != '' || loop_main_exerciseReps !=
'' || loop_main_exerciseWeight != ''){
                const mainExercise = await
req.db.addUserExercise(workoutId, loop_main_exerciseName, main_string,
loop_main_exerciseSets, loop_main_exerciseReps, loop_main_exerciseWeight);
                exerciseID_list.push(mainExercise.id);
            }
        }
    }


//-----------------------------------------------------------------------
-----------------------------------------------------------------------
-------

    //For the accessoryRowContainer, we loop through all the rows
```

```
    const accessoryRowCount = req.body.accessoryRowCount;

    //Starts with nothing, then adds one to each, so we begin with the
very first row
    const accessory_exerciseID = req.body.a_exercise_id;
    var accessory_exerciseName = req.body.a_exercise_dropdown;
    var accessory_exerciseSets = req.body.a_sets;
    var accessory_exerciseReps = req.body.a_reps;
    var accessory_exerciseWeight = req.body.a_weight;

    const accessory_string = "Accessory";

    //Check if the first exercise has an ID
    if(accessory_exerciseID != 'null'){
        exerciseID_list.push(accessory_exerciseID);
        //If it does, we update the entry in the table

        //As long as one of the fields are filled, we log the exercise, if
none of them are filled the exercise doesn't get logged. This will be the
same in the loop
        if (accessory_exerciseSets != '' || accessory_exerciseReps != ''
|| accessory_exerciseWeight != ''){
            const firstAccesory = await
req.db.updateUserExercise(accessory_exerciseID, workoutId,
accessory_exerciseName, accessory_string, accessory_exerciseSets,
accessory_exerciseReps, accessory_exerciseWeight);
        }


    } else {
        //If it does not, we create a new entry
        if (accessory_exerciseSets != '' || accessory_exerciseReps != ''
|| accessory_exerciseWeight != ''){
            const firstAccesory = await req.db.addUserExercise(workoutId,
accessory_exerciseName, accessory_string, accessory_exerciseSets,
accessory_exerciseReps, accessory_exerciseWeight);
            exerciseID_list.push(firstAccesory.id);
        }
    }
```

```
    //Now that the first one is handled, we loop through each accessory
row and update or add the exercises.

    //Loop through each accessory row and add the exercises
    for(var i = 1; i < accessoryRowCount; i++){
        var loop_accessory_exerciseID = req.body[`a_exercise_id${i}`];
        var loop_accessory_exerciseName =
req.body[`a_exercise_dropdown${i}`];
        var loop_accessory_exerciseSets = req.body[`a_sets${i}`];
        var loop_accessory_exerciseReps = req.body[`a_reps${i}`];
        var loop_accessory_exerciseWeight = req.body[`a_weight${i}`];

        if(loop_accessory_exerciseID != 'null'){
            //Append to the list of exercise IDs (for deletion checking
later)
            exerciseID_list.push(loop_accessory_exerciseID);

            //If there is a value, we update the entry
            if (loop_accessory_exerciseSets != '' ||
loop_accessory_exerciseReps != '' || loop_accessory_exerciseWeight != ''){
                const accesoryExercise = await
req.db.updateUserExercise(loop_accessory_exerciseID, workoutId,
loop_accessory_exerciseName, accessory_string,
loop_accessory_exerciseSets, loop_accessory_exerciseReps,
loop_accessory_exerciseWeight);
            }


        } else {
            //If it does not, we create a new entry
            if (loop_accessory_exerciseSets != '' ||
loop_accessory_exerciseReps != '' || loop_accessory_exerciseWeight != ''){
                const accesoryExercise = await
req.db.addUserExercise(workoutId, loop_accessory_exerciseName,
accessory_string, loop_accessory_exerciseSets,
loop_accessory_exerciseReps, loop_accessory_exerciseWeight);
                exerciseID_list.push(accesoryExercise.id);
            }
        }
    }
```

```
    //The last thing we need to do is check for any deleted exercises from
the workout.

    //Get all exercise IDs that have this workout ID.
    const allExercises = await req.db.getAllWorkoutExercises(workoutId);
    var databaseIDs_list = [];

    for(const exercise of allExercises){
        databaseIDs_list.push(exercise.id);
    }

    //If the exercises in the list don't correspond with the exercises
from the database, we delete the ones that aren't in the list
    var idFound
    //Loop through databse IDs
    for(var i = 0; i < databaseIDs_list.length; i++){
        idFound = false
        //loop through exercise IDs
        for(var j = 0; j < exerciseID_list.length; j++){
            //If the database ID is found in the exercise list we exit the
inner loop to check the next ID
            if(databaseIDs_list[i] == exerciseID_list[j]){
                idFound = true;
                j = exerciseID_list.length;
            }
        }
        //Check at the end if the ID was found. If it is in the database,
but not the user list, that means they deleted it.
        if(idFound == false){
            //Call the delete function here
            await req.db.deleteUserExercise(databaseIDs_list[i]);
        }
    }

    res.redirect('/dashboard');
});

//Renders the page to log a sport activity (non-gym activity)
router.get('/:id/newSportsActivity', logged_in, async (req, res) => {
```

```
    const userId = req.session.user ? req.session.user.id : -1;
    const user = await req.db.findUserById(userId);

    //Retrieve all of the possible sports from the sports table
    const sports = await req.db.getSports();

    res.render('newSportActivity', { user: user, sports: sports });
})

//Sport activity page functionality (saving, etc)
router.post('/:id/newSportsActivity', async (req, res) => {

    //Get the current date (for the sports activity entry)
    const currentDate = new Date()

    //Format the date
    const formattedDate = formatDate(currentDate);

    //Get the duration of the workout (for the duration entry)
    const startTimeStr = req.body.startTime;
    const endTimeStr = req.body.endTime;

    //Calculate Duration
    const activityDuration = calcDuration(startTimeStr, endTimeStr);

    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    //Retreive the sport the person is logging
    const sport = req.body.sport_dropdown;

    //Creates a new sport activity entry and returns the generated ID.
    const activityId = await req.db.createSportsActivity(userId, sport,
formattedDate, activityDuration, startTimeStr, endTimeStr);

    res.redirect('/dashboard');
})

//Renders the page to edit a sport activity
router.get('/sportsActivities/:id', logged_in, async (req, res) => {
```

```
    //In this router.get we need to do something extra instead of just
rendering something, which is double checking that the user ID matches the
user id that is connected to the sports activity
    // This is done to ensure that a user can only edit their own sports
activities.

    //Get the user id from the database
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    //Find the workout from the database based on the ID in the URL
    const activityId = req.params.id;
    const sportsActivity = await
req.db.findSportsActivityById(activityId);

    //Retrieve all of the possible sports from the sports table
    const sports = await req.db.getSports();

    //Store the start and end times to pass into the render call
    const startTime = sportsActivity.start_time;
    const endTime = sportsActivity.end_time;

    //Check the user's id matches with the sport activity's user id to
ensure that the activity they are trying to view is in fact theirs.
    if(sportsActivity.user_id == userId){
        //Render the edit page
        res.render('editSportsActivity', { user: user, sports: sports,
sportsActivity: sportsActivity, startTime: startTime, endTime: endTime})
    } else {
        //Render unauthorized if they don't match
        res.render('unauthorized', { userUnauthorized: true });
    }
})

//Sport activity edit page functionality (saving, etc)
router.post('/sportsActivities/:id', async (req, res) => {
    //We don't need the get the current date here because that is already
saved, so it doesn't need to be touched.

    //Get the duration of the workout (for the workout duration entry)
```

```
    const startTimeStr = req.body.startTime;
    const endTimeStr = req.body.endTime;

    //Calculate Duration
    const activityDuration = calcDuration(startTimeStr, endTimeStr);

    //Get the current user id (for the workout entry)
    const userId = req.session.user.id;
    const user = await req.db.findUserById(userId);

    //Get the sports activity id from the url
    const activityId = req.body.sportsActivityId;

    //Get the sports activity name
    const sport = req.body.sport_dropdown;

    //Update the database entry
    const edittedSportsActivityId = await
req.db.updateSportsActivity(activityId, sport, activityDuration,
startTimeStr, endTimeStr);

    res.redirect('/dashboard');
})

module.exports = router;
```

CommonFunctions.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');

/*

    NAME:
        formatDate() - Formats a date entry to "YYYY:MM:DD" format.

    SYNOPSIS:
        const formatDate(date);

        date --> The date to be formatted (Ex: "Mon July 4 2024") to
"YYYY:MM:DD" (object)

    DESCRIPTION:
        Takes the date object entered, and gets the year, month, and day,
formatting it into
        YYYY:MM:DD

    RETURNS:
        Returns a string of the formatted date.

*/
const formatDate = (date) => {
    const currentDate = date
    const year = currentDate.getFullYear();
    const month = String(currentDate.getMonth() + 1).padStart(2, '0');
    const day = String(currentDate.getDate()).padStart(2, '0');

    const formattedDate = `${year}-${month}-${day}`;

    return formattedDate;
}

/*
    Function: calcDuration
    Parameters: start_time (string), end_time (string)
    Returns: The difference in minutes between two start times.
```

```
    NAME:
        calcDuration() - Calculates the difference in minutes between a
start and end time.

    SYNOPSIS:
        const calcDuration(start_time, end_time);

        start_time --> The start time (string)
        end_time -->  The end time (string)

    DESCRIPTION:
        Takes the start time and end time of a workout and calculates the
difference between the two
        in order for the duration of the workout to be saved in the
workout entry

    RETURNS:
        Returns an integer of the difference between the times in minutes.

*/
const calcDuration = (start_time, end_time) => {

    function parseTime(timeString) {
        const [hours, minutes] = timeString.split(':').map(Number);
        const date = new Date();
        date.setHours(hours, minutes, 0, 0);
        return date;
    }

    // Parse the start and end times
    const start = parseTime(start_time);
    const end = parseTime(end_time);

    //If the end time is before the start time, that means it ends on the
next day (for those late night lifters)
    // so, we need to account for that
    if(end < start){
        end.setDate(end.getDate() + 1);
    }
```

```
    //Calculates the workout time in milliseconds
    const diffMilliseconds = end - start;

    //Convert the milliseconds to minutes
    const diffMinutes = diffMilliseconds / (1000 * 60);

    return diffMinutes;
}


//This exports the two functions so they can be used in other files.
module.exports = {
    formatDate,
    calcDuration,
};
```

WeeklyBreakdown.js

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');

//This function will calculate the weekly workout breakdown
/*

    NAME:
        calculateWeeklyBreakdown() - Calculates the breakdown of all
workouts/exercises/activities from a certain week

    SYNOPSIS:
        const calculateWeeklyBreakdown(userId, week_workouts,
week_exercises, week_sportActivities, exerciseTable)

        user_id --> The ID of the user the breakdown is for (integer)
        week_workouts --> All of the workout entries for the week (object
array)
        week_exercises --> All of the logged exercises for the week
(object array)
        week_sportActivities --> All of the logged sports activity for the
week (object array)
        exerciseTable --> The table of exercise information from the
database (object array)

    DESCRIPTION:
        This function takes all of the workouts/exercises/sports
activities for the week and calculates a breakdown/recap.
        This includes the amount of time spent in the gym during that
week/the amount of time active outside of the gym,
        as well as the percentages each of the major muscle groups and
workout categories take up of their totals,
        and the longest amount of time spent at once for that certain
week.

    RETURNS:
        An object array with all of the information generated in the
breakdown.
```

```
*/
const calculateWeeklyBreakdown = (userId, week_workouts, week_exercises,
week_sportActivities, exerciseTable) => {
    //The object array to store all of the breakdown information as it is
calculated
    const fullBreakdown = {};

    //STEP 1: AMOUNT OF TIME SPENT IN THE GYM / AMOUNT OF TIME ACTIVE
(SPORTS/ETC)
    //Loop through workouts and add together time
    let totalTimeGym = 0;
    for(const workout of week_workouts){
        totalTimeGym = totalTimeGym + workout.duration_minutes;
    }
    fullBreakdown['totalGymTime'] = totalTimeGym;

    let totalTimeSports = 0;
    for(const activity of week_sportActivities){
        totalTimeSports = totalTimeSports + activity.duration_minutes;
    }
    fullBreakdown['totalSportsTime'] = totalTimeSports;


    //STEP 2: PERCENTAGES OF EACH OF THE MAJOR MUSCLE GROUPS

    //Pass this week's exercises into a function to return the percentage
breakdown of muscle groups
    const breakdownMuscleGroup =
calcMuscleGroupPercentages(week_exercises, exerciseTable);
    fullBreakdown['MuscleGroupPercent'] = breakdownMuscleGroup;

    //Pass this week's exercises into a function to return the percentage
breakdown of workout categories
    const breakdownCategories = calcCategoryPercentages(week_exercises,
exerciseTable);
    fullBreakdown['CategoryPercent'] = breakdownCategories;

    //STEP 3: LONGEST AMOUNT OF TIME SPENT AT THE GYM
```

```
    //Pass this week's workouts into a function to return the workout with
the longest time.
    const longestWorkout = findLongestWorkout(week_workouts);
    fullBreakdown['longestWorkout'] = longestWorkout;

    return fullBreakdown;
}

//This function finds the workout in the list with the longest amount of
time spent doing said workout.
/*

    NAME:
        findLongestWorkout() - Finds the workout with the longest time

    SYNOPSIS:
        const findLongestWorkout(workouts);

        workouts --> An object array of all the workouts to compare with
each other (object array)

    DESCRIPTION:
        This function takes all of the workouts given, and loops through
them to find the workout that
        has the longest duration.

    RETURNS:
        Returns a workout object.

*/
const findLongestWorkout = (workouts) => {
    let longestTimeGym = 0;
    let longestWorkout = null;
    for(const workout of workouts){
        if(workout.duration_minutes > longestTimeGym){
            //This means it is the longest time found
            longestTimeGym = workout.duration_minutes;
            longestWorkout = workout;
        }
    }
```

```
    return longestWorkout;
}

/*

    NAME:
        calcMuscleGroupPercentages() - Calculates the percentages of the
muscle groups

    SYNOPSIS:
        const calcMuscleGroupPercentages(exercises, exerciseTable);

        exercises --> All of the exercises that are going to be tallied
(object array)
        exerciseTable --> All of the Exercises table information (object
array) (To check the muscle groups of each exercise)

    DESCRIPTION:
        This function takes the exercises the user logged that week and
looks at every single one of them, and uses the
        ExercisesTable to check all of the muscle groups that exercise
targets (quads, biceps, etc).

    RETURNS:
        An array of objects containing the percentages of muscle groups
hit for the week (sorted in descending order).

*/
const calcMuscleGroupPercentages = (exercises, exerciseTable) => {
    //Tally for each major muscle group
    const tally = {};
    //Total amount of muscle groups trained
    let totalCount = 0;

    var exerciseDetails;
    var muscleGroupsString;

    exercises.forEach(exercise => {
```

```
        exerciseDetails = exerciseTable.find(item => item.name ===
exercise.exercise_name);
        muscleGroupsString = exerciseDetails.muscleGroups;

        //Since the musclegroups are stored as a string, we need to strip
that string
        let muscleGroupsArray = muscleGroupsString.split(', ').map(word =>
word.trim());

        //Loop through the muscle groups
        for(var i = 0; i < muscleGroupsArray.length; i++){
            if(tally[muscleGroupsArray[i]]){
                tally[muscleGroupsArray[i]]++;
            } else {
                tally[muscleGroupsArray[i]] = 1;
            }

            totalCount++;
        }
    })

    //Now that we have the counts of each muscle group, we calculate what
percentage of the total they each are
    for(const key in tally){
        const count = tally[key];

        tally[key] = {
            count: count,
            percentage: ((count / totalCount) * 100).toFixed(2)
        };
    }

    //Sort by highest-lowest percentage
    const sorted = percentageSortDesc(tally);

    return sorted;
}

/*
```

```
    NAME:
        calcCategoryPercentages() - Calculates the percentages of the
workout categories (upper body, lower body, etc.)

    SYNOPSIS:
        const calcCategoryPercentages(exercises, exerciseTable);

        exercises --> All of the exercises that are going to be tallied
(object array)
        exerciseTable --> All of the Exercises table information (object
array) (To check the category of each exercise)

    DESCRIPTION:
        This function takes the exercises the user logged that week and
looks at every single one of them, and uses the
        ExercisesTable to check if it is upper body, lower body, or total
body, and tallies up each entry.

    RETURNS:
        An array of objects containing the percentages of
upper/lower/total body exercises for the week (sorted in descending
order).

*/
const calcCategoryPercentages = (exercises, exerciseTable) => {
    //Tally for each workout category
    const tally = {};
    //Total amount of workout exercises
    let totalCount = 0;

    var exerciseDetails;
    var category;

    exercises.forEach(exercise => {
        exerciseDetails = exerciseTable.find(item => item.name ===
exercise.exercise_name);
        category = exerciseDetails.bodyPart;
        if(tally[category]) {
            tally[category]++;
        } else {
```

```
            tally[category] = 1;
        }

        //Increment the total count
        totalCount++;
    })

    //Now that we have the counts of each category calculated, we
calculate what percentage of the total they each are.
    for(const key in tally){
        const count = tally[key];

        tally[key] = {
            count: count,
            percentage: ((count / totalCount) * 100).toFixed(2)
        };
    }

    //Sort by highest-lowest percentage
    const sorted = percentageSortDesc(tally);

    return sorted;
}

//This function takes an array of objects that have percentages and sorts
it in descending order
/*

    NAME:
        percentageSortDesc() - Sorts a dictionary based on entries

    SYNOPSIS:
        const percentageSortDesc(tally);

        tally - A dictionary of entries and their counts

    DESCRIPTION:
        This function takes a dictionary of entries and their total
counts/percentages and sorts them
        in descending order.
```

```
    RETURNS:
        The passed in object, but sorted in descending order.


*/
const percentageSortDesc = (tally) => {
    //Converts the object to an array of entries
    let sortedPercent = Object.entries(tally).sort((a, b) => {
        //Sort by percentage descending
        return parseFloat(b[1].percentage) - parseFloat(a[1].percentage);
    });

    //Converts the sorted array back to an object
    let sortedPercentObj = Object.fromEntries(sortedPercent);

    return sortedPercentObj;
}


module.exports = {
    calculateWeeklyBreakdown,
};
```

404.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="/logo.png", alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    body
        .container.mt-5
            .row.justify-content-center
                .col-md-6
                    .login-card.text-center.shadow-lg.border-0.rounded
                        .card-body
                            h4.card-title 404
                            p.card-text Page Not Found!
```

Dashboard.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav(style='display: flex; align-items: center;')
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
                                li
                                    a.dropdown-item(href="/logout")
```

```
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout



    body

        //- This is the navigation sidebar for the dashboard
        div(class="container-fluid")
            .row
                //- Sidebar
                div(id="sidebar" class="d-flex flex-column flex-shrink-0
p-3 text-bg-dark bg-dark" style="width: 280px; height: 100vh")
                    a(class="d-flex align-items-center mb-3 mb-md-0
me-md-auto text-white text-decoration-none")
                        svg(class="bi me-2", width="40", height="32")
                            span(class="fs-4")
                            |    #{user.first_name}'s Dashboard
                    hr(class="my-3", style="color: gray")
                    ul(class="nav nav-pills flex-column mb-auto")
                        li(class="nav-item")
                            a(class="nav-link active",
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-heart-pulse")
                                    |   Workouts
                        li(class="nav-item")
                            a(href="/recap" class="nav-link"
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi
bi-arrow-counterclockwise")
                                    |   Latest Recap
                        li(class="nav-item")
                            a(href='/friends' class="nav-link"
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
```

```
                                    i(class="bi bi-person-lines-fill")
                                    |   Friends

                div(class="col-md-9" style="color: white")
                    .row
                        h1(style="margin-left: 10px") #{user.first_name}'s
Workouts
                        a(href=user.id + '/newWorkout', class="btn
btn-primary rounded-pill col-md-3" style="margin-left: 10px") New Workout
                        a(href=user.id + '/newSportsActivity', class="btn
btn-primary rounded-pill col-md-3" style="margin-left: 10px") New Sports
Activity
                    p
                    .row
                        each workout in workouts
                            .col-md-2
                                a(href="/workouts/"+workout.id class="card
workout-card" style="margin: 5px 0; text-decoration: none")
                                    .card-body
                                        h5.card-title(style="color:
black") #{workout.date}
                                        p.card-text(style="color: black")
#{workout.duration_minutes} minute workout
                    p
                    .row
                        h1(style="margin-left: 10px") #{user.first_name}'s
Sports Activities
                    .row
                        each sport in sports
                            .col-md-2
                                a(href="/sportsActivities/"+sport.id
class="card workout-card" style="margin: 5px 0; text-decoration: none")
                                    .card-body
                                        h5.card-title(style="color:
black") #{sport.date}
                                        p.card-text(style="color: black")
#{sport.duration_minutes} minutes of #{sport.sport}
```

EditSportsActivity.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="/logo.png", alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    body
        .container.mt-5
            form(id='postForm', action=`/sportsActivities/:id`,
method='post').form
                .row.justify-content-center
                    .col-md-6
                        .login-card.text-center.shadow-lg.border-0.rounded
                            .card-header
                                h4.card-title Edit Sports Activity
                            .card-body
                                .container(id='sportActivityContainer')
                                    .row
                                        label(for='sport') Select Sport
                                    .row.justify-content-center
                                        select(name='sport_dropdown',
id='sport_dropdown', class='scrollable-dropdown-sports')
                                            each sport in sports
                                                if sport.sport ==
sportsActivity.sport

option(value=sports.sport, selected)= sport.sport
                                                else

option(value=sport.sport)= sport.sport
                                    p
                                    .row.justify-content-center
                                        .col-md-4.text-center
                                            label(for='startTime') Time
Started

input.form-control(type='time', id='startTime', name='startTime',
value=startTime, required)
                                        .col-md-4.text-center
                                            label(for='endTime') Time
Ended
```

```
input.form-control(type='time', id='endTime', name='endTime',
value=endTime, required)
                                    p
                                    .row
                                        .col
                                            a(type="button",
href='/dashboard' id="cancelButton", class="btn btn-danger btn-block
rounded-pill") CANCEL
                                        .col
                                            input(type="hidden",
name="sportsActivityId", value=sportsActivity.id)
                                            button(type="submit",
class="btn btn-success btn-block rounded-pill") Save Changes
```

EditWorkout.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="/logo.png", alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout
                    else
                        li.nav-item
                            a.nav-link(href='/login') Login

    //These are some hidden inputs in order to pass the main and accessory
counts into the script.
    input(type="hidden", name="m_count", id="m_count", value=m_count)
    input(type="hidden", name="a_count", id="a_count", value=a_count)


script(src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.
js")
    script.
        var mainRowCount = $('#m_count').val();
        var accessoryRowCount = $("#a_count").val();
        //- Main lift rows addition
        $(document).ready(function() {
            $("#addMainRowButton").click(function() {
                //Clones the target row
                var $row = $('#mainRow').clone();
                mainRowCount++;
                console.log(mainRowCount);

                var $rowCount = $('#mainRowContainer .row').length;

                //This function makes sure the exercise selection ids
maintain uniqueness

$row.find('select[name="m_exercise_dropdown"]').each(function(){
                    var currentName = $(this).attr('name')
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //This function makes sure the form input ids maintain
uniqueness (reps, sets, weight)
```

```
                $row.find('input').each(function() {
                    var currentName = $(this).attr('name');
                    $(this).attr('name', (currentName + $rowCount))

                    if($(this).attr('id') === `m_exercise_id`){
                        $(this).attr('value', 'null');
                    } else {
                        $(this).attr('id', (currentName + $rowCount))
                    }
                })

                //Clears the text boxes on the cloned row
                $row.find(":text").val("");

                //Appends the targeted row to the overbearing container
                $("#mainRowContainer").append($row);
            });
        });

        //- Accesory lift rows addition
        $(document).ready(function() {
            $("#addAccessoryRowButton").click(function() {
                //Clones the target row
                var $row = $('#accessoryRow').clone();
                accessoryRowCount++;
                console.log(accessoryRowCount);

                var $rowCount = $('#accessoryRowContainer .row').length;

                //This function makes sure the exercise selection ids
maintain uniqueness

$row.find('select[name="a_exercise_dropdown"]').each(function(){
                    var currentName = $(this).attr('name')
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //This function makes sure the form input ids maintain
uniqueness (reps, sets, weight)
```

```
                $row.find('input').each(function() {
                    var currentName = $(this).attr('name');
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //Clears the text boxes on the cloned row
                $row.find(":text").val("");

                //Appends the targeted row to the overbearing container
                $("#accessoryRowContainer").append($row);
            });
        });

        //- Main lift rows deletion
        $(document).ready(function() {
            $('#removeMainRowButton').click(function(){
                if(mainRowCount > 1){
                    mainRowCount--;
                    console.log(mainRowCount);
                }

                //Check the count of rows
                var rowCount = $('#mainRowContainer .row').length;

                //This ensures that there is always at least one row to
copy for the addition.+
                if(rowCount >= 2){
                    //Removes the latest row
                    $('#mainRowContainer .row:last').remove();
                }
            });
        });

        //- Accesory lift rows deletion
        $(document).ready(function() {
            $('#removeAccessoryRowButton').click(function(){
                if(accessoryRowCount > 1){
                    accessoryRowCount--;
                    console.log(accessoryRowCount);
```

```
                }

                //Check the count of rows
                var rowCount = $('#accessoryRowContainer .row').length;

                //This ensures that there is always at least one row to
copy for the addition
                if(rowCount >= 2){
                    //Removes the latest row
                    $('#accessoryRowContainer .row:last').remove();
                }
            });
        });

        //- This function passes the row counts when the page is submitted
        $(document).ready(function() {
            $('#postForm').submit(function(){
                $('#mainRowCount').val(mainRowCount);
                $('#accessoryRowCount').val(accessoryRowCount);
            })
        })

    body
        .container
            form(id='postForm', action=`/workouts/:id`,
method='post').form
                .row-justify-content-center.mt-4
                    .col-justfiy-content-center.md-6
                        .card.mains-card
                            .card-header
                                h4.card-title Main Lifts
                            .card-body
                                .container(id='mainRowContainer')
                                    - var rowCount = 0
                                    - var first_row = ''
                                    - var temp
                                    each userExercise in userExercises
                                        if userExercise.classification ==
'Main'
                                            if rowCount === 0
```

```
                                                    - temp = rowCount
                                                    - rowCount = first_row
                                                .row(id='mainRow')
                                                    if userExercise.id != null
                                                        input(type="hidden",
name=`m_exercise_id${rowCount}`, id='m_exercise_id' value=userExercise.id)
                                                    .col
                                                        label(for='exercise')
Select Exercise:

select(name=`m_exercise_dropdown${rowCount}`,
id=`m_exercise_dropdown${rowCount}`, class='scrollable-dropdown')
                                                            each exercise in
exercises
                                                                if
exercise.classification == 'Main'
                                                                    if
exercise.name == userExercise.exercise_name

option(value=exercise.name, selected)= exercise.name
                                                                    else

option(value=exercise.name)= exercise.name
                                                    .col
                                                        label(for='sets') Set
Count:

input.form-control(type='text', value=userExercise.sets, placeholder="Ex.
2", name=`m_sets${rowCount}`, id=`m_sets${rowCount}`)
                                                    .col
                                                        label(for='reps') Rep
Count:

input.form-control(type='text', value=userExercise.reps, placeholder="Ex.
5", name=`m_reps${rowCount}`, id=`m_reps${rowCount}`)
                                                    .col
                                                        label(for='weight')
Weight (lbs):

input.form-control(type='text', value=userExercise.weight,
```

```
placeholder="Ex. 225", name=`m_weight${rowCount}`,
id=`m_weight${rowCount}`)
                                                p
                                                hr
                                            if temp === 0
                                                - rowCount = 1
                                                - temp = 'null'
                                            else
                                                - rowCount++

                                .row
                                    .col
                                        button(type="button",
id='addMainRowButton' class="btn btn-success btn-block") Add Another Main
                                    .col-auto
                                        button(type="button",
id='removeMainRowButton', class="btn btn-danger btn-block") Delete Last
Row
                .row-justify-content-center.mt-4
                    .col-justfiy-content-center.md-6
                        .card.accesory-card
                            .card-header
                                h4.card-title Accessories
                            .card-body
                                .container(id='accessoryRowContainer')
                                    - var a_rowCount = 0
                                    - var a_first_row = ''
                                    - var temp1
                                    each userExercise in userExercises
                                        if a_rowCount === 0
                                            - temp1 = a_rowCount
                                            - a_rowCount = a_first_row
                                        if userExercise.classification ==
'Accessory'
                                            .row(id='accessoryRow')
                                                if userExercise.id != null
                                                    input(type="hidden",
name=`a_exercise_id${a_rowCount}`, id='a_exercise_id'
value=userExercise.id)
                                                .col
```

```
                                                    label(for='exercise')
Select Exercise:

select(name=`a_exercise_dropdown${a_rowCount}`,
id=`a_exercise_dropdown${a_rowCount}`, class='scrollable-dropdown')
                                                        each exercise in
exercises
                                                            if
exercise.classification == 'Accessory'
                                                                if
exercise.name == userExercise.exercise_name

option(value=exercise.name, selected)= exercise.name
                                                                else

option(value=exercise.name)= exercise.name
                                                .col
                                                    label(for='sets') Set
Count:

input.form-control(type='text', value=userExercise.sets, placeholder="Ex.
2", name=`a_sets${a_rowCount}`, id=`a_sets${a_rowCount}`)
                                                .col
                                                    label(for='reps') Rep
Count:

input.form-control(type='text', value=userExercise.reps, placeholder="Ex.
5", name=`a_reps${a_rowCount}`, id=`a_reps${a_rowCount}`)
                                                .col
                                                    label(for='weight')
Weight (lbs):

input.form-control(type='text', value=userExercise.weight,
placeholder="Ex. 225", name=`a_weight${a_rowCount}`,
id=`a_weight${a_rowCount}`)
                                                p
                                                hr
                                            if temp1 === 0
                                                - a_rowCount = 1
                                                - temp1 = 'null'
```

```
                                                else
                                                    - a_rowCount++

                                .row
                                    .col
                                        button(type="button",
id='addAccessoryRowButton' class="btn btn-success btn-block") Add Another
Accessory
                                    .col-auto
                                        button(type="button",
id='removeAccessoryRowButton', class="btn btn-danger btn-block") Delete
Last Row
                .row
                    .col-md-6
                        .card.my-4
                            .card-header
                                h4.card-title Time Spent
                            .card-body
                                .container(id='timeLoggingContainer')
                                    .col
                                        .row
                                            .col-md-4.text-center
                                                label(for='startTime')
Time Started

input.form-control(type='time', id='startTime', name='startTime',
value=startTime, required)
                                            .col-md-4.text-center
                                                label(for='endTime') Time
Ended

input.form-control(type='time', id='endTime', name='endTime',
value=endTime, required)
                    .col-md-6
                        .card.my-4
                            .card-header
                                h4.card-title Alter Workout
                            .card-body
                                .container-submissionContainer
                                    .row
```

```
                                        .col
                                            a(type="button",
href='/dashboard' id="cancelButton", class="btn btn-danger btn-block")
CANCEL
                                        .col-auto
                                            input(type="hidden",
name="mainRowCount", id="mainRowCount")
                                            input(type="hidden",
name="accessoryRowCount", id="accessoryRowCount")
                                            input(type="hidden",
name="workoutId", value=workout.id)
                                            button(type="submit",
class="btn btn-info btn-block") Save Workout
```

FindFriends.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                ul.navbar-nav(style='display: flex; align-items: center;')
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
                                li
                                    a.dropdown-item(href="/logout")
```

```
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    body
        .container.mt-4
            h2(style="color: white") Search for a User
            input#searchInput.form-control(type="text" placeholder="Type a
username...")
            #userList.user-list.mt-3

        script.
            const users = !{JSON.stringify(allUsers)};
            const searchInput = document.getElementById('searchInput');
            const userList = document.getElementById('userList');

            searchInput.addEventListener('input', function() {
                const query = this.value.toLowerCase();
                userList.innerHTML = ''; //This clears the previous
results

                if(query){
                    const filteredUsers = users.filter(user => {
                        return (
                            user.username.toLowerCase().includes(query) ||
                            user.first_name.toLowerCase().includes(query)
||
                            user.last_name.toLowerCase().includes(query)
||
                            `${user.first_name}
${user.last_name}`.toLowerCase().includes(query)
                        );
                    });
                    filteredUsers.forEach(user => {
                        const userItem = document.createElement('div');
                        userItem.className = 'user-item';
                        userItem.textContent = `${user.first_name}
${user.last_name} (${user.username})`;
                        userItem.addEventListener('click', () => {
                            window.location.href = `/u/${user.username}`;
//Redirects to the user's profile when clicked
```

```
                        })
                        userList.appendChild(userItem);
                    });
                }
            });
```

Friends.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav(style='display: flex; align-items: center;')
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
                                li
                                    a.dropdown-item(href="/logout")
```

```
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    body

        //- This is the navigation sidebar for the dashboard
        div(class="container-fluid")
            .row
                //- Sidebar
                div(id="sidebar" class="d-flex flex-column flex-shrink-0
p-3 text-bg-dark bg-dark" style="width: 280px; height: 100vh")
                    a(class="d-flex align-items-center mb-3 mb-md-0
me-md-auto text-white text-decoration-none")
                        svg(class="bi me-2", width="40", height="32")
                            span(class="fs-4")
                            |    #{user.first_name}'s Dashboard
                    hr(class="my-3", style="color: gray")
                    ul(class="nav nav-pills flex-column mb-auto")
                        li(class="nav-item")
                            a(href="/dashboard" class="nav-link",
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-heart-pulse")
                                    |   Workouts
                        li(class="nav-item")
                            a(href="/recap" class="nav-link"
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi
bi-arrow-counterclockwise")
                                    |   Latest Recap
                        li(class="nav-item")
                            a(class="nav-link active" aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-person-lines-fill")
                                    |   Friends
                div(class="col-md-9" style="color: white")
```

```
                    .container.mt-5
                        each friend in friends
                            .row.justify-content-center
                                .col-md-6
                                    a(href="/u/"+friend.username
class="card workout-card" style="margin: 5px 0; text-decoration: none")
                                        .card-body
                                            h4.card-title
#{friend.first_name} #{friend.last_name}
                                            p.card-text
                                                | Username:
#{friend.username}
```

Home.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="logo.png", width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout
                    else
                        li.nav-item
                            a.nav-link(href='/login') Login

    body
    .container
        .row.justify-content-center.align-items-center.py-5
            .col-md-6.text-center
                h1.display-4.mb-4.text-white Welcome to Gymbuds!
                img.img-fluid.mb-4(src="welcome-image.jpg", alt="Welcome
Image")
                p.lead.mb-4.text-white
                    | Gymbuds is a web application used to log your gym
workouts! Our main feature is logging what workouts you do in each
session, along with the rep schemes and the weights that you
                    | used.
                    |
                    | But that's not all! Once a week, the app will break
down what you did and supply a weekly recap consisting of the amount of
time you spent working out, and what muscle groups
                    | you hit!
                if !user
                    a(class="btn btn-outline-info btn-block btn-lg",
href="/login") Get Started!
                else
                    a(class="btn btn-outline-info btn-block btn-lg",
href="/dashboard") Go To Your Dashboard
```

Layout.pug

```
doctype html
html
    head
        title Gymbuds
        link(rel="icon", type="image/x=icon", href="/logo.png")
        meta(name="viewport",
content="width=device=width,initial-scale=1")
        script(src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js")
        link(, rel="stylesheet",
href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css")
        link(rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-i
cons.css")

link(href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap
.min.css", rel="stylesheet",
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65Vohhpuu
COmLASjC", crossorigin="anonymous")
        link(rel="stylesheet", href="/style.css")
    body
        block content


script(src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap
.bundle.min.js",
integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/t
WtIaxVXM", crossorigin="anonymous")
```

NewSportActivity.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="/logo.png", alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    body
        .container.mt-5
            form(id='postForm', action=`/${user.id}/newSportsActivity`,
method='post').form
                .row.justify-content-center
                    .col-md-6
                        .login-card.text-center.shadow-lg.border-0.rounded
                            .card-header
                                h4.card-title Log a New Sports Activity
                            .card-body
                                .container(id='sportActivityContainer')
                                    .row
                                        label(for='sport') Select Sport
                                    .row.justify-content-center
                                        select(name='sport_dropdown',
id='sport_dropdown', class='scrollable-dropdown-sports')
                                            each sport in sports
                                                option(value=sport.sport)=
sport.sport
                                    p
                                    .row.justify-content-center
                                        .col-md-4.text-center
                                            label(for='startTime') Time
Started

input.form-control(type='time', id='startTime', name='startTime',
required)
                                        .col-md-4.text-center
                                            label(for='endTime') Time
Ended

input.form-control(type='time', id='endTime', name='endTime', required)
                                    p
                                    .row
```

```
                                        button(type="submit", class="btn
btn-success btn-block rounded-pill") Log Sports Activity
```

NewWorkout.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="/logo.png", alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout
                    else
                        li.nav-item
                            a.nav-link(href='/login') Login


script(src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.
js")

    //- Below is all of the scripts for dynamic exercise addition/deletion
    script.
        var mainRowCount = 1;
        var accessoryRowCount = 1;
        //- Main lift rows addition
        $(document).ready(function() {
            $("#addMainRowButton").click(function() {

                //Clones the target row
                var $row = $('#mainRow').clone();
                mainRowCount++;
                console.log(mainRowCount);

                var $rowCount = $('#mainRowContainer .row').length;

                //This function makes sure the exercise selection ids
maintain uniqueness

$row.find('select[name="m_exercise_dropdown"]').each(function(){
                    var currentName = $(this).attr('name')
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //This function makes sure the form input ids maintain
uniqueness (reps, sets, weight)
                $row.find('input').each(function() {
                    var currentName = $(this).attr('name');
```

```
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //Clears the text boxes on the cloned row
                $row.find(":text").val("");

                //Appends the targeted row to the overbearing container
                $("#mainRowContainer").append($row);
            });
        });

        //- Accesory lift rows addition
        $(document).ready(function() {
            $("#addAccesoryRowButton").click(function() {
                //Clones the target row
                var $row = $('#accesoryRow').clone();
                accessoryRowCount++;
                console.log(accessoryRowCount);

                var $rowCount = $('#accesoryRowContainer .row').length;

                //This function makes sure the exercise selection ids
maintain uniqueness

$row.find('select[name="a_exercise_dropdown"]').each(function(){
                    var currentName = $(this).attr('name')
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //This function makes sure the form input ids maintain
uniqueness (reps, sets, weight)
                $row.find('input').each(function() {
                    var currentName = $(this).attr('name');
                    $(this).attr('name', (currentName + $rowCount))
                    $(this).attr('id', (currentName + $rowCount))
                })

                //Clears the text boxes on the cloned row
```

```
                $row.find(":text").val("");

                //Appends the targeted row to the overbearing container
                $("#accesoryRowContainer").append($row);
            });
        });

        //- Main lift rows deletion
        $(document).ready(function() {
            $('#removeMainRowButton').click(function(){
                if(mainRowCount > 1){
                    mainRowCount--;
                    console.log(mainRowCount);
                }

                //Check the count of rows
                var rowCount = $('#mainRowContainer .row').length;

                //This ensures that there is always at least one row to
copy for the addition.+
                if(rowCount >= 2){
                    //Removes the latest row
                    $('#mainRowContainer .row:last').remove();
                }
            })
        })

        //- Accesory lift rows deletion
        $(document).ready(function() {
            $('#removeAccesoryRowButton').click(function(){
                if(accessoryRowCount > 1){
                    accessoryRowCount--;
                    console.log(accessoryRowCount);
                }

                //Check the count of rows
                var rowCount = $('#accesoryRowContainer .row').length;

                //This ensures that there is always at least one row to
copy for the addition.+
```

```
                if(rowCount >= 2){
                    //Removes the latest row
                    $('#accesoryRowContainer .row:last').remove();
                }
            })
        })

        //- This function passes the row counts when the page is submitted
        $(document).ready(function() {
            $('#postForm').submit(function(){
                $('#mainRowCount').val(mainRowCount);
                $('#accessoryRowCount').val(accessoryRowCount);
            })
        })




    body
        .container
            form(id='postForm', action=`/${user.id}/newWorkout`,
method='post').form
                .row-justify-content-center.mt-4
                    .col-justfiy-content-center.md-6
                        .card.mains-card
                            .card-header
                                h4.card-title Main Lifts
                            .card-body
                                .container(id='mainRowContainer')
                                    .row(id='mainRow')
                                        .col
                                            label(for='exercise') Select
Exercise:

select(name='m_exercise_dropdown', id='m_exercise_dropdown',
class='scrollable-dropdown')
                                                each exercise in exercises
                                                    if
exercise.classification == 'Main'
```

```
option(value=exercise.name)= exercise.name
                                        .col
                                            label(for='sets') Set Count:

input.form-control(type='text', placeholder="Ex. 2", name='m_sets',
id='m_sets')
                                        .col
                                            label(for='reps') Rep Count:

input.form-control(type='text', placeholder="Ex. 5", name='m_reps',
id='m_reps')
                                        .col
                                            label(for='weight') Weight
(lbs):

input.form-control(type='text', placeholder="Ex. 225", name='m_weight',
id='m_weight')
                                        p
                                        hr
                                .row
                                    .col
                                        button(type="button",
id='addMainRowButton' class="btn btn-success btn-block") Add Another Main
                                    .col-auto
                                        button(type="button",
id='removeMainRowButton', class="btn btn-danger btn-block") Delete Last
Row
                .row-justify-content-center.mt-4
                    .col-justfiy-content-center.md-6
                        .card.accesory-card
                            .card-header
                                h4.card-title Accessories
                            .card-body
                                .container(id='accesoryRowContainer')
                                    .row(id='accesoryRow')
                                        .col
                                            label(for='exercise') Select
Exercise:
```

```
select(name='a_exercise_dropdown', id='a_exercise_dropdown',
class='scrollable-dropdown')
                                                each exercise in exercises
                                                    if
exercise.classification == 'Accessory'

option(value=exercise.name)= exercise.name
                                        .col
                                            label(for='sets') Set Count:

input.form-control(type='text', placeholder="Ex. 3", name='a_sets',
id='a_sets')
                                        .col
                                            label(for='reps') Rep Count:

input.form-control(type='text', placeholder="Ex. 10", name='a_reps',
id='a_reps')
                                        .col
                                            label(for='weight') Weight
(lbs):

input.form-control(type='text', placeholder="Ex. 85", name='a_weight',
id='a_weight')
                                        p
                                        hr
                                .row
                                    .col
                                        button(type="button",
id="addAccesoryRowButton", class="btn btn-success btn-block") Add Another
Accessory
                                    .col-auto
                                        button(type="button",
id='removeAccesoryRowButton', class="btn btn-danger btn-block") Delete
Last Row
                .row
                    .col-md-6
                        .card.my-4
                            .card-header
                                h4.card-title Time Spent
```

```
                            .card-body
                                .container(id='timeLoggingContainer')
                                    .col
                                        .row
                                            .col-md-4.text-center
                                                label(for='startTime')
Time Started

input.form-control(type='time', id='startTime', name='startTime',
required)
                                            .col-md-4.text-center
                                                label(for='endTime') Time
Ended

input.form-control(type='time', id='endTime', name='endTime', required)
                    .col-md-6
                        .card.my-4
                            .card-header
                                h4.card-title Finish Workout
                            .card-body
                                .container-submissionContainer
                                    .row-justify-content-center.mt-4.md-6
                                        .col-saveButton
                                            input(type="hidden",
name="mainRowCount", id="mainRowCount")
                                            input(type="hidden",
name="accessoryRowCount", id="accessoryRowCount")
                                            button(type="submit",
class="btn btn-info btn-block") Save Workout
```

Profile.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout
```

```
                    else
                        li.nav-item
                            a.nav-link(href='/login') Login

    body
        .container
            .row-justify-content-center.mt-5
                .col-justify-content-center.md-6
                    .card.profile-card
                        .card-header
                            h2.card-title.text-center Account Settings
                        .card-body
                            .row
                                h5.card-title User Information
                            .row
                                .col
                                    section Name: #{user.first_name}
#{user.last_name}
                                .col
                                    section Username: #{user.username}
                            hr
                            .row
                                h5.card-title Edit User Information
                            .row
                                form(action='/profile',
method='post').form
                                    .row
                                        .col
                                            label.form-label(for='first')
New First Name

input.form-control(type='text', placeholder='Enter new first name',
name='first')
                                        .col
                                            label.form-label(for='last')
New Last Name

input.form-control(type='text', placeholder='Enter new last name',
name='last')
                                    p
```

```
                                    .row
                                        .col

label.form-label(for='username') New Username

input.form-control(type='text', placeholder='Enter new username',
name='username')
                                    p
                                    .row
                                        .col
                                            button(type="submit",
class="btn btn-success btn-block", href="/profile") Update
                                        .col-auto
                                            a(href='/pswdchange',
class="btn btn-warning btn-block") Change Password
                            p
                            if message
                                .alert.alert-danger(class="text-center")
                                    i(class="bi
bi-excalamation-triangle-fill me-3 justify-content-center") #{message}
```

PswdChange.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon

    body
        .container
            .row-justify-content-center.mt-5
                .col-justify-content-center.md-6
                    .card.profile-card
                        .card-header
                            h2.card-title.text-center Change Password
                        .card-body
                            .row
                                form(action='/pswdchange',
method='post').form
                                    .row
                                        .col

label.form-label(for='password') New Password

input.form-control(type='password', placeholder='Enter new password',
name='password')
                                        .col

label.form-label(for='password2') Re-enter New Password

input.form-control(type='password', placeholder='Re-enter new password',
name='password2')
                                    p
```

```
                                    .row
                                        .col
                                            button(type="submit",
class="btn btn-success btn-block", href="/pswdchange") Update
                                        .col-auto
                                            a(href='/profile', class="btn
btn-danger btn-block") Cancel
                            p
                            if success == false
                                .alert.alert-danger(class="text-center")
                                    i(class="bi
bi-exclamation-triangle-fill me-3 justify-content-center",
style="font-style: normal") #{message}
                            if success == true
                                .alert.alert-success(class="text-center")
                                    i(class="bi bi-check-circle-fill me-3
justify-content-center", style="font-style: normal")  #{message}
```

PublicProfile.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src="/logo.png", alt='logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/dashboard') Dashboard
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
```

```
                                li
                                    a.dropdown-item(href="/logout")
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout
                    else
                        li.nav-item
                            a.nav-link(href='/login') Login

    body
        if userPage === 'undefined'

.container.d-flex.justify-content-center.align-items-center(style="height:
80vh;")
                    .card.text-center(style="width: 24rem;")
                        .card-body
                            h1.card-title User Not Found!
                            p.card-text Sorry, the user you are looking
for does not exist.
                            if user
                                a.btn.btn-primary(href='/dashboard')
Return to Dashboard
                            else
                                a.btn.btn-primary(href='/') Return Home
        else if !user
            .container.mt-5
                .row.justify-content-center
                    .col-md-6

.card-profile.text-center.shadow-lg.border-0.rounded
                            .card-body
                                h4.card-title #{userPage.first_name}
#{userPage.last_name}
                                p.card-text
                                    | Username: #{userPage.username}
                                a.btn.btn-primary(href='/login') Login To
Add as Friend
        else if user
            if user.id === userPage.id
                .container.mt-5
                    .row.justify-content-center
```

```
                        .col-md-6

.card-profile.text-center.shadow-lg.border-0.rounded
                                .card-body
                                    h4.card-title #{userPage.first_name}
#{userPage.last_name}
                                    p.card-text
                                        | Username: #{userPage.username}
                                    a.btn.btn-primary(href='/profile')
View Profile Settings
            else
                .container.mt-5
                    .row.justify-content-center
                        .col-md-6

.card-profile.text-center.shadow-lg.border-0.rounded
                                .card-body
                                    h4.card-title #{userPage.first_name}
#{userPage.last_name}
                                    p.card-text
                                        | Username: #{userPage.username}
                                    if !isFriend
                                        form(action='/add-friend',
method='POST')
                                            input(type='hidden',
name='friend_id', value=userPage.id)
                                            input(type='hidden',
name='user_id', value=user.id)

button.btn.btn-primary(type='submit') Add as Friend
                                    else
                                        p.text-success You are friends!
                                        form(action='/remove-friend',
method='POST')
                                            input(type='hidden',
name='friend_id', value=userPage.id)
                                            input(type='hidden',
name='user_id', value=user.id)

button.btn.btn-danger(type='submit') Remove Friend
```

Recap.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav(style='display: flex; align-items: center;')
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
                                li
                                    a.dropdown-item(href="/logout")
```

```
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    script.
        //This script function is to ensure the progress bar sections are
different colors
        document.addEventListener('DOMContentLoaded', function() {
            const progressBars =
document.querySelectorAll('.progress-bar');
            const colors = [
                '#C0392B', // Dark Red
                '#27AE60', // Dark Green
                '#2980B9', // Dark Blue
                '#8E44AD', // Dark Purple
                '#D35400', // Dark Orange
                '#16A085', // Dark Teal
                '#2C3E50', // Dark Gray-Blue
                '#7D3C98', // Dark Magenta
                '#6C3483', // Dark Indigo
                '#A04000'  // Dark Brown
            ];

            progressBars.forEach((bar, index) => {
                const color = colors[index % colors.length];
                bar.style.backgroundColor = color;
            });
        })

        //This script is to add tooltips to the progress bar (when you
hover over it tells you the percentage)
        document.addEventListener('DOMContentLoaded', function () {
            var tooltipTriggerList =
[].slice.call(document.querySelectorAll('[data-bs-toggle="tooltip"]'))
            var toolTipList = tooltipTriggerList.map(function
(tooltipTriggerEl) {
                return new bootstrap.Tooltip(tooltipTriggerEl)
            })
        });

    body
```

```
        //- This is the navigation sidebar for the dashboard
        div(class="container-fluid")
            .row
                //- Sidebar
                div(id="sidebar" class="d-flex flex-column flex-shrink-0
p-3 text-bg-dark bg-dark" style="width: 280px; height: 100vh")
                    a(class="d-flex align-items-center mb-3 mb-md-0
me-md-auto text-white text-decoration-none")
                        svg(class="bi me-2", width="40", height="32")
                            span(class="fs-4")
                            |    #{user.first_name}'s Dashboard
                    hr(class="my-3", style="color: gray")
                    ul(class="nav nav-pills flex-column mb-auto")
                        li(class="nav-item")
                            a(href="/dashboard" class="nav-link",
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-heart-pulse")
                                    |   Workouts
                        li(class="nav-item")
                            a(class="nav-link active" aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi
bi-arrow-counterclockwise")
                                    |   Latest Recap
                        li(class="nav-item")
                            a(href='/friends' class="nav-link"
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-person-lines-fill")
                                    |   Friends

                div(class="col-md-9" style="color: white")
                    h1 Weekly Workout Summary

                    if noWorkouts === true
```

```
.container.d-flex.justify-content-center.align-items-center(style="height:
80vh;")
                            .card.text-center(style="width: 24rem;")
                                .card-body(style="color: black")
                                    h1.card-title No Workouts to Recap!
                                    p.card-text There's no workouts to
recap yet this week!
                                    a.btn.btn-primary(href='/dashboard')
Return to Dashboard
                    if noWorkouts === false
                        //- Total Gym and Sports Time
                        .row(style="color: black")
                            .col-md-6
                                .workout-card.mb-4
                                    .card-body
                                        h5.card-title Total Time Working
out in the Gym
                                        if recap.totalGymTime == null
                                            p.card-text No Time Yet
                                        if recap.totalGymTime != null
                                            p.card-text
#{recap.totalGymTime} minutes
                            .col-md-6
                                .workout-card.mb-4
                                    .card-body
                                        h5.card-title Total Time Active
Outside of the Gym
                                        if recap.totalSportsTime == null
                                            p.card-text No Time Yet
                                        if recap.totalSportsTime != null
                                            p.card-text
#{recap.totalSportsTime} minutes

                        //- Percentage Breakdown of Muscle Group Usage
                        .card.mb-4(style="color: black")
                            .card-body
                                h5.card-title Muscle Group Usage
                                .progress(style="height: 25px;")
```

```
                                    each info, group in
recap.MuscleGroupPercent
                                        .progress-bar(
                                            role="progressbar",
                                            style=`width:
${info.percentage}%`,
                                            aria-valuenow=info.percentage,
                                            aria-valuemin="0",
                                            aria-valuemax="100",
                                            data-bs-toggle="tooltip",
                                            data-bs-placement="top",
                                            title=`${group}:
${info.percentage}%`
                                        )
                                            | #{group}

                        //- Percentage Breakdown of Workout Category Usage
                        .card.mb-4(style="color: black")
                            .card-body
                                h5.card-title Workout Category Usage
                                .progress(style="height: 25px;")
                                    each info, category in
recap.CategoryPercent
                                        .progress-bar(
                                            role="progressbar",
                                            style=`width:
${info.percentage}%`,
                                            aria-valuenow=info.percentage,
                                            aria-valuemin="0",
                                            aria-valuemax="100",
                                            data-bs-toggle="tooltip",
                                            data-bs-placement="top",
                                            title=`${category}:
${info.percentage}%`
                                        )
                                            | #{category}

                        //- Longest Workout of the Week
                        .workout-card.mb-4
                            .card-body
```

```
                                    h5.card-title Longest Workout of the Week
                                    if recap.longestWorkout == null
                                        p.card-text No Workouts Yet!
                                    if recap.longestWorkout != null
                                        p.card-text Duration:
#{recap.longestWorkout.duration_minutes} minutes
                                        p.card-text Date:
#{recap.longestWorkout.date}
```

Recaps-Weekly.pug

```
extends layout.pug

block content
    nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
        .container-fluid
            a.navbar-brand(href='/')
                img.img-fluid(src='logo.png', alt='Logo', width='50')
                |   Gymbuds
            button.navbar-toggler(type='button',
data-bs-toggle='collapse', data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
                span.navbar-toggler-icon
            #navbarSupportedContent.collapse.navbar-collapse
                ul.navbar-nav.me-auto.mb-2.mb-lg-0
                    if user
                        li.nav-item
                            a.nav-link(href='/findFriends') Find New
Friends
                ul.navbar-nav(style='display: flex; align-items: center;')
                    if user
                        li.nav-item.dropdown
                            a.nav-link(class="btn dropdown-toggle",
data-bs-toggle="dropdown", aria-haspopup="true", aria-expanded="true")
                                i(class="bi bi-person-circle
profile-icon")

ul.dropdown-menu.dropdown-menu-end(aria-labelledby="dropdownMenuButton")
                                li

.dropdown-item.dropdown-header(style="text-align: center; color: #000;")
#{user.first_name} #{user.last_name}
                                div.dropdown-divider
                                li
                                    a.dropdown-item(href="/profile")
                                        i(class="bi bi-gear-fill"
style="font-style: normal")  Profile Settings
                                li
                                    a.dropdown-item(href="/logout")
```

```
                                        i(class="bi bi-box-arrow-right"
style="font-style: normal")  Logout

    body

        //- This is the navigation sidebar for the dashboard
        div(class="container-fluid")
            .row
                //- Sidebar
                div(id="sidebar" class="d-flex flex-column flex-shrink-0
p-3 text-bg-dark bg-dark" style="width: 280px; height: 100vh")
                    a(class="d-flex align-items-center mb-3 mb-md-0
me-md-auto text-white text-decoration-none")
                        svg(class="bi me-2", width="40", height="32")
                            span(class="fs-4")
                            |    #{user.first_name}'s Dashboard
                    hr(class="my-3", style="color: gray")
                    ul(class="nav nav-pills flex-column mb-auto")
                        li(class="nav-item")
                            a(href="/dashboard" class="nav-link",
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-heart-pulse")
                                    |   Workouts
                        li(class="nav-item")
                            a(href="/recap" class="nav-link"
aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi
bi-arrow-counterclockwise")
                                    |   Latest Recap
                        li(class="nav-item")
                            a(class="nav-link active" aria-current="page")
                                svg(class="bi me-2", width="16",
height="16")
                                    i(class="bi bi-journals")
                                    |   Weekly Recaps
                        li(class="nav-item")
```

```
                                a(href='/friends' class="nav-link"
aria-current="page")
                                    svg(class="bi me-2", width="16",
height="16")
                                        i(class="bi bi-person-lines-fill")
                                        |   Friends
```

Signup.pug

```
extends layout.pug

block content
  nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
      .container-fluid
          a.navbar-brand(href='/')
              img.img-fluid(src='logo.png', alt='Logo', width='50')
              |   Gymbuds
          button.navbar-toggler(type='button', data-bs-toggle='collapse',
data-bs-target='#navbarSupportedContent',
aria-controls='navbarSupportedContent', aria-expanded='false',
aria-label='Toggle navigation')
              span.navbar-toggler-icon

  body
      .container
        .row.justify-content-center.mt-5
          .col-md-6
            .card.login-card
              .card-header
                h5.card-title.text-center Create an Account for Gymbuds
              .card-body
                form(action='/signup' method='post').form
                  .mb-3
                    label.form-label(for='first') First Name:
                    input.form-control(type='text', placeholder='Enter
first name', name='first', required)
                  .mb-3
                    label.form-label(for='last') Last Name:
                    input.form-control(type='text', placeholder='Enter
last name', name='last', required)
                  .mb-3
                    label.form-label(for='username') Username:
                    input.form-control(type='text', placeholder='Enter
username', name='username', required)
                  .mb-3
                    label.form-label(for='password') Password:
                    input.form-control(type='password', placeholder='Enter
password', name='password', required)
```

```
                    .mb-3
                      label.form-label(for='password2') Re-enter Password:
                      input.form-control(type='password',
placeholder='Re-enter password', name='password2', required)
                    .row
                      .col
                        button(type="submit", class="btn btn-success
btn-block") Sign Up
                      .col-auto
                        a(href='/login', class="btn btn-info btn-block") Go
to Login
                    p
                    if message
                      .alert.alert-danger(class="text-center")
                        i(class="bi bi-exclamation-triangle-fill me-3
justify-content-center")  #{message}
```

Unauthorized.pug

```
extends layout.pug

block content
    .container
        .header(style="color: white")
            h1 User Unauthorized!
        .row
            .col
                .card(class="text-white bg-dark mb-3")
                    .card-body
                        .row
                            .col
                                if (userUnauthorized)
                                    h2 Sorry, that workout isn't yours.
                                    p
                                        .form-group
                                            div(class="d-grid gap-2
d-md-flex justify-content-md-start")
                                                a(class="btn btn-info",
href='/dashboard', role="button") Return To Your Dashboard
                                else
                                    h2 Please log in to access this page.
                                    p
                                        .form-group
                                            div(class="d-grid gap-2
d-md-flex justify-content-md-start")
                                                a(class="btn btn-success",
href='/login', role='button') Log In
                                                a(class="btn btn-info",
href='/', role="button") Return Home
```

DBindex.js

```
const assert = require('assert');
const sqlite3 = require('sqlite3').verbose();

//DataStore Class
//All of the basic functions for interacting with the database, all
specific functions will call one of these in order to generate a sql
query.
class DataStore {
    constructor() {
        // Read Configuration
        this.path = process.env.DBPATH;
        this.db = new sqlite3.Database('./gymbuds.db');
    }

    connect() {
        return new Promise((resolve, reject) => {
            this.db.serialize(() => {
                console.log('Connected to database ');
                resolve;
            });

            this.db.on('error', (err) => {
                console.error('Database error:', err.message);
                reject(err);
            });
        });
    }


    /*

        NAME:
            schema() - Creates a table in the database

        SYNOPSIS:
            schema(table, schema, pkey, fkey);

            table --> The name of the table to be created (text)
            schema --> The names of the table data categories (text)
```

```
            pkey --> The primary key of the table (integer)
            fkey --> A foreign key if needed (integer)

        DESCRIPTION:
            This function takes information passed in and generates a SQL
query to
            create a table according to information passed in.

        RETURNS:
            Returns a blank promise if successful, and outputs if the
creation is successful or not into console.
    */
    schema(table, schema, pkey, fkey) {
        let sql = `CREATE TABLE IF NOT EXISTS "${table}"
            (${schema.map(c => `"${c.name}" ${c.type}`).join(", ")},
            PRIMARY KEY ("${pkey}")`;

        if(fkey){
            sql += fkey;
        } else{
            sql += ')';
        }

        return new Promise((resolve, reject) => {
          this.db.run(sql, (err) => {
            if (err) {
              console.error('Error creating table:', err.message);
              reject(err);
            } else {
              console.log('Table created or already exists');
              resolve();
            }
          });
        });
    }

    /*

        NAME:
```

```
            read() - Generates SQL to read from a certain table in
database

        SYNOPSIS:
            read(table, query);

            table --> The name of the table that will be read from (text)
            query --> The object containing the search parameters (object
array)

        DESCRIPTION:
            This functions takes a table name and values and generates an
SQL query
            to retrieve information from the database.

        RETURNS:
            Returns a promise containing all rows found from the table
given the query parameters.
            Reports error otherwise.

    */
    read(table, query){
        let sql = `SELECT * from ${table}`;
        if (query.length > 0) {
            sql += ` WHERE ${query.map(d => `${d.column} =
'${d.value}'`).join(' and ')}`
        }

        return new Promise((resolve, reject) => {
            this.db.get(sql, (err, rows) => {
                if (err) {
                    console.error('Error:', err.message);
                    reject(err);
                } else {
                    console.log('Successful lookup');
                    resolve(rows);
                }
            });
        });
    }
```

```
    //Update an entry in a table
    /*

        NAME:
            update() - Generates SQL to update an entry in the database

        SYNOPSIS:
            update(table, data, query);

            table --> The name of the table to be accessed (text)
            data --> The new information of columns of data that is being
changed (object array)
            query --> The ID of the entry that is being changed (integer)

        DESCRIPTION:
            This takes the array of data, and maps all of it into a string
of text, that is an
            SQL query, and runs the generated SQL.

        RETURNS:
            Returns a blank promise if successful, and logs to console
whether or not it is successful.

    */
    update(table, data, query){
        const params = Array(data.length).fill('?')
        let sql = `UPDATE ${table} set ${data.map(d => `${d.column}=?`)}
where ${query.map(d => `${d.column} = ?`).join(' and ')}`;
        const _data = data.map(d => d.value).concat(query.map(q =>
q.value));

        return new Promise((resolve, reject) => {
            this.db.run(sql, _data, (err) => {
              if (err) {
                console.error('Error updating table:', err.message);
                reject(err);
              } else {
                console.log('Table successfully updated');
                resolve();
```

```
                }
            });
        });
    }

    /*

        NAME:
            delete() - Generates SQL to delete an entry from a specified
table in the database

        SYNOPSIS:
            delete(table, query);

            table --> The name of the table to be deleted (text)
            query --> The ID of the row to be deleted (integer)

        DESCRIPTION:
            This function takes the table name and id (query) and maps
that information to a string of text,
            and passes that string of text to the databse as an SQL query,
running that query and deleting
            the row specified from the query.

        RETURNS:
            Returns a blank promise. Logs to console if successful or
unsuccessful.
    */
    delete(table, query){
        let sql = `DELETE from ${table}`;
        if (query.length > 0) {
            sql += ` WHERE ${query.map(d => `${d.column} =
'${d.value}'`).join(' and ')}`
        }

        return new Promise((resolve, reject) => {
            this.db.run(sql, (err) => {
              if (err) {
                console.error('Error deleting from table:', err.message);
                reject(err);
```

```
            } else {
              console.log('Table entry successfully deleted');
              resolve();
            }
          });
        });
    }


    /*

        NAME:
            create() - Generates an SQL query to create an entry in a
specified table

        SYNOPSIS:
            create(table, data);

            table --> The name of the table to be updated (text)
            data --> The data of the new table entry (object array)

        DESCRIPTION:
            This function takes a table name and column data and maps that
data to a string, and
            passes that string to the database as an SQL query, running
that query and creating an
            entry in the table.

        RETURNS:
            Returns a promise with the ID of the created row. Logs to
console if successful/unsuccessful.

    */
    create(table, data) {
        const params = Array(data.length).fill('?')
        const sql = `INSERT into ${table} (${data.map(d =>
d.column).join(',')}) values (${params.join(',')})`;
        console.log(sql, data.map(d => d.value));

        let lastID = null;
```

```
        //Insert the item into the specified table
        return new Promise((resolve, reject) => {
            this.db.run(sql, data.map(d => d.value), function(err) {
                if (err) {
                    reject(err);
                    return;
                } else {
                    lastID = this.lastID;
                    resolve(lastID);
                }
            });
        })
    }

    /*

        NAME:
            isTableEmpty() - Checks if a specified table is empty

        SYNOPSIS:
            isTableEmpty(table);

            table --> The name of the table to be accessed/checked (text)

        DESCRIPTION:
            Maps the table name to an SQL query that selects the amount of
            rows in the table.

        RETURNS:
            True if the amount of rows is 0, false otherwise.

    */
    isTableEmpty(table){
        return new Promise((resolve, reject) => {
            this.db.get(`SELECT COUNT(*) AS count FROM ${table}`, (err,
row) => {
                if (err) {
                    reject(err);
                } else {
```

```
                    const rowCount = row.count;
                    resolve(rowCount === 0);
                }
            });
        });
    }

    /*

        NAME:
            getAll() - Returns all rows from a specified table in the
database

        SYNOPSIS:
            getAll(table);

            table --> The name of the table to be queried from (text)

        DESCRIPTION:
            Maps the name of the table into a string of text, then passes
that
            string of text to the database as an SQL query.

        RETURNS:
            An object array of all of the rows retrieved from the
specified
            table in the database

    */
    getAll(table){
        return new Promise((resolve, reject) => {
            this.db.all(`SELECT * FROM ${table}`, (err, rows) => {
                if (err) {
                    reject(err);
                } else {
                    resolve(rows);
                }
            });
        });
    }
```

```
    /*

        NAME:
            getAllWhere() - Returns all rows from a specified table with
the specified data

        SYNOPSIS:
            getAllWhere(table, query);

            table --> The name of the table to be accessed (text)
            query --> The data and ID for the SQL query (object array)

        DESCRIPTION:
            This function takes the information from the query and maps
the table name and information
            from the query into a string of text, which is then passed
into the database as SQL query.

        RETURNS:
            An object array of all rows found from the table with the
specified information.

    */
    getAllWhere(table, query){
        let sql = `SELECT * from ${table}`;
        if (query.length > 0) {
            sql += ` WHERE ${query.map(d => `${d.column} =
'${d.value}'`).join(' and ')}`
        }

        return new Promise((resolve, reject) => {
            this.db.all(sql, (err, rows) => {
                if (err) {
                    console.error('Error:', err.message);
                    reject(err);
                } else {
                    console.log('Successful lookup');
                    resolve(rows);
                }
```

```
            });
        });
    }

    /*

        NAME:
            getAllInRange() - Retrieves all rows in specified table from
specified user in a specific date range

        SYNOPSIS:
            getAllInRange(table, user_id, query);

            table --> The name of the table to be accessed (text)
            user_id --> The ID of the user the rows in the table will
belong to (integer)
            query --> An object containing the two dates the data needs to
be in-between (object array)

        DESCRIPTION:
            This function takes the table name, user ID, and two dates,
and maps it all to a string of
            text, which is then passed to the database as an SQL query. In
this specific case, a query that
            returns all rows between two specified dates.

        RETURNS:
            Returns an object array of all rows between the two dates from
the specific user.


    */
    getAllInRange(table, user_id, query){
        let sql = `SELECT * from ${table} WHERE user_id = ${user_id}`;
        if (query.length > 0){
            sql += ` AND date BETWEEN ${query.map(d =>
`'${d.value}'`).join(' and ')}`
        }

        return new Promise((resolve, reject) => {
```

```
            this.db.all(sql, (err, rows) => {
                if (err) {
                    console.error('Error: ', err.message);
                    reject(err);
                } else {
                    console.log('Successful lookup');
                    resolve(rows);
                }
            });
        });
    }

}

module.exports = DataStore;
```

UserDB.js

```
require('dotenv').config();
const fs = require('fs');
const DataStore = require('./DBindex');
const Database = require('./DBindex');
const sqlite3 = require('sqlite3').verbose();

class UserDB {
    constructor(databasePath) {
      this.db = new Database(databasePath);
    }

    //Initializes the database connection
    async initialize() {
      try {
        await this.db.connect();
      } catch (error) {
        console.error('Error initializing database:', error.message);
      }
    }

    //Makes the user table (upon very first run)
    async makeUserTable(){
        try{
            await this.db.schema('Users', [
                { name: 'id', type: 'INTEGER' },
                { name: 'first_name', type: 'TEXT' },
                { name: 'last_name', type: 'TEXT' },
                { name: 'username', type: 'TEXT' },
                { name: 'password', type: 'TEXT' }
            ], 'id');
        } catch (error) {
            console.error('Error creating user table', error.message);
        }
    }

    //Makes the friends table (upon very first run)
    async makeFriendsTable(){
        try {
            await this.db.schema('Friends', [
```

```
                { name: 'id', type: 'INTEGER' },
                { name: 'user_id', type: 'INTEGER' },
                { name: 'friend_id', type: 'INTEGER' }
            ], 'id', ', FOREIGN KEY ("user_id") REFERENCES Users ("id"),
FOREIGN KEY ("friend_id") REFERENCES Users ("id"), UNIQUE ("user_id",
"friend_id") )')
        } catch (error) {
            console.error('Error creating friends table', error.message);
        }
    }

    //Makes the workout table (upon very first run)
    async makeWorkoutTable(){
        try{
            await this.db.schema('Workouts', [
                { name: 'id', type: 'INTEGER' },
                { name: 'user_id', type: 'INTEGER' },
                { name: 'date', type: 'DATE' },
                { name: 'duration_minutes', type: 'INTEGER' },
                { name: 'start_time', type: 'TEXT'},
                { name: 'end_time', type: 'TEXT'}
            ], 'id', ', FOREIGN KEY ("user_id") REFERENCES Users ("id")
)');
        } catch (error) {
            console.error('Error creating workout table', error.message);
        }
    }

    //Makes the sport activity table (upon very first run)
    async makeSportsActivityTable(){
        try {
            await this.db.schema('SportActivity', [
                { name: 'id', type: 'INTEGER'},
                { name: 'user_id', type: 'INTEGER'},
                { name: 'sport', type: 'TEXT' },
                { name: 'date', type: 'DATE' },
                { name: 'duration_minutes', type: 'INTEGER' },
                { name: 'start_time', type: 'TEXT'},
                { name: 'end_time', type: 'TEXT'}
```

```
            ], 'id', ', FOREIGN KEY ("user_id") REFERENCES Users ("id")
)');
        } catch (error) {
            console.error('Error creating sports activity table',
error.message);
        }
    }

    //Makes the table for exercises (upon very first run)
    async makeExercisesTable(){
        try{
            await this.db.schema('Exercises', [
                { name: 'id', type: 'INTEGER' },
                { name: 'name', type: 'TEXT'},
                { name: 'classification', type: 'TEXT'},
                { name: 'muscleGroups', type: 'TEXT'},
                { name: 'bodyPart', type: 'TEXT'}
            ], 'id');
        } catch (error) {
            console.error('Error creating workout table', error.message);
        }
    }

    //Makes the table to store all the sports for the users to choose from
(upon very first run)
    async makeSportsTable(){
        try{
            await this.db.schema('Sports', [
                { name: 'id', type: 'INTEGER' },
                { name: 'sport', type: 'TEXT'},
            ], 'id');
        } catch (error) {
            console.error('Error creating sports table: ', error.message);
        }
    }

    //Makes the table to store all the exercises in the user's workouts
(upon very first run)
    async makeUserExercisesTable(){
        try{
```

```
            await this.db.schema('UserExercises', [
                { name: 'id', type: 'INTEGER' },
                { name: 'workout_id', type: 'INTEGER' },
                { name: 'exercise_name', type: 'TEXT' },
                { name: 'classification', type: 'TEXT' },
                { name: 'sets', type: 'INTEGER' },
                { name: 'reps', type: 'INTEGER' },
                { name: 'weight', type: 'INTEGER' }
            ], 'id', ', FOREIGN KEY ("workout_id") REFERENCES Workouts
("id") )');
        } catch (error) {
            console.error('Error creating User Exercises Table',
error.message);
        }
    }

    //Makes the table to store all of the user's weekly recaps
    async makeRecapTable(){
        try{
            await this.db.schema('Recaps', [
                { name: 'id', type: 'INTEGER' },
                { name: 'user_id', type: 'INTEGER' },
                { name: 'total_time', type: 'INTEGER' },
                { name: 'longest_time', type: 'INTEGER' },
                { name: 'total_body', type: 'REAL' },
                { name: 'upper_body', type: 'REAL' },
                { name: 'lower_body', type: 'REAL' },
                { name: 'monday_date', type: 'TEXT' },
                { name: 'sunday_date', type: 'TEXT' }
            ], 'id', ', FOREIGN KEY ("user_id") REFERENCES Users ("id")
)');
        } catch (error) {
            console.error('Error creating recaps table', error.message);
        }
    }



    /*
```

```
    NAME:
        createWorkout() - Creates a workout in the database

    SYNOPSIS:
        async createWorkout(user_id, date, start_time, end_time,
duration_minutes);

        user_id --> The ID of the user creating a workout (integer)
        date --> The date that the workout is being created (YYYY:MM:DD
format)
        start_time - The time the workout began (24:00 Format)
        end_time --> The time the workout ended (24:00 Format)
        duration_minutes --> The duration of the workout in minutes
(integer)

    DESCRIPTION:
        This function takes all of the details of a workout entry and
calls a function to
        query the database with an "INSERT" function with the input
information to add a new
        entry to the 'Workouts' table

    RETURNS:
        Returns the ID of the workout entry created. Reports an error to
console if unsuccessful.

    */
    async createWorkout(user_id, date, start_time, end_time,
duration_minutes){
        try{
            const id = await this.db.create('Workouts', [
                { column: 'user_id', value: user_id },
                { column: 'date', value: date},
                { column: 'duration_minutes', value: duration_minutes },
                { column: 'start_time', value: start_time },
                { column: 'end_time', value: end_time }
            ])
            return id;
        } catch (error) {
            console.error('Error creating a new workout: ', error);
```

```
        }
    }

    /*

    NAME:
        updateWorkout() - Updates a workout in the database

    SYNOPSIS:
        async updateWorkout(workout_id, start_time, end_time,
duration_minutes)

        workout_id --> The ID of the workout that will be updated/changed.
(integer)
        start_time - The time the workout began (24:00 Format)
        end_time --> The time the workout ended (24:00 Format)
        duration_minutes --> The duration of the workout in minutes
(integer)

    DESCRIPTION:
        This function takes all of the details of a workout entry (minus
the date because that doesn't change)
        and calls a function to query the database with an "UPDATE"
function with the input information to change
        the details of the workout entry under the given workout ID.

    RETURNS:
        Reports an error to console if unsuccessful.

    */
    async updateWorkout(workout_id, start_time, end_time,
duration_minutes){
        try{
            const id = await this.db.update('Workouts', [
                { column: 'start_time', value: start_time },
                { column: 'end_time', value: end_time},
                { column: 'duration_minutes', value: duration_minutes }
            ], [{ column: 'id', value: workout_id }]);
        } catch (error) {
            console.error('Error updating workout: ', error);
```

```
        }
    }

    //Creates a sports activity
    /*

    NAME:
        createSportsActivity() - Creates a sports activity entry in the
database

    SYNOPSIS:
        async createSportsActivity(user_id, sport, date, duration_minutes,
start_time, end_time);

        user_id --> The ID of the user creating the sports activity entry
(integer)
        sport --> The name of the sport the user is logging (text)
        date --> The date that the workout is being created (YYYY:MM:DD
format)
        start_time - The time the workout began (24:00 Format)
        end_time --> The time the workout ended (24:00 Format)
        duration_minutes --> The duration of the workout in minutes
(integer)


    DESCRIPTION:
        This function takes all of the details of a sports activity entry
and calls a function to
        query the database with an "INSERT" function with the input
information to create a new entry
        in the "SportActivity" table in the database.

    RETURNS:
        Returns the ID of the Sport Activity entry created. Reports an
error to console if unsuccessful.

    */
    async createSportsActivity(user_id, sport, date, duration_minutes,
start_time, end_time){
        try{
```

```
            const id = await this.db.create('SportActivity', [
                { column: 'user_id', value: user_id },
                { column: 'sport', value: sport },
                { column: 'date', value: date },
                { column: 'duration_minutes', value: duration_minutes },
                { column: 'start_time', value: start_time},
                { column: 'end_time', value: end_time}
            ])
            return id;
        } catch (error) {
            console.error('Error creating a new sports activity entry: ',
error);
        }
    }

    /*

    NAME:
        updateSportActivity() - Updates a SportActivity entry in the
database

    SYNOPSIS:
        async updateSportActivity(activity_id, sport, duration_minutes,
start_time, end_time);

        activity_id --> The ID of the activity that will be
updated/changed. (integer)
        sport --> The name of the sport being logged (text)
        start_time - The time the workout began (24:00 Format)
        end_time --> The time the workout ended (24:00 Format)
        duration_minutes --> The duration of the workout in minutes
(integer)

    DESCRIPTION:
        This function takes all of the details of a sports activity entry
(minus the date because that doesn't change)
        and calls a function to query the database with an "UPDATE"
function with the input information to change
        the details of the sports activity entry under the given workout
ID.
```

```
    RETURNS:
        Returns the ID of the entry in the table changed. Reports an error
to console if unsuccessful.

    */
    async updateSportsActivity(activity_id, sport, duration_minutes,
start_time, end_time){
        try{
            const id = await this.db.update('SportActivity', [
                { column: 'sport', value: sport },
                { column: 'duration_minutes', value: duration_minutes },
                { column: 'start_time', value: start_time},
                { column: 'end_time', value: end_time}
            ], [{ column: 'id', value: activity_id }]);
            return id;
        } catch (error) {
            console.error('Error updating sports activity entry: ',
error);
        }

    }

    /*

    NAME:
        addUserExercise() - Creates/Adds a user exercise entry in the
database

    SYNOPSIS:
        async addUserExercise(workout_id, exercise_name, classification,
sets, reps, weight);

        workout_id --> The ID of the workout associated with this certain
exercise (integer)
        exercise_name --> The name of the exercise being logged (text)
        classification --> The type (main, accessory) of the exercise
being logged (text)
        sets --> The amount of sets of the exercise (integer)
        reps --> The amount of reps of the exercise (integer)
```

```
        weight --> The weight of the exercise (integer)

    DESCRIPTION:
        This function takes all of the details of an exercise entry and
calls a function to
        query the database with an "INSERT" function with the input
information to create an
        entry in the 'UserExercises' table.

    RETURNS:
        Returns of ID of the entry in the table created. Reports an error
to console if unsuccessful.

    */
    async addUserExercise(workout_id, exercise_name, classification, sets,
reps, weight){
        try {
            const id = await this.db.create('UserExercises', [
                { column: 'workout_id', value: workout_id },
                { column: 'exercise_name', value: exercise_name },
                { column: 'classification', value: classification },
                { column: 'sets', value: sets },
                { column: 'reps', value: reps },
                { column: 'weight', value: weight }
            ])
            return id;
        } catch (error) {
            console.error('Error creating a new exercise in user
exercises: ', error);
        }
    }

    /*

    NAME:
        updateUserExercise() - Updates a user exercise entry in the
database

    SYNOPSIS:
```

```
        async updateUserExercise(exercise_id, workout_id, exercise_name,
classification, sets, reps, weight);

        exercise_id --> The ID of the exercise entry that will be
updated/changed. (integer)
        workout_id --> The ID of the workout associated with the entry
(integer)
        exercise_name --> The name of the exercise being logged (text)
        sets --> The amount of sets of the exercise (integer)
        reps --> The amount of reps of the exercise (integer)
        weight --> The weight of the exercise (integer)

    DESCRIPTION:
        This function takes all of the details of a user exercise entry
and calls a function to query the database with
        an "UPDATE" function with the input information to change the
details of the user exercise entry under the given
        exercise ID.

    RETURNS:
        Reports an error to console if unsuccessful.

    */
    async updateUserExercise(exercise_id, workout_id, exercise_name,
classification, sets, reps, weight){
        try {
            const id = await this.db.update('userExercises', [
                { column: 'exercise_name', value: exercise_name },
                { column: 'classification', value: classification },
                { column: 'sets', value: sets },
                { column: 'reps', value: reps },
                { column: 'weight', value: weight }
            ], [{ column: 'id', value: exercise_id }]);
        } catch (error) {
            console.error('Error updating exercise entry: ',
error.message);
        }
    }

    /*
```

```
    NAME:
        deleteUserExercise() - Deletes a user exercise entry in the
database

    SYNOPSIS:
        async deleteUserExercise(exercise_id);

        exercise_id --> The ID of the exercise to be deleted (integer)

    DESCRIPTION:
        This function takes an ID of an exercise, then calls a function to
query the database with a
        "DELETE" function to delete the entry with the given ID.

    RETURNS:
        Reports an error if unsuccessful.

    */
    async deleteUserExercise(exercise_id){
        try {
            await this.db.delete('userExercises', [{ column: 'id', value:
exercise_id}])
        } catch (error) {
            console.error('Error deleting user exercise: ',
error.message);
        }
    }

    /*

    NAME:
        fillExercisesTable() - Fills the Exercises database table from a
JSON file

    SYNOPSIS:
        async fillExercisesTable();

    DESCRIPTION:
```

```
        This function checks if the 'Exercises' table is empty, and if it
is not, it parses the data
        from the 'exercises.json' file and stores all of it into the
'Exercises' table.

    RETURNS:
        Reports an error if unsuccessful.

    */
    async fillExercisesTable(){
        //Checks if Exercises is empty, because if it is not empty then we
are just re-entering data again unecessarily
        const empty = await this.db.isTableEmpty('Exercises');

        //If Exercises is empty, we fill the table from the JSON file.
        if(empty){
            try{
                const jsonData = fs.readFileSync('exercises.json',
'utf-8');

                const parsedData = JSON.parse(jsonData);

                //Loop through the parsed data from the JSON (the
exercises) and add them to the table by calling create
                parsedData.forEach(exercise => {
                    this.db.create('Exercises', [
                        { column: 'name', value: exercise.name },
                        { column: 'classification', value:
exercise.classification },
                        { column: 'muscleGroups', value:
exercise.muscleGroups.join(', ')},
                        { column: 'bodyPart', value: exercise.bodyPart }
                    ])
                });
            } catch (error) {
                console.error('Error reading or parsing JSON file: ',
error);
            }
        }
    }
```

```
    /*

    NAME:
        fillSportsTable() - Fills the Exercises database table from a JSON
file

    SYNOPSIS:
        async fillSportsTable();

    DESCRIPTION:
        This function checks if the 'Sports' table is empty, and if it is
not, it parses the data
        from the 'sports.json' file and stores all of it into the 'Sports'
table.

    RETURNS:
        Reports an error if unsuccessful.

    */
    async fillSportsTable(){
        //Checks if sports is empty, because if it is not empty then we
are just ren-entering data again unecessarily
        const empty = await this.db.isTableEmpty('Sports');

        if(empty){
            try {
                const jsonData = fs.readFileSync('sports.json', 'utf-8');

                const parsedData = JSON.parse(jsonData);

                //Loop through the parsed data from the JSON (the sports)
and add them to the table by calling create
                parsedData.forEach(sport => {
                    this.db.create('Sports', [
                        { column: 'sport', value: sport.name }
                    ]);
                });
            } catch (error) {
```

```
                console.error('Error reading or parsing JSON file: ',
error);
            }
        }
    }

    /*

    NAME:
        createUser() - Creates a user in the database

    SYNOPSIS:
        async createUser(first, last, username, password);

        first --> The user's first name (text)
        last --> The user's last name (text)
        username --> The user's username (text)
        password --> The user's hashed password (text)

    DESCRIPTION:
        This function takes all of the details of an new user and calls a
function to
        query the database with an "INSERT" function with the input
information to create a user
        in the 'Users' table.

    RETURNS:
        The ID of the user created. Reports an error if unsuccessful.

    */
    async createUser(first, last, username, password){
        try{
            const id = await this.db.create('Users', [
                { column: 'first_name', value: first },
                { column: 'last_name', value: last },
                { column: 'username', value: username },
                { column: 'password', value: password }
            ])
            return id;
        } catch (error) {
```

```
            console.error('Error adding user:', error.message);
        }
    }

    /*
    NAME:
        updateUser() - Updates a user in the database

    SYNOPSIS:
        async updateUser(id, first, last, username);

        id --> The ID of the user being updated (integer)
        first --> The user's first name to be potentially changed (text)
        last --> The user's last name to be potentially changed (text)
        username --> The user's username to be potentially changed (text)

    DESCRIPTION:
        This function takes all of the details of a user and calls a
function to query the database with an
        "UPDATE" function with the input information to change the details
of the user under the given
        user ID.

    RETURNS:
        Reports an error to console if unsuccessful.

    */
    async updateUser(id, first, last, username){
        try{
            await this.db.update('Users', [
                { column: 'first_name', value: first },
                { column: 'last_name', value: last },
                { column: 'username', value: username }
            ], [{ column: 'id', value: id }]);
        } catch (error) {
            console.error("Error updating user: ", error.message);
        }
    }

    /*
```

```
    NAME:
        updateUserPassword() - Updates a user's password in the database

    SYNOPSIS:
        async updateUser(id, password);

        id --> The ID of the user being updated (integer)
        password --> The new password of the user.

    DESCRIPTION:
        This function takes all of the details of a user and calls a
function to query the database with an
        "UPDATE" function with the input information to change the details
of the user under the given
        user ID.

    RETURNS:
        Reports an error to console if unsuccessful.

    */
    async updateUserPassword(id, password){
        try{
            await this.db.update('Users', [
                { column: 'password', value: password }
            ], [{ column: 'id', value: id }]);
        } catch (error) {
            console.error("Error updating user password: ",
error.message);
        }
    }

    /*
    NAME:
        findUserByUsername() - Finds a user from the database given their
username

    SYNOPSIS:
        async findUserByUsername(username);
```

```
        username --> The user's username to be used as a search query
(text)

    DESCRIPTION:
        This function takes the username of a user and calls a function to
query the database for the user
        with the stated username.

    RETURNS:
        Returns an object of the user's details. Reports an error to
console if unsuccessful.

    */
    async findUserByUsername(username) {
        try {
            const user = await this.db.read('Users', [{ column:
'username', value: username }]);
            console.log(user);
            return user;
        } catch (error) {
            console.error('Error finding user by username: ',
error.message);
        }
    }

    /*
    NAME:
        findUserById() - Finds a user from the database given their id

    SYNOPSIS:
        async findUserById(id);

        id --> The user's ID to be used as a search query (text)

    DESCRIPTION:
        This function takes the ID of a user and calls a function to query
the database for the user
        with the stated user ID.

    RETURNS:
```

```
        Returns an object of the user's details. Reports an error to
console if unsuccessful.

    */
    async findUserById(id) {
        try {
            const user = await this.db.read('Users', [{ column: 'id',
value: id }]);
            return user;
        } catch (error) {
            console.error('Error finding user by id: ', error.message);
        }

    }

    /*
    NAME:
        findWorkoutById() - Finds a workout from the database given its ID

    SYNOPSIS:
        async findWorkoutById(id);

        id --> The workout's ID to be used as a search query (text)

    DESCRIPTION:
        This function takes the ID of a workout and calls a function to
query the database for the workout
        with the stated ID.

    RETURNS:
        Returns an object of the workout's details. Reports an error to
console if unsuccessful.

    */
    async findWorkoutById(id) {
        try {
            const workout = await this.db.read('Workouts', [{ column:
'id', value: id }]);
            return workout;
        } catch (error) {
```

```
            console.error('Error finding the workout by id: ',
error.message);
        }
    }

    /*
    NAME:
        findUserExerciseById() - Finds a user exercise from the database
given its ID.

    SYNOPSIS:
        async findUserExerciseById(id);

        id --> The user exercises's ID to be used as a search query (text)

    DESCRIPTION:
        This function takes the ID of a user logged exercise and calls a
function to query the database for the user
        exercise with the stated ID.

    RETURNS:
        Returns an object of the user exercise's details. Reports an error
to console if unsuccessful.

    */
    async findUserExerciseById(id) {
        try {
            const exercise = await this.db.read('userExercises', [{
column: 'id', value: id}]);
            return exercise;
        } catch (error) {
            console.error('Error in finding user Exercise by ID: ',
error.message);
        }
    }

    /*
    NAME:
        findSportsActivityById() - Finds a sports activity from the
database given its ID.
```

```
    SYNOPSIS:
        async findSportsActivityById(id);

        id --> The user sport activity's ID to be used as a search query
(text)

    DESCRIPTION:
        This function takes the ID of a user logged sports activity and
calls a function to query the database for the user
        logged sports activity with the stated ID.

    RETURNS:
        Returns an object of the sports activity's details. Reports an
error to console if unsuccessful.

    */
    async findSportsActivityById(id) {
        try {
            const sportActivity = await this.db.read('SportActivity', [{
column: 'id', value: id }]);
            return sportActivity;
        } catch (error) {
            console.error('Error finding the sport activity by id: ',
error.message);
        }
    }

    /*
    NAME:
        getAllUsers() - Gets all of the user's from the 'Users' table.

    SYNOPSIS:
        async getAllUsers();

    DESCRIPTION:
        This function calls a function to query the 'Users' table and
retreives all of the entries in that table.

    RETURNS:
```

```
        Returns all of the users in the table in the form of an array of
objects. Reports an error to console if unsuccessful.

    */
    async getAllUsers(){
        try {
            const users = await this.db.getAll('Users');
            return users;
        } catch (error) {
            console.error("Error finding all users: ", error.message);
        }
    }


    /*

    NAME:
        getUserFirstLast(); - Gets the user's first and last name from
database

    SYNOPSIS:
        async getUserFirstLast(id);

        id --> The ID of the user being searched for (integer)

    DESCRIPTION:
        This function takes a user's ID and calls a function to find a
user by the ID to retrive the
        user's information.

    RETURNS:
        Returns the user's first and last name.

    */
    async getUserFirstLast(id){
        user = this.findUserById(id);
        return user.first + user.last;
    }

    /*
```

```
    NAME:
        getExercises() - Gets all of the exercises from the 'Exercises'
table.

    SYNOPSIS:
        async getExercises();

    DESCRIPTION:
        This function calls a function to query the 'Exercises' table and
retrieves all of the entries in that table.

    RETURNS:
        Returns all of the exercises in the table in the form of an array
of objects. Reports an error to console if unsuccessful.

    */
    async getExercises(){
        try{
            let exercises = await this.db.getAll('Exercises');
            return exercises;
        } catch (error) {
            console.error('Error retrieving exercises:', error.message);
        }
    }

    //Finds the exercise from the exercises table by the name
    /*
    NAME:
        findExerciseByName() - Finds an exercise from the 'Exercises'
table by the name

    SYNOPSIS:
        async findExerciseByName(exercise_name);

        exercise_name --> The name of the exercise to be searched for
(text)

    DESCRIPTION:
        This function takes an exercise name, and calls a function to
query the 'Exercises' table to retrieve the information
```

```
        of the exercise with that name.

    RETURNS:
        Returns the exercise information from the table in the form of an
object. Reports an error to console if unsuccessful.

    */
    async findExerciseByName(exercise_name){
        try{
            const exercise = await this.db.read('Exercises', [{ column:
'name', value: exercise_name }]);
            return exercise;
        } catch (error) {
            console.error('Error finding the exercise by name: ',
error.message);
        }
    }

    /*
    NAME:
        getSports() - Gets all of the sports from the 'Sports' table.

    SYNOPSIS:
        async getSports();

    DESCRIPTION:
        This function calls a function to query the 'Sports' table and
retrieves all of the entries in that table.

    RETURNS:
        Returns all of the sports in the table in the form of an array of
objects. Reports an error to console if unsuccessful.

    */
    async getSports(){
        try{
            let sports = await this.db.getAll('Sports');
            return sports;
        } catch (error) {
            console.error('Error retreiving sports: ', error.message);
```

```
        }
    }

    /*
    NAME:
        getAllWorkoutExercises() - Gets all of the exercises from a
workout from database

    SYNOPSIS:
        async getAllWorkoutExercises(workout_id);

        workout_id --> The ID of the workout the exercises are associated
with (integer)

    DESCRIPTION:
        This function calls a function to query the 'UserExercises' table
and retrieves all of the exercises with stated workout ID.

    RETURNS:
        Returns all of the exercises with the set workout ID in the form
of an array of objects. Reports an error to console if unsuccessful.

    */
    async getAllWorkoutExercises(workout_id){
        try {
            let exercises = await this.db.getAllWhere('UserExercises', [{
column: 'workout_id', value: workout_id }]);
            return exercises;
        } catch (error) {
            console.error('Error retreiving workout exercises from
userExercises: ', error.message);
        }
    }

    /*
    NAME:
        getAllWorkouts() - Gets all of the workouts from a specified user
from database

    SYNOPSIS:
```

```
        async getAllWorkouts(id);

        id --> The ID of the user (integer)

    DESCRIPTION:
        This function calls a function to query the 'Workouts' table and
retrieves all of the workouts with stated user ID.

    RETURNS:
        Returns all of the workouts with the set user ID in the form of an
array of objects. Reports an error to console if unsuccessful.

    */
    async getAllWorkouts(id){
        const user = this.findUserById(id);

        try{
            let workouts = await this.db.getAllWhere('Workouts', [{
column: 'user_id', value: id }]);
            return workouts;
        } catch (error) {
            console.error('Error retreiving the user workouts: ',
error.message);
        }
    }

    /*
    NAME:
        getAllSportsActivity() - Gets all of the sports activity logs from
a specified user from database

    SYNOPSIS:
        async getAllSportsActivity(id);

        id --> The ID of the user (integer)

    DESCRIPTION:
        This function calls a function to query the 'SportActivity' table
and retrieves all of the sports activity
        log with the stated workout ID.
```

```
    RETURNS:
        Returns all of the sports activity logs with the set user ID in
the form of an array of objects. Reports an error to console if
unsuccessful.

    */
    async getAllSportsActivity(id){
        const user = this.findUserById(id);

        try{
            let sports = await this.db.getAllWhere('SportActivity', [{
column: 'user_id', value: id }]);
            return sports;
        } catch (error) {
            console.error('Error retreiving the user sports activity logs:
', error.message);
        }
    }

    /*
    NAME:
        getAllWorkoutsForWeek() - Gets all of the workouts of a user in
set week from database

    SYNOPSIS:
        async getAllWorkoutsForWeek(id, start_date, end_date);

        id --> The ID of the user (integer)
        start_date --> The first date of the week being searched
(YYYY:MM:DD Format)
        end_date --> The last date of the week being searched (YYYY:MM:DD
Format)

    DESCRIPTION:
        This function takes the user ID, and the start/end date of the
specified week, and calls a function to query the 'workouts' table
        in the database to return all workouts between the two dates.

    RETURNS:
```

```
        Returns an array of objects containing the workouts in specified
week. Reports an error to console if unsuccessful.

    */
    async getAllWorkoutsForWeek(id, start_date, end_date){
        try {
            let workouts = await this.db.getAllInRange('Workouts', id, [
                { column: 'date', value: start_date },
                { column: 'date', value: end_date }
            ]);
            return workouts;
        } catch (error) {
            console.error('Error retrieving all workouts within set range:
', error.message);
        }
    }

    /*
    NAME:
        getAllSportsForWeek() - Gets all of the workouts of a user in set
week from database

    SYNOPSIS:
        async getAllSportsForWeek(id, start_date, end_date);

        id --> The ID of the user (integer)
        start_date --> The first date of the week being searched
(YYYY:MM:DD Format)
        end_date --> The last date of the week being searched (YYYY:MM:DD
Format)

    DESCRIPTION:
        This function takes the user ID, and the start/end date of the
specified week, and calls a function to query the 'SportActivity' table
        in the database to return all logged sport activities between the
two dates.

    RETURNS:
        Returns an array of objects containing the sports activies in
specified week. Reports an error to console if unsuccessful.
```

```
    */
    async getAllSportsForWeek(id, start_date, end_date){
        try {
            let sports = await this.db.getAllInRange('SportActivity', id,
[
                { column: 'date', value: start_date },
                { column: 'date', value: end_date }
            ]);
            return sports;
        } catch (error) {
            console.error('Error retrieving all sport activites within set
range: ', error.message);
        }
    }

    /*
    NAME:
        getAllFriends() - Gets all of the friends of a certain user from
database

    SYNOPSIS:
        async getAllFriends(user_id);

        user_id --> The ID of the user (integer)

    DESCRIPTION:
        This function takes the user ID, and calls a function to query the
'Friends' table and search for all entries
        where the user_id is the same as the specified user_id.

    RETURNS:
        Returns an array of all the user's friends. Reports an error to
console if unsuccessful.

    */
    async getAllFriends(user_id){
        try {
            const friends = await this.db.getAllWhere('Friends', [{
column: 'user_id', value: user_id }]);
```

```
            return friends;
        } catch (error) {
            console.error('Error finding all friends of certain user: ',
error.message);
        }
    }

    /*

    NAME:
        addFriend(); - Creates an entry in the 'Friends' table

    SYNOPSIS:
        async addFriend(user_id, friend_id);

        user_id --> The ID of the user adding a friend (integer)
        friend_id --> The ID of the user that is being added as a friend
(integer)

    DESCRIPTION:
        This function takes the User ID and the friend's ID and calls a
function to query the 'Friends'
        table with an "INSERT" to create an entry with the user ID and
friend ID.

    RETURNS:
        The ID of the entry created. Reports an error if unsuccessful.

    */
    async addFriend(user_id, friend_id){
        try {
            const id = await this.db.create('Friends', [
                { column: 'user_id', value: user_id },
                { column: 'friend_id', value: friend_id }
            ])
            return id;
        } catch (error) {
            console.error('Error adding friend: ', error.message);
        }
    }
```

```
    /*

    NAME:
        removeFriend(); - Removes an entry from the 'Friends' table

    SYNOPSIS:
        async removeFriend(user_id, friend_id);

        user_id --> The ID of the user removing a friend (integer)
        friend_id --> The ID of the user that is being removed as a friend
(integer)

    DESCRIPTION:
        This function takes the User ID and the friend's ID and calls a
function to query the 'Friends'
        table with an "DELETE" to delete the entry with the two IDs to
'remove' them as friends

    RETURNS:
        Reports an error if unsuccessful.

    */
    async removeFriend(user_id, friend_id){
        try {
            await this.db.delete('Friends', [
                { column: 'user_id', value: user_id },
                { column: 'friend_id', value: friend_id }
            ])
        } catch (error) {
            console.error("Error removing friend: ", error.message);
        }
    }

    /*

    NAME:
        checkFriendStatus(); - Checks the status of two user IDs to see if
they are friends.
```

```
    SYNOPSIS:
        async checkFriendStatus(user_id, friend_id);

        user_id --> The ID of the user (integer)
        friend_id --> The ID of the friend (integer)

    DESCRIPTION:
        This function takes the User ID and the friend's ID and calls a
function to query the 'Friends'
        table to find the entry with both user_id and friend_id.

    RETURNS:
        The ID of the table entry found. Reports an error if unsuccessful.

    */
    async checkFriendStatus(user_id, friend_id){
        try {
            const id = await this.db.read('Friends', [
                { column: 'user_id', value: user_id },
                { column: 'friend_id', value: friend_id }
            ])
            return id;
        } catch (error) {
            console.error("Error checking friend status: ",
error.message);
        }
    }

    close() {
      this.db.close();
    }
  }


module.exports = UserDB;
```

Gymbuds.js

```
//Declare the requirements
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const crypto = require('crypto');
const bcrypt = require('bcryptjs');

const UserDB = require('./userDB');
const db = new UserDB('./gymbuds.db');

//Initializes the database and makes all of the needed tables in the
database.
db.initialize();
db.makeUserTable();
db.makeWorkoutTable();
db.makeExercisesTable();
db.makeUserExercisesTable();
db.makeFriendsTable();
db.makeSportsActivityTable();
db.makeSportsTable();
db.makeRecapTable();

//Only happens on creation - fills the necessary tables from the JSON
files.
db.fillExercisesTable();
db.fillSportsTable();

//Declare the express app
const app = express();
//This makes it so the HTML is in a human-readable format when rendered
app.locals.pretty = true;

app.use(express.urlencoded({ extended : true}));
app.use(express.static('public'));
app.use(bodyParser.json());

//Generate a secure random secret key
const secret = crypto.randomBytes(64).toString('hex');
```

```
// Gets call on every request, before the routes.
// We can inject dependencies into the req (or res)
// so the routes have access to them.
app.use((req, res, next) => {
    console.log("Adding DB to request");
    req.db = db;
    next();
})

//Use express-session with the crypto middleware to have a randomly
generated secret
app.use(session({
    secret: secret,
    resave: false,
    saveUninitialized: true,
    cookie: { secure: false }
}))

app.use((req, res, next) => {
    if(req.session.user){
        res.locals.user = {
            id: req.session.user.id,
            username: req.session.user.username
        }
    }
    next()
})

app.set('view engine', 'pug');

//This tells express to read all of the javascript files
app.use('/', require('./routes/accounts'))
app.use('/', require('./routes/home'))
app.use('/', require('./routes/dashboard'))
app.use('/', require('./routes/workoutLog'))
app.use('/', require('./routes/recap'))
app.use('/', require('./routes/friends'))

//This tells express to read the service javascript files
```

```
const { calculateWeeklyBreakdown } =
require('./services/weeklyBreakdown');

//This renders a custom page for 404 errors.
app.use((req, res, next) => {
    res.status(404).render('404');
})

app.listen(8080, () => {
    console.log('Server is running on port 8080')
});
```

Exercises.json

```
[
    {
        "name": "Squat",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Glutes", "Hamstrings"],
        "bodyPart": "Lower Body"
    },
    {
        "name": "Bench Press",
        "classification": "Main",
        "muscleGroups": ["Chest", "Shoulders", "Triceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Deadlift",
        "classification": "Main",
        "muscleGroups": ["Lower Back", "Glutes", "Hamstrings"],
        "bodyPart": "Lower Body"
    },
    {
        "name": "Pull-Up",
        "classification": "Accessory",
        "muscleGroups": ["Back", "Biceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Leg Press",
        "classification": "Accessory",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes"],
        "bodyPart": "Lower Body"
    },
    {
        "name": "Shoulder Press",
        "classification": "Accessory",
        "muscleGroups": ["Shoulders", "Triceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Snatch",
```

```
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes", "Back",
"Shoulders", "Traps"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Clean and Jerk",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes", "Back",
"Shoulders", "Traps", "Triceps"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Overhead Squat",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Glutes", "Hamstrings",
"Shoulders", "Triceps", "Core"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Clean",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes", "Back",
"Shoulders", "Traps", "Triceps"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Barbell RDL",
        "classification": "Accessory",
        "muscleGroups": ["Hamstrings", "Glutes", "Lower Back"],
        "bodyPart": "Lower Body"
    },
    {
        "name": "Dumbbell RDL",
        "classification": "Accessory",
        "muscleGroups": ["Hamstrings", "Glutes", "Lower Back"],
        "bodyPart": "Lower Body"
    },
    {
        "name": "Incline Barbell Bench",
```

```
        "classification": "Accessory",
        "muscleGroups": ["Chest", "Shoulders", "Triceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Incline DB Bench",
        "classification": "Accessory",
        "muscleGroups": ["Chest", "Shoulders", "Triceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Tricep Pushdown",
        "classification": "Accessory",
        "muscleGroups": ["Triceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Bicep Curls",
        "classification": "Accessory",
        "muscleGroups": ["Biceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Lat Pulldowns",
        "classification": "Accessory",
        "muscleGroups": ["Back", "Biceps"],
        "bodyPart": "Upper Body"
    },
    {
        "name": "Hang Cleans",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes", "Back",
"Shoulders", "Traps"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Front Squats",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Glutes", "Hamstrings", "Core"],
        "bodyPart": "Total Body"
```

```
    },
    {
        "name": "Safety Bar Squats",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Glutes", "Hamstrings", "Lower
Back", "Core"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Hang Snatch Pull",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes", "Back",
"Shoulders", "Traps"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Hang Clean Pull",
        "classification": "Main",
        "muscleGroups": ["Quadriceps", "Hamstrings", "Glutes", "Back",
"Shoulders", "Traps"],
        "bodyPart": "Total Body"
    },
    {
        "name": "Rows",
        "classification": "Accessory",
        "muscleGroups": ["Back", "Biceps"],
        "bodyPart": "Upper Body"
    }
]
```

Sports.json

```
[
    {
        "name": "Soccer"
    },
    {
        "name": "Basketball"
    },
    {
        "name": "Tennis"
    },
    {
        "name": "Baseball"
    },
    {
        "name": "Volleyball"
    },
    {
        "name": "Swimming"
    },
    {
        "name": "Running"
    },
    {
        "name": "Cycling"
    },
    {
        "name": "Golf"
    },
    {
        "name": "Badminton"
    },
    {
        "name": "Table Tennis"
    },
    {
        "name": "Yoga"
    },
    {
        "name": "Hiking"
```

```
    },
    {
        "name": "Skiing"
    },
    {
        "name": "Snowboarding"
    },
    {
        "name": "Surfing"
    },
    {
        "name": "Skateboarding"
    },
    {
        "name": "Rock Climbing"
    },
    {
        "name": "Bowling"
    },
    {
        "name": "Cricket"
    },
    {
        "name": "Rugby"
    },
    {
        "name": "Softball"
    },
    {
        "name": "Dancing"
    },
    {
        "name": "Martial Arts"
    },
    {
        "name": "Boxing"
    },
    {
        "name": "Kickboxing"
    },
```

```
    {
        "name": "Pilates"
    },
    {
        "name": "Horseback Riding"
    },
    {
        "name": "Kayaking"
    },
    {
        "name": "Rowing"
    }
]
```

Style.css

```
/* Background gradient color */
body {
    background: linear-gradient(to top, #1a1353, #3a3939) !important; /*
Gradient colors */
  min-height: 100vh !important; /* Ensure gradient covers the whole
viewport height */
}

/* Custom navbar css */
.navbar {
    background-image: linear-gradient(55deg, rgb(81, 6, 104), black);
}

/* This is the custom css for the login container */
.login-card{
    background-color: lightblue;
    color: black;
}

/* This is the custom css for the profile icon in the rop right of the
navbar */
.profile-icon{
    font-size: 20px;
    color: white;
}

/* This is the custom css for the profile settings container */
.profile-card{
    background-color: lightblue;
    color: black;
}

/* Custom css for the dropdown of exercises */
.scrollable-dropdown {
    max-width: 8rem;
    max-height: 200px;
    overflow-y: auto;
}
```

```
.scrollable-dropdown-sports {
    width: auto;
}

.container-submissionContainer {
    display: flex;
    justify-content: center; /* Center horizontally */
    align-items: center; /* Center vertically */
}

.col-saveButton {
    background-color: #010407;
    color: #000000;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

/* This is for the dahsboard, to make the workout link cards look better
and be a bit more dynamic */
.workout-card {
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.workout-card:hover {
    transform: scale(1.05);
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
}

.workout-card {
    background-color: #f8f9fa; /* Light background color for the workout
cards */
    color: black;
    border: 1px solid #ddd;
    border-radius: 0.5rem;
}

/* This is for the public profile page, to make the card look a bit more
dynamic */
.card-profile {
```

```
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    background-color: white
}

.card-profile:hover {
    transform: scale(1.05);
    box-shadow: 0 8px 16px rgba(216, 211, 211, 0.2);
}

.progress-bar {
    border: 2px solid #333;
    box-sizing: border-box;
}

.progress-bar:hover {
    transform: scale(1.05);
}

/* All for user search page */
.user-list{
    max-height: 200px;
    overflow-y: auto;
}

.user-item{
    padding: 10px;
    border-bottom: 1px solid #ccc;
    cursor: pointer;
    background-color: white;
    text-decoration: black;
}

.user-item:hover{
    background-color: #7cc9ec;
}
```