CIS 422 Project1

Programmer's Documentation

04/27/2020

Team Members (and initials):

Irfan Filipovic (IF), Shane Folden (SF), Man Him Fung (MF), Mason Jones (MJ), Siqi Wang (SW)

1. INTRODUCTION

The following document covers the important aspects of the Social Distance Assistance app with respect to the application's code, its files and function. It also explains some of the common tasks and how they are done, as well as some known bugs and issues.

All application code is written in Swift using Apple's XCode. The backend is written in Javascript, PHP, HTML/XML.

2. APPLICATION SOURCE FILES

2.1. PHONE APPLICATION

2.1.1 ViewController.Swift

ViewController.Swift contains all code required to track user location and transmit it to the database.

2.1.1(a) ViewController

When the view loads the application will instantiate variables, create a location manager and establish user interface elements such as buttons and connect them to touch listener functions. A request for access to location services is made which must be accepted with always, or changed to always in device settings.

There are two pairs of buttons, location buttons and at home buttons. The location buttons start and stop updating location, once the start is pressed it is disabled and stop is enabled. The opposite holds true for the stop button. The at home buttons indicate whether location should be obfuscated or not, and operate the same as the location buttons. Status is displayed on the buttons.

2.1.1(b) func locationManager(manager,locations)

As the location is updated by the mobile device services the locationManager listens for updates which are present in locations. The most recent entry at index 0 being the current location or last location updated for the device.

The listener stores the current location values including date and time, and then verifies if the current time is at 5 minutes. If the time is at 5 minutes and zero seconds the manager will first check if the user is at home, if so, offsetting the latitude by

.1-.25 miles, and then will create a POST request to ‘422backend.php’, containing the required information.

2.1.2 SceneDelegate.Swift

SceneDelegate.Swift handles changes in scene. It connects view to window on connection, and reloads view to application on background and continues location updates.

2.1.2(a) func scene(UIScene, UISceneSession, UIScene.ConnectionOptions)

i. Loads Main.storyboard

ii. onLoad ViewController.swift functions are called

2.1.2(b) func sceneDidEnterBackground(UIScene)

i. Reload ViewController to continue location tracking

1. Raises error as there is no background task to terminate.
2. Continues regardless with no negative effect.

2.1.3 AppDelegate.Swift

AppDelegate.Swift does the general app configuration, loading configuration settings from Info.plist

2.1.3 (a) func application(UIApplication, configurationForConnecting, UISceneSession, UIScene.ConnectionOptions)

i. Uses info.plist to build a configuration

ii. Includes Settings such as background modes, location authorization states, Main.storyboard as Application view, LaunchScreen.storyboard as View on Load/Launch

iii. Returns UISceneConfiguration("LocationBase", connectingSceneSession.role)

2.1.4 Info.plist

Info.plist is a list of configuration settings that gets called from AppDelegate.Swift which are used for constructing the base app. Some of the settings in this file include:

- Location Authorization states
- Supported Device Orientations
- Application logo file
- Background modes,
- LaunchScreen.storyboard as View on Launch

- Required Device Capabilities

2.1.5 LaunchScreen.storyboard

This file dictates the design of the App's splash page. It contains values for the background color and the text that is displayed when the app launches.

2.1.6 Main.storyboard

This file dictates the design of the App's main view. It contains values for the background color, text displayed, buttons, and a spinner. There are many constraints located in this file to maintain correct spacing for all group member devices.

2.1.7 logo.png

A base picture of a red location pin used for the application logo. The icon displayed on the user device, along with the application name.

2.2 DATABASE

2.2.1 Database set up

(i) Mysql install
Before accessing the mysql database, we need to install mysql by using command mysqlctl install, which will create a mysql data directory and initial tables. User also need to set up a database password of his database

(ii) Mysqlctl start
In order to start the database, user need to use the command “mysqlctl start”. Then it will start the server and provide a port number to the user, which is used for connecting to the database. For this project, the port we using is 3797

2.2.2 Query

In this project, we used MySQL Workbench to modify and review the database. It can allow us to review all the data in the database. Also, it provides a platform for us to test queries and see the result at the same time. The query that we used is "SELECT dg.Identity AS ID , dg.Date AS date, MAX(dg.Time) AS Recent_Time, dg.Latitude AS latitude, dg.Longitude AS longitude, dg.Time_at_Location AS time_at FROM (SELECT g1.* FROM Geospatial_Data AS g1 LEFT JOIN Geospatial_Data AS g2 ON g1.Identity = g2.Identity AND g1.Date < g2.Date WHERE g2.Identity IS NULL) AS dg GROUP BY dg.Identity ORDER BY DATE DESC, Recent_Time DESC ", which can find out the most recent location of the app user.

2.3 BACKUP DATABASE

2.3.1 backup.sh

Backup.sh contains a shell script that is used to collect all data from Mysql server. All user tracking data will be stored in a .sql file. Which .sql file will store in the local host. The script will create a folder which is named by the date like "$MONTH-$DAY", such as "04-27". Inside that folder, it creates another fold which is named by the hour like "$HOUR".For example, the time now is 10pm, then the folder will be named as "22". Then the sql file will be stored in that folder. The function that is used in the script file called mysqldump(). It will collect all the data from our Mysql database with all the tables.

2.3.2 backup_git.sh

Backup.sh contains a shell script that is used to upload all the data that we backed up before. To avoid memory lost in the local host. In the script, it contains 3 commands, which are git add, git commit, and git push. These are the basic command for uploading files to github

2.3.3 Crontab

Cron is a time-based job scheduler for Unix OS. It enables users to set up and maintain software environments using cron to schedule jobs to run at fixed time, or date. We schedule it to run the shell script every six hours. So that we can backup our database every six hours.

2.4 WEBSITE

2.4.1 422backend.php

connectionData.txt

Contains the server address, database name, username, password, and port necessary to connect to the MySQL database on ix-dev.

$_POST[]

These assignments assign userID, Date, Time, Latitude, and Longitude to variables using data sent directly from the iPhone app.

result1

This is the query that inserts data into our database. It uses the variables assigned from $_POST[].

result2

This query is the “last location per distinct user” query. It only returns one row per distinct Identity that contains their most recent location data. This result is used to create and format our .json file.

result3

This query is used to output a tab-delimited text file of our cumulative location data.

2.4.2 422map.html

function uploadGJ()

This function uses XMLHttpRequest(), part of jQuery, to POST the text from our formatted .json file into a HTML div called json_area.

json_area

This is an invisible HTML div where the text from our .json file that contains the results of our “last location per distinct user” query is sent.

map

This is an HTML div where our mapboxgl.Map object will be stored.

mapboxgl.Map

This is the main map object that visually hosts our location data. Before the map is instantiated, mapboxgl is passed a valid accessToken in order to ensure the person who created the map is the one posting data to it. Then, a new mapboxgl.Map object is initialized and assigned a container, our ‘map’ div, and a style, the URL assigned to us via studio.mapbox.com.

mapboxgl.Marker

Once uploadGJ() posts our data to the json_area HTML div, we are able to access via document.getElementById(‘json_area’).innerHTML. The data is then re-parsed into JSON and passed to a forEach(function(marker)) function. Here, each datapoint is used in document.createElement(‘div’); and then assigned a className, backgroundImage, width, and height. Before continuing to the next datapoint however, a new mapboxgl.Marker is created for this data point by passing the div as a

parameter to mapboxgl.Marker(). From here, the Marker's coordinates and popup box are created. Finally, the dot is added to our map.

2.4.3 422Project1.html

iframe-map

This is where 422map.html's map will be displayed for everyday users to see. This section was designed with close attention to ensure it cooperates with 422map.html correctly.

2.4.4 422style.css

This file holds all CSS style options for our 422Project1.html page.

2.4.5 test422style.css

This file holds all CSS style options for our 422map.html page.

2.4.6 connectionData.txt

Contains the server address, database name, username, password, and port necessary to connect to the MySQL database on ix-dev.

2.4.7 results.json

This file contains our formatted text to be utilized in 422map.html. This is also accessible at https://ix.cs.uoregon.edu/~masonj/results.json

2.4.8 Query_result.txt

This file contains our tab-delimited output text for our cumulative location data. This is visible live at https://ix.cs.uoregon.edu/~masonj/Query_result.txt

3. COMMON TASKS

3.1 Location tracking

On the first use of the application the user will be prompted for access to location services, the user should select 'always' or 'once'. During this time the application will have displayed the LaunchScreen which directs the user to accept location services. If 'once' is selected the user will be prompted a second time, thereby storing that result upon further use of the app.

The user will have access to a screen with four buttons. To employ location tracking the user must press the Start Button, which calls the location manager to start updating locations which will process the checks for posts to the database. To disable location tracking the user must press the Stop Button, which stops the manager.

3.2 Obfuscating location

When the app is launched, the UserDefault.standard boolean 'launchedBefore' is checked to see if the app has been launched on this device. If the device has not been launched before, a random value between +/- .0007 and .0036 is generated. This equates to +/- .05-.25 miles when added to a latitude value. This value is then added to the UserDefaults.Standard with a key of "latOffset". Values in UserDefaults.Standard do not change if the app is closed and reopened, and so the latOffset will not change for a specific user.

A boolean value 'atHome' is created to determine if the user is at home. There are two buttons that are related to obfuscating location: 'At Home' and 'Not at Home'. The 'At Home' button calls the function 'startHomeButton' which sets the atHome variable to true. The 'Not at Home' button does the opposite, setting 'atHome' to false.

When a new location is about to be sent to the database, atHome is checked, and if it is True, the latOffset is added to the latitude value, which is then sent to the database.

3.3 Viewing current location data

To view the current data that is being tracked, the user must be on the project html webpage, which displays a map. The webpage will automatically pull the most recently updated location and display a marker, which may be clicked on. The marker will then display the user id, date, time, and time at location.

4. OUTSTANDING ISSUES AND KNOWN BUGS

4.1 No Time at Location

With the current implementation of location tracking and database posts, there is no implementation to track how long a user has been at a single location. The data sent to the database always contains a zero. This bug can still be fixed to function during the week of location tracking.

4.2 Conflicting Constraints

In order to create UI elements, constraints are needed to dictate where the app is on the screen. These constraints can be anything from specifying a distance in between

two elements, linking the dimensions of two elements, or specifying that an element should be centered in a certain dimension. Several constraints are conflicting which prompts several warnings in the debug logs however it does not affect usability or the UI layout.