**HOME WORKOUT**

**AND**

**Analytics Suite**

**FINAL Report**

# *Student Name: Smart Egbuchulem*

# *Student Number: 300333966*

# *Course: Applied Research Project CSIS-4495-050*

# TABLE OF CONTENTS

1. Introduction: ………………………………………………………………………… 3
2. Proposed research project: ………………………………………...…………….. 6
3. Project planning and timeline: ……………………………………….…………… 8
4. Implemented features: …………………………………………………………… 10
   1. Backend code: ……………………………………..…………….……… 11
   2. Frontend code: …………………………………………………….….….. 18
5. Lessons Learned and Future Work ………………………………..………...…... 29
6. Conclusion: ………………………………………………..……………….……… 31
7. References: ………………………………………………………..……………… 32
8. Appendix: ……………………………………………………………………………. 35
   1. Appendix A: Installation Guide: ……………………………………………… 36
   2. Appendix B: User Guide: ………………………………………………… .. 37
   3. Appendix C: Dataset and API Used: ……………………………………… 38
   4. Appendix D: Hardware, Software, Cloud, Architecture: ………………….. 39
   5. Appendix E: Code Explanation: …………………………………………… 40

# Introduction

The Home Workout and Analytics Suite is a web application which would analyze the health and fitness needs of individuals. This platform would integrate analytics with personalized data to enable users stay informed while engaging in home workout. Using visualizations, the web app would enable users to have a grasp of their fitness journey. The research focusses on understanding user behaviors and challenges faced, and its purpose is to reshape how people approach their health and fitness goals through data-driven insights and intelligent recommendations.

**The key questions to address in research would be:**

User Engagement and Motivation: How can the web app include personalized challenges to encourage users to use it and motivate them towards a healthier lifestyle? What features would encourage users to input their data and participate in the analytics?

Personalized Health insight: What level of personalization will the users need for health insights and recommendations based on their individual data and perceived need?

User Education: How would the web application provide users with education on interpreting their health analytics and enable them make informed decisions?

**Relevant literature Research**

In a research article entitled *Impact of Physical Inactivity on the World’s Major Non-Communicable Diseases,* Lee et al. (2013) explores the global impact of physical inactivity on major non-communicable diseases (NCDs), including coronary heart disease (CHD), type 2 diabetes, breast cancer, and colon cancer, along with its influence on life expectancy. It goes on to explain that worldwide, physical inactivity is found to be responsible for 6-10% of the burden of CHD, type 2 diabetes, breast cancer, colon cancer, and premature mortality. Eliminating physical inactivity is estimated to increase the world's life expectancy by 0.68 years.

While these health issues are significant, in an article written by Cronkleton(2019) he shows that a previous research suggested that strength training a form of exercise involving moving the body against resistance, which can include body weight or free weights like dumbbells could help mitigate these health challenges. These are versatile workouts that can be performed almost anywhere. The benefits of strength training are extensive, as research suggests it can contribute to a reduction in the symptoms of many chronic conditions, including back pain, diabetes, arthritis, and heart disease. In addition, it helps in building lean muscle mass, reducing body fat, enhancing calorie burning efficiency, boosting metabolism, improving bone density, increasing flexibility, and benefiting cognitive functions.

**An existing knowledge gap**

The research done so far has not addressed **Customized Risk Assessment:** an assessment of a user’s risk of major Non-Communicable Diseases (NCDs) based on their physical activity levels. This web application would include a progress tracking system that visually displays the potential health gains achieved through regular physical activity. The users would see improvements in their risk factors for coronary heart disease which would create a connection between workouts and health outcomes.

**Hypotheses, Assumptions, and Potential Benefits of research**

**Hypotheses:**

Fitness Improvement: The app will improve users' physical fitness.

Adherence to app usage: The app's features will encourage user adherence and engagement.

Health and Well-being: Regular app use will positively impact users' overall health.

**Assumptions:**

User Commitment: Users are committed to following the app's workout plans.

Follow Recommendations: Users will follow provided recommendations based on their health data analysis.

**Potential Benefits:**

Convenience: The app offers convenient home workouts.

Personalization: Tailored workouts for individual preferences and fitness levels.

Motivation: Visualizing progress boosts user motivation.

Healthier Lifestyle: Heeding recommendations will make users adopt a healthier lifestyle.

# Proposed Research Project

**Research Design, Objectives, Methodology and its Justification**

**Research Design:**

The research is based on user suggestions. These will shape the features to be included in the application. User perspectives and preferences will be inquired and noted. Improvements on already existing features will be iteratively implemented.

In the design, a user creates an account. Next, the user is profiled by entering their age, gender, height, weight, medical history, and fitness goals, etc. Following that, the user takes an initial health assessment where they provide their blood pressure and resting heart rate. The user is prompted to set their fitness goals. Subsequently, they see workout suggestions based on their fitness level and select a workout to get started with. During the workout, the app records each completed session (duration and intensity). Once a workout session is completed, the user provides their blood pressure and resting heart rate. Following this, the user can visualize on charts: their heart rate analytics, their exercise analytics, calorie expenditure and goal progress. A health risk assessment is also provided to offer insight into potential health risks. Additionally, users receive suggestions on healthy nutrition.

**Objectives:**

User Experience Assessment: Evaluate the overall user experience by analyzing the web app’s usability and the ease of navigating through various features.

Feature Preferences: Identify specific features that users would want to see in the web app, exploring which features contribute most to their engagement.

Barriers to Engagement: Investigate potential obstacles which might hinder consistent app usage.

**Methodology:** User surveys and interviews, allowing users to share their thoughts, preferences, and suggestions

**Justification:** This research methodology allows for an understanding of users' perspectives. It provides a detailed understanding of the users' motivations, preferences, and challenges which will impact on the usage of the web application.

**Data Collection Methods:**

Surveys: I will conduct online surveys to gather data on user preferences.

Interviews: I will conduct interviews with a selected sample of British Columbia residents to get insight on user experiences and expectations.

Sample Size:A diverse sample of British Columbia residents will be sampled to cut across different racial, age and social economic status.

**Technologies I will be using:**

Operating System: Windows 11

Programming language(s)/Scripting or framework: JavaScript(React JS and Node JS/Express JS)

Database: Mongo DB

Front-end and backend: React JS and Node JS

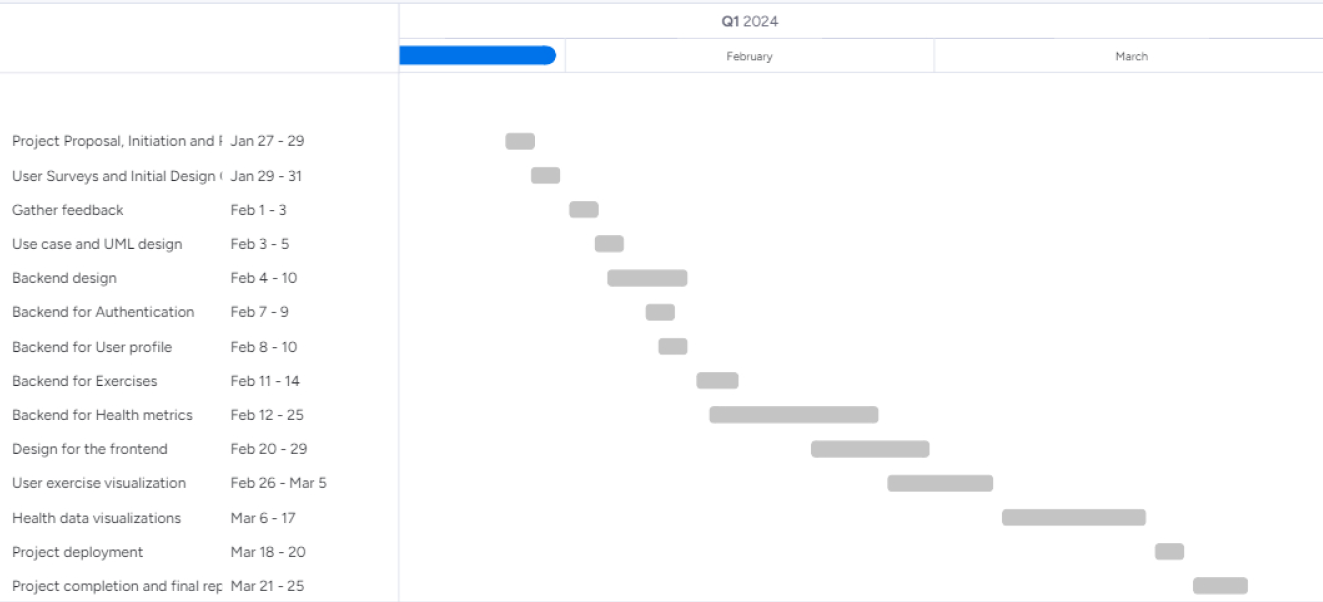
**Expected Results:**

The research will result in learning which features appeal to users the most and help in improving the app's performance. By understanding what users want, I can make the app work better and provide better user experience. The application would promote better health and motivate users to pursue healthier lifestyles which will increase life span.

# Project Planning and Timeline

|  |  |  |
| --- | --- | --- |
| **Milestones** | **Deliverables.** | **Deadlines** |
| Project proposal, initiation and research | * Define project objectives and scope. * Conduct preliminary research on user needs | January 29 |
| User Surveys and Initial Design Concepts | * Create and distribute user surveys. * Generate initial design concepts based on survey insights. | January 31 |
| Gather feedback | * Collect feedback from a small group of potential users. | February 3 |
| Use case and UML design | * Design UML for the app use cases | February 5 |
| Backend design | * Backend design for the application | February 10 |
| Server side creation and database integration | * Authentication * User profile * Exercises * Health metrics | February 25 |
| Frontend design and integration to backend | * Design for the frontend * User exercise visualization * Health data visualizations | March 17 |
| Project deployment | 1. Deploy application using Namecheap hosting | March 20 |
| Project completion and final report | 1. Written report on project | March 25 |

**Project Gantt Chart:**



# IMPLEMENTED FEATURES

Introduction

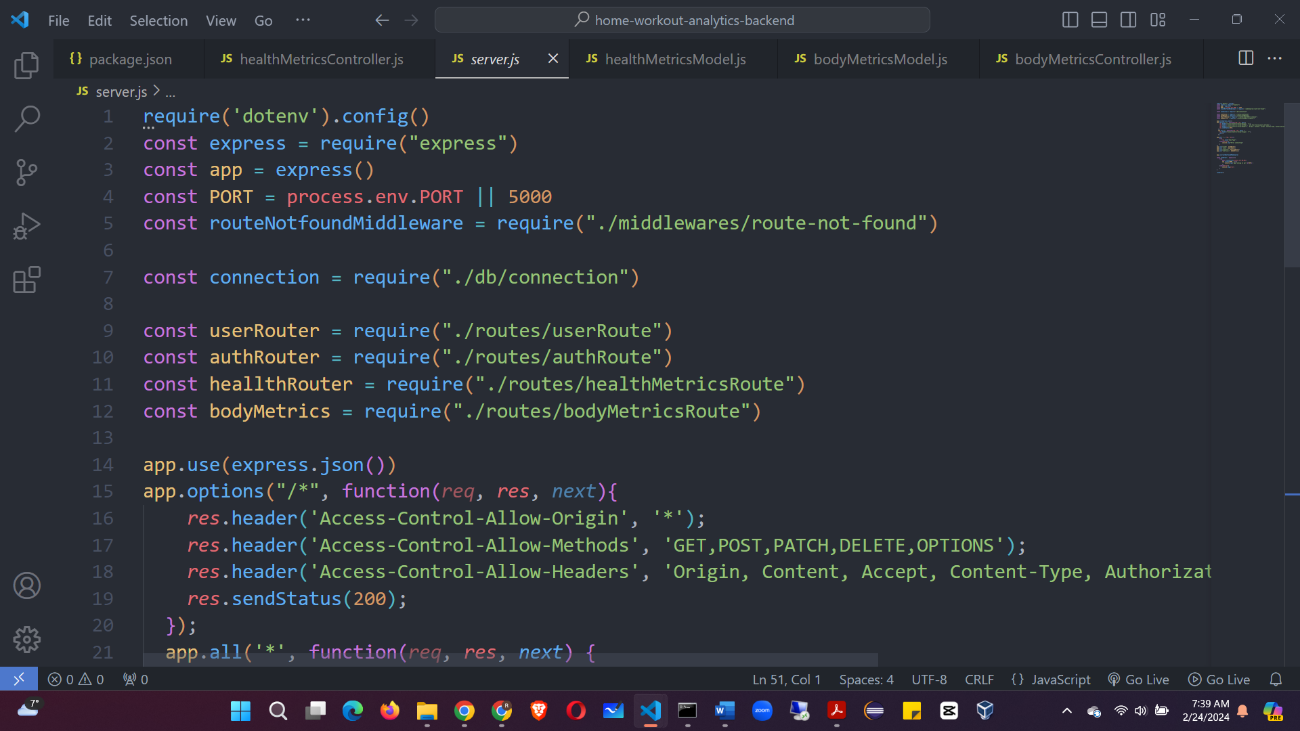
In this section, I will provide an overview of the implemented features in the web application, covering the backend and frontend components. On the backend, I established a structure, created data models, controllers, and routes. On the frontend, I developed authentication pages, created user profiling, health assessment, and body metrics pages to gather user data, and created a workout data resource. Additionally, I created workout sessions based on the workout data resource and generated user reports, health risk analyses, and nutrition tips based on user health and exercise data.

# BACKEND CODE:

# Authentication, User Data, Health Data and Body Metrics

**Introduction**

This section focuses on the backend for the web application, which enables communication between the frontend and the database. In the backend, I created data models for users, health assessments, and body metrics, and implemented controllers to manage authentication and data operations. Additionally, I configured routes to enable accessibility to the backend API. The following shows the different sections and their respective purposes.

****

**Img 1: Server**

**Server (*server.js*)**

I created this as the main entry point to the backend application. It has connection to the database, definition of routes for user authentication, record of health and body metrics data. Additionally, I used environment variables for configuration, to make it adaptable to different environments. The server is run using an *async* function. *See Appendix E. Backend. Server.js* for more explanation.

**Models**

**A screenshot of a computer program

Description automatically generated**

**Img 2: User Model**

**User Model (*userModel.js*):**

This model represents user data. As Img 2 shows, the schema has fields for the user's personal information. The schema also includes validation for email format, password length, and constraints for age, gender, height, and weight. Additionally, I included a pre(“save”, ….) middleware to hash the user's password before saving it to the database.

**Health Metrics Model (*healthMetricsModel.js*):** This models the user’s health data. The schema includes fields for the user's id, systolic blood pressure, diastolic blood pressure, and resting heart rate. Each record is created with a date. If a new record is entered for the same date, it updates the previous record.

**Body Metrics Model (*bodyMetricsModel.js*):** This models the user’s body metrics. The schema includes fields for the user's id, chest circumference, waist circumference and hip circumference. Each record is created with a date. If a new record is entered for the same date, it updates the previous record.

See *Appendix E. Backend. BodyMetricsModel.js, HealthMetricsModel.js, UserModel.js:* for more explanation.

**Controllers**

I created controller functions (*authController.js, userController.js, healthMetricsController.js, bodyMetricsController.js*) to handle actions, such as user authentication, user profile management, health assessment record, and body metrics record. These controllers utilize the models.

**Authentication Controller (*authController.js*):** The auth controller has the following functions: *signUpUser()* and *signInUser()*.

signUpUser: It first checks for existing *email* and *username* to prevent duplication. It also converts user input to lowercase for consistency. Next it creates a new user, hashes the *password*, and saves it. Then the method returns a *JSON* response with the new user details.

signInUser: The method first converts the provided *email* or *username* to lowercase. It next retrieves user data with provided *email* or *username*. It then compares passwords using *bcrypt*. And finally returns a *JSON* response indicating success or failure.

**A screen shot of a computer

Description automatically generated**

**Img 3: User Controller**

**User Controller (*userController.js*)**

The userController has 2 functions: *getUserProfile()* and *saveAdditionalUserInfoById()*

getUserProfile: As shown in Img 3, this function retrieves a user's profile information by their *id*. If the user is not found, the function returns a 404 error. If the user exists, it returns a *JSON* response with the user’s details.

saveAdditionalUserInfoById: This method updates additional user information by using the *findByIdAndUpdate()* method to modify the user's data. It returns a *JSON* response indicating the successful saving of additional information.

**Health Metrics Controller (*healthMetricsController.js*):** This controller has 2 functions: *recordHealthMetrics()* and *getHealthMetrics()*.

recordHealthMetrics: This method records a workout health assessment for a user identified by their *id*. It extracts the systolic pressure, diastolic pressure, and resting heart rate from the request’s body. It then generates the current date in the 'YYYY-MM-DD' format. Next, it creates a new object with the extracted data and saves it using the user’s *id*.

getHealthMetrics: This method retrieves workout health metrics for a user. It extracts the user’s *id* from the URL. It first checks to see if the user exists. If the user does not exist it returns an error. If it finds the user, it uses the *id* to search *HealthMetrics* for the user’s health data and then it returns a *JSON* response with health metrics recorded.

**Body Metrics Controller (*bodyMetricsController.js*):** This controller has 2 functions: *recordBodyMetrics()* and *getBodyMetrics()*.

recordBodyMetrics: This method records body metrics for a user identified by their *id*. It extracts the chest, waist and hip circumference from the request’s body. It then generates the current date in the 'YYYY-MM-DD' format. Next, it creates a new object with the extracted data and saves it using the user’s *id*.

getBodyMetrics: This method retrieves a user’s body metrics. It extracts the user’s *id* from the URL. It then checks to see if the user exists. If the user does not exist it returns an error. If it finds the user, it uses the *id* to search *BodyMetrics* for the user’s health data and then returns a *JSON* response with the body metrics.

See *Appendix E. Backend. AuthContoller.js, BodyMetricsContoller.js, HealthMetricsContoller.js, UserContoller.js* for more explanation.

**Routes:**

**A screen shot of a computer

Description automatically generated**

**Img 4: User Route**

**Routes (*****authRoute.js, bodyMetricsRoute.js, healthMetricsRoute.js, userRoute.js*)**

I set up routes (authRoute.js, bodyMetricsRoute.js, healthMetricsRoute.js, userRoute.js) for different actions, such as user authentication, retrieving user profile, handling health and body metrics data. The routes shown above, in Img 4 are of *userRoute.js* – the functions included in it are getting and updating a user. These routes specify the endpoints that clients can interact with.

*See Appendix E. Backend. AuthRoute.js, BodyMetricsRoute.js, HealthMetricsRoute.js, UserRoute.js* for more explanation.

**Removed Middleware (*route-not-found.js*):**

I removed the *route-not-found* middleware which handled routes not specified in the application routes. I now handle such unlisted routes in the frontend to reduce requests made to the backend.

# FRONTEND CODE:

# Pages, Components, Context API, Resources

**Introduction**

This section focuses on the frontend of the web application. I used React Context API to manage some state values and give some structure to the data flow. This allowed for efficient state management and data sharing across different components. I developed authentication pages for user login and registration. I also created user profiling, health assessment, and body metrics pages to gather essential user data before and after workout sessions. Additionally, I created a workout session page, which fetches data from workout data resource, and I also generated user reports, health risk analyses, and nutrition tips to provide valuable insights based on user health and exercise data. The folder structure includes directories such as pages, components, context, reducer, and resources. The following shows the different sections and their respective purposes.

**Pages:**

This directory holds the pages which include the home page, authentication pages, user profiling page, record health metrics page, individual workout pages, workout session page, record body metrics page, health report page, health risk assessment page and nutrition tips page.

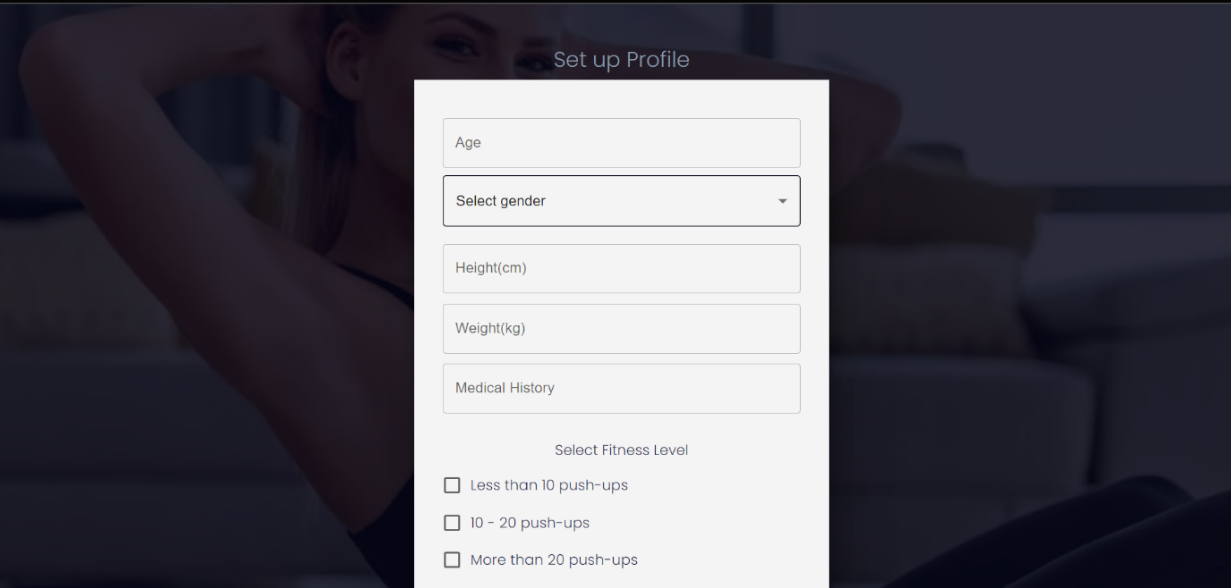
A screenshot of a person working out

Description automatically generated

**Img 5: Sign-up page**

**Authentication(*signup.jsx and signin.jsx*):** This folder holds 2 folders: the sign-in and sign-up folders. In each folder is a react file which creates a form, accepts user input and registers or authenticates the user using the endpoints created in the backend. If a user’s credentials are incorrect, an alert is generated to warn the user. It saves the user’s login credentials. This would be fetched and updated in the user profiling page. Img 5 shows one of the authentication pages: the sign-up page.

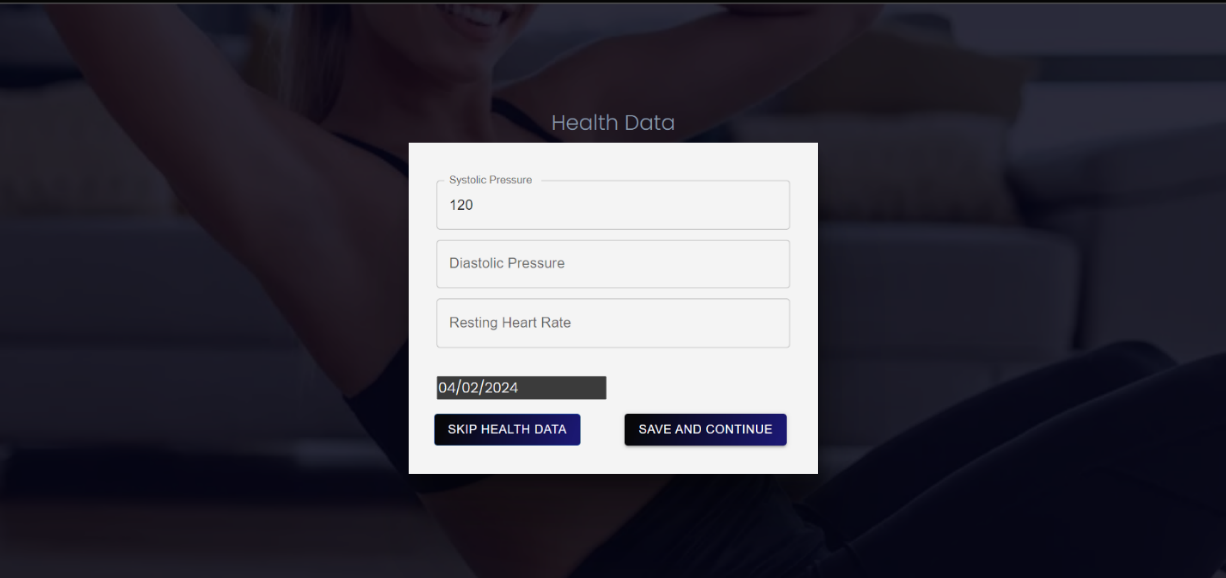
See *Appendix E. Frontend. Signup.jsx* and *Appendix E. Frontend. Signin.jsx* for more explanation.

****

**Img 6: User Profiling**

**User Profiling(*userprofiling.jsx*):** As seen in Img 6, this page enables a user to create his profile by adding his age, height, weight, medical history, fitness level and fitness goals. The user data created in the sign-up process is fetched and updated with the additional values.

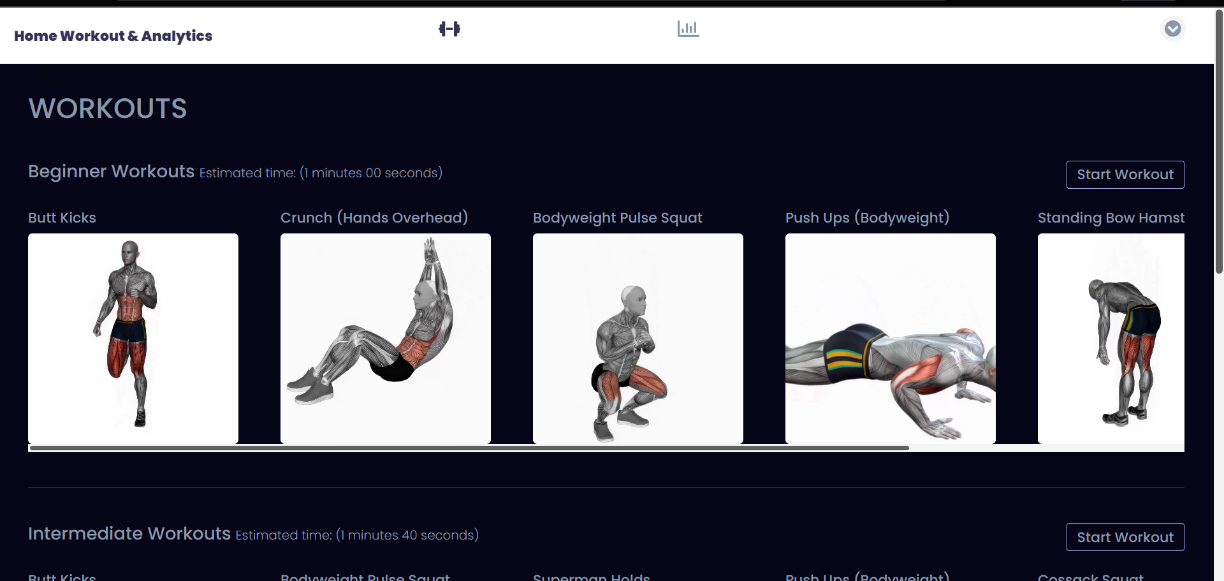
See *Appendix E. Frontend. UserProfiling.jsx* more explanation.

****

**Img 7: Record Health Metrics**

**Record Health Metrics(*recordHealthMetrics.jsx*):** This folder contains the health metrics recording page which is used to record the user’s systolic pressure, diastolic pressure and resting heart rate. The page is seen rendered in Img 7.

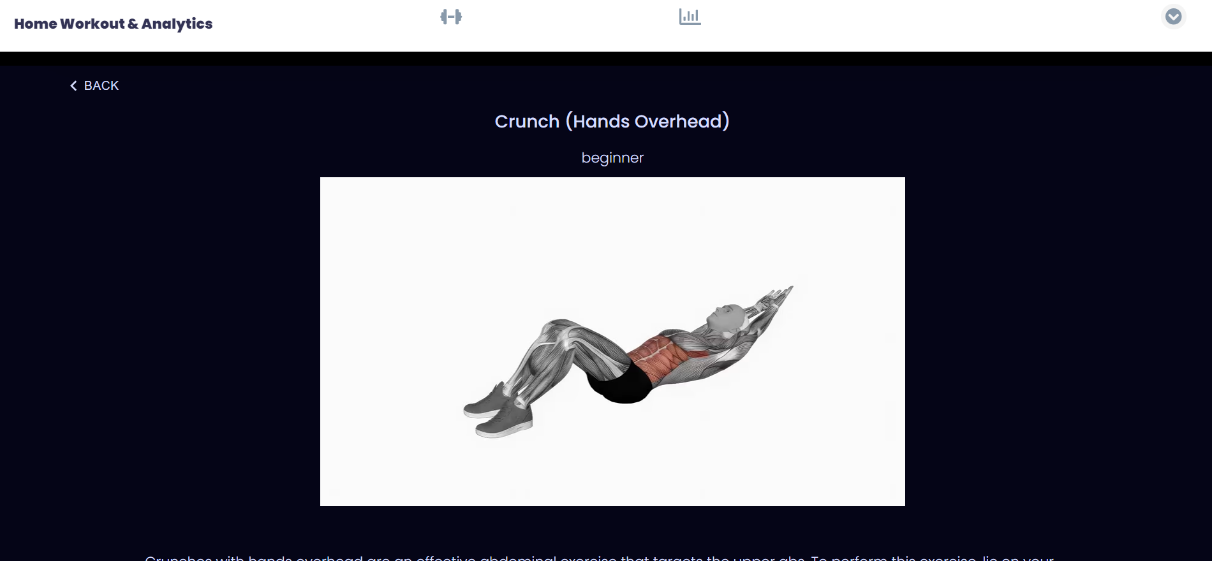
See *Appendix E. Frontend. RecordHealthMetrics.jsx* for more explanation.

****

**Img 8: Home Page**

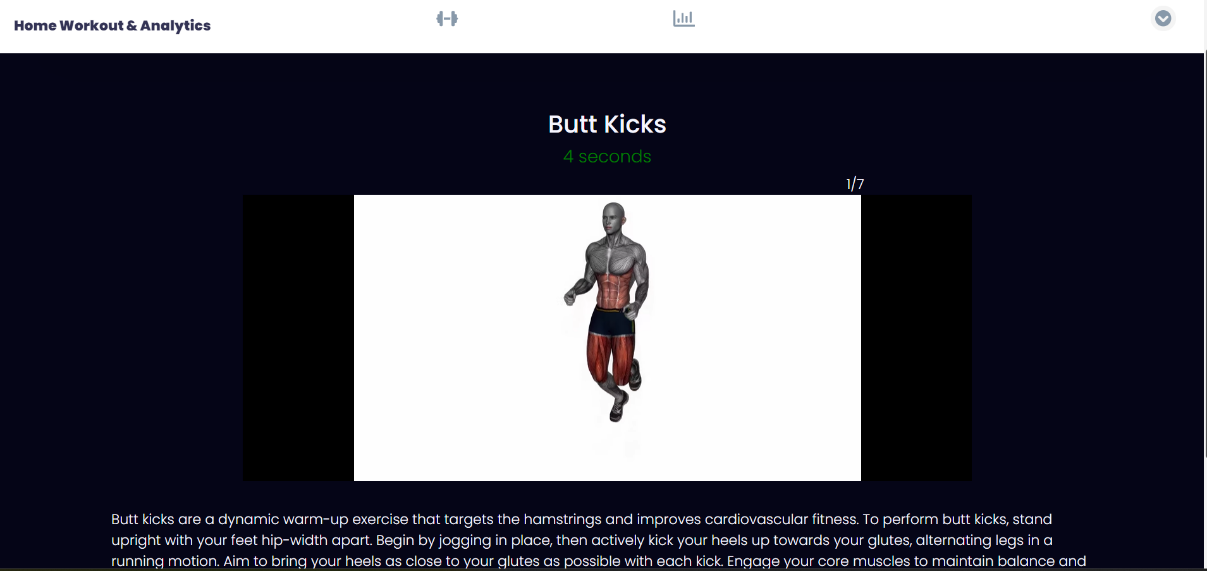
**Home Page:** The home page shown in Img 8, fetches and renders the exercises from the *Resources* directory and displays them in 3 difficulty levels. When users sign-up, they select their fitness level. They can then select a workout session based on their fitness level.

See *Appendix E. Frontend. HomePage.jsx* for more explanation.



**Img 9: Individual Workout Page**

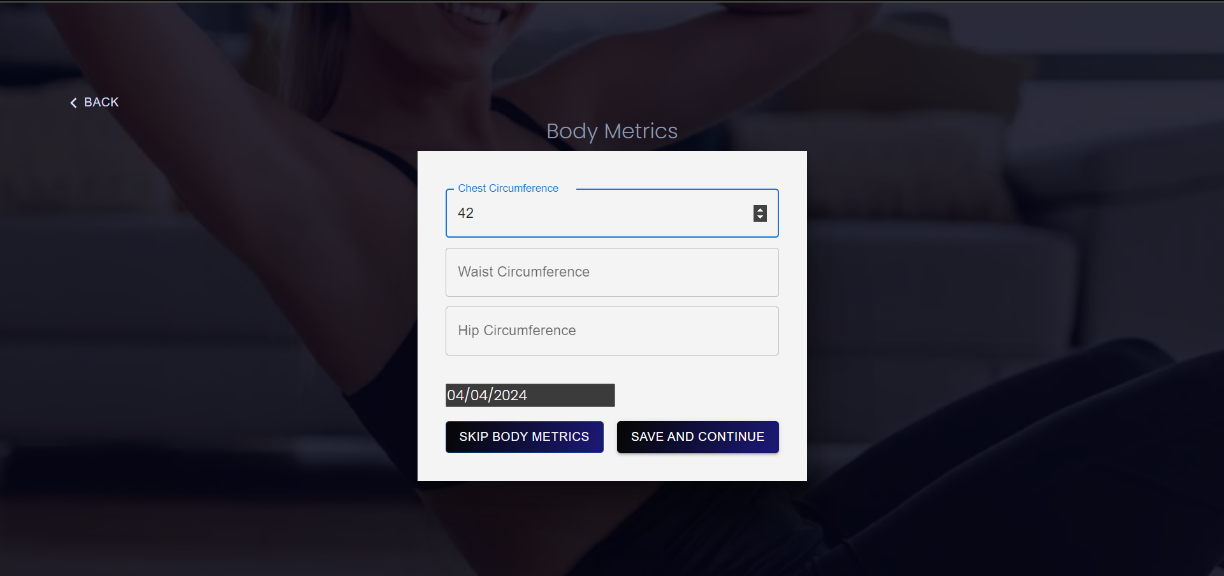
**Individual Workout Page(*workout.jsx*):** This is a page which shows the details of each exercise. It provides explanations of how each exercise is done. It is accessed by clicking on the exercises’ picture on the Home page.

****

**Img 10: Workout Session**

**Workout Session(*workoutSessions.jsx*):** This page holds the workout session. Exercises are filtered based on the *category*. Once the page loads, the workout session starts with the *preparationTime* of the first exercise. After the *preparationTime*, the main timer (called *duration*) for that exercise starts counting down. The page displays a video demonstration of the current exercise and its details, as well as the number of exercises in the workout session and the position of the current exercise. After the first exercise is completed, the second one starts its *preparationTimer.* As the workouts are going on, the calories burned are also being calculated from the exercise data.

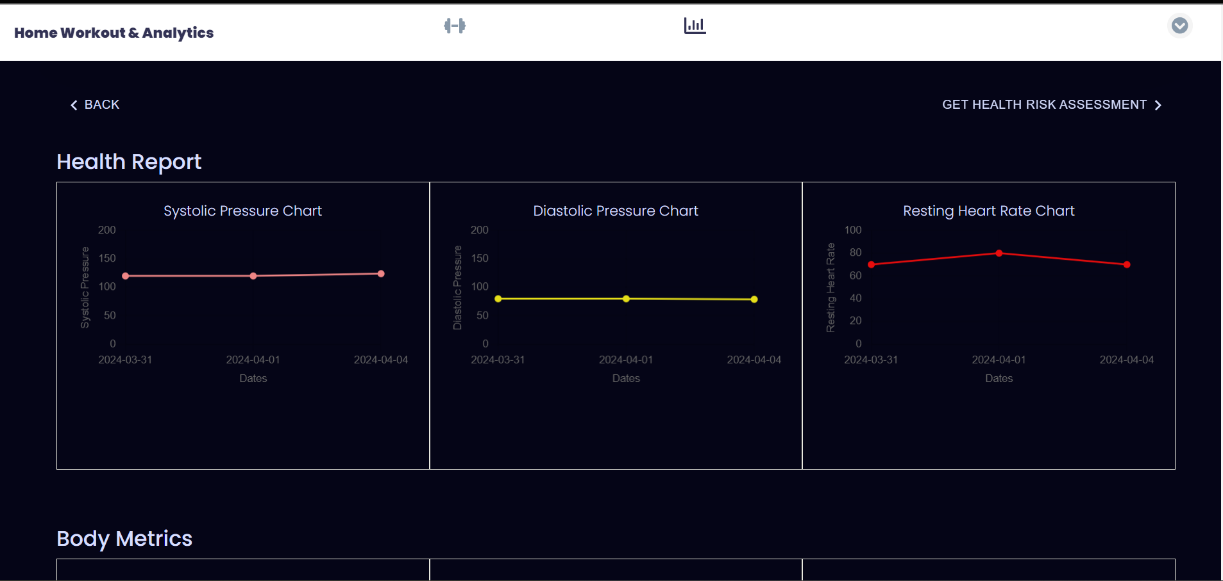
See *Appendix E. Frontend. WorkoutSessions.jsx* for more explanation.



**Img 11: Record Body Metrics**

**Record Body Metrics (*recordBodyMetrics.jsx*):** Shown above in Img 11, this page takes records of the user’s body measurements after each workout session. So, the user would need a tape measure to take measurements of their chest, waist and hip circumference and this record is stored.

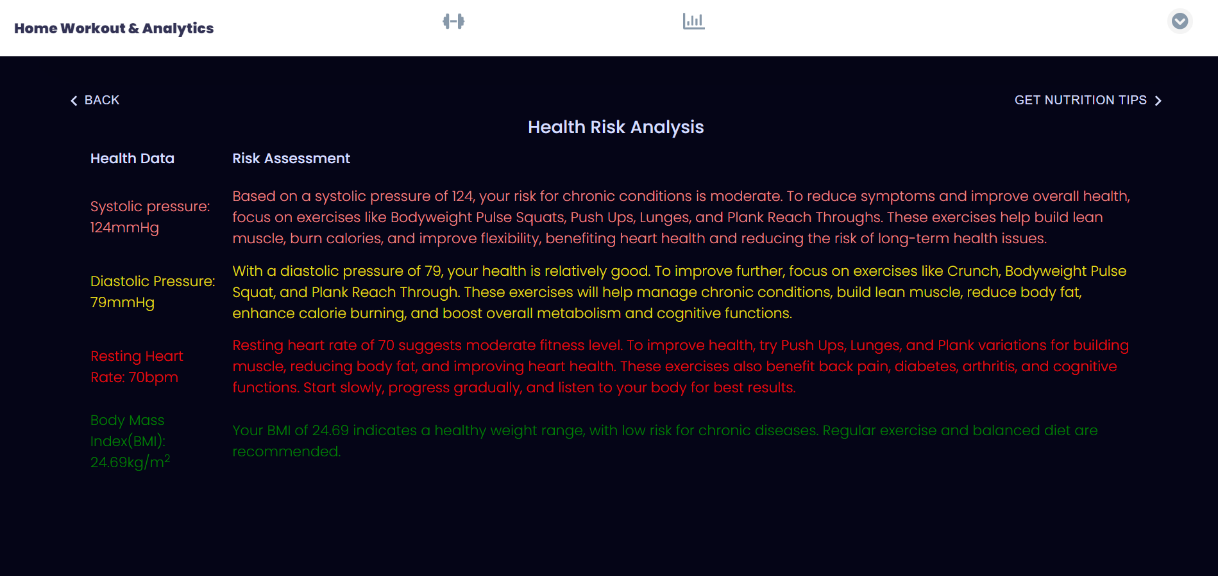
See *Appendix E. Frontend. RecordBodyMetrics.jsx* for more explanation.

****

**Img 12: Health Report Page**

**Health Report Page (*report.jsx*):** Shown in Img 12, this page displays on a graph the daily health report of the user including the user’s systolic and diastolic pressures along with their resting heart rate. It also displays the user’s body measurements on separate graphs. On this page are also the user’s BMI and a graphical display of the user’s daily calorie expenditure.

See *Appendix E. Frontend. Report.jsx* for more explanation.



**Img 13: Health Risk Assessment Page**

**Health Risk Assessment Page (*riskAssessment.jsx*):** On this page, a third-party API (*ChatGPT*) is utilized to provide an assessment of the user’s health given their provided health data. It also provides fitness recommendations to the user. The user’s data is fetched. This data is then added as part of the optional parameters to the *ChatGPT* endpoint. This is done for each health parameter as well as the BMI. The response is fetched and set to be displayed to the user as seen in Img 13 above.

See *Appendix E. Frontend. RiskAssessment.jsx* for more explanation.



**Nutrition Tips Page**

**Nutrition Tips Page(*nutritionTips.jsx*):** This page fetches the user’s health data, and when the health data is available along with the user’s data it uses a *useEffect()* to call the *chatResponseForNutritionTips()* function. This function uses the user’s health data, medial history, fitness goals, fitness level and BMI to provide a tailored nutrition recommendation as response from *ChatGPT* API.

See *Appendix E. Frontend. NutritionTips.jsx* for more explanation.

**Components:**

The components include the *topbar*, the *sidebar*, and the *backdrop*.

**Topbar and Sidebar(*topbar.jsx and sidebar.jsx*):** The topbar and sidebar are the navigation bar located at the top of each page in desktop mode and the navigation bar which shows on the side in mobile mode respectively. It includes the web application’s name, the navigation links and the sign-out button when the user is already signed-in.

**Backdrop:** This is a component used to create an overlay effect in mobile mode when the sidebar is open.

**Context and Reducer:** The context and reducer contain functions used in managing the state of the entire web application such as setting log-in, opening the sidebar and fetching the user.

See *Appendix E. Frontend. Context API* for more explanation.

**Resources:**

This directory holds other utility files and data needed in the application such as the URL for the backend endpoint and the data for workouts.

**Use Case Diagram:** The Use case diagram provides a high-level overview of the app's functionalities.

A diagram of a workout

Description automatically generated

This use case diagram shows the user’s and the application’s use cases.

**Assumptions Made:**

Assumptions are based on the user having a blood pressure checker that provides measurements for systolic pressure, diastolic pressure, and resting heart rate. Also, that the user would have a tape measure for body metrics measurement such as chest, waist and hip circumference.

**Action Taken / Changes in Direction:**

I developed my own workout API to supply the required data. I began by obtaining copyright-free exercise videos, images and related information for each exercise. I also deemed it better to enable users record their blood pressure statistics before workout sessions since this would give inaccurate health information if taken immediately after workout sessions.

# LESSONS LEARNED AND FUTURE WORK

While working on this project, I learned a few things and would improve on the following:

**User data collection:** In collecting the user’s data when the user sign’s up, it would be better to make the medical history a bit more elaborate by providing users with a series of checkboxes to fill instead of using a text area.

**Exercise recommendation:** It could be better to make workout recommendations not only based on fitness levels, but also on medical history and fitness goals. This could be done by getting the user’s fitness levels, fitness goals and medical history, feeding it as parameters to *ChatGPT* API to make recommendations for a certain number of exercises based on the user’s fitness level (for example, to make more exercises recommendations for higher fitness levels), then also make the recommendations in view of the user’s medial history and fitness goals. The recommended exercises would then be used to filter exercises in the exercise dataset and used as the current workout (*currentWorkouts*) list for each exercise session.

**Workout progression:** It would also be better to analyze user’s current fitness level and encourage improvement by gradually increasing the intensity, duration, and complexity of the workouts. This will prevent boredom and muscle growth stagnation.

**Creating and celebrating milestones:** Another feature which would make the application better is the inclusion of milestones which could be achieved after a user completes several consecutive days of exercises. For instance, going for an entire month without a break would be a level of achievement, or going from beginner level fitness to advanced level fitness within a month could also be another achievement.

**Goal progress:** In the future I would like to add a goal progress tracker so some goals can be tracked. This would involve making the user provide both a current value and a value for their future goal. For instance, when providing their weight, users would also be prompted to provide their target weight and target time frame to achieve the weight. With these values, users’ progress can be tracked over time to check progress towards their goals.

# CONCLUSION

The Home Workout and Analytics web application is an easy-to-use web application for fitness and health analytics. The backend works with the frontend to promote fitness and enable users gain insight into their health analysis, thus providing a practical fitness web application.

The user experience starts with authentication pages which server for user registration and login, while user profiling, health assessment, and body metrics pages gather essential user data to personalize the user experience. The generation of user reports, health risk analyses, and nutrition tips adds enhances user experience by providing insights based on user health and exercise data. Additionally, the use of a custom workout API ensures that available workouts are practical and targeted to the applications goals. The project would empower users in their journey towards better health and fitness.

Lessons learned during the development process include the importance of better user data collection methods, exercise recommendation strategies, the implementation of milestones and goal tracking features. These features would be implemented in subsequent iterations of the application.

The Home Workout and Analytics web application utilizes technology to promote health and fitness and establishes groundwork for future improvements and expansions.

# References

Lee, I., Shiroma, E. J., Lobelo, F., Puska, P., Blair, S. N., & Katzmarzyk, P.T(2013). Impact of Physical Inactivity on the World’s Major Non-Communicable Diseases. *National Library of Medicine*, 380(9838), 219–229. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3645500/>

Cronkleton, E. (2019). How to Get a Full-Body Strength Training Workout at Home. *Healthline*. <https://www.healthline.com/health/exercise-fitness/strength-training-at-home>

Google. (n.d.). *02 - Conditioning Exercise.* Compendium of Physical Activities. <https://sites.google.com/site/compendiumofphysicalactivities/Activity-Categories/conditioning-exercise>

Kaminski, J. (n.d.). *Metabolic Equivalents for Weight Loss: What Are They & How to Calculate Them.* NASM. <https://blog.nasm.org/metabolic-equivalents-for-weight-loss#:~:text=HOW%20TO%20CALCULATE%20METS>

Exercise Animatic. (n.d.). Butt kicks fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711631372>

Exercise Animatic. (n.d.). Standing bow hamstring stretch fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711584062>

Exercise Animatic. (n.d.). Dips between Chairs fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711587358>

Exercise Animatic. (n.d.). Crunch (straight leg up) fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711593939>

Exercise Animatic. (n.d.). Adductor stretch fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711581408>

Exercise Animatic. (n.d.). Lunges alternate leg with pause bodyweight fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711563253>

Exercise Animatic. (n.d.). Boat Stretch fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711581842>

Exercise Animatic. (n.d.). Cossack Squat fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711556047>

Exercise Animatic. (n.d.). Crunch (hands overhead) fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711593889>

Exercise Animatic. (n.d.). Cobra push up fitness exercise workout animation video muscle highlight 4K [Vidoe]. Adobe. <https://stock.adobe.com/ca/711571751>

Exercise Animatic. (n.d.). Bodyweight Pulse Squat arms to chest fitness exercise workout animation video muscle highlight 4K [video]. Adobe. <https://stock.adobe.com/ca/711554294>

Exercise Animatic. (n.d.). Back/superman holds fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711616796>

Exercise Animatic. (n.d.). Chair triceps dips fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711587669>

Exercise Animatic. (n.d.). Plank reach through fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711601950>

Exercise Animatic. (n.d.). Jack knives fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711598194>

Exercise Animatic. (n.d.). Brazilian crunches fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711594081>

Exercise Animatic. (n.d.). Push ups bodyweight fitness exercise workout animation video muscle highlight 4K [Video]. Adobe. <https://stock.adobe.com/ca/711547882>

# APPENDIX

1. Appendix A: Installation Guide:………………………………………………………35
2. Appendix B: User Guide: ..…………………………………………………………...36
3. Appendix C: Dataset and API Used: ..………………………………….…………...37
4. Appendix D: Hardware, Software, Cloud, Architecture: ......……………………...38
5. Appendix E: Code Explanation: ..……………………………………………………39

# A

# Installation Guide

**Steps in installing and running the Home Workout and Analytics Web Application**

Start the backend server by running ***node server.js***

Start the frontend application by running ***npm install***then ***npm run dev***

***Note:***

* ***The node\_modules in the frontend was deleted to enable ease of submission. Please install using “npm install”***
* ***Also, the frontend is configured to access the backend on 'http://localhost:5000'***

After running the above commands, the web application opens on your browser.

# b

# USER Guide

After installation, follow these steps to use the web application:

* The first page to load after installation would be the sign-in page.
* Click on ***Register***, to register an account.
* After registering an account, create a user profile.
* After creating a user profile, fill in and submit the health data form.
* Next, select a workout to get started with.
* After the workout session, record your body metrics.
* Next the health and body reports can be viewed.
* After viewing the health report, click no ***Get Health Risk Assessment*** to view your health risk assessment.
* After viewing your health risk assessment, click on ***Get Nutrition Tips,*** to get smart nutrition tips.
* To record data for another date, use another date while filling in the Health Data (Metrics) and Body Metrics forms. This will plot graphs of health data and body metrics against the dates.

# C

# Dataset and API Used

The API used in Home Workout and Analytics web application was designed specifically for the application.

I researched on workouts which would best suit the purpose of the web application. To get an accurate relationship between the exercise duration, intensity, and the Metabolic Equivalent of each exercise so as to get an accurate calorie consumption, I consulted some web sites listed in the References section. I also used video resources from *Adobe Stock* for which I obtained an [Enhanced License](https://stock.adobe.com/ca/license-terms) which allows for unlimited web views, modification of asset and up to 500,000 copies or views of the asset. This is a client-side API, stored and used in the frontend of the application.

# D

# Hardware, Software, Cloud, Architecture

In the design of Home Workout and Analytics web application,

The hardware used was HP Laptop 15-dw3xxx.

The software used are: Visual Studio Code editor, Postman, Chrome browser, Node version 20.10.0 and a host of third-party node packet manager (NPM) installations.

The cloud infrastructure used was Mongo DB Atlas.

The architecture of the Home Workout and Analytics application is based on a React frontend, a Node JS (Express) backend, and MongoDB Atlas database management. Additionally, a client-side API is used to fetch exercise data for the application. With this setup React handles the user interface, Express manages server-side logic, and MongoDB Atlas stores and manages the application's data. The backend is structured starting with the server’s main application which uses the routes (endpoints) to access the controllers. The controllers in turn have access to the models which dictate the structure of the data which is stored on Mongo DB Atlas.

# E

# Code Explanation

**Backend:**

Server.js: The **server.js** file starts the backend of the application. It starts off by requiring and configuring *dotenv* which is used to access the environmental variables. It requires and uses *express* to create the backend application. A connection is created using *mongoose.connect()* and is required in the *server.js* to make a connection to the Mongo DB Atlas. The *server.js* file also requires and uses the routes for authentication, health records and body metrics records and user. Next, the application uses *express.json()*  to enable the use of *JSON* in the requests. Next the application configures middleware to enable *CORS* when the application is later deployed. The application then runs the server using the *runServer()* method, which is a method that waits for connection and then listens to port 5000 after a connection is established. (See Img 1)

AuthRoute.js, BodyMetricsRoute.js, HealthMetricsRoute.js, UserRoute.js: These files contain routes which require *express* and then create a *router* using *express.Router()* function. This *router* uses its *get, post and patch* methods to enable access to different endpoints in the backend. It also uses as a second parameter to its arguments - the controller function which would run on the given route. These controller functions are *required* from the controllers section discussed below. (See Img 4)

AuthContoller.js, BodyMetricsContoller.js, HealthMetricsContoller.js, UserContoller.js: These files require the needed model(s) and then create functions that utilize the models to either create, read or update a record. If a value (such as an id or an email) is passed in through the *req.params* or *req.body*, the value is obtained and checked to ensure a record exists with such value. If it doesn’t exist an error response is returned. If the value does exist, the necessary procedure is taken.

To create a record, a *new* object of the model is created using the values sent in from *req.body* and using the *save()*  method, it is saved in the database.

To read a record from the database, an identifier (e.g. *id*) is obtained from the *req.params* and used to search for and return the requested value by using the *findById()* or *findOne()* methods.

To update a record, the identifier (*id)* is obtained from the *req.params,* The new values to update with are obtained from *req,body*. A *new* object of the model is created using the new values. The *findByIdAndUpdate()* method is then used to update the database record where the identifier matches the one returned in *req.parms.* (See Img 3)

BodyMetricsModel.js, HealthMetricsModel.js, UserModel.js: *Mongoose* is required, it is then used to create a new mongoose schema which is a structure of the model. This schema is like a blueprint that outlines the expected properties and their types for the model. The schema created is then exported to be used in the controller functions. (See Img 2)

**Frontend:**

Main.jsx: This file is where the React app is rendered to *index.html.* It contains the *App.jsx* and *AppProvider* which comes from the *app-context,jsx*. The *AppProvider* wraps around the entire application as it handles application-wide state.

App.jsx: This file uses the *BrowserRouter, Routes and Route* from *react-router-dom* to create a navigation router for the application. The *Route* component takes the component to be rendered in its *element* prop, and then renders it on the path specified on its *path* prop. There is also a wildcard path (path=’\*’) which renders an error page when an accessed route is not specified.

Resources: This folder contains the *api.jsx* which contains the backend URL, the *data.jsx*, which contains the exercise API.

Context API: The context API includes the *app-context.jsx* contained in the Context folder and the *app-reducer.jsx* contained in the Reducer folder. Both are essential in managing the state of the application. To explain how they both work, I’ll use one instance – that of setting the value of the current user when a user signs in.

When a user signs in, the backend responds with the user’s data, which includes the user’s id and email. This data is contained in the *loginData* object which is returned from the backend. The *setLoginValues()* method is used to call the *setCurrentUser()* method, passing the *loginData* as a parameter. This method and its parameter are then passed to the *app-context*.*jsx*, using the *UseAppcontext()* function. In *app-context.jsx*, the *setCurrentUser(),* still bearing the *loginData* object returned from the backend sets up the local storage to store the current userwith the name “HomeWorkoutAnalyticsCurrentUser”. The *currentUser* state property calls the *getCurrentUser()* function, which is a function that fetches the value of the current user from the local storage using the stored name “HomeWorkoutAnalyticsCurrentUser”.

The *currentUserParsed* property is set by calling the *fetchCurrentUser()* function which fetches the user’s complete data from the database using the *id* from *currentUser*. It is called when *currenUser* is set (changes in value). The returned user’s data is dispatched to the *reducer.jsx* using a constant named *CURRENTUSERPARSED* which is a string of value “CURRENTUSERPARSED”. In the *reducer.jsx,* the value of *CURRENTUSERPARSED* is then passed to the state property called *currentUserParsed.* With this, the user’s complete data is now available for use in the entire application as *currentUserParsed*.

Signup.jsx: The signup page contains a form which includes the user’s *firstname*, *lastname*, *email*, *username* and *password*. The values are set using the *setValues()* method, which is a function that takes the *name* on an input and matches it with the *value*. On clicking the “Sing up” button, the *signup* function is called, which submits the *formValues* and using an *axios POST* request. If sign up is successful, the user is redirected to the /*profiling* page to set up more profiles to his/her data.

Signin.jsx: This page contains a form which has inputs for the user’s *email* and *password*. On entering the user’s credentials, the *setValues()* method is called to set the values according to the *name* of the *value*. On clicking the “Sign in” button, the *signIn()* function is called which makes an *axios* *POST* call using the values provided. The backend authenticates the user and returns a successful or unsuccessful response to the user. If successful, the user is redirected to the /*record-health-metrics* page.

UserProfiling.jsx: This page is used to complete the user’s profile after they sign-up for an account. It contains a form which collects the user’s age, gender, height fitness goals, etc. On filling in the form and selecting the checkboxes and radio button as needed, the *setValues()*, *handleFitnessLevelChange(), handleCheckboxChange()* functions are called. These functions set the values for the form inputs. On clicking “Save and continue”, the *submitUserProfile()* function is called. In the function, it checks for selected fitness goals checkboxes. Any checked box is added to *selectedSport()* array. The input values are then used to send an *axios* *PATCH* request to the backend. If successful, the user is redirected to /*record-health-metrics* page.

RecordHealthMetrics.jsx: This page is used to collect data on the user’s health, specifically the user’s heart health. Data collected here includes the Systolic pressure, Diastolic pressure and Resting heart rate. The user has the option to skip recording the health data. However, if a user skips recording the health data and there is no previous record for the given date, that date would not have a record of activity and no data would be stored for the date with regards to health.

This is how it is implemented: When a user wants to skip health data, the *handleSkipHealthData() f*unction is called first. It tries to fetch the health data for the given date. If it does fetch health data for that date, the user is redirected to the Home page (“/”) and can continue using the application. However if it doesn’t fetch any data for that date, the *setConfirmationOpen()* function is called. It opens the confirmation dialog box and if the user clicks on “No”, the dialog box closes and allows the user to provide his health data. However, if the user clicks on “Yes”, the user thus confirms that he/she would skip and is redirected to the Home page (“/”) to continue using the application.

When a user provide their health data, the *submitHealthData()* is called when they click on “Save and continue”. This function uses an *axios POST* call to save the health record to the database and redirects the user to the Home page (“/”).

Additionally, if the date is changed, the *handleDateChanged()* function is called. It stores to the local storage the date which is selected with the key “HomeWorkoutAnalyticsExerciseDate”. If no date is selected, it defaults to the current date. This date selection is important so as to keep a consistent record of the daily health metrics. After the date is stored here, it would later be retrieved in the *workoutSessions.jsx* to enable the calories burned for that specific date to be stored in the database for that date.

When a previous body metrics record exists for a given date, any new form submission which has the same date will replace the already existing record in the database. If none exists, a new record is created for that date.

HomePage.jsx: The homepage is where a user selects a workout to perform. If a user is not authenticated and tries to access the homepage, they are redirected to /*sign-in*. The homepage uses the exercise data to create sections of exercises. It distinguishes the exercises based on *category* (i.e. beginner, intermediate or advanced). When a user fills in there profile data, they are asked to provide their current fitness level. Their level of fitness is used to make a recommendation to them for a workout session.

The home page also displays the estimated time each workout will take. This is obtained from *WorkoutData*. As the *getEstimatedTime()* method is called at each *category* of exercise, it has passed into it as a parameter, the *category* of the exercise. In the *getEstimatedTime()* function, the exercises in the selected workout are filtered. The *duration* and *preparationTIme* for all the filtered exercises are summed up. The *totalTime* is then passed into a function which converts it to a string showing it in minutes and seconds. This is then displayed as the estimated time for the workout session.

WorkoutSessions.jsx: This page contains the workout logic. The workout session starts by calling a *useEffect()* which first filters the exercises based on the *category*. Next, *setCurrentWorkouts()* function is used to set the *currentWorkouts* using the *DummyWorkoutData* and the filtered workout (named *mainCurrentWorkouts*). The *DummyWorkoutData* contains no real value. It’s only use to enable the workout session run starting from the beginning without loss of exercise data, since without using it the workout will start at position 1 (instead of 0), missing the first exercise in the workout. This will be explained further later.

Since the *currentWorkouts* are set, another *useEffect()* checks if the *timer === 0.* If it is, it sets the *nextExerciseIndex* to *currentExerciseIndex + 1.* This is where the use of the *DummyWorkoutData* comes in. Since the *DummyWorkoutData* has its duration set to 0, this check is truthy, and the *setCurrentExerciseIndex()* sets the *currentExerciseIndex* to the value of *currentExerciseIndex + 1,* and the first exercise (which is currently in position 2 in the *currentWorkouts*) starts the workout session. At this point, the *setPreparationTimer()* is also called and it sets the *preparationTimer* to the value of the *preparationTime* of the exercise in the position of *nextExerciseIndex* (which is now the *currentExercise*). Similarly, the *setTimer()* sets the *timer* to the value of the *duration* property in that *currentExercise.*

Once the *currentExerciseIndex* has changed (recall that *setCurrentExerciseIndex()* set the *currentExerciseIndex* to the value of *currentExerciseIndex + 1*), another *useEffect()* sets the c*urrentExercise* to the position of *currentExerciseIndex* in the *currentWorkouts* array*.* During this time, the metabolic equivalent for the specific exercise is used to calculate the calorie burned. Using this formular METS X 3.5 X BW (KG) / 200 = KCAL/MIN by Kaminski (n.d.), the KCAL/MIN is calculated. From this, the calories burned for the given time is gotten by multiplying the calories per minute by the duration in minutes.

When setting the details of the *currentExercise* (using the *setCurrentExercise()* function) If no workouts are available (i.e. if *nextExerciseIndex* >= *currentWorkouts.length*), the *currentExercise* is set to null. The *preparationTimer* starts counting down using *setInterval()*. When the *preparationTimer* countdown gets to 0, the *timer* for the exercise *duration* starts. At the same time, the countdown for the exercise *duration* begins using *setInterval()*. When the *timer* reaches 0, the application switches to the next exercise. It gets the index of the next exercise by simply adding 1 to the *currentExerciseIndex*. If there is no other exercise in this current index, the *currentExercise* is set to null, and “Workout complete” is shown. Otherwise, if there is another exercise in the new index, the *setCurrentExerciseIndex*() method sets the new *currentExerciseIndex*, thus calling any functions which depend on its change specifically calling the set*CurrentExercise()* and the set*PreparationTimer()* methods to set up the new *currentExercise*. The *setTimer()* method is a dependent of p*reparationTimer* and thus gets called when the *preparationTimer* is equal to 0.

In summary, the *currentWorkouts* is set using the exercise *category*. This calls the *setCurrentExercise()* and the *setPreparationTimer()* methods for that exercise. The *preparationTImer* starts counting down. When it gets to 0, the *setTimer()* method which is a dependent of it is called and also starts counting down. When the *timer* counts down to 0, a check is made to see if there is another exercise in the next index by calculating *nextExerciseIndex* which is equal to (*currentExerciseIndex* + 1). If no exercise is found in the new index, the *currentExercise* is set to null, otherwise, the current index is set to the *nextExerciseIndex* and the *preparationTimer* for the new exercise in the new index is started. The cycle continues until there is no exercise in the index of *currentExerciseIndex* *+ 1*.

The page also displays the number of exercises in the workout session and the position of the current exercise so a user would know his/her progress. This is done by displaying:

{*currentExerciseIndex*} / {*currentWorkouts.length*}.

RecordBodyMetrics.jsx: This page is used to collect data on the user’s body metrics: the chest, waist and hip circumference. The user has the option to skip recording the body metrics. However, if a user skips recording the body metrics and there is no previous record for the given date, that date would not have a record of activity and no data would be stored for the date with regards to body metrics.

Similar to *RecordHealthMetrics.jsx*, This is how it is implemented: When a user wants to skip recording body metrics, the *handleSkipBodyhData() f*unction is called first. It tries to fetch the body metrics data for the given date. If it does fetch body metrics data for that date, the user is redirected to the Home page (“/”) and can continue using the application. However if it doesn’t fetch any data for that date, the *setConfirmationOpen()* function is called. It opens the confirmation dialog box and if the user clicks on the “No”, the dialog box closes and allows the user to provide his body metrics data. However, if the user clicks on “Yes”, the user has thus confirmed he/she would skip and is redirected to the Home page (“/”) to continue using the application.

When a user provide their body metrics data, the *submitBodyMetrics()* functionis called when they click on “Save and continue”. This function uses an *axios POST* call to save the body metrics record to the database and redirects the user to the Home page (“/”).

Also, if the date is changed, the *handleDateChanged()* function is called. If no date is selected, it defaults to the current date. This date selection helps in keeping a consistent record of the daily body metrics.

When a previous body metrics record exists for a given date, any new form submission which has the same date will replace the already existing record in the database. If none exists, a new record is created for that date.

Report.jsx: This page displays the report of the user – both health and body metrics – over a given period. It also enables a user to visualize their calories expenditure and current BMI. First, the health data is fetched using *fetchHealthData()*  and the *systolicPressure*, *diastolicPressure*, *restingHeartRate* and *caloriesBurned* are set. Next the *fetchBodyMetrics()* method fetches the body metrics and sets the *chestCircumference*, *waistCircumference* and *hipCircumference*. *Chart, LineController, LineElement, PointElement, LinearScale, Title and CategoryScale* are imported from *chart.js.* The imported – besides *Chart* - are registered to *Chart.*

Chart elements are created for each data received from the database, each containing *chartData* and *chartOptions objects*. The *chartData* object contains the *labels* and *datasets,* and the *chartOptions* object contains scales for the *x* and *y* axis and suggested maximum and minimum scales. The chart elements are rendered as components on the *results.jsx* page displaying a graphical representation of the user’s health and body data over the given period of time.

RiskAssessment.jsx: This page displays the user’s health risk assessment. It uses *ChatGPT* API to get response using the user’s health data. It starts by calculating the user’s BMI. The names of all the exercises are also gotten from the *WorkoutData.* Then a method is called for each health data, which sends an *axios POST* call to the *ChatGPT* API. This call, takes as parameter *chartGtpEndpoint,* which is the *ChatGPT* URL. It also takes a message parameter which specifies the *role* and *content* for the request. The *role* is set to “user” and the *content* is set to make an inquiry of *ChatGPT*. The inquiry is to make a health risk assessment using the health data fetched above and to recommend exercises for the user. This is done for each of the health values. If no health data is available, the message “No health data available” is displayed. Otherwise, a table for the health data is displayed with the response for each health data from *ChatGPT* API. The BMI calculated above is also used in getting a response from *ChatGPT.* It is set to have a different *className* depending on the value of the BMI. If the BMI is less than 18.5, the user is underweight. Between 18.5 and 24.9, the user is of normal weight. Between 25 and 29.9, the user is overweight. And if the BMI is over 40, the user is obese. The text color varies with the BMI value due to the *className* given.

While the data to be displayed is still being fetched, a loading page is displayed which is the *Puff* variant of *LoadingIcons* from *react-loading-icons.*

NutritionTips.jsx: In this page, a user gets tips to help improve his nutrition. The page starts by calculating the BMI. Then an *axios POST* request is sent to *ChatGPT* API with a *content* which asks the API to respond with nutrition tips based on the user’s health data. The user’s fitness goals, fitness level, medical history and BMI are also included as parameters to the request to create a customized response for the user. If no health data is available, the message “No health data available” is displayed. Otherwise, the nutrition tip is displayed.

While the data to be displayed is still being fetched, a loading page is displayed which is the *Puff* variant of *LoadingIcons* from *react-loading-icons.*