Internet of Things Apps Platform

A User Guide

Submitted by

Group Number - 5

Team 1 :

Sourabh Dhanotia	201405605
Lokesh Walase	201405597
Pankaj Shipte	201405614

Team 2 :

Atul Rajmane	201405529
Prerna Chauhan	201405544
Aditi Jain	201405549
Abhinaba Sarkar	201405616

Team 3 :

Veerendra Bidare	201405571
Swapnil Pawar	201405513
Abhishek Mungoli	201405577

Under the guidance of

Mr. Ramesh Loganathan

For the course

Internals of Application Server

IIIT, Hyderabad

April 4, 2015

Contents

List of Figures

List of Listings

Chapter 1

Admin Panel Manual

1. Admin Logs in to our web interface present at `http://www.link-to-our-site.com` with the credentials provided.

2. Admin can add a new sensor, view the sensors connected to the system. To add a new sensor admin has to fill up a small web form wherein he/she enters the Gateway ID to which the sensors will be connected along with other information. This other information includes manufacturer, sensor family, the type of data that it generates, the interval at which it generates the very data, .jar file for the type handler.
   Browse facility allows admin to browse through all the sensors by gateways or by the active status.

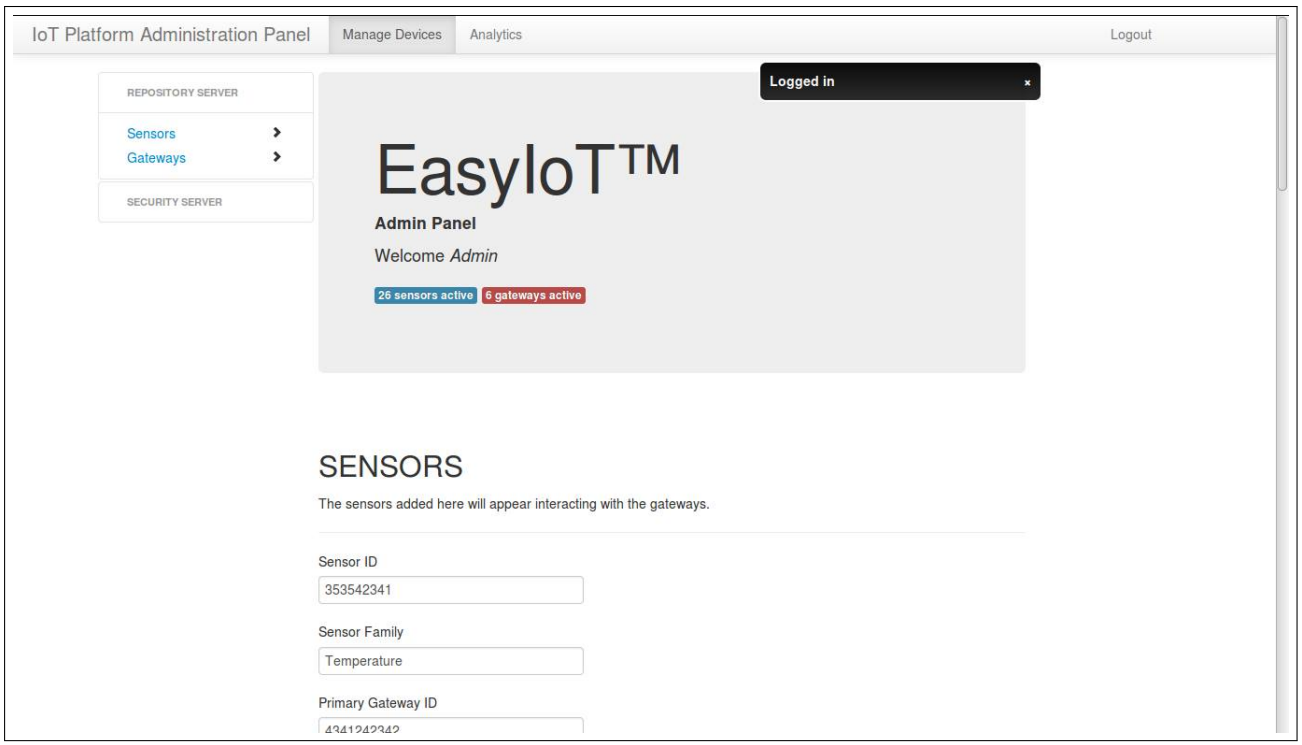

Figure 1.1: *Admin Panel Screenshot*

Chapter 2

RESTful APIs Manual

The primary way for apps to read data from the platform is using REST APIs. It's a low-level HTTP-based API that you can use to query data, and a variety of other tasks that an app-tier might need to do. The REST API will have multiple versions available, and the first version of which will have the following APIs exposed.

2.1 Query APIs

2.1.1 Repository APIs

1. **/repository/gateways**

   In our IOT platform there are many gateways connected to the rules server where each one is connected to a set of sensors.

   A requirement may so arise that developer wants to know all the gateways registered with the platform. So we have provided a GET API that returns a list of all such gateways.

Steps to access the API are:

(a) Obtain a secure token by sending a permanent key provided at the time of registering the Application and send it in the request as a query parameter.

(b) Apart from the access token this API does not expect anything else and returns the desired list if the access token is valid.

An example response is shown below:


```
{
    "status": 1,
    "statusDesc": "Success",
    "version": 1,
    "timestamp": 1427836755984,
    "comments": "All the gateways registered with the system",
    "request": {
        "accessKey": "1ADBCD234"
    },
```


Listing 1: The response of /repository/gateways?accessKey=1ADBCD234 - Part 1

```
    "data": {
        "gateways": [
            {
                "gatewayID": "IJKL2345",
                "locationID": 1
            },
            {
                "gatewayID": "ABCD1234",
                "locationID": 2
            }
        ]
    }
}
```


Listing 2: The response of /repository/gateways?accessKey=1ADBCD234 - Part 2

2. **/repository/sensors/family/FID**

   Sensors of various families are supoorted by the platform. Since there are many families an application developer may want a list of sensors belonging to a particular family. This GET API returns all the sensors of a requested family that are registered with the platform.

   Steps to access the API are:

   (a) Obtain a secure token by sending a permanent key provided at the time of registering the Application and send it in the request as a query parameter.

(b) Apart from the access token this API does not expect anything else and returns the desired list if the access token is valid.

An example response is shown below:


```
{
    "status": 1,
    "statusDesc": "Success",
    "version": 1,
    "timestamp": 1427837882673,
    "comments": "Data of all the Temperature sensors",
    "request": {
        "accessKey": "1ADBCD234"
    },
    "data": {
        "sensors": [
            {
                "sensorID": 1,
                "sensorFamily": "Temperature",
                "unitOfMeasure": "Celsius",
                "locationId": 1,
                "TypeHandler": "Handler1",
                "Protocol": "WiFi"
            },
```


Listing 3: The response of /repository/sensors/families/1?accessKey=1ADBCD234

```
            {
                "sensorID": 2,
                "sensorFamily": "Temperature",
                "unitOfMeasure": "Celsius",
                "locationId": 2,
                "TypeHandler": "Handler2",
                "Protocol": "BLE"
            }
        ]
    }
}
```


Listing 4: The response of /repository/sensors/families/1?accessKey=1ADBCD234

3. **/repository/sensors/protocol/PNAME**

   Sensors of heterogeneous protocols are supported by the platform. Since there are many protocols an application developer may want a list of sensors using a certain protocol to transmit the data. This GET API returns all the sensors using a requested protocol that are registered with the platform.

   Steps to access the API are:

   (a) Obtain a secure token by sending a permanent key provided at the time of registering the Application and send it in the request as a query parameter.

   (b) Apart from the access token this API does not expect anything else

and returns the desired list if the access token is valid.

An example response is shown below:


```
{
    "status": 1,
    "statusDesc": "Success",
    "version": 1,
    "timestamp": 1427838749931,
    "comments": "Data of all the sensors running on WiFi",
    "request": {
        "accessKey": "1ADBCD234"
    },
    "data": {
        "sensorData": [
            {
                "sensorID": 1,
                "Protocol": "BLE",
                "sensorFamily": "Temperature",
                "unitOfMeasure": "Degrees",
                "locationId": 1,
                "TypeHandler": "Handler1",
            },
```


Listing 5: The response of /repository/sensors/protocol/WiFi?accessKey=1ADBCD234 - Part 1

```
            {
                "sensorID": 1,
                "sensorFamily": "Humidity",
                "unitOfMeasure": "Water vapour content in percentages",
                "locationId": 1,
                "TypeHandler": "Handler2",
                "Protocol": "WiFi"
            }
        ]
    }
}
```


Listing 6: The response of /repository/sensors/protocol/WiFi?accessKey=1ADBCD234 - Part 2

2.1.2 Registry APIs

1. **/registry/gateway/GID**

   In our IOT platform there are many gateways connected to the rules server where each one is connected to a set of sensors.

   A requirement may so arise that developer wants to know all the sensors connected to a particular gateway. So we have provided a GET API that returns a list of all the sensors given a gateway ID.

   Steps to access the API are:

   (a) Obtain a secure token by sending a permanent key provided at the

time of registering the Application and send it in the request as a query parameter.

(b) Apart from the access token this API does not expect anything else and returns the desired list of sensors if the access token is valid.

(c) But it does entertain an optional query parameter "active". When this parameter is sent in with a value equal to "1", all the active sensors connected to the requested gateway are returned.

An example response is shown below:


```
{
    "status": 1,
    "statusDesc": "Success",
    "version": 1,
    "GatewayID": "100",
    "timestamp": 1427837345068,
    "comments": "All sensors connected to Gateway 100",
    "request": {
        "lat": "35",
        "long": "45",
        "range": "50",
        "accessKey": "1234567890"
    },
```


Listing 7: The response of /registry/gateways/100?accessKey=1234567890 - Part 1

```
    "data": {
        "sensorData": [
            {
                "sensorID": "3",
                "sensorFamily": "Humidity",
                "sensorStatus": "on",
                "lastReportedTime": 1427837340068,
                "sensorProtocol": "BLE",
                "locationId": 1
            },
            {
                "sensorID": "9",
                "sensorFamily": "Temperature",
                "sensorStatus": "off",
                "lastReportedTime": 1427837336068,
                "sensorProtocol": "Wifi",
                "locationId": 2
            }
        ]
    }
}
```


Listing 8: The response of /registry/gateways/100?accessKey=1234567890 - Part 2

2. **/registry/sensors/location**

   One of the simplest use case for sensory data would be to request the data of all the sensors in a certain range of given location. To serve this use case we have provided a GET API that returns a list of all the

sensors in a given range of a given location. Range and location details are sent in as query parameters.

Steps to access the API are:

(a) Obtain a secure token by sending a permanent key provided at the time of registering the Application and send it in the request as a query parameter.

(b) Apart from the access token, this API expects lattitude, longitude and range as query parameters 'lat', 'long', and 'range' respectively. If the provided access token is valid then the list of all the sensors satisfying the range criterion is returned.

(c) API also entertains an optional query parameter "active". When this parameter is sent in with a value equal to "1", all the active sensors satisfying the given criteria are returned.

An example response is shown below:

```
{
    "status": 1,
    "statusDesc": "Success",
    "version": 1,
    "timestamp": 1427837345068,
    "comments": "All sensors in the range of 50 meters",
    "request": {
        "lat": "35",
        "long": "45",
        "range": "50",
        "accessKey": "1234567890"
    },
    "data": {
        "sensorData": [
            {
                "sensorID": "3",
                "sensorFamily": "Humidity",
                "sensorStatus": "on",
                "lastReportedTime": 1427837340068,
                "sensorProtocol": "BLE",
                "locationId": 1
            },
```


Listing 9: The response to the request of getting all the sensors in the range of 50 meters of a location having lattitude and longitude 35 and 45 respectively - Part 1.

```
            {
                "sensorID": "9",
                "sensorFamily": "Temperature",
                "sensorStatus": "off",
                "lastReportedTime": 1427837336068,
                "sensorProtocol": "Wifi",
                "locationId": 2
            }
        ]
    }
}
```


Listing 10: The response to the request of getting all the sensors in the range of 50 meters of a location having lattitude and longitude 35 and 45 respectively - Part 2.

3. **/registry/sensors/family/FID**

   Sensors of various families are supoorted by the platform. Since there are many families an application developer may want a list of sensors of a certain family. This GET API returns all the sensors of a requested family that are registered with the platform or only the active ones.

   Steps to access the API are:

   (a) Obtain a secure token by sending a permanent key provided at the time of registering the Application and send it in the request as a query parameter.

   (b) Apart from the access token this API does not expect anything else

and returns the desired list of sensors if the access token is valid.

(c) But it does entertain an optional query parameter "active". When this parameter is sent in with a value equal to "1", all the active sensors of the requested family are returned.

An example response is shown below:


```
{
    "status": 1,
    "statusDesc": "Success",
    "version": 1,
    "timestamp": 1427837859976,
    "comments": "All the active Temperature sensors",
    "data": {
        "sensorData": [
            {
                "sensorID": "3",
                "sensorFamily": "Temperature",
                "sensorStatus": "on",
                "lastReportedTime": 1427837854976,
                "sensorProtocol": "BLE",
                "locationId": 1
            },
```


Listing 11: The response to the request for getting the active sensors of family 1 - Part 1

```
            {
                "sensorID": "9",
                "sensorFamily": "Temperature",
                "sensorStatus": "on",
                "lastReportedTime": 1427837850976,
                "sensorProtocol": "Wifi",
                "locationId": 89
            }
        ]
    }
}
```


Listing 12: The response to the request for getting the active sensors of family 1 - Part 2

2.2 Command APIs

The 2 APIs exposed in Command APIs are :

1. Often the logic server may need the data of all sensors of a type in a specific time window specified by the start time and end time. It could be the data of all temperature sensors in a building or the data of location sensors deployed on mobile devices spread across an area. A command API call can be made to fetch the data of a single sensor at that instant as well.

   The command propagates through the gateway which keeps checking for commands at regular interval and executes it to fetch data from corresponding sensor. A sensor also acting as an actuator can take the command and execute it. For example, if a sensor is feeding an On/Off switch, it can receive the command and impact the switch accordingly. Eg: /v1/temperature, /v1/humidity

```
"/v1/<sensor-family>" : {
// Sensor family will be the one of the sensor
//families registered in the sensor registry.
    "Payload" : {
        "start" : "<start timestamp>",
        "end"   : "<end timestamp>",
        "access\_key" : "<secret key which the app uses
         to communicate with API>"
    }
    ...
    ...
    // new entry goes here
}
```

Listing 13: Payload to /v1/<sensor-family>

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

    "request": {
    "sensorFamily": "<sensor_family_sent_in_request>",
    "startTime": "<start_time_sent_in_request>",
    "endTime": "<start_time_sent_in_request>",
    },

    "data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>",
    "locationId":"<location Id>"
    },
    {"sensorId" : "<Id>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>",
    "locationId":"<location Id>"
    },
    ...,
    ...,

    ]
}
```

Listing 14: The response of /v1/<sensor_family>

2. The application can also retrieve data of all type of sensors based on a location which could be crucial in some location centric application, which requires to know pressure, temperature, humidity etc values to make a decision.

```
{
    "/v1/location/<locationId>" {
    // will be the one of the sensor family
    //registered in the sensor registry.
        "Payload"  : {
            "start" : "<start timestamp>",
            "end"   : "<end timestamp>",
            "access\_key" : "<secret key which the app uses to
            communicate with API>",
        }
        ...
        ...
        // new entry goes here
    }
}
```

Listing 15: Payload to /v1/location/<locationId>

Eg: /v1/100

The above API will provide data of all type of sensors at that location within the time window specified by the start time and end time. The response will be in the following format

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
    "request": {
        "locationId": "<location_id_sent_in_request>",
        "startTime": "<start_time_sent_in_request>",
        "endTime": "<start_time_sent_in_request>"
    },
    "data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    {"sensorId" : "<Id>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ...,
    ...,

    ]
    }
}
```

Listing 16: The response of /v1/location/<locationId>

2.3 Callback APIs

The IoT platform has a feature called 'Callback Updates' that enables apps to subscribe to changes(or define a set of rules) in certain pieces of data. When a change occurs, an HTTP 'POST' request will be sent to a callback URL belonging to that app. This makes apps more efficient, as they know exactly when a change has happened, and don't need to rely on continuous or even periodic API requests when changes aren't happening.

2.3.1 Creating a Callback

Real Time updates are subscribed to by using the /v1/callback/ <sensorFamilyId> Callback API.

2.3.2 Setting Up your Callback URL

First, a callback URL needs to be prepared by the application Logic server. This URL should be accessible by Filter server, and should be able to receive both the POST data that is sent when an update happens, but also accept GET requests in order to verify subscriptions.

This URL should always return a 200 OK HTTP response when invoked by Filter Server. The API will register various callbacks for sensor types mentioned above by sensorFamilyId.

```
{
    "/v1/callback/<sensorFamilyId>" {
        "payload:" {
        "minimum-threshold"  : "<min threshold of physical quantity
         the sensor measures>",
        "maximum-threshold"  : "<max threshold of physical quantity
        the sensor measures>",
        "interval"           : "<time interval in seconds if data
        has to be sent in intervals>",
        "callbackOnInterval" : "<boolean value>"
        //true, if for every above mentioned interval,
        //data needs to be sent. Else, false
        "pushURL"         : "The callback URL to which data
        should be pushed"
        "access_key"      : "<secret key which the app uses to
        communicate with API>"
        },
        ...,
        ...,
    }
}
```

Listing 17: Payload to /v1/callback/<sensorFamilyId>

```
{
    "triggerType"    : <min_threshold/max_threshold/interval>
    "value"          : <value in units measured by that sensor type>
}
```

Listing 18: The JSON response posted to pushURL above.

2.4 Basic APIs

In order to make API calls to our platform, one needs to log into platform by making a basic API call and get the access key which provides temporary, secure access to platform APIs. One important aspect to understand about access key is that they are portable. Once you have an access token you can use it to make calls from a mobile client, a web browser, or from application logic server to Filter server.

```
{
    v1/auth/: {
        "Payload" : {
        "username"   : "<username registered with platform>",
        "password"   : "<password assigned by the platform>",
    }
}
```

Listing 19: Payload to v1/auth/

The JSON response below will grant access key.

```
{
    "Response" {
        "access_key"     : <access key generated and valid for
        time mentioned below>
        "validForTime"   :<Time in seconds after which the session
        expires between filter server and logic server>
    }
}
```

Listing 20: The JSON response granting the access key.

2.5 Handling Errors

Improper requests made to our APIs can result in a number of different error responses, and there are a few basic recovery steps. The important step is to check for status and status description to identify the error. Receiving Error Codes

The following represents a common error response resulting from a failed API request:


```
{
    "status": 0,
    "statusDesc": "Payload Incorrect",
    "version": 1,
    "timestamp": 1428079710466,
    "comments": null,
    "request": {
        "startTime": "1",
        "endTime": "1"
    },
    "data": null
}
```


Listing 21: The JSON response indicating error.

Chapter 3

Block Diagrams

3.1 Gateway Block Diagrams

i. Gateway Module Block Diagram

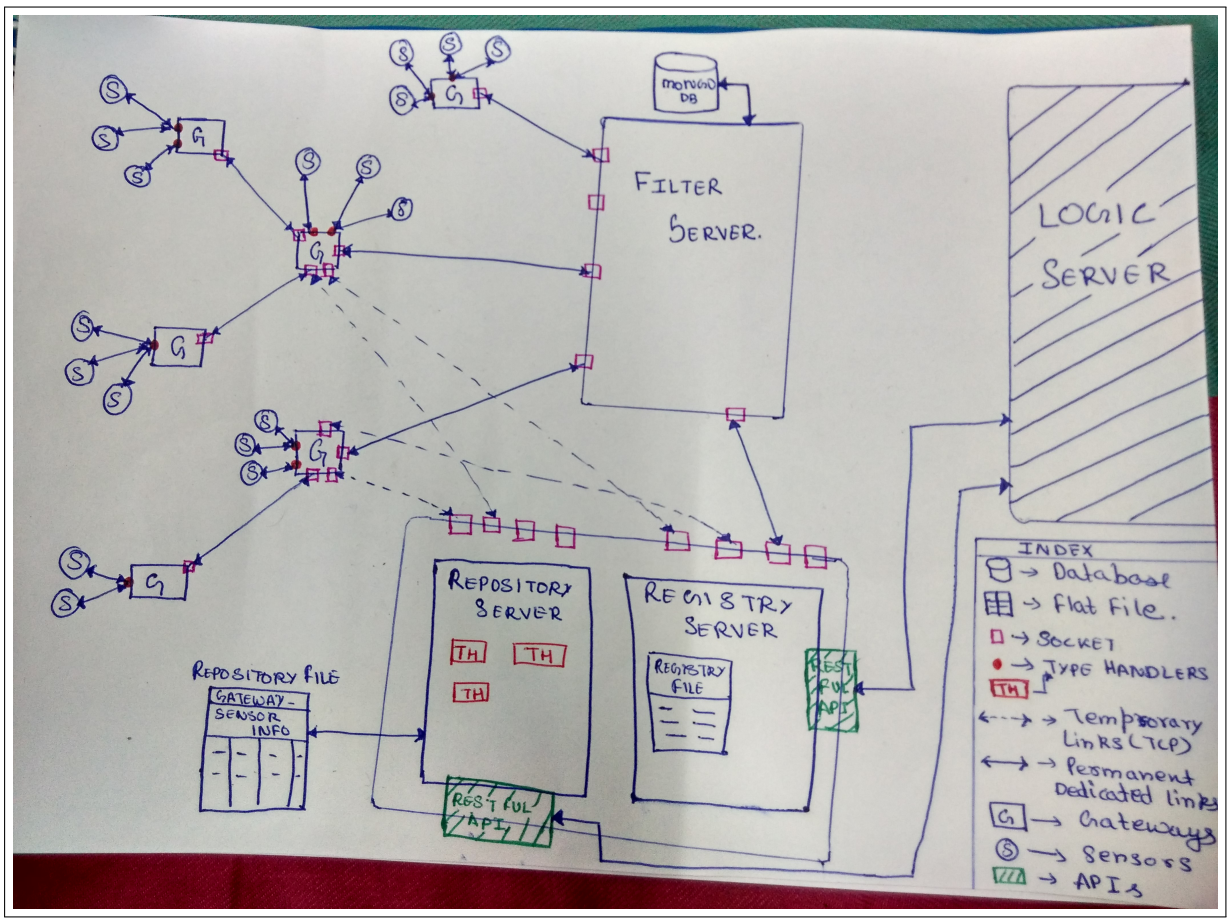

Figure 3.1: *Gateway Module*

ii. Gateway Module Interactions Block Diagram

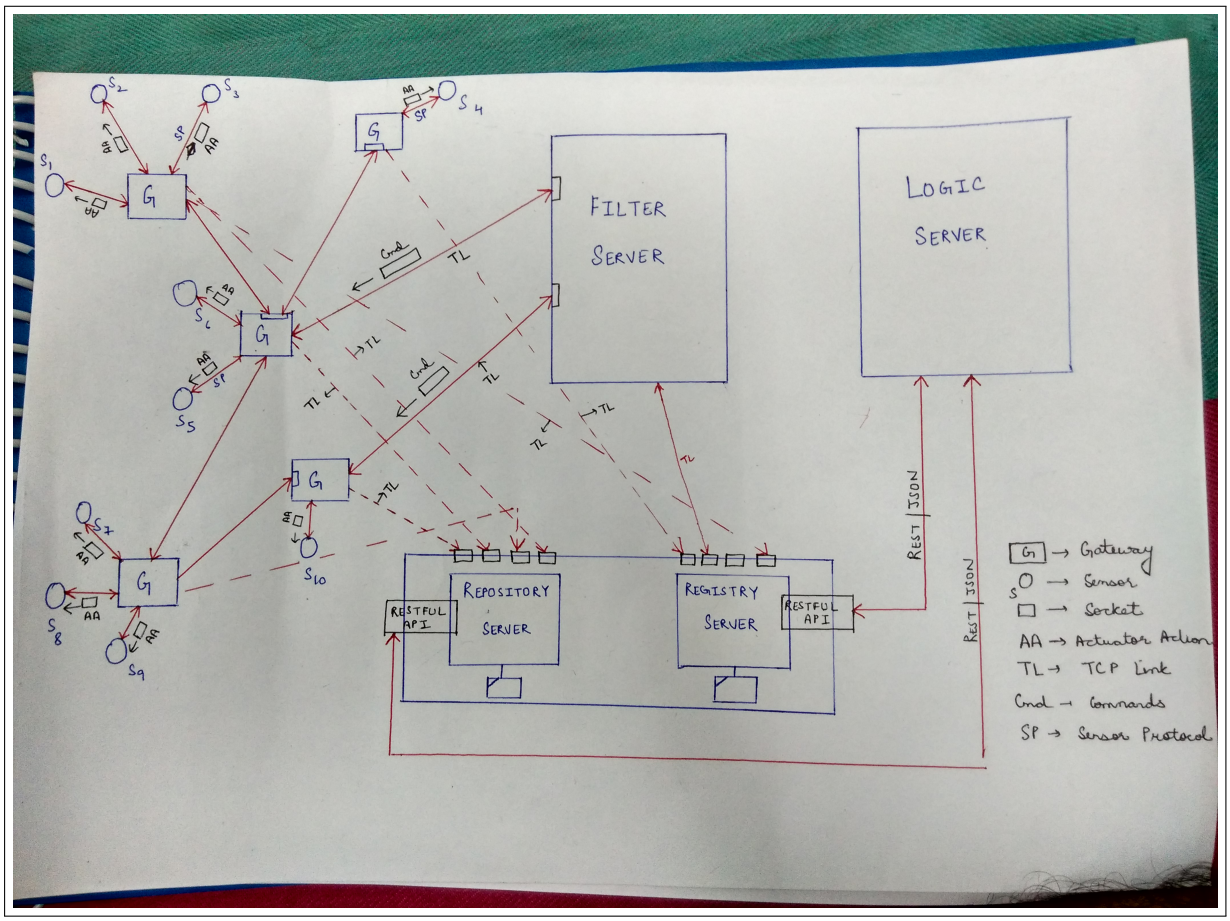

Figure 3.2: *Gateway Module Interactions*

3.2 Filter Server Block Diagrams

i. Filter Server Module Block Diagram

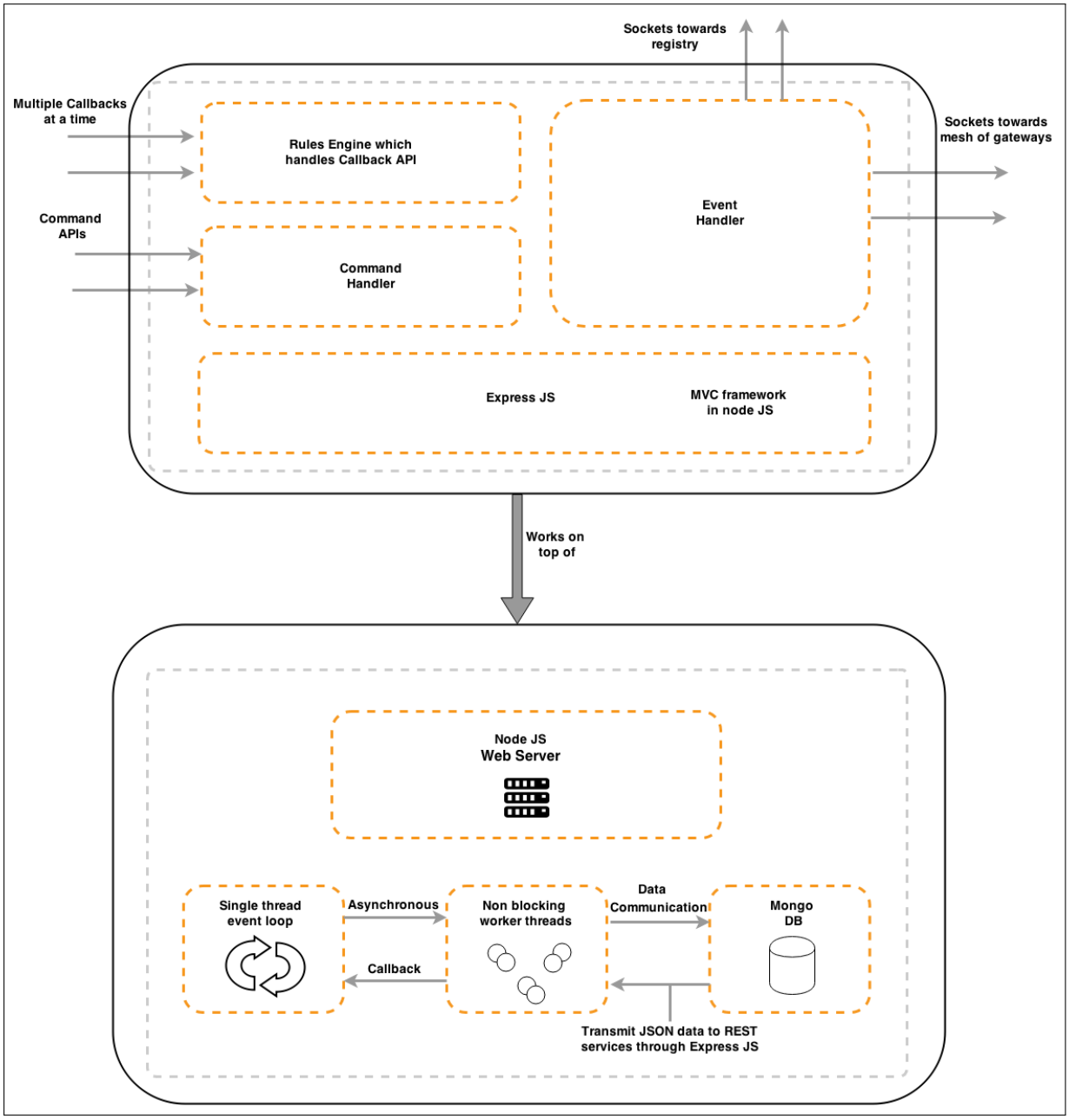

Figure 3.3: *Filter Server Module*

ii. Filter Server Interactions Block Diagram

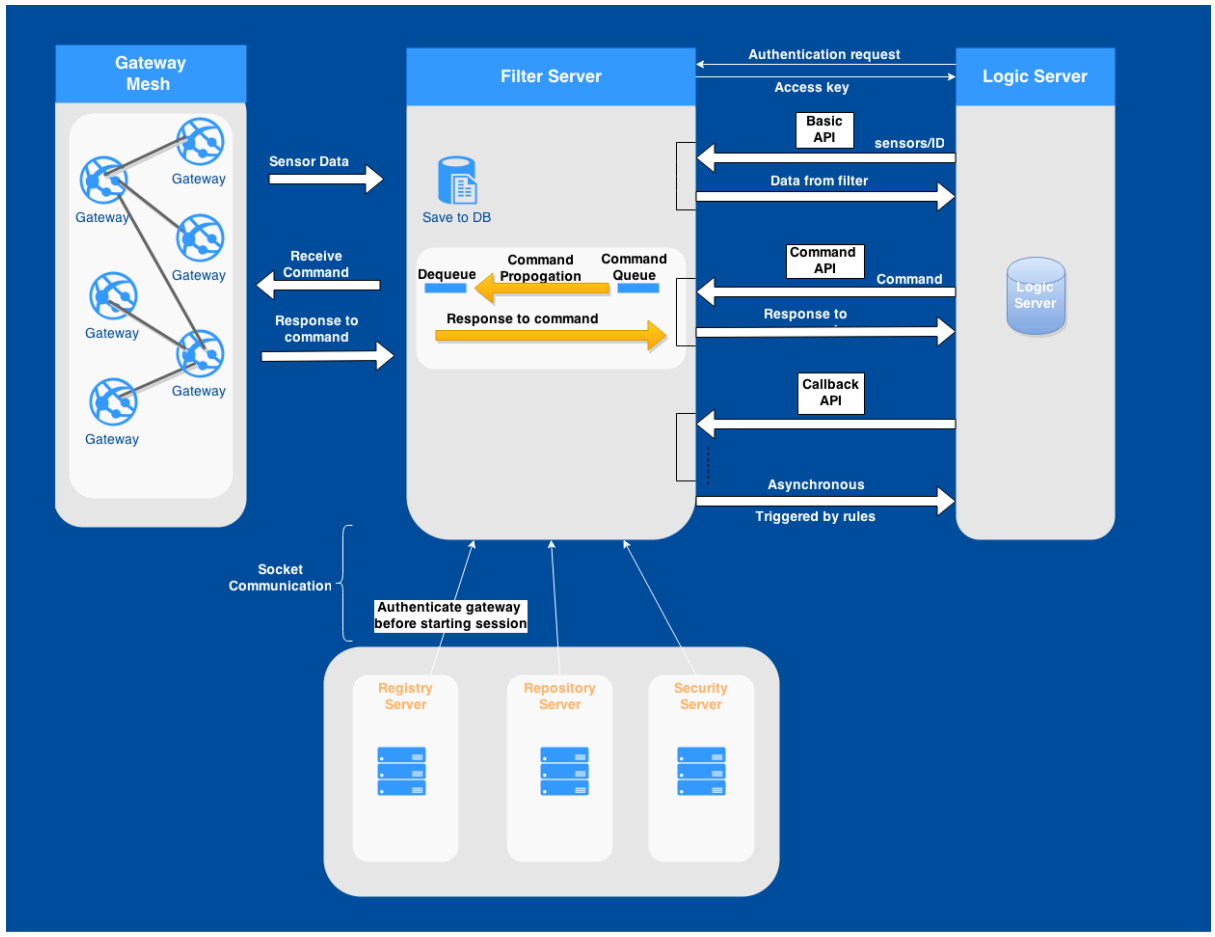

Figure 3.4: *Filter Server Module Interactions*

3.3 Application Tier Block Diagram

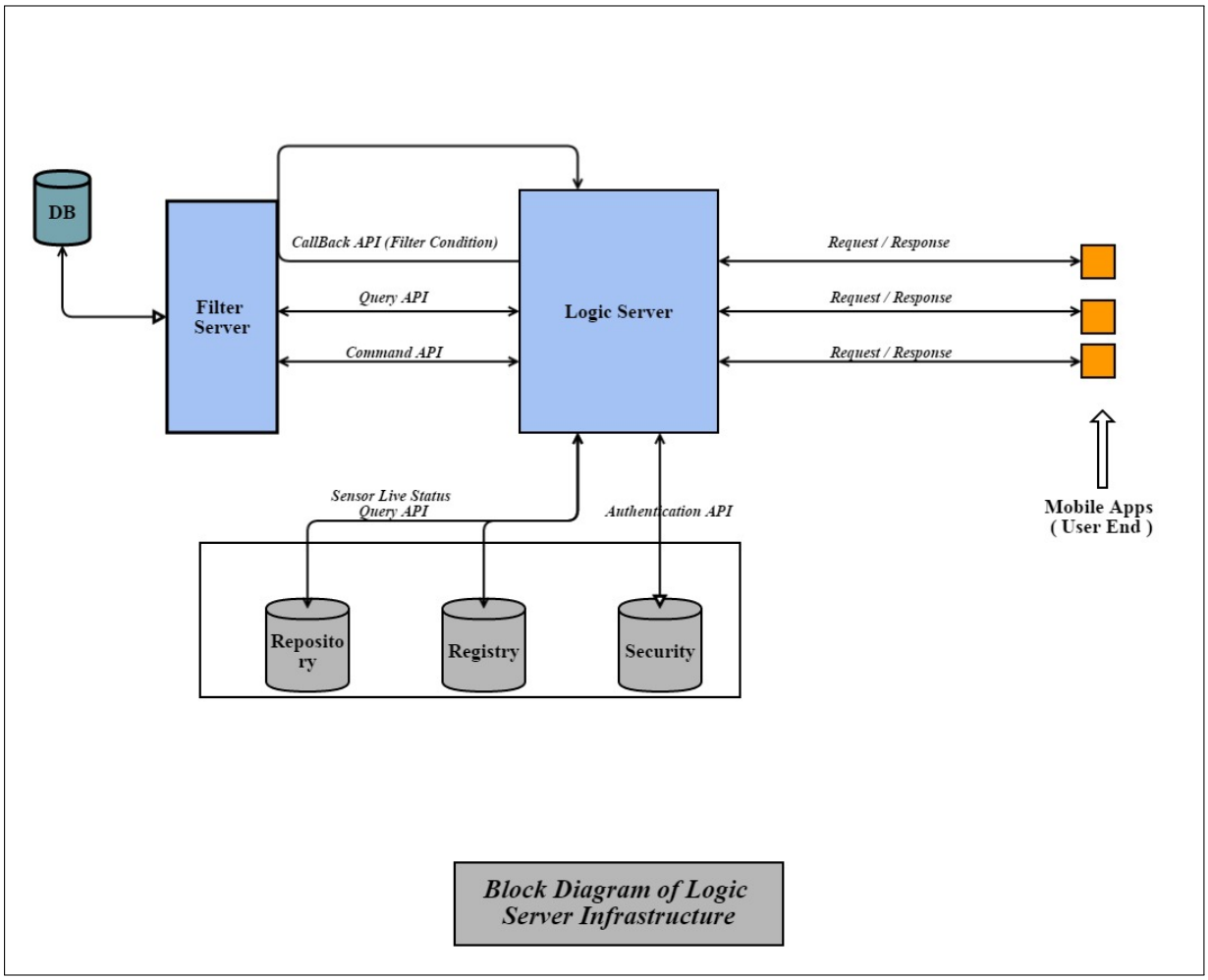

Figure 3.5: *Application Tier Block Diagram*

3.4 IOT Platform Overall Block Diagram

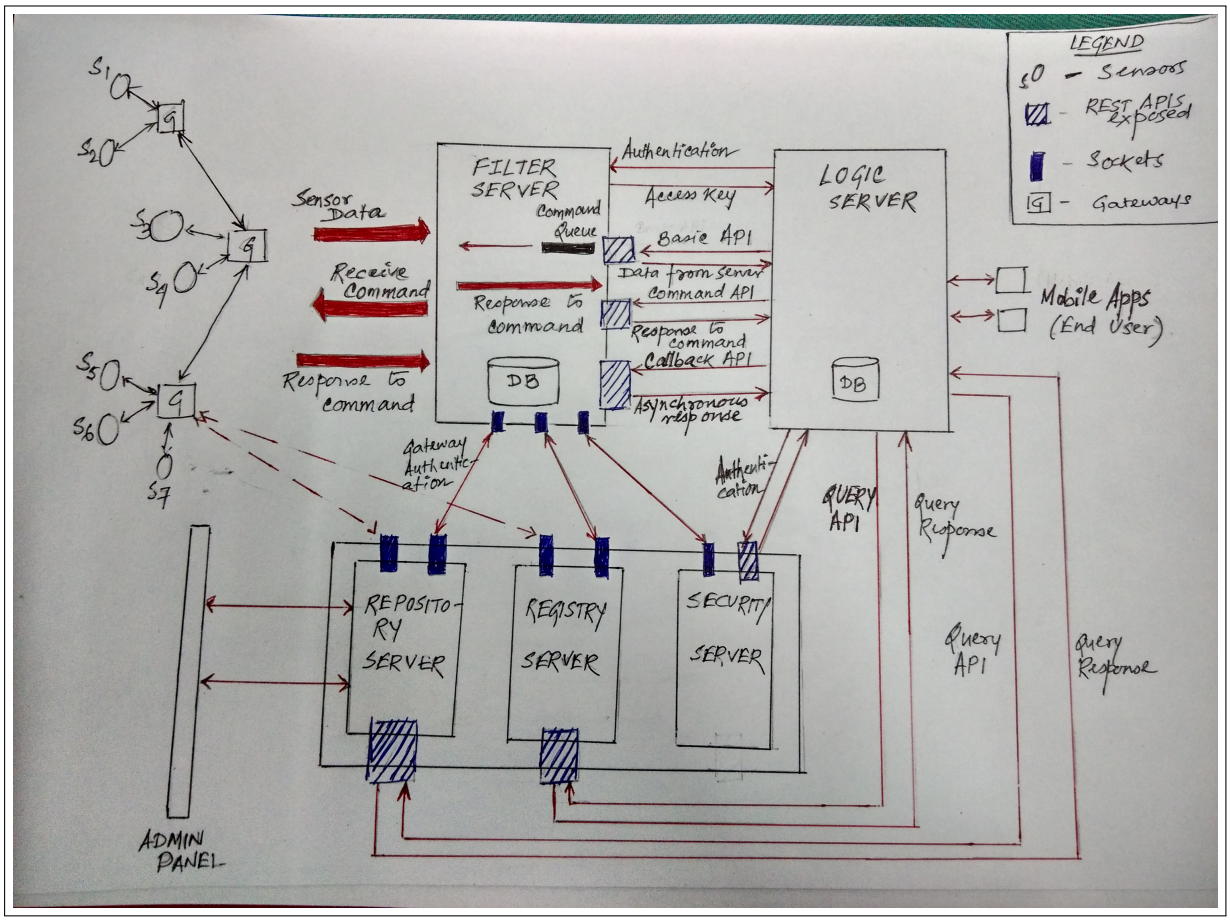

Figure 3.6: *IOT Platform Overall Block Diagram*