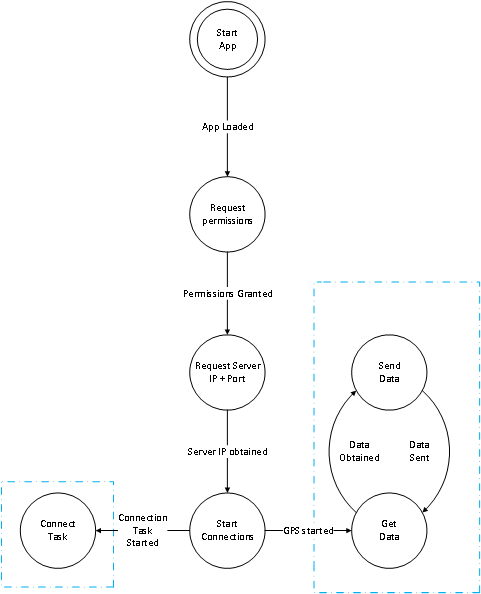
Android GPS Design

## Client done by: Matt Goerwell

## Client FSM:



## Client Pseudo:

**START APP**

The entry point of the app. It will initialize our storage variables, and proceed onwards.

1. Load app
2. Initialize settings storage
3. Get location manager
4. Get device IP
5. Go to Request Permissions State

**REQUEST PERMISSIONS**

This state merely ensures that we can use the permissions in our manifest

1. Request GPS permission
2. Request Internet Permissions
3. Go to Request Server IP + Port State

**REQUEST SERVER IP AND PORT**

This state will function as a pair of textboxes that respond once the user clicks a submit button. It will check that the server exists via lookup, then proceed to the Set-up TCP Connection State

1. Extract IP
2. Extract Port
3. Go to Start Connections State with IP and Port

**START CONNECTIONS**

This establishes the connection to be used by the client.

1. Parse IP and Port to usable states
2. Handle Errors
3. Start Connect Task with IP address and port
4. Request GPS updates, using Get Data State as the update handler

**GET DATA**

This state serves as the callback function that is used whenever the system detects a location update

1. Parse location update for latitude and longitude
2. Create a message containing device IP, latitude and longitude
3. If Connection is set up
   1. Go to Send Data state with message

**SEND DATA**

This state takes the passed in message and echoes it to the server.

1. Access server input stream
2. write message to server.

**CONNECT TASK**

This state represents the steps needed to actually connect to our server, which due to android limitations must be implemented on its own thread.

1. Parse IP and port
2. Attempt to connect
3. If connection failed
   1. disable GPS updates
4. While GPS updates are on
   1. Ensure GPS update status is current (*thread consideration*)
5. Close Socket

## Web Server done by: Alex Zielinski

## Web Server FSM

C:\Users\Bobo\Downloads\AlexZielinskiWebServer.png

## Web Server Pseudo

**AUTHENTICATE USER**

When the user enters the web page they need to be authenticated. So when the user enters the web page they will be prompted to enter a username and password.

1. Prompt user with a username and password field
2. Compare with username and password credentials saved in webserver config file
3. Grant access if successful (else notify user of incorrect username or password and re-prompt)

**INIT GOOGLE MAPS API**

This state is responsible for initializing Google maps within the web browser.

1. Init Google Maps API key
2. Display Google Maps in web browser

**READ COORDINATE VALUES FROM DB AND PLOT**

This state is responsible for retrieving client coordinate values stored in a database. It will then plot the points on the map and link the points together to show the (real-time) movement history of a client

1. Read database client info entries
2. Check if client is a new client
   1. If new client then store the client ID locally on the web server and assign a color
3. Plot coordinate point with according client color
4. Link previous point with new point

## Server done by: Robert Arendac

## Server FSM:

## 

## Server Pseudo:

**PROCESS COMMAND-LINE ARGS**

1. If more than two arguments were specified
   * 1. print error and usage message
     2. Exit
2. if one argument is specified
   * 1. set port as default
3. otherwise
   * 1. set port as argument 2

**SETUP TCP SOCKET**

1. Create a socket to listen to
2. Set socket options
3. Initialize address information
4. Bind address to the listen socket
5. Listen for connections
6. Set up an array of client descriptors to be used

**MONITOR CONNECTIONS**

1. While the server is alive
   * 1. If the listen socket is set
        1. Go to ADD NEW CLIENT
     2. Go to READ SOCKET

**ADD NEW CLIENT**

1. Accept the connection
2. Add the new socket descriptor to the client container
3. Add the new socket to the socket set
4. Make sure the max number of clients has not been reached

**READ SOCKET**

1. Go through each client in the client container
   * 1. If the socket is set
        + 1. read the socket
          2. If connection was closed

Go to Remove client

Exit state

* + - * 1. Go to STORE INFO IN FILE

**REMOVE CLIENT**

1. Set that clients value in the client container to an invalid value
2. close the socket
3. Increase the number of available clients

**STORE INFO IN FILE**

1. Check to see if the JSON file exists
2. If it does exist
   1. Open it for reading and writing
   2. Format a JSON string for appending
   3. Seek to the closing brace in the file
3. Otherwise
   1. Open the file for writing
   2. Format a JSON string for a new object
4. Place the JSON string in the file
5. Close the file