Gym App documentation and Development log

# Introduction

I started this log to track notes on where I left off, what I have left to do, and how my app works. I will update this document as I go along building my app so I can document this progress and keep myself accountable. This might not be the most organized document, but I will start with taking notes on my app so with rising grad school work, I can come back and hit the ground running without a lot of re-learning my app. For this reason each progress week might not be linear/contiguous, but slow and steady wins the race.

# Notes on how this works, since its been a while

1. All frontend logic auth is handled via utils/AuthProvider and utils/AxiosInterceptorHandler
   1. AuthProvider wraps our entire app and returns loggedIn state value prop so our entire app can access it globally (via prop drilling technically)
   2. AxiosInterceptorHandler uses AuthProvider to set login status and handle if we’re authed or not. It handles all logic involving JWT tokens when calling something.
      1. instance uses request/response interceptors to attach tokens to every request made to backend, handle invalid cases and attempt to refresh, or sign out.
      2. authInstance handles edge case for our initial login logic so we can get/set our tokens and whatnot
2. General Backend structure and authentication
   1. Backend follows a controller/service/route architecture. The route calls the controller, which handles the input json data and makes sure certain fields that are required are present. The controller then calls the service which is in charge of actually making the database calls and the processing logic. This data is returned to the controller, which then returns the appropriate https status code according to the success

* 1. Routing starts at index.js and has routes for /Auth, /Workout, /Nutrition.
     1. Auth has a middleware called verify token which simply returns if token is valid or not. Our frontend handles what endpoint to call if it isn’t. Like usual, we have a /login endpoint and a /register endpoint that handle tokens accordingly. Most/All Endpoints for /Auth and /Workout will need a valid token
     2. We have endpoints for creating a workout for a user and creating a set for a workout (including reps). For each set we also allow a note to be provided.
     3. /Workout endpoints.

1. Expo Router Navigation starts in app/\_layout.tsx and renders login. If user is valid (checked in login page via above authProvider), we navigate to app/(protected)/\_layout. By default it renders the Home page.

# Progress Summary (Week of 8/3)

* **Backend**
  + Learned the **controller/service architecture**:
    - Controllers handle incoming requests/responses.
    - Services contain reusable business logic.
  + Implemented **JWT authentication** with access and refresh tokens.
  + Added functionality to create and manage custom users in the backend.
* **Frontend**
  + Started implementing the **JWT sign-in flow**.
  + Set up **Expo Router** for navigation.
  + Created an **AuthProvider (Auth Context)** to manage global authentication state.
  + Integrated **Expo SecureStore** for secure storage of access and refresh tokens.
  + Learned how to integrate Auth Context with token storage to allow automatic sign-in detection.
  + Explored **auto-refreshing tokens** when expired.
  + Began setting up **route protection** based on authentication state (isValidUser).
* **Other**
  + Learned how **splash screens** work and how they fit into the authentication check process (not yet implemented).

**Next Steps**

* Complete **JWT authentication logic** on the frontend.
* Implement **Axios interceptors** for:
  + Automatically attaching access tokens to requests.
  + Refreshing tokens when expired.

# Progress Summary (Week of 10/19)