CSCC01

System Design Document

Porom Kamal, Dominik Luszczynski, Ryan Ramroop, Kevin Xiong, Bryan Wan, Aviraj Waraich, Jason Kenneth Setiawan

October 7th, 2022

Table of Contents

CRC Cards

The CRC Cards have been broken into two categories (Backend, and Frontend) for backend and frontend classes respectively

Backend:

Class Name: /config/db.js	
Parent/Subclass: None	
Responsibilities: ● Read our MongoDB connection URI from our default.json file ● Establish a connection to our MongoDB database using URI	**Collaborators:** Npm provided packages/classes ● mongoose.js ● config.js

Class Name: /config/awsS3.js	
Parent/Subclass: None	
Responsibilities: ● Establish a connection to AWS. ● Creates buckets for profile pictures and social posts.	**Collaborators:** Npm provided packages/classes ● mongoose.js ● config.js

Class Name: /models/User.js	
Parent/Subclass: None	
Responsibilities: ● Defines a User model, with properties/fields such as name, email, and password, ie defines how a user object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/BF.js	
Parent/Subclass: None	
Responsibilities: ● Defines a BF (body fat percentage) model, with properties/fields such as BF, date, userID, ie defines how a BF object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/BMI.js	
Parent/Subclass: None	
Responsibilities: ● Defines a BMI model, with properties/fields such as BMI, date, userID, ie defines how a BMI object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/BodyWeight.js	
Parent/Subclass: None	
Responsibilities: ● Defines a BodyWeight model, with properties/fields such as weight, date, userID, ie defines how a BodyWeight object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/MealPlan.js
Parent/Subclass: None

Responsibilities: ● Defines a Meal plan model, with properties/fields such as name, calories, ie defines how a MealPlan object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/SocialPost.js	
Parent/Subclass: None	
Responsibilities: ● Defines a SocialPost model, with properties/fields such as userId, message, ie defines how a SocialPost object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/WorkoutPlan.js	
Parent/Subclass: None	
Responsibilities: ● Defines a workout plan model, with properties/fields such as planName, workouts, ie defines how a WorkoutPlan object will be formatted in our MongoDB database.	**Collaborators:** Npm provided packages/classes ● mongoose.js

Class Name: /models/Workouts.js	
Parent/Subclass: None	
Responsibilities: ● Defines a workouts model, with properties/fields such as muscle group, workout name, ie defines how a Workout object will be	**Collaborators:** Npm provided packages/classes ● mongoose.js

formatted in our MongoDB database.	

Class Name: index.js	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**
● Instantiate Express Nodejs server	● /config/db.js
● Adds routes/endpoints to the server	● /routes/authRoutes
● Starts the server locally on port 5000, or port defined by .env file.	npm provided packages/classes
	● Express.js
	● cors
	● dotenv

Class Name: /controllers/authControllers	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**
● Define functions which handle the logic behind user authentication server code, such as login, and register.	● /models/User.js
● These functions will be called through their respective endpoints/routes in the server.	npm provided packages/classes
	● jwt
	● bcrypt
	● express-validator
	● express-async-handler

Class Name: /controllers/nutrtionController	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**
● Define functions which handle the logic behind nutrition server code,	● /models/User.js
	● /models/MealPlan.js

such as retrieving meal plans, or updating calorie goals. ● These functions will be called through their respective endpoints/routes in the server.	npm provided packages/classes ● jwt ● bcrypt ● express-validator ● express-async-handler

Class Name: /controllers/userController	
Parent/Subclass: None	
Responsibilities: ● Define functions which handle the logic behind retrieving user data from MongoDB. ● These functions will be called through their respective endpoints/routes in the server.	**Collaborators:** ● /models/User.js npm provided packages/classes ● jwt ● bcrypt ● express-validator ● express-async-handler

Class Name: /controllers/devController	
Parent/Subclass: None	
Responsibilities: ● Define functions which handle the logic behind clearing data from the database. ● These functions will be called through their respective endpoints/routes in the server.	**Collaborators:** ● /models/User.js ● /models/MealPlan.js npm provided packages/classes ● jwt ● bcrypt ● express-validator ● express-async-handler

Class Name: /controllers/workoutController	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**

• Define functions which handle the logic behind retrieving and adding workout information to MongoDb. • These functions will be called through their respective endpoints/routes in the server.	• /models/Workouts.js • /models/WorkoutPlan.js npm provided packages/classes • jwt • bcrypt • express-validator • express-async-handler

Class Name: /controllers/progressController	
Parent/Subclass: None	
Responsibilities: • Define functions which handle the logic behind retrieving and adding user progress information (bmi, bf%) to MongoDb. • These functions will be called through their respective endpoints/routes in the server.	**Collaborators:** • /models/BF.js • /models/BMI.js • /models/BpdyWeight.js npm provided packages/classes • jwt • bcrypt • express-validator • express-async-handler

Class Name: /controllers/SocialController	
Parent/Subclass: None	
Responsibilities: • Define functions which handle the logic behind retrieving and adding social posts to MongoDb. • These functions will be called through their respective endpoints/routes in the server.	**Collaborators:** • /models/SocialPost.js npm provided packages/classes • jwt • bcrypt • express-validator • express-async-handler

Class Name: /routes/authRoutes

Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically the user authentication URLs (login, register). ● Assign functions from the /controllers/authController file to their respective routes; to be called when the endpoint is requested.	**Collaborators:** ● /controllers/authControllers npm provided packages/classes ● express

Class Name: /routes/nutritionRoutes	
Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically the nutrition data URLs (get meal plans, update calorie goals). ● Assign functions from the /controllers/nutritionController file to their respective routes; to be called when the endpoint is requested.	**Collaborators:** ● /controllers/nutritionController npm provided packages/classes ● express

Class Name: /routes/userRoutes	
Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically the user data URLs (get user data). ● Assign functions from the /controllers/userController file to their respective routes; to be called when the endpoint is requested.	**Collaborators:** ● /controllers/usernutritionController npm provided packages/classes ● express

Class Name: /routes/developmentRoutes	
Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically developer necessities (clearing meal plans, clearing users). ● Assign functions from the /controllers/nutritionController file to their respective routes; to be called when the endpoint is requested.	**Collaborators:** ● /controllers/devController npm provided packages/classes ● express

Class Name: /routes/workoutRoutes	
Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically developer necessities (getting workout plans, adding plans). ● Assign functions from the /controllers/workoutController file to their respective routes; to be called when the endpoint is requested.	**Collaborators:** ● /controllers/devController npm provided packages/classes ● express

Class Name: /routes/progressRoutes	
Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically developer necessities (getting bodyweight, bmi, body fat %). ● Assign functions from the /controllers/progressController file	**Collaborators:** ● /controllers/devController npm provided packages/classes ● express

to their respective routes; to be called when the endpoint is requested.	

Class Name: /routes/SocialRoutes	
Parent/Subclass: None	
Responsibilities: ● Define the URLs for the endpoints in the server, specifically developer necessities (getting posts, adding a social post). ● Assign functions from the /controllers/socialController file to their respective routes; to be called when the endpoint is requested.	**Collaborators:** ● /controllers/devController npm provided packages/classes ● express

Frontend

(Note: React, and react-native are implicit collaborators)

Class Name: App.js	
Parent/Subclass: None	
Responsibilities: ● Entry point into our front end ● Loads fonts into our project ● Directs responsibility to our stack navigator class, AppStack.	**Collaborators:** ● /navigator/AppStack npm provided packages/classes ● Expo-font ● react-native-root-siblings

Class Name: /navigator/AppStack.js	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**

• Maintains a ledger of different screens so a developer can easily write code to navigate from one screen to another • Creates a stack object which keeps track of screens that the user has visited.	• /screens/Login.js • /screens/Register.js • /screens/TempLanding.js • /screens/CalorieCalculator.js • /components/Navbar.js • /screens/Nutrition • /screens/NutritionAddFood • /screens/NutritionFoodViewer • /screens/NutritionPlans • /screens/NutritionMealAdder • /screens/NutritionMealPlanInfo • /screens/Workout • /screens/WorkoutAddPlan • /screens/WorkoutLogs • /screens/WorkoutExplore

Class Name: /screens/Register.js	
Parent/Subclass: None	
Responsibilities: • The UI for the register page, which contains an email field, and password field. • Entering valid login information will redirect you to the app landing page.	**Collaborators:** • /components/colors.js • /requests/userRequests.js • /components/ErrorMsg.js • /components/ShowPasswordBtn.js npm provided packages/classes • react-native-root-toast • axios

Class Name: /screens/CalorieCalculator.js	
Parent/Subclass: None	
Responsibilities: • UI for calorie calculator interface, includes a form which outputs the users recommended calorie goals, based on their measurements, and personal goals. • Saves the Calorie Goals to the database via a post request to our	**Collaborators:** • /components/colors

backend.	

Class Name: /components/Navbar.js	
Parent/Subclass:	
Responsibilities: ● Creates a navigation bar, which allows the user to navigate throughout the app.	**Collaborators:** ● /screens/Nutrition ● /screens/Social ● /screens/Profile ● /screens/Workout ● /screens/WorkoutLogs ● /screens/WorkoutExplore

Class Name: /screens/Nutrition.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the nutrition page. ● Displays the user's calorie goal, and displays a button to navigate to the calorie calculator, and add food feature.	**Collaborators:** ● None

Class Name: /screens/NutritionAddFood.js	
Parent/Subclass: None	
Responsibilities: ● The UI for food creation feature including: name, calories, carbs, proteins, fat, sodium, and sugar	**Collaborators:** ● None

Class Name: /screens/NutritionFoodViewer.js
Parent/Subclass: None

Responsibilities: ● The UI for the displaying the data within a food instance including: name, calories, carbs, proteins, fat, sodium, and sugar	**Collaborators:** ● None

Class Name: /screens/NutritionPlans.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the nutrition plans page. ● Displays the user's library of meal plans and a Button to create a new meal plan.	**Collaborators:** ● None

Class Name: /screens/NutritionMealAdder.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the Meal Plan creation page ● Displays text input boxes for the user to add items to a new meal plan and a button to save their changes.	**Collaborators:** ● None

Class Name: /screens/NutritionAddFood.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the log food page ● Displays text input boxes for the user to add items to a new food and a button to save their changes.	**Collaborators:** ● None

Class Name: /screens/NutritionMealPlanInfo.js	
Parent/Subclass: None	
Responsibilities: ● The UI for displaying the contents of a Meal Plan ● Displays data of a given meal plan including: Plan name, total calories, total protein	**Collaborators:** ● None

Class Name: /screens/NutritionFoodViewer.js	
Parent/Subclass: None	
Responsibilities: ● The UI for displaying the contents of a logged food ● Displays data of a given food including: Food name, total calories, total protein	**Collaborators:** ● None

Class Name: /screens/NutritionMealPlanEditor.js	
Parent/Subclass: None	
Responsibilities: ● The UI for allowing a user to edit and delete meal plans. ● Displays data of a given meal plan including: Plan name, total calories, total protein, as well as the option to edit said values.	**Collaborators:** ● None

Class Name: /screens/ReviewMealPlan.js	
Parent/Subclass: None	
Responsibilities: ● The UI for allowing see meal plans	**Collaborators:** ● None

in the explore page. ● Also has a section for the user to contribute to a meal plans rating.	

Class Name: /screens/Login.js	
Parent/Subclass: None	
Responsibilities: ● UI for the login page ● Includes fields for email, and password, which will navigate the user to the app landing page, if the credentials are correct.	**Collaborators:** ● None

Class Name:/screens/Profile.js	
Parent/Subclass: None	
Responsibilities: ● UI for the profile screen, including profile picture, bio, name. ● Access point to the user's profile settings, and a logout button. ● Has a button allowing users to go to a page to edit their profile.	**Collaborators:** ● None

Class Name:/screens/Profile_Edit.js	
Parent/Subclass: None	
Responsibilities: ● UI for the profile edit screen. ● Allows users to edit their profile, including name, bio, email, and profile picture.	**Collaborators:** ● None

Class Name: /screens/Social.js

Parent/Subclass: None	
Responsibilities: ● UI For the social page ● Access point for the social page features, which includes searching for people, scrolling through posts, and adding posts.	**Collaborators:** ● None

Class Name: /screens/Workout.js	
Parent/Subclass: None	
Responsibilities: ● Landing page for the workout tab ● Access point for workout plans saved to the user's profile and creating new plans	**Collaborators:** ● None

Class Name: /screens/WorkoutAddPlan.js	
Parent/Subclass: None	
Responsibilities: ● The UI for adding a workout plan. ● Displays text input boxes for the user to add items to a workout and a button to save their changes.	**Collaborators:** ● None

Class Name: /screens/WorkoutLogs.js	
Parent/Subclass: None	
Responsibilities: ● Access point for the logging workout feature	**Collaborators:** ● None

Class Name: /screens/WorkoutExplore.js

Parent/Subclass: None	
Responsibilities: ● Access point for the workout features, such as viewing another user's workout playlists and logging playlists	**Collaborators:** ● None

Class Name: /screens/SocialHome.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the home page on Social, allowing users access to view all posts created in the application.	**Collaborators:** ● None

Class Name: /screens/SocialExplore.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the explore page on Social. ● Allows users to search for any content creator, allowing them to view their profile and follow/unfollow them.	**Collaborators:** ● None

Class Name: /screens/SocialCreate.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the create page on Social. ● Allows users to create a post (image or text).	**Collaborators:** ● None

Class Name: /screens/ViewProfile.js	
Parent/Subclass: None	
Responsibilities: ● The UI for users to view other peoples profiles. ● Allows users to follow said user.	**Collaborators:** ● None

Class Name: /screens/Progress.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the progress page. ● Allows users to traverse from Body Weight, BMI and BF % tracking.	**Collaborators:** ● None

Class Name: /screens/ProgressAddBodyWeight.js	
Parent/Subclass: None	
Responsibilities: ● The UI for adding a body weight page. ● Allows users to input their body weight and save it to the tracking system.	**Collaborators:** ● None

Class Name: /screens/ProgressAddBodyWeight.js	
Parent/Subclass: None	
Responsibilities: ● The UI for adding a body weight page. ● Allows users to input their body weight and save it to the tracking	**Collaborators:** ● None

system.	

Class Name: /screens/ProgressBF.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the Body fat % section in Progress. ● Allows users to view and track their body fat %. ● Includes a button sending users to a calculator for their BF %.	**Collaborators:** ● None

Class Name: /screens/ProgressBMI.js	
Parent/Subclass: None	
Responsibilities: ● The UI for the BMI section in Progress. ● Allows users to view and track their BMI. ● Includes a button sending users to a calculator for their BMI.	**Collaborators:** ● None

Class Name: /screens/BFCalculator.js	
Parent/Subclass: None	
Responsibilities: ● The UI for calculating BF %. ● Allows users to input information about their body to calculate their BF %. ● Button allowing users to save their calculated percentage.	**Collaborators:** ● None

Class Name: /screens/BMICalculator.js	
Parent/Subclass: None	
Responsibilities: ● The UI for calculating BMI. ● Allows users to input information about their body to calculate their BMI. ● Button allowing users to save their calculated BMI.	**Collaborators:** ● None

Class Name: /components/colors.js	
Parent/Subclass: None	
Responsibilities: ● Contains app’s colour values as constants.	**Collaborators:** ● None

Class Name: /components/ErrorMsg.js	
Parent/Subclass: None	
Responsibilities: ● Component which renders an error message text field.	**Collaborators:** ● None

Class Name: /components/ShowPasswordBtn.js	
Parent/Subclass: None	
Responsibilities: ● Render a clickable button with “show” text.	**Collaborators:** ● None

Class Name: /components/BFItem.js	
Parent/Subclass: None	
Responsibilities: ● Component which renders a Body Fat % item.	**Collaborators:** ● None

Class Name: /components/BMIItem.js	
Parent/Subclass: None	
Responsibilities: ● Component which renders a BMI item.	**Collaborators:** ● None

Class Name: /components/BodyWeightItem.js	
Parent/Subclass: None	
Responsibilities: ● Component which renders a Body Weight item.	**Collaborators:** ● None

Class Name: /components/ButtonSwitch.js	
Parent/Subclass: None	
Responsibilities: ● Component which renders a button which switches text after click.	**Collaborators:** ● None

Class Name: /components/MealPlanItem.js	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**

• Component which renders a Meal Plan item.	• None

Class Name: /components/NavBar.js	
Parent/Subclass: None	
Responsibilities: • Component which renders the NavBar allowing users to switch through pages easily and effectively.	**Collaborators:** • None

Class Name: /components/NutritionNav.js	
Parent/Subclass: None	
Responsibilities: • Component which renders the NavBar in the Nutrition section allowing users to switch through pages easily and effectively.	**Collaborators:** • None

Class Name: /components/socialcard.js	
Parent/Subclass: None	
Responsibilities: • Component which renders a card for a social post.	**Collaborators:** • None

Class Name: /components/WorkoutItem.js	
Parent/Subclass: None	
Responsibilities:	**Collaborators:**

• Component which renders a Workout item.	• None

Class Name: /components/WorkoutPlanCard.js	
Parent/Subclass: None	
Responsibilities: • Component which renders a Workout Plan item.	**Collaborators:** • None

Class Name: /components/WorkoutSearchItem.js	
Parent/Subclass: None	
Responsibilities: • Component which renders a Workout item when searching.	**Collaborators:** • None

Class Name: /requests/userRequests.js	
Parent/Subclass: None	
Responsibilities: • Contain functions which have code to send user requests to our backend.	**Collaborators:** • None

Class Name: /requests/SocialRequest.js	
Parent/Subclass: None	
Responsibilities: • Contain functions which have code to send social requests to our backend.	**Collaborators:** • None

System Interaction Description

GainzHub assumes that the user has the latest Node.js, and npm installed. GainzHub also assumes the user is using Windows 10+, macOS, or Linux.

To run our backend, the user must first **cd into the backend folder (GainzHub/backend)** and run: **npm install**, to install all the necessary dependencies. It is also recommended to include a file in the root backend folder called **.env**, and input the text: **JWT_SECRET** = **"SECRET",** within the file. Next, the user can start the server using the command: **npm start**.

To run our front end, the user must **cd into the frontend folder (GainzHub/frontend)**, and run: **npm install --force**. They must then run: **npm i -g expo-cli.** To finally run the front end they must run the command: **npm start.** Upon starting, the expo-CLI will ask the user to select a device to run the code on. It is recommended that the user run the app on the web browser, by pressing: **w**, in the terminal. After the web browser opens, it is highly recommended that they toggle the device toolbar, by clicking **F12,** or opening **Inspect Element**, and then toggling the phone icon, in the top left of the menu ( ). They can also use an AVD(Recommended), IOS emulator, or physical device. For AVD, they must launch their AVD, and then click: **a**, in the terminal.

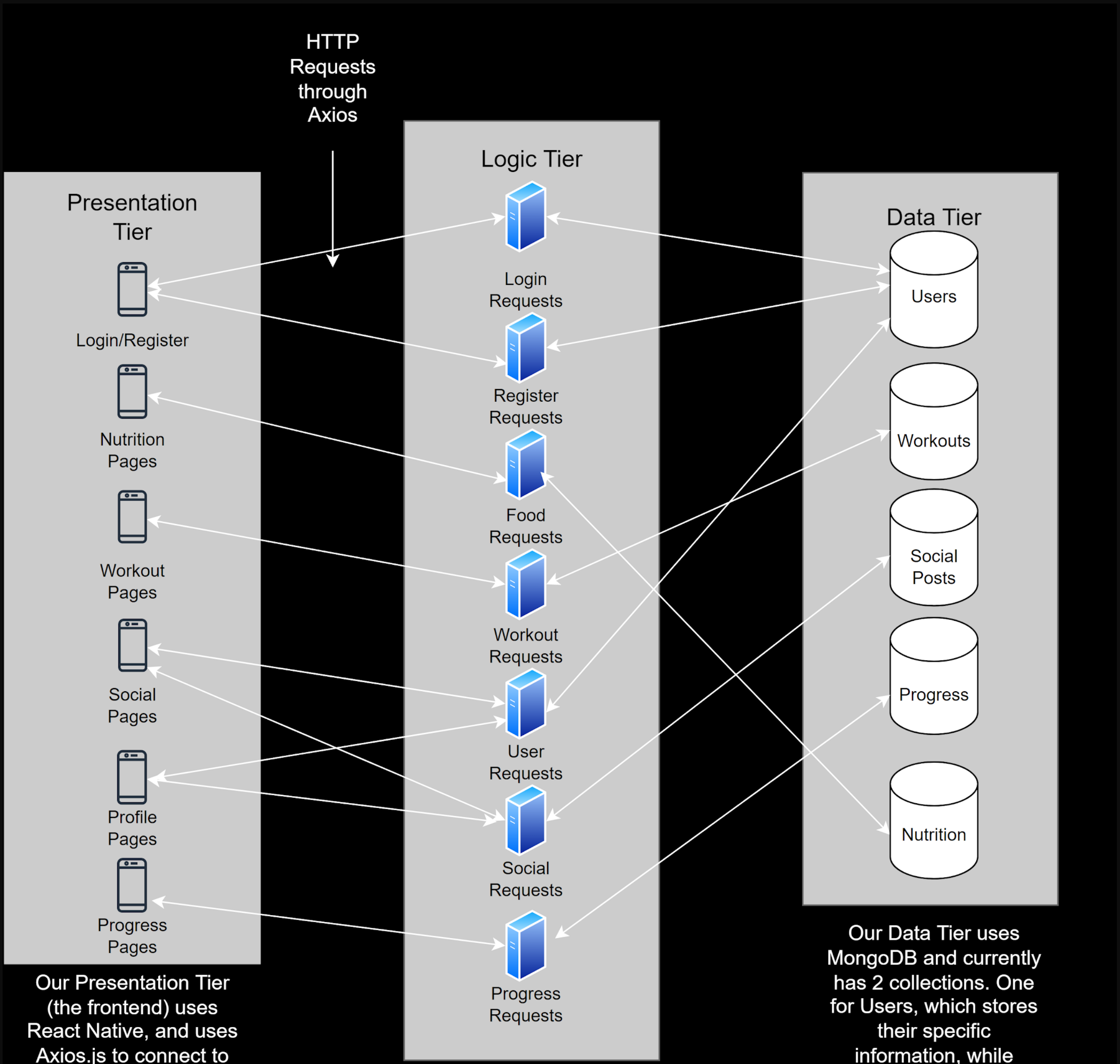
System Architecture

The system architecture we chose for this project is a three-tiered architecture. In terms of the tech stack, we used react-native for frontend, which communicates to our backend server using HTTP requests (using Axios.js). Our backend uses nodejs, and expressjs to start the server on port 5000 or the port defined in the .env file. The server connects to our MongoDB database using mongoosejs. This follows the three-tiered architecture, as our presentation tier is our frontend (React-native), our logic tier is our backend (Node, express), and our data tier is MongoDB (See Diagram on next page).

Consider the following link describing the Three-Tiered Architecture:

IBM: https://www.ibm.com/cloud/learn/three-tier-architecture

System Architecture Diagram



Our Presentation Tier (the frontend) uses React Native, and uses Axios.js to connect to the server and make specific requests through HTTP requests

Our Backend uses Express and Node.js to connect the database and the frontend. Our backend handles different HTTP requests from Axios and performs the request with the information from the database.

Our Data Tier uses MongoDB and currently has 2 collections. One for Users, which stores their specific information, while workouts contains a set of exercise that could be used universally.

System Decomposition

The user interacts with and sees the frontend (presentation tier) which is coded in react-native. At its core the frontend is responsible for responding to user input, when going through the various screens in the app, as well as input data to send requests to the server, such as setting a calorie goal or adding a workout playlist. The front end sends HTTP requests to our backend (logic tier) via Axios. Our logic tier handles the logic for sending data to our database, accessing data from our database, and returning it to the front end. To give an example of the interaction between the tiers, and how errors are handled. Let's take the login feature, in the presentation tier, the user will input their login credentials, validation of user input fields are done client side first, so if a field is not inputted, or if the email is not formatted correctly, a toast will be displayed to the user detailing which fields need to be filled in, as well as red highlighting on the respective fields. After the user inputs the form in a correct format the data is sent to our logic tier, via Axios. The endpoint to handle login in our backend will first run middleware to validate the user input again, if the user input is not in the correct format, then the logic tier, returns an error 403 (incorrect format) to our front end, which will then be displayed as a toast to our user. If the middleware passes, the respective endpoint controller function is ran, to validate the user credentials, by comparing the password hash values with the MongoDB database (data tier), using mongoosejs to send queries, if the credentials are invalid, an error 400 is returned to the front, else a status 200 is returned to the frontend, along with a JWT token which is stored in the machines local storage. Any network or system errors in our server (logic tier) or MongoDB database (data tier) will be manifested to the user in the frontend as a toast, describing the current issue.