FitSync: Comprehensive Fitness Tracking Application

Team Members

1 Team Members

- Ganesh: 2022115035
- Jayasurya: 2022115031

Contents

2 Abstract

FitSync is an innovative fitness application designed to revolutionize personal health management. By leveraging cutting-edge technology, FitSync provides a comprehensive solution for tracking workouts, monitoring nutrition, and achieving fitness goals. The application integrates features such as user authentication, BMI calculation, workout tracking, water intake monitoring, food chart management, and progress sharing. Built with a Java backend using the Spark framework and MongoDB for data persistence, FitSync offers a robust and scalable platform for fitness enthusiasts. The frontend is developed using React Native, ensuring a seamless and responsive user experience across mobile devices.

3 Detailed Feature Documentation

3.1 User Authentication

The user authentication system in FitSync is implemented using Firebase Authentication, providing a secure and reliable way for users to create accounts and access the application.

3.1.1 Sign Up

Users can create a new account by providing their email and password. The system validates the input and creates a new user record in Firebase.

3.1.2 Sign In

Registered users can sign in using their email and password. The system verifies the credentials against the Firebase user database.

3.1.3 Sign Out

Users can securely sign out of the application, terminating their current session.

3.2 BMI Calculator

The Body Mass Index (BMI) calculator allows users to assess their body composition quickly.

3.2.1 Functionality

Users input their weight (in kilograms) and height (in centimeters). The system calculates the BMI using the formula: BMI = weight / ((height / 100) * (height / 100)).

3.2.2 Implementation

The BMI calculation is performed server-side, ensuring consistency across all client devices.

3.3 Workout Tracking

FitSync provides a comprehensive workout tracking system to help users monitor their fitness activities.

3.3.1 Start Workout

Users can initiate a new workout session, specifying the type of workout (e.g., "Push-up Session").

3.3.2 End Workout

Upon completing a workout, users can end the session. The system records the duration and other relevant details.

3.3.3 Next Workout

FitSync suggests the next workout in a predefined sequence, helping users maintain a balanced fitness routine.

3.3.4 Data Storage

Workout data is stored in MongoDB, allowing for efficient retrieval and analysis of user fitness patterns.

3.4 Water Intake Monitoring

FitSync helps users stay hydrated by tracking their daily water intake.

3.4.1 Record Water Intake

Users can log their water consumption throughout the day.

3.4.2 Water Level Tracking

The system maintains a running total of the user's water intake, providing visual feedback on hydration levels.

3.4.3 Data Persistence

Water intake data is stored in MongoDB, allowing for historical tracking and analysis of hydration habits.

3.5 Food Chart Management

FitSync includes a comprehensive food chart to help users make informed dietary choices.

3.5.1 Food Database

The application maintains a database of common food items, including their nutritional information (calories, protein, carbohydrates, and fat content).

3.5.2 Retrieval and Display

Users can access the food chart, which displays a list of food items along with their nutritional values.

3.5.3 Data Storage

Food chart data is stored in MongoDB, allowing for easy updates and maintenance of the nutritional database.

3.6 Progress Sharing

FitSync incorporates social features, allowing users to share their fitness progress with others.

3.6.1 Create Posts

Users can create posts about their workouts, achievements, or fitness tips.

3.6.2 Like and Comment

The system supports social interactions, allowing users to like and comment on posts from other users.

3.6.3 Data Management

Post data, including likes and comments, is stored in MongoDB, enabling efficient retrieval and display of user-generated content.

4 Technical Implementation

4.1 Backend Architecture

FitSync's backend is built using Java with the Spark framework, providing a lightweight and flexible foundation for the application's API.

4.1.1 Controller Structure

The backend is organized into several controllers, each responsible for a specific feature:

- AuthController: Manages user authentication
- BMIController: Handles BMI calculations
- WorkoutController: Manages workout sessions
- WaterReminderController: Tracks water intake
- FoodChartController: Manages the food database and retrieval
- ProgressSharingController: Handles social interactions and posts

4.1.2 Database Integration

MongoDB is used as the primary database, providing flexible and scalable data storage for user information, workout data, and other application content.

4.2 Frontend Development

The frontend of FitSync is developed using React Native, ensuring a native-like experience on both iOS and Android platforms.

4.2.1 User Interface

The UI is designed to be intuitive and responsive, with easy-to-use components for tracking workouts, logging water intake, and accessing the food chart.

4.2.2 State Management

React's state management is utilized to maintain a smooth and responsive user experience, with real-time updates for workout progress, water intake, and social interactions.

5 Code Snippets

Below are key code snippets from the FitSync project, showcasing the implementation of all major features:

5.1 Authentication Controller

```java
package com.fitnessapp;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseAuthException;
import com.google.firebase.auth.UserRecord;
import com.google.gson.Gson;

import spark.Request;
import spark.Response;
import static spark.Spark.post;

public class AuthController {

    private static final Gson gson = new Gson();

    public static void init() {
        post("/signup", AuthController::signUp);
        post("/signin", AuthController::signIn);
        post("/signout", AuthController::signOut);
    }

    private static Object signUp(Request req, Response res) {
        try {
            UserCredentials credentials = gson.fromJson(req.body(), UserCredentials.class);
            UserRecord.CreateRequest request = new UserRecord.CreateRequest()
                    .setEmail(credentials.email)
                    .setPassword(credentials.password);

            UserRecord userRecord = FirebaseAuth.getInstance().createUser(request);
            res.status(200);
            return gson.toJson(new AuthResponse(userRecord.getUid(), userRecord.getEmail(), true));
        } catch (FirebaseAuthException e) {
            res.status(400);
            return gson.toJson(new ErrorResponse(e.getMessage()));
        }
    }

    private static Object signIn(Request req, Response res) {
        try {
            UserCredentials credentials = gson.fromJson(req.body(), UserCredentials.class);
            String uid = FirebaseAuth.getInstance().getUserByEmail(credentials.email).getUid();
            res.status(200);
            Boolean success = true;
            return gson.toJson(new AuthResponse(uid, credentials.email, success));
        } catch (FirebaseAuthException e) {
            res.status(400);
            return gson.toJson(new ErrorResponse("Invalid email or password"));
        }
    }
```

```
    private static Object signOut(Request req, Response res) {
        // Firebase doesn't have a server-side sign-out mechanism
        // Client-side sign-out is sufficient
        res.status(200);
        return gson.toJson(new SuccessResponse("Signed out successfully"));
    }

    private static class UserCredentials {
        String email;
        String password;
    }

    private static class AuthResponse {
        String uid;
        String email;
        Boolean success;

        AuthResponse(String uid, String email, Boolean success) {
            this.uid = uid;
            this.email = email;
            this.success = success;
        }
    }

    private static class ErrorResponse {
        String error;

        ErrorResponse(String error) {
            this.error = error;
        }
    }

    private static class SuccessResponse {
        String message;

        SuccessResponse(String message) {
            this.message = message;
        }
    }
}
```

5.2 BMI Controller

```
package com.fitnessapp;

import com.google.gson.Gson;

import static spark.Spark.post;

public class BMIController {

    public static void init() {
        post("/bmi", (req, res) -> {
            Gson gson = new Gson();
            BMIInput input = gson.fromJson(req.body(), BMIInput.class);
            double bmi = calculateBMI(input.weight, input.height);
            return gson.toJson(new BMIResult(bmi));
        });
    }

    private static double calculateBMI(double weight, double height) {
        return weight / ((height / 100) * (height / 100));
    }

    private static class BMIInput {
        double weight;
        double height;
    }

    private static class BMIResult {
        double bmi;

        BMIResult(double bmi) {
            this.bmi = bmi;
        }
    }
}
```

5.3 Workout Controller

```java
package com.fitnessapp;

import org.bson.Document;

import com.google.gson.Gson;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import static spark.Spark.post;

public class WorkoutController {

    private static MongoClient mongoClient;
    private static MongoDatabase database;
    private static MongoCollection<Document> workoutsCollection;

    public static void init() {

        mongoClient = MongoClients.create("mongodb://localhost:27017");
        database = mongoClient.getDatabase("fitnessApp");
        workoutsCollection = database.getCollection("workouts");

        // Start a workout
        post("/workouts/start", (req, res) -> {
            Workout workout = new Workout("Push-up Session");  // Can be dynamically set
            return new Gson().toJson(workout);
        });

        // End a workout and save result to database
        post("/workouts/end", (req, res) -> {
            Gson gson = new Gson();
            WorkoutResult result = gson.fromJson(req.body(), WorkoutResult.class);

            saveWorkoutResult(result);

            return gson.toJson(new WorkoutResponse("Workout ended successfully"));
        });

        // Start next workout in the sequence
        post("/workouts/next", (req, res) -> {
            Workout nextWorkout = new Workout("Squat Session");  // Dynamically pick the next workout
            return new Gson().toJson(nextWorkout);
        });
    }

    // Save workout result to MongoDB
    private static void saveWorkoutResult(WorkoutResult result) {
        try {
            Document workoutDoc = new Document("duration", result.duration)
                    .append("timestamp", System.currentTimeMillis());  // Storing timestamp when the
    workout ended
            workoutsCollection.insertOne(workoutDoc);
            System.out.println("Workout saved to database.");
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to save workout to database.");
        }
    }

    private static class Workout {
        String name;

        Workout(String name) {
            this.name = name;
        }
    }

    private static class WorkoutResult {
        int duration;
    }

    private static class WorkoutResponse {
        String message;

        WorkoutResponse(String message) {
            this.message = message;
        }
```

```
    }
}
```

5.4 Water Reminder Controller

```
package com.fitnessapp;

import org.bson.Document;

import com.google.gson.Gson;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import static spark.Spark.post;

public class WaterReminderController {

    private static MongoClient mongoClient;
    private static MongoDatabase database;
    private static MongoCollection<Document> waterCollection;

    private static int waterLevel = 0;

    public static void init() {
        // Initialize MongoDB connection
        mongoClient = MongoClients.create("mongodb://localhost:27017");  // Adjust for your MongoDB server URL
        database = mongoClient.getDatabase("fitnessApp");
        waterCollection = database.getCollection("waterIntakes");

        post("/water", (req, res) -> {
            Gson gson = new Gson();
            WaterInput input = gson.fromJson(req.body(), WaterInput.class);

            // Update water level and save to database
            waterLevel += input.amount;
            saveWaterIntake(input.amount);

            return gson.toJson(new WaterResult(waterLevel));
        });
    }

    // Save water intake to MongoDB
    private static void saveWaterIntake(int amount) {
        try {
            Document waterDoc = new Document("amount", amount)
                    .append("waterLevel", waterLevel)
                    .append("timestamp", System.currentTimeMillis());  // Storing the timestamp of water intake
            waterCollection.insertOne(waterDoc);
            System.out.println("Water intake saved to database.");
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to save water intake to database.");
        }
    }

    private static class WaterInput {
        int amount;
    }

    private static class WaterResult {
        int waterLevel;

        WaterResult(int waterLevel) {
            this.waterLevel = waterLevel;
        }
    }
}
```

5.5 Food Chart Controller

```
package com.fitnessapp;

import java.util.ArrayList;
```

```
import java.util.List;

import org.bson.Document;

import com.google.gson.Gson;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import static spark.Spark.get;

public class FoodChartController {

    private static MongoCollection<Document> foodCollection;

    public static void init(MongoDatabase database) {
        foodCollection = database.getCollection("food");

        get("/food-chart", (req, res) -> {
            List<FoodItem> foodChart = getFoodChart();
            return new Gson().toJson(foodChart);
        });
    }

    private static List<FoodItem> getFoodChart() {
        List<FoodItem> foodChart = new ArrayList<>();
        for (Document doc : foodCollection.find()) {
            foodChart.add(new FoodItem(
                    doc.getString("name"),
                    doc.getInteger("calories"),
                    doc.getDouble("protein"),
                    doc.getDouble("carbs"),
                    doc.getDouble("fat")
            ));
        }
        return foodChart;
    }

    private static class FoodItem {
        String name;
        int calories;
        double protein, carbs, fat;

        FoodItem(String name, int calories, double protein, double carbs, double fat) {
            this.name = name;
            this.calories = calories;
            this.protein = protein;
            this.carbs = carbs;
            this.fat = fat;
        }
    }
}
```

5.6 Progress Sharing Controller (Social Features)

```
package com.fitnessapp;

import java.util.ArrayList;
import java.util.List;

import org.bson.Document;
import org.bson.types.ObjectId;

import com.google.gson.Gson;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import spark.Request;
import spark.Response;
import static spark.Spark.get;
import static spark.Spark.post;

public class ProgressSharingController {

    private static final Gson gson = new Gson();
    private static MongoCollection<Document> posts;

    public static void init(MongoDatabase database) {
        posts = database.getCollection("posts");

```

```
        get("/posts", ProgressSharingController::getPosts);
        post("/posts", ProgressSharingController::createPost);
        post("/posts/:id/like", ProgressSharingController::likePost);
        post("/posts/:id/comment", ProgressSharingController::commentOnPost);
    }

    private static Object getPosts(Request req, Response res) {
        List<Document> postList = posts.find().into(new ArrayList<>());
        res.type("application/json");
        return gson.toJson(postList);
    }

    private static Object createPost(Request req, Response res) {
        String text = req.queryParams("text");
        String mediaUrl = req.queryParams("mediaUrl");
        String email = req.queryParams("email");

        if (text == null || text.trim().isEmpty()) {
            res.status(400);
            return "{\"error\":\"Text is required for creating a post.\"}";
        }

        if (email == null || email.trim().isEmpty()) {
            res.status(400);
            return "{\"error\":\"Email is required for creating a post.\"}";
        }

        Document post = new Document("text", text)
                .append("mediaUrl", mediaUrl != null ? mediaUrl : "")
                .append("email", email)
                .append("likes", 0)
                .append("comments", new ArrayList<String>());

        posts.insertOne(post);

        res.status(201);
        return gson.toJson(post);
    }

    private static Object likePost(Request req, Response res) {
        String postId = req.params(":id");
        Document post = posts.find(new Document("_id", new ObjectId(postId))).first();

        if (post != null) {
            int likes = post.getInteger("likes", 0) + 1;
            posts.updateOne(new Document("_id", new ObjectId(postId)),
                    new Document("$set", new Document("likes", likes)));
            post.put("likes", likes);
            res.status(200);
        } else {
            res.status(404);
            return "{\"error\":\"Post not found.\"}";
        }

        res.type("application/json");
        return gson.toJson(post);
    }

    private static Object commentOnPost(Request req, Response res) {
        String postId = req.params(":id");
        String comment = req.queryParams("text");

        if (comment == null || comment.trim().isEmpty()) {
            res.status(400);
            return "{\"error\":\"Comment text is required.\"}";
        }

        Document post = posts.find(new Document("_id", new ObjectId(postId))).first();

        if (post != null) {
            List<String> comments = post.getList("comments", String.class);
            comments.add(comment);
            posts.updateOne(new Document("_id", new ObjectId(postId)),
                    new Document("$set", new Document("comments", comments)));
            post.put("comments", comments);
            res.status(200);
        } else {
            res.status(404);
            return "{\"error\":\"Post not found.\"}";
        }

```

```
        res.type("application/json");
        return gson.toJson(post);
    }
}
```

5.7 Main Application Entry Point

```
package com.fitnessapp;

import java.io.FileInputStream;
import java.io.IOException;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoDatabase;

import static spark.Spark.port;

public class Main {

    public static void main(String[] args) throws IOException {
        port(4567);

        // Initialize Firebase Admin SDK
        FileInputStream serviceAccount = new FileInputStream("src/main/resources/google-services.json");
        FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                .build();
        FirebaseApp.initializeApp(options);

        MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");
        MongoDatabase database = mongoClient.getDatabase("fitnessapp");

        AuthController.init();
        BMIController.init();
        YouTubeScraperController.init();
        WaterReminderController.init();
        FoodChartController.init(database);
        WorkoutController.init();
        ProgressSharingController.init(database);
    }
}
```

These code snippets provide a comprehensive overview of the FitSync application's backend implementation, showcasing the various controllers responsible for different features such as authentication, BMI calculation, workout tracking, water intake monitoring, food chart management, and progress sharing. The Main class serves as the entry point, initializing all necessary components and setting up the MongoDB connection.

6 Future Enhancements

While FitSync already offers a comprehensive set of features, there are several potential enhancements that could further improve the application:

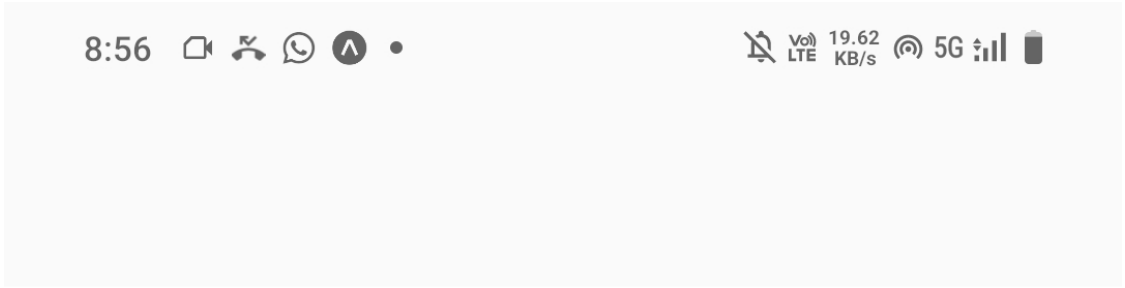
- Integration with wearable devices for automatic workout and health data tracking
- Personalized workout recommendations based on user goals and progress
- Advanced analytics and visualizations for long-term progress tracking
- Integration with nutrition APIs for more detailed and up-to-date food information
- Gamification elements to increase user engagement and motivation

7 Conclusion

FitSync represents a significant step forward in personal fitness management applications. By combining workout tracking, nutrition monitoring, and social features, it provides users with a comprehensive tool for achieving their fitness goals. The robust backend architecture and user-friendly frontend ensure that FitSync is both powerful and accessible, making it an invaluable asset for fitness enthusiasts of all levels.

8 Screenshot - Uniqueness

The screenshot above showcases FitSync's unique integrated fitness tracking dashboard, which combines workout tracking, water intake monitoring, and nutritional information in a single, user-friendly interface. This comprehensive approach sets FitSync apart from other fitness applications by providing users with a holistic view of their health and fitness journey.



Write your post

Enter media URL

Post

ganeshsriramulu2@gmail.com

1 likes

Example

ganeshsriramulu2@gmail.com King 👑

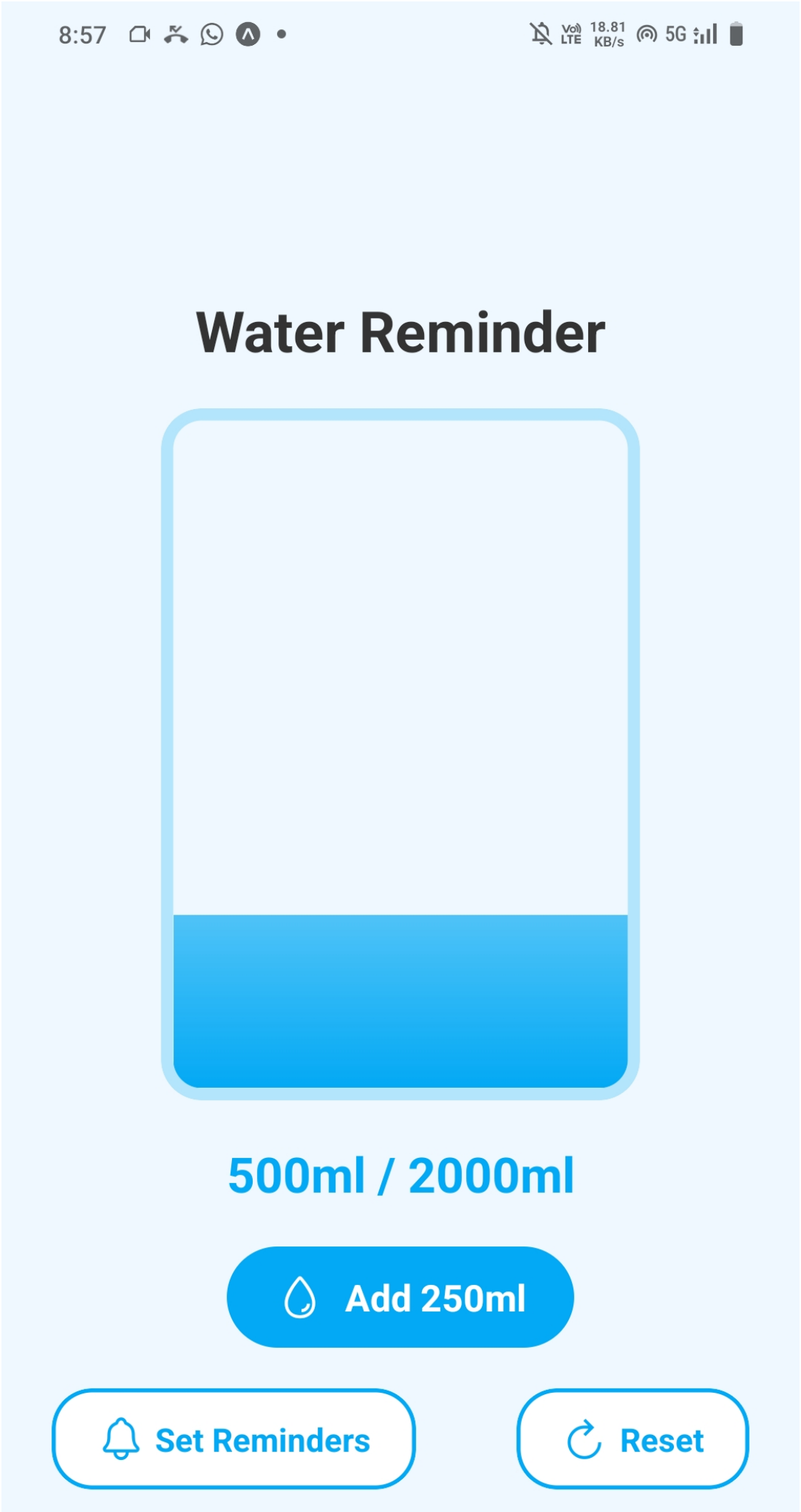
8:50 131.8 KB/s 5G

Workouts

Cycling

2s

End Workout


8:57
18.81 KB/s
5G
Water Reminder
500ml / 2000ml
Add 250ml
Set Reminders
Reset

Welcome to Your Fitness Journey

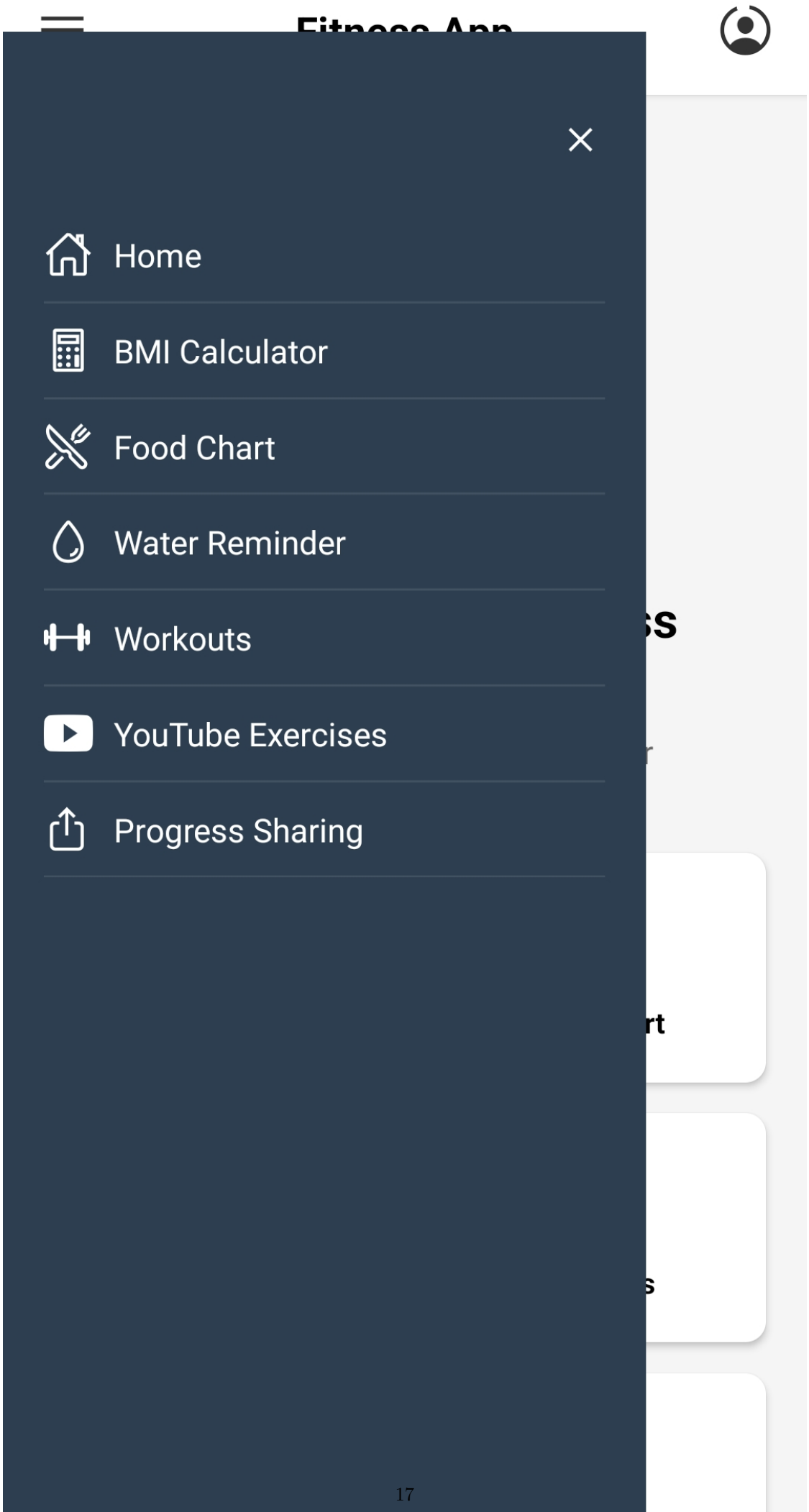
Achieve your health goals with our comprehensive tools

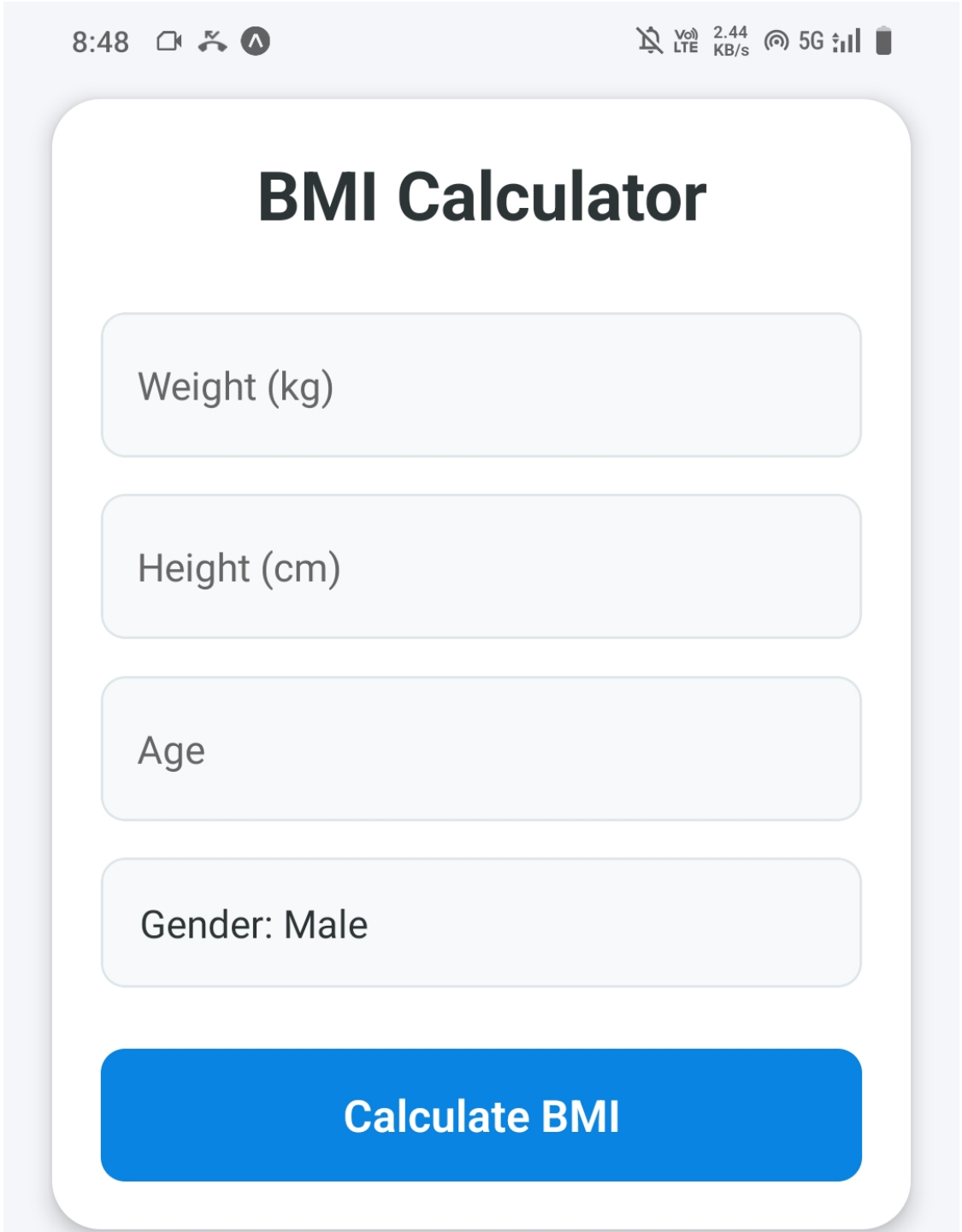
BMI Calculator

Food Chart

Water Reminder

Workouts


Home
BMI Calculator
Food Chart
Water Reminder
Workouts
YouTube Exercises
Progress Sharing


8:48
2.44 KB/s
5G
BMI Calculator
Weight (kg)
Height (cm)
Age
Gender: Male
Calculate BMI

Underweight

<18.5

Breakfast

Recommended Foods

Eggs, Toast, Peanut Butter

Nutrition Facts

Calories 350 kcal

Protein 20g

Key Nutrients

Protein, Carbs, Healthy Fats

Follow for 3 months

Underweight

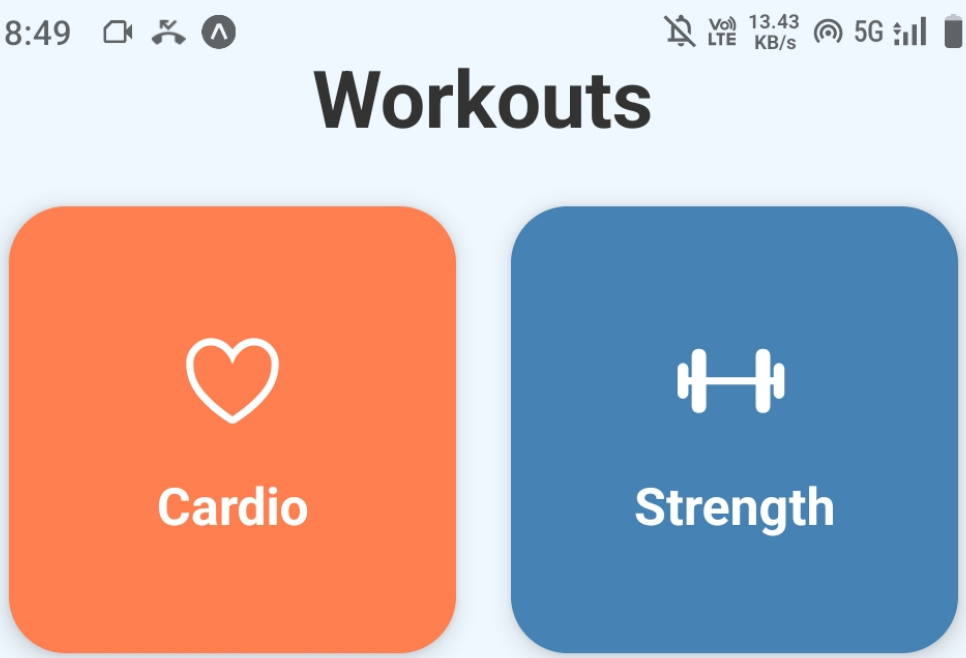
<18.5

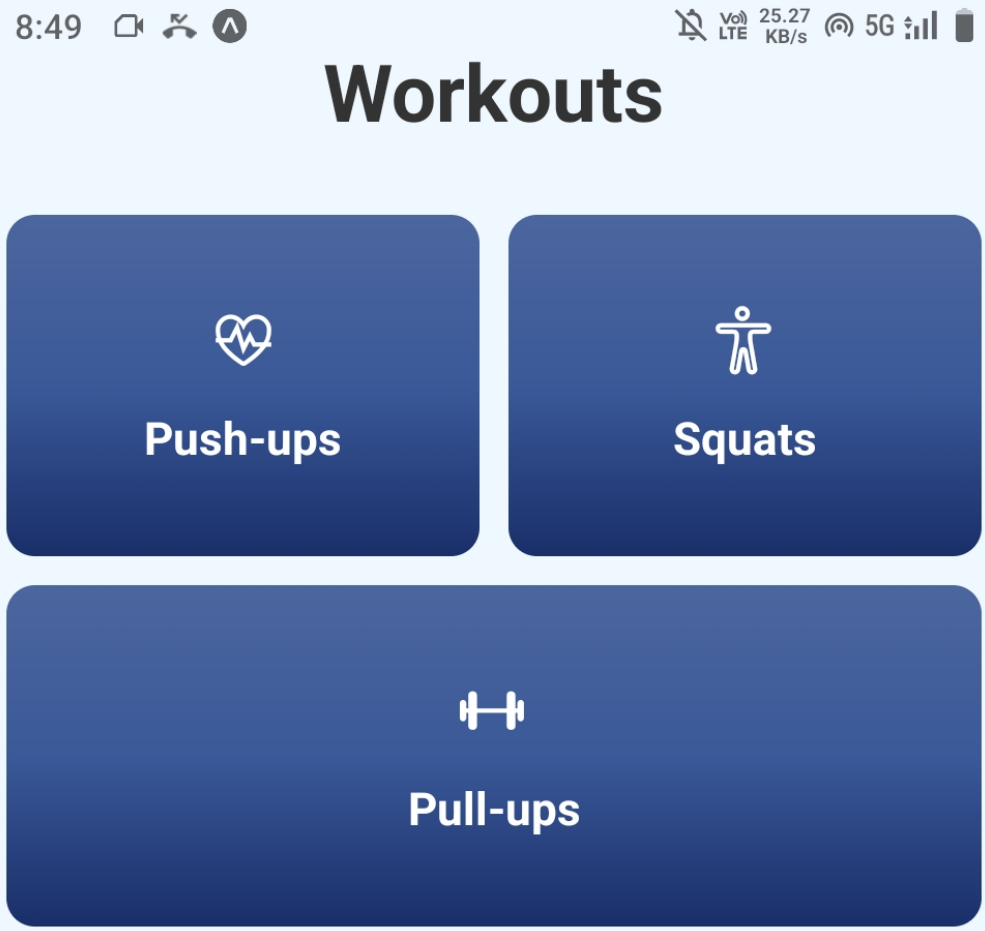
Lunch

Recommended Foods

Brown Rice, Lentils, Chicken Curry

Nutrition Facts


8:49
13.43 KB/s
5G
Workouts
Cardio
Strength


8:49
25.27 KB/s
5G
Workouts
Push-ups
Squats
Pull-ups

Fitness App

Welcome to Your Fitness Journey

User Session

ganeshsriramulu2@gmail.com

Sign Out

Close

Water Reminder

Workouts