Group 4

The Health and Fitness Software Architecture Document

Version 1.2

Revision History

Date	Version	Description	Author
16/11/2024	1.0	Introduction	Trương Thuận Kiệt
		Architectural Goals	Trương Thuận Kiệt
		Use case model	Nguyễn Huỳnh Minh Quang
		Architectural Constraints	Nguyễn Huỳnh Minh Quang
21/11/2024	1.1	Draw class diagram for all components	Vũ Thái Thiện
		Draw database	Vũ Thái Thiện
		Write description for component Model	Nguyễn Huỳnh Minh Quang
		Write description for component View	Trương Thuận Kiệt
		Write description for component Controller	Ngô Thanh Phương Dương
		Draw MVC model	Văn Diệp Bảo Duy
		Draw Domain View Diagram	Văn Diệp Bảo Duy
		Draw Domain Controller Diagram	Văn Diệp Bảo Duy
		Draw Domain Model Diagram	Văn Diệp Bảo Duy
4/12/2024	1.2	Implementation View	Trương Thuận Kiệt
		Deployment	Ngô Thanh Phương Dương
		Draw Deployment Diagram	Văn Diệp Bảo Duy
		Draw Implementation Diagram	Văn Diệp Bảo Duy

Table of Contents

Software Architecture Document

1. Introduction

This Software Architecture Document outlines the architecture for the **Health and Fitness Web Application**, which aims to provide users with tools and resources to manage their health and fitness effectively. The application includes features like personalized workout plans, nutritional guidance, activity tracking, and progress monitoring.

1.1 Purpose

- Define the architectural structure of Health and Fitness Web
- Ensure that the design aligns with functional and non-functional requirements
- Serve as a guide for developers, stakeholders and project managers throughout the development cycle

1.2 Scope

- **User Registration and Profile Management**: Users can create accounts, update profiles, and set fitness goals.
- **Workout and Nutrition Plans**: Personalized recommendations based on user preferences, goals, and activity levels.
- **Activity Tracking**: Integration with wearable devices and manual logging of workouts, steps, and calories.

1.3 Definitions, Acronyms, and Abbreviations

- **SAD:** Software Architecture Document
- **UI:** User Interface
- **API:** Application Programming Interface
- **NFR:** Non-Functional Requirements
- **DBMS:** Database Management System

1.4 References

- Vision document (PA02)
- Use-case specification (PA02)

1.5 Overview

- **Architectural Goals and Constraints**: Identifies key objectives and limitations shaping the system.
- **Use-Case Model**: Includes diagrams and scenarios describing the functional behavior of the system.
- **Logical View**: Details the components, their responsibilities, and their relationships. Includes diagrams like class and sequence diagrams.
- **Deployment**: Maps components to physical devices or systems.
- **Implementation View**: Provides folder structures and additional implementation details for system components.

2. Architectural Goals and Constraints

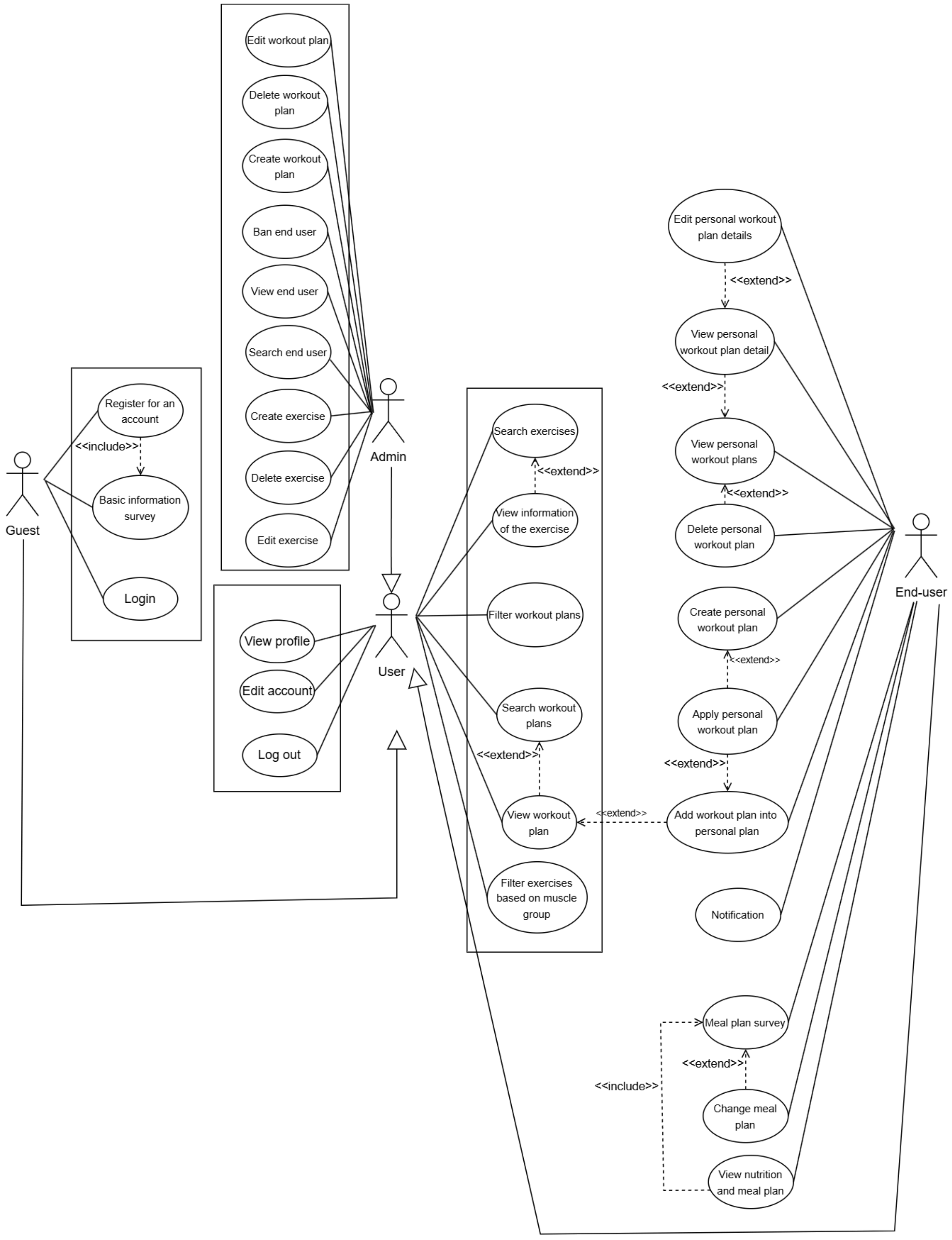
2.1 Goals

- Ensure security and privacy of user data, particularly sensitive information such as health metrics and personal details.
- Design for scalability to support a growing number of users and data.
- Provide high availability with minimal downtime to ensure uninterrupted service.
- Ensure the application is responsive and delivers a seamless user experience across devices (web and mobile).
- Promote reuse of components to reduce development time for future features or updates.

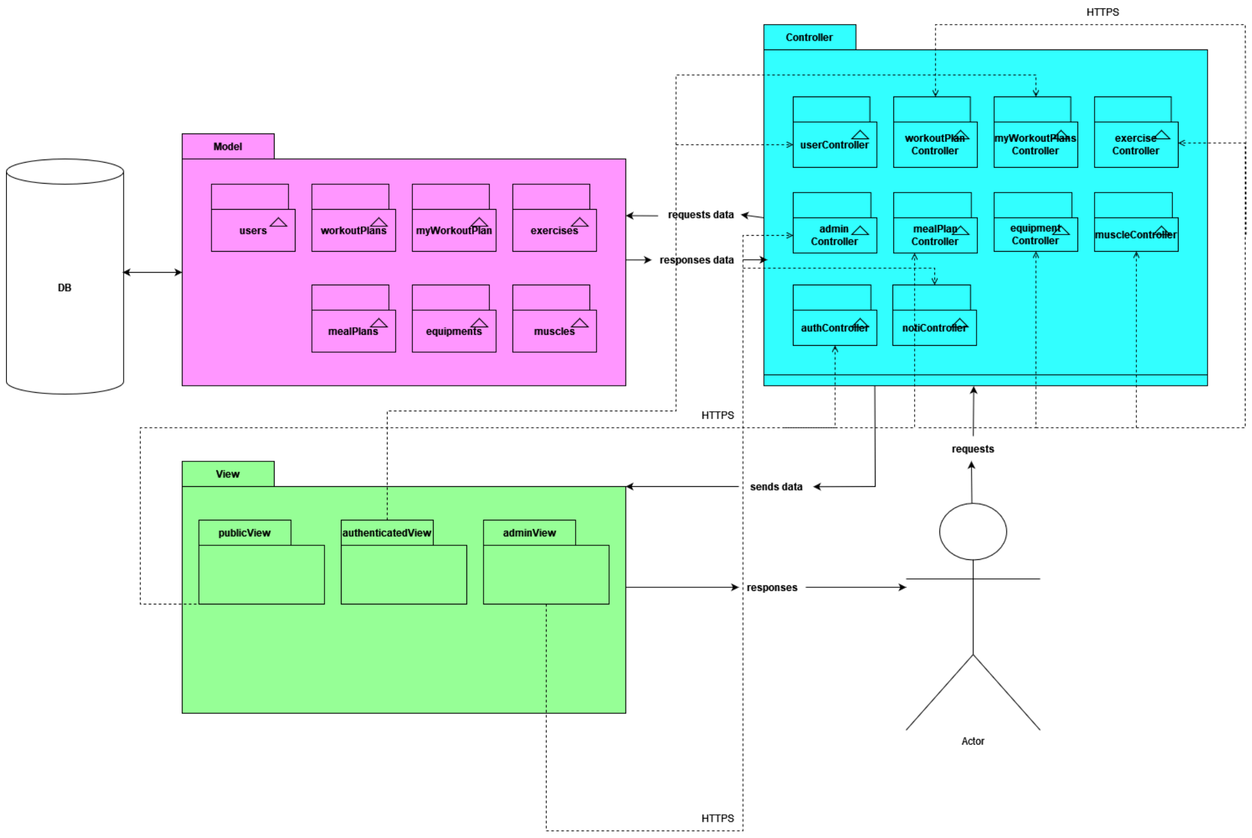
2.2 Constraints

- **Technology Stack:** ReactJS, Tailwind CSS, Node.js, Firebase.
- **Compatibility:** Ensure the system works on major browsers (Chrome, Firefox, Safari, Edge) and operating systems (Windows, macOS, iOS, Android).
- **Scalability:**
  - Ensure the app can handle increased user traffic as it grows, including a surge in concurrent users during peak times..
  - Employ scalable backend solutions such as cloud services (Firebase) and database sharding or replication for performance optimization.
- **Performance and Speed:**
  - Optimize load times to enhance user experience, especially on mobile devices.
  - Use techniques like lazy loading, caching, and a Content Delivery Network (CDN) for static assets.
- **Reliability:**
  - Maintain high availability through fault-tolerant design, such as microservices architecture and failover mechanisms.
  - Implement a robust monitoring system to detect and resolve issues proactively.
- **User-Centric Design:**
  - Offer a responsive and intuitive interface optimized for both desktop and mobile platforms.
  - Prioritize accessibility for a diverse user base.
- **Security:** Protect sensitive user data (e.g., password, user information) through encryption and secure storage practices.
- **Customizability:** Allow users to personalize their experience, including setting goals, tracking progress, and choosing features like diet plans or workout routines.
- **Integration with Wearables and APIs:** Enable integration with popular fitness trackers and third-party APIs for comprehensive health insights.
- **Modular Architecture:** Design with modularity to facilitate adding features without disrupting existing functionality.
- **Documentation:** Maintain comprehensive user and technical documentation as part of the system architecture

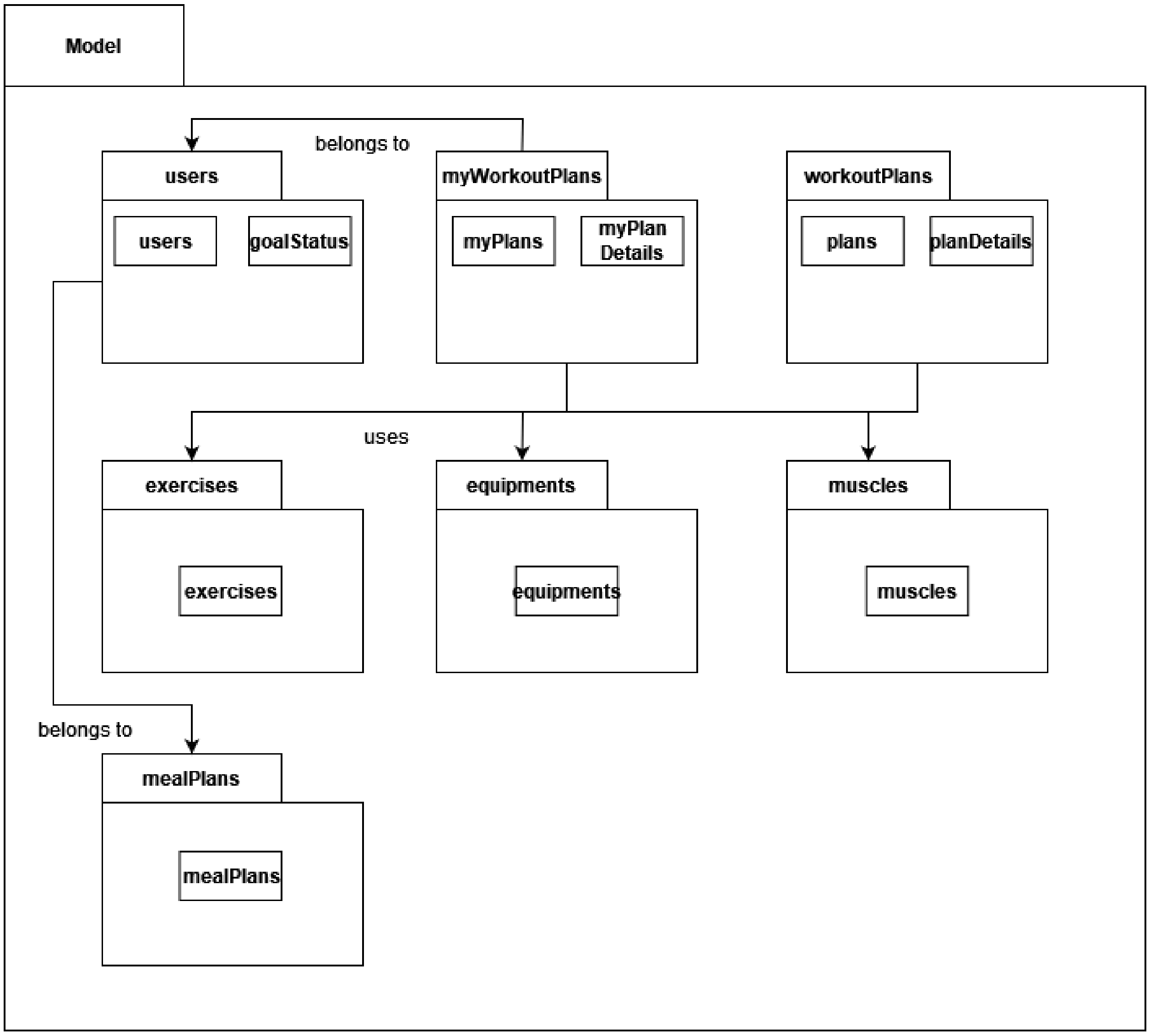
3. Use-Case Model



4. Logical View



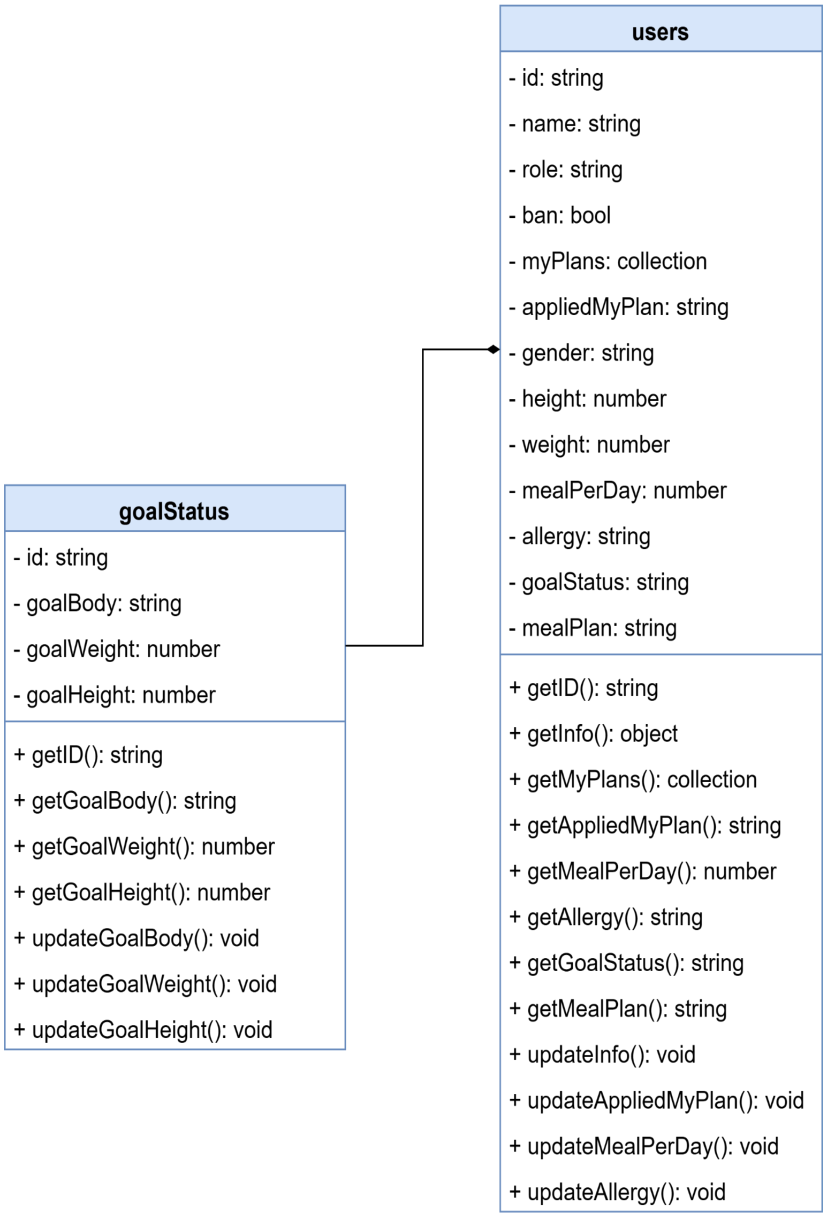
4.1 Component: Model



Model
belongs to
users
users
goalStatus
myWorkoutPlans
myPlans
myPlan Details
workoutPlans
plans
planDetails
uses
exercises
exercises
equipments
equipments
muscles
muscles
belongs to
mealPlans
mealPlans

4.1.1 Component: users

- Class Diagram:

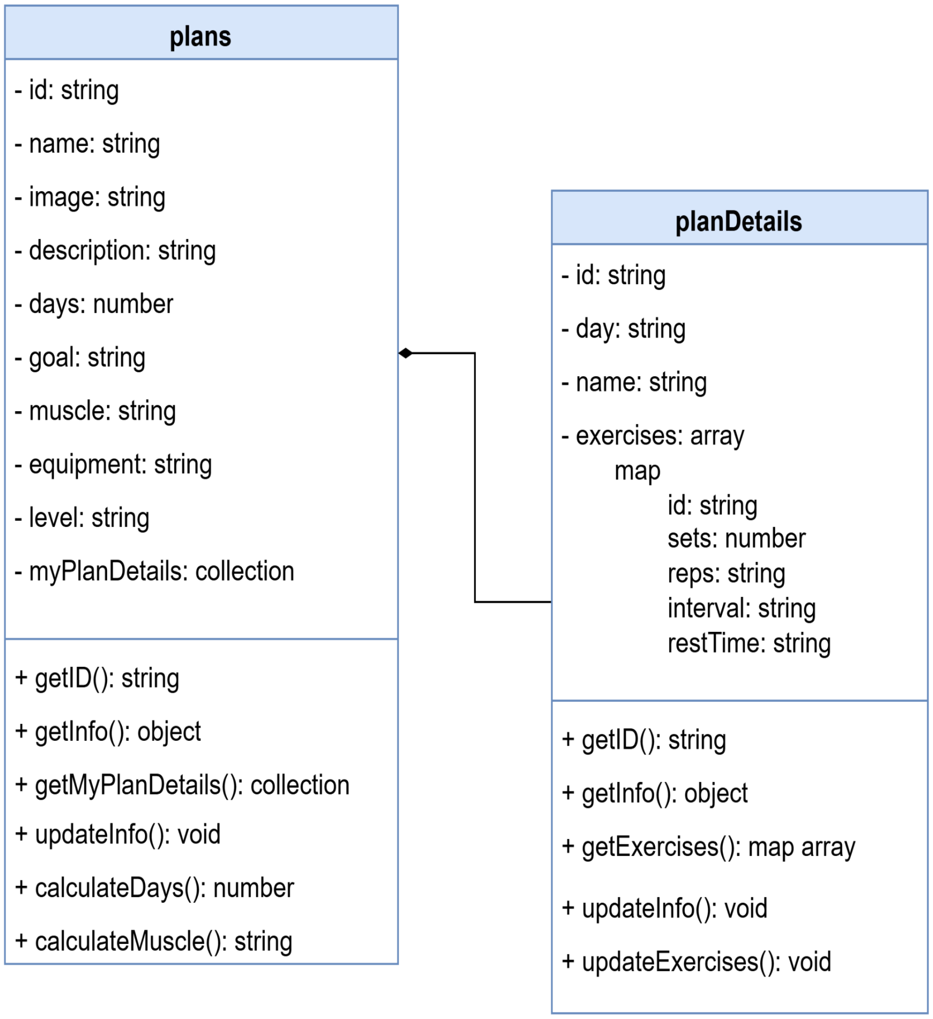


- Description:

Class/Struct	Attributes	Operations
Class:users	**id:** A unique identifier for the user. **name:** The name of the user. **role:** The role of the user. **ban:** Indicates if the user is banned (true for banned, false otherwise). **myPlans:** A collection of plans created or joined by the user. **appliedMyPlan:** The ID of the plan currently applied by the user. **gender:** The gender of the user**.** **height:** The height of the user. **weight:** The weight of the user. **mealPerDay:** The number of meals the user has per day. **allergy:** A list of food allergies. **goalStatus:** The current status of the user's goal. **mealPlan:** The name or ID of the current meal plan.	**getID():** Returns the user's unique ID. **getInfo():** Returns all user information as an object. **getMyPlans():** Retrieves the collection of personal plans. **getAppliedMyPlan():** Returns the ID of the currently applied plan. **getMealPerDay():** Returns the number of meals the user eats daily. **getAllergy():** retrieves the user's allergy information. **getGoalStatus():** Returns the user's current goal status. **getMealPlan():** Returns the current meal plan. **updateInfo():** Updates the user's personal information **updateAppliedMyPlan():** Updates the currently applied plan for the user. **updateMealPerDay():** Updates the number of meals the user eats per day. **updateAllergy():** Updates the user's allergy information.
Class: goalStatus	**id:** A unique identifier for the goal status. **goalBody:** A description of the user's goal. **goalWeight:** The target weight the user wants to achieve. **goalHeight:** The target height the user wants.	**getID():** Returns the ID of the goal status. **getGoalBody():** Returns the description of the goal. **getGoalWeight():** Retrieves the target weight. **getGoalHeight():** Retrieves the target height. **updateGoalBody():** Updates the goal description. **updateGoalWeight():** Updates the target weight. **updateGoalHeight():** Updates the target height.

4.1.2 Component: workoutPlans

- Class diagram:



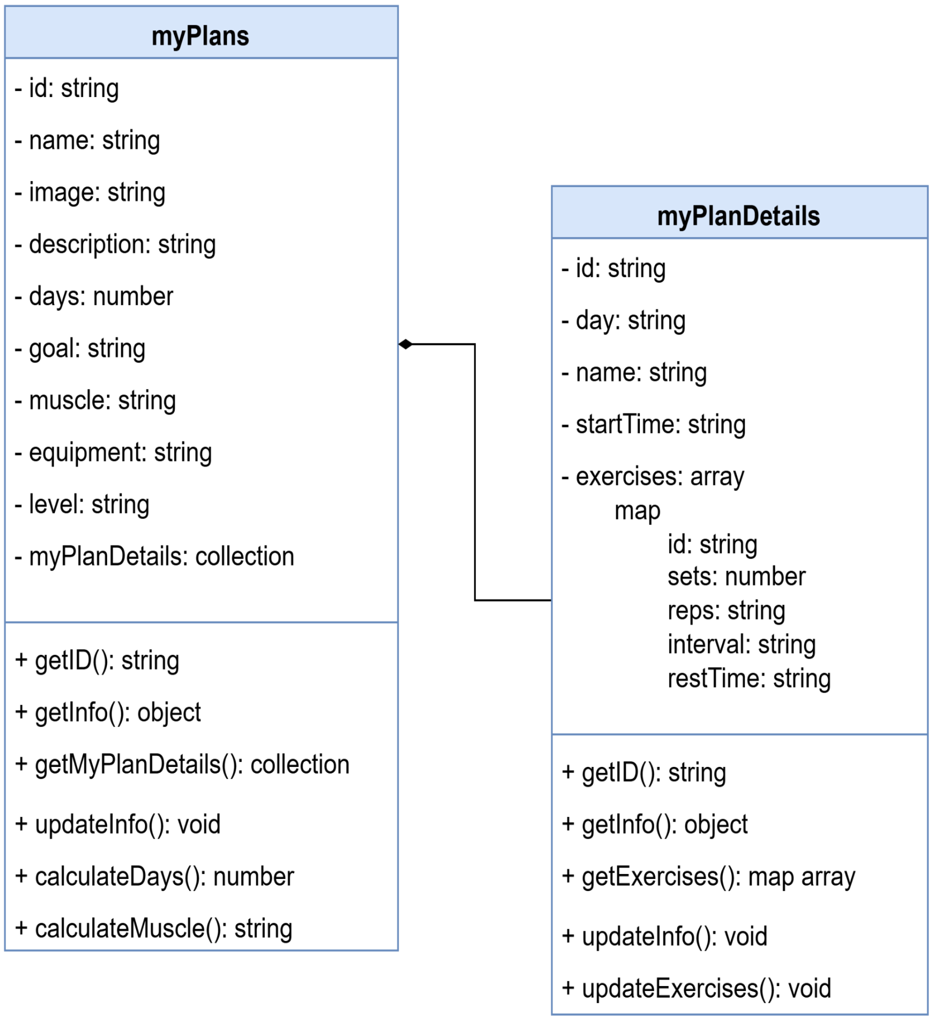
- **Description**

Class/Struct	Attributes	Operations
Class: plans	**id:** A unique identifier for the workout plan. **name:** The name of the workout plan. **image:** An image associated with the workout plan. **description:** A detailed description of the workout plan. **days:** The number of days in the workout plan.	**getID():** Returns the unique identifier of the workout plan. **getInfo():** Returns an object containing various information about the workout plan, such as its name, description, goal, muscle group, and equipment requirements. **getMyPlanDetails():** Returns the collection of planDetails objects associated with the workout plan.

	goal: The overall goal of the workout plan (e.g., muscle gain, weight loss, endurance). **muscle:** The primary muscle group targeted by the workout plan. **equipment:** The required equipment for the workout plan**.** **level:** The difficulty level of the workout plan (e.g., beginner, intermediate, advanced). **myPlanDetails:** A collection of plan details objects, each representing a day's workout within the plan.	**updateInfo():** Updates the information of the workout plan, such as its name, description, goal, muscle group, or equipment requirements. **calculateDays():** Calculates the number of days in the workout plan based on the myPlanDetails collection. **calculateMuscle():** Calculates the primary muscle group targeted by the workout plan based on the exercises in the myPlanDetails collection.
Class: planDetails	**id:** A unique identifier for the workout plan detail. **day:** The day number within the workout plan. **name:** The name or title of the workout for that day. **exercises:** An array of exercises, where each exercise is represented as a map with the following properties: **id:** A unique identifier for the exercise. **sets:** The number of sets to perform for the exercise. **reps:** The number of repetitions per set. **interval:** The rest interval between sets. **restTime:** The total rest time for the exercise.	**getID():** Returns the unique identifier of the workout plan detail. **getInfo():** Returns an object containing information about the workout plan detail, such as the day number, name, and a list of exercises. **getExercises():** Returns the array of exercises for the workout plan detail. **updateInfo():** Updates the information of the workout plan detail, such as the day number or name. **updateExercises():** Updates the list of exercises for the workout plan detail, including adding, removing, or modifying exercises.

4.1.3 Component: myWorkoutPlans

- Class diagram:



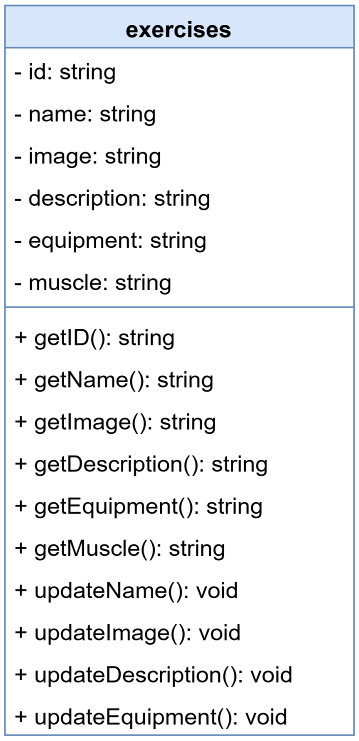
-

Class/Struct	Attributes	Operations
Class: myPlans	**id**: Unique identifier of the plan. **name**: Name of the plan. **image**: Image associated with the plan. **description**: Description of the plan. **days**: Number of days in the plan. **goal**: Overall goal of the plan. **muscle**: Target muscle group(s). **equipment**: Required equipment for the plan.	**getID()**: Retrieves the unique identifier of the plan. **getInfo()**: Retrieves information about the plan. **getMyPlanDetails()**: Retrieves the collection of myPlanDetails objects associated with the plan. **updateInfo()**: Updates the information of the plan. **calculateDays()**: Calculates the total number of days in the plan.

	level: Difficulty level of the plan. **myPlanDetails**: Collection of myPlanDetails objects, representing the individual workout sessions within the plan.	**calculateMuscle()**: Calculates the primary muscle group(s) targeted by the plan.
Class: myPlanDetails	**id**: Unique identifier of the workout session. **day**: Day of the week when the workout is scheduled. **name**: Name of the workout session. **startTime**: Start time of the workout session. **exercises**: Array of exercises in the workout session. - **sets:** Number of sets. - **reps:** Number of repetitions per set. - **interval:** Time interval between sets. - **restTime:** Rest time after the last set.	**getID():** Retrieves the unique identifier of the workout session. **getInfo()**: Retrieves information about the workout session. **getExercises()**: Retrieves the map of exercises in the workout session. **updateInfo()**: Updates the information of the workout session. **updateExercises()**: Updates the exercises in the workout session.

4.1.4 Component: exercises

- Class Diagram:

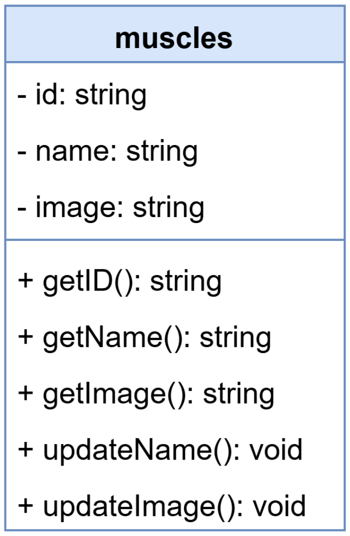


- Description:

Class/Struct	Attributes	Operations
exercises	**id**: A unique identifier for the exercise, likely represented as a string. **name**: The name of the exercise. **image**: A string representing the path or URL to an image associated with the exercise. **description**: A textual description of the exercise. **equipment**: A string indicating the equipment required for the exercise, if any. **muscle**: A string specifying the muscle group(s) targeted by the exercise.	**getID()**: Returns the **id** attribute of the exercise. **getName()**: Returns the **name** attribute of the exercise. **getImage()**: Returns the **image** attribute of the exercise. **getDescription()**: Returns the **description** attribute of the exercise. **getEquipment()**: Returns the **equipment** attribute of the exercise. **getMuscle()**: Returns the **muscle** attribute of the exercise. **updateName()**: Updates the **name** attribute of the exercise. **updateImage()**: Updates the **image** attribute of the exercise. **updateDescription()**: Updates the **description** attribute of the exercise. **updateEquipment()**: Updates the **equipment** attribute of the exercise.

4.1.5 Component: muscles

- Class Diagram:

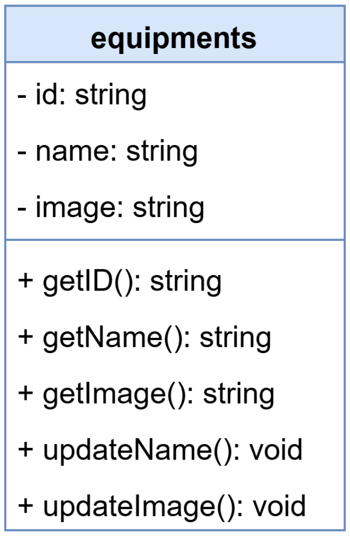


- Description

Class/Struct	Attributes	Operations
muscles	**id**: A unique identifier for the muscle. **name:** The name of the muscle. **image:** A path or URL to an image representing the muscle.	**getID():** Returns the unique identifier of the muscle. **getName():** Returns the name of the muscle. **getImage():** Returns the path or URL to the muscle image. **updateName():** Updates the name of the muscle. **updateImage():** Updates the path or URL to the muscle image.

4.1.6 Component: equipments

- Class diagram:



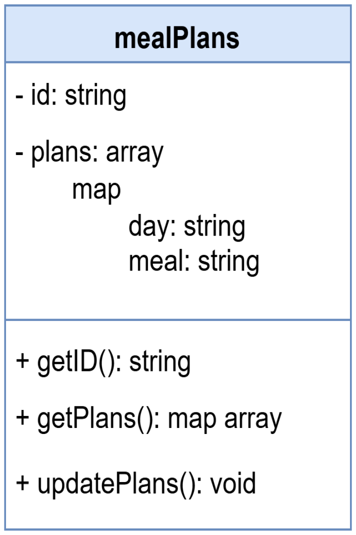
- Description

Class/Struct	Attributes	Operations
equipments	**id:** A unique string identifier for the equipment. **name:** A string representing the name of the equipment. **image:** A string representing the path or URL of the image associated with the equipment.	**getID():** Returns the unique identifier of the equipment. **getName()::** Returns the name of the equipment. **getImage():** Returns the path or URL of the image associated with the equipment. **updateName():** Updates the name of the equipment.

		updateImage(): Updates the path or URL of the image associated with the equipment.

4.1.7 Component: mealPlans

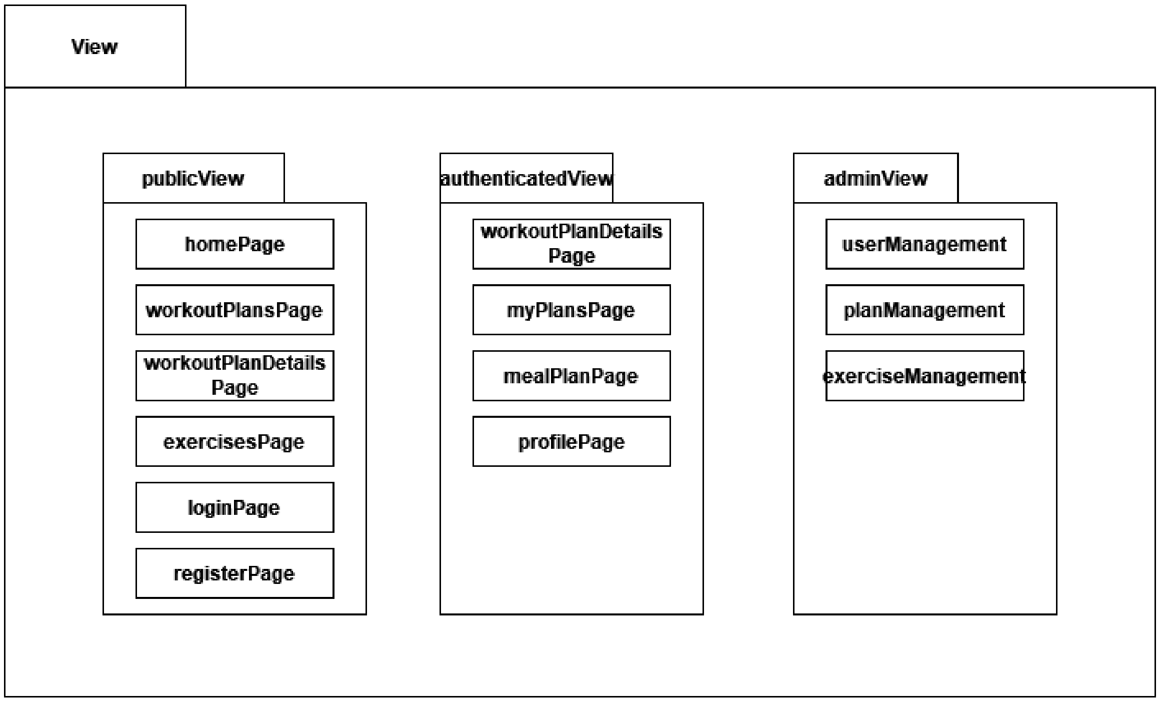
- Class diagram:



- Description

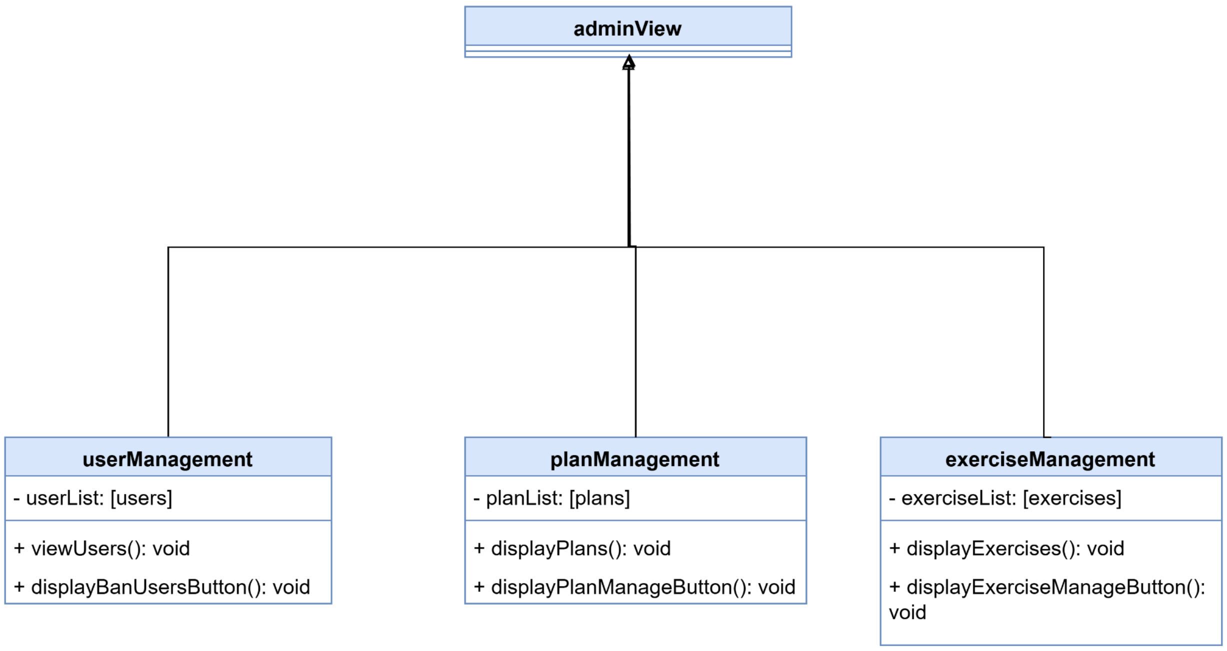
Class/Struct	Attributes	Operations
mealPlans	**id**: A unique identifier for the meal plan. **plans**: An array of meal plan entries, each represented as a map with two key-value pairs: - **day**: A string representing the day of the week. - **meal**: A string representing the meal for that day.	**getID()**: Returns the unique identifier of the meal plan. **getPlans()**: Returns the array of meal plan entries. **updatePlans()**: Updates the meal plan entries.

4.2 Component: View



4.2.1 Component: adminView

- Class diagram:



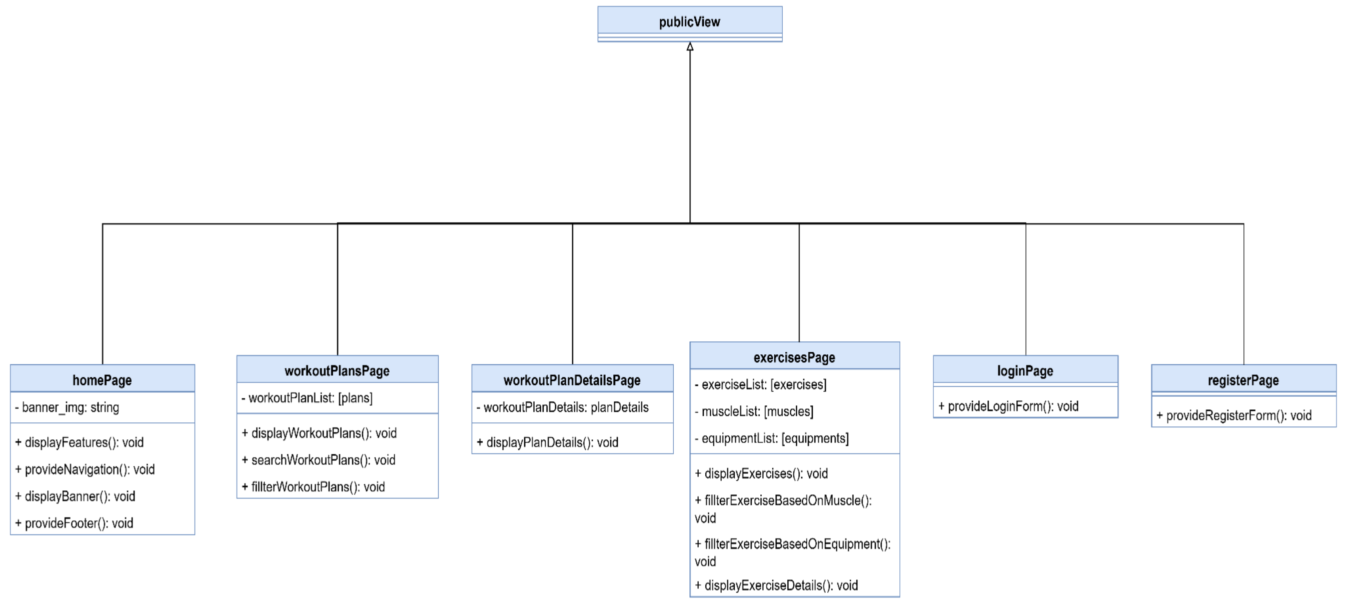
adminView
userManagement
- userList: [users]
+ viewUsers(): void
+ displayBanUsersButton(): void
planManagement
- planList: [plans]
+ displayPlans(): void
+ displayPlanManageButton(): void
exerciseManagement
- exerciseList: [exercises]
+ displayExercises(): void
+ displayExerciseManageButton(): void

- Description:

Class/Struct	Attributes	Operations
Class: adminView		
Class: userManagement	**userList:** A list of user objects.	**viewUsers()**: Displays a list of users. **displayBanUserButton()**: Display ban user button.
Class: planManagement	**planList**: A list of plan objects.	**displayPlans()**: Displays a list of plans. **displayPlanManageButton(): void()**: Displays add/delete/edit plan button.
Class: exerciseManagement	**exerciseList**: A list of exercise objects.	**displayExercises()**: Displays a list of exercises. **displayExerciseManageButton()**: Displays add/delete/edit exercise button.

4.2.2 Component: publicView

- Class diagram

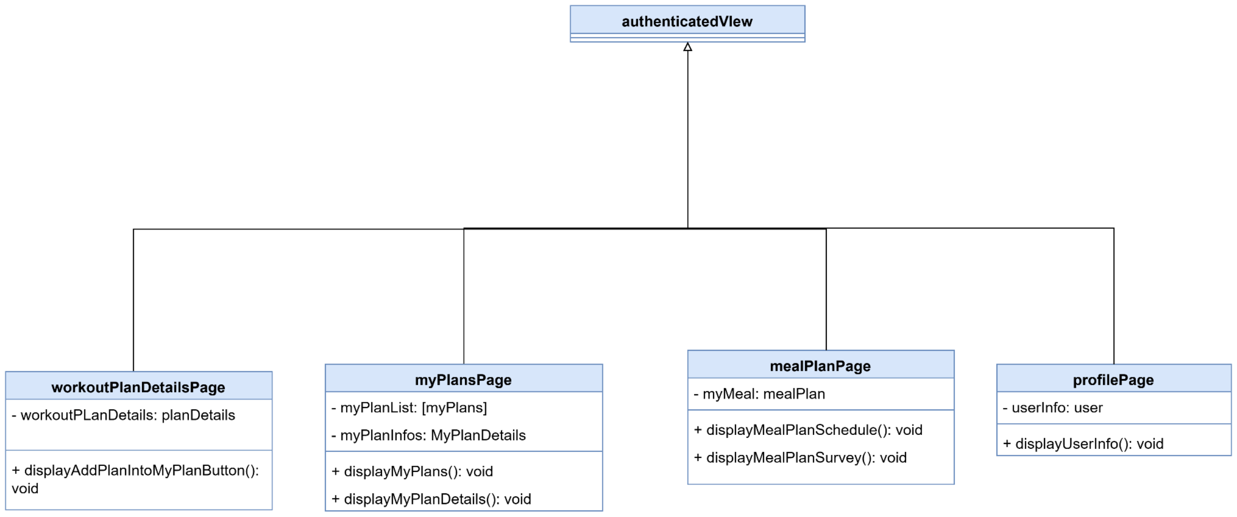

publicView
homePage
- banner_img: string
+ displayFeatures(): void
+ provideNavigation(): void
+ displayBanner(): void
+ provideFooter(): void
workoutPlansPage
- workoutPlanList: [plans]
+ displayWorkoutPlans(): void
+ searchWorkoutPlans(): void
+ fillterWorkoutPlans(): void
workoutPlanDetailsPage
- workoutPlanDetails: planDetails
+ displayPlanDetails(): void
exercisesPage
- exerciseList: [exercises]
- muscleList: [muscles]
- equipmentList: [equipments]
+ displayExercises(): void
+ fillterExerciseBasedOnMuscle(): void
+ fillterExerciseBasedOnEquipment(): void
+ displayExerciseDetails(): void
loginPage
+ provideLoginForm(): void
registerPage
+ provideRegisterForm(): void

- Description

Class/Struct	Attributes	Operations
Class: publicView		
Class: homePage	banner_img: Banner image of homepage	**displayFeatures():** Displays features of the home page like login, register, … **provideNavigation()**: Navigate users when click to each section
Class: workoutPlansPage	workoutPlanList: List of workout plans	**displayWorkoutPlans():** Renders the list of workout plans. **searchWorkoutPlans**(): Filters the workout plans based on search criteria. **filterWorkoutPlans**(): Filters the workout plans based on specific criteria (e.g., difficulty, duration, muscle group).
Class: workoutPlanDetailsPage	workoutPlanDetailsPage: Details of each workout plan	**displayPlanDetails**(): Renders the workout plan details, including exercises, sets, reps, and rest times.
Class: exercisesPage	exerciseList: List of exercises muscleList: List of muscles equipmentList: List of equipment that needed for exercise	**displayExercises**(): Renders the list of exercises. **filterExerciseBasedOnMuscle**(): Filters the exercises based on muscle group. **filterExerciseBasedOnEquipment**(): Filters the exercises based on equipment type. **displayExerciseDetails**(): Renders the details of a specific exercise, including instructions, images, and videos.
Class: loginPage		**provideLoginForm**(): Renders the login form.
Class: registerPage		**provideRegisterForm**(): Renders the registration form.

4.2.3 Component: authenticatedView

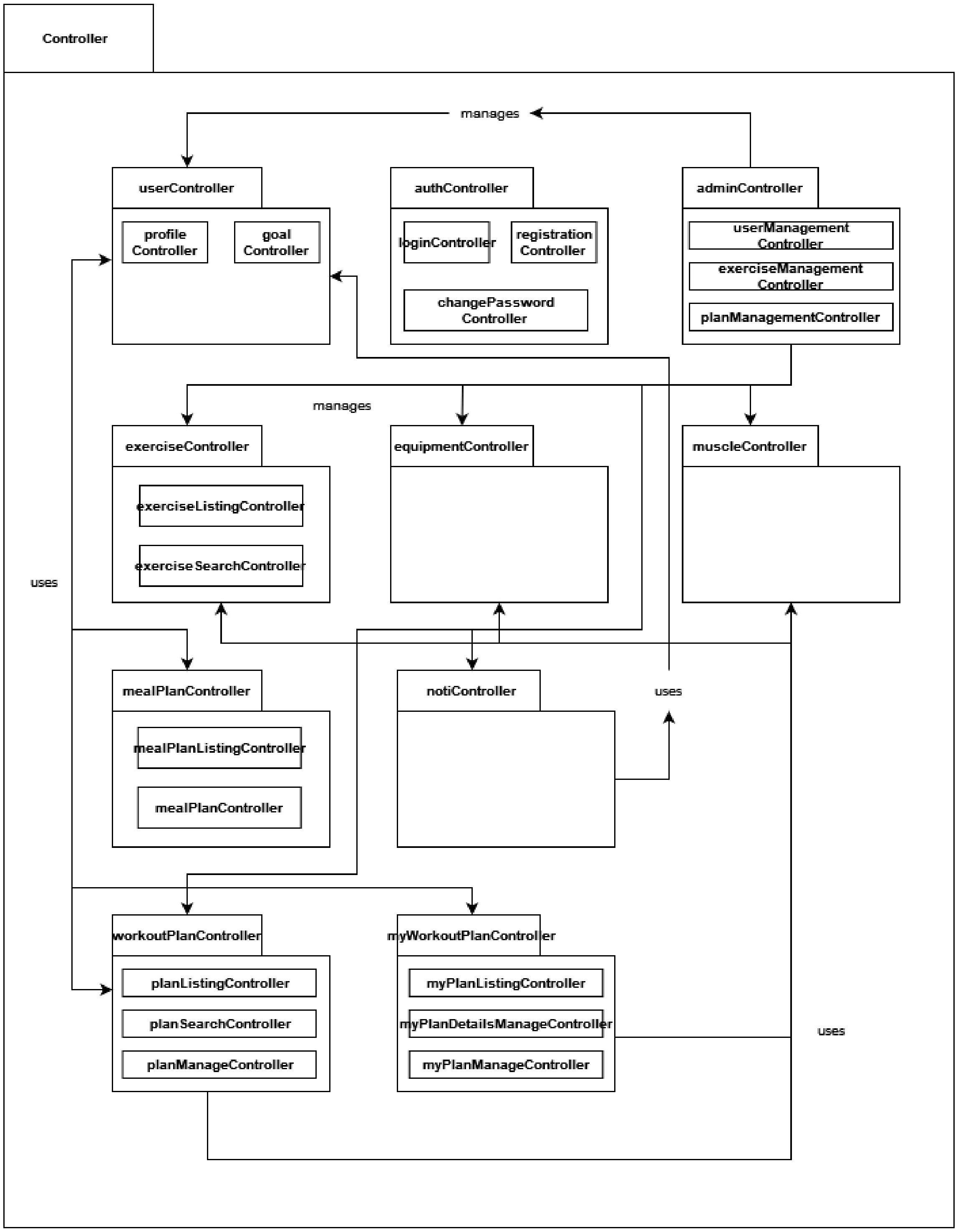
- Class diagram:


authenticatedVIew
workoutPlanDetailsPage
- workoutPLanDetails: planDetails
+ displayAddPlanIntoMyPlanButton(): void
myPlansPage
- myPlanList: [myPlans]
- myPlanInfos: MyPlanDetails
+ displayMyPlans(): void
+ displayMyPlanDetails(): void
mealPlanPage
- myMeal: mealPlan
+ displayMealPlanSchedule(): void
+ displayMealPlanSurvey(): void
profilePage
- userInfo: user
+ displayUserInfo(): void

- Description:

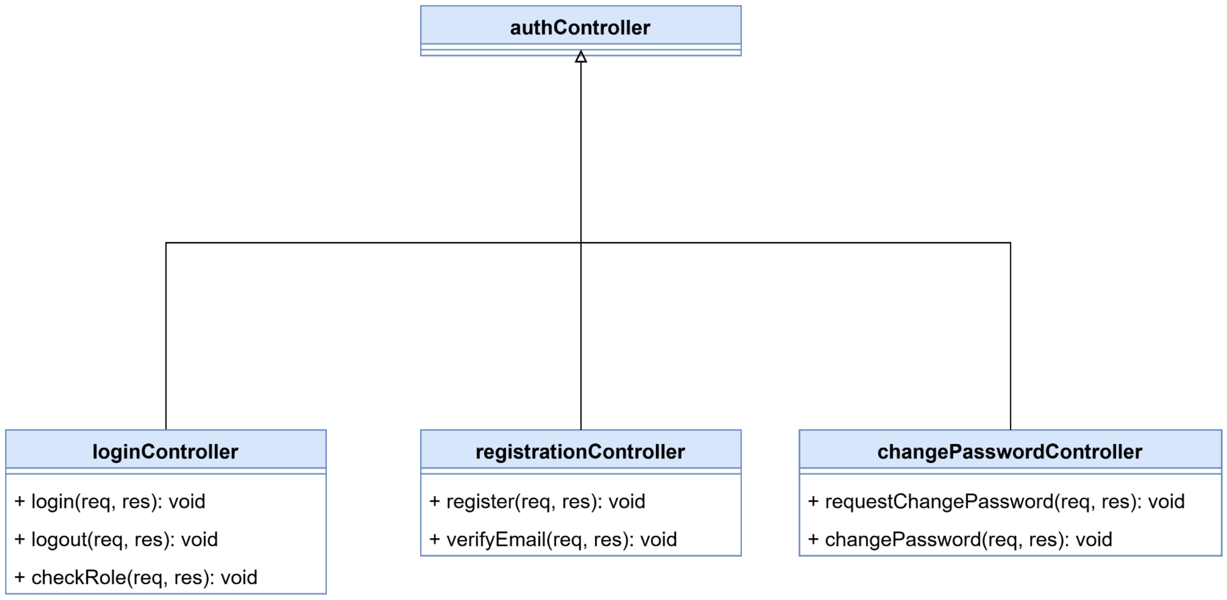
Class/Struct	Attributes	Operations
Class: adminView		
Class: workoutPlanDetailPage	**workoutPlanDetails:** Details of workout plan	**displayAddPlanIntoMyPlanButton()**: A button for end users to press to add plan.
Class: myPlanPage	**myPlanList**: A list of plan objects. **myPlanInfos**: Information of each plan	**displayMyPlans()**: Displays a list of plans. **displayMyPlanDetails():** Displays details of each plan
Class: mealPlanPage	**myMeal**: A list of meal objects.	**displayMealPlanSchedule()**: Displays a schedule of diet. **displayMealPlanSurvey()**: Display a survey of meal plans
Class: profilePage	**userInfo:** Information of each end user	**displayUserInfo():** Displays information of each user

4.3 Component: Controller



4.3.1 Component: authController

- Class diagram:



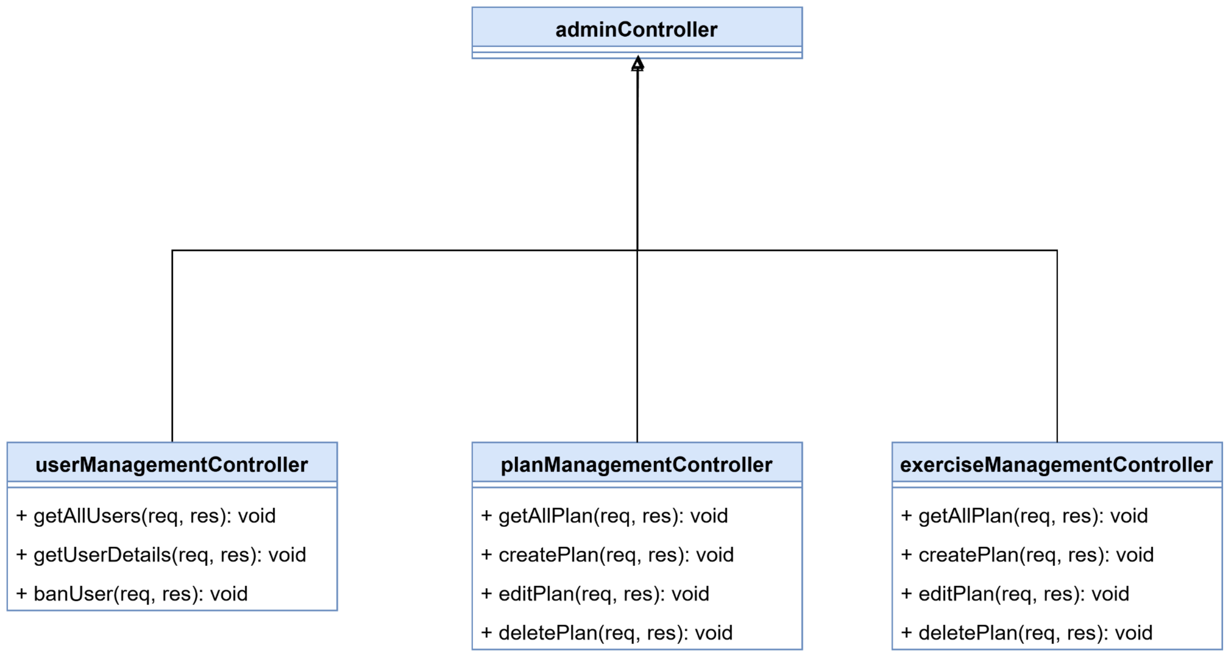
authController
loginController
+ login(req, res): void
+ logout(req, res): void
+ checkRole(req, res): void
registrationController
+ register(req, res): void
+ verifyEmail(req, res): void
changePasswordController
+ requestChangePassword(req, res): void
+ changePassword(req, res): void

- Description

Class/Struct	Attributes	Operations
Class: loginController		**login(req, res):** Authenticates user credentials, generates authentication tokens, and sets up user sessions. **logout(req, res):** Invalidates user sessions and clears authentication tokens. **checkRole(req, res):** Verifies user roles and permissions for authorized access to resources.
Class: registrationController		**register(req, res):** Validates user input, creates new user accounts, and sends verification emails or other confirmation mechanisms. **verifyEmail(req, res):** Verifies email addresses and activates user accounts.
Class: changePasswordController		**requestChangePassword(req, res):** Initiates the password change process, possibly sending a reset link or code to the user. **changePassword(req, res):** Validates the reset link or code, verifies the new password strength, and updates the user's password.

4.3.2 Component: adminController

- Class diagram:



adminController
userManagementController
+ getAllUsers(req, res): void
+ getUserDetails(req, res): void
+ banUser(req, res): void
planManagementController
+ getAllPlan(req, res): void
+ createPlan(req, res): void
+ editPlan(req, res): void
+ deletePlan(req, res): void
exerciseManagementController
+ getAllPlan(req, res): void
+ createPlan(req, res): void
+ editPlan(req, res): void
+ deletePlan(req, res): void

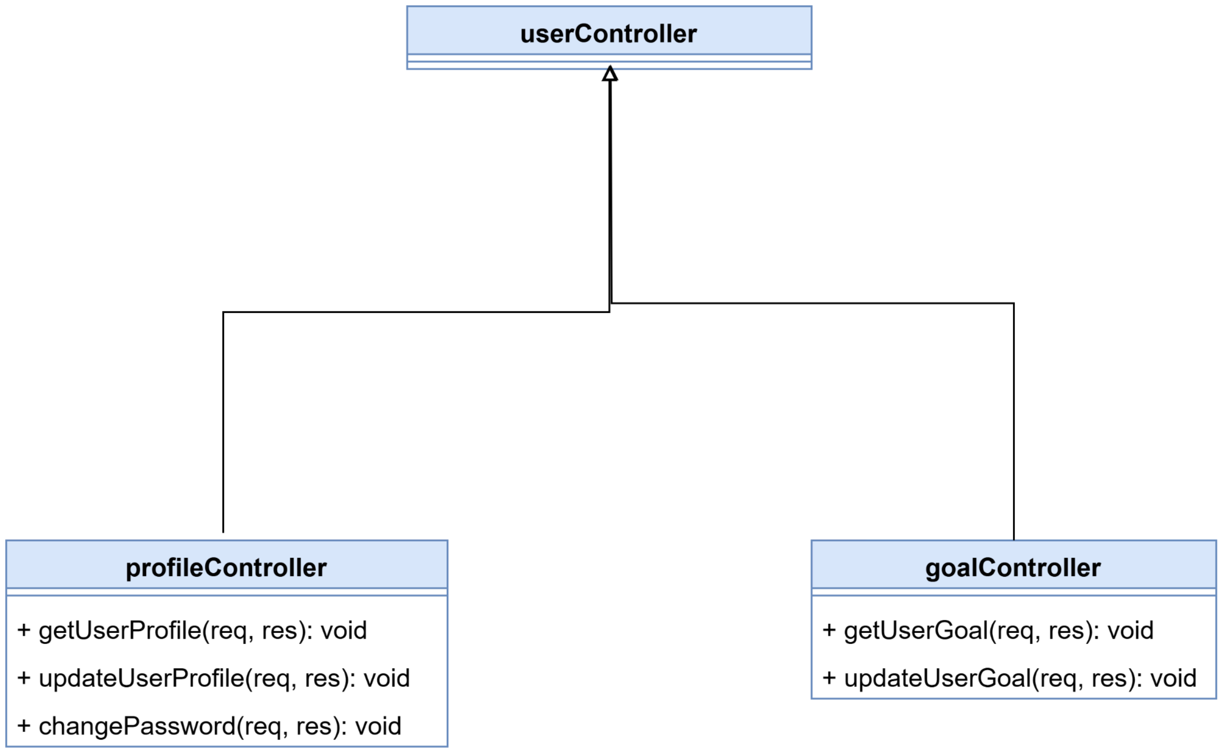
- **Description**

Class/Struct	Attributes	Operations
Class: userManagementController		**getAllUsers(req, res):** Retrieves a list of all users from the database. **getUserDetails(req, res):** Retrieves detailed information about a specific user. **banUser(req, res):** Bans a specific user by updating their status in the database.
Class: planManagementController		**getAllPlans(req, res):** Retrieves a list of all plans from the database. **createPlan(req, res):** Creates a new plan based on the provided plan data. **editPlan(req, res):** Updates an existing plan based on the provided plan ID and updated plan data. **deletePlan(req, res):** Deletes an existing plan based on the provided plan ID.
Class: exerciseManagementController		**getAllPlans(req, res):** Retrieves a list of all exercises from the underlying data store.

		createPlan(req, res): Creates a new exercise based on the provided exercise data. **editPlan(req, res):** Updates an existing exercise based on the provided exercise ID and updated exercise data. **deletePlan(req, res):** Deletes an existing exercise based on the provided exercise ID.

4.3.3 Component: userController

- Class diagram:


userController
profileController
+ getUserProfile(req, res): void
+ updateUserProfile(req, res): void
+ changePassword(req, res): void
goalController
+ getUserGoal(req, res): void
+ updateUserGoal(req, res): void

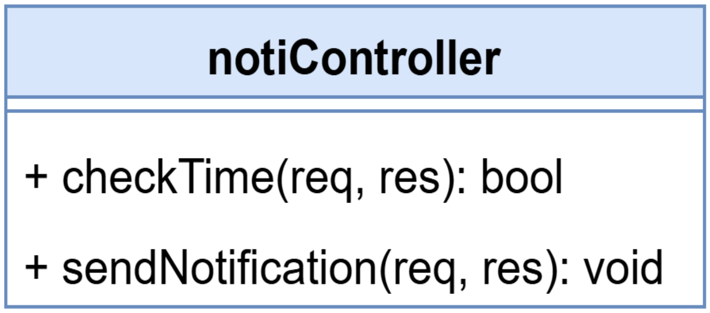
- Description:

Class/Struct	Attributes	Operations
Class: userController		
Class: profileController		**getUserProfile(req, res)**: Retrieves the profile information of a specific user. **updateUserProfile(req, res):**

		Updates the profile information of a specific user. **changePassword(req, res):** Allows a user to change their password.
Class: goalController		**getUserGoal(req, res)**: Retrieves the goal information of a specific user. **updateUserGoal(req, res)**: Updates the goal information of a specific user.

4.3.4 Component: notiController

- Class diagram:

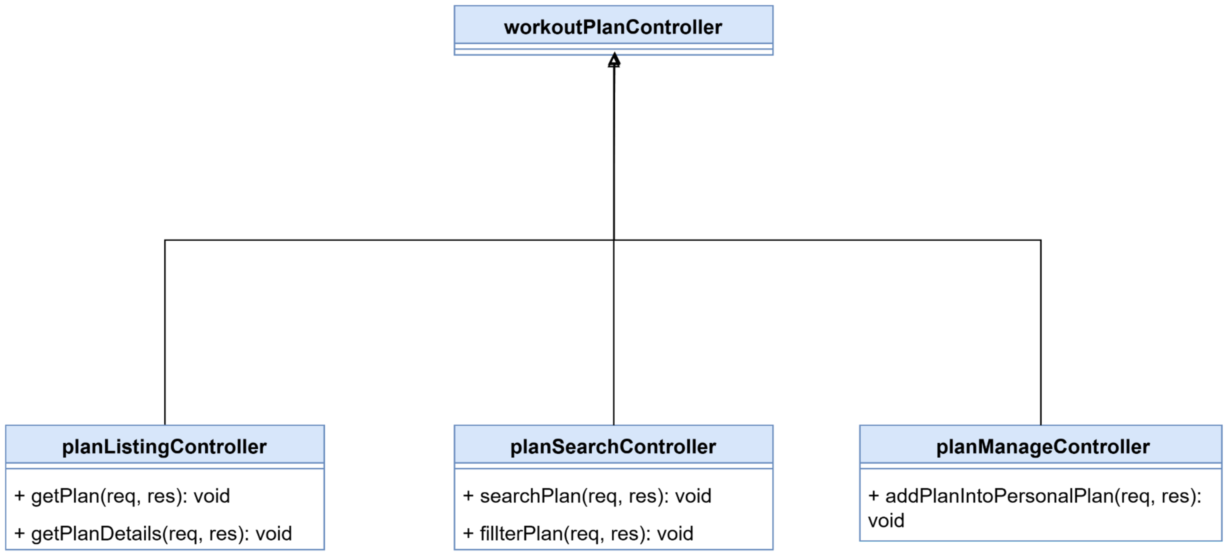

notiController
+ checkTime(req, res): bool
+ sendNotification(req, res): void

- Description:

Class/Struct	Attributes	Operations
Class: notiController		**checkTime(req, res)**: Check the current time against predefined notification schedules. **sendNotification(req, res):** Handles the actual sending of notifications to users

4.3.5 Component: workoutPlanController

- Class diagram:

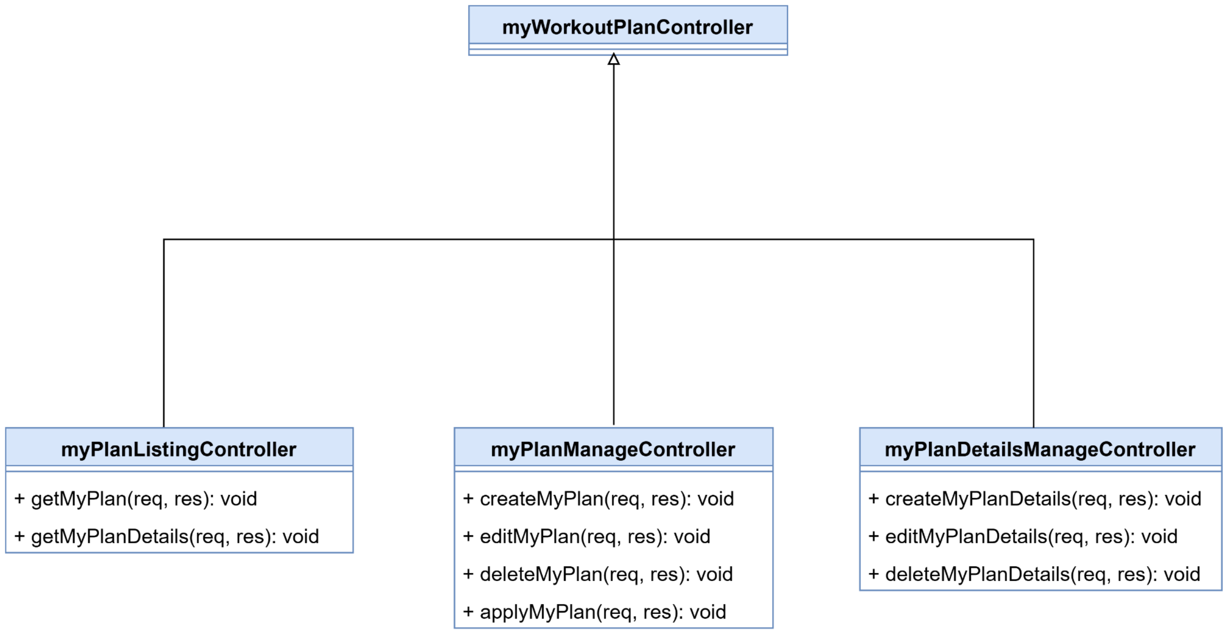

workoutPlanController
planListingController
+ getPlan(req, res): void
+ getPlanDetails(req, res): void
planSearchController
+ searchPlan(req, res): void
+ fillterPlan(req, res): void
planManageController
+ addPlanIntoPersonalPlan(req, res): void

- Description:

Class/Struct	Attributes	Operations
Class: workoutPlanController		
Class: planListingController		**getPlan(req, res):** Retrieves a list of workout plans. **getPlanDetails(req, res):** Retrieves detailed information about a specific workout plan.
Class: planSearchController		**searchPlan(req, res):** Searches for workout plans based on specific criteria. **filterPlan(req, res):** Filters workout plans based on various criteria.
Class: planManageController		**addPlanIntoPersonalPlan(req, res):** Adds a workout plan to a user's personalized plan.

4.3.6 Component: myWorkoutPlanController

- Class Diagram



myWorkoutPlanController
myPlanListingController
+ getMyPlan(req, res): void
+ getMyPlanDetails(req, res): void
myPlanManageController
+ createMyPlan(req, res): void
+ editMyPlan(req, res): void
+ deleteMyPlan(req, res): void
+ applyMyPlan(req, res): void
myPlanDetailsManageController
+ createMyPlanDetails(req, res): void
+ editMyPlanDetails(req, res): void
+ deleteMyPlanDetails(req, res): void

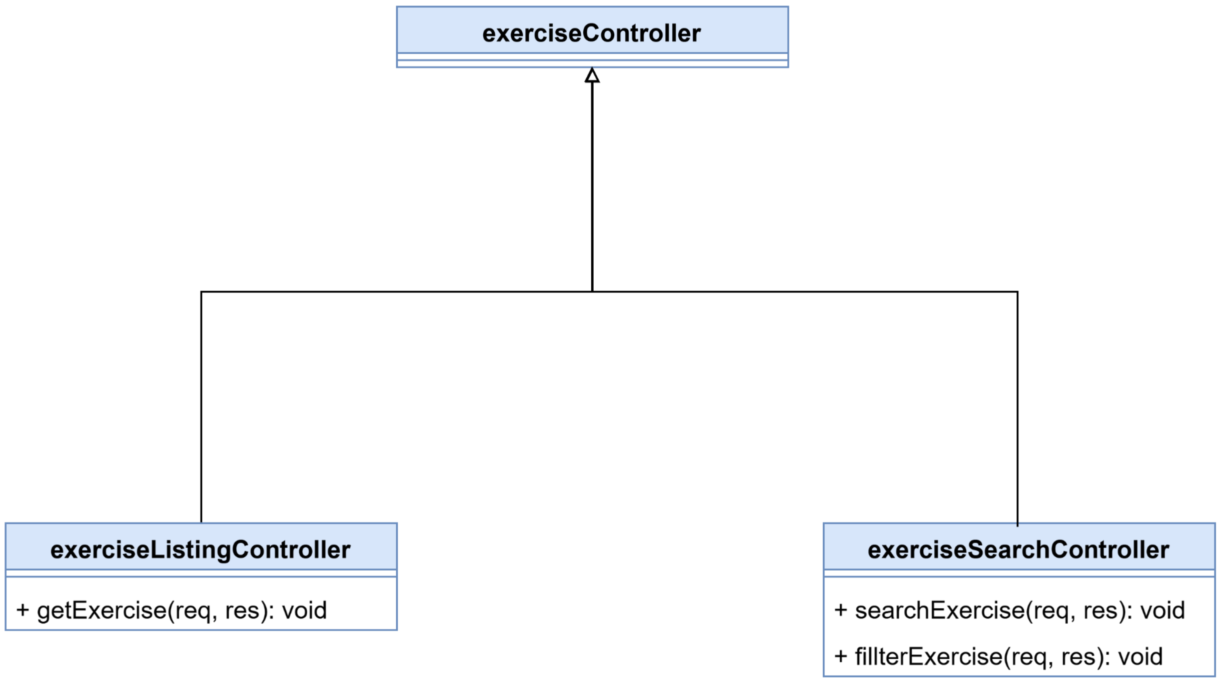
- **Description**

Class/Struct	Attributes	Operations
Class: myWorkoutPlanController		**getMyPlan(req, res):** Retrieves an existing workout plan. **getMyPlanDetails(req, res)**: Retrieves details of a specific workout plan. **createMyPlan(req, res):** Creates a new workout plan. **editMyPlan(req, res):** Edits an existing workout plan. **deleteMyPlan(req, res):** Deletes an existing workout plan. **applyMyPlan(req, res):** Applies a workout plan to a user's routine.
Class: myPlanListingController		**getMyPlan(req, res):** Retrieves a list of workout plans. **getMyPlanDetails(req, res):** Retrieves details of a specific workout plan from the list.
Class: myPlanManageController		**createMyPlan(req, res):** Creates a new workout plan. **editMyPlan(req, res):** Edits an existing workout plan. **deleteMyPlan(req, res):** Deletes an existing workout plan.

Class: myPlanDetailsManageController		**createMyPlanDetails(req, res):** Creates details for a new workout plan. **editMyPlanDetails(req, res):** Edits the details of an existing workout plan. **deleteMyPlanDetails(req, res):** Deletes the details of an existing workout plan.

4.3.7 Component: exerciseController

- Class diagram:


exerciseController
exerciseListingController
+ getExercise(req, res): void
exerciseSearchController
+ searchExercise(req, res): void
+ fillterExercise(req, res): void

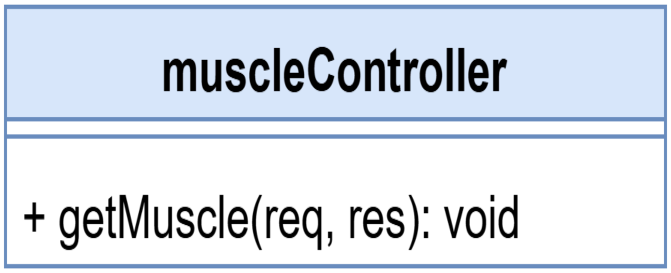
- **Description**

Class/Struct	Attributes	Operations
Class: exerciseListingController		**getExercise(req, res):** Fetches exercise data based on request parameters.
Class: exerciseSearchController		**searchExercise(req, res):** Searches for exercise data based on search criteria. **filterExercise(req, res):** Filters

		exercise data based on specific filters.

4.3.8 Component: muscleController

- Class diagram

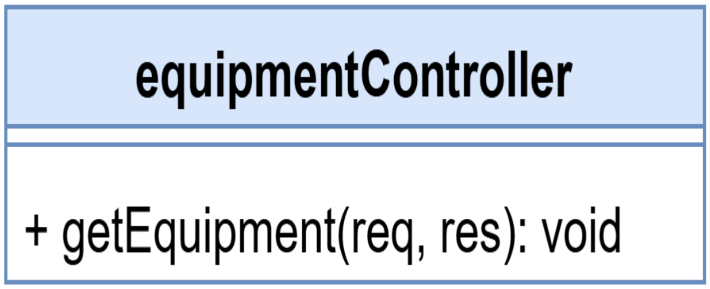


- **Description**

Class/Struct	Attributes	Operations
Class: muscleController		**getMuscle(req, res):** Retrieves a list of all muscles from the database.

4.3.9 Component: equipmentController

- Class diagram:

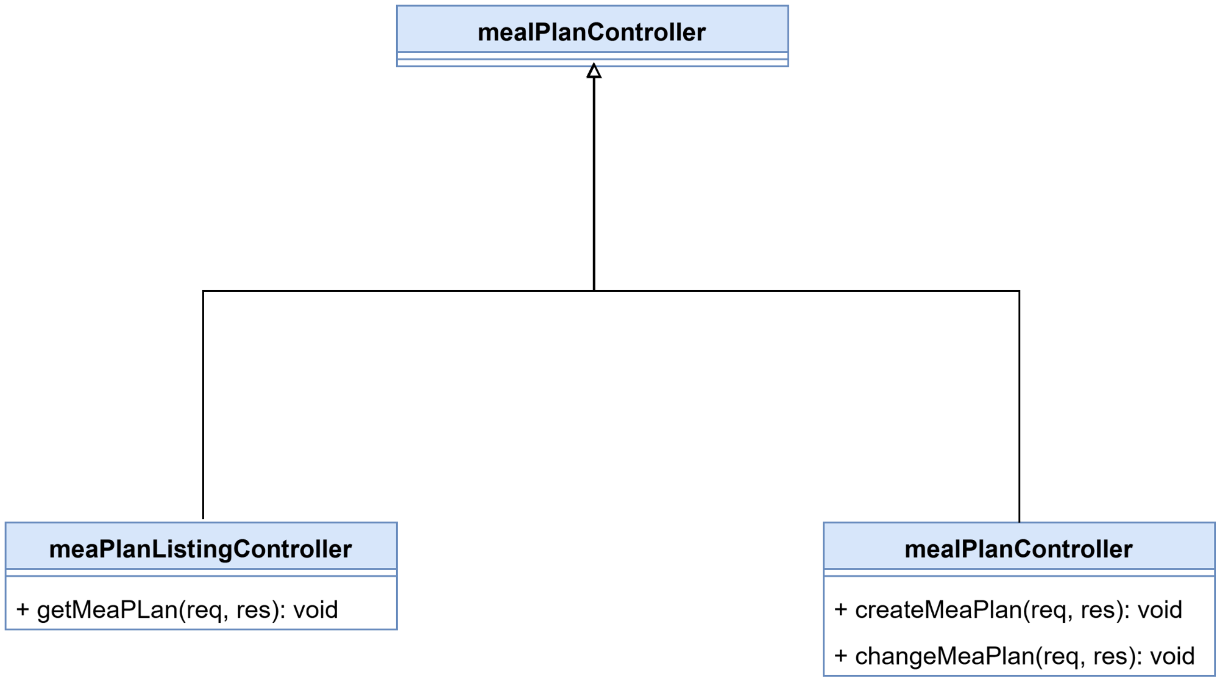


- **Description**

Class/Struct	Attributes	Operations
Class: equipmentController		**getEquipment(req, res):** Retrieves a list of all equipment from the database.

4.3.10 Component: mealPlanController

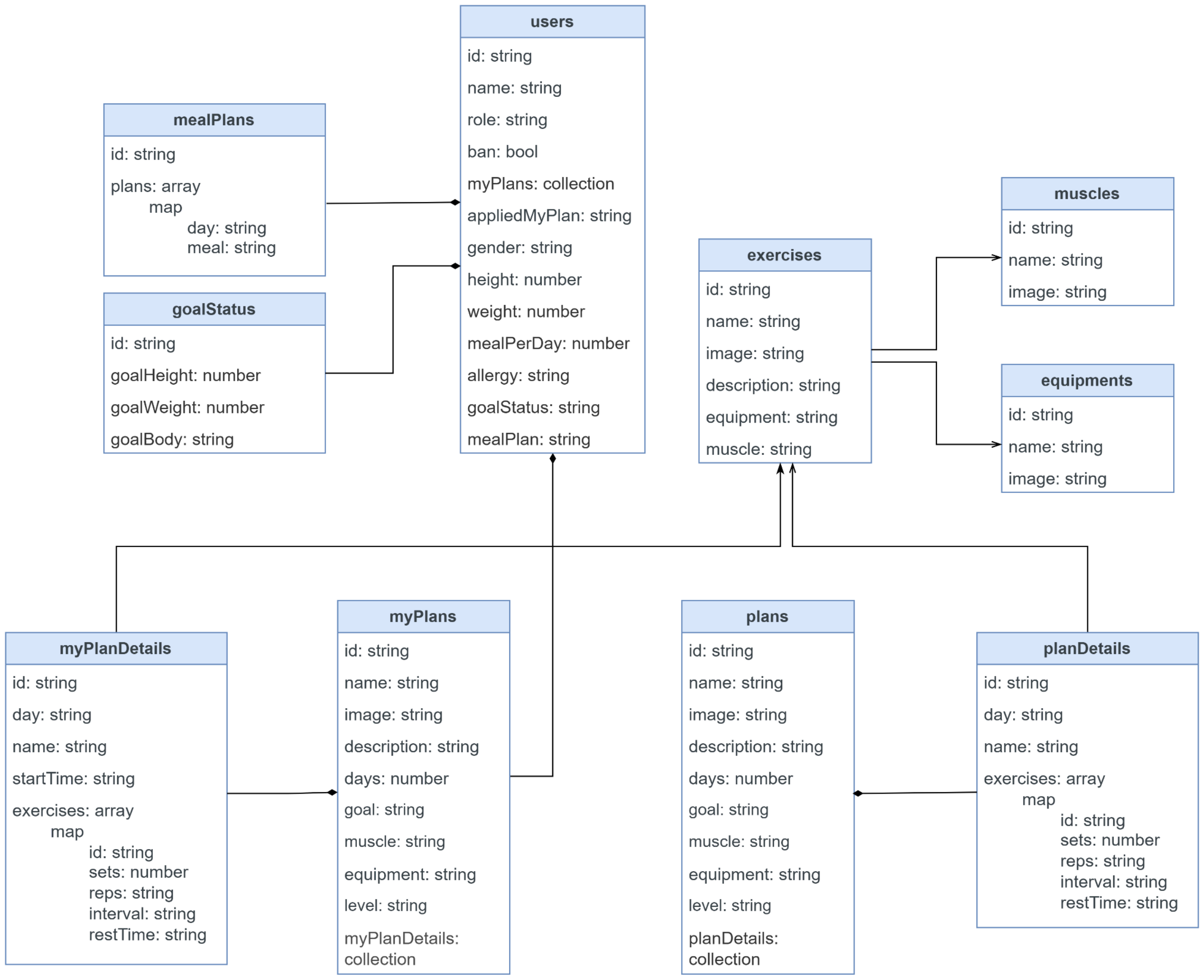
- Class diagram:


mealPlanController
meaPlanListingController
+ getMeaPLan(req, res): void
mealPlanController
+ createMeaPlan(req, res): void
+ changeMeaPlan(req, res): void

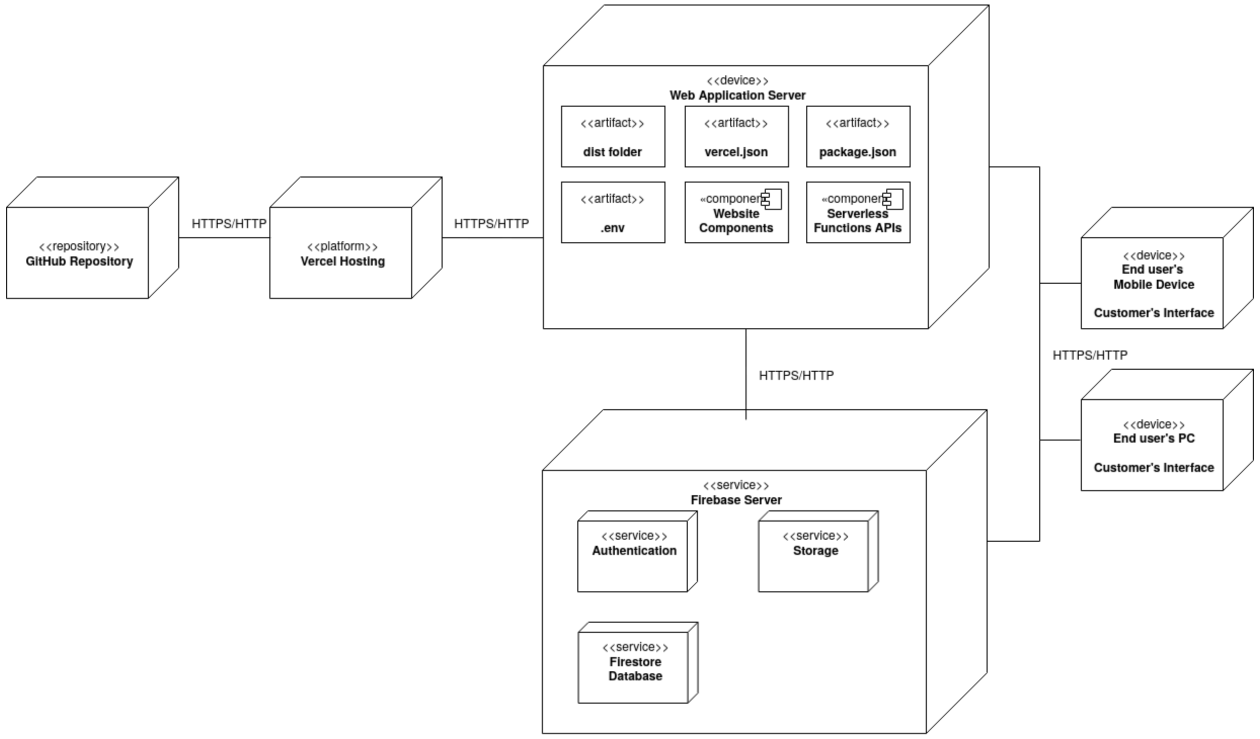
- **Description**

Class/Struct	Attributes	Operations
Class: mealPlanListingController		**getMealPlan(req, res):** Retrieves a list of all meal plans from the database.
Class: mealPlanController		**createMealPlan(req, res):** Creates a new meal plan based on the provided meal plan data. **changeMealPlan(req, res):** Handles modifications to existing meal plans.

4.4 Component: Database



5. Deployment



<<repository>> GitHub Repository
HTTPS/HTTP
<<platform>> Vercel Hosting
HTTPS/HTTP
<<device>> Web Application Server
<<artifact>> dist folder
<<artifact>> vercel.json
<<artifact>> package.json
<<artifact>> .env
«component» Website Components
«component» Serverless Functions APIs
<<device>> End user's Mobile Device Customer's Interface
HTTPS/HTTP
<<device>> End user's PC Customer's Interface
HTTPS/HTTP
<<service>> Firebase Server
<<service>> Authentication
<<service>> Storage
<<service>> Firestore Database

1. **Code Repository and Hosting**
   - **GitHub Repository:** this is the source control for my application code, containing all the project's files, including the web application code, documents and PA submission files.
   - **Hosting:** The code is deployed from the GitHub repository to **Vercel Hosting**. Vercel hosts the application and provides the necessary infrastructure for serving the web application over the internet via HTTPS/HTTP.
2. **Web Application Server**
   - **Artifacts:**
     - **dist folder:** contains the compiled and optimized static assets for the frontend.
     - **vercel.json:** configuration file used to define deployment settings.
     - **package.json:** configuration file used to define project dependencies.
     - **.env:** stores environment variables for secure access to API keys and sensitive data.
   - **Components**
     - **Website components:** these are the frontend user interface elements of the application, responsible for rendering the user experience.
     - **Serverless function APIs:** These APIs handle backend logic and provide server-side functionality, such as database interactions, authentication and business logic.
3. **Backend Services**
   - **Authentication:** Handles user login and signup functionality.
   - **Firestore Database:** Stores application data, such as user profiles, transaction logs or application-specific data.
   - **Storage:** Provides cloud storage for storing files like user-uploaded images or other media.
4. **End User Access**
   - The application is accessible to end users through their devices, including mobile devices

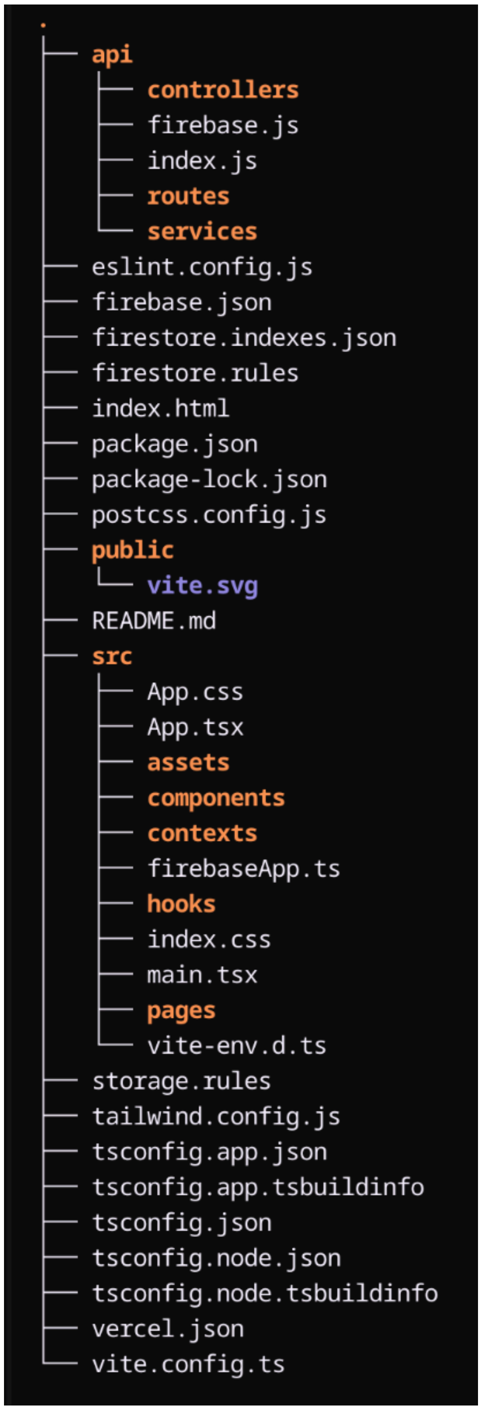
and personal computers (PCs).

- Users interact with the application via a browser-based interface over HTTPS/HTTP. This ensures secure communication between the client devices and the application server.

5. **Communication and Data Flow:**
   - Data flows between the GitHub Repository, Vercel Hosting, Firebase Server, and end-user devices using HTTPS/HTTP protocols, ensuring secure and efficient transmission of information.

6. Implementation View

1. **Root Directory**
   - .env
   - **package.json:** Defines the project's dependencies and scripts, which lists the required Node.js modules and specifies commands for tasks like building, testing and running the application.
   - **README.md:** Provides an overview of the project, including instructions on how to set up and run the application.
   - **vite.svg:** A small SVG icon likely used for branding or as a placeholder.
2. **API Directory**
   - **controllers/:** Contains controllers for handling API requests. These controllers define the logic for processing incoming requests and generating appropriate responses.
   - **firebase.js:** Handles interactions with Firebase, a backend platform for building web and mobile applications.
   - **index.js:** The main entry point for the API server.
   - **routes/:** Defines and maps the API routes to the corresponding controller functions.
   - **services/:** Contains reusable services for common tasks, such as database interactions or external API calls.
3. **Public Directory**
   - **index.html:** The main HTML file for the web application, which serves as the entry point for the front end
4. **Src Directory**
   - **App.css:** Contains global CSS styles for the application.
   - **App.tsx:** The main React component that renders the entire application.
   - **assets:** Stores static assets like images, fonts, and other media files.
   - **components:** Contains reusable React components for building the user interface.
   - **contexts**: Contains React providers for managing global state, such as user authentication or theme settings.
   - **firebaseApp.ts:** Handles Firebase initialization and configuration.
   - **hooks:** Contains custom React hooks for common functionalities, such as data fetching or state management.
   - **index.css:** Contains CSS styles specific to the index page.
   - **main.tsx:** The entry point for the React application.
   - **pages:** Contains React components for individual pages or screens of the application.
   - **vite-end.d.ts:** Defines custom TypeScript types for environment variables.
   - **storage.rules:** Firebase rules for controlling access to the application's storage.
   - **tailwind.config.js:** Configuration file for Tailwind CSS, a utility-first CSS framework.
   - **tsconfig.app.json:** TypeScript configuration file for the application.
   - **tsconfig.app.tsbuildinfo:** TypeScript build information file.
   - **tsconfig.json:** Main TypeScript configuration file.
   - **tsconfig.node.json:** TypeScript configuration file for Node.js.
   - **tsconfig.node.tsbuildinfo:** TypeScripts build information file for Node.js.
   - **vercel.json:** Configuration file for Vercal, a platform for deploying web applications.
   - **vite.config.ts:** Configuration file for Vite, a build tool for modern web applications



```
.
├── api
│   ├── controllers
│   ├── firebase.js
│   ├── index.js
│   ├── routes
│   └── services
├── eslint.config.js
├── firebase.json
├── firestore.indexes.json
├── firestore.rules
├── index.html
├── package.json
├── package-lock.json
├── postcss.config.js
├── public
│   └── vite.svg
├── README.md
├── src
│   ├── App.css
│   ├── App.tsx
│   ├── assets
│   ├── components
│   ├── contexts
│   ├── firebaseApp.ts
│   ├── hooks
│   ├── index.css
│   ├── main.tsx
│   ├── pages
│   └── vite-env.d.ts
├── storage.rules
├── tailwind.config.js
├── tsconfig.app.json
├── tsconfig.app.tsbuildinfo
├── tsconfig.json
├── tsconfig.node.json
├── tsconfig.node.tsbuildinfo
├── vercel.json
└── vite.config.ts
```