**FITNESS TRACKER**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology/Master of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**K.Sai Siddhartha-(AP23110010560)**

**P.Hemanth Patel-(AP23110010559)**

**T.Nandan-(AP23110010526)**

**V.Karthikeya-(AP23110010533)**

**A picture containing text

Description automatically generated**

Under the Guidance of

**(Dr.Kavitha Rani )**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[November, 2024]**

**Certificate**

Date: 18-Nov-24

This is to certify that the work present in this Project entitled “FITNESS TRACKER” has been carried out by **K.Sai Siddhartha,P.Hemanth,T.Nandan,V.karithkeya** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

**Supervisor,**

Dr. kavathi Rani

SRM UNIVERSITY,AP

**Acknowledgments:**

I would like to thank my teacher, Dr . Kavitha Rani,for giving me the opportunity to work on this project. This project taught me a lot about different core concepts of ObjectOriented Programming, such as data encapsulation and static member variables``, exception handling, objects, and classes etc. I am extremely grateful and express my profound gratitude and indebtedness to my project guide teacher for the kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.No** | **CONTENTS** | **PAGE NUMBER** |
| 1 | Title Page | 1 |
| 2 | Certificate | 2 |
| 3 | Acknowledgement | 3 |
| 4 | Abstract | 5 |
| 5 | Contributions | 6 |
| 6 | Introduction | 7 |
| 7 | Functionalities | 8 |
| 8 | Objective | 11 |
| 9 | Code Implementation | 13 |
| 10 | Conclusion | 26 |

# Abstract

The Fitness Tracker application is designed to assist users in monitoring and managing their exercise routines. The program allows users to create profiles, log exercise sessions, set exercise goals, view exercise history, and save data to a file. Predefined exercises with corresponding calorie burn rates are available for users to choose from. User profiles store exercise details, track total duration, total calories burned, and user-specific exercise goals. The program employs object-oriented principles with classes such as Exercise, ExerciseGoal, and UserProfile. The main application, FitnessTracker, enables user interactions through a console-based menu system. The modular design ensures scalability and ease of maintenance, providing a comprehensive fitness tracking solution for users.

**CONTRIBUTIONS:**

File handling and main function-K.Sai Siddhartha

Class Exercise Goal and userprofile -P.Hemanth

Class exercise-V.karthikeya

Class fitness tracker and report-T.Nandan

# Introduction

In the pursuit of a healthier and more active lifestyle, the Fitness Tracker project stands as a testament to the convergence of technology, health, and user-centric design. This project was conceived with the objective of creating an advanced fitness application that goes beyond the conventional, offering users a comprehensive and intuitive platform to manage their exercise routines effectively. As we delve into this project report, we will unravel the intricacies of the Fitness Tracker, exploring the underlying technologies, design principles, and the collaborative efforts that brought this innovative solution to fruition.

**Project Overview:**

The Fitness Tracker project was initiated with a clear vision — to develop a tool that not only records exercise data but also serves as a dynamic companion on the fitness journey. With a focus on leveraging technology to empower users, the project aimed to create an application that seamlessly integrates into users' lives, providing insights, motivation, and a personalized approach to fitness.

**Functionalities**

Let's break down the functionality of each class in the Fitness Tracker application:

**1. Exercise Class:**

- Purpose: Represents an exercise logged by the user.

- Attributes:

- name: Name of the exercise.

- duration: Duration of the exercise in minutes.

- caloriesBurned: Calculated calories burned based on the exercise type and duration.

- Methods:

- calculateCaloriesBurned(): Computes the calories burned based on the exercise type and duration.

- getCaloriesBurnRateByName(): Retrieves the predefined calorie burn rate for a given exercise.

**2. ExerciseGoal Class:**

- Purpose: Represents a user's exercise goal.

- Attributes:

- type: Type of exercise goal (e.g., "Cardio," "Strength").

- target: Target duration for the exercise goal in minutes.

- timeframe: Timeframe for achieving the exercise goal (e.g., "Day," "Week").

**3. UserProfile Class:**

- Purpose: Represents a user profile, storing exercise data and goals.

- Attributes:

- username: User's unique identifier.

- exercises: Array to store Exercise objects.

- exerciseCount: Number of exercises logged by the user.

- totalDuration: Total duration of all logged exercises.

- totalCaloriesBurned: Total calories burned from all logged exercises.

- exerciseGoal: ExerciseGoal object representing the user's exercise goal.

- Methods:

- addExercise(exercise): Adds an exercise to the user's profile, updating total duration and calories burned.

- setExerciseGoal(goal): Sets the user's exercise goal.

- displaySummary(): Displays a summary of the user's exercises, total duration, total calories burned, and exercise goal.

- saveDataToFile(): Saves the user's exercise data to a file.

- loadDataFromFile(): Loads exercise data from a file.

**4. FitnessTracker Class:**

- PurposeManages user profiles and provides overall application functionality.

- Attributes:

- userProfiles: Array to store UserProfile objects.

- userProfileCount: Number of user profiles.

- Methods:

- createUserProfile(username): Creates a new user profile.

- getUserProfile(username): Retrieves a user profile by username.

- displayAllUserSummaries(): Displays summaries for all user profiles.

**5. Main Function:**

- Purpose: Implements the main logic of the application, including user interaction and menu options.

- Functionality:

- User authentication (login or profile creation).

- Navigation through application options (logging exercises, setting goals, viewing summaries, etc.).

- Saving and loading data to/from files.

- Interaction with UserProfile and FitnessTracker classes.

The classes work together to create a comprehensive fitness tracking system, allowing users to log exercises, set goals, view summaries, and track their progress over time. The use of classes and object-oriented principles contributes to a modular, organized, and extensible design.

**Objectives:**

Holistic Fitness Management:

The Fitness Tracker is not merely a digital logbook; it's a holistic fitness management system. From logging exercises and setting personalized goals to analyzing historical data, it serves as a reliable companion in achieving and maintaining a healthy lifestyle.

Intuitive User Interface:

Navigate effortlessly through a thoughtfully designed user interface. The clear and intuitive menu system ensures a seamless experience for users of all levels, providing accessibility without compromising on functionality.

Data Persistence and Continuity:

Exercise data is not ephemeral. The application employs robust data persistence mechanisms, securely storing user information in files. This ensures that your fitness journey remains continuous, allowing for insightful reflections on progress.

## 

**Technical Brilliance:**

Object-Oriented Design (OOP):

At the heart of the Fitness Tracker is a robust implementation of Object-Oriented Programming (OOP) principles. Classes such as Exercise, ExerciseGoal, UserProfile, and FitnessTracker encapsulate functionalities, fostering code modularity and extensibility.

Error-Handling Resilience:

Emphasizing a commitment to user experience, the application incorporates advanced error-handling mechanisms. This ensures a reliable and resilient performance, even in the face of unexpected events.

Interactive User Experience:

Beyond functionality, the Fitness Tracker aims to engage users actively in their fitness management. The interactive user experience extends from logging exercises to setting and achieving personalized goals, creating a symbiotic relationship between technology and wellness.

Inspirations and Influences:

The application draws inspiration from leading fitness platforms such as FitBit and MyFitnessPal. This influence contributes to the creation of a tool that amalgamates industry best practices with innovative features.

Educational Support:

Acknowledging the learning aspect, the Fitness Tracker owes a debt to educational platforms such as Codecademy, which provided comprehensive resources and insights, aiding in the development journey.

Team Collaboration:

The realization of the Fitness Tracker was not a solitary endeavor. The collaborative efforts of our team made it possible. All team members played a pivotal role in shaping the application into what it is today.

**Code Implementation**

#include <iostream>

#include <fstream>

#include <string>

#include<stdlib.h>

using namespace std;

namespace FitnessTracker {

// Predefined exercises (name, calorie burn rate)

static const int Ex\_size = 15;

static string predefinedExerciseNames[Ex\_size] = {

"Running", "Cycling", "Swimming", "Jumping Jacks", "Weightlifting",

"Yoga", "Hiking", "Rowing", "Jump Rope", "Kickboxing",

"Pilates", "Zumba", "Aerobics", "Circuit Training", "Elliptical Training"

};

static double predefinedExerciseRates[Ex\_size] = {

10.0, 8.0, 12.0, 6.0, 5.0, 3.0, 7.0, 9.0, 11.0, 13.0, 4.0, 7.0, 6.0, 9.0, 8.0

};

class Exercise {

public:

Exercise() : name(""), duration(0) {} // Default constructor

Exercise(string name, int duration) : name(name), duration(duration) {

caloriesburned();

}

string getName() const {

return name;

}

int getDuration() const {

return duration;

}

double getCaloriesBurned() const {

return caloriesBurned;

}

private:

string name;

int duration;

double caloriesBurned;

void caloriesburned() {

const double caloriesBurnedRate = getCaloriesBurnRateByName(name);

caloriesBurned = duration \* caloriesBurnedRate;

}

double getCaloriesBurnRateByName(const string& exerciseName) const {

for (int i = 0; i < Ex\_size; ++i) {

if (predefinedExerciseNames[i] == exerciseName) {

return predefinedExerciseRates[i];

}

}

cout << "Exercise not found in predefined list. Using default calorie burn rate.\n";

return 7.0; // Default calorie burn rate (adjust as needed)

}

};

class ExerciseGoal {

public:

ExerciseGoal(string type, int target, string timeframe)

: type(type), target(target), timeframe(timeframe) {}

string getType() const {

return type;

}

int getTarget() const {

return target;

}

string getTimeframe() const {

return timeframe;

}

private:

string type;

int target;

string timeframe;

};

class UserProfile {

public:

UserProfile(string username = "") : username(username), exerciseCount(0), totalDuration(0), totalCaloriesBurned(0.0) {}

void addExercise(const Exercise& exercise) {

if (exerciseCount < MAX\_EXERCISES) {

exercises[exerciseCount++] = exercise;

totalDuration += exercise.getDuration();

totalCaloriesBurned += exercise.getCaloriesBurned();

} else {

cerr << "Exercise limit reached.\n";

}

}

void Setexercisegoal(const ExerciseGoal& goal) {

exerciseGoal = goal;

cout << "Exercise goal set: " << goal.getTarget() << " minutes per " << goal.getTimeframe() << endl;

}

void displaySummary() const {

cout << "Fitness Tracker for: " << username << "\n";

cout << "Exercises:\n";

for (int i = 0; i < exerciseCount; ++i) {

const Exercise& exercise = exercises[i];

cout << "Exercise: " << exercise.getName() << ", Duration: " << exercise.getDuration() << " minutes, Calories Burned: " << exercise.getCaloriesBurned() << " calories\n";

}

cout << "Total Duration: " << totalDuration << " minutes\n";

cout << "Total Calories Burned: " << totalCaloriesBurned << " calories\n";

if (exerciseGoal.getTarget() > 0) {

cout << "Exercise Goal: " << exerciseGoal.getTarget() << " minutes per " << exerciseGoal.getTimeframe() << endl;

}

}

void saveDataToFile() const {

ofstream outputFile(username + ".txt");

if (outputFile.is\_open()) {

outputFile << "Exercise,Duration,Calories Burned\n";

for (int i = 0; i < exerciseCount; ++i) {

const Exercise& exercise = exercises[i];

outputFile << exercise.getName() << "," << exercise.getDuration() << "," << exercise.getCaloriesBurned() << "\n";

}

outputFile << "\nTotal Duration," << "Total Calories Burned\n";

outputFile << totalDuration << "," << totalCaloriesBurned << "\n";

outputFile.close();

cout << "Data saved to file.\n";

} else {

cerr << "Failed to save data to file.\n";

}

}

string getUsername() const {

return username;

}

private:

static const int MAX\_EXERCISES = 100; // Maximum exercises that can be stored

string username;

Exercise exercises[MAX\_EXERCISES];

int exerciseCount;

int totalDuration;

double totalCaloriesBurned;

ExerciseGoal exerciseGoal{"", 0, ""};

};

class FitnessTracker {

public:

void createUserProfile(const string& username) {

if (userProfileCount < MAX\_USERS) {

userProfiles[userProfileCount++] = UserProfile(username);

} else {

cerr << "Maximum user limit reached. Cannot create a new profile.\n";

}

}

UserProfile\* getUserProfile(const string& username) {

for (int i = 0; i < userProfileCount; ++i) {

if (userProfiles[i].getUsername() == username) {

return &userProfiles[i];

}

}

return nullptr;

}

void displayAllUserSummaries() const {

for (int i = 0; i < userProfileCount; ++i) {

const UserProfile& user = userProfiles[i];

user.displaySummary();

cout << "-----------------------------\n";

}

}

private:

static const int MAX\_USERS = 100; // Maximum number of users

UserProfile userProfiles[MAX\_USERS];

int userProfileCount = 0;

};

} // namespace FitnessTracker

int main() {

FitnessTracker::FitnessTracker fitnessTracker;

here:

while (true) {

cout << "\nOptions:\n";

cout << "1. Log in\n";

cout << "2. Create a new profile\n";

cout << "3. Display All summaries\n";

cout << "4. Exit\n";

int loginChoice;

cout << "Enter your choice: ";

cin >> loginChoice;

system("CLS");

if(loginChoice>3){

cout<<"Invalid selection."<<endl;

goto here;

}

if (loginChoice == 4) {

break;

}

if (loginChoice==3) {

fitnessTracker.displayAllUserSummaries();

goto here;

}

string username;

cout << "Enter your username: ";

cin >> username;

FitnessTracker::UserProfile\* user = fitnessTracker.getUserProfile(username);

if (loginChoice == 1) {

system("CLS");

if (user) {

cout << "Logged in as " << username << ".\n";

} else {

cerr<< "User not found. Please create a new profile.\n";

continue;

}

} else if (loginChoice == 2) {

system("CLS");

if (!user) {

fitnessTracker.createUserProfile(username);

user = fitnessTracker.getUserProfile(username);

cout << "New profile created for " << username << ".\n";

} else {

cerr << "User already exists. Please log in or choose a different username.\n";

continue;

}

}

else {

cerr << "Invalid choice. Exiting...\n";

break;

}

while (true) {

cout << "\nOptions:\n";

cout << "1. Log an exercise session\n";

cout << "2. View exercise history\n";

cout << "3. Set exercise goals\n";

cout << "4. Save data to a file\n";

cout << "5. Log out\n";

cout << "6. Exit\n";

int choice;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

system("CLS");

cout << "\nSelect an exercise:\n";

for (int i = 0; i < FitnessTracker::Ex\_size; ++i) {

cout << i + 1 << ". " << FitnessTracker::predefinedExerciseNames[i] << endl;

}

int exerciseChoice;

cout << "Enter the number corresponding to the exercise: ";

cin >> exerciseChoice;

if (exerciseChoice >= 1 && exerciseChoice <= FitnessTracker::Ex\_size) {

string selectedExercise = FitnessTracker::predefinedExerciseNames[exerciseChoice - 1];

int duration;

cout << "Enter exercise duration (minutes): ";

cin >> duration;

FitnessTracker::Exercise exercise(selectedExercise, duration);

user->addExercise(exercise);

} else {

cout << "Invalid exercise selection.\n";

}

break;

}

case 2:

system("CLS");

user->displaySummary();

break;

case 3: {

system("CLS");

string type;

int target;

string timeframe;

cout << "Enter exercise goal type: ";

cin >> type;

cout << "Enter exercise goal target (minutes): ";

cin >> target;

cout << "Enter exercise goal timeframe: ";

cin >> timeframe;

FitnessTracker::ExerciseGoal goal(type, target, timeframe);

user->Setexercisegoal(goal);

break;

}

case 4:

system("CLS");

user->saveDataToFile();

break;

case 5:

system("CLS");

user->saveDataToFile();

cout << "Logged out.\n";

break;

case 6:

system("CLS");

user->saveDataToFile();

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

if (choice == 5) {

system("CLS");

user->saveDataToFile();

break;

}

}

}

return 0

}

**Conclusion:**

In conclusion, the Fitness Tracker code represents a robust and versatile solution designed to empower users on their fitness journey. The implementation of object-oriented programming principles, including the use of classes and encapsulation, fosters a modular and extensible codebase. This design choice allows for easy scalability, enabling the addition of new features and enhancements in the future.

The code successfully integrates a variety of functionalities, ranging from exercise logging and goal setting to data visualization and file handling. The incorporation of real-world exercise data and the option for users to customize their exercise routines adds a layer of practicality and personalization to the application.

The introduction of user profiles and the ability to manage multiple users enhance the code's usability in a real-world scenario, catering to a diverse user base. The seamless integration of user authentication, exercise logging, and goal setting provides a holistic fitness management experience.

The code's use of namespaces and adherence to best practices, including encapsulation and code modularity, contributes to its readability and maintainability. Furthermore, the incorporation of error handling mechanisms ensures a more robust and user-friendly application.

As we reflect on the Fitness Tracker code, it is evident that it not only meets the foundational requirements of a fitness tracking application but also lays the groundwork for potential future expansions. Whether utilized by individuals aiming for a healthier lifestyle or developers seeking to build upon its foundation, the Fitness Tracker code stands as a testament to the harmonious fusion of technology and fitness management.