GPS Recorder

COMP 4981 Project 3

Filip Gutica

Alex Lam

Sanders Lee

Sebastian Pelka

Contents

[Android Design: FSM 3](#_Toc414197006)

[Android Design: Pseudo-code 4](#_Toc414197007)

[Website Planning: 7](#_Toc414197008)

[Psuedocode 8](#_Toc414197009)

[Bonus: Server Configuration 10](#_Toc414197010)

[Server State Diagram 11](#_Toc414197011)

[Pseudocode: Server 12](#_Toc414197012)

[Testing Document 14](#_Toc414197013)

# Android Design: FSM



# Android Design: Pseudo-code

**Start**

On Create of main activity

Instantiate shared Preferences, location Manager, and the web view

Load the web view with our website

**Check In**

Get the device IP and MAC address

Request a connection with the server

Request a single location update

**Request Connection**

Get the server IP and Port from the shared preferences.

Instantiate a client socket passing the IP and Port as parameters to the constructor

**Request Location update**

Check enabled providers.   
 If network provider enabled use network provider

If GPS provider enabled but network provider is not enabled, use GPS provider

Else use the Passive provider

Instantiate a Location listener

Implement the onLocationChanged callback

Get the longitude, latitude and time from the location object

Write the longitude, latitude, ip address, mac address and time on the socket

Location manager request single update(provider to use, location listner)

**Write to socket**

Instantiate output stream

Set out put stream to the socket’s output stream

Write the passed string parameter onto the os stream

Close socket

**Config**

Inflate the config fragment where user can enter the server’s IP address, Port number, frequency of location updates and minimum distance change for a location update

**Start Service**

Calls On Create of our Service class

Starts the thread which this service will run on

Calls on start command of service class

Request connection

Instantiate shared preferences object

Get the device IP and MAC address

Start location discovery

Returns Start sticky so that the service will continue running even when the application is closed.

**Start Location Discovery**

Instantiate the location manager object

If network provider enabled use network provider

If GPS provider enabled but network provider is not enabled, use GPS provider

Else use the Passive provider

This will continuously listen for location updates from the location manager

Instantiate a Location listener

Implement the onLocationChanged callback

Get the longitude, latitude and time from the location object

Write the longitude, latitude, i`p address, mac address and time on the socket

Request Location Updates (provider, min time, min distance, location listener) - Continuously get location updates

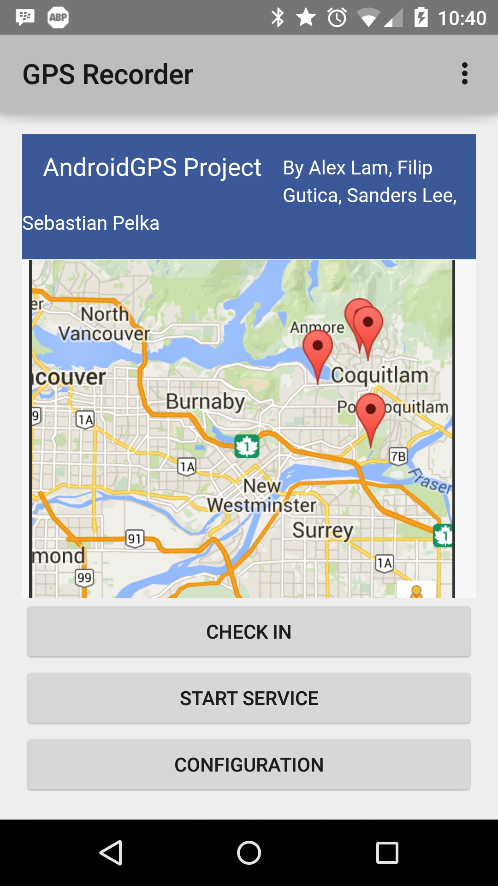
**Stop Service**

Calls on Destroy of service class

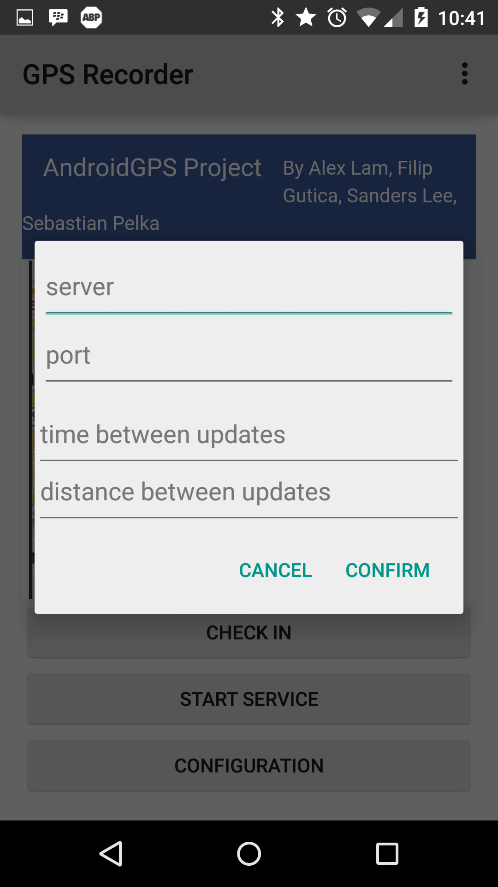
Stop the service thread

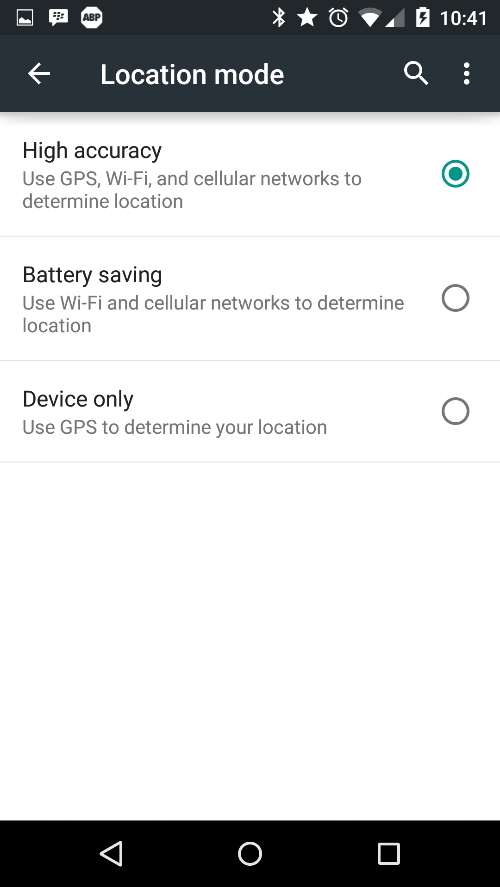
Close the client socket

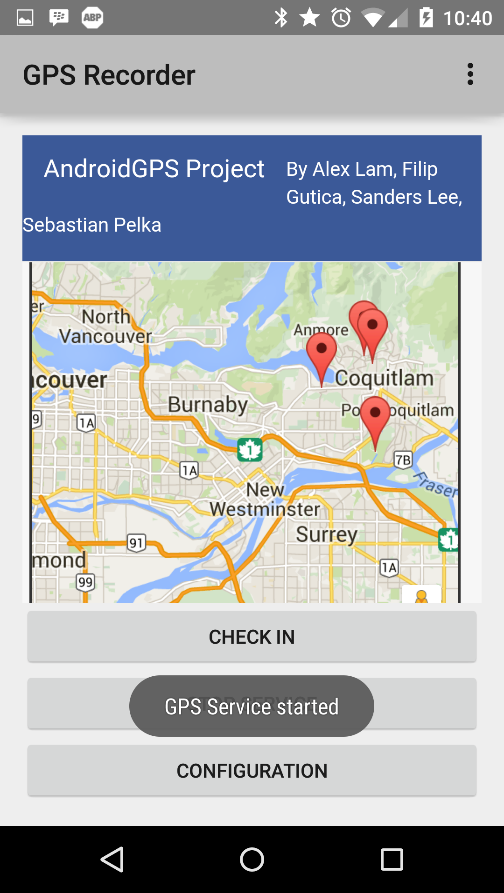
# Android Application instructions



When Application first loads, you will see our website loaded, with a map and a table of received location data.  
There are three buttons: Check In for a single location update of where you are at the moment, Start Service to start the automatic back ground location discovery service that will, on location changed, send your location to our servers, and a configuration button that you may use to set some configurations for the application.

Configure your application here after clicking on the “Configuration” button. You can specify a server IP or host name to connect to. The port for the server you are connecting to, the minimum time interval between location updates and the minimum distance between location updates.

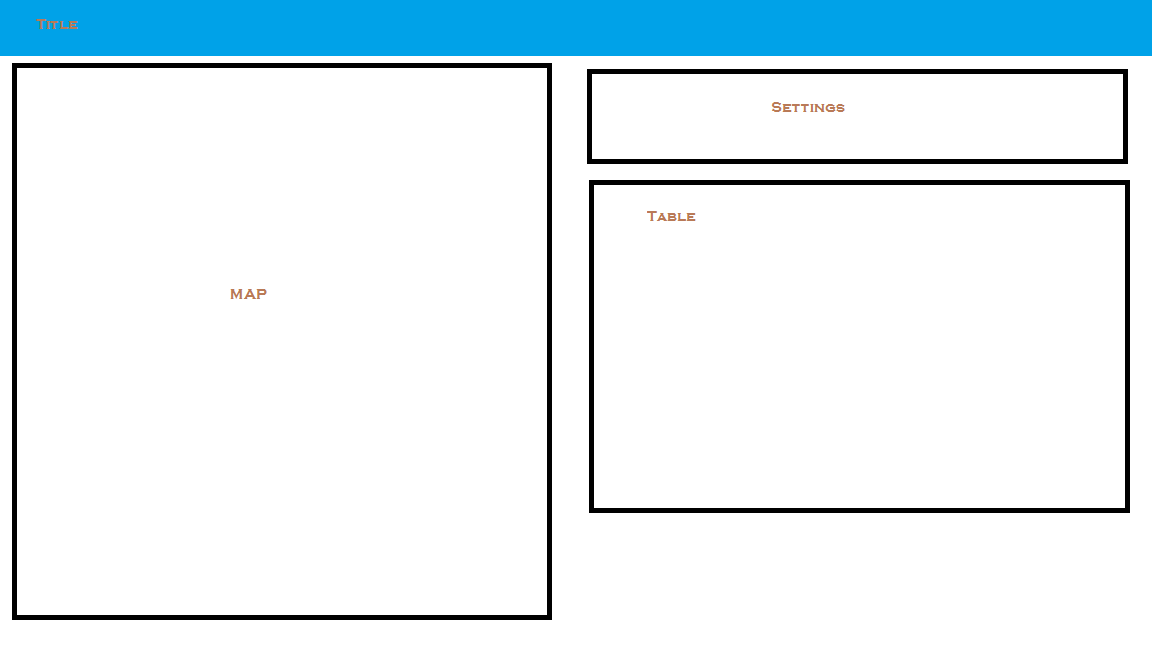
Upon clicking on the menu option on the top right hand corner of the app and selecting the “Settings” option you will be presented with this location mode page.   
Here you may set the mode for your device’s location discovery which will influence which provider is being used for location discovery.

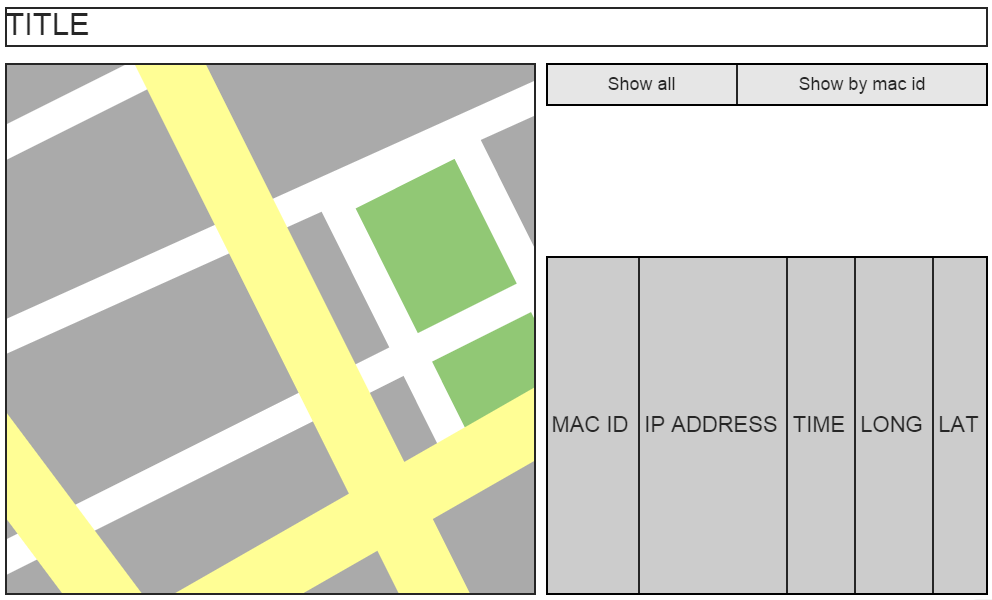
Upon Starting the service a toast will appear notifying you that the service has started. The button’s text will appear as “Stop service” and you will be able to see your location updates appear on the map in our website.

Website State Diagram

# Website Planning:

Simple sketch of what our site will look like:



Using a framework tool

# Psuedocode

Index.html:

Header

Import js

Import css

Body

Title css with banner

Left div

Map

Right div

Setting buttons

Table

Gmaps.js

Initialize

Create google map

Set center to BCIT

Generate table headers

Set mode to most recent markers

Load most recent markers

Refresh

Delete all current markers

If mode is “all current”

Load most recent markers

else

Load markers by mac id

Delete all markers

Delete rows from table

Go through array of current markers and remove markers

Most recent markers

Load xml doc

Parse xml into array

If it’s a new mac id

Add to array

If it’s an existing mac id

Override old entry

Create markers

Make table row

Mac History markers ( mac id )

Load xml doc

Parse xml into array

If xml element mac id == mac id

Place markers

Add table rows

Set mode to all current

Mode = all Current

Refresh

Set mode to mac id

Get text from textbox

Mode = text

Refresh

Refresh

Set interval to call refresh

Refresh Off

Remove interval to call refresh

Load XML Doc

Using xmlhttprequest, load the coordinates xml

Return loaded xml

# Bonus: Server Configuration

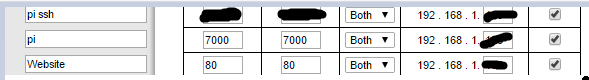
For this assignment, we have had Alex setup the website and server on a raspberry pi.

* <http://lamckalex.ddns.net/GPSAssign/>
* Login: dcomm
* Password: bcit

The configuration was as follows:

* Setup SSH
* Setup apache
* Setup .htaccess and .htpasswd

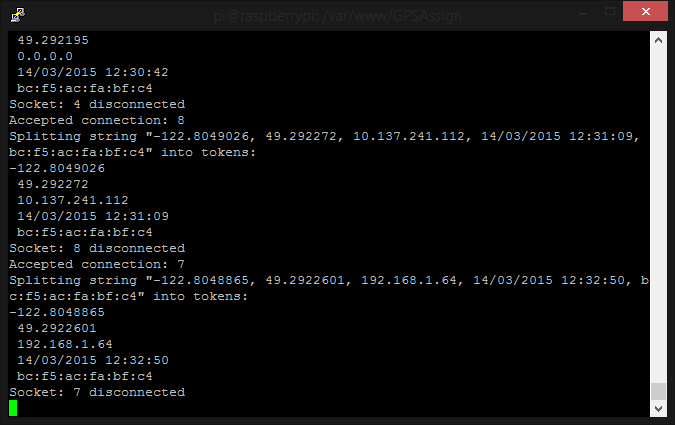
Setup Port Forwarding:

* Port ??? was forwarded for ssh.
* Port 80 was forwarded for the website
* Port 7000 was forwarded for the server
* 

To solve the issue with the IP being possible changed

* We have setup a domain with No-IP
* No-IP is a free service that can be installed onto the raspberry pi, it will provide updates to the server with its IP Address, and this allows the website to know what the correct IP for the server is even if the ISP decides to change the IP for the Raspberry Pi.

The server was then compiled on the Raspberry Pi and it is not running 24/7.

* 

Resources:

<http://www.instructables.com/id/Host-your-website-on-Raspberry-pi/?ALLSTEPS>

<http://httpd.apache.org/docs/2.2/howto/auth.html>

# Server State Diagram



# Pseudocode: Server

**Init Server**

Create a TCP socket for listening

If the socket fails

Terminate the program

Initialize a data structure to accept connections from any IP address

Bind the socket

If binding fails

Terminate the program

Begin listening for connections

Go into a read loop (the Listening state)

**Listening**

If a new connection occurs, fork a child process to handle the request

If the fork fails

Terminate the program

Else, go to the Waiting state

**Waiting**

Check the socket for data

If data is on the socket

Go to the Process GPS Data state

If the program terminates

Terminate the program

**Process GPS Data**

Read raw GPS data from android in

Convert the raw data into an easily useable form (a struct)

Read the contents of the XML document to a list

Append the new GPS coordinates to the list

Write the updated list back to the file

**Terminate Program**

{

Terminate all Child processes

Close all sockets

}

# Testing Document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test #** | **Method** | **Tool/App** | **Expected Outcome** | **Pass/Fail** |
| **1** | Create a fabricated XML document to see whether the server can read and parse the required fields for each GPS coordinate reading | server app | Each coordinate entry has all its fields loaded into a structure, including name, MAC, IP, date-time, longitude, & latitude | Pass |
| **2** | Enter fabricated GPS data to see whether the server can turn the data into an XML entry and put it into the XML document from the previous test | server app | A new entry with fabricated data is inserted into the coordinates XML document | Pass |
| **3** | Gather information from the Android client app and send it to the Linux server app to see whether the TCP connection and data transfer is working properly | server app + Android app + strace | The GPS data sent by the Android client is printed out on a terminal on the Linux server | Pass |
| **4** | Gather information from the Android client app and send it to the Linux server app to see whether the sent data is formatted correctly | server app + Android app + strace | The server can parse the information from the client and add that to the XML file in the proper format | Pass |
| **5** | Load the website to see whether it can read from a coordinates XML file and parse the data correctly | web browser | A table listing all coordinate entries is displayed on the web page | Pass |
| **6** | Load the website to see whether it can display all the coordinates read on a Google Map | web browser | The Google Map shows all the coordinates in the XML file as pins on the map | Pass |
| **7** | Click on the "All Current Positions" button to see whether the website can show just the last known position of each unique device | web browser | All unique devices are shown at their last known locations with no duplicates or missing entries | Pass |
| **8** | Enter a valid MAC address and click on the "History Of One MAC" button to see whether the website can show the history of the specified device | web browser | The location history of the device is shown in both table form and as pins on the Google Map | Pass |
| **9** | Enter an *invalid* MAC address and click on the "History Of One MAC" button to see whether the website can show the history of the specified device | web browser | No history is shown on either the table or on the Google Map, and the website does not crash | Pass |
| **10** | Multiple devices update their position while the website is set to "All Current Positions" mode and automatic refresh is turned *off* | web browser + server app + Android app | Last known positions of the devices involved are shown on both the table and the Google Map, not updated live | Pass |
| **11** | Multiple devices update their position while the website is set to "History Of One MAC" mode, with a valid MAC address specified, and automatic refresh is turned *off* | web browser + server app + Android app | The location history of the device is shown in both table form and as pins on the Google Map, with the most recent pin in red and all other pins in green, not updated live | Pass |
| **12** | Multiple devices update their position while the website is set to "History Of One MAC" mode, with an *invalid* MAC address specified, and automatic refresh is turned *off* | web browser + server app + Android app | No history is shown on either the table or on the Google Map, and the website does not crash | Pass |
| **13** | Multiple devices update their position while the website is set to "All Current Positions" mode and automatic refresh is turned on | web browser + server app + Android app | Last known positions of the devices involved are shown on both the table and the Google Map, updated live | Pass |
| **14** | Multiple devices update their position while the website is set to "History Of One MAC" mode, with a valid MAC address specified, and automatic refresh is turned on | web browser + server app + Android app | The location history of the device is shown in both table form and as pins on the Google Map, with the most recent pin in red and all other pins in green, updated live | Pass |
| **15** | Multiple devices update their position while the website is set to "History Of One MAC" mode, with an *invalid* MAC address specified, and automatic refresh is turned on | web browser + server app + Android app | No history is shown on either the table or on the Google Map, and the website does not crash | Pass |
| **16** | A single device updates its position in "GPS" mode while the website is set to "History Of One MAC" mode, with the device's MAC address specified, and automatic refresh is turned on | web browser + server app + Android app | The location history of the device is shown in both table form (no IP, accurate position) and as pins on the Google Map, with the most recent pin in red and all other pins in green, updated live | Pass |
| **17** | A single device updates its position in "Network" mode while the website is set to "History Of One MAC" mode, with the device's MAC address specified, and automatic refresh is turned on | web browser + server app + Android app | The location history of the device is shown in both table form (no IP, inaccurate position) and as pins on the Google Map, with the most recent pin in red and all other pins in green, updated live | Pass |
| **18** | A single device updates its position in "GPS + Network" mode while the website is set to "History Of One MAC" mode, with the device's MAC address specified, and automatic refresh is turned on | web browser + server app + Android app | The location history of the device is shown in both table form (has IP, accurate position) and as pins on the Google Map, with the most recent pin in red and all other pins in green, updated live | Pass |
| **19** |  |  |  |  |
| **20** |  |  |  |  |