

CSCC01

System Design Document

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

October 7th, 2023

Table of Contents

CRC Cards.....	2
System Interaction Description.....	9
System Architecture.....	10
System Decomposition.....	11

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: <ul style="list-style-type: none">• Read our MongoDB connection URI from our default.json file• Establish a connection to our MongoDB database using URI	Collaborators: Npm provided packages/classes <ul style="list-style-type: none">• mongoose.js• config.js

Class Name: /models/User.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none">• 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 <ul style="list-style-type: none">• mongoose.js

Class Name: index.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • Instantiate Express Nodejs server • Adds routes/endpoints to the server • Starts the server locally on port 5000, or port defined by .env file. 	Collaborators: <ul style="list-style-type: none"> • /config/db.js • /routes/authRoutes npm provided packages/classes <ul style="list-style-type: none"> • Express.js • cors • dotenv

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

Class Name: /routes/authRoutes	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • /controllers/authControllers npm provided packages/classes <ul style="list-style-type: none"> • express

Frontend

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

Class Name: App.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • Entry point into our front end • Loads fonts into our project • Directs responsibility to our stack navigator class, AppStack. 	Collaborators: <ul style="list-style-type: none"> • /navigator/AppStack npm provided packages/classes <ul style="list-style-type: none"> • Expo-font • react-native-root-siblings

Class Name: /navigator/AppStack.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • 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. 	Collaborators: <ul style="list-style-type: none"> • /screens/Login.js • /screens/Register.js • /screens/TempLanding.js • /screens/CalorieCalculator.js • /components/Navbar.js • /screens/Nutrition

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

Class Name: /screens/TempLanding.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • Temporary landing page for testing purposes • Development only, not to be routed to in final production code. 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /screens/CalorieCalculator.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • 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 backend. 	Collaborators: <ul style="list-style-type: none"> • /components/colors

Class Name: /components/Navbar.js	
Parent/Subclass:	
Responsibilities: <ul style="list-style-type: none"> • Creates a navigation bar, which allows the user to navigate throughout the app. 	Collaborators: <ul style="list-style-type: none"> • /screens/Nutrition • /screens/Social • /screens/Profile • /screens/Workout

Class Name: /screens/Nutrition.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • None

Class Name: /screens/Login.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • UI for the login page • Includes field for email, and password, which will navigate the user to the app landing page, if the credentials are correct. 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /screens/Profile.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • UI for the profile screen. • Access point to the user's profile settings, and a logout button. 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /screens/Social.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • UI For the social page • Access point for the social page features, which includes searching for people, scrolling through posts, and adding posts. 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /screens/Workout.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • UI for the workout page • Access point for the workout features, such as viewing workout playlists, adding playlists, and logging playlists 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /components/colors.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • Contains app's colour values as constants. 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /components/ErrorMsg.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • Component which renders an error message text field. 	Collaborators: <ul style="list-style-type: none"> • None


Class Name: /components/ShowPasswordBtn.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> • Render a clickable button with "show" text. 	Collaborators: <ul style="list-style-type: none"> • None

Class Name: /requests/userRequests.js	
Parent/Subclass: None	
Responsibilities: <ul style="list-style-type: none"> Contain functions which have code to send user requests to our backend. 	Collaborators: <ul style="list-style-type: none"> None

System Interaction Description

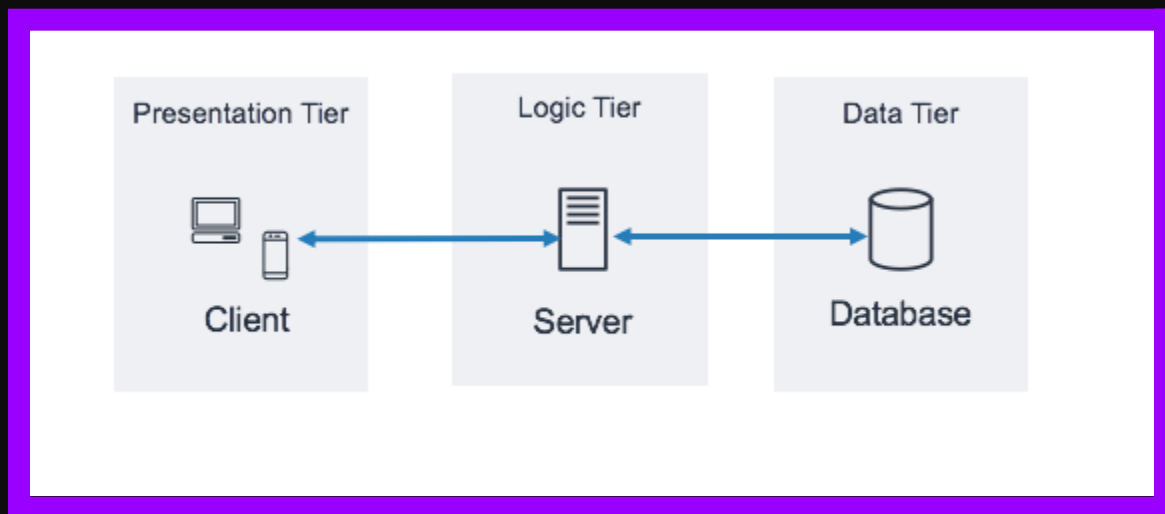
GainzHub assumes that the user has the latest Node.js, and npm installed. GainzHub also assumes the user is using Windows 10+, macOS, or Linux.

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 website, by pressing: **w**, in the terminal. After the website 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 mongoosjs. 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.



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. Lets 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.