**IIIT\_H/ Spring 2015**

**Internals of Applications Servers**

**Project Module2 : Filter Server, App Engine and Client App**

**1. Module Introduction**

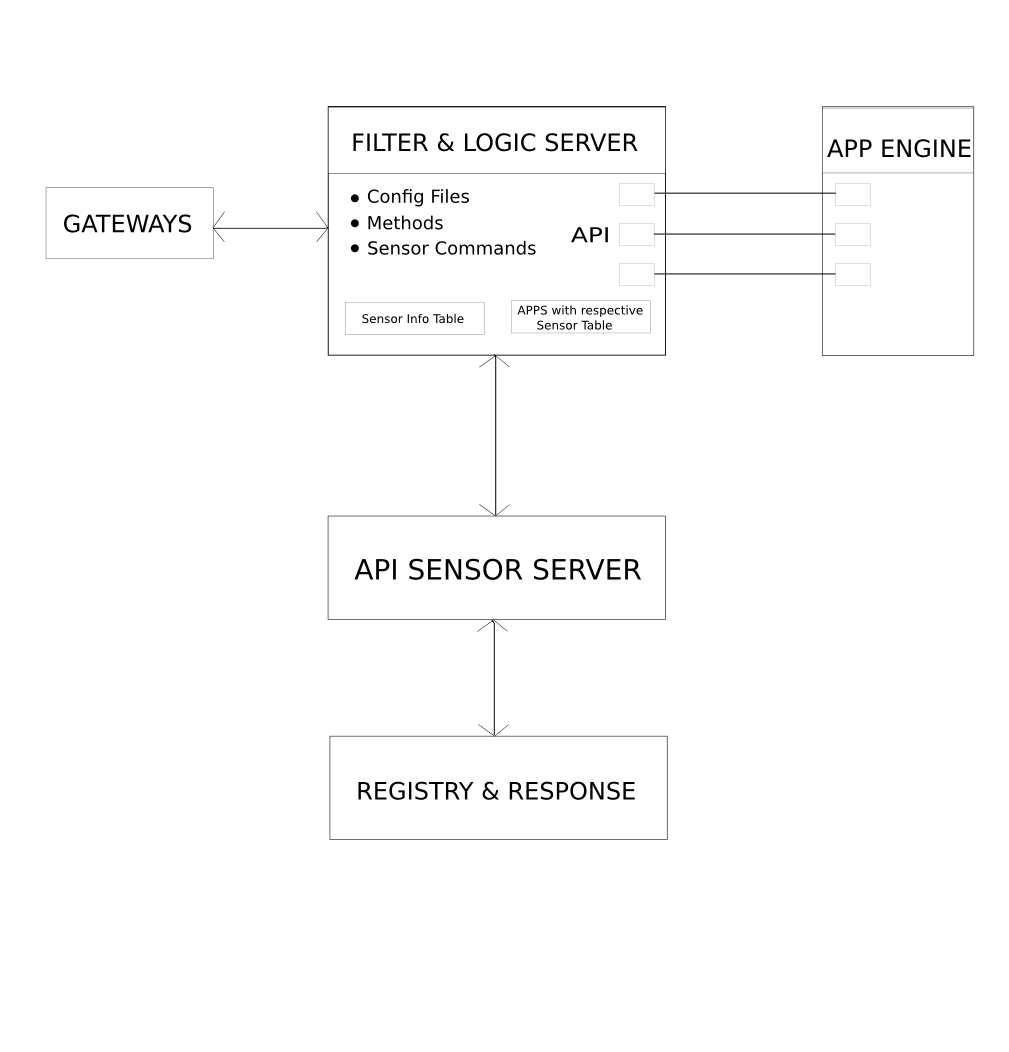
This module comprises Filter server, App engine and Client app which described below :

Filter Server communicates with Gateways, App Engine and sensor server. Gateways delivers sensor data and analyse it. If the sensor data is useful for any apps then it send to the App engine on specific

port with the help of written API. Registered sensor and apps can only communicate with filter server. It can also communicate with sensor with the help of sensor command provided by sensor device.

App Engine is the place where apps are deployed. Deployed app can configure XML file on Filter server. Apps on App engine will communicate with client app.

Client App receives information from App server( i.e. apps on App engine).



**2. Test cases (will clarify what your module will do. Build on your operations and use cases listed in requirements doc)**

**2. Test Cases**

2.1Test cases- used to test the team’s module

Use case : Critical Temperature Alert System

1. Test receive data from gateways
2. Test Validate data
3. Test Update databases with data
4. Test Send data to Logic Server
5. Test communication between filter server and app engine
6. Test data received by users application
7. Test registering an app with platfrom
8. Test UI provided for sample application
9. Test filter logic with various scenarios

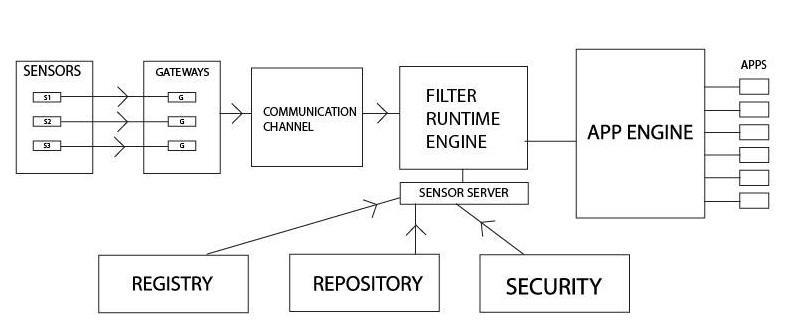
2.2Overall project test cases (relevant to the module)

Integration test cases:

1. Check if API calls are working properly at filter server and app engine
2. Check integration of app engine and applications
3. Test integration between various database components and servers
4. Test security of the application. ( Authorised calls should be able to access the APIs)

**3. Solution design considerations**

3.1 Design big picture



3.2 Environment to be used

* Apache Server
* Node.js
* Android

3.3 Technologies to be used

* Apache server

It is used for initiating server.

* Node.js

The whole logic of filter server will be written in Node.js because Node.js is good for

writing server side script and it manages 2-way communication very well.

* Android

This is used for developing client side applications.

3.4 Device info proxy server

Registry is a database which is used to store the sensor information such as sensor id, sensor type and sensor installation timestamp. Registry will be helpful and capable of differentiating between the various sensors and their data. It can be used as device info proxy server.

3.5 Communication overview

Communication will be done using Sockets , RMI or REST API depending upon appropriate resource availability.

3.6 Filter Server

All apps register with the filter server stating their requirements. When the filter server gets data from the sensors it sends only the relevant data, that is, only the data required by the app.

3.7 Logic server

Filter server will also work as logic server.

3.8 Interactions between modules

* + Gateway to Filter server : Gateways aggregate, prune and bundle the data, then encapsulate in IP for delivery to the filter server as a small data “stream” . This data packet is sent to the filter server in real time over the TCP/ UDP communication channel. Filter server listens on the specific port for incoming packet.
  + Filter server to App engine : Filter server provide interfaces to access different kind of sensory data. App engine interacts with Filter server by calling the registered service method through RPC at regular intervals or whenever needed

3.9 Wire and file formats

* + *Gateways to Filter server:*

<IP, port,<type, sensor id, location, sensed data, timestamp, checksum> >

3.10 User’s view

There will be an Android app which will provide following features:

List all services

Sensor Data

Thresholds

Locations Selection

Alerts

**5. User’s view of system**

* Config File maintained on Filter server which keeps analytics rule. It notifies to the application if it crosses particular threshold.
* Analytical rule are defined with the help of persistence database that hold sensor type and corresponding value that represent threshold value.

User interactions

User will use RPC procedures to add entries in config file.

**6. Key Data structures**

Tables are maintained on the Filter Server which is as follows :

1. Sensor info ( Active or down etc)
2. Apps and corresponding registered sensors

Wire formats

Sensor sends heart bit in particular time span. this packet contain sensor id (IMEI number) and timestamp.

<sensor\_id, timestamp>

Persisted data

Tables maintained are as follows :

1. sensor info

<sensor id, last\_uptime, last\_downtime, up\_since>

1. App and corresponding registered sensors

<App\_id, sensor\_id>

**7. Interactions & Interfaces**

1. Communication between Filter server and App engine

Filter Server API

1. sendData(sensor\_id , location , sense\_value )

It sends filtered data to app.

2. register\_sensor(App\_id, sensor\_id)

It is used for registering sensor on app listed sensor.

3. unregister\_sensor(App\_id, sensor\_id)

It is used for deleting sensor on from app listed sensor.

App Engine API

1. recieveData(sensor\_id , location , sense\_value )

App receives filtered data provided by Filter server.

2. register\_sensor(App\_id, sensor\_id)

It is used for registering sensor for app on Filter server.

3. unregister\_sensor(App\_id, sensor\_id)

It is used for registering sensor for app on Filter server.

2. Communication between Registry Repository module and Filter server

Filter Server API

1. is\_valid\_sensor(sensor\_id)

It checks sensor who provides data is valid sensor or not.

***Group 4:***

***1. Gangasagar Patil***

***2. Poorva Bhawsar***

***3. Sushant Makode***

***4. Kedar Biradar***

***5. Omprakash Shewale***

***6. Nikhil Barote***

***7. Harshad Jalan***

***8. Kapil Chhajer***

***9. Diwas Joshi***