Internet of Things Apps Platform

A Project Design Document
Version-1.0

For the module
Filter Server

Submitted by

Team 1 :

Veerendra Bidare	201405571
Swapnil Pawar	201405513
Abhishek Mungoli	201405577

Of
Group Number - 5

Under the guidance of
Mr. Ramesh Loganathan

For the course
Internals of Application Server
IIIT, Hyderabad

23rd March, 2015

Abstract

The Filter Server module is the heart of the IoT platform. It is implemented in Express.js which is a MVC framework in Node.js. It interacts with a mesh of Gateways at one end, and a logic server at the other. It collects the data from the mesh of Gateways, stores it in its back-end powered by MongoDB, and exposes RESTful APIs which is used by Logic Server to pull the data. It exposes three different types of APIs for three different scenario.

Contents

List of Figures

Chapter 1

Life Cycle of the Module

1.1 Bringing Module into Action

i. The Filter server module consists of REST APIs on top of node.js MVC framework, Express.js which uses MongoDB internally for scalable and flexible Database performance. The module comes into action when the application is first launched, and stopped only when the application is stopped end to end. During its functioning, its monitored for two sub-processes.

ii. One of them is its performance in collecting data from a mesh of gateways, checking the log files of APIs to monitor which APIs were called and what were the response codes. If any error code is encountered, the reason for failure is investigated and if required reported to app admin.

iii. The performance of APIs exposed to the application tier is also monitored in the same way as mentioned above.

iv. Additionally, there is a general monitoring system, to check if all the

APIs exposed are healthy by making continuous health ping all the time.

1.2 User's Application Components

Any User's application components will not be directly deployed on this module. However, application developers can use the documentation to build Logic server and its associated apps in the application tier which will communicate.

1.3 Design Considerations

As all the functionality of this module is based on REST, which internally uses HTTP states significantly. Any device or other module interacting with this module should be able to communicate using HTTP. REST uses JSON objects for response, which is supported by all heterogeneous technology stack.

Chapter 2

List of Sub Modules

2.1 Block Diagram

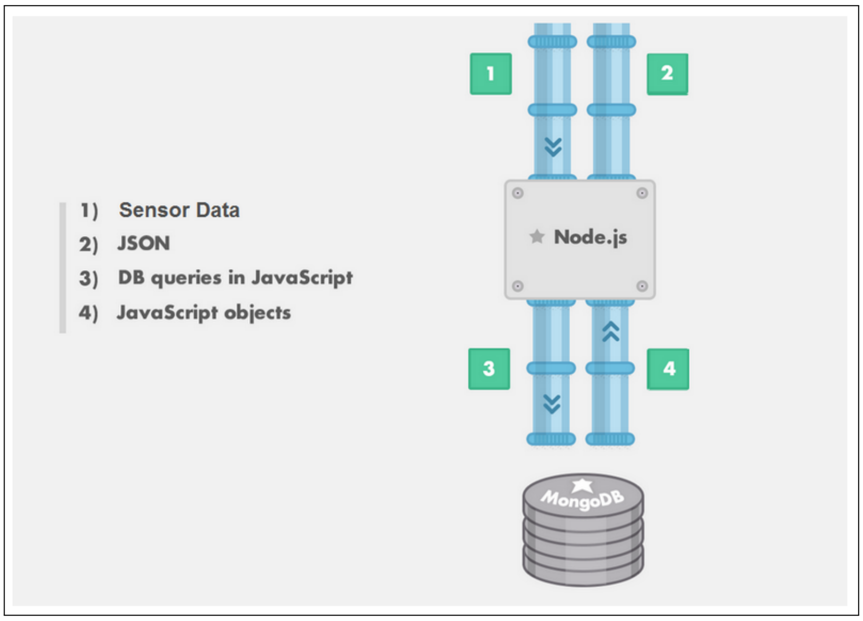

Figure 2.1: *Block Diagram of Sub Modules of Filter Server*

2.2 Interactions between Sub Modules

- The data format used is JSON. The ïňĄlter server pushes the commands to Gateways which in turn execute them for the data sent by the devices connected to it. These pushes are simulated pushes.

- The received data is processed by the server and pushed into the database.
- The server uses Node-JS for processing and MongoDb for storing data
- The results of the queries fired on the database are retrieved using javascript objects in form of JSON

Chapter 3

Overview of Sub Modules

3.1 Sub Modules and their functionality

i. HTTP requests - The HTTP requests will be used for CRUD operations. The operation and respective HTTP verbs are as follows.

   a. Create - PUT,

   b. Read - GET,

   c. Update - POST,

   d. Delete - DELETE

ii. JSON - All the data exchange will be using json objects.

iii. The performance of APIs exposed to the application tier is also monitored in the same way as mentioned above.

iv. DB queries - In order to serve the Command APIs and Callback APIs made by Logic server, filter server uses DB queries execute according to payload of APIs.The DB used at the back-end is MongoDB.

v. Javascript objects - Various javascript objects used in node.js server.

3.2 Interactions With Other Modules

Interactions with other modules is mainly with mesh of gateways and logic server at the application tier using RESTful Web Services explained in the next chapter.

Chapter 4

REST APIs exposed to Logic Server

4.1 Why REST

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 category of APIs exposed.

4.2 Category of APIs exposed

i. Command APIs.

ii. Callback APIs.

iii. Basic API.

4.3 Command APIs

The 2 APIs exposed in Command APIs are :

```
{
    "/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 1: Payload to /v1/<sensor-family>

Eg: /v1/temperature, /v1/humidity

The above API will provide data of the specific type of sensors 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": {
    "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 2: The response of /v1/<sensor_family>

```
{
    "/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 3: 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 4: The response of /v1/location/<locationId>

4.4 Callback APIs

The second type of API is callback API, in which logic server registers the rules.Based on these rules, the Filter server makes a callback.

```
{
    "/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 URL to which data should be pushed"
        "access_key"      : "<secret key which the app uses to
        communicate with API>"
        },
        ...,
        ...,
    }
}
```

Listing 5: Payload to /v1/callback/<sensorFamilyId>

The above API will register various callbacks for sensor types mentioned above by sensorFamilyId.

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

Listing 6: The JSON response posted to pushURL above.

4.5 Basic APIs

There is a basic API which is used to create a session between Filter server and Logic server.

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

Listing 7: The JSON response posted to pushURL above.

The JSON response posted to pushURL above.

```
{
    "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 8: The JSON response posted to pushURL above.

Chapter 5

REST APIs exposed to mesh of Gateways

5.1 APIs exposed to Gateway

```
{
    "v1/gateways/post" : {
      "Payload" : {
        "gatwayID" : "<Id>",
        "isDirect" : "yes/no",
        "originGatewayId" : "key",
        "timestamp" : "<timestamp of recording by sensor>",
        "data" : {
          "sensorId" : "<Id>",
          "sensorData" : "<composite object>",
          //array of JSON objects
        }
      },
    }
    ...
    ...

    // new entry goes here
}
```

Listing 9: Payload for Gateway API

5.2 Response of Filter Server

```
{
    "gatewayResponse" : {
      "status" : "0/1/2"{
        // 0 : Failure
        // 1 : Partial Success
        // 2: Completely Successful
      },
    ...
    ...

    // new entry goes here
}
```

Listing 10: Payload for Gateway API