Fitness Tracker

A Terminal Based Tracker

CIS 25 Final Project | By: Minh Le

Why Did I Build A Fitness Tracker?

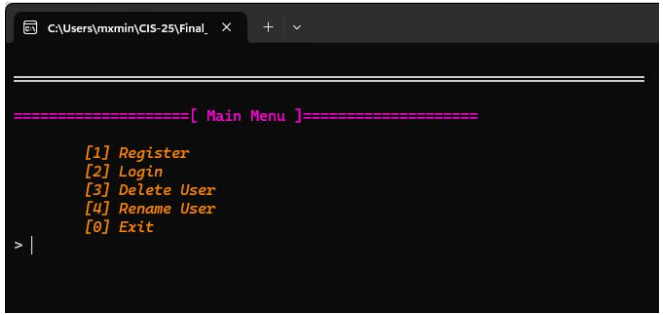
I wanted to develop something that others would find useful. Since I recently began going to the gym, I wanted a way to track my workouts and diet!

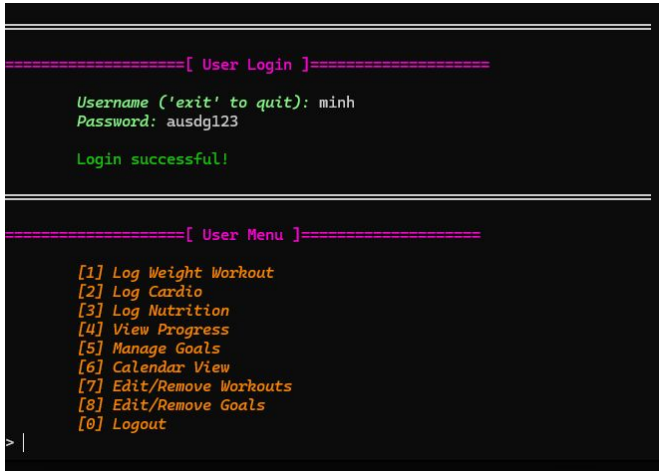
- Wanted a private, offline tracker for fitness data, unlike cloud-based web/mobile apps
- Practice C++ OOP, file I/O, modern terminal UI (box drawing, colors)
- Needed to support **multiple users with isolated data**
- Challenge: make a terminal app look as clean and nerdy as a web dashboard!

*Note: In the future, I want to learn to use Qt and build applications outside of the terminal.

Features

- Register/login users; each gets their own persistent local data
- Log workouts, cardio, nutrition, and goals
- Colorized, boxed tables for every section
- Edit/remove any entry, instantly update files
- Calendar view: filter by week, month, or custom date range





How Does the Program Work?

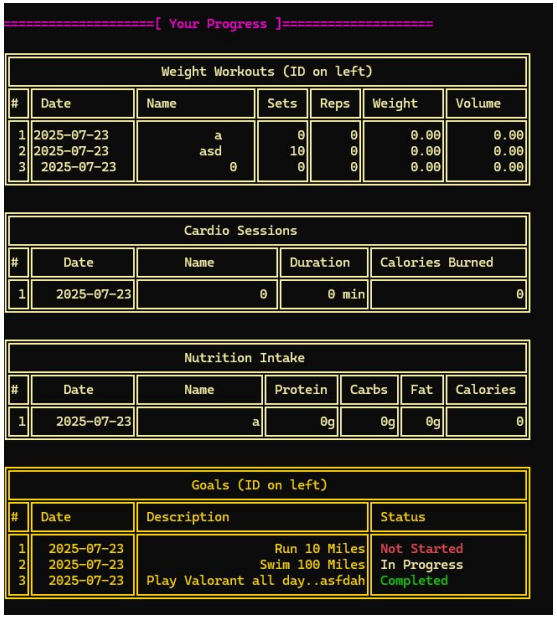
- Startup: Main menu, user picks Register/Login
- User Menu: Once authenticated, select what to log/view/edit
- Logging: Data entered via prompts, validated, stored in files
- Viewing: Tables drawn with Unicode boxes, using std::setw for alignment
- Editing/Removing: Pick by number, prompts guide through changes
- Calendar/Progress: Filter and display data for selected periods
- Exit: All changes saved automatically

Important Functions:

- **FitnessApp::run()**
- **FitnessApp::userMenu()**
- **FitnessApp::viewProgress()**

Terminal GUI

- Unicode box-drawing for tables (╔═╦═╗)
- All columns use **std::setw** and **std::setprecision** for perfect alignment
- Color codes (ANSI escapes): magenta for headers, orange for menus, green/red for feedback
- Each table (workouts, goals, etc.) coded with custom separators for clarity
- No broken lines, no overflow – even with long names!
    - (There all niche cases, but overall, it functions and looks great!)



Data Persistence: "Where Does My Data Go?"

- Each user gets a folder: data/users/<username>/
- Files:
  - workout.txt – logs of workouts
  - goal.txt – goals
  - nutrition.txt – food/macros
  - cardio.txt – cardio sessions
- Format: Each entry is a line; fields separated by | or labels
- User credentials: Stored in users.txt (plaintext; could be improved with hashing)
- On edit/remove: Lines updated/deleted instantly

How Is Data Represented?

WorkoutEntry struct:

```
struct WorkoutEntry {
    std::string name;
    int sets;
    int reps;
    double weight;
    std::string date;
};
```

Goal struct & enum:

```
enum class GoalStatus {
NotStarted, InProgress,
Completed };
struct Goal {
    std::string description;
    std::string date;
    GoalStatus status;
};
```

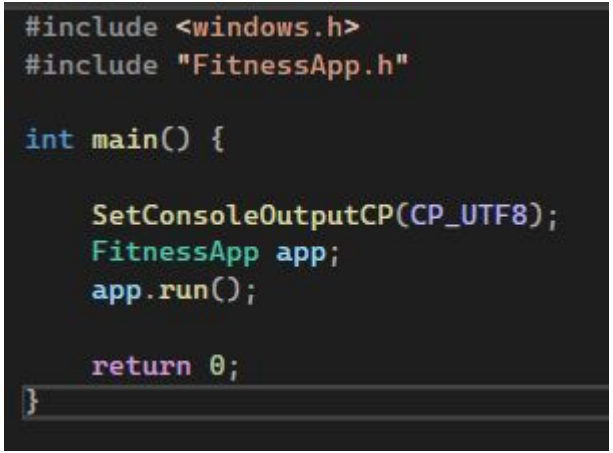
NutritionEntry struct:

```
struct NutritionEntry {
    std::string date;
    std::string food;
    double protein, carbs,
fat, calories;
};
```

How data flows:
User enters data → struct created → added to vector → serialized to file
When viewing: file is parsed → structs loaded → tables drawn

main.cpp

- All logic is encapsulated within FitnessApp class.
- **SetConsoleOutputCP(CP_UTF8)** allows the Visual Studios 2022 terminal to display special unicode characters.



FitnessApp Class

- Holds pointers to User, Workout, Nutrition, GoalManager
- Orchestrates authentication, menu navigation, all user interaction
- Passes data between UI and domain classes
- Handles validation, file I/O, and calls to colorized table functions

Key Methods:

- run(), userMenu(), logWorkout(), logNutrition(), logCardio()
- viewProgress(), viewCalendar(), manageGoals(), editRemoveWorkouts(), editRemoveGoals()

User Class: Account & Data Management

- Manages username/password, loads/saves user data files
- Handles registration, login, deletion, renaming
- Static methods for credential checking, folder ops

Example:

- **loadUserData(filename)** → returns file contents as string
- **saveUserData(filename, data)** → writes data to file

Workout Class

- Stores vector of WorkoutEntry structs
- Methods to add, edit, remove, serialize workouts

Data flow:

- Add → push_back to vector → **serializeForUser()** writes to file
- Load → parse file → fill vector
- Edit/Remove → update vector → **serializeForUser()** overwrites file

Uses **std::vector** for dynamic storage

GoalManager Class: Tracking Goals

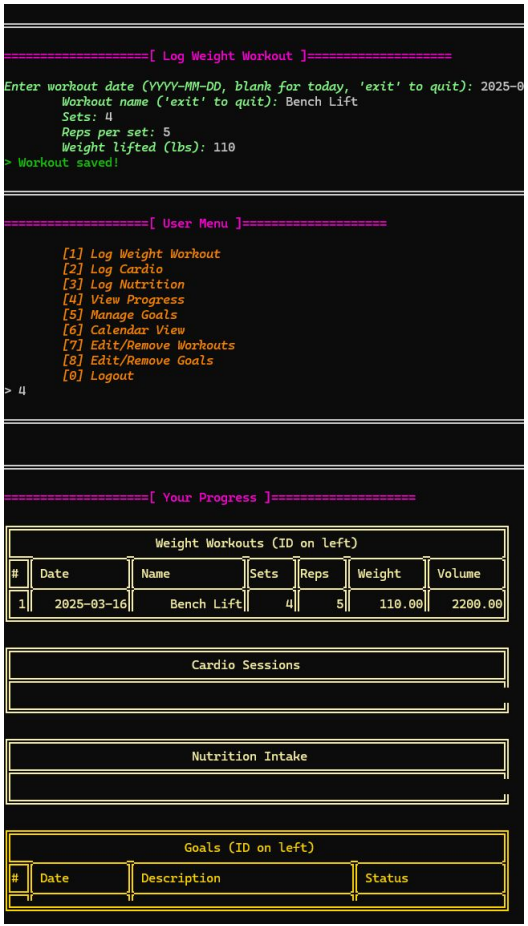
- Stores vector of Goal structs
- **Methods: addGoal(), markGoalCompleted(), markGoalInProgress(), editGoal(), removeGoal()**
- serializeForUser() writes current goals to file
- loadFromUser() parses file into vector

Nutrition Class

- Vector of NutritionEntry structs
- **Methods: addEntry(), loadFromUser(), serializeForUser()**
- Data: date, food, protein, carbs, fat, calories

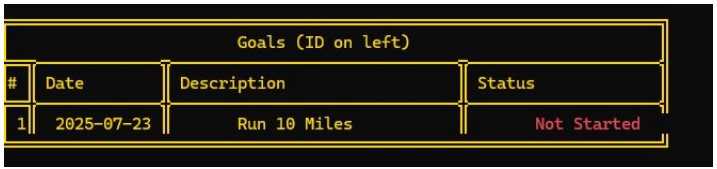
Demo: Logging a Workout

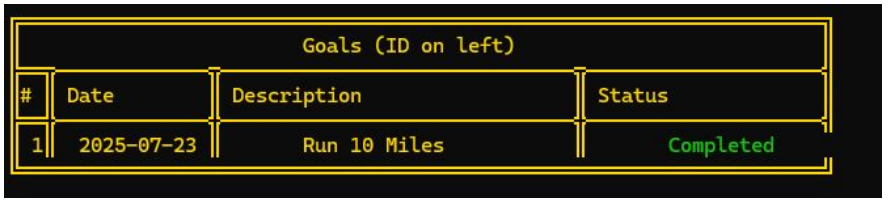
- User prompted for date, name, sets, reps, weight
- Entry created with struct, saved to file, displayed in aligned table
- Precision:
  - Table uses std::setprecision(2) for floating points
  - Always lines up, even for fractional weights!



Demo: Managing Fitness Goals

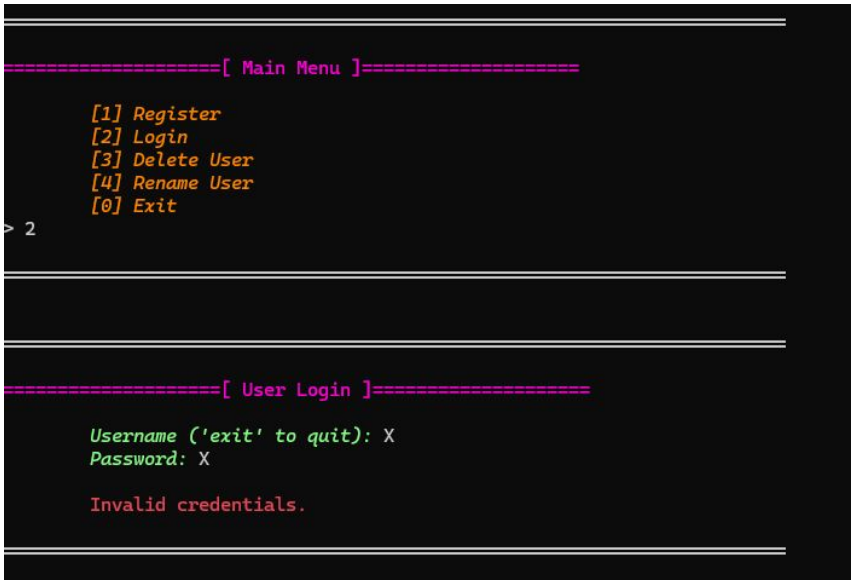
- Add, edit, complete, remove goals
- Status colored: green (Completed), yellow (In Progress), red (Not Started)
- Table visually separates each entry
- GoalStatus passed as enum





Error Handling & User Feedback

- Invalid input? "Invalid option" in bold red
- Success? Bold green message
- Warnings? Bold yellow, can't miss it
- All feedback color-coded via ANSI escapes
- "exit" command works universally for safe quitting



Limitations & Future Work

- No web/mobile interface, terminal only
- No charts/graphs (just tables, but nerdy ones!)
- Plaintext password storage (could be improved)
- Future plans:
    - Data export (CSV, maybe SQLite)
    - Password hashing
    - More workout types, deeper analytics
    - Maybe a GUI version with ncurses or Qt?

DEMO