Fitness App Project Report
Databases
Samuel Cobb, Isaac Gasparri, and Aidan Schooley
Github Repository Link: https://github.com/aidanschooley/FitnessApp

1. Project Functionality and Workflow

We have designed a fitness app to help users better themselves by being active. Once you have signed up and logged in, our app provides a place for users to upload workouts, receive a personalized AI summary of their activities, and then be able to go back and view them on the dashboard. Users can also create fitness goals, toward a specific distance or duration, and set deadlines for those goals to be completed. On the dashboard, you can also view and add other data points, like water intake and calories consumed.

We began work by designing the database, using a MySQL server. From there, began overall app design began, with the frontend through React and the backend using Express. Once the core was working, we began designing specialized features. This included the ability to communicate with an AI model to generate activity summaries, a login system for users, and, most importantly, the ability to create and upload activities to the database. After that was just adding more "fun" features. This included adding goals, tracking records in the database, designing a homepage to display activities, and letting the AI model look at goals while giving advice in summaries.

A user logging into the platform for the first time would be met with our login screen, where they can also create an account. Once logged in, the user is taken to the dashboard, with areas ready for data to appear, including a workout feed and health stats. By clicking the menu, the user can select the workout page, where they can input data from their most recent activity. Currently, our functionality supports running, biking, and swimming, but can easily be expanded. Then, the user can decide if they want to see an AI summary of their activity, and then create it! A pop-up screen shows the user their summary, if it was requested, and they can confirm all of their inputted data was correct. Then, submitting it will save it to the database, and the user will be redirected to the dashboard, where they can see the newly added workout in their feed! Users can then navigate to the goals page and create a new goal, based on sport and either time or distance. They can also assign a deadline date to complete it by! Once a goal is complete, they can mark it as such as well.

2. System Design and Architecture

We built the architecture of the app in a three-layer system: Frontend, Backend, and the Database, with each structure having a clear responsibility and communicating with each layer. The frontend collects data and connects to the backend through HTTP requests (API). The backend then validates the data and sends an SQL request to the database, which is done through MYSQL. The database stores persistent user activity and data and enforces relationships through foreign keys. Specifically:

Frontend:

Handles all user input data (Login, registration, activity uploads, health uploads, etc.) and sends requests through API to the backend in a JSON structure. It does this through 4 HTTP requests:

POST- Creates data
GET- Gets the data
PUT- Edits the data
DELETE- Deletes the data

Example:

POST /api/auth/login
GET /api/activities/filter?userid=10
POST /api/activities/upload
DELETE /api/activities/10

The frontend is built using a combination of React and Node.js.

Backend:

The backend has a very separate delegation process that handles the information. The routes are handled using Express, and the baselines are defined in the index.js file (/api/health, or /api/auth). The routes are then further defined in the routes folder, which has more specific routes, such as /api/health/upload, which is then forwarded to the controller. The controller then validates the inputs, dynamically builds the SQL query, queries the database, and sends a JSON response to the frontend, if required.

Database:

The database is built focused on the user in focus, and everything connects to the user through foreign keys. This ensures that the relationship between the different tables can be referenced properly for the separate users. All passwords are stored using bcrypt and can never be accessed by us or anyone else. When a user is deleted, there is an ON DELETE CASCADE function that deletes all data stored in the database that is connected to the user id.

3. API Documentation

I. /api/users/:id
   A. Get
      1. Input: userid (parameter)
      2. Output: idUsers, Username, email, datecreated
      3. Usage: user info data, finds if that is not a user
   B. Put
      1. Input: userId (parameter), body: new {username, email}
      2. Output: updates database- “User updated successfully.”
      3. Usage: updates user information

   C. Delete
      1. Input: userId (parameter)
      2. Output: delete user in database- “User deleted successfully.”
      3. Usage: deletes user

II. /api/auth/register
   A. Post
   B. Input: username, email, password
   C. Output: adds user to database- “User registered successfully.”
   D. Usage: creates user, hashes password

III. /api/auth/login
   A. Post
   B. Input: username, email, password
   C. Output: message: “Login successful”, user: {idUsers, username, email}
   D. Usage: Signs in the user by checking fields (password with bcrypt)

IV. /api/chatbot
   A. Post
   B. Input: activityType, userData
   C. Output: aiResponse
   D. Usage: uses AI to summarize a workout.

V. /api/activities/upload
   A. Post
   B. Input: userid*, distance*, duration*, activityType*, ispublic, notes, pace, elevationGained, cadence, intensity, picture, rpm, stroke
      *required
   C. Output: inputs to activity table, checks if records/goal completed, if record/goal, inserts to record/goal table, “Activity uploaded successfully.”
   D. Usage:

VI. /api/activities/summary

   A. Get
   B. Input: userid (parameter)
   C. Output: activity data from the userid
   D. Usage: to display in the feed

VII. /api/activities/filter
   A. Get
   B. Input: userid, activitytype, data
   C. Output: activities that fulfill the filter (activityType, dateCreated)
   D. Usage: helps with sorting the feed

VIII. /api/activities/:id
   A. Get
   B. Input: idActivity (parameter)
   C. Output: activity data
   D. Usage: selecting a certain activity

IX. /api/activities/:id
   A. Put
   B. Input: idActivity (parameter), distance, duration, notes, ispublic
   C. Output: update database, “Activity updated successfully.”
   D. Usage: update activity

X. /api/activities:id
   A. Delete
   B. Input: idActivity (parameter)
   C. Output: deletes activity from database, “Actively deleted successfully.”
   D. Usage: delete activity

XI. /api/goals/newGoal
   A. Get
   B. Input: userId*, distance, duration, pace, stroke, targetDate, activityType
   C. Output: Goal inserted into goals table, “Goal created successfully.”
   D. Usage: created Goal

XII. /api/goals/:userId
   A. Get
   B. Input: userId (parameter)
   C. Output: Rows for all goals of the user
   D. Usage: displays goals

XIII. /api/goals/:goalId
   A. Put
   B. Input: goalId (parameter), distance, duration, pace, dateCompleted, deadlineDate, stroke, activityType
   C. Output: Updates goal from the goals table, “Goal updated successfully.”
   D. Usage: updates goal

XIV. /api/goals/:goalId
   A. Delete
   B. Input: goalId (Parameter)
   C. Output: Delete from goals table, “Goal deleted successfully.”
   D. Usage: deletes goal

XV. /api/records/makeRecord
   A. Post
   B. Input: distance, duration, pace, stroke, Users_idUsers, activityType
   C. Output: Record inserted into records table, “Record created successfully.”
   D. Usage: records a record

XVI. /api/records/:userId
   A. Get
   B. Input: userId (parameter)
   C. Output: rows of records for the user
   D. Usage: displays all records

XVII. /api/health/set
   A. Post
   B. Input: weight, height, BMI, age, userId
   C. Output: Insert into the currentHealth table, “Current health data set successfully
   D. Usage: set all health data

XVIII. /api/health/update
   A. Put
   B. Input: userId, options: weight, height, age
   C. Output: update currenthealth table
   D. Usage: update current health

XIX. /api/health/goals/set
   A. Put
   B. Input: weightGoal, calorieGoal, waterGoal, userId
   C. Output: updates weight goals - Health goals set successfully
   D. Usage: Sets health Goals

XX. /api/health/goals/:userId
   A. Get
   B. Input: userId (parameter)
   C. Output: rows of Health Goals
   D. Usage: display all health goals

XXI. /api/health/calories/update
   A. Post
   B. Input: amount, time, meal, userId - Calorie intake updated successfully."
   C. Output: Insert calories into the calories table

    D. Usage: calories update

XXII. /api/health/calories/:userId
    A. Get
    B. Input: userId (parameter)
    C. Output: rows of Calories
    D. Usage: display calories for the user

XXIII. /api/health/hydration/update
    A. Post
    B. Input: amount, time, userId - Hydration data updated successfully
    C. Output: Insert water into the water table
    D. Usage: calories update

XXIV. /api/health/hydration/:userId
    A. Get
    B. Input: userId (parameter)
    C. Output: rows of Water
    D. Usage: display water for the user

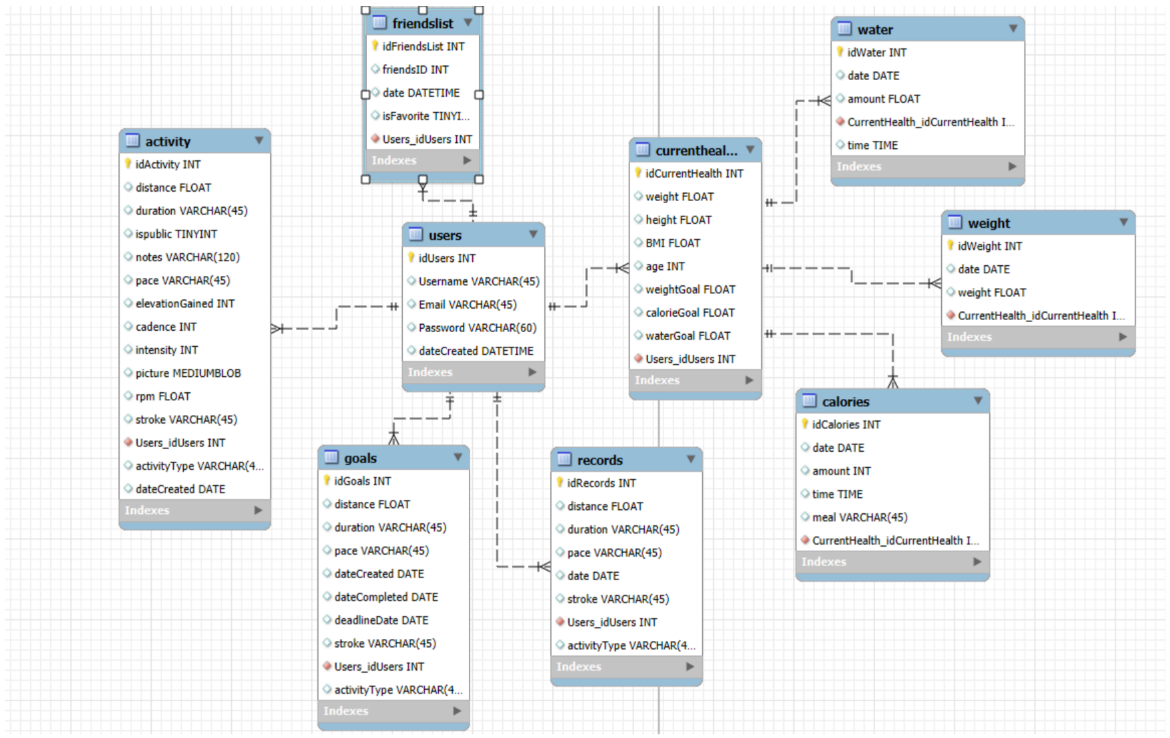
XXV. /api/health/weight/update
    A. Post
    B. Input: weight, userId - Weight data updated successfully
    C. Output: Insert the weight into the weight table
    D. Usage: weight update

XXVI. /api/health/weight/:userId
    A. Get
    B. Input: userId (parameter)
    C. Output: rows of Weight
    D. Usage: display calories for weight

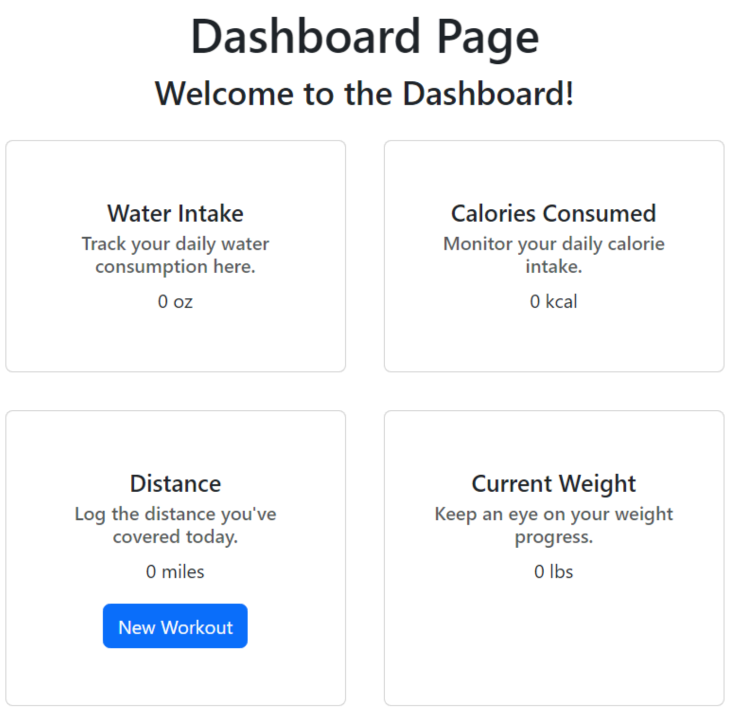
XXVII. /api/health/:userId
    A. Get
    B. Input: userId (parameter)
    C. Output: rows of health
    D. Usage: display health for user

4. Visual Demonstrations

a. Database Schema



b. Dashboard



Dashboard Page

Welcome to the Dashboard!

Water Intake
Track your daily water consumption here.
0 oz

Calories Consumed
Monitor your daily calorie intake.
0 kcal

Distance
Log the distance you've covered today.
0 miles
New Workout

Current Weight
Keep an eye on your weight progress.
0 lbs

Feed

Running

2000 miles 00:00:30

00:00:10 min/mi Cadence: 0 spm

Elevation: 0 ft Intensity: N/A

Today

Running

2 miles 00:00:02

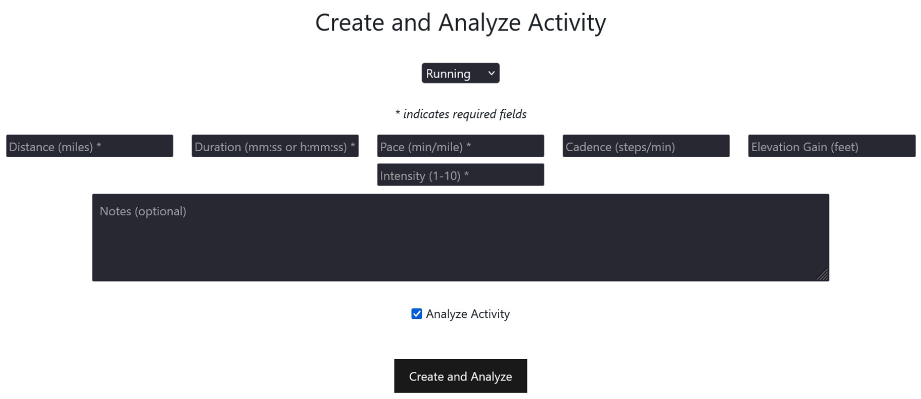
00:00:02 min/mi Cadence: 10 spm

Elevation: 0 ft Intensity: 10

Today

c. Workout Creation Screens

i. Running



Create and Analyze Activity

Running

** indicates required fields*

Distance (miles) * | Duration (mm:ss or h:mm:ss) * | Pace (min/mile) * | Cadence (steps/min) | Elevation Gain (feet)

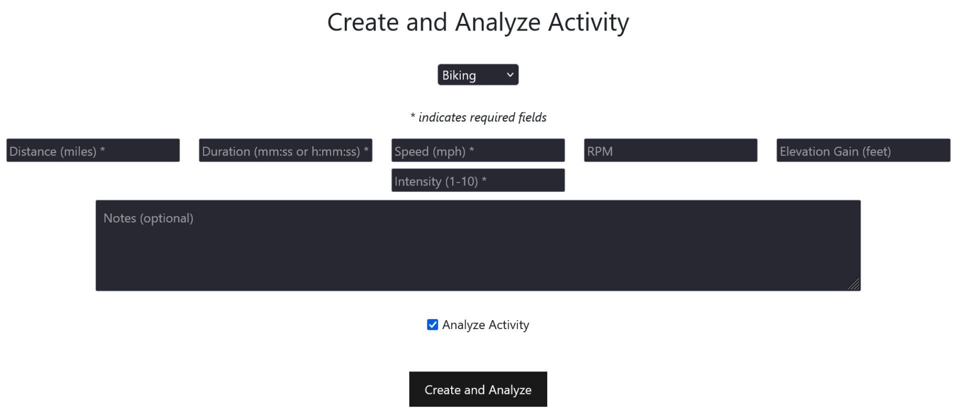
Intensity (1-10) *

Notes (optional)

✓ Analyze Activity

Create and Analyze

ii. Biking



Create and Analyze Activity

Biking

** indicates required fields*

Distance (miles) * | Duration (mm:ss or h:mm:ss) * | Speed (mph) * | RPM | Elevation Gain (feet)

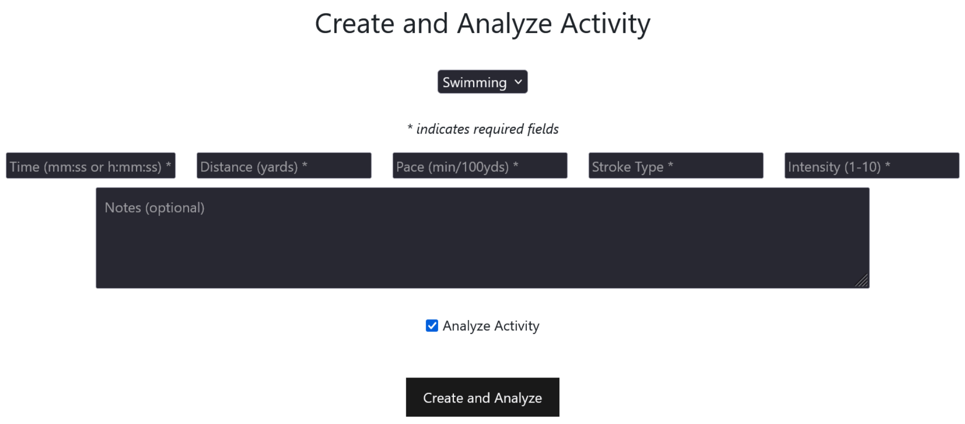
Intensity (1-10) *

Notes (optional)

✓ Analyze Activity

Create and Analyze

iii. Swimming



Create and Analyze Activity

Swimming

** indicates required fields*

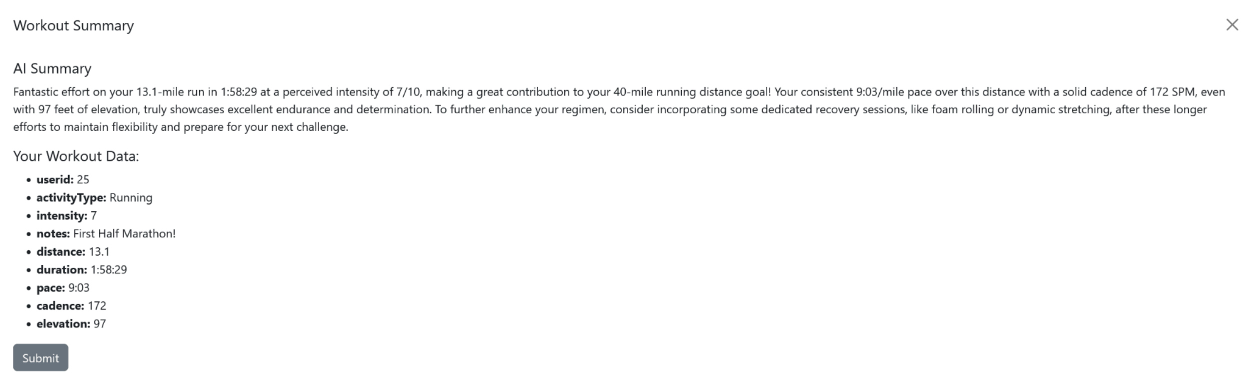
Time (mm:ss or h:mm:ss) * | Distance (yards) * | Pace (min/100yds) * | Stroke Type * | Intensity (1-10) *

Notes (optional)

✓ Analyze Activity

Create and Analyze

d. Workout Details Confirmation Screen



Workout Summary

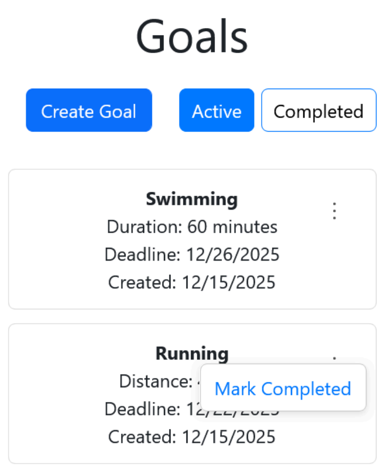
AI Summary

Fantastic effort on your 13.1-mile run in 1:58:29 at a perceived intensity of 7/10, making a great contribution to your 40-mile running distance goal! Your consistent 9:03/mile pace over this distance with a solid cadence of 172 SPM, even with 97 feet of elevation, truly showcases excellent endurance and determination. To further enhance your regimen, consider incorporating some dedicated recovery sessions, like foam rolling or dynamic stretching, after these longer efforts to maintain flexibility and prepare for your next challenge.

Your Workout Data:

- **userid:** 25
- **activityType:** Running
- **intensity:** 7
- **notes:** First Half Marathon!
- **distance:** 13.1
- **duration:** 1:58:29
- **pace:** 9:03
- **cadence:** 172
- **elevation:** 97

e. Goals Page

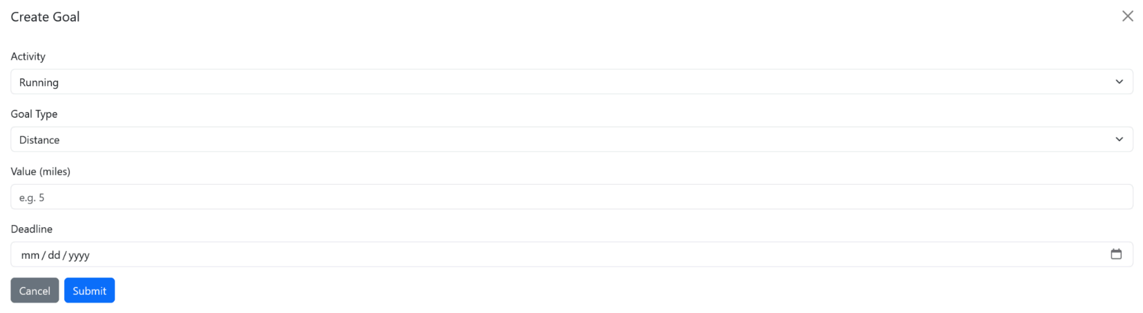


Goals

Swimming
Duration: 60 minutes
Deadline: 12/26/2025
Created: 12/15/2025

Running
Distance:
Deadline: 12/
Created: 12/15/2025

f. Creating a Goal Pop-Up Screen



Create Goal

Activity: Running

Goal Type: Distance

Value (miles): e.g. 5

Deadline: mm / dd / yyyy

5. Installation Guide

Dependencies:

bootstrap	^5.38
date-fns	^4.1.0
react	^19.2.0
react-dom	^19.2.0
react-icons	^5.5.0
react-router-dom	^7.10.1
bcryptjs	^3.0.3
cors	^2.8.5
datetime	^0.0.3
dotenv	^17.2.3
express	^5.2.1
jsonwebtoken	^9.0.3
mysql2	^3.15.3
openai	^4.35.0

Set Up Google Gemini functionality:

Navigate to https://aistudio.google.com/app/api-keys and create an API key to use for this app. Then, in your .env file, include the following line:

```
GEMINI_API_KEY='Your-API-Key-Here'
```

Set up MySQL Database:

First, you need to install MySQL Workbench and connect to the server containing the database with a unique login. Based on your clearance, you will have certain permissions. The login information must also be saved to the .env file to allow connection from the website. Enter the following lines:

```
DB_HOST=host_IP_address
DB_PORT=port_number
DB_USER=your_username
```

```
DB_PASSWORD=your_password
DB_NAME=FitnessApp
```

Node setup:

To install Node.js, you'll have to follow this link to the Node website. After the installer is done.

Server Startup:

There are two separate servers, one for the frontend and one for the backend. To start the backend server, first navigate to the folder that the fitnessApp is present in through a command prompt, for example, if the path is:

```
C:\Users\[your user]\OneDrive\Documents\DataBases\fitnessapp\backend
```

Then you will need to do the following commands:

```
Cd Onedrive
Cd Documents
Cd Databases
Cd fitnessapp
Cd backend
```

After you have reached the backend from the command prompt, you can then run:

```
Npm start
```

This will start the backend server.

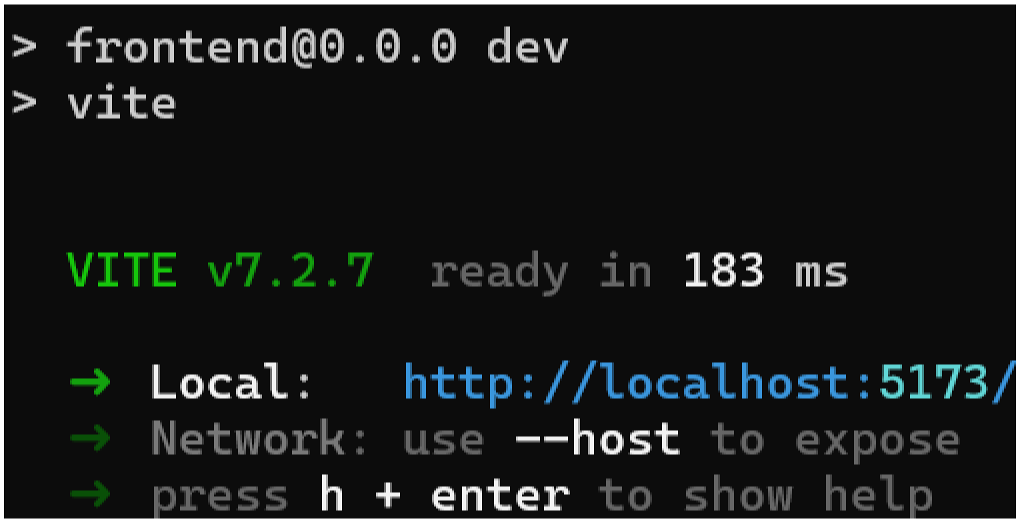
To start the frontend server, you will need to open up another command prompt, following the line of code for the backend, except that the last line will be replaced with

```
Cd frontend
```

After you have reached the frontend from the command prompt, you will then run

```
Npm start
```

This will start both the frontend and backend servers. Inside the command prompt for the frontend server will look like this:



```
> frontend@0.0.0 dev
> vite


  VITE v7.2.7  ready in 183 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

If you Ctrl+click the http:// link, it will then open up a webpage that will display the app.

Congratulations! You have successfully set up the FitnessApp.