Group 8

Avoid Rickshaw

Source Code Documentation

Anirudha Paul 141 0091 042

Muyeed Ahmed 141 1256 042

Asiful Haque Latif 141 0125 042

Foysal Amin Adnan 141 0727 042

Table of Contents

Project Overview:

Avoid Rickshaw is a fitness application which motivates people to avoid rickshaw for commuting short distance .Instead of taking rickshaw, people just have to open this app, click a button and start walking. After reaching destination it will show the amount of money this initiative saved and the amount of calories burnt along with a graph representation of previous history. This will encourage people to walk to stay fit and save money.

Design Overview:

Avoid Rickshaw is a Tizen Native Application Project which uses Basic UI with Edje data collection (EDC) .

- It uses two types of method for data storing - **App Preference** and **SQLite database .**
- For UI design **EFL library** is used with naviframe container.
- For collecting and validating data it uses **GPS tracker and Accelerometer** sensor
- For Graph drawing an open source library **Cairo** is used which is added as a native API of Tizen library .

The main code structure is branched into 4 branches.

view.c	Handles the User interface
data.c	Collect and process data from sensors and database
Sqlitedbhelper.c	Manages the SQLite database (Creates table , insert and collect data etc)
graph.c	Uses Cairo library to draw the graph based on previous history

The Core Structure:

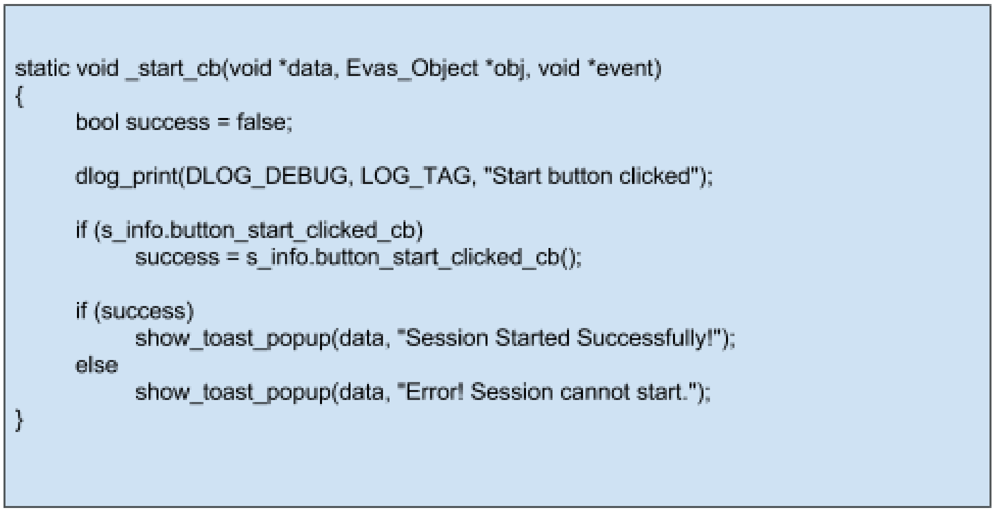
1.view.c:

Summery:

Creates essential objects: window, conformant and layout. Initialize data required for view module initialization . This code uses Stack type system to handle different types of window . It has a parent layout . While we need a new window to show different view , it creates a child node for this and destroy the child after returning back to the parent layout.

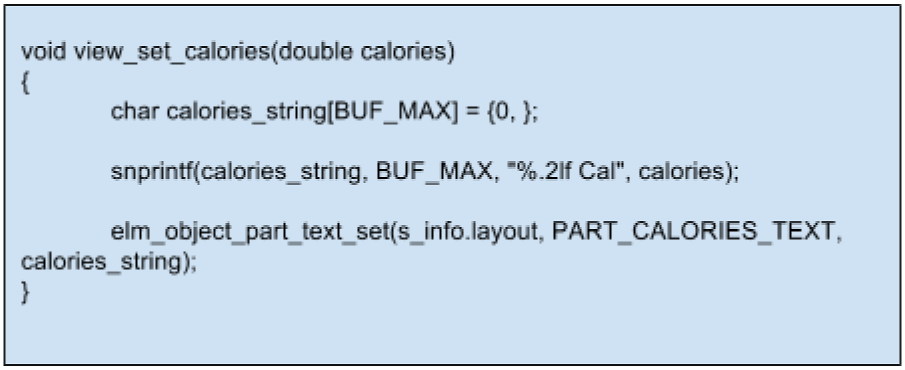
Sample callback function :

Internal callback function invoked on 'Start' button click.



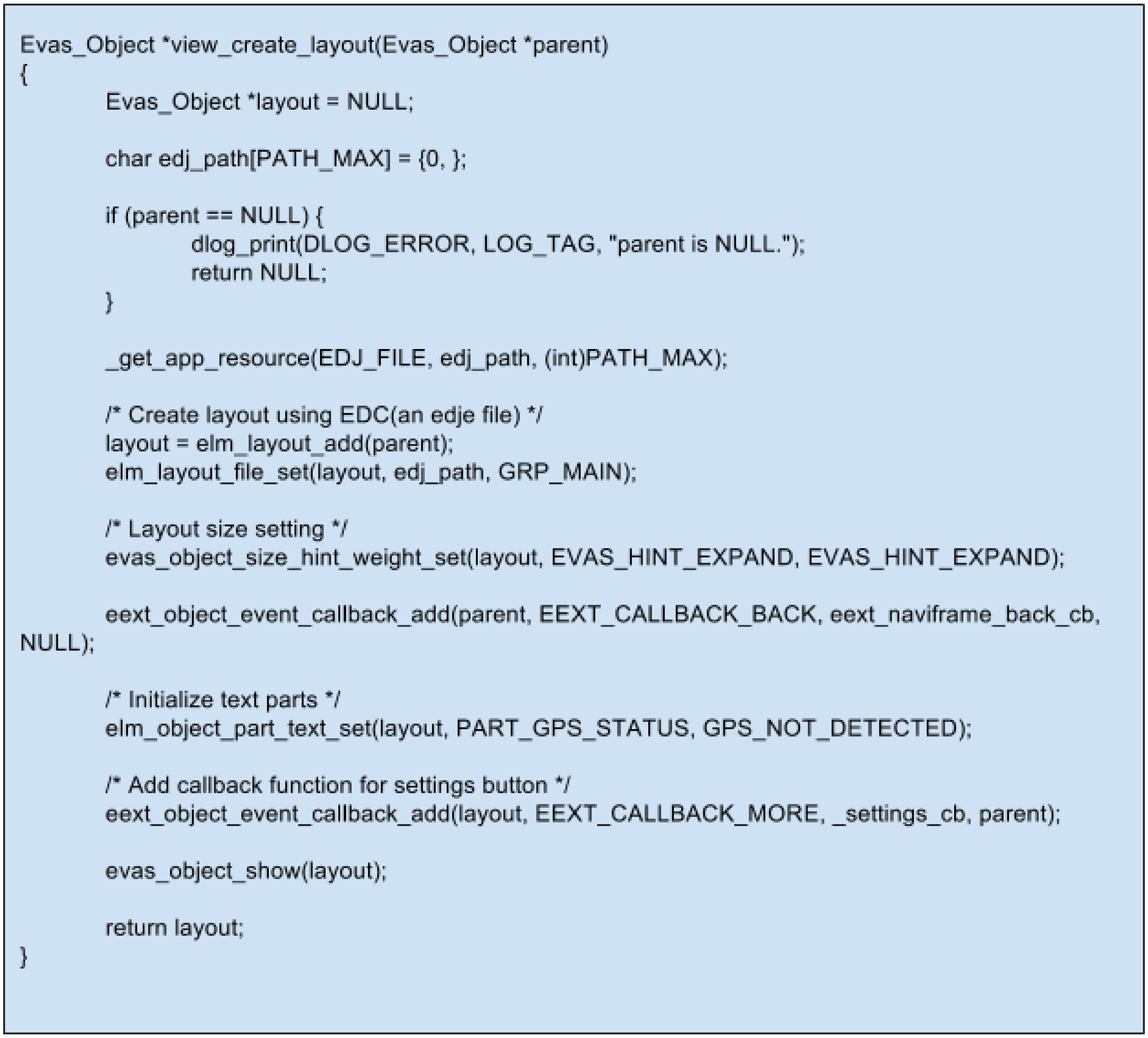
```
static void _start_cb(void *data, Evas_Object *obj, void *event)
{
        bool success = false;

        dlog_print(DLOG_DEBUG, LOG_TAG, "Start button clicked");

        if (s_info.button_start_clicked_cb)
                success = s_info.button_start_clicked_cb();

        if (success)
                show_toast_popup(data, "Session Started Successfully!");
        else
                show_toast_popup(data, "Error! Session cannot start.");
}
```

Sample textbox setter :



```
void view_set_calories(double calories)
{
            char calories_string[BUF_MAX] = {0, };

            snprintf(calories_string, BUF_MAX, "%.2lf Cal", calories);

            elm_object_part_text_set(s_info.layout, PART_CALORIES_TEXT,
calories_string);
}
```

Sample Child layout for a parent layout object :



```
Evas_Object *view_create_layout(Evas_Object *parent)
{
        Evas_Object *layout = NULL;

        char edj_path[PATH_MAX] = {0, };

        if (parent == NULL) {
                dlog_print(DLOG_ERROR, LOG_TAG, "parent is NULL.");
                return NULL;
        }

        _get_app_resource(EDJ_FILE, edj_path, (int)PATH_MAX);

        /* Create layout using EDC(an edje file) */
        layout = elm_layout_add(parent);
        elm_layout_file_set(layout, edj_path, GRP_MAIN);

        /* Layout size setting */
        evas_object_size_hint_weight_set(layout, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);

        eext_object_event_callback_add(parent, EEXT_CALLBACK_BACK, eext_naviframe_back_cb,
NULL);

        /* Initialize text parts */
        elm_object_part_text_set(layout, PART_GPS_STATUS, GPS_NOT_DETECTED);

        /* Add callback function for settings button */
        eext_object_event_callback_add(layout, EEXT_CALLBACK_MORE, _settings_cb, parent);

        evas_object_show(layout);

        return layout;
}
```

2. Data.c:

Summery:

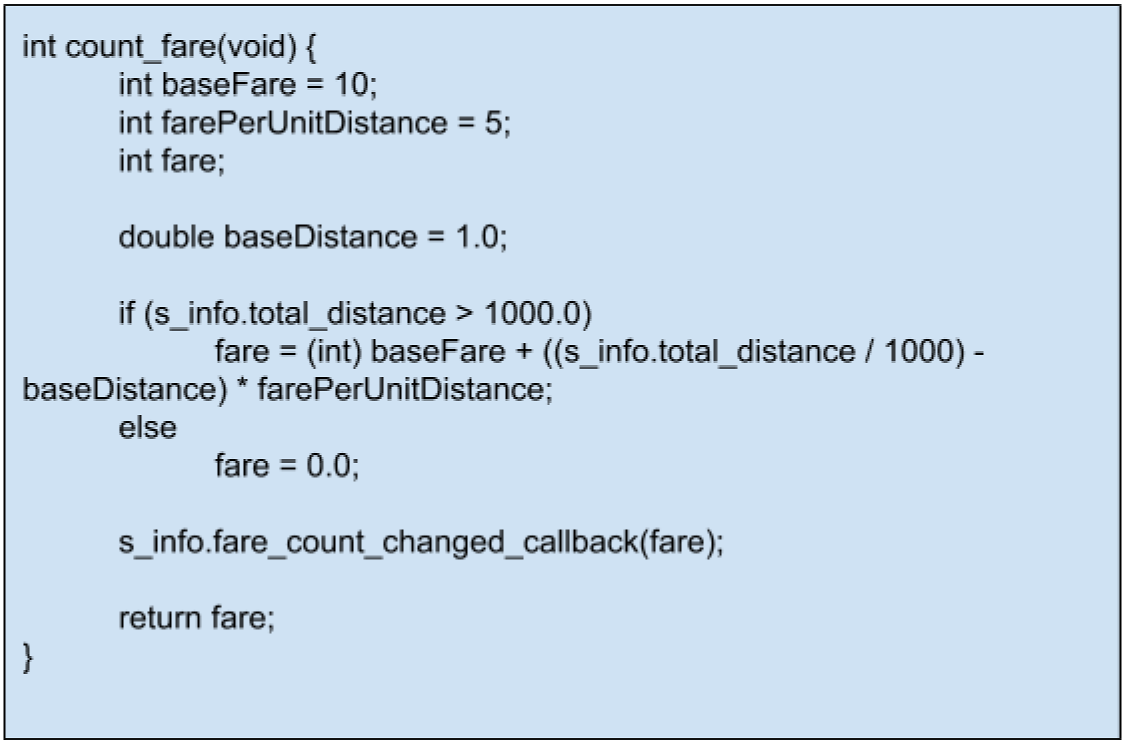
This file contains all the codes to retrieve , calculate and save everything back again to the database . It contains the two core algorithm of this application Fare calculation and Calorie burn calculation . This code also retrieve data from accelerometer to cross check and validate the data from GPS so that no data other than from walking or running bias the calculation.

Code for Fare calculation :

It uses simplified **Max Plus Methods for NonLinear Control and Estimation** to calculate estimated Rickshaw fare .

Needed Parameter :

- *Base Fare (To set a minimum fare)*
- *Base Distance (Minimum distance to activate base fare*
- *Fare per unit distance*
- *Distance covered*



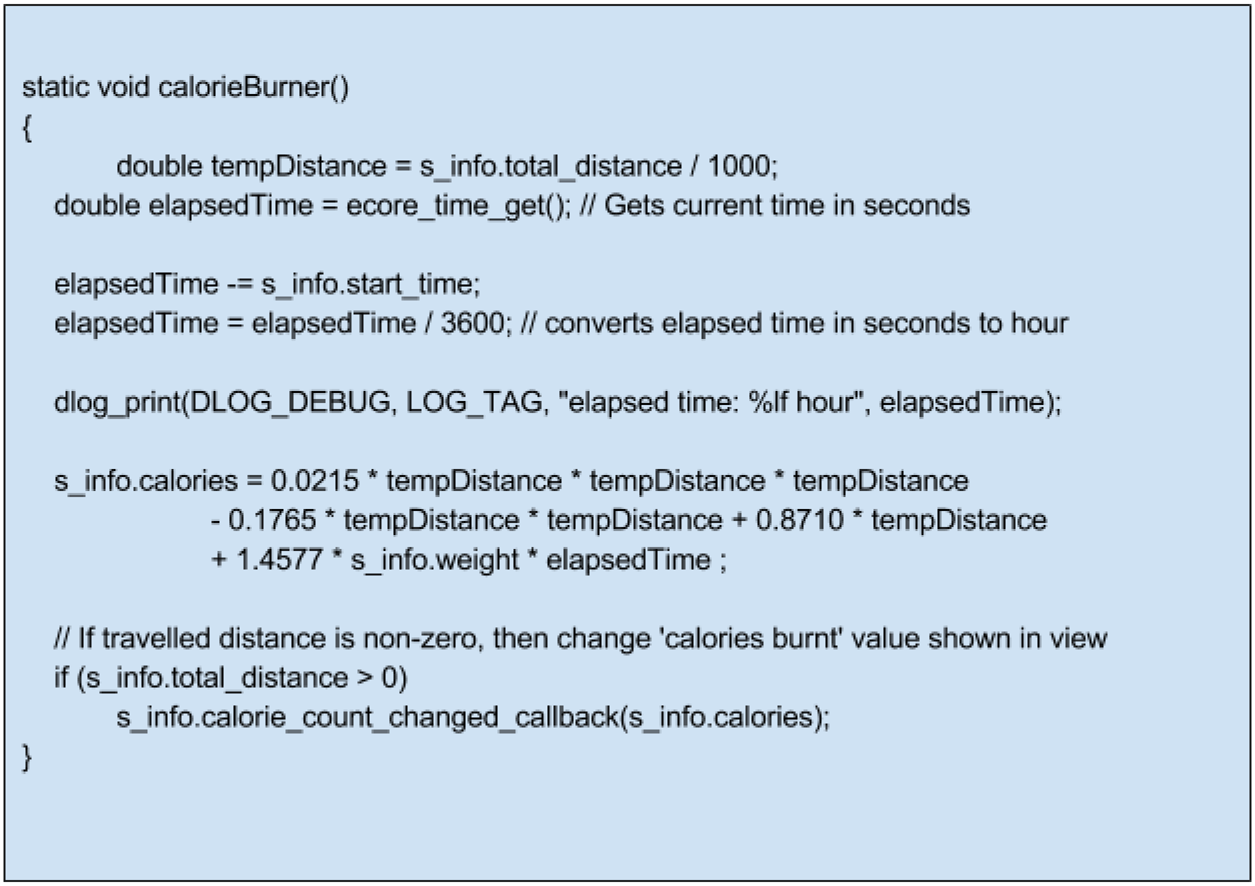
```
int count_fare(void) {
        int baseFare = 10;
        int farePerUnitDistance = 5;
        int fare;

        double baseDistance = 1.0;

        if (s_info.total_distance > 1000.0)
                fare = (int) baseFare + ((s_info.total_distance / 1000) -
baseDistance) * farePerUnitDistance;
        else
                fare = 0.0;

        s_info.fare_count_changed_callback(fare);

        return fare;
}
```

Code for Calorie Burn Calculation:

This function calculates calories burnt . For simplicity it thinks that walking surface grade is 0% this mean the surface is plane.

Needed Parameter :

- *Distance*
- *Time*
- *Weight (By default set to 70KG but user can manually control it)*



```
static void calorieBurner()
{
            double tempDistance = s_info.total_distance / 1000;
    double elapsedTime = ecore_time_get(); // Gets current time in seconds

    elapsedTime -= s_info.start_time;
    elapsedTime = elapsedTime / 3600; // converts elapsed time in seconds to hour

    dlog_print(DLOG_DEBUG, LOG_TAG, "elapsed time: %lf hour", elapsedTime);

    s_info.calories = 0.0215 * tempDistance * tempDistance * tempDistance
                    - 0.1765 * tempDistance * tempDistance + 0.8710 * tempDistance
                    + 1.4577 * s_info.weight * elapsedTime ;

    // If travelled distance is non-zero, then change 'calories burnt' value shown in view
    if (s_info.total_distance > 0)
            s_info.calorie_count_changed_callback(s_info.calories);
}
```

3. Graph.c:

Overview :

It retrieves last Seven days data of Fare saved and Calories burnt and show it on a graph . It also show the average of the data .

API Used :

- Cairo

Reference Link:

1. https://developer.tizen.org/development/api-guides/native-application/graphics/cairo
2. http://zetcode.com/gfx/cairo/basicdrawing/
3. https://www.cairographics.org/manual/

Description of the major functions used:

- ***cairo_move_to (cairo_t *cr, double x, double y)***

  Move the current reference point to the (x,y) coordinate

- ***cairo_line_to (cairo_t *cr, double x, double y)***

  Adds a line from current reference point to (x,y) coordinate

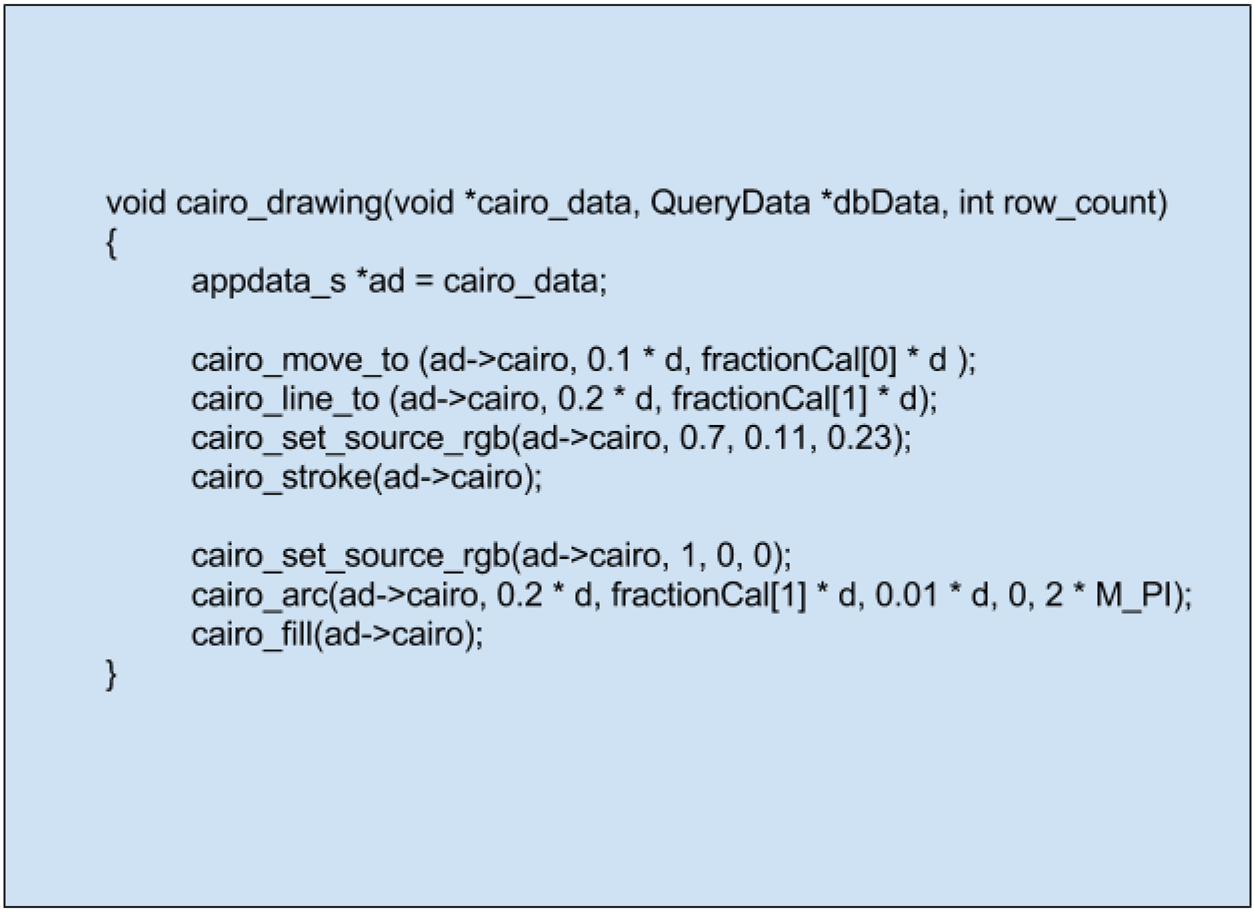
- ***cairo_stroke()***

  It bolds the line drawn so that it can be visible

- ***cairo_arc()***

  It draws a circle . It is used to highlight points

Sample code :



```
void cairo_drawing(void *cairo_data, QueryData *dbData, int row_count)
{
        appdata_s *ad = cairo_data;

        cairo_move_to (ad->cairo, 0.1 * d, fractionCal[0] * d );
        cairo_line_to (ad->cairo, 0.2 * d, fractionCal[1] * d);
        cairo_set_source_rgb(ad->cairo, 0.7, 0.11, 0.23);
        cairo_stroke(ad->cairo);

        cairo_set_source_rgb(ad->cairo, 1, 0, 0);
        cairo_arc(ad->cairo, 0.2 * d, fractionCal[1] * d, 0.01 * d, 0, 2 * M_PI);
        cairo_fill(ad->cairo);
}
```

4. Sqlitedbhelper.c:

Overview:

Handles the SQLite Database . Creates data table and store data for last 30 Days . Necessary queries are also written here.

Source Material:

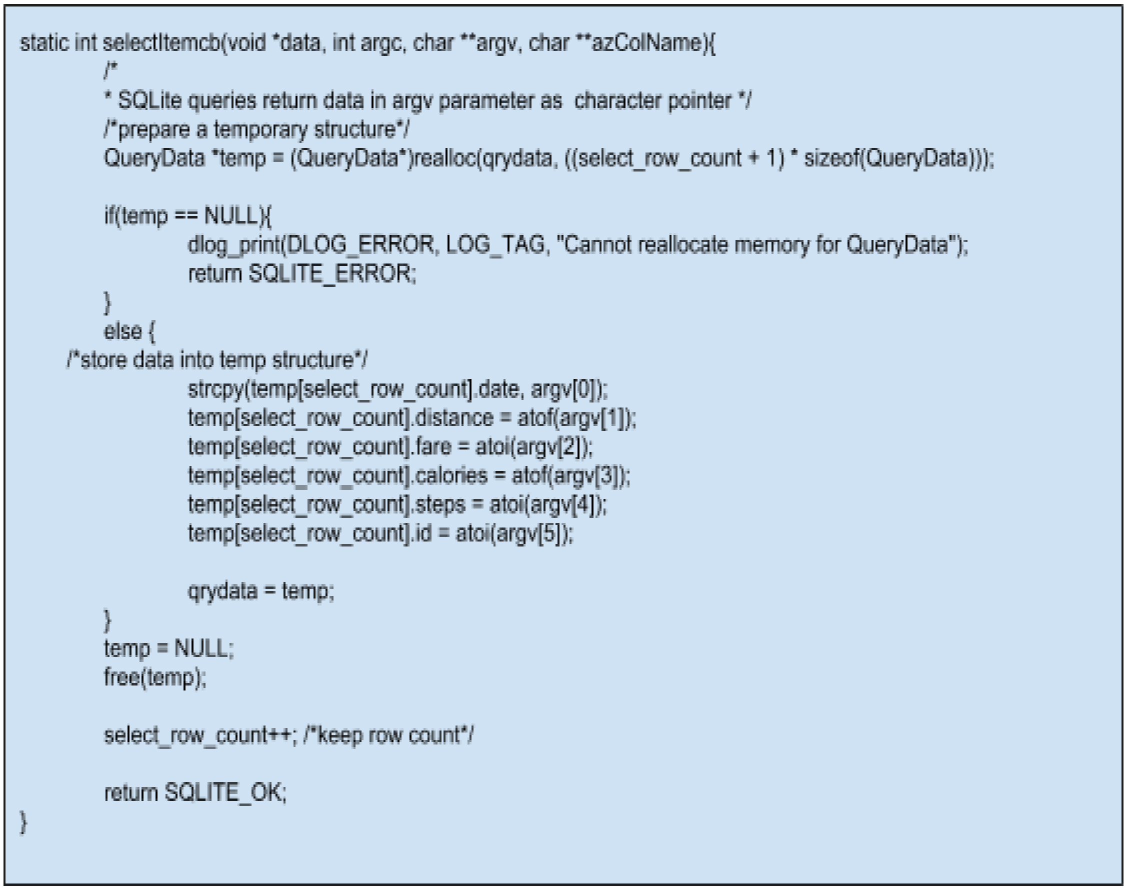
- https://developer.tizen.org/forums/native-application-development/complete-tutorial-sqlite-database-crud-operation-and-data-access-tizen-native-application

Sample code:

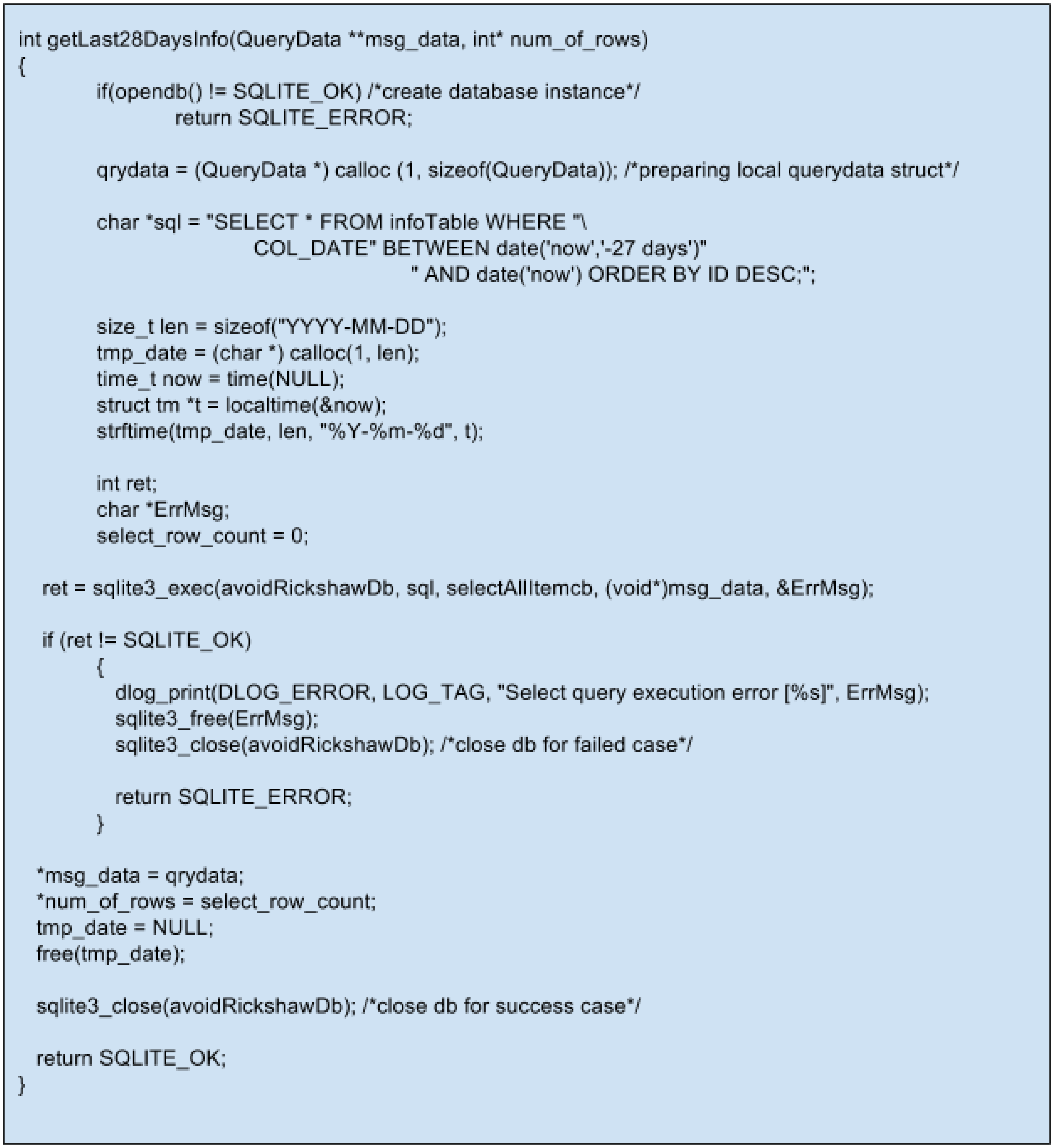
One of the most important function written in this file is

int getLast28DaysInfo(QueryData **msg_data, int* num_of_rows)

This function not only query for last 28 days with current date included but uses a callback function **selectAllItemcb** .
This callback will be called for each row fetched from database. we need to handle retrieved elements for each row manually and store data for further use .



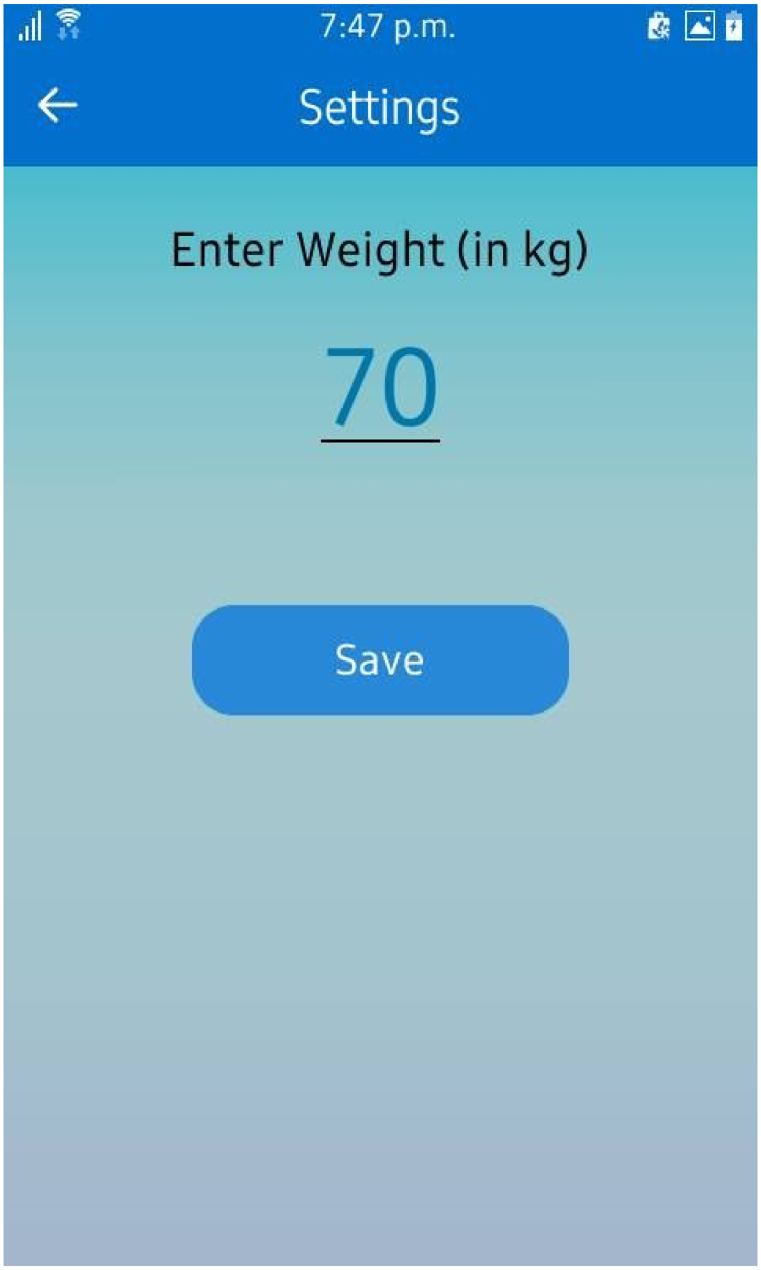
```
static int selectItemcb(void *data, int argc, char **argv, char **azColName){
            /*
            * SQLite queries return data in argv parameter as  character pointer */
            /*prepare a temporary structure*/
            QueryData *temp = (QueryData*)realloc(qrydata, ((select_row_count + 1) * sizeof(QueryData)));

            if(temp == NULL){
                        dlog_print(DLOG_ERROR, LOG_TAG, "Cannot reallocate memory for QueryData");
                        return SQLITE_ERROR;
            }
            else {
      /*store data into temp structure*/
                        strcpy(temp[select_row_count].date, argv[0]);
                        temp[select_row_count].distance = atof(argv[1]);
                        temp[select_row_count].fare = atoi(argv[2]);
                        temp[select_row_count].calories = atof(argv[3]);
                        temp[select_row_count].steps = atoi(argv[4]);
                        temp[select_row_count].id = atoi(argv[5]);

                        qrydata = temp;
            }
            temp = NULL;
            free(temp);

            select_row_count++; /*keep row count*/

            return SQLITE_OK;
}
```



```
int getLast28DaysInfo(QueryData **msg_data, int* num_of_rows)
{
        if(opendb() != SQLITE_OK) /*create database instance*/
                return SQLITE_ERROR;

        qrydata = (QueryData *) calloc (1, sizeof(QueryData)); /*preparing local querydata struct*/

        char *sql = "SELECT * FROM infoTable WHERE "\
                        COL_DATE" BETWEEN date('now','-27 days')"
                                        " AND date('now') ORDER BY ID DESC;";

        size_t len = sizeof("YYYY-MM-DD");
        tmp_date = (char *) calloc(1, len);
        time_t now = time(NULL);
        struct tm *t = localtime(&now);
        strftime(tmp_date, len, "%Y-%m-%d", t);

        int ret;
        char *ErrMsg;
        select_row_count = 0;

  ret = sqlite3_exec(avoidRickshawDb, sql, selectAllItemcb, (void*)msg_data, &ErrMsg);

  if (ret != SQLITE_OK)
        {
          dlog_print(DLOG_ERROR, LOG_TAG, "Select query execution error [%s]", ErrMsg);
          sqlite3_free(ErrMsg);
          sqlite3_close(avoidRickshawDb); /*close db for failed case*/

          return SQLITE_ERROR;
        }

  *msg_data = qrydata;
  *num_of_rows = select_row_count;
  tmp_date = NULL;
  free(tmp_date);

  sqlite3_close(avoidRickshawDb); /*close db for success case*/

  return SQLITE_OK;
}
```

Screenshots

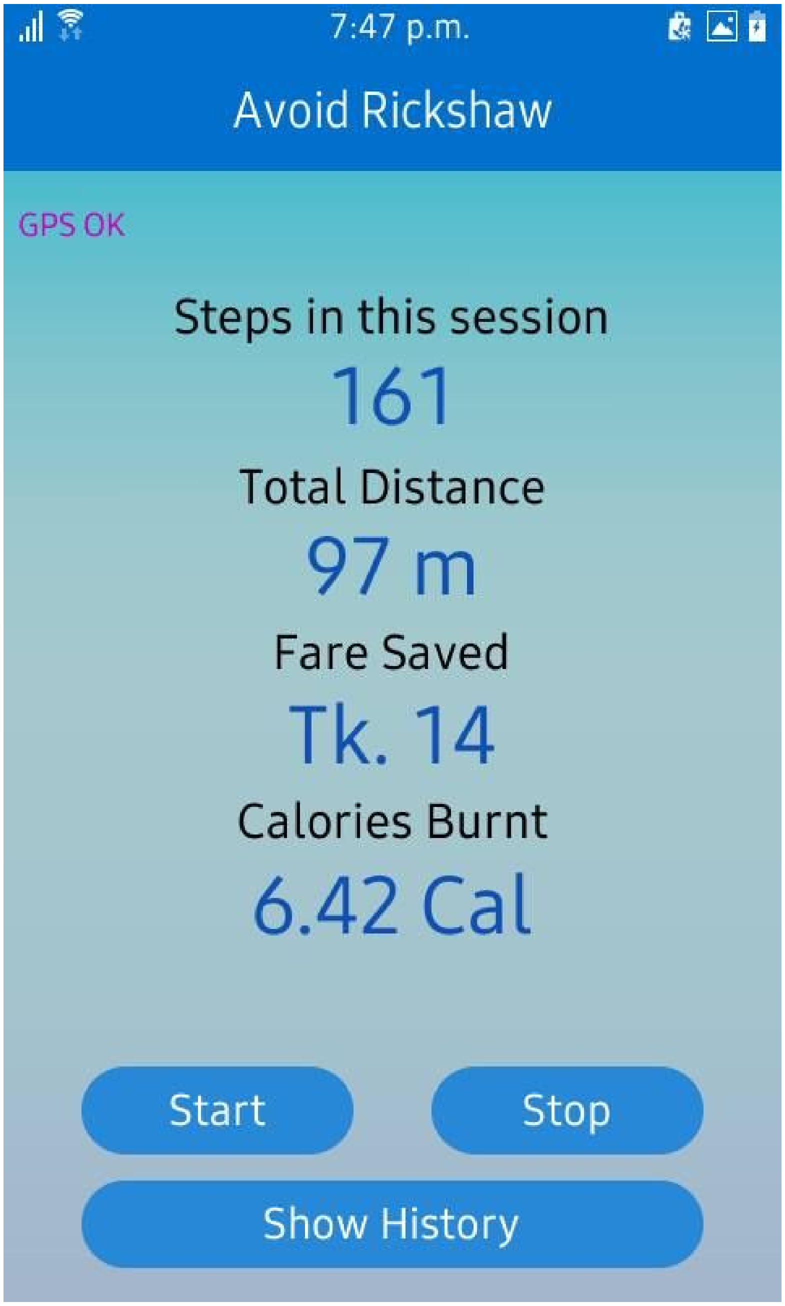
User input :



Enter Weight (in kg)

70

Home interface:



Avoid Rickshaw

GPS OK

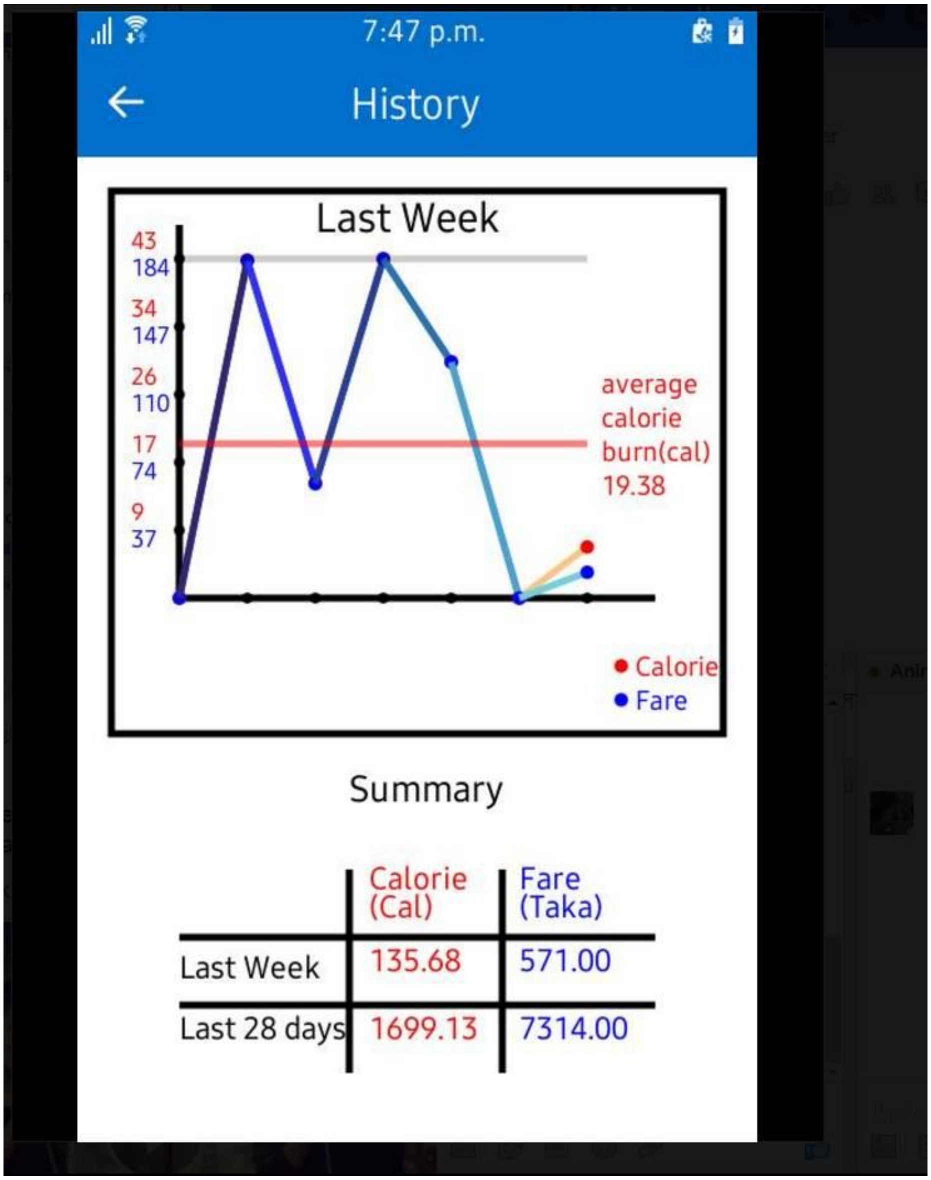
Steps in this session
161

Total Distance
97 m

Fare Saved
Tk. 14

Calories Burnt
6.42 Cal

History Graph:



Summary

	Calorie (Cal)	Fare (Taka)
Last Week	135.68	571.00
Last 28 days	1699.13	7314.00

Future Improvements:

- Use cloud storage to keep history
- Online synchronization
- Facebook , Twitter etc social media integration for keeping track of friends activity
- Online challenge , reward badge integration

Problems:

- Lack of clear instruction for Device certification
- Lack of accuracy in GPS (Geolocation API)
- Developer unfriendly UI builder
- Lack of dedicated Graph API

Reference Used For Calorie burn calculation:

- Margaria R, Cerretelli P, Aghemo P, Sassi G. Energy cost of running. J Appl Physiol. 1963 Mar;18:367-70.
- Margaria, R., 1938. Sulla fisiologia, e specialmente sul consumo energetico, della marcia e della corsa a varie velocita ed inclinazioni del terreno. Atti Accad. Naz. Lincei Classe Sci. Fis. Mat. Nat. Serie VI 7, 299–368.
- American College of Sports Medicine: ACSM's Metabolic Calculations Handbook, 2007, Baltimore, MD. Also available online at: ACSM Metabolic Equation

Links

Github Link :

https://github.com/AsifulNobel/AvoidRickshaw

Presentation link :

https://prezi.com/nglijidoulca/avoid-rickshaw/?utm_campaign=share&utm_medium=copy