# *SOFTWARE PROJECT FINAL REPORT*

Ethan Honis

Date: December 3, 2024

## 1. Introduction

### 1.1. **Purpose and Scope**

#### Purpose

ClimbTrak is designed to empower rock climbers by providing a platform for tracking personal progress and managing route data. This is necessary since there is no tracking software for indoor climbing. By integrating authentication, it ensures secure access to user-specific data, offering a personalized and efficient experience for climbers of all levels.

#### Scope

The core functionalities, including user authentication, route logging and grading, and progress tracking. Built with technologies like React, Nextjs, Tailwind, Prisma, and Auth.js, ClimbTrak prioritizes scalability, security, and user-friendly design. The current scope emphasizes climbers and gyms. Custom tailoring to the On the Rocks Climbing gym in Elyria, Ohio.

### 1.2. **Product Overview**

#### Capabilities

User Authentication: Secure access to personal data through modern authentication methods.

Route Logging: Detailed records of climbing routes, including grades, send attempts, location, color, and time it was done.

Progress Tracking: Graphs and other charts detailing your progress as a climber

User Interface: Clean and intuitive design to make sure the user has an easy time navigating through pages and interacting.

Data Integrity: Type checking with Prisma to ensure the user data is held securely and accurately.

#### Scenarios

Personal Goal Setting: Climbers can set a personal goal for themselves and see their progress as time goes on.

Route Analysis: Users can have a digital memory of all the routes they completed so they can review and make changes to their climbing to climb more effectively.

Grading Routes: Users can grade routes to show other climbers what the actual grade might be. Grading can be subjective but having a way to average all the grades will have a better time when attempting a route

## 2. Project Management Plan

### 2.1. Lifecycle Model Used

Agile development lifecycle was chosen for iterative and incremental changes based on user feedback.

### 2.2. Risk Analysis

Security Risks: Potential vulnerabilities is user authentication.

Mitigation: Using Auth.js and industry best practices to ensure user data is handled correctly

Scope Creep: Adding features that could delay the schedule

Mitigation: Sticking to a fixed schedule and have proper sprint plan.

Technology Risks: Bugs or compatibility issues in third-party libraries.
Mitigation: Regular updates and fallbacks for critical dependencies

Data Loss: When deploying, user data could be lost.

Mitigation: Regular backups

### 2.3. Hardware and Software Resource Requirements

Development Machines: Laptops or desktops that have at least 8GB RAM and 128GB of storage

User Machines: Any computer or phone that can run a browser

Server: Any hosting services that can handle user data

### 2.4. Deliverables and schedule

**Deliverables:**
1. Functional web app with user authentication, route logging and interaction, and progress tracking

2. User Manual

3. Maintenance Manual

4. Presentation

5. Project Report

**Schedule:**

August 30th – September 7th : Gathering Idea and Requirements

September 8th – November 15th : Implementation and Development

November 15th - November 30th : Testing and Debugging

Final Deployment: December 3rd

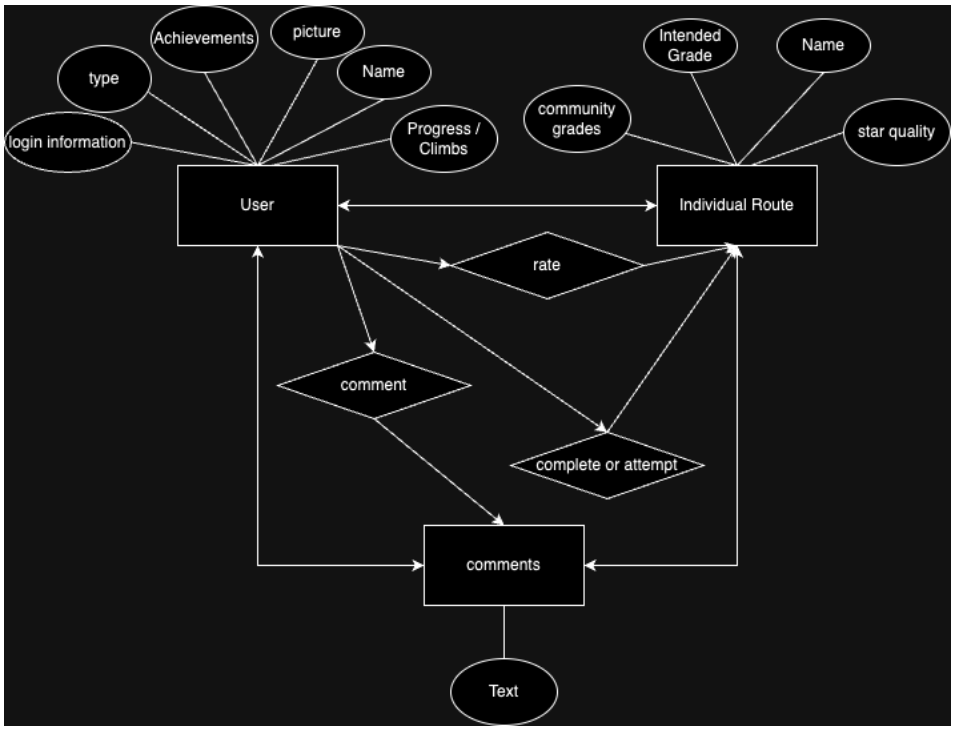
## 3. Requirement Specifications

### 3.1. Stakeholders for the system

Stakeholders for the system would be the climbers (end users), development team, and gym owners

### 3.2. Use cases

#### 3.2.1. Graphic use case model




#### 3.2.2. Textual Description for each use case

Progress logging: A user can log climbs, including the grade, date, and personal performance

Viewing Gym Routes: A user can view a list of available routes in the gym, sorted by difficulty, popularity, or setter.

Management Access: Admins can manage route data, user feedback, and gym-specific settings.

Route Feedback: A user can rate a climbing route based on difficulty and quality.

Profile Management: A user can update their personal information, profile picture, and climbing stats.

### 3.3. Rationale for your use case model

This use case model implements the core functionalities of ClimbTrak with user and admin needs. Focusing on tasks climbers frequently perform, such as logging progress or viewing routes. Also implements user feedback to improve routes and user experience.

### 3.4. Non-functional requirements

The performance must be able to a handle at least 50 users for each gym at once and database queries should execute under 1000ms. The UI must be responsive and able to be used with mobile. Codebase must also be maintainable and follow best practices for readability.

## 4. Architecture

### 4.1. **Architectural style used**

Client – Server Architecture

### 4.2. **Architectural model**

**Client (Frontend):**

Built using Next.js and Tailwind CSS.

Provides an interactive user interface for logging routes, viewing progress, and accessing data securely.

Interacts with the backend via API requests.

**Server (Backend):**

Developed with Node.js and Prisma for data handling.

APIs for authentication, data logging, and analytics.

Ensures security and efficiency in handling user data.

**Database:**

PostgreSQL is used as the relational database to store user profiles, climbing routes, and activity logs.

Prisma ORM abstracts database interactions and ensures scalability.

**Authentication Service:**

Auth.js is integrated for secure user authentication and session management.

Manages user roles and access control.

**Hosting and Deployment:**

Vercel hosts the frontend and backend services, ensuring a seamless deployment and runtime environment.

### 4.3. Technology and software

**Technology**

**Frontend:** Next.js (App Router), Tailwind CSS.

**Backend:** Node.js, Prisma ORM.

**Authentication:** Auth.js.

**Database:** PostgreSQL.

**Deployment:** Vercel.

**Software:**

**Development Tools:** Visual Studio Code, Git/Github.

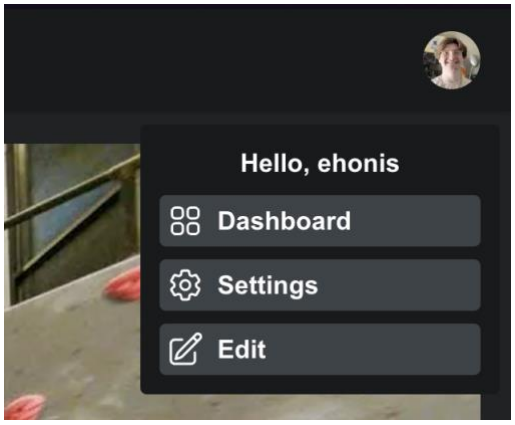
### 4.4. Rationale for your architectural style and model

The main rationale for using this architectural style is scalability, modularity, and Security. For scalability, client – server ensures the separation of how the client-side operations happen so no matter how many users are actively using the site, the server can handle it without affecting the client. For modularity, using React and Nextjs make reusing components easy and make everything work together. And for security, having the server handle user information and critical data separates the user from this information.

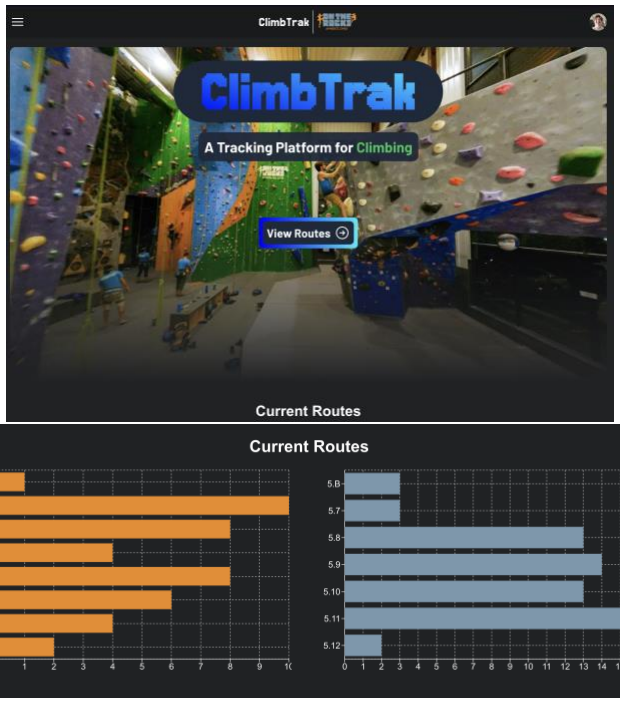
## 5. Design

### 5.1. User Interface design

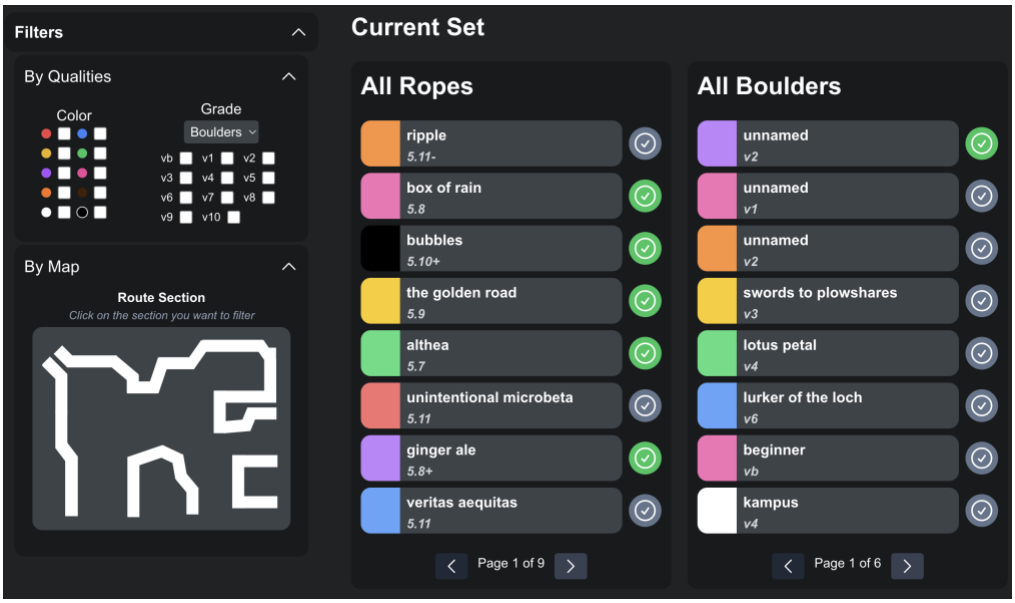
**Navbar:** This is present throughout the whole app giving the user ease of navigation. They can navigate to the routes page from the sidebar (part of the navbar), click on the logo to navigate to the home page, or go to the sign in page. If they are logged in, they go to the dashboard, settings, or edit page if they are a admin.


**Home Page:** A picture of the gym and the ClimbTrak logo on top with a short cut to the routes page. This page also has basic statistics of how many routes are currently in the gym.

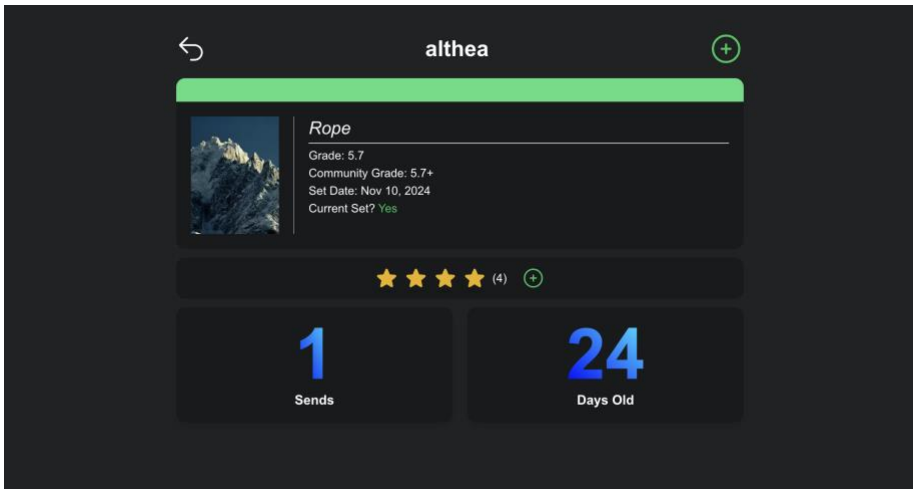


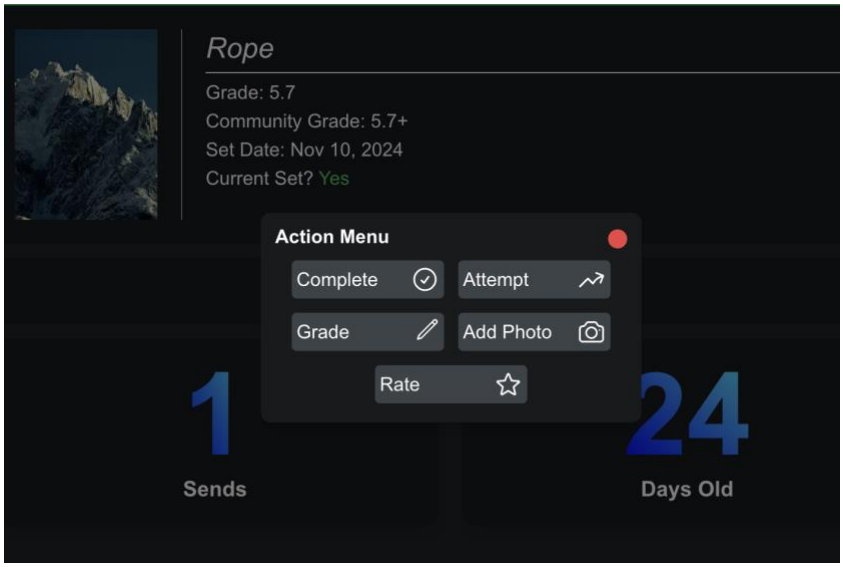

**Routes Page:** This page will render something different if the user is logged in or not. For a guest, they can see all the routes and can filter through them with a combination of colors, grade, and location. If they are logged in, they can have the option to quick complete any of the routes.



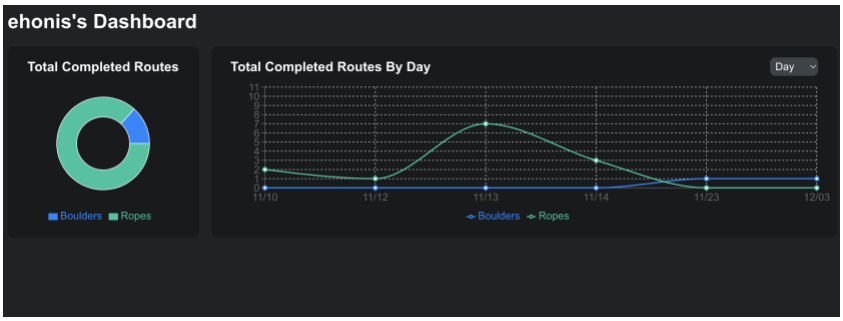

**Individual Route Page:** the individual route page is when a user clicks one of the route tiles (from above) to get more information on the route. This is where most

of the interactivity is had. There is an action button in the top right where the user can grade, complete, upload an image, attempt, and even rate the route. Exiting out of this menu you can see the following information: Name, Type, Grade, Community Grade, Set Date, Rating, Sends, And Days old. If the user is not signed in, they will not be able to use the action button.

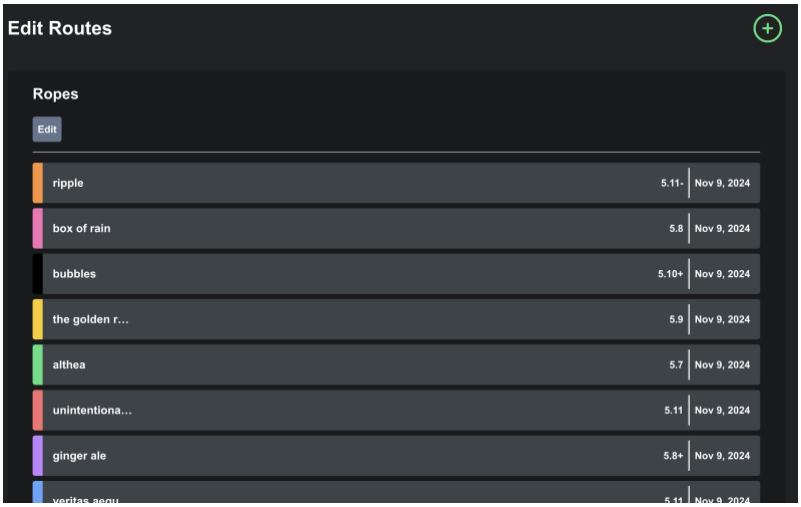



**User Dashboard:** The user dashboard is where the user can see their progress. They can see how many routes they completed and even select a time frame in which these routes where completed.

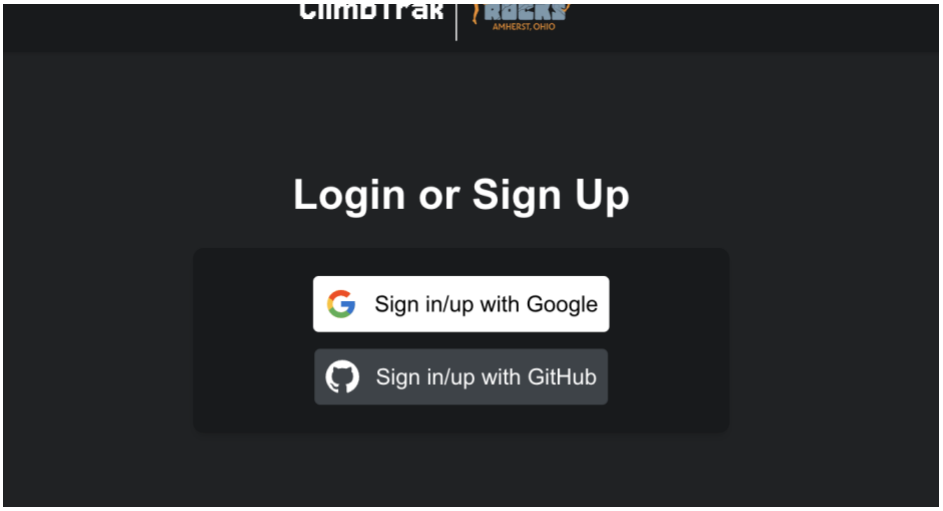



**Admin Edit Page:** A list of routes that the admin can delete or in the top right corner they can add a new route in the new page




**New Page:** This has some features that are not ready to ship yet, but the main function of adding new routes is implemented and works. You can add multiple routes.


**Sign in Page:** When a user is logged out, they will see two options. Sign in with Google or Github.




## 5.2. Components design (static and dynamic models of each component)

**Static Models: Navbar, Routes, Home, Dashboard, Edit, and New.**

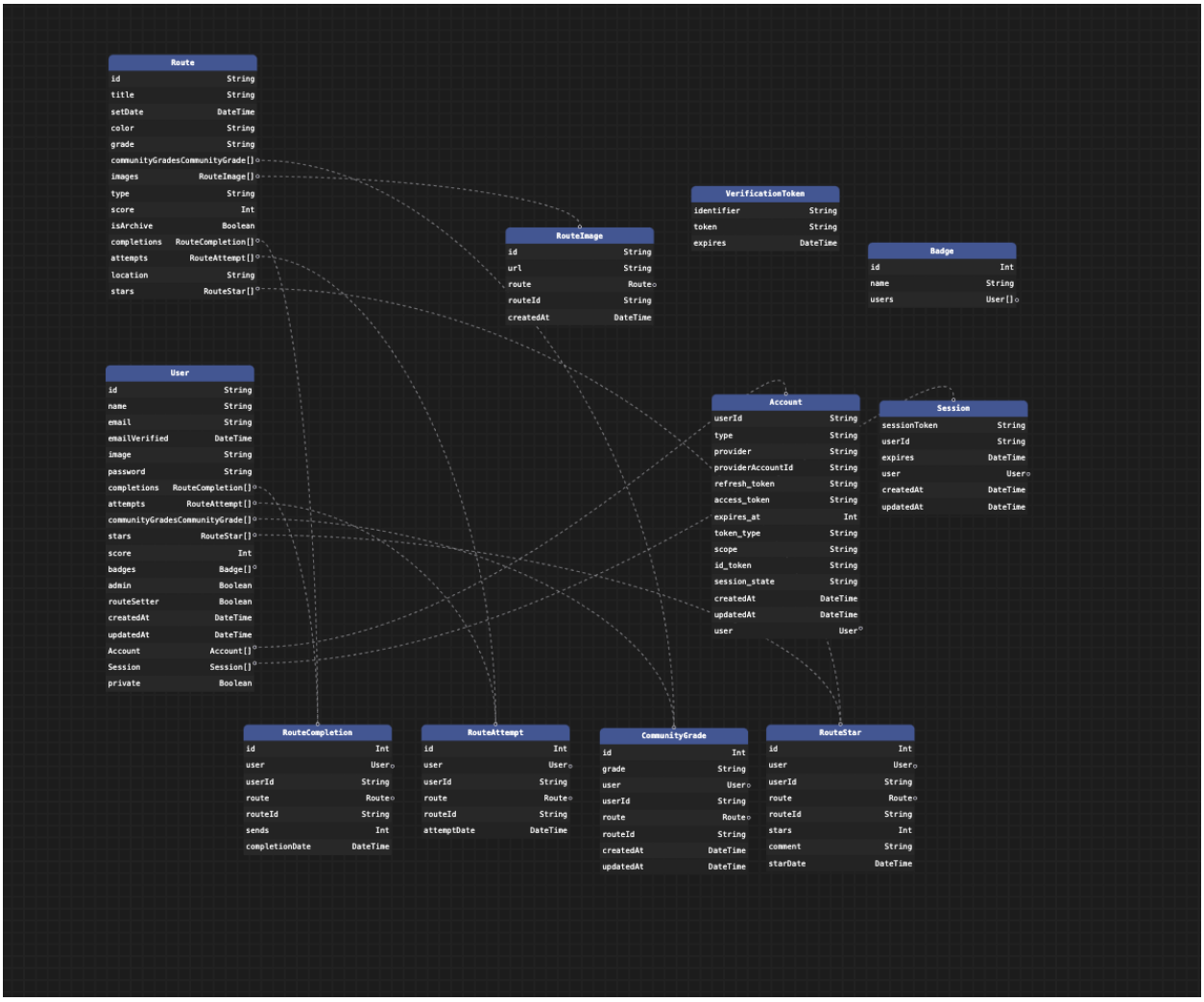
**Dynamic Models:**

**Login Flow:** When a user clicks on a login method, they will be redirected the authentication service they selected and then from their they are redirected back to the dashboard app. If they are brand new, their information will be stored in the database, so their data is stored.

**Route Creation:** When a user is logged in and an admin, they can navigate to the edit page to start creating a new route. They click on the plus sign which directs them to the new page where they can start filling out route information for submission. Once they fill out the necessary information, they can submit, and the routes will be updated on the route page.

**Route Tracking and Grading:** Once the user is logged in, either as a user or admin, they can go to any individual route page and log their grade, completion, or rating. Once they fill out the information and submit everything is updated.

### 5.3. Database design




### 5.4. Rationale for your detailed design models

The main rationale for using these design models is scalability and modularity. Having a clean and user-friendly UI is important to keep users informed and on the site. Being able to scale is very important since more features will be introduced as time goes on.

## 6. Test Management

### 6.1. Test Cases

| ID | **1** |
|---|---|
| Test Input | **User creates a route or completes a route** |
| Expected Output | **Data is fed to the database and uploaded to the server** |
| Description | **When a user inputs data to the server this is correctly handled by the database, server, and updates on the client efficiently** |

| ID | **2** |
|---|---|
| Test Input | **User clicks individual elements on the page** |
| Expected Output | **The Page changes or updates the page in some manner** |
| Description | **Making sure when the user clicks on the page it updates the page in some way and does not get hung up on anything.** |

| ID | **3** |
|---|---|
| Test Input | **User clicks on an element and navigates to another page** |
| Expected Output | **The page is correctly navigated to and ensures the user that they are the correct person to be on the page(admin or regular user)** |
| Description | **When a user navigates to a certain page the page is displayed correct and makes sure the user is allowed to be there.** |

| ID | **4** |
|---|---|
| Test Input | **a user logs in or signs up** |
| Expected Output | **,the data is correctly filled in the database and is updated on the server and client** |
| Description | **When a user logs in or signs up this data is passed to the database and the page reloads and successfully loads the correct data to the user.** |

| ID | **5** |
|---|---|
| Test Input | **Deployment of the application** |
| Expected Output | **The website will be up on the designated domain and is working with the environment** |
| Description | **When the website is deployed, making sure that all Api keys are correct for production and everything is working inside the production app.** |

### 6.2. Techniques used for test case generation

The main technique used was alpha testing and review. After implementation, testing the feature inside the app was the main method. So, after implementation of a individual feature, Test it for edge cases and find individual bugs. Address these bugs and develop to fix them. After doing these tests, friends would come and would do a stress test of all the features to see if there's any stones left unturned.

### 6.4. Test results and assessments

Because of the scale of the application, many bugs were found, but they were addressed. Some minor bugs are still within the app, but they do not affect functionality of the app and do not render the app useless.

## 7. Conclusions

### 7.1. Outcomes of the project

The main outcome of this project is learning web development and all the intricacies that come with it. Coming from a JavaScript background, learning React and Nextjs came easy. Learning server functions and API requests was the biggest learning part. Learning how real full stack web apps work is fascinating and makes learning them fun and interactive. The web app achieved all the basic requirements that were set for it and accomplishment goals with room left over to improve performance.

### 7.3. Future development

Future development is already happening. Since this will be used in the actual gym, adding features is paramount for keeping users active and happy. Taking user feedback is important as well, so interacting with them to get end user experience will help development. Features such as events, image handling, and badges are in the pipeline as we speak.