TECHNISCHE UNIVERSITÄT DARMSTADT

Internet Practical Course Telecooperation Project Documentation

Submitted by Team Charlie:
Ashwin Arora
Dwarakanandan B M
Amos Newswanger
Kinshuk Kislay

1 Table of Contents

2 Introduction

Yet Another Fitness Tracker is an application developed for the module Internet practical course Telecooperation. The application is divided into two parts i.e. the android application (or the front-end) and the nodejs server (or the backend) The android application tracks the users 24/7 and records the following activities: Walking, Running, Cycling and Vehicle driving. Users can view their activities data in the application and also see the path they have taken during the activities. The phones internal sensors and features are used to track the various activities and their various components such as the step count, calories burned etc. The android application is backed by a node server hosted on an ec2 aws server instance. The data of the users is saved using MongoDB to facilitate fast and efficient communication. All data between the server and the android application is transferred using REST api's over http. Furthermore users also have the option to add other users as their friends in the application and check out their achievements also. The users can also compete with each other in contests to be the very best at fitness.

The following sections of this document describe the architecture implemented during this project along with an overview of the different components and libraries used and also the user manual for the application. The architecture for the front end and the backend can be found in section 2 and the user manual is documented in Section 3.

3 Technical Description

This section of the document describes the architecture used in the frontend and the backend of the application. An overview of the libraries used during the development of this project are also provided.

3.1 Front End

The Front-End for Yet another Fitness Tracker is an Android application, primarily built using the Android Platform APIs. The application adheres to android best practices and recommended architecture for robust, production-quality apps. The figure below provides a high-level view of the overall application architecture.

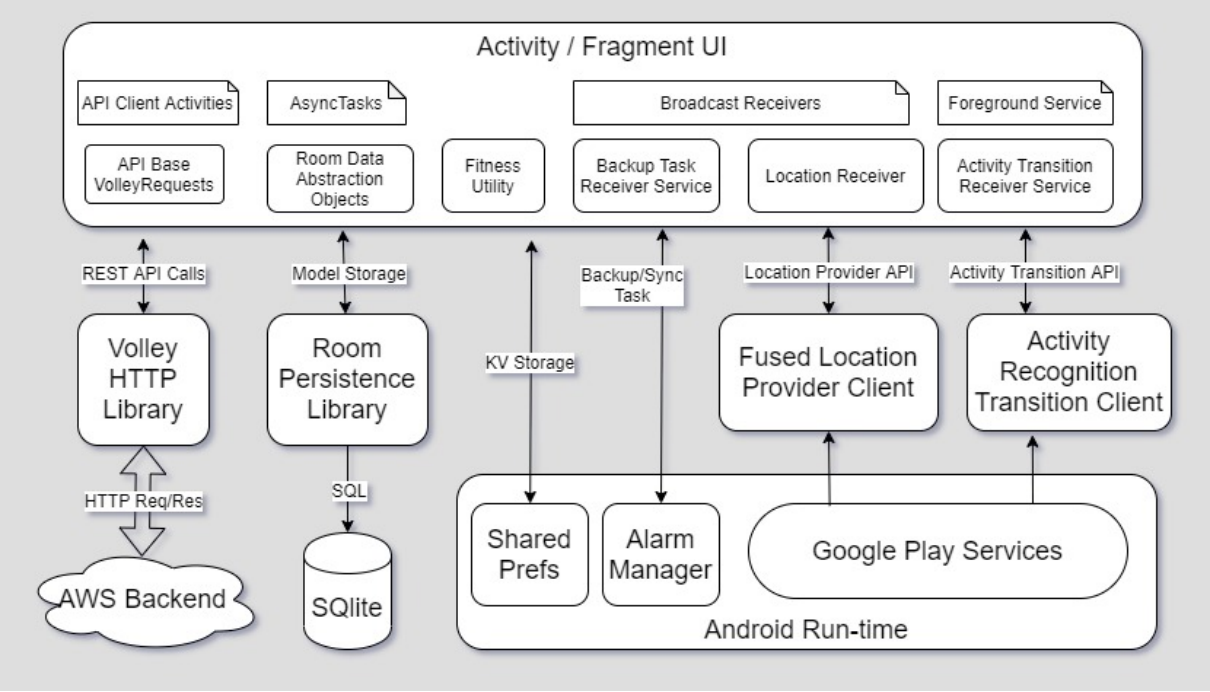

Figure 1 : Front End Architecture

The user interface view/model primarily consists of Activities and Fragments that represent the contract between the Android Operating System and our application. *Google Play Services*, which is a proprietary background service and API package is heavily used for implementing the location and activity features of our app.

Activity Recognition Transition Client and API is one such package provided by google play services that is used to detect changes in user's activity. To ensure round the clock monitoring, an Activity Transition Receiver Service is implemented as a *Foreground Service* and registers itself to get updates from the aforementioned Activity Transition Client.

To implement robust and battery efficient location monitoring, we use the *Fused Location Provider* offered by google play services. A Location Receiver is registered to this Location Provider Client to receive accurate location updates. The receiver registers to the android runtime's *Broadcast Receiver* implementation to get location updates.

For implementing periodic backup and synchronization with the application backend, we use the *Alarm Manager*, which provides access to the system alarm services. This Backup Task Receiver is again registered as a *Broadcast Receiver* with the android system.

Coming to the persistence layer, we use both the *Shared Preferences* framework offered by the Android runtime, as well as the *SQLite Relational Database*. For simple key-value pairs such as the user's Email address, Session

keys or User profile elements, we use the Shared Preferences. For storing more complex Models such as Fitness Activities or User Locations we use the SQLite database through an abstraction framework called the *Room Persistence Library*. Data Abstraction Objects or DAO are created for each of our Database Relations and Room provides a seamless interface between the actual SQLite DB and our application.

Next, an HTTP Library called *Volley* is used to build the Representational state transfer Application Programming Interface Clients (REST APIs) between our application frontend and backend. A class called ApiBase along with CustomStringRequest is implemented to provide basic functionality required by all APIs such as cookie persistence, HTTP content header attachments, etc. Volley takes care of performing the actual HTTP request and response.

As standard practice, to ensure the main User Interface thread (UI thread) does not perform long running tasks, we use the *AsyncTask* framework provided by the Android platform to perform all our database operations on the background thread. This is also enforced by frameworks such as Room and Volley.

3.1.1 Build Environment Specifications

- Build gradle Version = 3.5.3
- Minimum Supported Android Software Development Kit Level = 24
- Target Android Software Development Kit Level = 29
- Java Development Kit Version = 1.8

3.1.2 Runtime Permission Requirements

- android.hardware.sensor.TYPE_ACCELEROMETER
- android.permission.ACTIVITY_RECOGNITION
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_BACKGROUND_LOCATION
- android.permission.INTERNET
- android.permission.FOREGROUND_SERVICE
- android.permission.ACCESS_NETWORK_STATE

3.2 Backend

The backend is built using Node.js 12 for the application server and MongoDB for the database. We used ExpressJS for the web server, Mongoose for database interface and model definition, and PassportJS for defining our authentication strategies. We used the Yarn package manager to manage all dependencies

The server is currently deployed on an AWS server behind an NGINX reverse proxy. It is set up as a systemd service so that it can be automatically restarted by the system in case it crashes.

The below figure provides a high level overview of the backend application architecture.

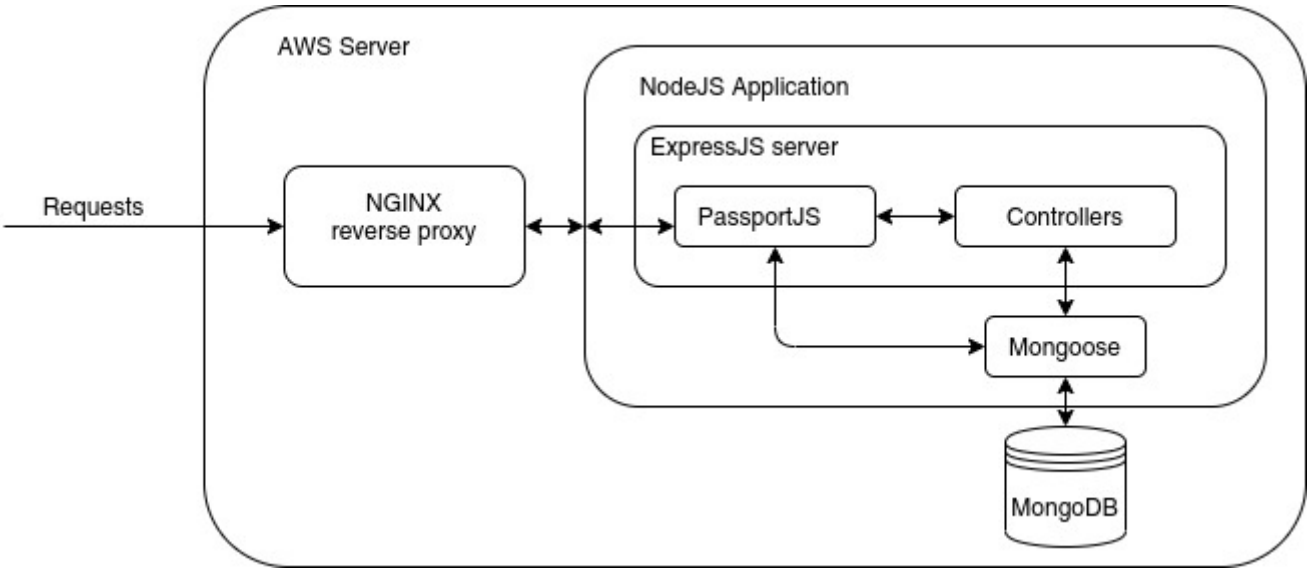

Figure 2 : Backend Architecture

3.2.1 User System

A user is uniquely identified by their email. In addition to this, we store the user's password hash, "remember me" token hash, name, age, gender, height, activity goals, list of weights on a given date, friends, and achievements. The user password is hashed using bcrypt with 10 salt rounds before being stored in the database.

Authentication is managed with the PassportJS library using two strategies:

- *Passport-local:* The user is authenticated using their email and password. The email is used to look up the user in the database and then authenticated by comparing the password to the hash with bcrypt. If this is successful, a session cookie is generated, which is used to authenticate all further requests in the session. If the user chooses the "remember me" option when signing in, a token is generated and stored in the database as a hash and returned to the client as a separate cookie along with the user id.
- *Passport-remember-me*: If the current session cookie expires or the server is restarted, the remember-me token from the remember-me cookie can be used to authenticate the user. The user id in the remember-me cookie is used to look up the user, and then the token is compared to the hash stored in the database using bcrypt. If successful, a new session cookie is generated to authenticate any further requests, and the remember-me token and cookie is regenerated.

The REST api provides functions to register, login, logout, update a user, add weight, update weight, update password, get profile, and get all users.

3.2.2 Activity System

Activities are stored in the database as a separate collection, linked to users by the user id. For each activity we store: start time, end time, user id, activity type, distance in meters, step count, duration in minutes, and location data as GeoJSON. Activity types are walking, bicycling, running, vehicle, and sleeping.

The REST api provides functions to insert a new activity, update an activity by activity id, get an activity by id, and get a list of activities with a start and end time for the current user.

3.2.3 Friend System

Friends are stored in the user collection. Each friend item specifies the friend's user id, and whether it is an incoming request, outgoing request, accepted friend, or rejected friend. Two users are considered to be friends when they both have the other listed as an accepted friend.

The REST API provides functions to request a friend, confirm a friend, reject a friend, get their list of friends, and get a friends activity.

3.2.4 Achievement System

Achievements are stored in the user collection. Each achievement has a list of goal levels.

We define 9 achievements:

- Total distance in km. Levels: 10, 50, 100, 500, 1000, 5000.
- Total steps. Levels: 5000, 50000, 100000, 1000000.
- Maximum distance in a single day in km. Levels: 1, 2, 5, 10, 20, 30, 40, 50, 100.
- Maximum distance in a month in km. Levels: 10, 25, 50, 100.
- Total weight loss in kg. Levels: 1, 2, 5, 10, 20, 30, 40, 50.
- Total running distance in km. Levels: 10, 50, 100, 500, 1000, 5000.
- Total walking distance in km. Levels: 10, 50, 100, 500, 1000, 5000.
- Total biking distance in km. Levels: 10, 50, 100, 500, 1000, 5000.
- Number of contests won

Achievements are awarded whenever a user surpasses one of the goal levels specified for each achievement. The relevant achievement values are calculated each time an activity is added, a weight is added, or a contest is won.

3.2.5 Contest System

Contests are stored in a separate collection. Each contest has a name, description, start time, end time, creator user id, private or public, goal, participant list, and start and end location if required. The goal can be steps, distance, or location. If the contest is public, anyone can see and join it. If it is private, then only friends of the creator can see and join it. The ranking is calculated once every five minutes until the contest is over.

The REST API provides functions to get a list of contests, create a contest, and join a contest

3.2.6 Build Environment Specifications

- NodeJS version : 12
- Express version : 4.17
- Mongoose version : 5.9
- MongoDB version : 4.2
- PassportJS version: 0.4

4 User Manual

This section describes the different views of the frontend android application, the relationships between them and an over view of how to navigate through the application.

4.1 Views

The following figure describes all Views and, Transitions between these Views. Individual views and their functions are explained below.

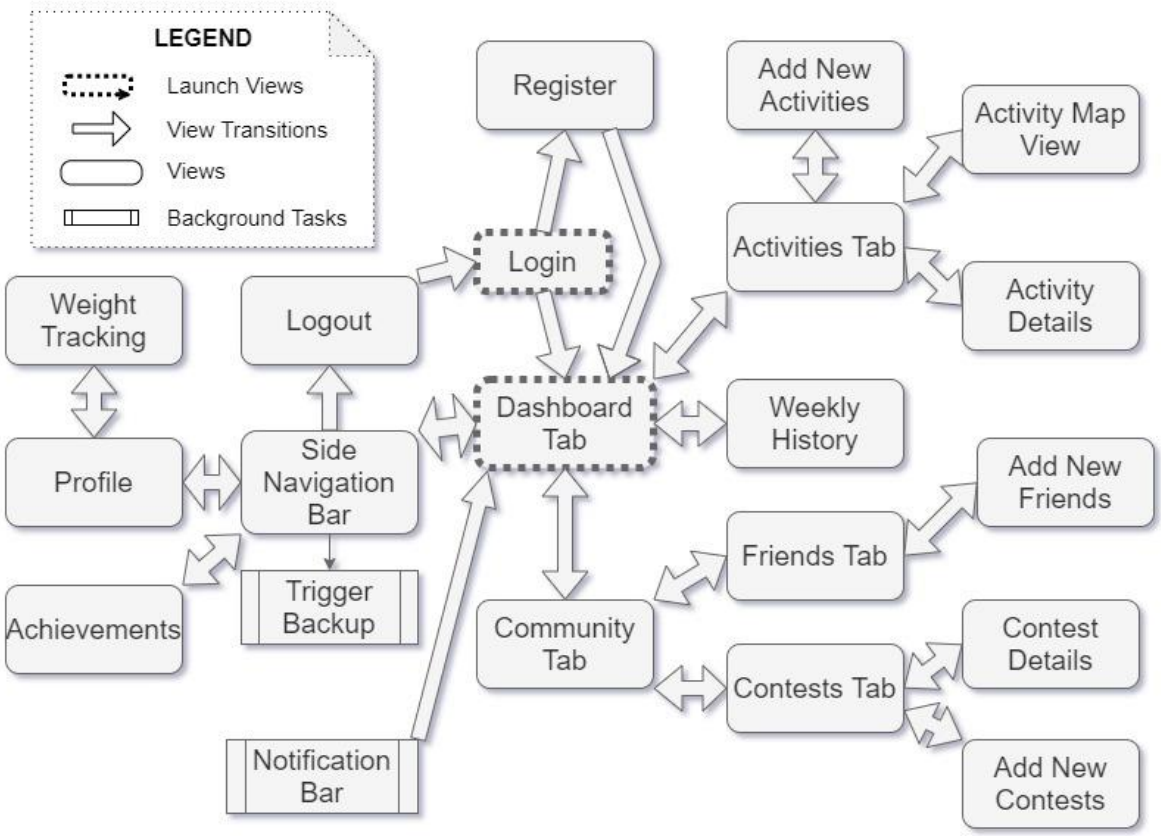

Figure 3 : Views and Transitions of the Frontend

4.2 Application Screenshots

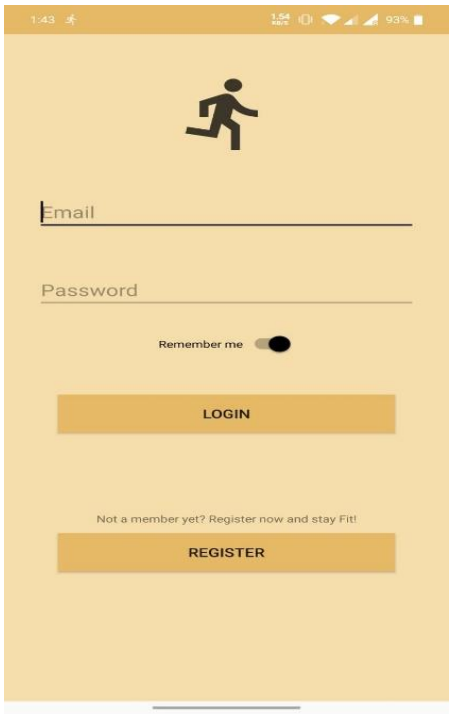

Figure 4 : Login View

Login View:
Provides a form with two fields, email and password that are used to perform the login operation. User can choose whether his/her email should be remembered.
New users without login credentials can transition to the register view by tapping register.

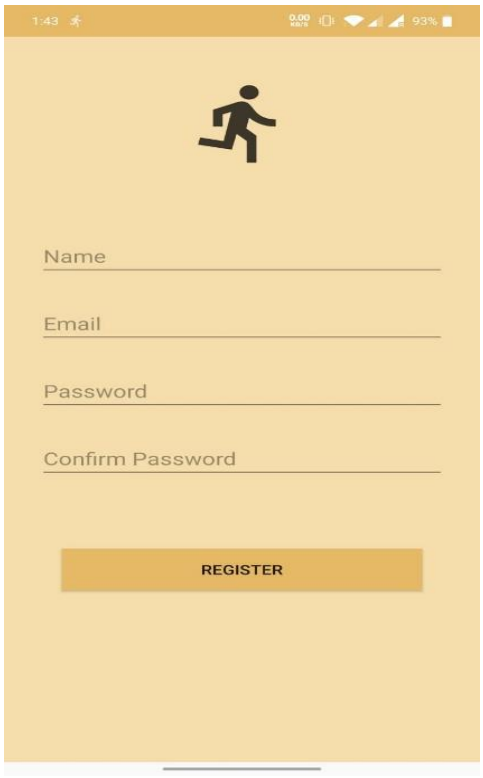

Figure 5 : Registration View

Registration View:
Provides a form with four fields. Name, Email, Password and Confirm Password. Used to register new users.

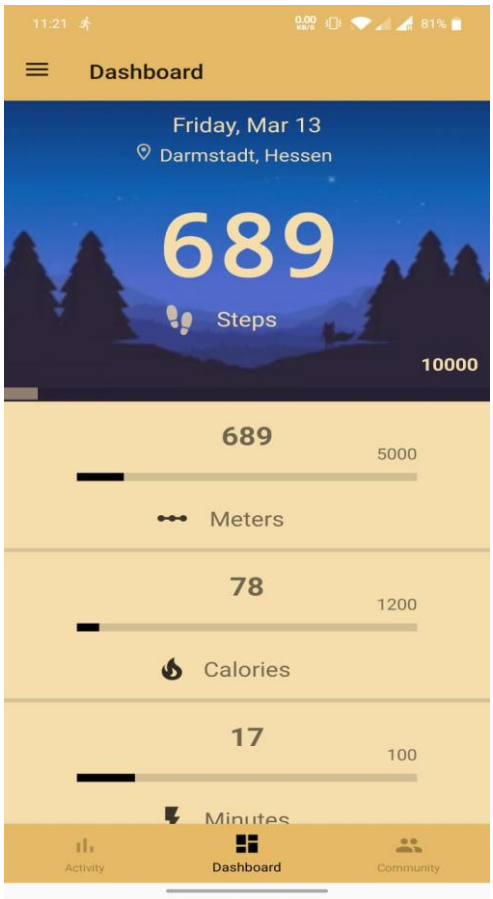

Figure 6 : Dashboard View

Dashboard View:
This is the main landing screen when the app is opened. It gives an overview of all the Fitness parameters and goals set by the user.
User location and Time is also shown.
Tapping on any of the fitness parameters will open the weekly history view for that parameter.
The bottom navigation bar can be used to transition to either the activity or Community tabs.
The hamburger icon can used to open the side navigation bar.

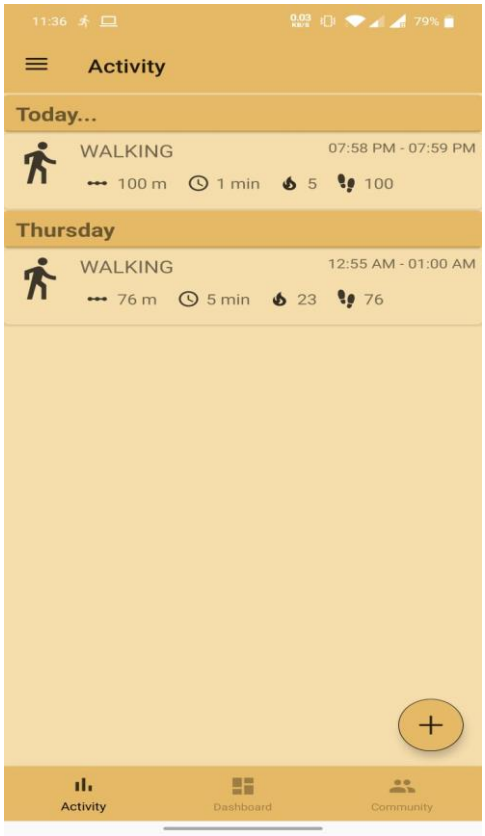

Figure 7 : Activity View

Activity View:
This view shows a list of all Fitness Activities recorded by the user. Activities from the last seven days are shown.
Tapping on any of the activities will open a detailed view with a Map showing location data relevant to that activity.
Long pressing any activity will open a detailed view where all parameters of the activity can be modified.
Tapping on the Day name will open a Map View showing location data for all activities recorded on that day.
The floating plus button can be used to log new activities manually.

Figure 8 : Activity Map View

Activity Map View:
This view shows a Map View with all location data relevant to that activity. A red marker indicates the end location of a activity, a green marker indicates start location of an activity. A line tracing all location data is also plotted.
If no valid location data is found, a message indicating the same is displayed.
The bottom strip shows some activiy details

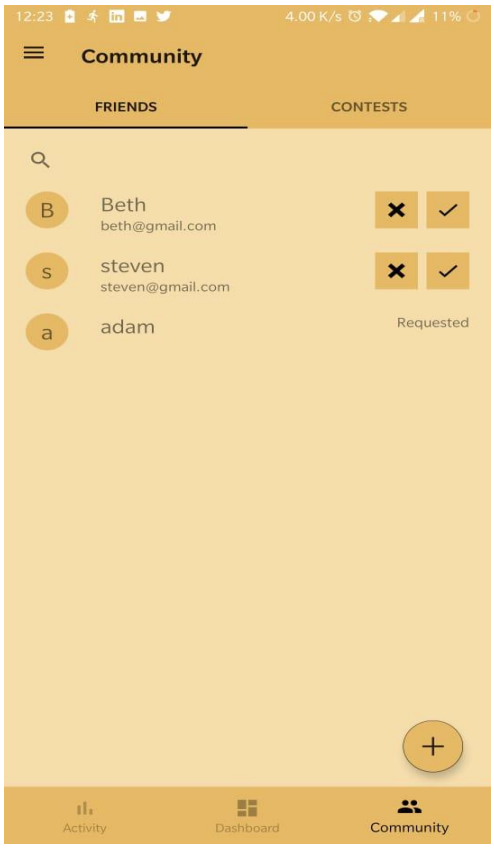

Figure 9 : Community Friends View

Community Friends View:
This View has two tabs within it. The landing tab is the Friends tab. This shows a list of friends that the user has added, also any pending friend requests from other users are shown.
The search bar can be used to filter the list of friends.
The floating plus button can be used to browse all users and send friend requests.
The user can confirm a friend or reject the request. If user sends a request to someone it is added as requested.

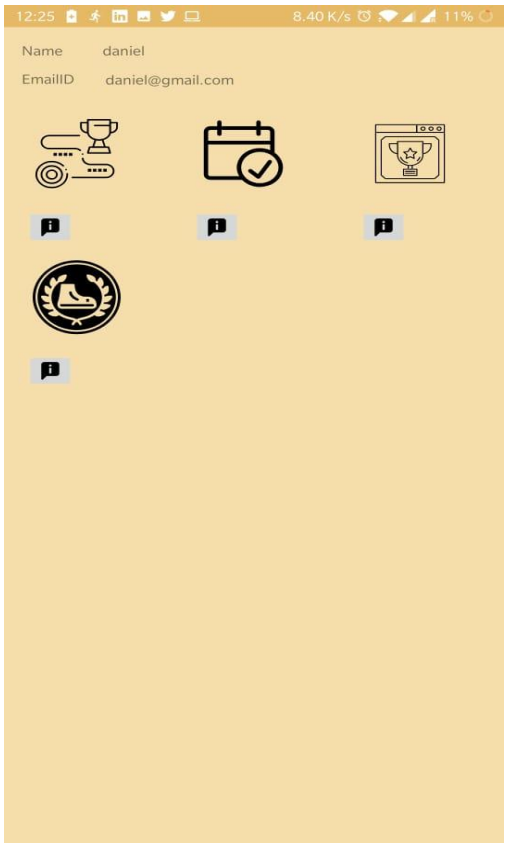

Figure 10 : Friends Achievements View

Friends Achievement View:
On Clicking on a confirmed friend, friends achievements along with his user-name and email Id are displayed on the screen which consists of all the achievements i.e. the badges which that friend has unlocked till now. On clicking the info button it displays the badge type and for which activity he has received this badge.

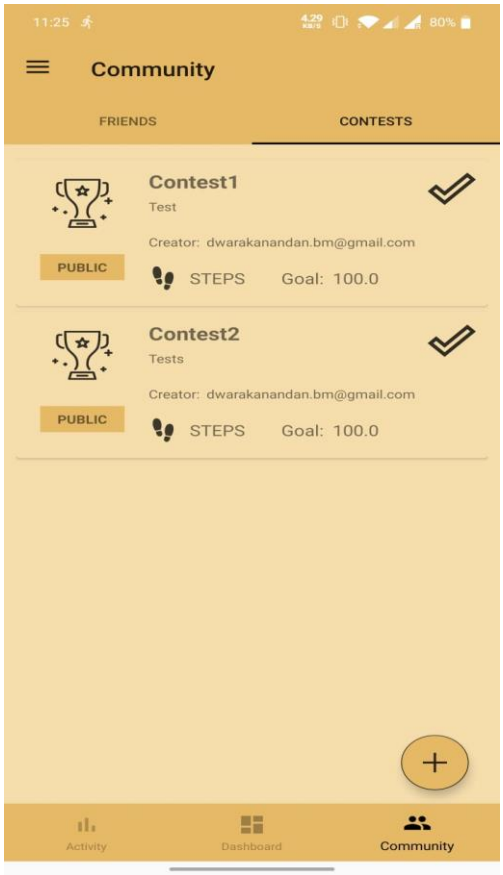

Figure 11 : Community Contest View

Community Contests View:
This is the second tab within the community view. A list of all contests are shown.
Contests are either Private or Public. Private contests are only visible to the creator and his/her friends.
The floating plus button can be used to start new contests.
Tapping on any contest will open a detailed view of that contest.

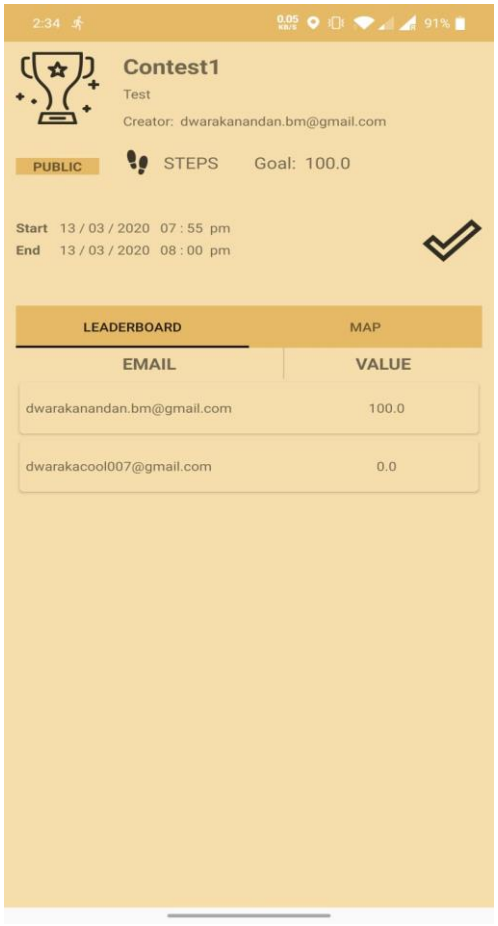

Figure 12 : Contest Details VIew

Contest Details View:
This view shows all details relevant to the contest like goal, goal type,etc.
Users can join this contest by tapping the Join button.
The Leaderboard tab shows all participants in this contest and their current progress in this contest.
The Map tab shows a map with markers of locations for all participants in this contest who are also friends.

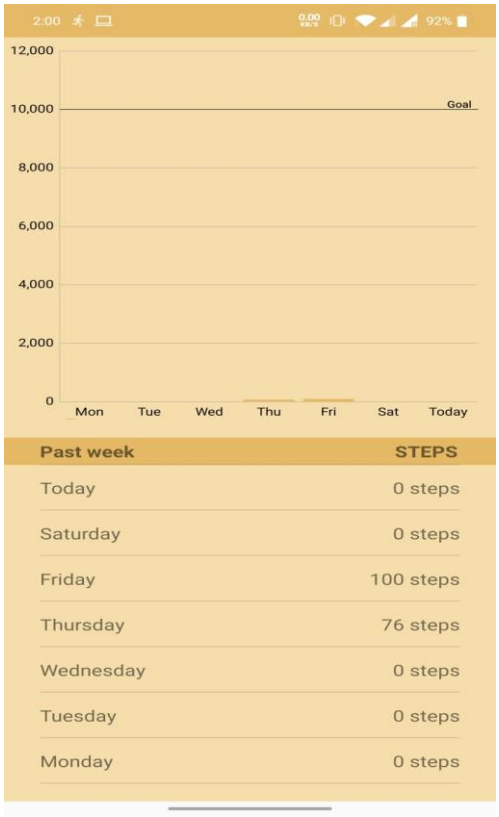

Figure 13 : History View

History View:
This view shows a weekly history for this Fitness paramter. A Bar graph is drawn showing the parameter value, and a line indicates the Goal set by the user for this Fitness parameter.

Figure 14 : Side Nav View

Side Navigation View:
This navigation view provides buttons for users to navigate to various parts of the app.
The profile button transitions to the User Profile view.
The Achievements button transitions to the Achievements View.
The Backup button triggers a backup of all User Data to the AWS backend such as Profile, location and Activity data.
The Logout button will log the user out and invalidate his session. All User Data stored locally will be cleared.

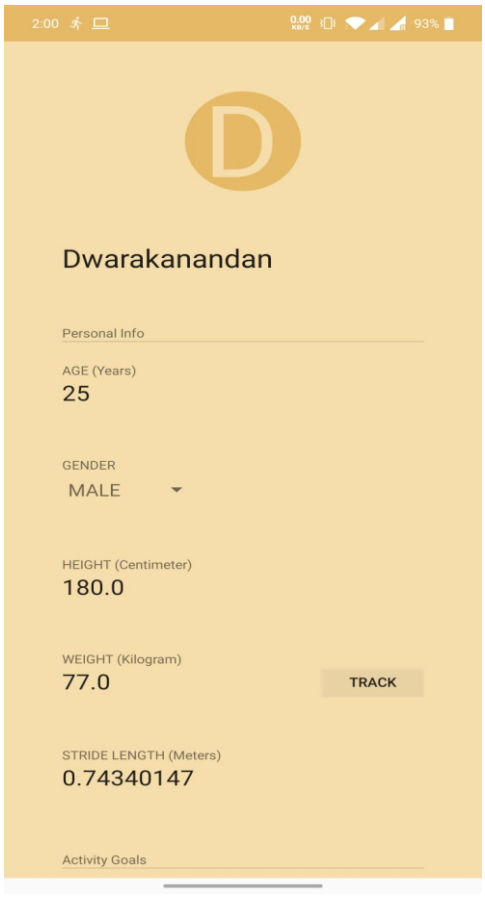

Figure 15 : Profile View

User Profile View:
This view shows a form for various Profile Parameters such as Age, Name, Gender etc. Fitness Goals can also be modified as part of this view.
The Track button transtions to the Weight tracking view.

Figure 16 : Weight Tracking View

Weight Tracking View:
This view shows a Line graph with two lines, one for User weight history and another for the corresponding Body Mass Index.
The floating plus button can be used to log new Weight entries for a given day.

Figure 17 : Achievements View

Achievements View:
This view shows a grid of achievements and awards won by the user.
They include personal achievements such as reaching Fitness goals and also awards won in community contests.
Tapping on an achievement/award will display a message detailing the achievement/award.

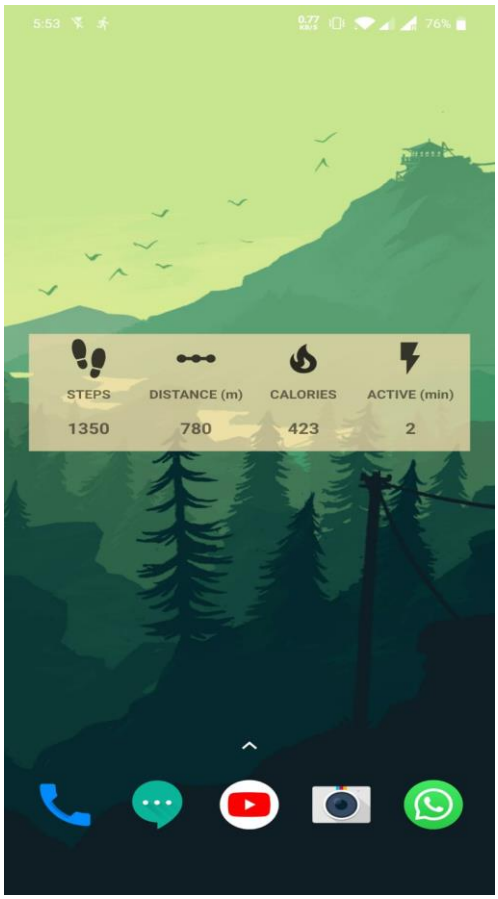

Figure 18 : Application Widget

Application Widget:
This is a Home screen Widget that gives a quick overview of all Fitness parameters like Step count, distance covered, calories burnt and Active minutes. It is updated with the aggregate summary once every 30 min.

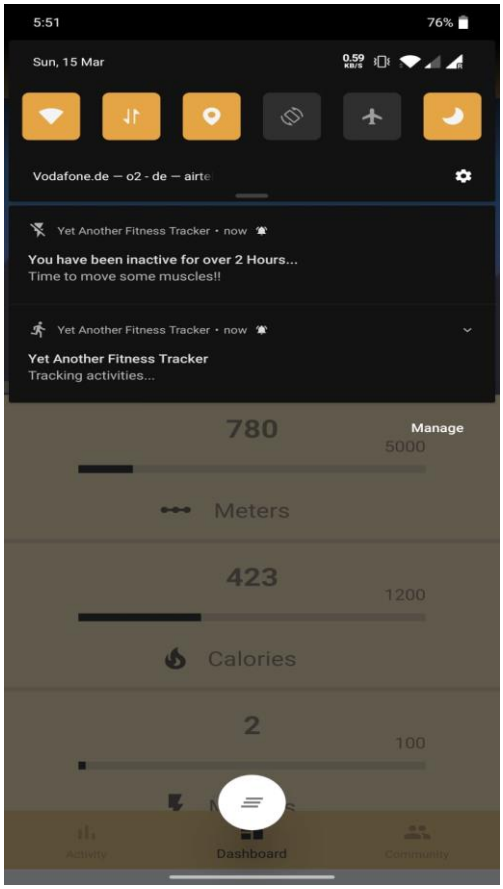

Figure 19 : Smart Notifications

Smart Notifications:
If the user is detected to be inactive for over two hours, a notification is automatically triggered, that recommends him to move. The user can swipe to dismiss this.
Another persistant notification is always show as part of the Activity foreground service that detects Activity Transitions.

List of Figures