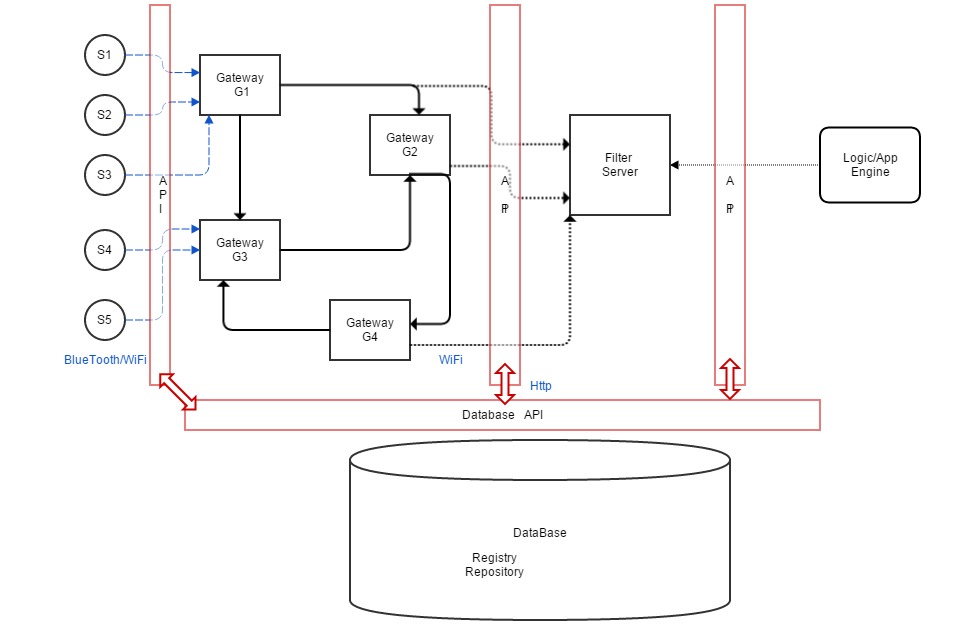
**PROJECT PHASE 3 DESCRIPTION**

**Introduction to the project :** The whole setup is as shown in diagram 1.1 .



**Part1 :** There is a device gateway - sensor network where different types of sensors collect data from their surroundings and repeatedly send this collected information to the gateway it is connected to. This communication between the sensor devices and the Gateway server takes place through an API.

**Network Topology** **:** There is a network topology where each Gateway server is connected to some of the sensor devices and other gateway servers.

**\*NOTE\*:**  The REPOSITORY is central-static whereas the REGISTRY is central-dynamic.

* The Gateway servers have a unique network id and a static topology between the Gateway servers (information stored in central Repository) is considered whereas the sensors attached to each gateway are dynamic.
* Each gateway server can be connected to one or more types of sensors , if a new Gateway server or a new sensor device is registered with a gateway this information has to be updated in a central repository at the database.
* The sensor devices have to send heartbeat messages to the Gateway so as to keep its state updated.
* Once the sensor is declared inactive (after timeout at Gateway) only that particular communication link between sensor device and Gateway gets down But if The Gateway is down then all the sensors connected to the Gateway are down along with the Gateway.
* Some of the Gateway servers have direct links to communicate with the Filter Server

but if the communication link gets down due to some fault at the gateway, rerouting has

to take place.

**Setup :**  An API is provided for the Gateway servers , Filter server and Logic server to communicate with the database (Mongo DB) .

At the time of boot-up all the Gateway servers communicate with the Database to fetch the static information using the repository.

A security check is performed at the database if the Gateway server requesting is an authenticated server.

The information provided by the repository is Table 1.2 :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Gateway Id | Sensor Id | Sensor Type | Sensor Status | Protocol | Type-Handler |

Once the Gateway has this information it can load the API(available at repository) at its end to communicate with the sensors attached.

**Sensor-Gateway communication :** Apart from heartbeat status messages , the sensors have to send the data being collected to the Gateway server to which it is attached in the form of a key-values pairs as:

|  |  |
| --- | --- |
| Sensor ID | sensed Data |

The gateway server has to then convert this data into a tuple as follows and send this information (Table 1.3 ) to the central registry which gets updated from time to time.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sensor Id | Sensor Type | Sensor Data | Sensor Location <x,y> | Time stamp | CheckSum | Sensor Status |

Gps Information

|  |  |
| --- | --- |
| Longitude | Latitude |

**Gateway-Filter Server communication :** Certain gateways communicate with the filter server directly in the form of push-messages to the Filter-server .Push Messages are sent whenever there is some update with the data sensed corresponding to those servers which are registered

alongwith Logic/App server for providing information.

**Part2 :** The Database is implemented in mongoDB which is the Core of the project where all the important information is being stored and all the three major parts , the Gateway server , the Filter server and the Logic/App server communicate through a common API to fetch and gain control of the information for their central functioning and also for storing the updates made.

**Database:**

1. **Registry -** This part of the DB will consist of information which will be updated dynamically timely.

* Sensor Information (Table 1.3) - A new record will be inserted by the Gateway API, as soon as it receives data from sensor.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sensor Id | Sensor Type | Sensor Data | Sensor Location <x,y> | Time stamp | CheckSum | Sensor Status |

The sensor location consists of 2-tuple -

|  |  |
| --- | --- |
| Longitude | Latitude |

The App Engine first gets a list of sensor IDs according to the sensor type, and location (longitude/latitude) lying inside the range specified by it.

The filter Server registers the App engine, and keeps the lst of sensor IDs. Whenever some new data is inserted in this table, Filter Server retrieves it based on the query given by App Engine.

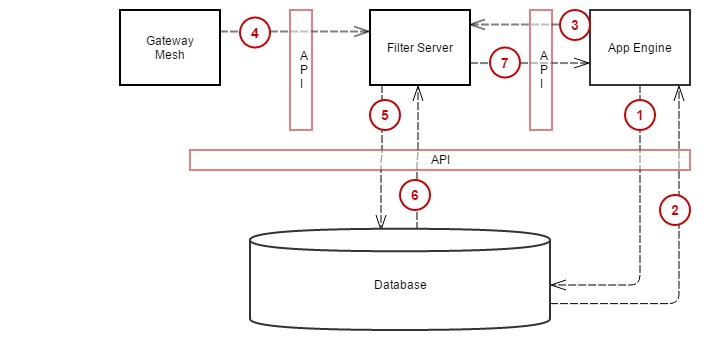
**2. Repository -**  At the time of boot-up all the Gateway servers communicate with the Database to fetch the static information using the repository. A security check is performed at the database if the Gateway server requesting is an authenticated server.

* Gateway Information (Table 1.2) :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Gateway Id | Sensor Id | Sensor Type | Sensor Status | Protocol | Type-Handler |

Once the Gateway has this information it can load the API(available at repository) at its end to communicate with the sensors attached.

**Part3 :**



For gaining access to processed and filtered information the Logic-server / App-engine has to register itself with the Filter-Server but for this purpose it first needs to gain control over the sensors that are specific to the type of information needed by the App-engine and that too within a certain range.

1. For this purpose the Logic-server/App-Engine creates a tuple -

|  |  |  |  |
| --- | --- | --- | --- |
| Type | X | Y | Range |

**Logic App - Database Communication :** The Logic App communicates through the API

and sends this tuple to the Database to retrieve the list of sensors that will give the best results as per the requirements mentioned above.

Here, X represents the Latitude and Y represents the Longitude of the GPS coordinate , which will act as the reference point for finding the sensors lying in range ‘Range’ wrt (X,Y) by calculating the euclidian distance as explained in below API :

**findSensors\_inRange(X,Y,Range) :**

for(each sensor location in DB: <latitude,longitude>)

{

dist=Math.sqrt(Math.pow((X-latitude),2)+Math.pow((Y-longitude),2));

if(dist<=Range){

return sensorID;

}

}

2) The API interface called in above step does the following :

* For storing the result an array of type Sensor Id is created as <List\_of\_sensors>.
* It checks the **Table 1.3** in central repository and corresponding to every Sensor Id matches the Sensor Type with the Type mentioned in the requirement.
* If the type matches then , calculate euclidean-distance of the location(GPS coordinates) with the reference coordinates (X,Y) given as input.
* If the result lies in the specified range then add Sensor Id to the array <List\_of\_sensors>.
* Once the whole table is read the array <List\_of\_sensors> is returned to the Logic server/ app-engine in step 2 .

3) The App engine , after getting the list of all sensor IDs within its range will prepare a structure as below , and send it through the API to filter Server to register itself.

**registerApp(AppID, <list\_of\_sensorIDs>):**

|  |  |
| --- | --- |
| App Engine ID | <List of Sensor IDs> |

The Filter Server will maintain this info in its database.

4) Certain gateways communicate with the filter server directly in the form of push-messages to the Filter-server .Push Messages are sent whenever there is some update with the data sensed corresponding to those servers which are registered alongwith Logic/App server for providing information.

**sensorUpdate(SensorID ):**

The Filter Server will see the list of the server IDs requested by App Engine, and will match the sensor IDs sent by gateways. if they match, the filter server will retrieve data from DB.

5) Filter Server will also maintain info about the queries performed by App engine.

**getQuery(AppID):**

|  |  |
| --- | --- |
| App Engine ID | SQL Select statement |

For any active App , it will get the select query, and the list of sensor IDs, and retrieve the data from DB for those sensor IDs, matching the criteria in Select statement.

**getDatafromDB(SQLPreparedStmt)**

6) The data corresponding to the SQL queries will be sent back to the Filter Server in form of a list <list of sensorData>.

7) The Filter Server will send back the data retrieved from DB to the App Engine.