An application-based Data-Driven AI Fitness Trainer integrating Deep Learning Algorithms and Computer Vision

Language of work - Python | Dart

File Structure Directory

```
AI_Fitness_Trainer/
    ├── .dart_tool/
    ├── .idea/
    ├── .vscode/
    ├── android/
    ├── assets/
    ├── build/
    ├── copyright/
    ├── functions/
    │   ├── __pycache__/
    │   ├── node_modules/
    │   ├── package-lock.json
    ├── src/
    │   ├── .eslintrc.js
    │   ├── .gitignore
    │   ├── app.py
    │   ├── diet_api.py
    │   ├── diet_recommendation.py
    │   ├── eslintrc.config.js
    │   ├── exercise_detection.py
    │   ├── exercise_pose_model.h5
    │   ├── label_encoder.pkl
    │   ├── label_encoders_mlp.pkl
    │   ├── package.json
    │   ├── tsconfig.json
    │   ├── updated_categorized_food_dataset.csv
    │   ├── Updated_UserParameters_dataset.csv
    │   ├── workout_recommendation_api.py
    │   ├── workout_recommendation_mlp_model.pkl
    ├── ios/
    ├── lib/
    │   ├── auth_page.dart
    │   ├── DietRecommendationPage.dart
    │   ├── firebase_options.dart
    │   ├── home_screen.dart
    │   ├── landing_page.dart
```

```
│   ├── ProfileSetupScreen.dart
│   ├── VirtualFitnessTrainerScreen.dart
│   ├── WorkoutRecommendationScreen.dart
├── linux/
├── macos/
├── web/
├── windows/
├── .firebase/
├── flutter-plugins
├── flutter-plugins-dependencies
├── .gitignore
├── ai_fitness_trainer.iml
├── analysis_options.yaml
├── firebase.json
├── GenerateCopyrightPdf.dart
├── pubspec.lock
├── pubspec.yaml
├── README.md
```

Application Screens :-

1. lib/auth_page.dart

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'home_screen.dart'; // Import HomeScreen for redirection

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

  @override
  _AuthScreenState createState() => _AuthScreenState();
}

class _AuthScreenState extends State<AuthScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  bool isLogin = true;
  String errorMessage = '';
```

```
Future<void> _authenticate() async {
  try {
    UserCredential userCredential;
    if (isLogin) {
      userCredential = await _auth.signInWithEmailAndPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim(),
      );
    } else {
      userCredential = await _auth.createUserWithEmailAndPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim(),
      );
    }

    // Navigate to HomeScreen on successful authentication
    if (userCredential.user != null) {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => HomeScreen()),
      );
    }
  } on FirebaseAuthException catch (e) {
    setState(() {
      errorMessage = e.message ?? 'An error occurred';
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text(isLogin ? 'Sign In' : 'Sign Up')),
    body: Padding(
      padding: EdgeInsets.all(16.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          TextField(
```

```
            controller: _emailController,
            decoration: InputDecoration(labelText: 'Email'),
            keyboardType: TextInputType.emailAddress,
          ),
          TextField(
            controller: _passwordController,
            decoration: InputDecoration(labelText: 'Password'),
            obscureText: true,
          ),
          SizedBox(height: 10),
          if (errorMessage.isNotEmpty)
            Text(errorMessage, style: TextStyle(color: Colors.red)),
          SizedBox(height: 10),
          ElevatedButton(
            onPressed: _authenticate,
            child: Text(isLogin ? 'Sign In' : 'Sign Up'),
          ),
          TextButton(
            onPressed: () {
              setState(() {
                isLogin = !isLogin;
                errorMessage = '';
              });
            },
            child: Text(
              isLogin
                  ? 'Create an account'
                  : 'Already have an account? Sign In',
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

2. lib/firebase_options.dart

```
import 'package:firebase_core/firebase_core.dart';
```

```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    return FirebaseOptions(
      apiKey: "AIzaSyAkLhVnAdqfF7yS316aCUzvkNibYvw7XzU",
      authDomain: "ai-fitness-trainer-7a636.firebaseapp.com",
      projectId: "ai-fitness-trainer-7a636",
      storageBucket:
          "ai-fitness-trainer-7a636.appspot.com", // Fixed storageBucket
      messagingSenderId: "190246922760",
      appId: "1:190246922760:web:be947a5be63a474d237fbd",
      measurementId: "G-072Q8WH399", // Optional for analytics
    );
  }
}
```

3. lib/home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'DietRecommendationPage.dart'; // Ensure this file exists
import 'WorkoutRecommendationScreen.dart'; // Import the WorkoutRecommendationScreen
import 'ProfileSetupScreen.dart'; // New page for profile setup
import 'VirtualFitnessTrainerScreen.dart';

class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  List<String> goals = [];
  Map<String, bool> completedGoals = {};
  int streak = 5; // Example streak count
  final TextEditingController _goalController = TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
```

```
@override
void initState() {
  super.initState();
  _loadGoals();
}

// Load goals from Firebase
Future<void> _loadGoals() async {
  String uid = _auth.currentUser?.uid ?? "";
  if (uid.isEmpty) return;

  DocumentSnapshot userDoc =
      await _firestore.collection("users").doc(uid).get();
  if (userDoc.exists) {
    Map<String, dynamic>? data = userDoc.data() as Map<String, dynamic>?;
    if (data != null && data.containsKey("goals")) {
      setState(() {
        goals = List<String>.from(data["goals"]);
        completedGoals = Map<String, bool>.from(data["completedGoals"] ?? {});
      });
    }
  }
}

// Add goal
void addGoal() {
  if (_goalController.text.isNotEmpty) {
    setState(() {
      goals.add(_goalController.text);
      completedGoals[_goalController.text] = false;
      _goalController.clear();
    });
    _saveGoals();
  }
}

// Toggle goal completion
void toggleGoal(String goal) {
  setState(() {
    completedGoals[goal] = !(completedGoals[goal] ?? false);
```

```
  });
  _saveGoals();
}

// Calculate goal progress percentage
double getCompletedPercentage() {
  if (goals.isEmpty) return 0;
  int completed = completedGoals.values.where((done) => done).length;
  return completed / goals.length;
}

// Save Goals to Firestore
Future<void> _saveGoals() async {
  String uid = _auth.currentUser?.uid ?? "";
  if (uid.isEmpty) return;

  await _firestore.collection("users").doc(uid).set({
    "goals": goals,
    "completedGoals": completedGoals,
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF171717),
    body: SafeArea(
      child: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              "Good Morning, User",
              style: TextStyle(
                fontSize: 28,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
```

```
          SizedBox(height: 8),
          Text(
            "Hope you're getting enough exercise and staying hydrated",
            style: TextStyle(fontSize: 16, color: Colors.white70),
          ),
          SizedBox(height: 20),

          // Profile Setup Button
          _buildProfileSetupButton(),

          SizedBox(height: 20),
          _buildGoalsSection(),
          SizedBox(height: 20),
          _buildProgressChart(),
          SizedBox(height: 20),
          _buildStreakSection(),
          SizedBox(height: 20),
          _buildNavigationButtons(),
        ],
      ),
    ),
  ),
 );
}

// Profile Setup Button
Widget _buildProfileSetupButton() {
 return Container(
  padding: EdgeInsets.all(16),
  decoration: BoxDecoration(
   color: Color(0xFF22CCA5),
   borderRadius: BorderRadius.circular(12),
  ),
  child: Column(
   crossAxisAlignment: CrossAxisAlignment.start,
   children: [
    Text(
     "Complete Your Profile",
     style: TextStyle(
      fontSize: 18,
```

```
          fontWeight: FontWeight.bold,
          color: Colors.black,
        ),
      ),
      SizedBox(height: 10),
      ElevatedButton(
        style: ElevatedButton.styleFrom(backgroundColor: Color(0xFF171717)),
        onPressed: () {
          // Navigate to profile setup page
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => ProfileSetup()),
          );
        },
        child: Text(
          "Go to Profile Setup",
          style: TextStyle(color: Colors.white),
        ),
      ),
    ],
  ),
 );
}

// Goals Section
Widget _buildGoalsSection() {
 return Container(
  decoration: BoxDecoration(
   color: Color(0xFF22CCA5),
   borderRadius: BorderRadius.circular(12),
  ),
  padding: EdgeInsets.all(16),
  child: Column(
   crossAxisAlignment: CrossAxisAlignment.start,
   children: [
    Text(
     "GOALS",
     style: TextStyle(
      fontSize: 18,
      fontWeight: FontWeight.bold,
```

```
          color: Colors.black,
        ),
      ),
      SizedBox(height: 10),
      TextField(
        controller: _goalController,
        decoration: InputDecoration(
          hintText: "Enter a goal",
          suffixIcon: IconButton(icon: Icon(Icons.add), onPressed: addGoal),
        ),
      ),
      SizedBox(height: 10),
      Column(
        children:
            goals.map((goal) {
              return CheckboxListTile(
                title: Text(goal, style: TextStyle(color: Colors.black)),
                value: completedGoals[goal] ?? false,
                onChanged: (value) => toggleGoal(goal),
                activeColor: Colors.black,
              );
            }).toList(),
      ),
    ],
  ),
 );
}

// Progress Chart
Widget _buildProgressChart() {
  return Container(
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(12),
    ),
    padding: EdgeInsets.all(16),
    child: Column(
      children: [
        Text(
          "Goal Progress",
```

```
        style: TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.bold,
          color: Colors.black,
        ),
      ),
      SizedBox(height: 10),
      SizedBox(
        height: 150,
        child: PieChart(
          PieChartData(
            sections: [
              PieChartSectionData(
                value: getCompletedPercentage(),
                color: Colors.green,
                title: "${(getCompletedPercentage() * 100).toInt()}%",
                radius: 50,
                titleStyle: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
              PieChartSectionData(
                value: 1 - getCompletedPercentage(),
                color: Colors.grey,
                radius: 50,
              ),
            ],
          ),
        ),
      ),
    ],
  ),
 );
}

// Streak Section
Widget _buildStreakSection() {
 return Container(
```

```
    decoration: BoxDecoration(
      color: Color(0xFF22CCA5),
      borderRadius: BorderRadius.circular(12),
    ),
    padding: EdgeInsets.all(16),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Text(
          "Streak: $streak days",
          style: TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.bold,
            color: Colors.black,
          ),
        ),
      ],
    ),
  );
}

// Navigation Buttons
Widget _buildNavigationButtons() {
  return Column(
    children: [
      _buildNavButton("Diet Recommendation", Icons.restaurant, () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => DietRecommendationScreen()),
        );
      }),
      SizedBox(height: 10),
      _buildNavButton("Workout Recommendation", Icons.fitness_center, () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => WorkoutRecommendationScreen(),
          ),
        );
      }),
```

```
      SizedBox(height: 10),
      _buildNavButton("Virtual AI Fitness Trainer", Icons.accessibility, () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => VirtualFitnessTrainerScreen(),
          ),
        );
      }),
    ],
  );
 }

 // Custom Button Widget
 Widget _buildNavButton(String text, IconData icon, Function() onPressed) {
  return ElevatedButton.icon(
    onPressed: onPressed,
    icon: Icon(icon, color: Colors.white),
    label: Text(text, style: TextStyle(color: Colors.white)),
    style: ElevatedButton.styleFrom(
      backgroundColor: Color(0xFF171717),
      padding: EdgeInsets.symmetric(vertical: 15, horizontal: 20),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
    ),
  );
 }
}
```

4. lib/landing_page.dart

```
import 'package:flutter/material.dart';

class LandingPage extends StatelessWidget {
 const LandingPage({super.key});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.black,
    body: Column(
      mainAxisAlignment: MainAxisAlignment.center,
```

```
children: [
  Text(
    "WELCOME TO",
    style: TextStyle(
      fontSize: 24,
      fontWeight: FontWeight.bold,
      color: Colors.tealAccent,
      letterSpacing: 2,
    ),
  ),
  SizedBox(height: 10),
  Text(
    "AIFT",
    style: TextStyle(
      fontSize: 48,
      fontWeight: FontWeight.bold,
      color: Colors.white,
    ),
  ),
  SizedBox(height: 20),
  Image.asset(
    'assets/images/streak.png', // Replace with actual image path
    width: 350, // Adjusted to a smaller size
    height: 400, // Ensuring it fits well
    fit: BoxFit.contain,
  ),
  Spacer(),
  Padding(
    padding: const EdgeInsets.symmetric(
      horizontal: 20.0,
      vertical: 10.0,
    ),
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: Color(0xFF22CCA5),
        minimumSize: Size(double.infinity, 60),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(30),
        ),
      ),
```

```
            onPressed: () {
              Navigator.pushNamed(context, '/auth');
            },
            child: Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Text(
                  "Get Started",
                  style: TextStyle(
                    fontSize: 18,
                    color: Colors.black,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                SizedBox(width: 10),
                Icon(Icons.arrow_forward, color: Colors.black),
              ],
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```

5. lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';
import 'landing_page.dart';
import 'auth_page.dart';
import 'home_screen.dart'; // Ensure HomeScreen is imported
import 'package:firebase_auth/firebase_auth.dart';
import 'DietRecommendationPage.dart';

void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
 runApp(MyApp()); // Removed 'const'
```

```
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'AI Fitness Trainer',
      theme: ThemeData.dark(),
      home: StreamBuilder<User?>(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(
              child: CircularProgressIndicator(),
            ); // Show loading indicator
          }
          if (snapshot.hasData) {
            return HomeScreen(); // If user is signed in, go to HomeScreen
          }
          return LandingPage(); // Otherwise, show Landing Page
        },
      ),
      routes: {
        '/auth': (context) => AuthScreen(),
        '/home': (context) => HomeScreen(),
        '/diet':
            (context) => DietRecommendationScreen(), // Ensure this page exists
      },
    );
  }
}
```

6. lib/ProfileSetupScreen.dart

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';

class ProfileSetup extends StatefulWidget {
```

```
  const ProfileSetup({super.key});

  @override
  _ProfileSetupState createState() => _ProfileSetupState();
}

class _ProfileSetupState extends State<ProfileSetup> {
  final TextEditingController _birthdateController = TextEditingController();
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _phoneController = TextEditingController();

  String _selectedGender = 'Male'; // Default selected gender
  bool _isValidDate = true; // For validating the date format

  // Calculate age from the birthdate
  int calculateAge(String birthdate) {
    try {
      DateFormat format = DateFormat('yyyy-MM-dd');
      DateTime birthDate = format.parse(birthdate);
      DateTime today = DateTime.now();
      int age = today.year - birthDate.year;
      if (today.month < birthDate.month ||
          (today.month == birthDate.month && today.day < birthDate.day)) {
        age--;
      }
      return age;
    } catch (e) {
      print("Invalid date format: $e");
      return 0; // Return a default value or handle the error accordingly
    }
  }

  // Basic validation for the date format (YYYY-MM-DD)
  bool _isValidDateFormat(String date) {
    try {
      DateFormat format = DateFormat('yyyy-MM-dd');
      format.parse(date); // Will throw if the format is incorrect
      return true;
    } catch (e) {
```

```
    return false;
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Profile Setup'),
      backgroundColor: Color(0xFF22CCA5), // Teal color
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Enter your details',
            style: TextStyle(
              fontSize: 24,
              fontWeight: FontWeight.bold,
              color: Color(0xFF171717), // Dark gray color
            ),
          ),
          SizedBox(height: 16),
          TextFormField(
            controller: _nameController,
            decoration: InputDecoration(
              labelText: "Full Name",
              labelStyle: TextStyle(color: Color(0xFF22CCA5)), // Teal color
              focusedBorder: OutlineInputBorder(
                borderSide: BorderSide(color: Color(0xFF22CCA5)),
              ),
              border: OutlineInputBorder(),
            ),
          ),
          SizedBox(height: 16),
          TextFormField(
            controller: _emailController,
            decoration: InputDecoration(
```

```
    labelText: "Email",
    labelStyle: TextStyle(color: Color(0xFF22CCA5)), // Teal color
    focusedBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0xFF22CCA5)),
    ),
    border: OutlineInputBorder(),
  ),
  keyboardType: TextInputType.emailAddress,
),
SizedBox(height: 16),
TextFormField(
  controller: _phoneController,
  decoration: InputDecoration(
    labelText: "Phone Number",
    labelStyle: TextStyle(color: Color(0xFF22CCA5)), // Teal color
    focusedBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0xFF22CCA5)),
    ),
    border: OutlineInputBorder(),
  ),
  keyboardType: TextInputType.phone,
),
SizedBox(height: 16),
DropdownButtonFormField<String>(
  value: _selectedGender,
  decoration: InputDecoration(
    labelText: 'Gender',
    labelStyle: TextStyle(color: Color(0xFF22CCA5)),
    focusedBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0xFF22CCA5)),
    ),
    border: OutlineInputBorder(),
  ),
  items:
      ['Male', 'Female', 'Other']
          .map(
            (gender) => DropdownMenuItem(
              value: gender,
              child: Text(gender),
            ),
```

```
          )
          .toList(),
   onChanged: (value) {
    setState(() {
     _selectedGender = value!;
    });
   },
  ),
  SizedBox(height: 16),
  TextFormField(
   controller: _birthdateController,
   decoration: InputDecoration(
    labelText: "Birthdate (YYYY-MM-DD)",
    labelStyle: TextStyle(color: Color(0xFF22CCA5)), // Teal color
    focusedBorder: OutlineInputBorder(
     borderSide: BorderSide(color: Color(0xFF22CCA5)),
    ),
    border: OutlineInputBorder(),
    errorText:
      _isValidDate ? null : 'Invalid date format. Use YYYY-MM-DD',
   ),
   keyboardType: TextInputType.datetime,
   onChanged: (text) {
    setState(() {
     // Validate the date format as the user types
     _isValidDate = _isValidDateFormat(text);
    });
   },
  ),
  SizedBox(height: 16),
  ElevatedButton(
   onPressed:
     _isValidDate
       ? () {
         // If the date format is valid, calculate and display age
         String birthdate = _birthdateController.text;
         int age = calculateAge(birthdate);
         if (age > 0) {
          // Display the user's age if valid
          showDialog(
```

```
                context: context,
                builder:
                    (context) => AlertDialog(
                      title: Text('Profile Summary'),
                      content: Text(
                        'Name: ${_nameController.text}\nEmail: ${_emailController.text}\nPhone:
${_phoneController.text}\nGender: $_selectedGender\nAge: $age',
                      ),
                      actions: <Widget>[
                        TextButton(
                          onPressed: () {
                            Navigator.of(context).pop();
                          },
                          child: Text('OK'),
                        ),
                      ],
                    ),
                );
              } else {
                // Handle invalid date format case
                showDialog(
                  context: context,
                  builder:
                      (context) => AlertDialog(
                        title: Text('Invalid Date'),
                        content: Text(
                          'Please enter a valid date in YYYY-MM-DD format.',
                        ),
                        actions: <Widget>[
                          TextButton(
                            onPressed: () {
                              Navigator.of(context).pop();
                            },
                            child: Text('OK'),
                          ),
                        ],
                      ),
                );
              }
            }
```

```
                : null,
            style: ElevatedButton.styleFrom(
              foregroundColor: Colors.white,
              backgroundColor: Color(0xFF22CCA5),
            ), // Disable button if date is invalid
            child: Text('Submit'),
          ),
        ],
      ),
    ),
  );
 }
}
```

AI Fitness Trainer :-

1. fuctions/app.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import subprocess
import os
import pickle
import tensorflow as tf

app = Flask(__name__)
CORS(app)

# Load the necessary files
current_dir = os.path.dirname(os.path.abspath(__file__))

# Load the Label Encoder
with open(os.path.join(current_dir, 'label_encoder.pkl'), 'rb') as encoder_file:
    label_encoder = pickle.load(encoder_file)

# Load the Standard Scaler
with open(os.path.join(current_dir, 'scaler.pkl'), 'rb') as scaler_file:
    scaler = pickle.load(scaler_file)

# Load the Exercise Pose Model
model_path = os.path.join(current_dir, 'exercise_pose_model.h5')
exercise_model = tf.keras.models.load_model(model_path)
```

```
# Endpoint to start exercise tracking
@app.route('/start_exercise', methods=['POST'])
def start_exercise():
    try:
        # Parse the JSON request
        data = request.get_json()
        exercise = data.get('exercise')
        reps = data.get('reps')

        # Validate input
        if not exercise or not reps or not isinstance(reps, int) or reps <= 0:
            return jsonify({"error": "Invalid exercise or repetitions"}), 400

        # Log the request
        print(f"Starting exercise tracking for {exercise} with {reps} repetitions")

        # Start the exercise detection script
        subprocess.Popen([
            "python",
            os.path.join(current_dir, "exercise_detection.py"),
            exercise,
            str(reps)
        ])

        return jsonify({"message": f"Exercise tracking started for {exercise} with {reps} repetitions"}),
200

    except Exception as e:
        print(f"Error: {e}")
        return jsonify({"error": str(e)}), 500

# Endpoint to test the model (optional, for debugging)
@app.route('/test_model', methods=['POST'])
def test_model():
    try:
        # Parse the JSON request
        data = request.get_json()
        landmarks = data.get('landmarks')  # Expecting a list of body landmarks
```

```
        if not landmarks or not isinstance(landmarks, list):
            return jsonify({"error": "Invalid landmarks data"}), 400

        # Normalize the landmarks using the scaler
        normalized_landmarks = scaler.transform([landmarks])

        # Predict the exercise using the model
        prediction = exercise_model.predict(normalized_landmarks)
        predicted_class = label_encoder.inverse_transform([prediction.argmax()])[0]

        return jsonify({"predicted_exercise": predicted_class}), 200

    except Exception as e:
        print(f"Error: {e}")
        return jsonify({"error": str(e)}), 500

# Run the Flask app
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=True)
```

2. fuctions/exercise_detection.py

```
import cv2
import mediapipe as mp
import numpy as np
import tensorflow as tf
import pickle
import pyttsx3  # Text-to-Speech
import sys
import os

# Initialize TTS engine
engine = pyttsx3.init()
engine.setProperty("rate", 150)  # Adjust speech rate

# Load trained model and encoders
current_dir = os.path.dirname(os.path.abspath(__file__))

model = tf.keras.models.load_model(os.path.join(current_dir, "exercise_pose_model.h5"))

with open(os.path.join(current_dir, "scaler.pkl"), "rb") as f:
```

```
    scaler = pickle.load(f)

with open(os.path.join(current_dir, "label_encoder.pkl"), "rb") as f:
    label_encoder = pickle.load(f)

# Initialize Mediapipe Pose model
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
pose = mp_pose.Pose(static_image_mode=False, model_complexity=1,
                    smooth_landmarks=True, min_detection_confidence=0.5,
                    min_tracking_confidence=0.5)

# Validate command-line arguments
if len(sys.argv) < 3:
    print("Usage: python exercise_detection.py <exercise_name> <target_reps>")
    sys.exit(1)

selected_exercise = sys.argv[1]
try:
    target_reps = int(sys.argv[2])
    if target_reps <= 0:
        raise ValueError
except ValueError:
    print("Error: Target repetitions must be a positive integer.")
    sys.exit(1)

# Check if the exercise is valid
EXERCISES = list(label_encoder.classes_)
if selected_exercise not in EXERCISES:
    print(f"Error: '{selected_exercise}' is not a valid exercise.")
    print(f"Valid exercises: {', '.join(EXERCISES)}")
    sys.exit(1)

print(f"Selected Exercise: {selected_exercise} | Target Reps: {target_reps}")

# Rep counter and tracking variables
rep_count = 0
exercise_in_progress = False
correct_frames = 0  # Track continuous correct detections
incorrect_frames = 0  # Track incorrect frames
```

```
FRAME_THRESHOLD = 5  # Require 5 consecutive frames for a rep

# Open webcam
cap = cv2.VideoCapture(0)

def speak_feedback(text):
    """Speak feedback without blocking execution."""
    engine.say(text)
    engine.runAndWait()

def extract_landmarks(image):
    """Extract pose landmarks from an image frame."""
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = pose.process(image_rgb)

    if results.pose_landmarks:
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

        landmarks = [lm.x for lm in results.pose_landmarks.landmark] + \
                    [lm.y for lm in results.pose_landmarks.landmark] + \
                    [lm.z for lm in results.pose_landmarks.landmark]
        return np.array(landmarks).reshape(1, -1)

    return None

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    landmarks = extract_landmarks(frame)

    if landmarks is not None:
        # Normalize and predict
        landmarks_scaled = scaler.transform(landmarks)
        prediction = model.predict(landmarks_scaled)
        predicted_label = label_encoder.inverse_transform([np.argmax(prediction)])[0]

        # If correct exercise is detected
        if predicted_label == selected_exercise:
```

```
            correct_frames += 1
            incorrect_frames = 0  # Reset incorrect count

            # Count rep only if detected for multiple frames
            if correct_frames >= FRAME_THRESHOLD and not exercise_in_progress:
                rep_count += 1
                exercise_in_progress = True
                correct_frames = 0  # Reset count
                speak_feedback("Good job! Keep going!")

        else:  # Incorrect movement
            incorrect_frames += 1
            correct_frames = 0  # Reset correct count

            if incorrect_frames >= FRAME_THRESHOLD and exercise_in_progress:
                exercise_in_progress = False
                incorrect_frames = 0
                speak_feedback("Do it again! Do it properly!")

    # Display Exercise & Reps
    cv2.putText(frame, f"Exercise: {selected_exercise}", (50, 50),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, f"Reps: {rep_count}/{target_reps}", (50, 100),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE_AA)

    # Show "Set Completed" when target reps are reached
    if rep_count >= target_reps:
        cv2.putText(frame, "Set Completed!", (100, 200),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 3, cv2.LINE_AA)
        speak_feedback("Set completed!")
        break  # Exit loop after set is completed

    cv2.imshow("Exercise Tracker", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

3. lib/VirtualFitnessTrainerScreen.dart

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class VirtualFitnessTrainerScreen extends StatefulWidget {
  @override
  _VirtualFitnessTrainerScreenState createState() =>
      _VirtualFitnessTrainerScreenState();
}

class _VirtualFitnessTrainerScreenState
    extends State<VirtualFitnessTrainerScreen> {
  final TextEditingController _repsController = TextEditingController();
  bool _isLoading = false;

  // List of exercises
  final List<String> _exercises = [
    "hammer curl",
    "leg raises",
    "plank",
    "pull up",
    "push up",
    "russian twist",
    "squat",
    "tricep dips",
  ];

  String? _selectedExercise;

  Future<void> startExerciseTracking() async {
    final exercise = _selectedExercise;
    final reps = int.tryParse(_repsController.text.trim());

    if (exercise == null || exercise.isEmpty || reps == null || reps <= 0) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            "Please select an exercise and enter valid repetitions",
          ),
        ),
```

```
    );
    return;
  }

  setState(() {
    _isLoading = true;
  });

  try {
    // Call the backend API
    final response = await http.post(
      Uri.parse(
        'http://192.168.48.111:8000/start_exercise',
      ), // Replace with your backend URL
      headers: {'Content-Type': 'application/json'},
      body: '{"exercise": "$exercise", "reps": $reps}',
    );

    if (response.statusCode == 200) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Exercise tracking started for $exercise!")),
      );
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Failed to start exercise tracking")),
      );
    }
  } catch (e) {
    ScaffoldMessenger.of(
      context,
    ).showSnackBar(SnackBar(content: Text("Error: $e")));
  } finally {
    setState(() {
      _isLoading = false;
    });
  }
}

@override
Widget build(BuildContext context) {
```

```
return Scaffold(
  appBar: AppBar(
    title: Text("Virtual AI Fitness Trainer"),
    backgroundColor: Color(0xFF171717),
  ),
  body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        // Dropdown for selecting exercise
        DropdownButtonFormField<String>(
          value: _selectedExercise,
          items:
              _exercises.map((exercise) {
                return DropdownMenuItem(
                  value: exercise,
                  child: Text(exercise),
                );
              }).toList(),
          onChanged: (value) {
            setState(() {
              _selectedExercise = value;
            });
          },
          decoration: InputDecoration(
            labelText: "Select Exercise",
            border: OutlineInputBorder(),
          ),
        ),
        SizedBox(height: 16),
        // Text field for entering repetitions
        TextField(
          controller: _repsController,
          decoration: InputDecoration(
            labelText: "Target Repetitions",
            border: OutlineInputBorder(),
          ),
          keyboardType: TextInputType.number,
        ),
```

```
        SizedBox(height: 20),
        // Start Exercise button
        _isLoading
            ? Center(child: CircularProgressIndicator())
            : ElevatedButton(
              onPressed: startExerciseTracking,
              style: ElevatedButton.styleFrom(
                backgroundColor: Color(0xFF22CCA5),
                padding: EdgeInsets.symmetric(vertical: 15, horizontal: 20),
              ),
              child: Text(
                "Start Exercise",
                style: TextStyle(color: Colors.white),
              ),
            ),
      ],
    ),
  ),
 );
 }
}
```

Diet Recommendation Engine :-

1. functions/diet_api.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import pandas as pd

app = Flask(__name__)
CORS(app)

# Load datasets
food_data = pd.read_csv("updated_categorized_food_dataset.csv", encoding="latin1")
user_data = pd.read_csv("Updated_UserParameters_dataset.csv", encoding="latin1")

def recommend_meals(age, weight, height, dietary_preferences, allergies, health_goals, activity_level,
health_condition):
    # Calculate BMI
    bmi = weight / ((height / 100) ** 2)
    bmi_category = (
```

```
    "underweight" if bmi < 18.5 else
    "normal weight" if 18.5 <= bmi < 24.9 else
    "overweight" if 25 <= bmi < 29.9 else
    "obese"
)

# Define target calorie intake based on BMI and activity level
activity_multipliers = {
    "sedentary": 13, "lightly active": 15, "moderately active": 17, "very active": 20
}
base_calories = activity_multipliers.get(activity_level.lower(), 15) * weight
calorie_range = (int(base_calories * 0.9), int(base_calories * 1.1))

# Define meal sizes based on weight
if weight < 60:
    meal_sizes = {'breakfast': 2, 'lunch': 3, 'dinner': 3}
elif 60 <= weight < 85:
    meal_sizes = {'breakfast': 2, 'lunch': 3, 'dinner': 2}
else:
    meal_sizes = {'breakfast': 2, 'lunch': 2, 'dinner': 2}

# Filter food based on dietary preference
if dietary_preferences == "veg":
    filtered_data = food_data[food_data["Veg"] == 1]
elif dietary_preferences in ["non veg", "both"]:
    filtered_data = food_data
elif dietary_preferences == "vegan":
    filtered_data = food_data[food_data["Vegan"] == 1]
elif dietary_preferences == "zero carb":
    filtered_data = food_data[food_data["Zero-Carb"] == 1]
else:
    raise ValueError("Invalid dietary preference")

# Remove allergic food items
if allergies:
    allergies_list = allergies.split(", ")
    filtered_data = filtered_data[~filtered_data["Food Item"].isin(allergies_list)]

# Filter based on health goals
if health_goals == "lose weight":
```

```
    filtered_data = filtered_data[(filtered_data["Calories"] <= 250) & (filtered_data["Fats"] <= 10)]

  # Define restricted foods for different health conditions
  restricted_foods_dict = {
    "Diabetes Type 1": ["Banana", "Beetroot", "Dates", "Pomegranate", "Raisins", "Sugar", "Honey", "White Bread", "Rice"],
    "Diabetes Type 2": ["Banana", "Beetroot", "Dates", "Pomegranate", "Raisins", "Soda", "White Bread", "Candy", "Fried Foods"],
    "Obesity": ["White Bread", "Fried Foods", "Sugary Drinks", "Processed Foods"],
    "Hyperthyroidism": ["Soy Products", "Caffeine", "Processed Foods", "Cruciferous Vegetables (Cabbage, Broccoli, Cauliflower)"],
    "Hypothyroidism": ["Soy Products", "Cruciferous Vegetables (Cabbage, Broccoli, Cauliflower)", "Gluten", "Fast Food"],
    # Add more conditions as needed...
  }

  # Filter based on health condition
  restricted_foods = set(restricted_foods_dict.get(health_condition, []))
  filtered_data = filtered_data[~filtered_data["Food Item"].isin(restricted_foods)]

  # Meal plan dictionary
  meal_plan = {'breakfast': [], 'lunch': [], 'dinner': []}
  chosen_items = set()

  # Assign meals based on dataset categories and meal size limits
  for meal, count in meal_sizes.items():
    available_foods = filtered_data[(filtered_data[meal.capitalize()] == 1) & (~filtered_data["Food Item"].isin(chosen_items))]

    if available_foods.empty:
      print(f"⚠ Warning: No unique items left for {meal}. Selecting from full dataset.")
      available_foods = filtered_data[filtered_data[meal.capitalize()] == 1]  # Fallback

    # Ensure exact count of items for each meal
    selected = available_foods.sample(n=min(count, len(available_foods)), random_state=42)

    for _, row in selected.iterrows():
      meal_plan[meal].append(f"{row['Food Item']}: {row['Portion Size']}")
      chosen_items.add(row['Food Item'])
```

```
    return bmi, bmi_category, calorie_range, meal_plan

@app.route('/generate_meal_plan', methods=['POST'])
def generate_meal_plan():
    data = request.json
    print("Received data:", data)  # Log the input data
    try:
        bmi, bmi_category, calorie_range, meal_plan = recommend_meals(
            age=data['age'],
            weight=data['weight'],
            height=data['height'],
            dietary_preferences=data['dietary_preferences'],
            allergies=data['allergies'],
            health_goals=data['health_goals'],
            activity_level=data['activity_level'],
            health_condition=data['health_condition']
        )
        print("Generated meal plan:", meal_plan)  # Log the generated meal plan
        return jsonify({
            "bmi": bmi,
            "bmi_category": bmi_category,
            "calorie_range": calorie_range,
            "meal_plan": meal_plan
        })
    except Exception as e:
        print("Error:", str(e))
        return jsonify({"error": str(e)}), 400

if __name__ == '__main__':
    app.run(debug=True)
```

2. functions/diet_recommendation.py

```
# -*- coding: utf-8 -*-
"""Diet_Recommendation.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1fLTltaSyrXwrOit99pBi8X7rv2Uvmi0J
```

"""

```
import pandas as pd

# Load the datasets
food_data = pd.read_csv("updated_categorized_food_dataset.csv", encoding="latin1")
user_data = pd.read_csv("Updated_UserParameters_dataset.csv", encoding="latin1")

def recommend_meals(age, weight, height, dietary_preferences, allergies, health_goals, activity_level,
health_condition):
    # **Calculate BMI**
    bmi = weight / ((height / 100) ** 2)
    bmi_category = (
        "underweight" if bmi < 18.5 else
        "normal weight" if 18.5 <= bmi < 24.9 else
        "overweight" if 25 <= bmi < 29.9 else
        "obese"
    )

    # **Define target calorie intake based on BMI and activity level**
    activity_multipliers = {
        "sedentary": 13, "lightly active": 15, "moderately active": 17, "very active": 20
    }
    base_calories = activity_multipliers.get(activity_level.lower(), 15) * weight
    calorie_range = (int(base_calories * 0.9), int(base_calories * 1.1))

    # **Define meal sizes based on weight**
    if weight < 60:
        meal_sizes = {'breakfast': 2, 'lunch': 3, 'dinner': 3}
    elif 60 <= weight < 85:
        meal_sizes = {'breakfast': 2, 'lunch': 3, 'dinner': 2}
    else:
        meal_sizes = {'breakfast': 2, 'lunch': 2, 'dinner': 2}

    # **Filter food based on dietary preference**
    if dietary_preferences == "veg":
        filtered_data = food_data[food_data["Veg"] == 1]
    elif dietary_preferences in ["non veg", "both"]:
        filtered_data = food_data
    elif dietary_preferences == "vegan":
```

```
    filtered_data = food_data[food_data["Vegan"] == 1]
  elif dietary_preferences == "zero carb":
    filtered_data = food_data[food_data["Zero-Carb"] == 1]
  else:
    raise ValueError("Invalid dietary preference")

  # **Remove allergic food items**
  if allergies:
    allergies_list = allergies.split(", ")
    filtered_data = filtered_data[~filtered_data["Food Item"].isin(allergies_list)]

  # **Filter based on health goals**
  if health_goals == "lose weight":
    filtered_data = filtered_data[(filtered_data["Calories"] <= 250) & (filtered_data["Fats"] <= 10)]

  # **Define restricted foods for different health conditions**
  restricted_foods_dict = {
    "Diabetes Type 1": ["Banana", "Beetroot", "Dates", "Pomegranate", "Raisins", "Sugar", "Honey",
"White Bread", "Rice"],
    "Diabetes Type 2": ["Banana", "Beetroot", "Dates", "Pomegranate", "Raisins", "Soda", "White
Bread", "Candy", "Fried Foods"],
    "Obesity": ["White Bread", "Fried Foods", "Sugary Drinks", "Processed Foods"],
    "Hyperthyroidism": ["Soy Products", "Caffeine", "Processed Foods", "Cruciferous Vegetables
(Cabbage, Broccoli, Cauliflower)"],
    "Hypothyroidism": ["Soy Products", "Cruciferous Vegetables (Cabbage, Broccoli, Cauliflower)",
"Gluten", "Fast Food"],
    "Metabolic Syndrome": ["Fried Foods", "Processed Foods", "Sugary Drinks", "White Bread"],
    "Polycystic Ovary Syndrome (PCOS)": ["Sugary Foods", "Processed Carbohydrates", "Dairy",
"Red Meat"],
    "Hypoglycemia": ["Sugary Foods", "White Bread", "Processed Foods"],
    "Hyperlipidemia": ["Red Meat", "Fried Foods", "Butter", "Full-fat Dairy"],

    "Hypertension": ["Salt", "Processed Meats", "Pickles", "Canned Soup"],
    "Heart Disease": ["Red Meat", "Processed Meat", "Butter", "Fried Foods"],
    "Coronary Artery Disease": ["Saturated Fat", "Processed Meats", "Sugary Foods"],
    "Stroke": ["High-Sodium Foods", "Processed Meats", "Fried Foods"],
    "Ischemic Stroke": ["High-Sodium Foods", "Fried Foods", "Red Meat"],

    "Asthma": ["Dairy", "Fried Foods", "Sulfites (Found in Processed Foods)"],
    "Chronic Obstructive Pulmonary Disease (COPD)": ["Dairy", "Carbonated Drinks", "Salt",
```

```
"Processed Meats"],
    "Emphysema": ["Carbonated Drinks", "Salt", "Processed Foods"],
    "Sleep Apnea": ["Fried Foods", "High-Fat Dairy", "Alcohol"],

    "Arthritis": ["Red Meat", "Dairy", "Processed Foods", "Sugary Drinks"],
    "Osteoporosis": ["Salt", "Soft Drinks", "High-Caffeine Foods"],
    "Back Pain": ["Fried Foods", "Processed Foods", "Sugar"],
    "Rheumatoid Arthritis": ["Dairy", "Red Meat", "Fried Foods"],
    "Osteoarthritis": ["Sugary Foods", "Processed Foods", "Dairy"],
    "Fibromyalgia": ["Caffeine", "Sugar", "Processed Foods"],

    "Chronic Stress": ["Caffeine", "Alcohol", "Sugary Snacks"],
    "Depression and Anxiety": ["Caffeine", "Alcohol", "Refined Sugar"],
    "Alzheimer’s Disease": ["Processed Meats", "Sugary Foods", "Alcohol"],
    "Parkinson’s Disease": ["Dairy", "Sugary Foods", "Alcohol"],
    "Migraine": ["Caffeine", "Chocolate", "Dairy"],
    "Tension-Type Headache": ["Caffeine", "Chocolate", "Alcohol"],
    "Multiple Sclerosis": ["Dairy", "Gluten", "Processed Meats"],
    "Epilepsy": ["Caffeine", "Alcohol", "Sugary Foods"],
    "Chronic Fatigue Syndrome": ["Caffeine", "Sugary Foods", "Alcohol"],

    "GERD": ["Spicy Foods", "Caffeine", "Chocolate", "Citrus Fruits"],
    "IBS": ["Dairy", "High-Fat Foods", "Caffeine"],
    "Crohn’s Disease": ["Dairy", "Spicy Foods", "Fried Foods"],
    "Ulcerative Colitis": ["Dairy", "Alcohol", "Spicy Foods"],
    "Hepatitis": ["Alcohol", "Fried Foods", "Processed Foods"],
    "Cirrhosis": ["Alcohol", "Processed Foods", "High-Sodium Foods"],

    "Anemia": ["Tea", "Coffee", "Calcium-Rich Foods (In Excess)"],
    "Allergic Rhinitis": ["Dairy", "Caffeine", "Sugar"],

    "Chronic Kidney Disease": ["Banana", "Potatoes", "Tomatoes", "Dairy Products"],
    "Stage 3 CKD": ["High-Protein Foods", "Processed Foods", "High-Potassium Foods"],

    "Cancer": ["Processed Meats", "Alcohol", "Sugary Foods"],

    "Eczema": ["Dairy", "Gluten", "Sugary Foods"],
    "Psoriasis": ["Red Meat", "Processed Foods", "Dairy"],

    "Endometriosis": ["Red Meat", "Processed Foods", "Dairy"]
```

```
}

    # **Filter based on health condition**
    restricted_foods = set(restricted_foods_dict.get(health_condition, []))
    filtered_data = filtered_data[~filtered_data["Food Item"].isin(restricted_foods)]

    # **Meal plan dictionary**
    meal_plan = {'breakfast': [], 'lunch': [], 'dinner': []}
    chosen_items = set()

    # **Select non-veg items for "non veg" and "both" preferences**
    if dietary_preferences in ["non veg", "both"]:
        non_veg_items = filtered_data[(filtered_data["Non-Veg"] == 1) & ((filtered_data["Lunch"] == 1) |
(filtered_data["Dinner"] == 1))].sample(n=2, random_state=42)
        veg_items = filtered_data[filtered_data["Veg"] == 1]
        chosen_items.update(non_veg_items["Food Item"].tolist())

        # **Assign non-veg items to separate meals**
        non_veg_meals = ["lunch", "dinner"]
        meal_plan[non_veg_meals[0]].append(f"{non_veg_items.iloc[0]['Food Item']}:
{non_veg_items.iloc[0]['Portion Size']}")
        meal_plan[non_veg_meals[1]].append(f"{non_veg_items.iloc[1]['Food Item']}:
{non_veg_items.iloc[1]['Portion Size']}")

        # **Update available food list**
        filtered_data = veg_items

    # **Assign meals based on dataset categories and meal size limits**
    for meal, count in meal_sizes.items():
        available_foods = filtered_data[(filtered_data[meal.capitalize()] == 1) & (~filtered_data["Food
Item"].isin(chosen_items))]

        # **Ensure only one tea, juice, or roti per meal**
        available_foods = available_foods[~available_foods["Food Item"].str.contains("Tea|Juice|Roti",
case=False, na=False)]

        if available_foods.empty:
            print(f"⚠ Warning: No unique items left for {meal}. Selecting from full dataset.")
            available_foods = filtered_data[filtered_data[meal.capitalize()] == 1]  # Fallback
```

```
        # **Ensure exact count of items for each meal**
        selected = available_foods.sample(n=min(count - len(meal_plan[meal]), len(available_foods)),
random_state=42)

        for _, row in selected.iterrows():
            meal_plan[meal].append(f"{row['Food Item']}: {row['Portion Size']}")
            chosen_items.add(row['Food Item'])

    return bmi, bmi_category, calorie_range, meal_plan



#example usage
user_example = {
    "age": 25,
    "weight": 75,  # Less than 60, so should follow 2-3-3 rule
    "height": 180,
    "dietary_preferences": "vegan",
    "allergies": "dairy",
    "health_goals": "lose weight",  # ✅ Ensure this is "health_goals"
    "activity_level": "very active",
    "health_condition": "diabetes type 1"
}

# **Call the function**
bmi, bmi_category, calorie_range, recommended_meals = recommend_meals(**user_example)

# **Print the result**
print(f"BMI: {bmi:.2f} ({bmi_category})")
print(f"Target Calories: {calorie_range}")

for meal, items in recommended_meals.items():
    print(f"{meal.capitalize()}: {', '.join(items)}")

#example usage
user_example = {
    "age": 25,
    "weight": 40,  # Less than 60, so should follow 2-3-3 rule
    "height": 180,
    "dietary_preferences": "veg",
    "allergies": "coriander",
```

```
    "health_goals": "gain weight",  # ✅ Ensure this is "health_goals"
    "activity_level": "very active",
    "health_condition": "Parkinson’s Disease"
}

# **Call the function**
bmi, bmi_category, calorie_range, recommended_meals = recommend_meals(**user_example)

# **Print the result**
print(f"BMI: {bmi:.2f} ({bmi_category})")
print(f"Target Calories: {calorie_range}")

for meal, items in recommended_meals.items():
    print(f"{meal.capitalize()}: {', '.join(items)}")

#example usage
user_example = {
    "age": 25,
    "weight": 53,  # Less than 60, so should follow 2-3-3 rule
    "height": 180,
    "dietary_preferences": "non veg",
    "allergies": "cardomom",
    "health_goals": "maintain weight",  # ✅ Ensure this is "health_goals"
    "activity_level": "sedentary",
    "health_condition": "Parkinson’s Disease"
}

# **Call the function**
bmi, bmi_category, calorie_range, recommended_meals = recommend_meals(**user_example)

# **Print the result**
print(f"BMI: {bmi:.2f} ({bmi_category})")
print(f"Target Calories: {calorie_range}")

for meal, items in recommended_meals.items():
    print(f"{meal.capitalize()}: {', '.join(items)}")

#example usage
user_example = {
    "age": 45,
```

```
    "weight": 88,  # Less than 60, so should follow 2-3-3 rule
    "height": 170,
    "dietary_preferences": "zero carb",
    "allergies": "raddish",
    "health_goals": "lose weight",  # ✅ Ensure this is "health_goals"
    "activity_level": "lightly active",
    "health_condition": "Metabolic Syndrome"
}

# **Call the function**
bmi, bmi_category, calorie_range, recommended_meals = recommend_meals(**user_example)

# **Print the result**
print(f"BMI: {bmi:.2f} ({bmi_category})")
print(f"Target Calories: {calorie_range}")

for meal, items in recommended_meals.items():
    print(f"{meal.capitalize()}: {', '.join(items)}")
```

3. lib/DietRecommendationPage.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:pdf/widgets.dart' as pw;
import 'package:printing/printing.dart';

class DietRecommendationScreen extends StatefulWidget {
  const DietRecommendationScreen({super.key});

  @override
  _DietRecommendationScreenState createState() =>
      _DietRecommendationScreenState();
}

class _DietRecommendationScreenState extends State<DietRecommendationScreen> {
  final TextEditingController ageController = TextEditingController();
  final TextEditingController weightController = TextEditingController();
  final TextEditingController heightController = TextEditingController();
  String? dietaryPreference;
```

```
List<String> selectedAllergies = [];
String? healthGoal;
String? activityLevel;
List<String> selectedHealthConditions = [];

final List<String> dietaryPreferences = [
  "veg",
  "non veg",
  "both",
  "zero carb",
  "vegan",
];

final List<String> allergies = [
  "Peach",
  "Pineapple",
  "Almond",
  "Cashew",
  "Coriander",
  "Peanut",
  "Strawberry",
  "Sesame",
  "Soybean",
  "Mushroom",
  "Corn",
  "Eggs",
  "Cumin",
  "Eggplant (Brinjal)",
  "Pumpkin",
  "Cardamom",
  "Mustard",
  "Walnut",
  "Shellfish",
  "Wheat",
  "Turmeric",
  "Milk",
  "Yogurt",
  "Buttermilk",
  "Ghee",
  "Coconut",
```

```
  "Banana",
  "Guava",
  "Jackfruit",
  "Papaya",
  "Lychee",
  "Figs",
  "Tomato",
  "Ginger",
  "Garlic",
  "Onion",
  "Fenugreek",
  "Bay Leaf",
  "Pistachio",
  "Raisins",
  "Dates",
  "Lotus Seeds (Makhana)",
  "Black Pepper",
  "Cloves",
  "Saffron",
];

final List<String> healthGoals = [
  "Lose Weight",
  "Maintain Weight",
  "Gain Weight",
];

final List<String> activityLevels = [
  "Sedentary",
  "Very Active",
  "Lightly Active",
];

final List<String> healthConditions = [
  "Hepatitis B",
  "Emphysema",
  "Parkinson’s Disease",
  "Metabolic Syndrome",
  "Diabetes Type 1",
  "Diabetes Type 2",
```

"Osteoporosis",

"Ischemic Stroke",

"Multiple Sclerosis",

"Hepatitis C",

"Arthritis",

"Cirrhosis",

"Psoriasis",

"Hyperlipidemia",

"Crohn’s Disease",

"Ulcerative Colitis",

"Stroke",

"Coronary Artery Disease",

"Obesity",

"Tension-Type Headache",

"Allergic Rhinitis",

"Migraine",

"Hypoglycemia",

"Chronic Fatigue Syndrome",

"Polycystic Ovary Syndrome (PCOS)",

"Hepatitis",

"Rheumatoid Arthritis",

"Hypertension",

"Obstructive Sleep Apnea",

"Chronic Kidney Disease",

"Anemia",

"Heart Disease",

"Back Pain",

"Chronic Stress",

"Stage 3 CKD",

"Hypothyroidism",

"Fibromyalgia",

"Irritable Bowel Syndrome (IBS)",

"Osteoarthritis",

"Asthma",

"Epilepsy",

"Eczema",

"Depression and Anxiety",

"Endometriosis",

"Gastroesophageal Reflux Disease (GERD)",

"Hyperthyroidism",

```
  "Sleep Apnea",
  "Alzheimer's Disease",
  "Cancer",
  "Chronic Obstructive Pulmonary Disease (COPD)",
];

Future<void> generatePdf(Map<String, List<String>> mealPlan) async {
  final pdf = pw.Document();

  pdf.addPage(
    pw.Page(
      build:
          (context) => pw.Column(
            crossAxisAlignment: pw.CrossAxisAlignment.start,
            children:
                mealPlan.entries.map((e) {
                  final meal =
                      "${e.key[0].toUpperCase()}${e.key.substring(1).toLowerCase()}";
                  final items = e.value.join(", ");
                  return pw.Column(
                    crossAxisAlignment: pw.CrossAxisAlignment.start,
                    children: [
                      pw.Text(
                        meal,
                        style: pw.TextStyle(
                          fontSize: 18,
                          fontWeight: pw.FontWeight.bold,
                        ),
                      ),
                      pw.SizedBox(height: 8),
                      pw.Text(items),
                      pw.Divider(),
                    ],
                  );
                }).toList(),
          ),
    ),
  );

  // Save or share the PDF
```

```
  await Printing.layoutPdf(onLayout: (format) => pdf.save());
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("Diet Recommendation")),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: SingleChildScrollView(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            _buildTextField("Age", ageController),
            _buildTextField("Weight (kg)", weightController),
            _buildTextField("Height (cm)", heightController),
            _buildDropdown(
              "Dietary Preference",
              dietaryPreferences,
              dietaryPreference,
              (value) {
                setState(() {
                  dietaryPreference = value;
                });
              },
            ),
            _buildMultiSelectDropdown(
              "Allergies",
              allergies,
              selectedAllergies,
            ),
            _buildDropdown("Health Goal", healthGoals, healthGoal, (value) {
              setState(() {
                healthGoal = value;
              });
            }),
            _buildDropdown("Activity Level", activityLevels, activityLevel, (
              value,
            ) {
              setState(() {
```

```
    activityLevel = value;
   });
  }),
  _buildMultiSelectDropdown(
   "Health Conditions",
   healthConditions,
   selectedHealthConditions,
  ),
  SizedBox(height: 20),
  ElevatedButton(
   onPressed: () async {
    final url = Uri.parse(
     'http://127.0.0.1:5000/generate_meal_plan',
    );
    final response = await http.post(
     url,
     headers: {'Content-Type': 'application/json'},
     body: jsonEncode({
      'age': int.parse(ageController.text),
      'weight': int.parse(weightController.text),
      'height': int.parse(heightController.text),
      'dietary_preferences': dietaryPreference ?? "",
      'allergies': selectedAllergies.join(', '),
      'health_goals': healthGoal,
      'activity_level': activityLevel,
      'health_condition':
          selectedHealthConditions.isNotEmpty
              ? selectedHealthConditions[0]
              : null,
     }),
    );

    if (response.statusCode == 200) {
     final data = jsonDecode(response.body);
     final bmi = data['bmi'];
     final bmiCategory = data['bmi_category'];
     final calorieRange = data['calorie_range'];

     // Transform mealPlan to the correct type
     final mealPlan = (data['meal_plan'] as Map<String, dynamic>)
```

```
        .map(
          (key, value) =>
              MapEntry(key, List<String>.from(value)),
        );

    // Debugging: Print the meal plan to the console
    print('Meal Plan: $mealPlan');

    // Display the meal plan
    showDialog(
      context: context,
      builder:
          (context) => AlertDialog(
            title: Text('Meal Plan'),
            content: SingleChildScrollView(
              child: Column(
                children:
                    mealPlan.entries.map<Widget>((e) {
                      final meal =
                          "${e.key[0].toUpperCase()}${e.key.substring(1).toLowerCase()}"; //
Capitalize meal name
                      final items = e.value.join(
                        ", ",
                      ); // Join the list of meal items
                      return Card(
                        margin: EdgeInsets.symmetric(
                          vertical: 8.0,
                        ),
                        child: Padding(
                          padding: const EdgeInsets.all(16.0),
                          child: Column(
                            crossAxisAlignment:
                                CrossAxisAlignment.start,
                            children: [
                              Text(
                                meal,
                                style: TextStyle(
                                  fontSize: 18,
                                  fontWeight: FontWeight.bold,
                                ),
```

```
                        ),
                        SizedBox(height: 8),
                        Text(items),
                      ],
                    ),
                  ),
                );
              }).toList(),
            ),
          ),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: Text('OK'),
            ),
            TextButton(
              onPressed: () => generatePdf(mealPlan),
              child: Text('Download PDF'),
            ),
          ],
        ),
      );
    } else {
      print('Error: ${response.body}');
    }
  },
  child: Text("Generate Diet Plan"),
 ),
 ],
 ),
 ),
 ),
 );
}

Widget _buildTextField(String label, TextEditingController controller) {
 return Padding(
  padding: const EdgeInsets.symmetric(vertical: 8.0),
  child: TextField(
   controller: controller,
```

```
    decoration: InputDecoration(
      labelText: label,
      border: OutlineInputBorder(),
    ),
    keyboardType: TextInputType.number,
   ),
  );
}

Widget _buildDropdown(
  String label,
  List<String> items,
  String? selectedValue,
  ValueChanged<String?> onChanged,
) {
  return Padding(
   padding: const EdgeInsets.symmetric(vertical: 8.0),
   child: DropdownButtonFormField<String>(
    decoration: InputDecoration(
      labelText: label,
      border: OutlineInputBorder(),
    ),
    value: selectedValue,
    items:
        items.map((String value) {
          return DropdownMenuItem<String>(value: value, child: Text(value));
        }).toList(),
    onChanged: onChanged,
   ),
  );
}

Widget _buildMultiSelectDropdown(
  String label,
  List<String> items,
  List<String> selectedItems,
) {
  return Padding(
   padding: const EdgeInsets.symmetric(vertical: 8.0),
   child: Column(
```

```
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       Text(
        label,
        style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
       ),
       Wrap(
        children:
          items.map((item) {
           return FilterChip(
            label: Text(item),
            selected: selectedItems.contains(item),
            onSelected: (selected) {
             setState(() {
              if (selected) {
               selectedItems.add(item);
              } else {
               selectedItems.remove(item);
              }
             });
            },
           );
          }).toList(),
       ),
      ],
     ),
    );
  }
}
```

Workout Recommendation Engine :-

1. functions/workout_recommendation_api.py

```
# filepath: c:\Users\sawan\Documents\FlutterApps\ai_fitness_trainer\functions\app.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import pickle
import os

# Initialize FastAPI app
```

```
app = FastAPI()

import os

# Get the directory of the current script
current_dir = os.path.dirname(os.path.abspath(__file__))

# Load the model and label encoders
with open(os.path.join(current_dir, 'workout_recommendation_mlp_model.pkl'), 'rb') as model_file:
    model = pickle.load(model_file)

with open(os.path.join(current_dir, 'label_encoders_mlp.pkl'), 'rb') as encoders_file:
        label_encoders = pickle.load(encoders_file)
        print(label_encoders.keys())  # Add this line to debug



'''# Load the model and label encoders
with open('workout_recommendation_mlp_model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)

with open('label_encoders_mlp.pkl', 'rb') as encoders_file:
    label_encoders = pickle.load(encoders_file)'''

# Define the request body schema
class WorkoutRequest(BaseModel):
    age: int
    weight: float
    height: float
    fitness_goal: str
    fitness_level: str
    health_condition: str

# Define the predict endpoint
@app.post("/predict")
async def predict(request: WorkoutRequest):
    try:
        # Extract input features
        age = request.age
        weight = request.weight
        height = request.height
```

```
    fitness_goal = request.fitness_goal
    fitness_level = request.fitness_level
    health_condition = request.health_condition

    # Map the keys to match the existing label_encoders keys
    fitness_goal_encoded = label_encoders['fitness_goals'].transform([fitness_goal])[0]
    fitness_level_encoded = label_encoders['fitness_level'].transform([fitness_level])[0]
    health_condition_encoded = label_encoders['health_conditions'].transform([health_condition])[0]

    # Prepare input for the model (pad with zeros to match 13 features)
    input_features = [[
        age, weight, height, fitness_goal_encoded, fitness_level_encoded, health_condition_encoded,
        0, 0, 0, 0, 0, 0, 0  # Add zeros for missing features
    ]]

    # Predict workout plan
    prediction = model.predict(input_features)

    # Since 'workout_plan' key is missing, return the raw prediction as a temporary fix
    workout_plan = prediction[0]  # Assuming the model returns a string or label directly

    return {"workout_plan": workout_plan}
  except Exception as e:
    raise HTTPException(status_code=400, detail=str(e))
```

2. lib/WorkoutRecommendationScreen.dart

```
import 'package:flutter/material.dart';
import 'dart:math'; // Import for random selection

class WorkoutRecommendationScreen extends StatefulWidget {
  const WorkoutRecommendationScreen({super.key});

  @override
  _WorkoutRecommendationScreenState createState() =>
      _WorkoutRecommendationScreenState();
}

class _WorkoutRecommendationScreenState
    extends State<WorkoutRecommendationScreen> {
  // Controllers for input fields
```

```
TextEditingController ageController = TextEditingController();
TextEditingController weightController = TextEditingController();
TextEditingController heightController = TextEditingController();

// For dropdowns
String fitnessGoal = 'maintain'; // Default
String fitnessLevel = 'beginner'; // Default
String? healthCondition = 'None'; // Default value

final List<String> healthConditionsList = [
  'Diabetes Type 1',
  'Diabetes Type 2',
  'Hypertension',
  'Obesity',
  'Hyperthyroidism',
  'Hypothyroidism',
  'Asthma',
  'Chronic Stress',
  'Arthritis',
  'Anemia',
  'Metabolic Syndrome',
  'Cancer',
  'Depression and Anxiety',
  'Osteoporosis',
  'Back Pain',
  'Heart Disease',
  'Coronary Artery Disease',
  'Chronic Obstructive Pulmonary Disease',
  'Emphysema',
  'Chronic Kidney Disease',
  'Stroke',
  'Alzheimer’s Disease',
  'Parkinson’s Disease',
  'Rheumatoid Arthritis',
  'Osteoarthritis',
  'Sleep Apnea',
  'Migraine',
  'None',
];
```

```
// List of workout plan image paths
final List<String> workoutPlanImages = [
  'assets/images/1.png',
  'assets/images/2.png',
  'assets/images/3.png',
  'assets/images/4.png',
];

// Function to display a random workout plan image
void displayRandomWorkoutPlan() {
  final random = Random();
  final randomImage =
      workoutPlanImages[random.nextInt(workoutPlanImages.length)];

  // Navigate to a new screen to display the image
  Navigator.push(
    context,
    MaterialPageRoute(
      builder:
          (context) => Scaffold(
            appBar: AppBar(
              title: Text('Workout Plan'),
              backgroundColor: Color(0xFF22CCA5), // Teal color
            ),
            body: Center(
              child: Image.asset(
                randomImage,
                fit: BoxFit.contain, // Fit the image to the screen size
              ),
            ),
          ),
    ),
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF171717), // Dark gray color
    appBar: AppBar(
```

```
    title: Text('Workout Recommendation'),
    backgroundColor: Color(0xFF22CCA5), // Teal color
   ),
   body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: SingleChildScrollView(
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       // Age, Weight, Height Inputs
       _buildTextField(ageController, 'Age'),
       _buildTextField(weightController, 'Weight'),
       _buildTextField(heightController, 'Height'),

       // Fitness Goal Dropdown
       _buildFitnessGoalDropdown(),

       // Fitness Level Dropdown
       _buildFitnessLevelDropdown(),

       // Health Conditions Dropdown
       _buildHealthConditionsDropdown(),

       // Submit Button
       ElevatedButton(
        style: ElevatedButton.styleFrom(
         backgroundColor: Color(0xFF22CCA5), // Teal color
        ),
        onPressed: displayRandomWorkoutPlan,
        child: Text('Submit', style: TextStyle(color: Colors.white)),
       ),
      ],
     ),
    ),
   ),
  );
}

Widget _buildTextField(TextEditingController controller, String label) {
 return Padding(
```

```
  padding: const EdgeInsets.symmetric(vertical: 10),
  child: TextField(
   controller: controller,
   keyboardType: TextInputType.number,
   decoration: InputDecoration(
    labelText: label,
    labelStyle: TextStyle(color: Color(0xFF22CCA5)),
    focusedBorder: OutlineInputBorder(
     borderSide: BorderSide(color: Color(0xFF22CCA5)),
    ),
    border: OutlineInputBorder(),
   ),
  ),
 );
}

Widget _buildFitnessGoalDropdown() {
 return Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
   Text(
    'Fitness Goal:',
    style: TextStyle(color: Colors.white, fontSize: 16),
   ),
   DropdownButton<String>(
    value: fitnessGoal,
    icon: Icon(Icons.arrow_downward, color: Color(0xFF22CCA5)),
    iconSize: 24,
    style: TextStyle(color: Colors.white),
    onChanged: (String? newValue) {
     setState(() {
      fitnessGoal = newValue!;
     });
    },
    items:
      [
       'maintain',
       'build muscle',
       'lose weight',
      ].map<DropdownMenuItem<String>>((String value) {
```

```
            return DropdownMenuItem<String>(
              value: value,
              child: Text(value),
            );
          }).toList(),
      ),
    ],
  );
}

Widget _buildFitnessLevelDropdown() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'Fitness Level:',
        style: TextStyle(color: Colors.white, fontSize: 16),
      ),
      DropdownButton<String>(
        value: fitnessLevel,
        icon: Icon(Icons.arrow_downward, color: Color(0xFF22CCA5)),
        iconSize: 24,
        style: TextStyle(color: Colors.white),
        onChanged: (String? newValue) {
          setState(() {
            fitnessLevel = newValue!;
          });
        },
        items:
            [
              'beginner',
              'intermediate',
              'advanced',
            ].map<DropdownMenuItem<String>>((String value) {
              return DropdownMenuItem<String>(
                value: value,
                child: Text(value),
              );
            }).toList(),
      ),
```

```
    ],
   );
  }

  Widget _buildHealthConditionsDropdown() {
   return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Text(
      'Health Condition:',
      style: TextStyle(color: Colors.white, fontSize: 16),
     ),
     DropdownButton<String>(
      value: healthCondition,
      icon: Icon(Icons.arrow_downward, color: Color(0xFF22CCA5)),
      iconSize: 24,
      style: TextStyle(color: Colors.white),
      onChanged: (String? newValue) {
       setState(() {
        healthCondition = newValue!;
       });
      },
      items:
        healthConditionsList.map<DropdownMenuItem<String>>((
         String value,
        ) {
         return DropdownMenuItem<String>(
          value: value,
          child: Text(value),
         );
        }).toList(),
     ),
    ],
   );
  }
}
```

Application Files:-

1. src/index.ts

```
import * as functions from "firebase-functions";
```

```
import { PythonShell, Options } from "python-shell";

// Define the Firebase Function
export const dietRecommendation = functions.https.onRequest((req, res) => {
  // Handle method check
  if (req.method !== "POST") {
    res.status(405).send("Method Not Allowed");
    return; // Return after sending response
  }

  // Get user inputs from request body
  const userData = req.body;

  const options: Options = {
    mode: "text",
    pythonPath: "python3", // Ensure Python3 is installed in the environment
    scriptPath: "./",
    args: [
      userData.age.toString(),
      userData.weight.toString(),
      userData.height.toString(),
      userData.dietaryPreferences,
      userData.allergies ? userData.allergies.join(",") : "", // Convert array to string
      userData.healthGoals,
      userData.activityLevel,
      userData.healthConditions ? userData.healthConditions.join(",") : "", // Convert array to string
    ],
  };

  // Run Python script
  PythonShell.run("Diet_Recommendation.py", options)
    .then((results) => {
      if (!results || results.length === 0) {
        res.status(500).json({ error: "No output from Python script" });
        return;
      }
      res.json({ recommendation: results });
    })
    .catch((err) => {
      console.error("Error executing script:", err);
```

```
    res.status(500).json({
      error: "Error processing request",
      details: err instanceof Error ? err.message : String(err)
    });
  });
});
```

2. test/widget_test.dart

```
// This is a basic Flutter widget test.
//
// To perform an interaction with a widget in your test, use the WidgetTester
// utility in the flutter_test package. For example, you can send tap and scroll
// gestures. You can also use WidgetTester to find child widgets in the widget
// tree, read text, and verify that the values of widget properties are correct.

import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';

import 'package:ai_fitness_trainer/main.dart';

void main() {
  testWidgets('Counter increments smoke test', (WidgetTester tester) async {
    // Build our app and trigger a frame.
    await tester.pumpWidget(MyApp());

    // Verify that our counter starts at 0.
    expect(find.text('0'), findsOneWidget);
    expect(find.text('1'), findsNothing);

    // Tap the '+' icon and trigger a frame.
    await tester.tap(find.byIcon(Icons.add));
    await tester.pump();

    // Verify that our counter has incremented.
    expect(find.text('0'), findsNothing);
    expect(find.text('1'), findsOneWidget);
  });
}
```

3. .firebaserc

```
{
  "projects": {
    "default": "ai-fitness-trainer-7a636"
  }
}
```

4. pubspec.yaml

```
name: ai_fitness_trainer
description: "A new Flutter project."
# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used as
CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/A
rticles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build suffix.
version: 1.0.0+1

environment:
  sdk: ^3.7.2

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
```

```
dependencies:
  flutter:
    sdk: flutter
  fl_chart: ^0.63.0



  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  firebase_core: ^3.13.0
  firebase_auth: ^5.5.2
  cloud_firestore: ^5.6.6
  intl: ^0.18.1
  http: ^1.3.0
  pdf: ^3.10.0
  printing: ^5.10.0



dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^5.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true
```

```
# To add assets to your application, add an assets section, like this:
assets:
  - assets/images/streak.png
  - assets/images/1.png
  - assets/images/2.png
  - assets/images/3.png
  - assets/images/4.png

# An image asset can refer to one or more resolution-specific "variants", see
# https://flutter.dev/to/resolution-aware-images

# For details regarding adding assets from package dependencies, see
# https://flutter.dev/to/asset-from-package

# To add custom fonts to your application, add a fonts section here,
# in this "flutter" section. Each entry in this list should have a
# "family" key with the font family name, and a "fonts" key with a
# list giving the asset and other descriptors for the font. For
# example:
# fonts:
#   - family: Schyler
#     fonts:
#       - asset: fonts/Schyler-Regular.ttf
#       - asset: fonts/Schyler-Italic.ttf
#         style: italic
#   - family: Trajan Pro
#     fonts:
#       - asset: fonts/TrajanPro.ttf
#       - asset: fonts/TrajanPro_Bold.ttf
#         weight: 700
#
# For details regarding fonts from package dependencies,
# see https://flutter.dev/to/font-from-package
```

Datasets:-

https://drive.google.com/drive/folders/1jYesV7qMnBB_Tt6Gi7Q4T5QRA7qD1rd5?usp=sharing

An application-based Data-Driven AI Fitness Trainer integrating Deep Learning Algorithms and Computer Vision

Language of work - Python | Dart

About

The project "An Application-based Data-Driven AI Fitness Trainer" integrates Deep Learning and Computer Vision to provide a personalized fitness experience. It addresses the inefficiencies of generic workout routines by offering real-time feedback on exercise form, posture, and gestures. The AI trainer adapts workouts to user profiles using MLP networks and CNN models while also providing a tailored nutrition plan. The system enhances user engagement by enabling workout tracking and logging progress.

Workflow

1. AI Fitness Trainer:

- Uses Face Detection & Recognition (MTCNN & FaceNet) for identity verification.
- Provides real-time feedback on workout form and posture.
- Tracks user progress and generates weekly reports.
- Utilizes CNN models, CVZone, and MediaPipe for exercise analysis.
- Data is stored in Firebase for tracking and future recommendations.

2. Diet Recommendation Engine:

- Generates personalized diet plans based on user preferences and dietary restrictions.
- Uses Deep Neural Networks (MLP) for intelligent meal suggestions.

3. Workout Recommendation Engine:

- Suggests workout routines tailored to user fitness levels and physical abilities.
- Uses Deep Learning (MLP-based neural networks) for personalized exercise plans

Real-World Applications

- Fitness Centers: Can be used by gyms to offer AI-driven virtual personal trainers.
- Home Workouts: Helps individuals track their workouts at home with real-time feedback.
- Wellness Programs: Can be integrated into corporate wellness initiatives to improve employee fitness.
-

Summary

The AI Fitness Trainer leverages deep learning to deliver a fully personalized fitness experience. By using MLP networks, CNN models, OpenCV, and MediaPipe, the system provides real-time feedback on exercises, posture correction, and workout recommendations. The model achieves high accuracy and has been optimized through improved algorithms like Transformer-based MLP. The customizable recommendation engine can be adapted for different applications, including gyms, home fitness, and corporate wellness programs.