

[Document title]

[Document subtitle]

[School]

[Course title]

COMP4985 A3 Design Document

# Android GPS WebApp Program

## Message Data Structure

We are using json objects streamed as bytes

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Name** | **Description** |
| Mac | Varchar9170 | Mac address of the device sent |
| Username | Varchar(32) | Username of the client is held here |
| ipaddr | Varchar(45) | The ip address (ipv6) |
| Longitude | Double | The longitude of the capture |
| Latitude | Double | The latitude of the capture |
| time | Datetime | The time of the capture |

# System Architecture



# Server

## Server Finite State Machine Design



## Server Pseudo Code

### Start

**Routine Name:** start or main   
**Activity Flow:**

1. Declare the port to listen to
2. Declare the Database information

### Get Libraries

**Activity Flow:**

1. Get the socket library
2. get the SQL database library
3. proceed to Create Socket [STATE]

### Create Socket

**Activity Flow:**

1. create a server TCP stream socket on the port
2. Procced to Idle [STATE]

### Listen Socket

**Activity Flow:**

1. Wait for an incoming connection request
2. When a connection is made, save the socket as *clientSocket*
3. proceed to Handle Client Connect [STATE] {*clientSocket*}
4. If the user exits, proceed to Exit [STATE]

### Handle Client Connect

**Activity Flow:**   
1. Get the socket library

2. get the SQL database library

### Accept

**Activity Flow:**   
1. Accept the incoming connection request

2. proceed to Receive Data [SUBSTATE]

### Receive Data

**Activity Flow:**

1. Wait for client to send GPS data; store as *buffer*
2. Convert *buffer* to JSON data and store as *jsondata*
3. proceed to write DB [SUBSTATE] {*jsondata*}

### Write DB

**Activity Flow:**

1. Map the values of the json data to database table values
2. Sanitize the data
3. Insert the data into the database
4. proceed to write Idle [STATE]

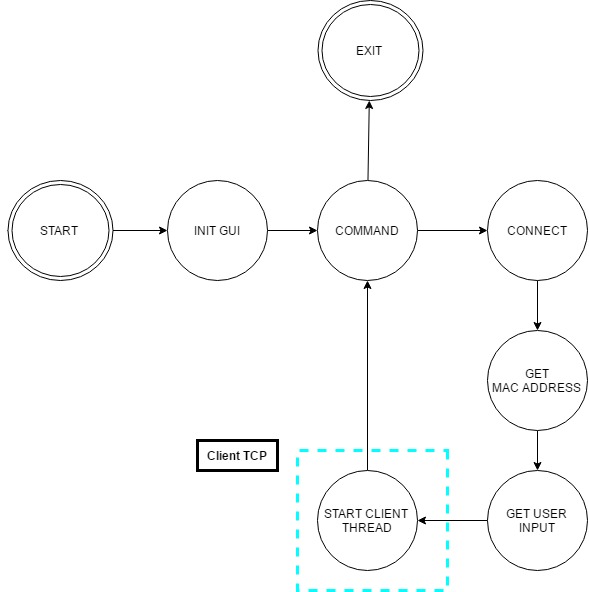
### Exit

**Activity Flow:**

1. Close Socket & Program

# Client

## Client Finite State Machine Design



## Client Pseudo Code

### Start

**Routine Name:** onCreate   
**Activity Flow:**

* Set permission for Wi-Fi
* Set permission for Location
* Go to Initialize GUI [STATE]

### Initialize GUI

**Activity Flow:**

* Set the content view
* Display mac address
* Setup input boxes for user
* Setup object to select various intervals of frequency
  + Seconds
  + Minutes
  + Hours
* Set the default frequency to Minutes
* Go to Command [STATE]

### Command

**Activity Flow:**

* If user updates any input boxes
  + Update values
* If user presses “Connect” button
  + Go to Connect [STATE]

### Connect

**Routine Name: ConnectButton**

**Activity Flow:**

* Check what the user set as the interval of frequency
* Create a frequency in milliseconds that follows the user’s request
* Start a timer with a delay of 1 millisecond
* Set the timer’s gaps by the frequency requested by the user in milliseconds
* Create a client thread to send data every time the scheduled timer is ready

### Client Thread

**Routine Name: run**

**Activity Flow:**

* Get the mac address of the current device
* Check the user’s input for updates
* Go to Create Socket [STATE]

### Create Socket

**Activity Flow:**

* Create a socket
* Set the socket information to the user’s input
  + Use the supplied “Server IP address”
  + Use the supplied “Server port number”
* Go to Send [STATE]

### Send

**Activity Flow:**

* Get a current timestamp
* Create a new JSON object
* Set the JSON object’s values to those entered by the user and the current latitude, longitude and timestamp
* Send the data through the socket
* Close the socket

# Web App

## Web App Finite State Machine Design

## 

## Web App Pseudo Code

### Idle

**Activity Flow:**

Request Googlemap API with API key and the callback function.

Connect to Database.

### Connect to Database

**Activity Flow:**

Connect to Database

if (success)

go to ‘Update Locations’.

else

go to ‘Show Error’

### Update Locations

**Activity Flow:**

Prepare a select query to obtain the information of all locations.

Execute the select query.

Store the results.

for 0 to the number of rows

bind(id, macAddress, username, longitude, latitude, time);

fetch();

Call addLocation(id, macAddress, username, longitude, latitude, time);

### Show Errors

**Activity Flow:**

Display the error message on the screen.

### Request Google map API

**Activity Flow:**

Connect to ‘https://maps.googleapis.com/maps/api/js?key=<API KEY>&callback=initMap>’.

var dataManger = new Map();

Function initMap Start

Initialize Google map.

var myOptions = Customize the options of Google map;

Go to ‘Display Locations’.

Function End

### Display Locations

**Activity Flow:**

Function addLocation(clientId, locationId, longitude, latitude, title) Start

mapData = new Map();

mapData.AddEntry(key = ‘lgn’, value = longitude);

mapData.AddEntry(key = ‘lat’, value = latitude);

mapData.AddEntry(key = ‘title’, value = title);

var clientDataMap = dataManger.getValue(key = clientId);

if (clientDataMap == null)

clientDataMap = new Map();

clientDataMap.AddEntry(key = locationId, value = mapData);

dataManger.AddEntry(key = clientId, value = clientDataMap);

Function End

var map = new google.maps.Map(the ID of a canvas for Google map, myOptions);

foreach (clientDataMap in dataManger)

foreach (mapData in clientDataMap)

var position = new google.maps.LatLng(mapData.getValue(key = ‘lat’),

mapData.getValue(key = ‘lgn’);

marker = new google.maps.Marker(position, map, mapData.getValue(key = ‘title’));