

Sense

Mobile Sensor Cloud – Crowdsourcing

Submitted by Project Group 6

Isha Patel

Priya Singara Velu

Saraswathi Thanu

Shraddha Kannav

CMPE 277 – Smartphone Application Programming

Prof. Chandrasekar Vuppalapati

Table of Contents

1 Project Description 4

1.1 Abstract 4

1.2 Project Goals and Objectives 5

1.3 Background, Problems and Motivation 6

1.4 Project Applications and Impact 7

2 Requirements 8

2.1 Functional requirements 8

2.2 Non-Functional Requirements 10

2.2.1 R2.2.1 Performance 10

2.2.2 R2.2.2 Capacity 11

2.2.3 R2.2.3 Usability 11

3 Web UI Design Principles 11

4 Storyboard and Wireframes 15

4.1 Storyboard 15

4.2 Wireframe 16

5 High Level Architecture Design 19

6 Component Level Design 21

6.1 Vendor registration module 21

6.2 Sensor Registration Module 22

6.3 Data Acquisition and storage module 23

7 Sequence – Workflow 25

7.1 Sequence 25

7.2 Workflow 25

8 Cloud Technologies Used and Descriptions 28

9 ReSTFul Interfaces 33

10 Design Implementation 35

10.1 Server Side Design 35

10.2 Client Side Design 36

11 Testing (UI or Stress test) 37

12 Automation Testing - Selenium 41

12.1 Pre requisites 43

12.2 Steps for installing Selendroid 43

13 Design Patterns 44

13.1 Front end design pattern 44

13.2 Middle tier design pattern 45

13.3 Cloud Design pattern 45

13.3.1 Relational Databases 46

13.3.2 Distributed component design pattern 47

13.3.3 Loosely coupled design 47

14 User Interface – Screens 48

15 Future Scope 51

16 Profiling 53

17 References 53

18 Presentation 55

# Project Description

## Abstract

Sensors have become an integral part of today’s technology. Today, the use of sensors is rapidly increasing than never before. This increase in sensors has led to the exponential increase in sensor data. The conventional methods of controlling and managing the sensors are through sensor networks and through Internet of things (IoT) web portals. Each of these methods has their issues when it comes to handling a large number of sensors and huge volumes of data. Sensor networks are very expensive to deploy and manage. In addition, there is poor accessibility with respect to data sharing. IoT methods have poor scalability, which makes them an inefficient way to manage the increasing sensors. Also, dissemination of such huge data to the public is laborious and expensive. In this project, we address the above issues by presenting a mobile sensor cloud infrastructure as a service (IaaS) application. This application overcomes the accessibility, scalability issues by using a cloud- enabled system and the high-cost issue by leveraging the crowd-sourced sensors and data.

Our approach involves simulation of sensor infrastructure and construction of a prototype that allows the user to allocate the sensors and provision the data. The application has been developed in Android operating system in conjunction with cloud services. We have used the agile software development methodology for application development. For requirements analysis, we used a combination of market research, literature review and considered the challenges in existing solutions. We used Object oriented design methodology. We have modeled various business processes using UML and implemented them using software design patterns. We have performed functional and nonfunctional testing on the application using a combination of manual and automation testing tools. In project management, we have factored in the cost, time, software quality, human resources and risks associated.

The deliverables for this project include a mobile sensor cloud infrastructure as a service (IaaS) with a mobile app and a project report. This application serves different audiences. A Sensor Network host or Smart Homeowner can join the crowd-­‐sourcing community to become a part of sensor infrastructure provider. A customer in this application can be considered as a user who would request the sensor infrastructure and data in a on-­‐ demand and a pay as you go model. Users can also view the data analyzed in graphical format. In addition, an administrator would be able to monitor the system and the data. The project report contains standard deliverables associated with application development including requirements analysis, preliminary design, implementation plan and test plan. Through this project, we aim to leverage the existing sensor hardware technologies to its full potential in order to overcome the limitations and take ‘Sensor Cloud Computing’ to the next level.

## Project Goals and Objectives

The goal of this project is to provide users with a rich crowd-sourced mobile application which would enable them to share their sensor data, view crowd-sourced sensor data and even provision other users’ sensors in an on demand basis. The benefit behind the application is the ability of a smartphone device user, businesses, governments etc. to gain access to powerful, processed and visualized data at almost zero cost by leveraging the potential of sensors in Smartphones and External sensor networks. This application specifically focuses on the use of environmental sensors and their data to derive valuable inferences and provide recommendations.

* Develop an easy to use mobile application for customers to register, provision sensors and share and view environmental sensor data.
* Represent the accumulated data through interactive maps to the customer, which the customer can view for different locations.
* Perform Analysis on the accumulated environmental data to perform predictions and provide recommendations about temperature, humidity and exposure to pollutants.

## Background, Problems and Motivation

Wireless sensors are starting to transform the world into a Sensor driven “Smart” world. These tiny powerful devices disrupt the conventional ways of using applications and bring a new wave of innovation to them. Sensors are the core of all “Smart” applications. Applications relating to healthcare, homes, cities, environment etc. are rapidly transforming into “Smart”-applications due to the immense potential and use of the sensors and sensor networks. For example, in a smart-home, one can connect all the wireless appliances, control and monitor them using a mobile phone. Numerous health conditions are monitored continuously using smart health care sensors; Environmental activity is monitored using sensors to make weather prediction, exposure estimation and much more. Today, a world without sensors and sensor networks is unimaginable.

Although sensor networks have immense technology and business potential, there are certain drawbacks associated with it. Sensor networks are extremely time-intensive and labor intensive to deploy. Many businesses are unable to utilize the power of sensor networks for the very same reason. Hence businesses are pushed to obtain sensor data from third party sources. Accessibility to such third party resources is often limited and poor. IOT methods of data sharing often fare poorly in scalability. Finally, dissemination of the sensor data is laborious and expensive.

Our project aims to address the above-mentioned issues by providing a unified platform where sensor data providers and consumers can come together in order to share and utilize the sensor data easily.

The key concept behind the project is crowdsourcing of sensors and sensor data. The sensors can be a regular sensor network, a smart-application sensor network and even sensors inside smartphones. Smartphones today come with numerous sensors built in them. They can be broadly classified as Environmental sensors, Position sensors, and Motion sensors. How are they different from regular external sensors? The fact is, they are not. Hence, in this project we emphasize equal importance to smartphone sensors as the other sensors and leverage its full potential.

## Project Applications and Impact

The smart clean environmentally friendly mobile app provides an interactive maps to customers, external sensor network communities and government agencies displaying real-time data like temperature, humidity, and pressure for the respective user locations. The app also helps in detecting toxic chemical leaks and pollution (both noise and air) levels. The app also send alert messages to the users mobile phone informing the current pollution level increase or raise in temperature etc. so that the users can be prepared accordingly without any surprises later.

By using the embedded mobile phone sensors or portable, small and usually Bluetooth or Wi-Fi enabled sensors, we can significantly increase the amount and precision of environmental data through crowdsourcing. These environmental sensors have the potential to really change the way that data is gathered, analyzed and consumed.

The important of crowdsourcing comes when we cannot fully rely on the data coming from the government's environmental sensors at their monitoring stations, because it doesn't give the whole picture to someone who's living near interstate or parking garage or industrial area. We need to consider the crowd sourced data meaning sensors data from individual smart phones as well. Having specific, real-time information will not only help someone with asthma know areas to avoid on any given day, but also provides environmental researches or scientists a better picture of where, when and why pollution is happening, which is necessary to take steps to make it better. The app will also provide recycle program based on the issues rose.

# Requirements

The functional and non-functional requirements for this application are listed below.

## Functional requirements

**R 1.1 Manage Audience**

* System shall allow registration.
* System shall allow membership.
* System shall provide login credentials
* System shall allow authentication.
* System shall allow viewing the profile.
* System shall allow updating the profile.
* System shall allow deleting the profile.
* System shall provide facilities like sign up and sign in Google and Facebook Logins

**R1.2 Manage Sensor Vendors**

* System shall allow two types of sensor vendor. They should be Smartphone sensor vendor and External sensor vendor
* System shall provide login credentials
* System shall provide facilities like sign up and sign in Google login
* System shall also provide facilities to view the profile.
* System shall allow to view sensor information
* System shall allow to choose sensors and give permissions
* System shall check that vendor will have network connection.
* System shall have vendor’s Geo-tagging on.

**R1.3 Manage Sensor registration**

* System shall allow one or more sensors to be registered.
* System shall check if sensor is available and accessible.
* System shall calibrate the sensor.
* System shall check for compatible interfacing and connection

**R1.4 Manage Data**

* System shall collect the data at regular intervals
* System shall reduce the data collection interval if value is out of threshold.
* System shall increase the interval if value is constant for longer time.
* System shall start collecting the data when user is connected to wireless connection.
* System shall store user data encrypted
* System shall protect the data while in transit
* System shall provide historic data

**R1.5 Manage interactive maps**

* System shall have access to the Google maps API’s.
* System shall access the current location.
* System shall overlay the data on maps.
* System shall allow for filters by location, time, days, and sensor type on the map.

**R1.6 Manage Sensor Monitoring**

* System shall indicate the health of the Sensor using different icons
* System shall show the location of sensor
* System shall show functional and non-functional sensors.
* System shall provide graphical representation of the sensor monitoring.

## Non-Functional Requirements

### R2.2.1 Performance

* System shall have very efficient design for big data analytics that will bring up the Google interactive maps in seconds on mobile phones.
* System shall stream the data at a very fast rate without much delay and providing real time updates.
* System shall collect and store all the sensor related data from mobile phone and external sensor networks at a faster speed.

### R2.2.2 Capacity

* System shall be able to store sensor data continuously on cloud without any issues.
* System shall be able to display environmental data (ambient humidity, ambient temperature, and ambient pressure) for approximately 25-30 places in a smart city in the customer’s mobile phone.
* System shall be able to display list of polluted places in a smart city.

### R2.2.3 Usability

* System shall have user-friendly app, which will be easy to use for customers, sensor vendors, Government agencies and communities.
* Customers without any kind of training shall use system.

# Web UI Design Principles

Designing your mobile application should be such that it is easy to use for any group of people. Generally your app targets some audience but it should generally be simple and subtle. Design is a very important aspect of app making as it is the first view people are going to see and first impression has to be good. Appearance matters a lot we agree or don’t. If the app is targeting young crowd then different style and outlook is needed. Your design can be complex and sometimes messy if needed. If your focus is professionals it has to be subtle and elegant. If targeting children it has to be bright and vibrant.

Some of the standard principles are developed so that everybody maintains some uniformity and basic rules and principles. These are not mandatory but it’s good if most of them are implemented. Some examples are:

* How your audience thinks: In Mobile sensor cloud people what is fast performance and fast display of result. The display can be as simple as you make or can be somewhat complex for performing main task. I.e. registering and unregistering sensors. Usability can be enhanced by continuously displaying things that interest the user according to his age group. Or while sensor data is uploading users can be kept engaged by showing some previous analysis reports or some of his savings/benefits.
* Usability: App has two factors: visual design and utility factor. The latter is the one that determines the success rate of the app. As user is the only source who is going to visit our web page and clicking on different aspects, we have to attract them. This ideology has evolved into a whole new user centric app design. The placement of boxes like text box, labels etc. is very important as user will develop friendliness with these components only. We all must have come across some apps where even normal presentation is messy. Some other factors that needs to be taken care of are also less readable stuff because user reads less. They scan more. Also quality matters more over quantity. Now nobody reads paragraphs to understand things. Pictorial presentation is far more scan able and fast. SO in our app we better display sensor pictures to make sense and increase readability.
* Fast Gratification: Users don’t spare much time on reading, navigating and researching your app structure. One of the key features to attract users is giving them instantly what they wants. User hardly spare 1-2 minutes on an average looking at their destined app. Your design should be such that user can easily get their work done and leave app with successful result. For sensor registration user needs all the sensor listing and undo process at same place. So we gave sensor registering just through a check box to maintain easy and fulfilling task done in seconds.
* Suggestions: Users are not always smart or not always know the in and out of how system works. Sometimes they might end up making bad or irrelevant decisions. So prompting user towards right direction by continuously popping messages or telling them the outcome f their task and giving them an option to undo it is a good way to enhance user experience. Also give as much control to user as possible. User should never feel things are going out of their way or expectations and they are entangled with the app structure. Giving user control over their usage makes them feel welcoming and use it more. Try asking user at various stages about their needs and end result.
* Feature explosion: In some apps users are not aware till end that your app has this feature. Sensor management and user expectation should be fetched at the beginning only rather than waiting till end. User should be aware about sensors available, how to register, how to set your goals for savings, data upload, cloud database management etc. User should get status of all these components whenever they ask. Writing should be as minimal as possible and effective. Messages should be conveyed in less writing as user is not going to read long stuff. Alerts and pop windows are also very important means of educating user about your app and controlling their reckless decisions. New feature added with should be made aware by continuous alerting and popping. User can make the best out of an app only when he/she knows all the features and their drawbacks.
* White spaces are important: A good UI design doesn’t mean that you fill your web page with contents everywhere. Some apps demonstrate this.



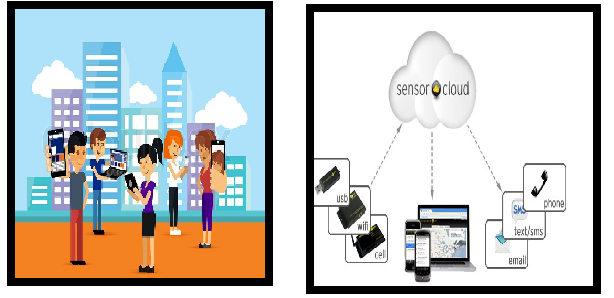
Figure 1: White spaces example

White spaces are equally important and increase readability of the user. You can always fill all your substance in one web page but is not recommended. Our app design has a nice grid view which enhances abstraction from underlying architecture and features available.

# Storyboard and Wireframes

## Storyboard





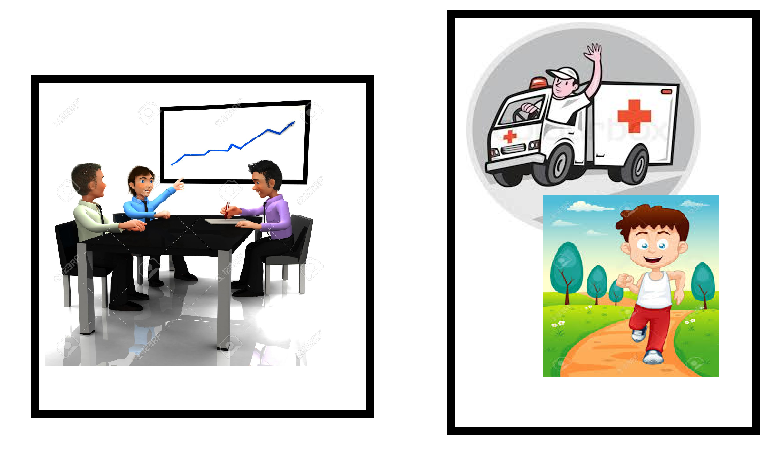


Figure 2: Storyboard

## Wireframe

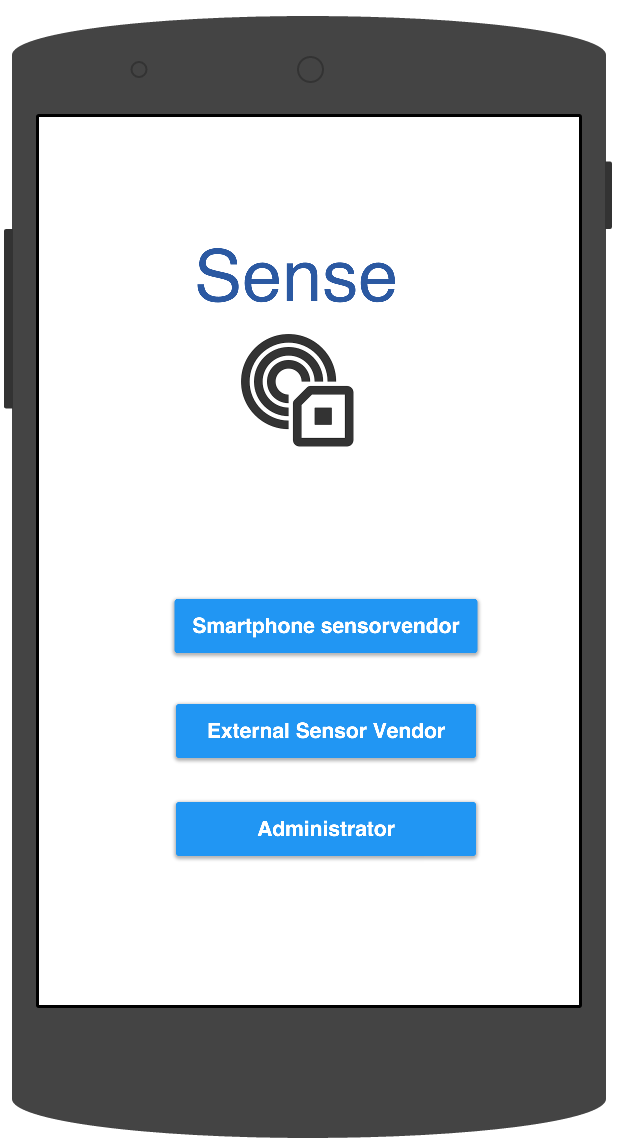
* Wireframe is an image model that is represented in a three dimensional frame representing vertical lines that defines structure of your web app. When you select a new window in android studio they ask you to select from predefined custom templates. This is nothing but wireframe.
* One of the important parts in UI design is wireframe. It basically is the layout of your page. And a good practice is to maintain a uniform layout throughout your app. It is decided on the basis of how you want to process user information, one, which are already present, and one which user enters. The basic structure is header part where your most important guideline heading or other details lie. Second is body and last is footer. By understanding your user you can also customized your wireframe design so user encourages usage and increases app sale.

Figure 3: Mock-Up - First Screen

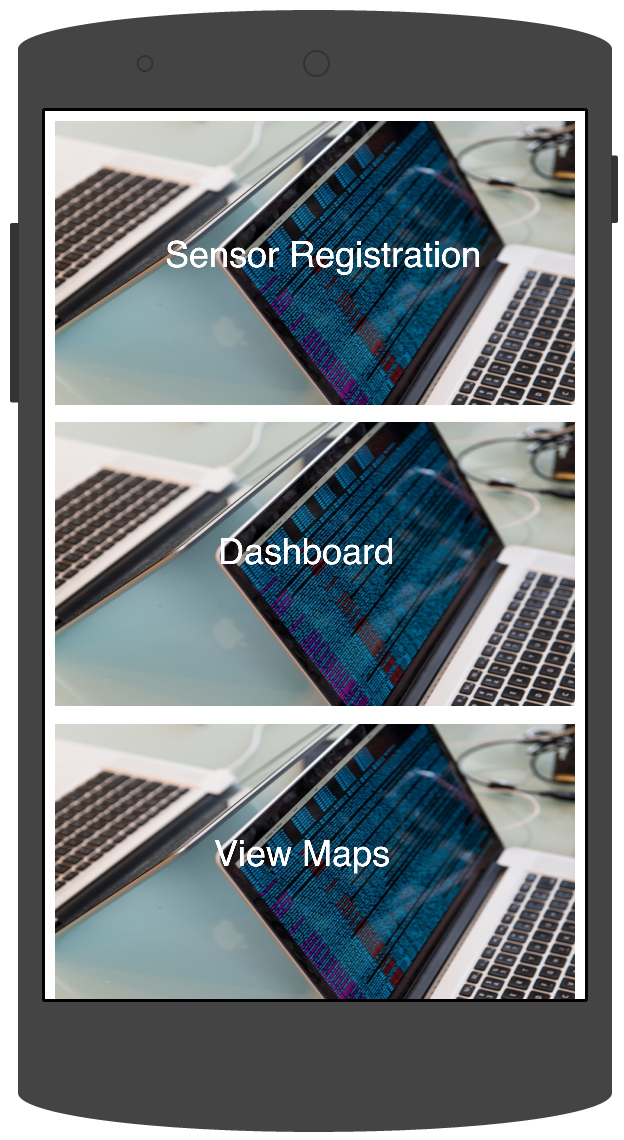


Figure 4: Mock-Up - Home Screen

* It is basically layout of your architectural design. You definitely want to make sure your information is displayed thoroughly and nicely. You cover every aspect of your design and make wise usage of the space available. There are behaviors related to menu and button features and usage pattern of your user. There is a definite arrangement of header, footer and body components. Some components can be completely omitted if they are not relevant or can be eliminated from certain pages where more emphasis needs to be given to body part.
* In our app list view and grid view are the basis of the design layout. User is generally interested in the list of sensors available to perform their task, use them to start getting data and uploading only specific part of the design.

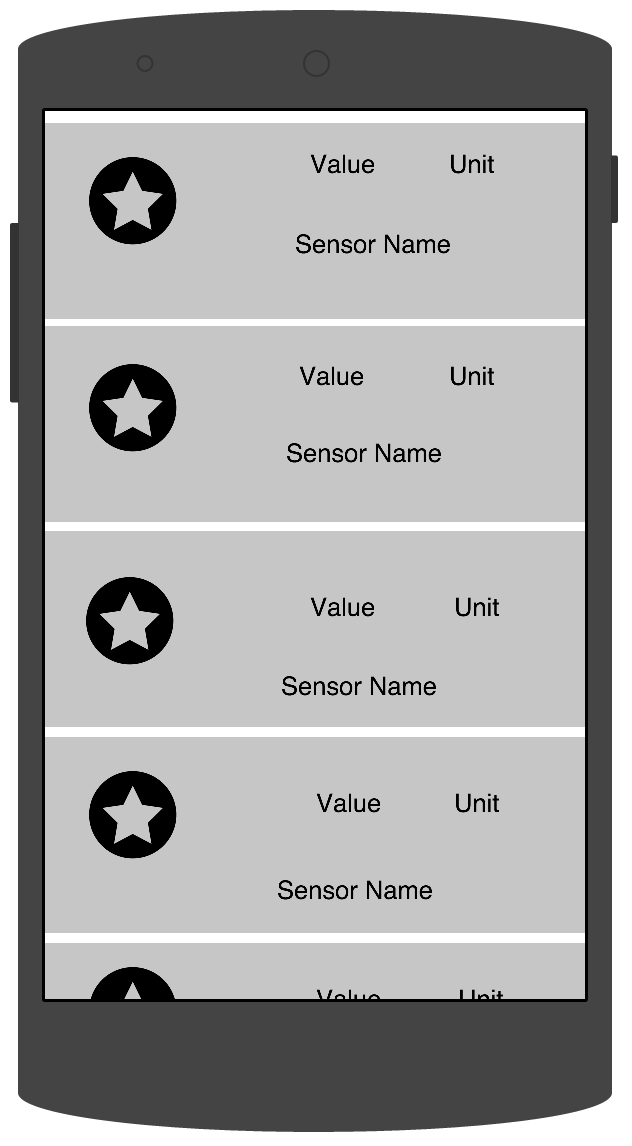
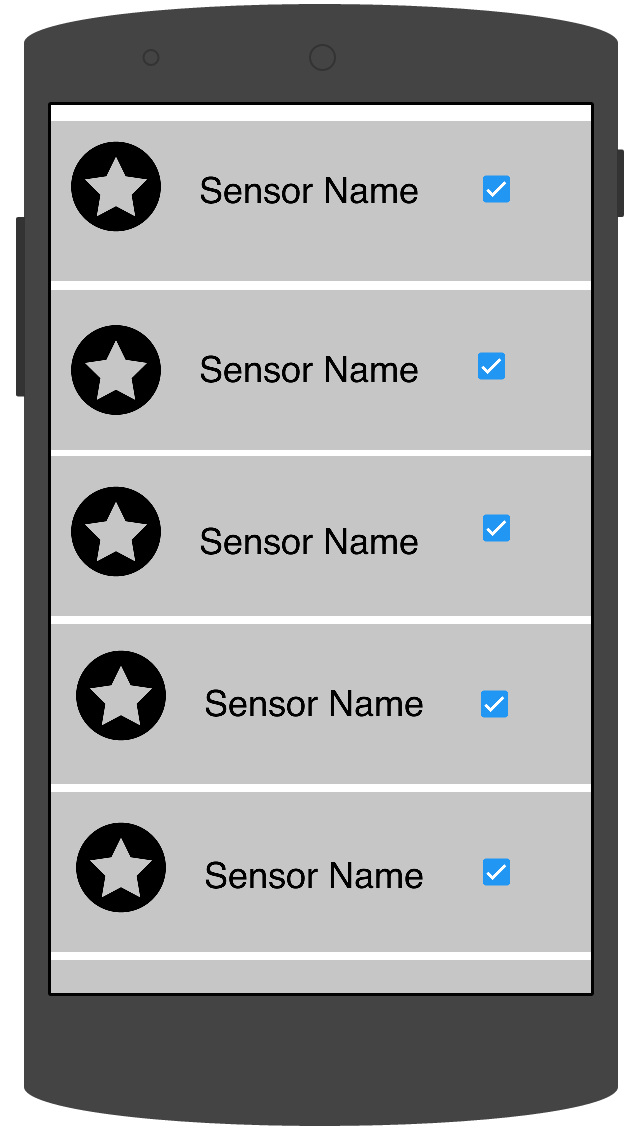


Figure 5: Mock-up - dashboard

Figure 6: Mock-up - Register Sensors

* Sensor display is an important feature and registering/unregistering should be easy to maintain for the user. Eventually there is just a body part where list is displayed. Header part contains multiple tabs and with each increase in a new feature a new tab is added and user is notified.

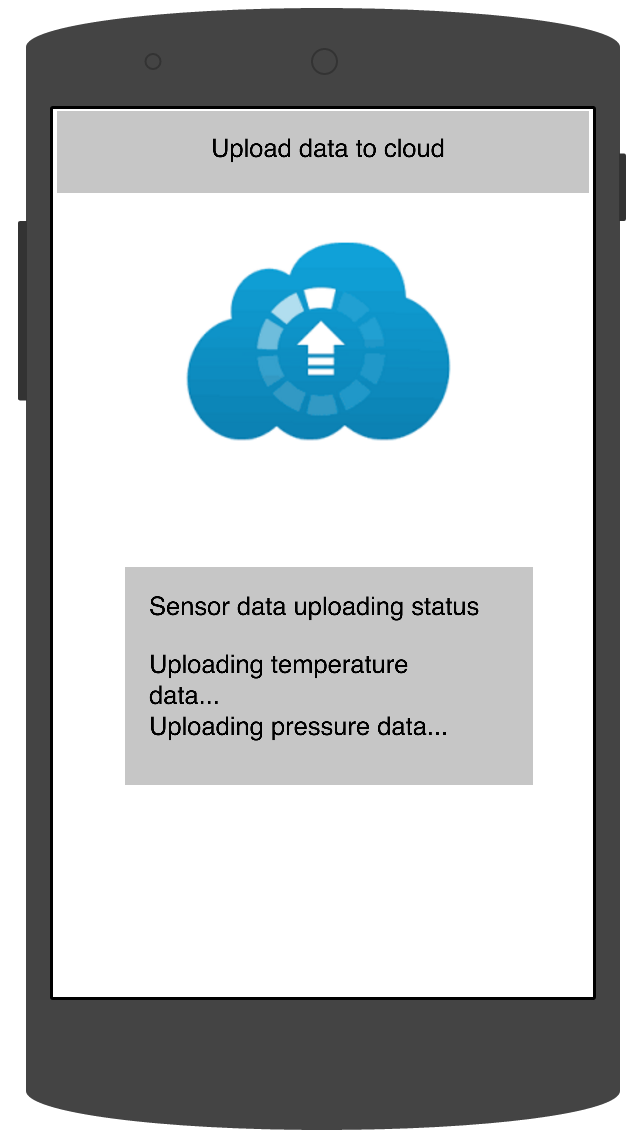
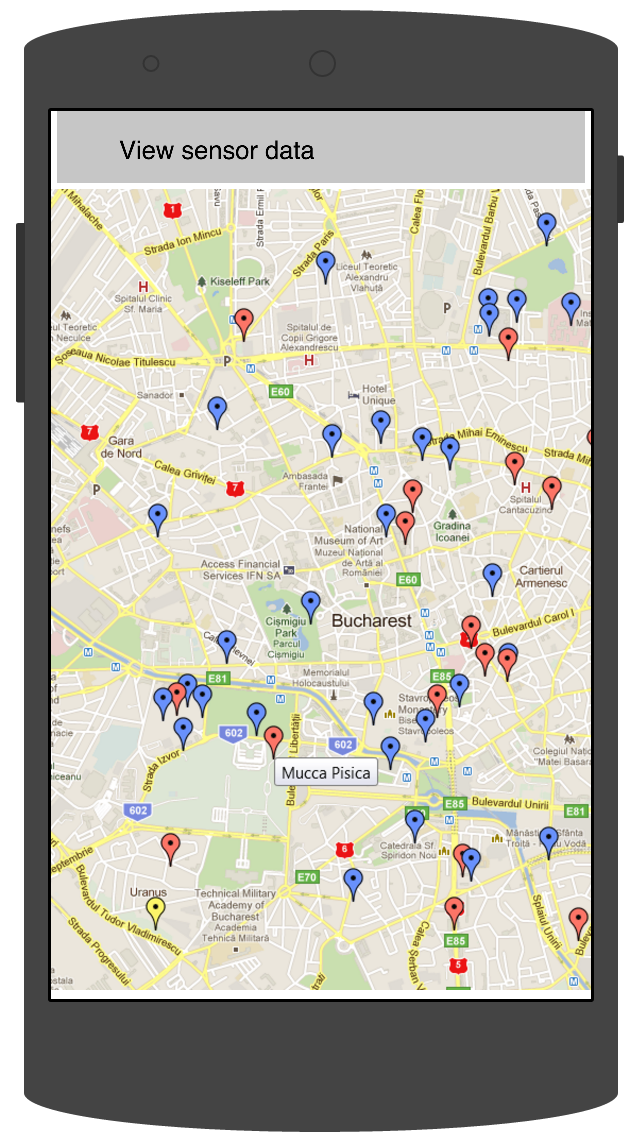


Figure 7: Mock-up - Maps screen

Figure 8: Mock-up - Upload data screen

# High Level Architecture Design

The architecture diagram shows the key components of the crowd sourced mobile sensor cloud system. It has five key components - Mobile Sensor cloud sources, Data Acquisition and Preprocessing Module, Cloud layer, Interactive Mobile Client and the Web Services layer.



Figure 9: Architecture Diagram

Mobile Sensor Cloud Sources: As mentioned earlier, the data providers for the crowd sourced mobile sensor cloud system are the inbuilt sensors in smart phones and any virtually pluggable wireless external sensor networks.

Data Acquisition & Preprocessing Module: The raw data acquisition component involves collection of raw streaming sensor data from various types of sensors like Environmental, Motion, Position sensors. Further processing involves sampling of the data in order to identify patterns, classify the data and build inference model. For e.g. User activity recognition could be used to identify the regularly visited areas by user and alert the user of pollution levels in these areas. Additionally, the data is geo tagged and contextually tagged. Geo tagging is the process of tagging the data with the geographical coordinates. Contextual Tagging is allowing the user to further describe the context of the data. For e.g., the pollution levels at morning, afternoon and night could vary. Hence, allowing such tagging enables to refine the analysis results.

Web Services layer: The use of web services decouples the sensor data providers and the sensing server. This allows for establishing platform independent applications accessing the server and promotes interoperability.

Cloud Sensing Server

The cloud-sensing server is the pivotal part of the application. It acts as the heart of the system connecting the data providers to the data users. The decision to use cloud services is a natural one owing to the huge amount of continuously streaming data. The cloud-sensing server has components to store the streaming sensor data from the data providers, perform computations on the data, and perform data analysis and mining. The Estimation & Processing module does computations of various sensor data. For e.g., the actual calculation pollution levels from the collected sensor data is done in this module. Analytics module provides a comparison of pollution levels across various regions, various time phases, to get an understanding of the progression that has happened.

# Component Level Design

The major components in the application and their design are shown below. This report has adopted the conventional view of the component level design.

## Vendor registration module

Macintosh HD:Users:Shraddha:Downloads:Sensor Vendor Registration Module.png

Figure 10: Vendor registration diagram

The application supports two categories of sensor vendors - the Smartphone sensor vendor and the External sensor vendor.

Smartphone sensor vendor:

A smartphone sensor vendor can be vendor who owns a smartphone that supports at least one sensor, runs on either iOS , Android or Windows phone and has access to the internet. Lack of any of the above requirements would make the vendor unable to share any data and thus not qualify him as a sensor vendor.

External sensor vendor:

An external sensor vendor can be a vendor who has access to a wireless sensor/sensor network and has authority to share that data. Examples of external sensor vendors are a smart home/city/healthcare/environment sensor network owner, a sensor manufacturer, and an individual with access to standalone sensors.

The application intends to provide flexible forms of registration and login options. Currently, it allows vendors to sign in using ‘Sense’ native login feature. Sign in through multiple platforms like Google and Facebook are in the future scope of the application.

Once registered, the vendor information is stored in the Parse cloud database’s User class. Parse provides a ready to use ‘User’ class that supports login, authentication, encryption, password reset, multiple platform support etc. This application leverages Parse’s capabilities to achieve the same.

## Sensor Registration Module

This is one of the core components of the application. The role of sensor vendors in this application is to act as sensor and data providers. This component manages the sensor provider part of the sensor vendors. Sensor registration needs information like Sensor name & model , manufacturer, data of manufacture , range, calibration , data format and unit, interface mechanism(wired, wireless , Bluetooth) , location , delay in data streaming etc.

For smart phone vendors, most of the required data can be obtained from the device in which the application is running. Additional information needs to be provided by the user.

Macintosh HD:Users:Shraddha:Downloads:Sensor Registration.png

Figure 11: Sensor registration Module

For external sensor vendors, information has to be provided manually, pairing devices has to be performed if required and the interfaces should be checked.

Registration data would eventually be stored in the Parse database, which then can be retrieved, updated, searched, deleted etc. This data can also be viewed by the admin for monitoring and control purposes.

## Data Acquisition and storage module

This module is another core module of the application. The ultimate goal of the application is to gather crowd-sourced sensor data and provide it to users in the most efficient way possible.

Once vendors register their sensors, the next step is to collect the data and store it in the cloud. Data upload happens on demand and can be turned off at any time by the vendor. Preprocessing of the data includes smoothing the data for stability, cleaning up of data. In the future, preprocessing module would include feature extraction, classification, and sampling in order to prevent dirty data getting stored on the cloud. The application uses HTTP callbacks and REST API for communicating with the Parse database. Data is sent as data objects containing the sensor type, data and time stamp information. A predefined interval is set in order to prevent excessive API calls to the cloud. In future implementations, the application would follow a ‘batching process’ in order to send sensor events to the cloud.

Macintosh HD:Users:Shraddha:Downloads:Data Acquisition and Storage.png

Figure 12: Data acquisition module

Crowd-sourced data representation: This is one of the core outcomes of the application. Crowd-sourced Sensor data collected through the above step is represented on interactive Google maps. The application provides capabilities like location based, date based and time based filtering options for the user to filter the data. Prediction and analytics are on the future scope of this application.

# Sequence – Workflow

## Sequence

The Diagram below is uploading sensor data sequence diagram

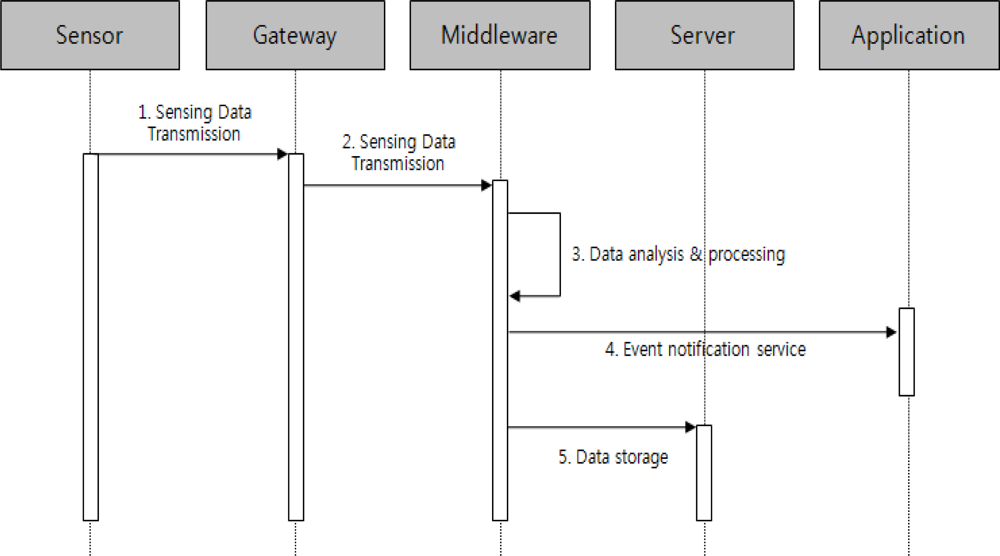


Figure 13: Sequence Diagram

## Workflow

The diagram below shows the workflow of the entire application. The application follows a different workflow for different types of audience.

Macintosh HD:Users:Shraddha:Downloads:Workflow diagram.png

Figure 14: Workflow

The application supports a smartphone sensor vendor, external sensor vendor , administrator and in the future, it would support the sensor data consumer. Registration, Login and Authentication are common for the above audience.

Smartphone sensor vendor: Upon logging in successfully, the smartphone sensor vendor can

* View various sensors available in the device , their dynamically changing data based on the environmental conditions.
* View the crowd sourced sensor data of Pressure on Google maps as an interactive representation.
* Choose to register or unregister any sensor at any time
* Choose to upload the sensor data on demand and receive notifications about the uploading process

External sensor vendor: Upon logging in successfully, the external sensor vendor can

* Register a sensor by providing the required sensor information
* Test the interface, connection of the sensors to ensure proper data continuity
* Initiate data uploading on demand
* Unregister any sensor at any time upon need.

Administrator: Upon logging in successfully, the administrator can

* Monitor sensors health using an interactive dashboard
* Add, delete, configure vendor and user accounts
* Add, delete, configure sensors from different vendors
* Cater to customer requests upon need

Sensor data consumer: Upon logging in successfully, the consumer can

* Request sensors of the required type to be allocated
* Provide the request information
* The application using an allocation algorithm would check for availability of the sensor(s) that could be best fit for the user.
* In case of sensors being available, the application would allocate the sensor(s) and notify the consumer about the allocation.
* If sensors were unavailable, the consumer would be notified and would have to request a different kind of sensor or request the sensor at a different time.

The above feature is in the future scope of the application.

# Cloud Technologies Used and Descriptions

* Parse Cloud: Parse is a powerful Infrastructure as a service model to boost up your app. It is built to power your app on any platform. This reduces the fear of platform dependencies. It has a huge set of inbuilt API to stretch your application achieve multiple task easily and also its easy to integrate with other cloud technologies. API documentation, examples and new features releasing alternate weeks helps to build and enhance features on a continuous basis. Rest API are also available so that HTTP calls can be made directly using standard get ,put ,post and delete commands.
* As infrastructure is provided by this platform we can concentrate more on the rest of the development and enhancements. A comprehensive and detailed description about different platform and user guide with examples make it easy for even a beginner to use. If you are new start with tutorials which explain basic setup, how to access database from your SAAS application. They provide code to connect to cloud from eclipse, android studio, iOS web kit tool, JavaScript etc.



Figure 15: Cloud connectivity with devices

* We have our database in cloud, which is used to store sensor data at regular intervals as predefined by the user or the system. This data can be further used for complex conclusions and creating alerts for user. A table is already formed in the cloud to nicely organize data in tabular form for easy access and readability. All sensor data are merged in the table and some field combination help us get specific data from the cloud. We are making rest calls to access data from the cloud in a secure way.

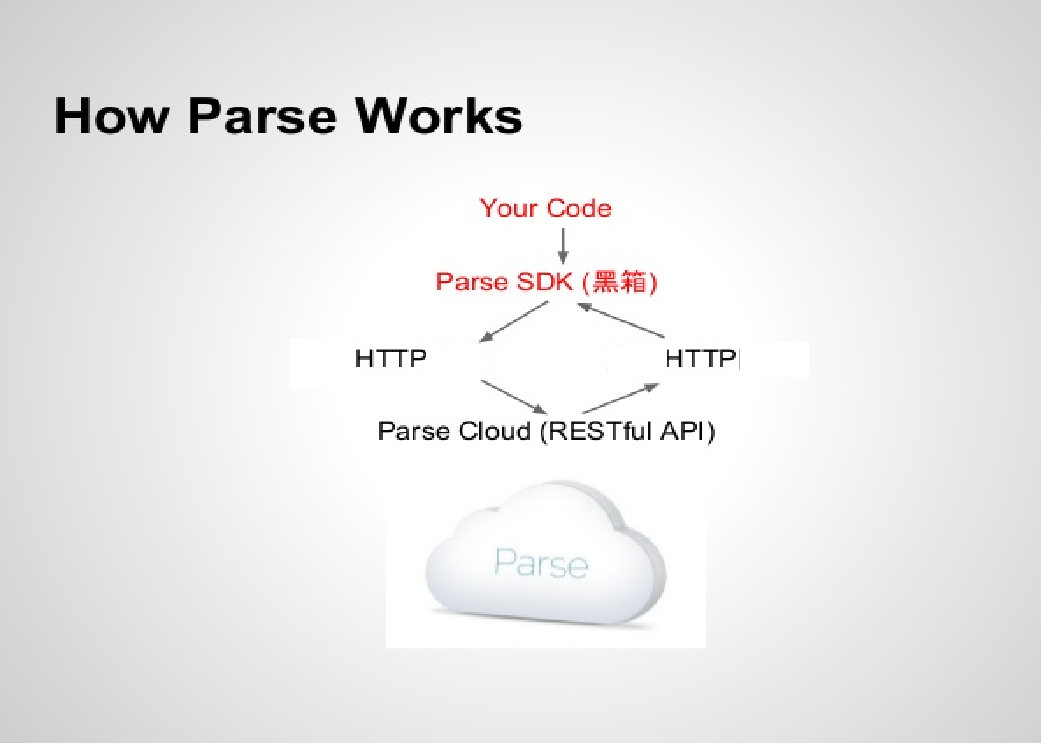


Figure 16: Parse

* Google app engine and Google endpoints: Google app engine is a very famous platform as a service model that allows developers to deploy and run their web app with minimal cost and high performance. This also provides a huge set of API to easily include more features and enhance performance, make integration with other cloud technologies easy and affordable.
* It has a good set of tutorials for beginners and demo code to integrate your code with app engine. In eclipse there is an inbuilt plugin available so that you can test your code every time from eclipse and run it as many times as you want. Google app engine supports and enhances many applications. These applications can be written in java, python, go or PHP.
* Some of the key features of Google app engine includes:
  + The data stored in the app engine is persistent, consistent and reliable. it can be fetched using queries, REST API and transactions.
  + It provides asynchronous task handling so that some task, which are equally important, can be performed outside normal flow. Request session continues to perform pre-defined task.
  + Special triggering events, which helps to instantiate various actions on a periodic basis to enhance and fulfill your app’s purpose.
  + As it is automatic scaled you don’t have to worry about the load. Also on crucial circumstances it does load balancing too.
  + It basically is a runtime environment for easy app making. It reduces cost to a very low level. There is no need to worry about extra space required, as it is automatic.
  + They provide detailed billing details and flexibility to end your tenure at any point.

Google app engine has its own software development kit (SDK) which helps make better use of the features it provides and connect to the app engine from outside world. We need to sign up for an account where we have billing details, a simulated coding environment, all the libraries and all the deployment tools. Based on the account you create you get a limited quota of memory available and design your app with that reference. Every bit of extra memory used is chargeable.

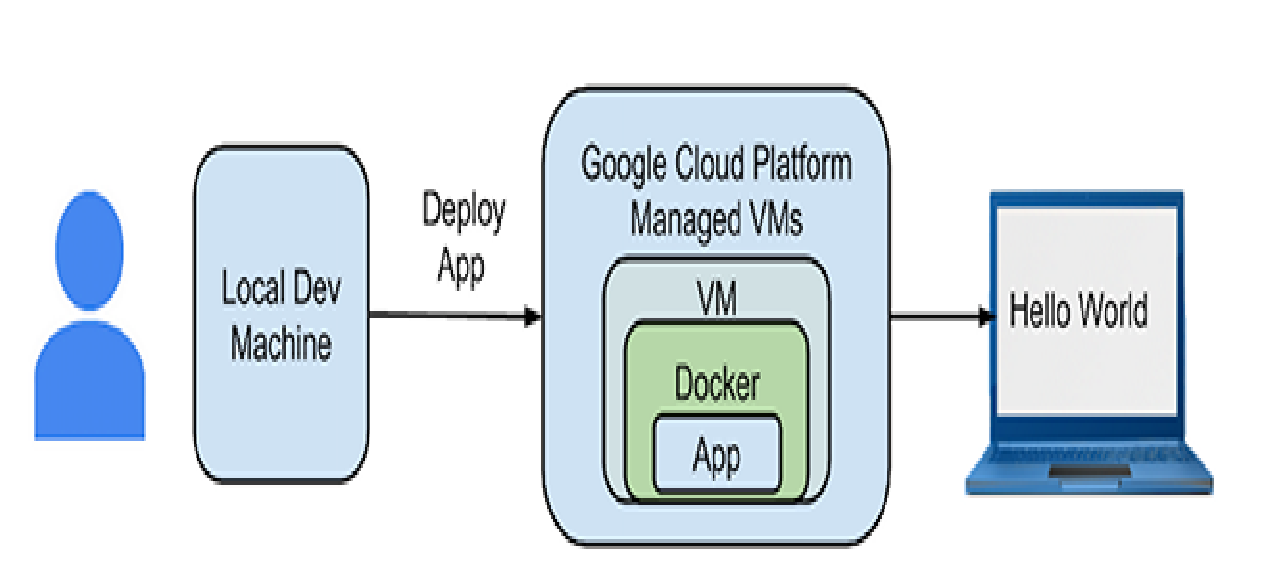
****

Figure 17: Google SDK

# ReSTFul Interfaces

As we have developed Mobile sensor cloud using crowdsourcing technique, we have database Parse to store the sensor data. We are uploading various sensor data in the Parse DB tables. But accessing database and making queries simultaneously from different users will be overhead to the database implementation. So, we developed ReST Apis for accessing this data.

ReST Api has following advantages, which overcome the database overhead.

* Use HTTP methods explicitly.
* Be stateless.
* Expose directory structure-like URIs.
* Transfer XML, JavaScript Object Notation (JSON), or both.

By using these functionalities we can use these ReST APIs from any platform, where we are going to use this sensor data like web application, iOS applications, etc…

Interfaces for Internet of things

We developed Sensor-Data interfaces for accessing data from database. These interfaces need gateways. IoT implementation is easy using ReST APIs. HTTP interfaces are used widely for ReST APIs.

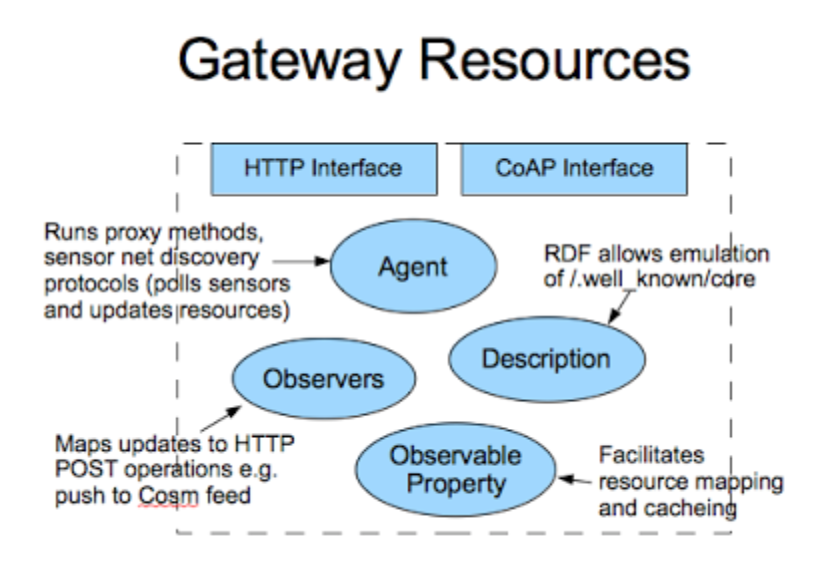


Figure 18: Gateway for ReST services

SensorData API

* /sensordata/type/{type} - Get the sensor data by Sensor type
* /sensordata/date/{date} - Get the sensor data by date
* /sensordata/location/{location} - Get the sensor data by location and radius
* /sensordata/daterange/{startdate}/{enddate} - Get the sensor data using date range

# Design Implementation

Our application falls in server client application part. On server end we have database, actual application implementation and cloud functionalities like scalability, availability, etc.

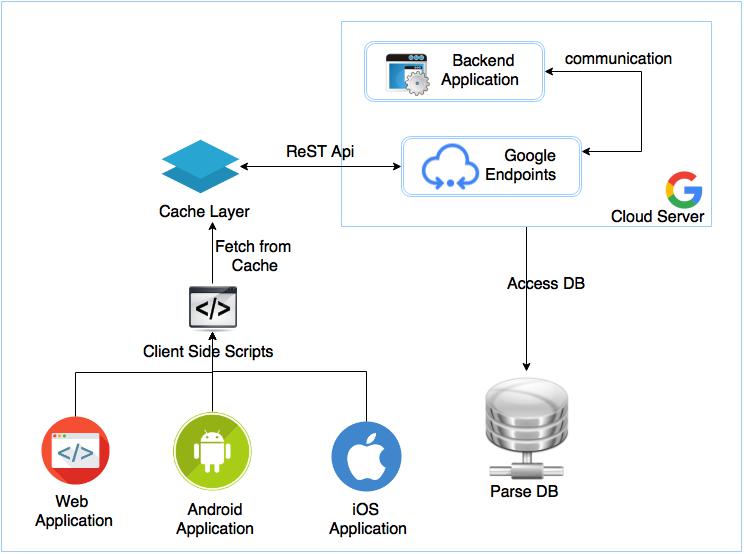


Figure 19: Design Implementation diagram

## Server Side Design

Database

We have implemented database in Parse creating a Sensor Data table with sensor type and value is the sensor at particular time of instance.

Backend

We have developed our application at Google cloud using their services of backend implementation, which can be accessed, by any of the User services.

Google endpoints

Using Google endpoints we have defined the ReST Apis for the application and Using these services we can call them from code as well as they provide ReST calls to the functions.

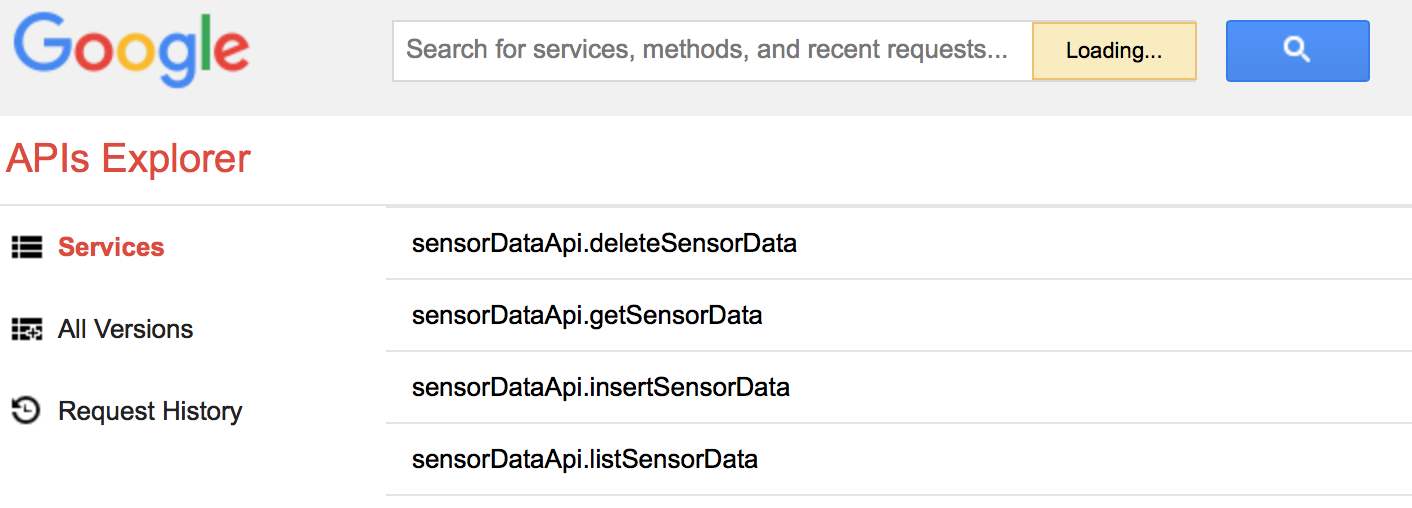


Figure 20: ReST APIs

## Client Side Design

Cache Layer

This layer serve as a cache storage for processed data or already accessed from Cloud Server. So, the accessing client cache will be faster in terms performance and time.

Client side scripts

These scripts can be used to call those methods, which have been developed using Google backend Endpoints APIs.

User Interface

We have currently developed a UI for Android Users. In future using same functionality we are going to develop iOS Application and Web Applications.

# Testing (UI or Stress test)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Feature Name** | **Test Case ID** | **Summary** | **Precondition** | **Execution Steps** | **Expected Result** | **Actual Result** |
| Install | 1 | Verify that application should be Installed successfully on the mobile device. |  | 1. Open the mobile app in Android Studio. 2. Run the mobile app from Android Studio. | The mobile application should be Installed successfully on the mobile device. | Pass |
| Application Start/ Restart | 2 | Verify that the mobile application launches within the stipulated time. | User should have installed the mobile app. | 1. Find the application icon and select it.  2. “Press a button” on the device to launch the app.  3.Observe the application launch In the timeline defined | Application must not take more than 25s to start. | Pass |
| Interruption by Calls | 3 | Verify that user should able to accept Phone calls when application is running and should continue from the same point. | User should have logged in to the mobile app. | 1. Open the application. 2. Navigate here and there for a moment. 3. Make a call from another device to the device where you have opened the application. 4. Pick up the call. 5. Now disconnect it and verify. | User should able to accept Phone calls when application is running and should continue from the same point. | Pass |
| Interruption by Messages | 4 | Verify that user should able to accept messages when application is running and should continue from the same point after reading the message. | User should have logged in to the mobile app. | 1. Open the application. 2. Navigate here a there for a moment. 3. Send a message from another device to the device where you have opened the application. 4. Read the message. 5. Close the message app and verify. | User should able to accept messages when application is running and should continue from the same point after reading the message. | Pass |
| Login | 5 | Verify that an existing user logins in to the application when entered correct username and password. | User should have installed the mobile app. | 1. Find the mobile application in your apps area.  2. Click the mobile application.  3.Click user sign in button.  4 Enter correct username and password.  5. System launches next screen. | If you are an existing user and entered correct username and password, system moves to next screen. | Pass |
| Incorrect Login | 6 | Verify that an appropriate error message is thrown, when user enters wrong username and password. | User should have installed the mobile app. | 1. Find the mobile application in your apps area.  2. Click the mobile application. 3.Click user sign in button.  4 Enter wrong username and password.  5. System throws an error message. | If you are an existing user and entered wrong username and password, system throws error message. | Pass |
| Continual Keypad Entry | 7 | Verify that continual key pad entry do not cause any problem. | User should have logged in to the mobile app. | 1. Logged in to the application. 2. Pressing any key or another key randomly.  3. System should not cause any problem. | Continual key pad entry do not cause any problem in the application. | Pass |
| Clear key | 8 | Verify that clear key should navigate the user to previous screen. | User should have logged in to the mobile app. | 1. Logged in to the application. 2. Pressing clear key.  3. System should return to previous screen. | Clear key works fine by navigating back to the previous screen. | Pass |
| End Key | 9 | Verify that End Key should navigate the user to native Original Equipment Manufacture's screen. | User should have logged in to the mobile app. | 1. Logged in to the application. 2. Pressing end key.  3. System should return to native Original Equipment Manufacture's screen. | End key works fine by navigating back to the Original Equipment Manufacture's screen. | Pass |
| Scrolling page | 10 | Verify that the page scrolling scenarios are being enabled in the application as necessary | User should have logged in to the mobile app. | 1. Logged in to the application. 2. Scroll down/up the screen.  3. System allows the scrolling. | Page scrolling works fine when user tries to scroll up or down. | Pass |
| Exit application | 11 | Verify that user should able to exit from application if we click on end key. | User should have logged in to the mobile app. | 1. Click on app and open it. 2. Now press the end key and verify. | User should able to exit from application if we click on end key. | Pass |
| Uninstall | 12 | Verify that application should be Uninstalled successfully from the mobile device. | User should have installed the mobile app. | 1. Click on settings. 2. Select the newly added application on test case id 1. 3. Click on Uninstall button. 4. Verify. | The application should be Uninstalled successfully from the mobile device. | Pass |

# Automation Testing - Selenium

We have used Selendroid for testing our mobile application.

Selendroid – Selenium testing for Android based applications. It is a test automation framework that drives off the User Interface of Android native and hybrid apps and the mobile web. Test scripts are written using the Selenium 2 client API.

Selendroid was used on real mobile devices for scaling and parallel testing. We choose Selendroid for testing our mobile app was for the below features which was provided by it.

* Compatibility with JSON wire protocol and selenium 3 ready
* Modifications are not required under test in order to automate it
* Android driver web view app test the mobile web using built in app
* Automating native or hybrid apps follow same concept
* UI Elements Can be found by different locator types
* Advanced user interactions and gestures are supported
* Selendroid can interact with many android devices
* Emulators start automatically
* Hot Plugging of hardware devices are supported by Selendroid
* Full Integration as a node into Selenium Grid for Scaling and parallel testing
* Multiple android target API support
* Test case development inspector is present
* Selendroid can be extended at runtime with your own extensions

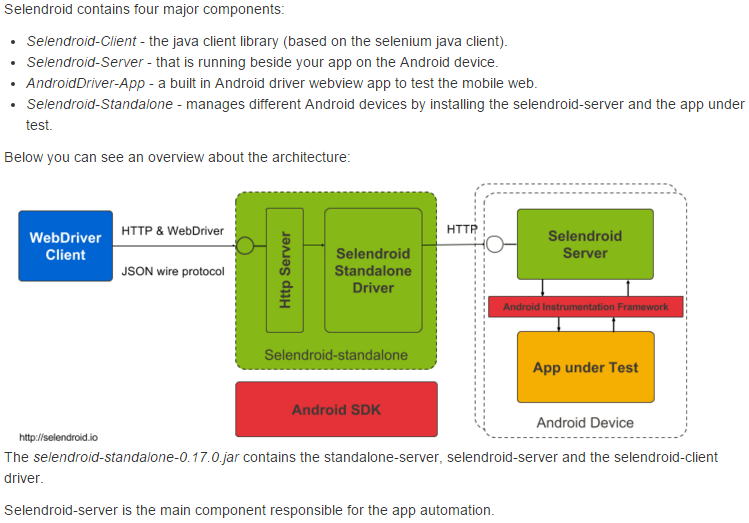


Figure 21: Selendroid – selenium testing

## Pre requisites

1. Selendroid requires the following permission for your app:

<uses-permission android:name="android.\*\*permission.INTERNET"/>

2. Java SDK (minimum 1.6) must be installed and JAVA\_HOME configured.

3. Latest Android-sdk must be installed and ANDROID\_HOME set.

4. Mobile device with its USB driver connected.

5. Mobile application APK(AUT)

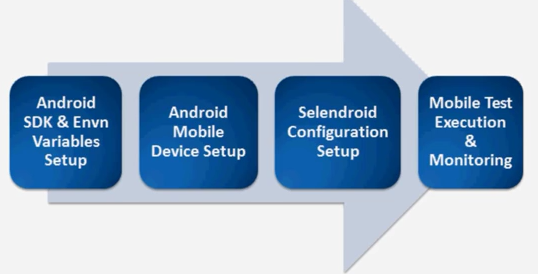


Figure 22: Selendroid steps

## Steps for installing Selendroid

* Download selendroid-standalone.jar and cd to the folder where you downloaded the jar, and run:

java -jar selendroid-standalone-0.17.0-with-dependencies.jar -app selendroid-test-app-0.17.0.apk

* Selendroid-standalone will start a http server on port 4444 and will scan all Android virtual devices (avd) that the user has created (~/.android/avd/).

You can check that the application(s) and the devices are recognized by opening a browser and navigating to:<http://localhost:4444/wd/hub/status>.

* Test the basic test cases of the mobile application.

# Design Patterns

A pattern can be defined as a reusable solution which can be applied to frequently occurring issues or problems in web or mobile application - in our project – we are Android Studio to develop a mobile application using Java programming language. Benefits of using Patterns are:

* They are proven solutions
* They can be easily reused
* They can be expressive

## Front end design pattern

The mobile front-end application was developed using Android Studio tool, where we used

Java programming language to develop the source files, XML to define the layout files and used MVC (Model-View-Controller) or MV\* (Model-View \*) design pattern.

MVC is an architectural design pattern, which encourages on a separation of concerns. It emphasis on the separation of business data (Models), which in our project are Sensor’s type, Sensor data and respective location data from user interfaces (Views), with a third component (Controllers) like servlet - traditionally managing logic and user-input from the mobile application.

## Middle tier design pattern

Name: Independent pattern

Context: Many middle tier architectures are tightly coupled thereby reducing flexibility. This makes the application less robust due to technology dependencies.

Solution: This application has a technology agnostic middle tier. The middle tier in our case is Google Cloud End points, which interact with the handler classes and data access layer, to and froth, to display the data in the mobile application.

The mobile application follows a clear segregation of handler classes and the data access layer. This segregation ensures ease of maintenance, development and debugging. The middle tier acts as a broker between the presentation tier and the data layer uses REST API for communication.

Reusability: Reusability is key to any middle tier architecture this application uses methods that are reusable and has clear segregation from the other methods for ease of use.

Platform agnostic: The middle tier is deployed on the Google app engine Java end points module currently making it platform agnostic.

## Cloud Design pattern

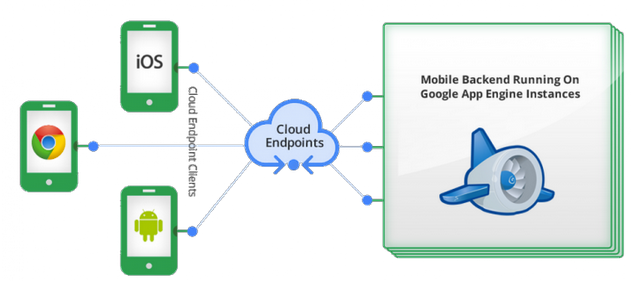


Figure 23: Google Endpoints

### Relational Databases

Our mobile application makes use of Relational Database management system in order to store data. Below are some of the design patterns followed in this project.

Name: Consistency design pattern

Context: In any RDBMS table, there should be well-defined tables in order to perform data manipulation efficiently. In addition, ACID properties are very essential in order to maintain the accuracy and the efficiency of the data stored. The data stored in the tables have dependencies on each other. If data at one place is modified or deleted, the data associated with this should also be modified or deleted accordingly. Else inconsistencies will be introduced will lead to the erroneous results even security threats.

In our application we used Sensors data table. The table has many columns like Sensor Type, Location, Sensor value, Date and Time.

Solution: In a relational database, data elements are stored in tables where each column represents an attribute of a data element with a well defined semantic. These attributes may be used in data queries to make them more expressive. As we have only one table that has all the necessary fields defined, we do not have any foreign key concept for doing any data manipulation in other tables. Hence we have a highly secure, consistent and independent table in a relational database management system.

### Distributed component design pattern

In a distributed application development, all the components are loosely coupled which enhance the reusability of modules through the distribution of object oriented systems.

In our project we are using more than one cloud technologies, which act independently, and have separate functions. Those are Parse storage cloud service and Google Cloud app engine End points that are integrated via java mobile application in Android Studio to form a distributed application. Also the interface between the data layer (backend) and the cloud end points (middle layer) uses RESTful services to access Sensor table. It retrieves the data as well as updates the sensor values in the sensor table on real time basis - hence indicating less dependency among the applications resulting in high elasticity and flexibility.

### Loosely coupled design

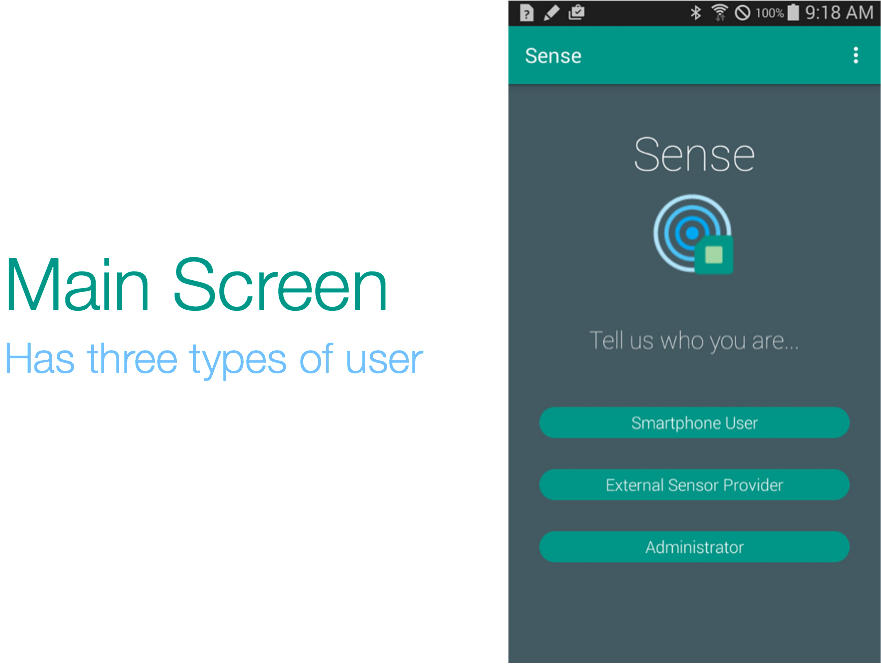
Context: Applications with one or more components that are tightly coupled are dependent on each other. This makes the application less flexible in terms of technologies used, execution environments, scalability, elasticity etc.

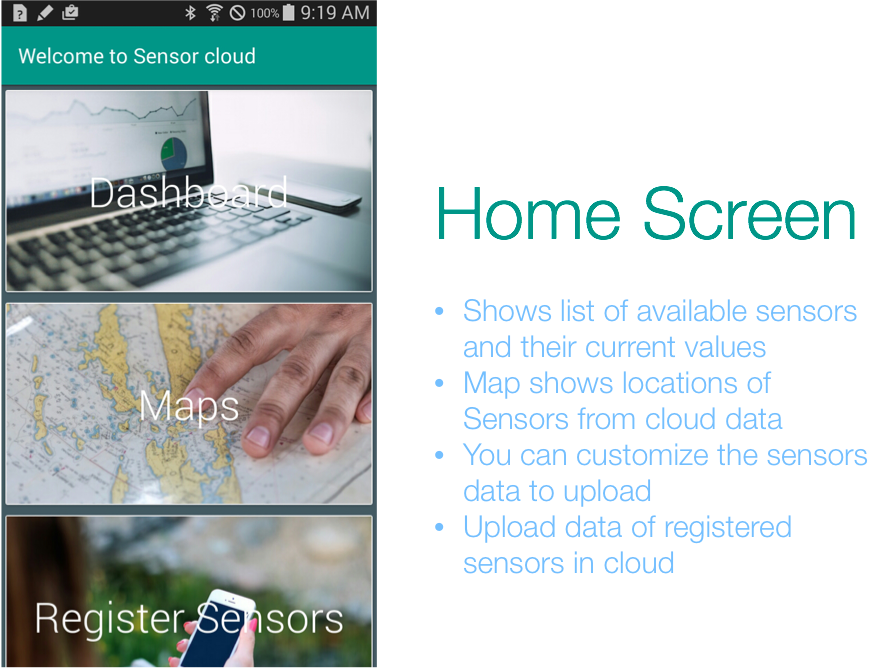
Solution: Loose coupling between the components greatly increases the flexibility and the robustness of the application. Making the components autonomous of the platform, reference, technology etc. will greatly improve flexibility and extensibility.

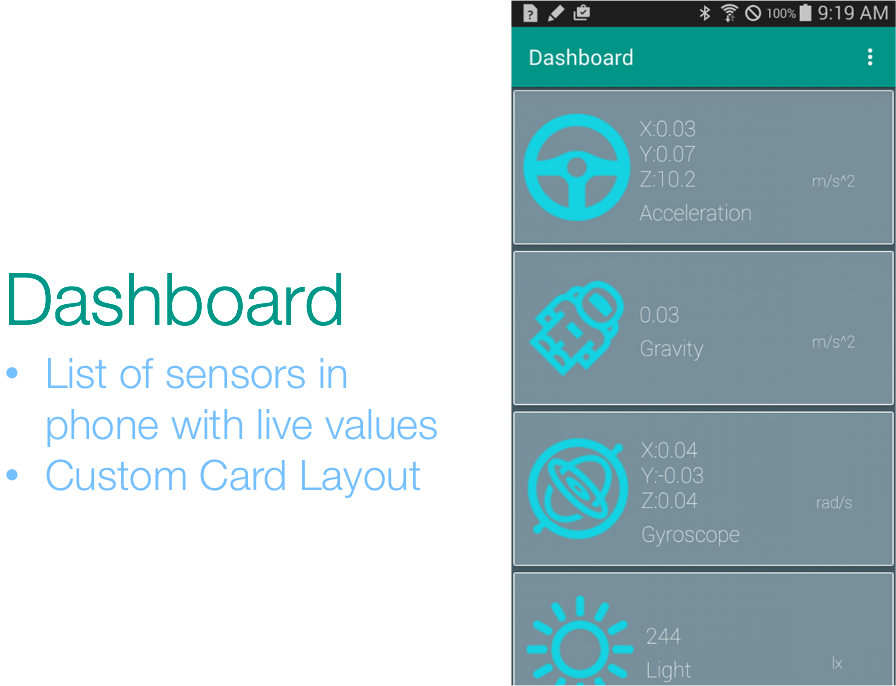
Result: We have implemented this design patterns in our project. We are designing the mobile front-end application using the Android Studio. We are using Java programming language for writing the source code and using xml for layout files. We have parse cloud database as a backend service, which stores the sensor value for the corresponding sensor type from the inbuilt mobile sensor phone and updates accordingly in the database.

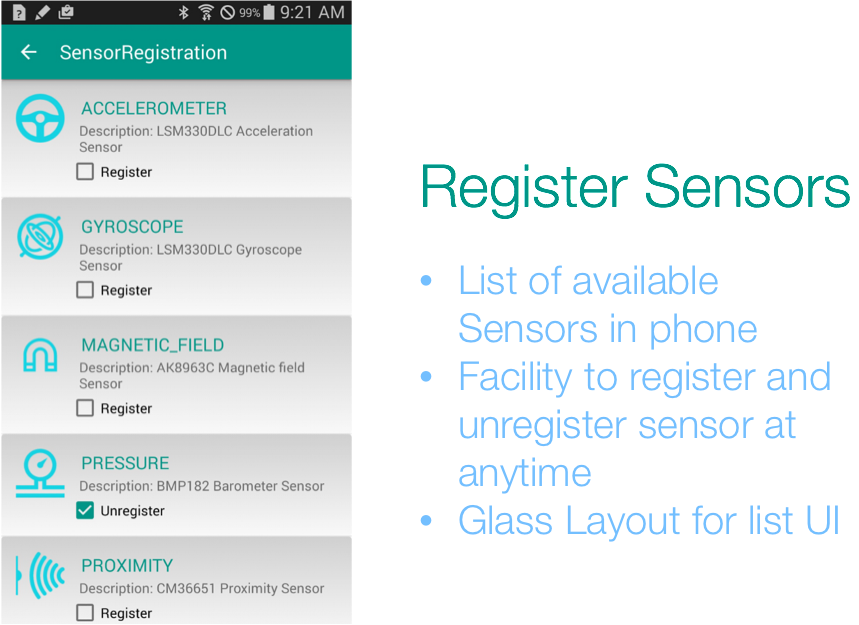
The mobile app displays these sensor values in the app using RESTful services, which helps to interact with the database to retrieve the real time sensor values.

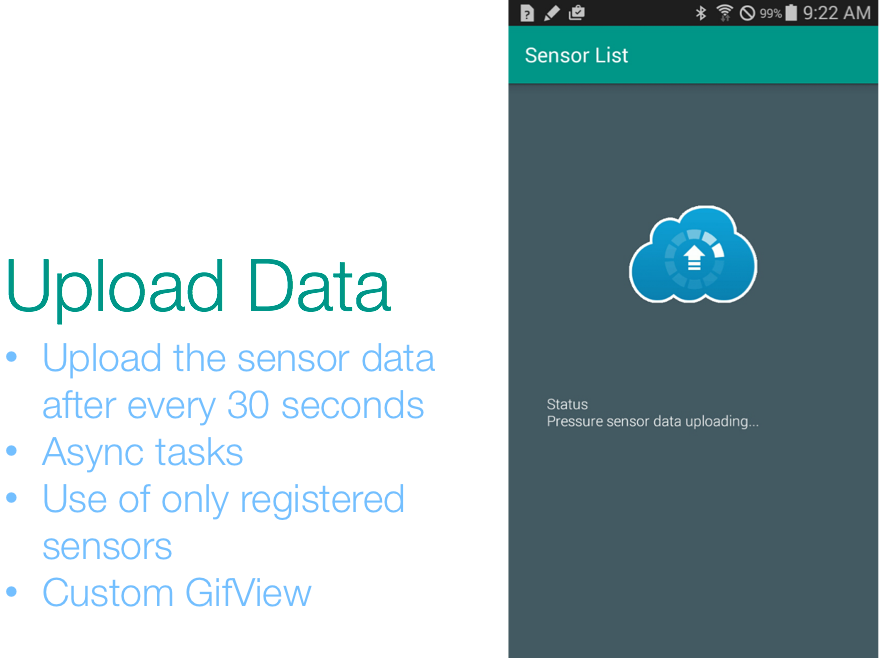
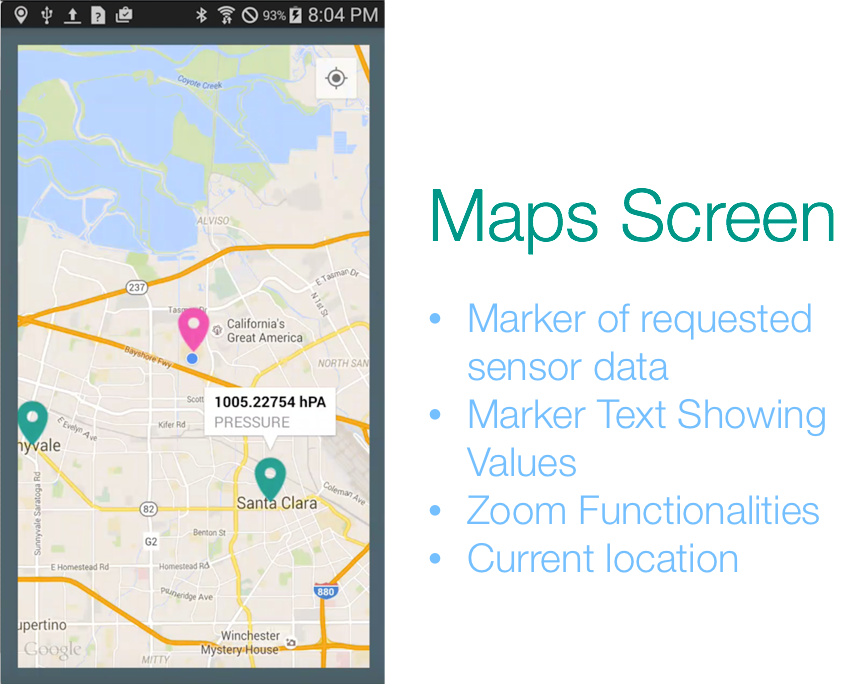
# User Interface – Screens











# Future Scope

Currently we are taking sensor data, allowing user to register/unregister sensor on the fly and uploading selected sensor data on cloud. Future scope involves using this sensor data for various beneficial applications as mentioned below:

* We can use location details collected from huge crowd to form crowd patterns. In case of emergency vehicle passing through such massive spots, we can alert these people by sending push notification to vacant the crowd and provide fast passage to the emergency vehicle.
* Light and temperature are two factors that get affected when air is polluted and level of some harmful gases is beyond threshold. A user registered with this app can monitor health of surrounding environmental conditions, can be aware of certain locations which contains gases he/she is allergic to , can find the best path for jogging selecting most pollution free area etc.
* Attracting customers by commercializing sensor data. Different sensor providers, manufacturers and companies can come forward to upload and share their sensor data for making our environment more healthy and giving a helping hand to community.

# Profiling

|  |  |  |
| --- | --- | --- |
|  | **Feature** | Contribution |
| 1 | Understanding the Project Scope | Priya, Saraswathi,  Shraddha, Isha |
| 2 | Technical feasibility Study | Priya, Saraswathi,  Shraddha, Isha |
| 3 | Web Design Implementation | Shraddha, Saraswathi |
| 4 | Sensor Code Implementation | Shraddha, Saraswathi |
| 5 | Cloud integration | Shraddha, Saraswathi, Priya |
| 6 | Research paper | Isha, Shraddha, Priya, Saraswathi |
| 7 | Unit testing | Priya, Saraswathi, Shraddha, Isha |
| 8 | Presentation slides | Isha, Priya |
| 9 | Backend Integration | Shraddha, Saraswathi |
| 10 | Cloud Database Creation | Shraddha |
| 11 | Integration Testing | Saraswathi, Shraddha, Priya, Isha |
| 12 | Sensor Research | Saraswathi, Isha, Priya , Shraddha |

# References

[1]. Buddhadeb Halder: Measuring Security, Privacy and Data Protection in Crowdsourcing,

<https://newnocturnal.files.wordpress.com/2014/07/measuring-security-privacy-and-data-protection-in-crowdsourcing.pdf>

[2]. Stevens M. & D’Hondt E.: Crowdsourcing of Pollution Data using Smartphones http://soft.vub.ac.be/Publications/2010/vub-tr-soft-10-15.pdf

[3]. Boulos M.N.K, Resch B, Crowley D.N., Bres-lin J.G, Sohn G., Burtner R, Pike W.A., Jezierski E., and Kuo-Yu Slayer Chuang: Crowdsourcing, citizen sensing and sensor web technologies http://www.biomedcentral.com/content/pdf/147 6-072X-10-67.pdf

[4]. Sougata Sen, Archan Misra, Rajesh Balan: The Case for Cloud-Enabled Mobile Sensing Ser-vices [ACM 978-1-4503-1519-7/12/08]

http://conferences.sigcomm.org/sigcomm/2012/ paper/mcc/p53.pdf

[5]. E. T.-H. Chu, Y.-L. Chen, J. W. S. Liu and J. K. Zao: Strategies for Crowdsourcing for Disaster Situation Information

http://eos.yuntech.edu.tw/eoslab/images/Crowd sourcing%20Strategies.pdf

[6]. Boulos M.N.K.: Creating Interactive maps using Google APIs - International Journal of Health Geographics [doi:10.1186/1476-072X-4-22]

http://www.ij-healthgeographics.com/content/4/1/22

[7]. Goodchild M: Citizens as sensors: the world of volunteered geography. Geo-Journal [2007, 69(4): 211-221]

http://link.springer.com/article/10.1007%2Fs10 708-007-9111-y

[8]. Kanhere SS: Participatory Sensing-Crowdsourcing Data from Mobile Smartphones in Urban Spaces. [6-9 June 2011; Luleå, Sweden 2011-3-6] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?a rnumber=6068482

[9]. Heimbuch J.: Sensordrone Turns Your Phone Into an Environmental Sensor [June 13, 2012] http://www.treehugger.com/gadgets/sensordron e-turns-phone-envirnomental-sensor.html

[10]. Saenz A: Japan’s Nuclear Woes Give Rise to Crowd-Sourced Radiation Maps In Asia and US [24 March 2011] http://singularityhub.com/2011/03/24/japans-nuclear-woes-give-rise-to-crowd-sourced-radiation-maps-in-asia-and-us/

[11]. Bhattacharya D.: Analytics on big fast data using real time stream data processing architec-ture [2013 EMC] http://www.happiestminds.com/sites/all/themes /happiestminds/pdf/ANALYTICS\_ON\_BIG\_FAST\_D ATA\_USING\_A\_REALTIME\_STREAM\_DATA\_PROCE SSING\_ARCHITECTURE.pdf

[12]. PollMap - Lunch Break Edition. http://nutrition.esri.com/lunchbreak/

[13]. GeoChat by InSTEDD (Innovative Support to Emergencies, Diseases and Disasters). http://instedd.org/technologies/geochat/

[14]. The Ushahidi Platform. http://ushahidi.com/products/ushahidi-platform

[15]. Crowdmap: <http://crowdmap.com/>

[16]. Sheng X, Xiao X, Tang J, Xue G: Sensing as a Service [978-1-4577-1767-3/12/$26.00 ©2012 IEEE] http://optimization.asu.edu/papers/XUE/XUE-CNF-2012-SENSORS-SaaS.pdf

# Presentation

For Code

GitHub Link <https://github.com/shraddhakannav/CMPE277>

For Presentation

<https://drive.google.com/file/d/0B0MFONbIKatFZlc1elQ2THVoZDA/view?usp=sharing>

For Video

<https://drive.google.com/file/d/0B2C8a0yPor-bRHV4VEt1ZkhDYjA/view?usp=sharing>