# Generic IOT Platform

## User Guide

**Group 1**

**Guided by**

**Ramesh Loganathan**

IIIT, HYDERABAD

## **Section 1: Admin privilege**

The admin is the user who has the privilege to add, remove and monitor gateways and sensors.

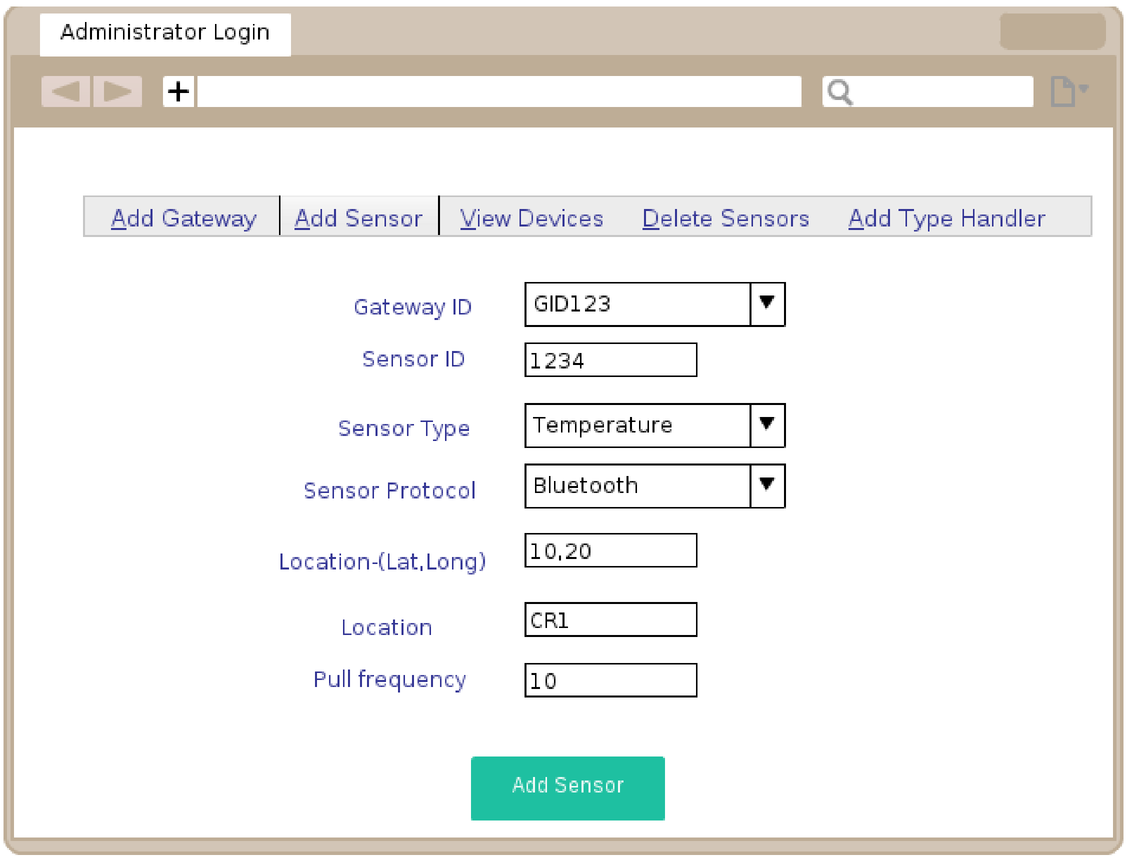
### 1. Manage devices

He can operate using two ways a) Using admin portal b) managing repository.xml

### a. Using Admin Portal

Login URL: https://GivenDomainName.com:Portnumber/admin.html

The admin needs to login using the valid credentials in order to enter to admin portal.



| | |
|---|---|
| Gateway ID | GID123 |
| Sensor ID | 1234 |
| Sensor Type | Temperature |
| Sensor Protocol | Bluetooth |
| Location-(Lat,Long) | 10,20 |
| Location | CR1 |
| Pull frequency | 10 |

1. For the addition of new sensor below information are required.
    a. Sensor Id – Unique Device ID (eg. RF3C6000MNA)
    b. Type Handler – this is a dropdown which shows the entire available communication medium such as Bluetooth, Wi-Fi etc…
    c. Type – what type of sensor it is (light sensor, temperature sensor etc..)

    d. GPS coordinates – coordinates on which the sensor resides. This is required because based on this the gateway associated will be decided.
    e. Location Name – Common textual location Identifier (room1, lab2 .. )
    f. Pull frequency – Frequency at which sensor sends the data.

2. **For deletion of existing sensor** – go to delete sensor tab and select the sensor to be deleted.
3. **To add new type handler:** Go to New Type handler tab upload the corresponding jar. Now the jar starts appearing in the drop down.
4. **Add gateway** using below information
    a. Gateway Id – Unique Device ID (eg. RF3C6000MNA)
    b. GPS location – coordinates on which the gateway resides. This is required because based on this the gateway associated will be decided.
    c. Location Name – Common textual location Identifier ( room1, lab2 .. )

### b. Using repository.xml

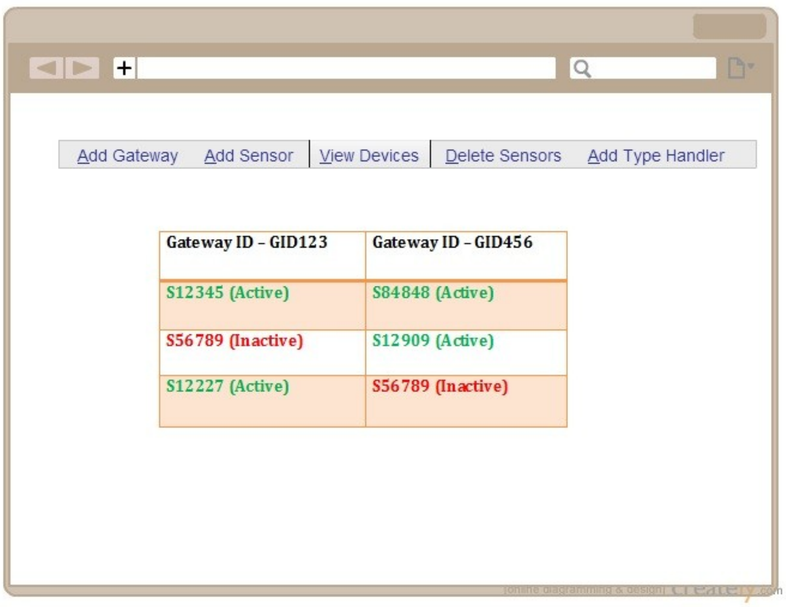
Enter the values in the repository.xml under corresponding gateway tag.

Repository.xml format:

```
<repository>

        <gateway id=”1”>

                <hardware_id>...</hardware_id>

                <location> …. </location>

                <location_name>….</location_name>

                <sensor id=”s1”>

                        <hardware_id>...</hardware_id>

                        <type_handler>….</type_handler>

                        <type>….</type>

                        <gps_location>...</gps_location>

                        <pull_frequency>...</pull_frequency>
        </sensor>
        <sensor id="s2">
            <hardware_id>...</hardware_id>
            <type_handler>....</type_handler>
            <type>....</type>
            <gps_location>...</gps_location>
            <pull_frequency>...</pull_frequency>
        </sensor>
    </gateway>
</repository>
```

2. **Monitoring of devices**

The state of the devices can be monitored from this page where the list of sensors registered will be shown along with status information.



| Gateway ID – GID123 | Gateway ID – GID456 |
|---|---|
| S12345 (Active) | S84848 (Active) |
| S56789 (Inactive) | S12909 (Active) |
| S12227 (Active) | S56789 (Inactive) |

After adding/deleting the new device to the platform it is necessary to restart all the servers. This can be achieved using either restart sever button in monitor page or by running restart.sh script.

## Section 2: Logical Server and Registry Interface

Following are the REST APIs exposed to Logical Server by the Registry

### 1. /getsensors/geolocation

This REST API is used by ***logical server*** to get all sensors located in a particular radius(in meters) of given ***geo co-ordinates (Latitude and Longitude)***. The input must be specified in JSON format as shown below. Output is "list of sensors" with its details.

**JSON input:**

```
- sessionId : S2dcf17181814b5b41506df
- latitude : 9.50523
- longitude : 51.31991
- radius : 10
```

**JSON output:**

```
- sessionId : S2dcf17181814b5b41506df
- sensorsinfo
    - sensorId : ERGG43534
    - type : temperature
    - location : CR1
    - geolocation
        - latitude : 9.50523
        - longitude : 51.31991
    - pollfrequency : 3
    - protocol : bluetooth
```

### 2. /getsensors/location

This REST API is used by ***logical server*** to get all sensors located of given ***location*** (common name given to a particular location). The input must be specified in JSON format as shown below. Output is “list of sensors” with its details.

**JSON input:**

```
- sessionId : S2dcf17181814b5b41506df
- location : CR1
```

**JSON output:**

```
- sessionId : S2dcf17181814b5b41506df
- sensorsinfo
    - sensorId : ERGG43534
    - type : temperature
    - location : CR1
    - geolocation
        - latitude : 9.50523
        - longitude : 51.31991
    - pollfrequency : 3
    - protocol : bluetooth
```

## Section 3: Application and IOT platform:

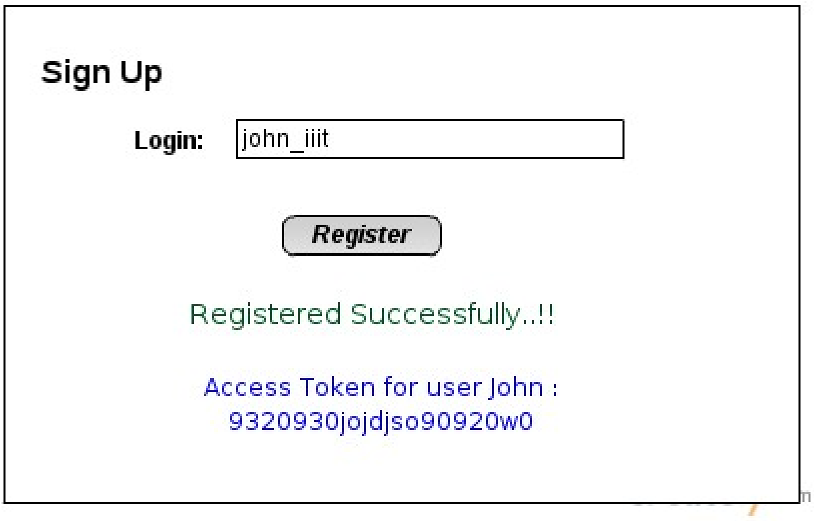
This section describes the registration process of application with the IOT platform. The platform is designed to generate a unique token for every application on registration. Process is described as follows.

### Register

Each application must register with the IOT platform in order to access the APIs exposed by the platform and sensor data collected by the platform. To register with the IOT platform, application must use the register page exposed by the platform and provide a unique username. A unique token will be sent back in response to the registration.

Application is expected to store the token for further API calls. This token along with username is sent in application requests which are validated by the **security layer** of the IOT platform.

Validity of the generated token expires once in every 6 months. If token is expired, 401 error code is sent as response to application requests. A fresh token will be sent which the application needs to use for all further http requests.


Sign Up

Login: john_iiit

Registered Successfully..!!

Access Token for user John : 9320930jojdjso90920w0

## Section 4: Logical Server and Filter Server interface

This section describes the interactions between logical server and filter server through the exposed REST APIs.

### 1. /getdata/sensorId

This REST API is used to retrieve the sensor data of a particular sensor with given sensor id. The input must be given in the form of GET parameter and the output will be in JSON format with the parameters as shown below.

**JSON output:**

```
- sensordata
  - sensorId : ERGG43534
  - type : temperature
  - unit : celsius
  - location : CR1
  - data : 34
  - timestamp : 1428007119
```

The API must be used when the application is interested in a single sensor data.

### 2. /getdata/geolocation

This REST API is used to get the sensor data based on the geo location. Input is sent in JSON format as shown in below figure. Expected parameters are sessionId, latitude, longitude, radius and type.

**sessionId** – id of current user session. Used for security purpose

**latitude, longitude** – Geo co-ordinates of the location from where the sensor data is required.

**Radius (optional)** – The parameter is specified in meters to indicate the sensor data requirement in given radius. This is an optional parameter. If not specified, data from exact location is fetched.

**Type (optional)** – This parameter defines the type of sensors from which the data is required. This parameter can be used by application it needs data of particular type of sensor where multiple types of sensors are installed. When parameter is not specified, data from all sensors in given location is returned.

Output contains array of JSON objects with sensor data in it. Along with data other details such as sensorId, type, unit, location, timestamp is also sent to help the application for processing of data.

**JSON input:**

```
- sessionId : S2dcf17181814b5b41506df
- latitude : 9.50523
- longitude : 51.31991
- radius : 10 // meters
- type : <temperature, image>
```

**JSON output:**

```
- sensordata
  - sensorId : ERFDF3534
  - type : temperature
  - unit : celsius
  - location : CR1
  - data : 34
  - timestamp : 1428007119
- sensordata
  - sensorId : ERGG43534
  - type : image
  - unit : jpeg
  - location : CR1
  - data : <binary>
  - timestamp : 1428007120
```

## 3. /getdata/location

This REST API is used to get the sensor data based on the *common name of location*. Input is sent in JSON format as shown in below figure. Expected parameters are sessionId, locationName and type.

**sessionId** – id of current user session. Used for security purpose

**locationName** – Common name of the location from where the sensor data is required. Example a room could be named as CR1. So the data from this room can be fetched by mentioning its name as CR1.

**Type (optional)** – This parameter defines the type of sensors from which the data is required. This parameter can be used by application it needs data of particular type of sensor where multiple types of sensors are installed. When parameter is not specified, data from all sensors in given location is returned.

Output contains array of JSON objects with sensor data in it. Along with data other details such as sensorId, type, unit, location, timestamp is also sent to help the application for processing of data.

**JSON input:**

```
- sessionId : S2dcf17181814b5b41506df
- locationName : CR1
- type : <temperature, image> // array
```

**JSON output:**

```
- sensordata
  - sensorId : ERFDF3534
  - type : temperature
  - unit : celsius
  - location : CR1
  - data : 34
  - timestamp : 1428007119
- sensordata
  - sensorId : ERGG43534
  - type : image
  - unit : jpeg
  - location : CR1
  - data : <binary>
  - timestamp : 1428007120
```

## 4. /getdata/registercallback/geolocation

This REST API is used to register call backs with the filter server based on geo location. Applications can use this call back when they want to get notified based on some event occurrence. For eg., when temperature in room goes beyond 40$^0$ C. The following are the parameters available in the call back.

**sessionId** – id of current user session. Used for security purpose

**latitude, longitude** – Geo co-ordinates of the location from where the sensor data is required.

**Radius (optional)** – The parameter is specified in meters to indicate the data requirement in given radius. This is an optional parameter. If not specified, data from exact location is fetched.

**Type** – This parameter defines the type of sensors from which the data is required. The parameter is mandatory to avoid the ambiguity that could be created when multiple types of sensors are installed in a particular location.

**minValue( conditionally optional)** : The value determines the min threshold below which notification is required. For eg., If the notification is required when temperature goes below $20^0$C, then 20 is mentioned in minValue. This parameter is contidionally optional meaning, either one of minValue or maxValue is required, otherwise error code **422** is sent to application.

**maxValue( conditionally optional)** : The value determines the min threshold above which notification is required. For eg., If the notification is required when temperature goes above $40^0$C, then 40 is mentioned in maxValue. This parameter is contidionally optional meaning, either one of minValue or maxValue is required, otherwise error code **422** is sent to application.

**tillWhen (optional)** : This parameter is specified when the application is interested in continuous notification periodically. For eg., if the application is interested to be notified when there is a motion in the room, then it can specify the image difference value threshold and the time until when it must be notified say until morning 10 am from current time. Thus, the notification is sent whenever motion is detected in the room till morning 10am.

**Frequency (conditionally optional )** : This parameter is co-related with *tillWhen*. It tells the frequency of check required to get the notifications. The value is specified in seconds. If the application is interested to get the call backs every 60 minutes once, then it sets this parameter. It is helpful to avoid getting call backs in high frequency.

Output gives the sensor data which has crossed the threshold value.

**JSON input:**

- sessionId : S2dcf17181814b5b41506df
- latitude : 9.50523
- longitude : 51.31991
- radius : 10
- type : temperature
- minvalue : 10
- maxvalue : 40
- tillWhen : 1428008119
- frequency : 60

**JSON output:**

- sensordata
  - sensorId : ERFDF3534
  - type : temperature
  - unit : celsius
  - location : CR1
  - data : 34
  - timestamp : 1428007119

## 5. /getdata/registercallback/location

This REST API is used to register call backs with the filter server based on common location name. Applications can use this call back when they want to get notified based on some event occurrence. For eg., when temperature in room goes beyond 40$^0$ C. The following are the parameters available in the call back.

**sessionId** – id of current user session. Used for security purpose

**locationName** – Common name of the location from where the sensor data is required. Example a room could be named as CR1. So the data from this room can be fetched by mentioning its name as CR1.

**Type** – This parameter defines the type of sensors from which the data is required. The parameter is mandatory to avoid the ambiguity that could be created when multiple types of sensors are installed in a particular location.

Other parameters are used in similar way as /getdata/registercallback/geolocation.

Output gives the sensor data which has crossed the threshold value.

**JSON input:**

```
- sessionId : S2dcf17181814b5b41506df
- locationName : CR1
- type : temperature // single valued
- minvalue : 10
- maxvalue : 40
- tillWhen : 1428008119
- frequency : 60
```

**JSON output:**

```
- sensordata
  - sensorId : ERFDF3534
  - type : temperature
  - unit : celsius
  - location : CR1
  - data : 34
  - timestamp : 1428007119
```

## 6. /getdata/registercallback/sensors

This callback is used to get the notification from specific sensors which the application is interested in. Application can use the sensors list returned from the api section 2.1/2.2. The following are the parameters used to register this call back.

**sessionId** – id of current user session. Used for security purpose.

**Sensors** – This is an array of sensor ids from which the data is required. Application must take care to send sensor ids of same type. Otherwise error **422** is sent.

Other parameters are used in similar way as /getdata/registercallback/geolocation.

Output gives the sensor data which has crossed the threshold value.

**JSON input:**

```
- sessionId : S2dcf17181814b5b41506df
- sensors
  - sId : ERFDF3534
  - sId : ERFDF3535
- minvalue : 10
- maxvalue : 40
- tillWhen : 1428008119
- frequency : 60
```

**JSON output:**

```
- sensordata
  - sensorId : ERFDF3534
  - type : temperature
  - unit : celsius
  - location : CR1
  - data : 34
  - timestamp : 1428007119
- sensordata
  - sensorId : ERFDF3535
  - type : temperature
  - unit : celsius
  - location : CR2
  - data : 37
  - timestamp : 1428007120
```

**List of Error codes:**

| Error code | Description |
|---|---|
| 401 | **Unauthorized** – This error is thrown when user provides wrong credentials in his application |
| 403 | **Forbidden** – Error is thrown when the user does not have proper privilege to access the url eg. Before login |
| 404 | **Not Found-** Error is thrown when requested API does not exists or the requested page does not exists. |
| 422 | **Unprocessable entity** – This error is thrown when the request contains improper JSON format or missing fields in JSON. |

**List of Status Codes:**

| **Status Code** | **Description** |
|---|---|
| 200 | Status OK |
| 206 | **Partial Content** – Sent when there is large sensor data needs to be sent to logical server where the data are sent in chunks |
| 204 | **No Content –** Sent when there is no data found for the query. |