

# IoT Platform For Monitoring End Devices and Controlling Them Through Applications

Group 1 : Team 3

May 8, 2021

Apurva Jadhav, Arushi Agarwal, Shreya Vanga

**Mentor and Professor :** Prof. Ramesh Loganathan

**TA:** Pratik Tiwari, Shubham Agarwal, Jay Krishna

# Contents

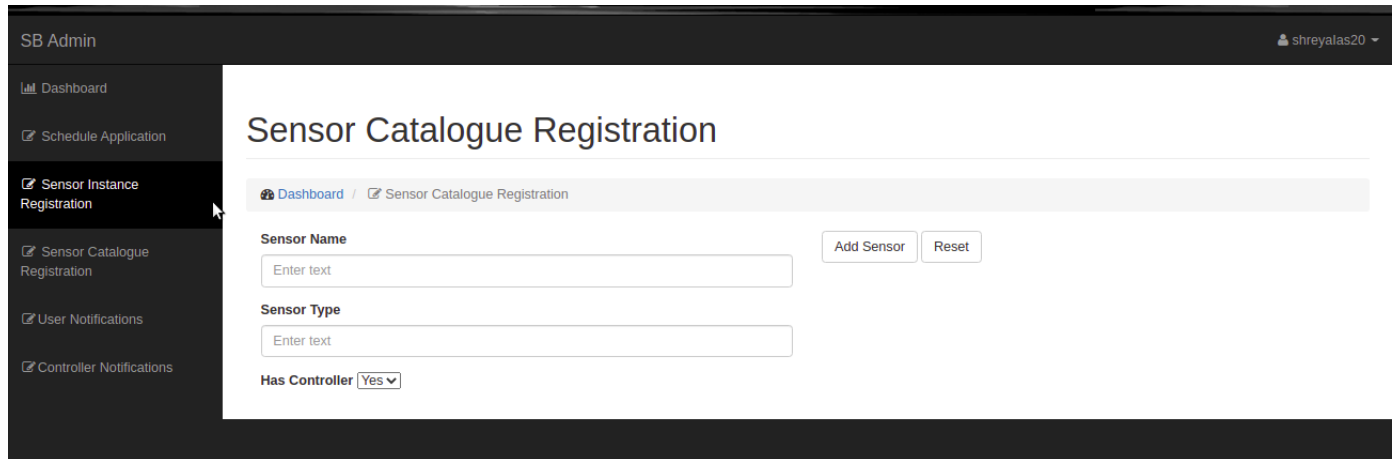
<b>1</b>	<b>Sensor Catalogue Registration</b>	<b>3</b>
1.1	User's view . . . . .	3
1.2	Internal implementation . . . . .	4
1.3	APIs Exposed . . . . .	4
<b>2</b>	<b>Sensor Instance Registration</b>	<b>5</b>
2.1	User's view . . . . .	5
2.2	Internal implementation . . . . .	6
2.3	APIs Exposed . . . . .	7
<b>3</b>	<b>Sensor Manager</b>	<b>7</b>
3.1	Basic Overview . . . . .	7
3.2	Responsibilities of Sensor manager . . . . .	7
3.3	Handling above requirements . . . . .	8
3.4	APIs Exposed . . . . .	8
3.5	Interaction with other modules . . . . .	10
<b>4</b>	<b>Technologies used</b>	<b>12</b>

# 1 Sensor Catalogue Registration

This platform allows new type of sensors to be installed and used by the applications. For this Sensor Catalogue Registration module is present in the system. The basic information to be supplied will be the the name of the sensor type and the types of data the sensor generates.

## 1.1 User's view

The user (configurer) will be provided with a UI to input the details about the type of the sensor as mentioned above.



The screenshot displays the 'SB Admin' web application interface. On the left is a dark sidebar with a menu containing 'Dashboard', 'Schedule Application', 'Sensor Instance Registration' (highlighted), 'Sensor Catalogue Registration', 'User Notifications', and 'Controller Notifications'. The main content area has a dark header with 'shreyalas20' in the top right. Below the header, the title 'Sensor Catalogue Registration' is centered. A breadcrumb trail shows 'Dashboard / Sensor Catalogue Registration'. The form contains three fields: 'Sensor Name' with an 'Add Sensor' button, 'Sensor Type' with a 'Reset' button, and 'Has Controller' with a dropdown menu set to 'Yes'.

- The user has to give the sensor name and its type as input and choose Yes/No option from the dropdown for controller.
- The data obtained from the UI is sent directly to the database.
- If the sensor name is not registered, it registers the sensor with the database

## 1.2 Internal implementation

The data input is captured through a JSON file and then processed and handled accordingly. The data acquired from this is stored in mongodb in a collection.

```
{
  "user_id" : "arushi",
  "sensor_catalogue_config" :
  {
    "sensor_type_1" :{
      "sensor_name": "temperature sensors",
      "sensor_type_data_type": "float",
      "sensor_geolocation" :
      {
        "lat" : "None",
        "long" : "None"
      },
      "sensor_address" :
      {
        "area" : "None",
        "building" : "None",
        "floor" : "",
        "room_no" : "None"
      },
      "sensor_data_storage_details":
      {
        "kafka":
        {
          "broker_ip":"None",
          "topic":"None"
        },
        "mongo_db":
        {
          "ip":"None",
          "port":"None",
          "passwd":"None",
          "document_name":"None"
        }
      },
      "has_controller" : "yes",
      "sensor_api":"None"
    },
  },
}
```

## 1.3 APIs Exposed

- **getAllSensorTypes** This will return a JSON with all the details about the sensors types registered till now.
- **addSensorType** This will get the new sensor type details and add them to the catalogue

and persist. This can be done by configurer or admin.

## 2 Sensor Instance Registration

After adding any new type of sensor, we also need to mention the details of its instance to actually onboard it to the platform. This module handles this part of the equation.

### 2.1 User's view

The user (configurer) will be provided with a UI to input the details about the sensor instance, given that the type of the sensor is already present in the catalogue. Any sensor instance from the types present in the catalogue already can be created. If its not present, you need to register it before adding the actual instance.

The screenshot displays the 'SB Admin' dashboard with a sidebar menu on the left containing links to 'Dashboard', 'Schedule Application', 'Sensor Instance Registration', 'User Notifications', and 'Controller Notifications'. The main content area is titled 'Sensor Instance Registration' and includes a breadcrumb trail 'Dashboard / Sensor Instance Registration'. A dropdown menu labeled 'Sensors' is set to 'stitch\_setting'. Below this, there are seven text input fields labeled 'Area', 'Building', 'Floor', 'Room', 'Place ID', 'Latitude', and 'Longitude', each with a placeholder 'Enter text'. A mouse cursor is positioned over the 'Building' field. At the bottom of the form is a button labeled 'Add Sensor Instance'.

- The user has to select one from the registered sensors.
- Fill in the other details and sensor instance is directly registered with the database.
- Multiple instances can be registered.

## 2.2 Internal implementation

The data input is captured through a JSON file and then processed and handled accordingly. The data acquired from this is stored in mongodb in a collection.

```
{
  "user_id" : "apurva",
  "sensor_reg_config" :
  {
    "sensor_1" :{
      "sensor_name": "temperature_sensors",
      "sensor_data_type": "int",
      "sensor_geolocation" :
      {
        "lat" : "167",
        "long" : "196"
      },
      "sensor_address" :
      {
        "area" : "Lapataganj",
        "building" : "Gorisaria and grandsons Garments Group",
        "floor" : "ground",
        "room_no" : "1"
      },
      "place_id" : "GGG_ground_floor_1",
      "sensor_data_storage_details":
      {
        "kafka":
        {
          "broker_ip":"localhost:9092",
          "topic":"sensor_topic_60"
        },
        "mongo_db":
        {
          "ip":"None",
          "port":"None",
          "passwd":"None",
          "document_name":"None"
        }
      },
      "has_controller" : "yes",
      "sensor_api":"None"
    },
    "sensor_2" :{ },
    "sensor_3" :{ },
    "sensor_4" :{ }
  }
}
```

## 2.3 APIs Exposed

- **getAllSensors** This will return a JSON with all the details about the sensors instances registered till now.
- **sensorRegistration** This will get the new sensor details and add them to the db and persist.
- **getPlaceIds** This will be used to get the place IDS from the UI. The place IDS signify the location specified by the user to run the service at.

The binding of sensors with the algorithm part is taken care by the sensor manager. It uses some of the APIs mentioned above to actually track the sensor based on either geolocation or the area and fetch the sensor id and accordingly bind the sensor feed to the algorithm that actually is running and requires it.

## 3 Sensor Manager

### 3.1 Basic Overview

Sensor Manager is in charge of associating the right sensor data stream with the appropriate algorithm. Sensor manager will narrow down the sensors from the sensor database based on the provided parameters like location, etc. Or if the option to use all the sensors of a type is chosen, the sensor manager will engage the sensors accordingly.

### 3.2 Responsibilities of Sensor manager

- Sensor manager module is in charge of linking sensor data to the deployed algorithm and transmitting the data according to the application developer's configuration.
- Sensor manager is also responsible for getting the list of all sensors registered with the platform.
- Sensor manager is in charge of storing sensor data, and transmitting data at the appropriate rate.
- Sensor manager will use the sensor details stored in the platform registry to dynamically construct a temporary Kafka topic for each service requested.

### 3.3 Handling above requirements

- Sensor manager is responsible for getting the list of all sensors registered with the platform. It will either use the place id (locations) to search for relevant sensors, else will return the default sensors.
- When sensors are registered, the corresponding information like kafka\_topics etc are stored in collection "sensors\_registered" and Sensor Manger fetches these details later from the collection.
- Below are the functions sensor manager provides for handling service request
  - While starting a service, the sensor manager collects all the sensor information like its raw data and the topic name where it is exposed and creates a temporary topic and dumps all the data to the temporary topic. The name of the topic is sent back to the deployer and in turn to the application algorithm to listen on it and get the necessary data required. After doing this, the pid of this process is noted along with the serviceid that is unique for an algorithm instance and its logged in the DB. These logs are useful when the module needs to be restarted in the previous state or also to stop servicing a request (i.e dumping sensor raw data to a temporary topic.)
  - To stop the service, Sensor Manager gets the process ID, from the MongoDB database (stored while logging), for the given serviceID and kills that pid. It also changes the state corresponding to the pid from "running" to "stopped" for logging consistency.
  - If the module fails due to some reason, Sensor Manager ensures fault tolerance by re-starting the service. For this, it fetches the process-id corresponding to the service from logging db and restarts serving the request with stored details in the collection : "sensor\_manager\_logger\_current"

### 3.4 APIs Exposed

- **sensorManagerStartService** For each service requested, it will dynamically create a temporary Kafka topic where the data from the sensors required for the service will be available.
- **stopService** It will take a service\_id as input and stop the service corresponding to it.

The input to this Flask API will look like :



```

{
  "username": "divyansh",
  "applicationname": "Application2",
  "servicename": "algo1",
  "serviceid": "111",
  "place_id": "bus_10",
  "config_file": {
    "Application1": {
      "algorithms": {
        "algo1": {
          "code": "control_sewing_service.py",
          "dependency": {

          },
          "environment": {
            "flask": "False",
            "python3-alpine": "False",
            "python3-packages": [

            ]
          }
        },
        "sensors": {
          "senosr_type_1": {
            "sensor_type": "stitch_setting",
            "all_sensors": "no"
          },
          "senosr_type_2": {
            "sensor_type": "thread_remaining",
            "all_sensors": "no"
          },
          "senosr_type_3": {
            "sensor_type": "hook_rotation",
            "all_sensors": "no"
          }
        }
      },
      "algo2": {
      }
    },
    "application_id": "Application1",
    "developer_id": "user_1"
  },
  "Application2": {
  }
}

```

Data in that temporary topic created :

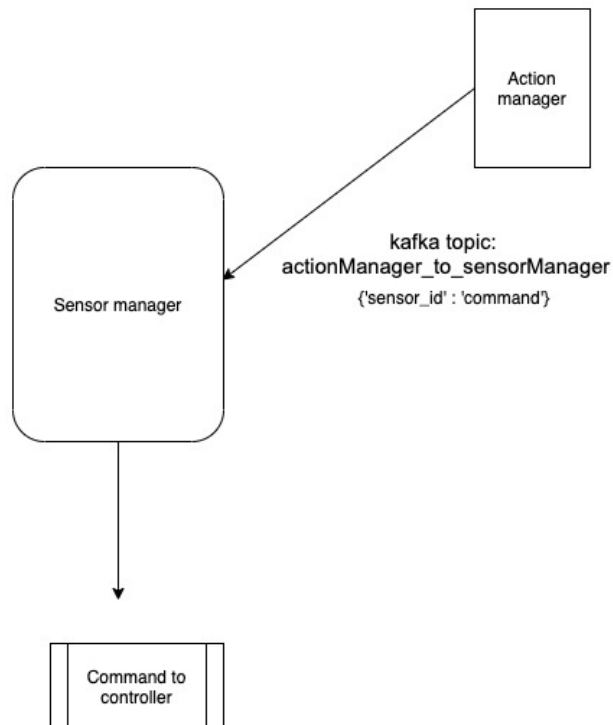
```
Apurvas-Mac:test apurvajadhav$ python3 application_consumer.py
f845f55d-dcee-4100-af8b-42ec84179eb9#10.345523604754359
f845f55d-dcee-4100-af8b-42ec84179eb9#7.404082493987109
f845f55d-dcee-4100-af8b-42ec84179eb9#38.859708510507595
f845f55d-dcee-4100-af8b-42ec84179eb9#20.669506981228608
^CTraceback (most recent call last):
```

### 3.5 Interaction with other modules

- **With action manager**

The action manager will send request to the sensor manager to handle and send back controller commands to the controller. For which the kafka topic : `actionManager_to_sensorManager` is being used. This will be produced by the action manager at the time of initialization and sensor manager will consume the contents published over here.

The data sent over the kafka topic will have the command respective to the sensor id, and this command can be then in turn sent to the controller corresponding to given sensor and manipulate it accordingly.



- **With sensors**

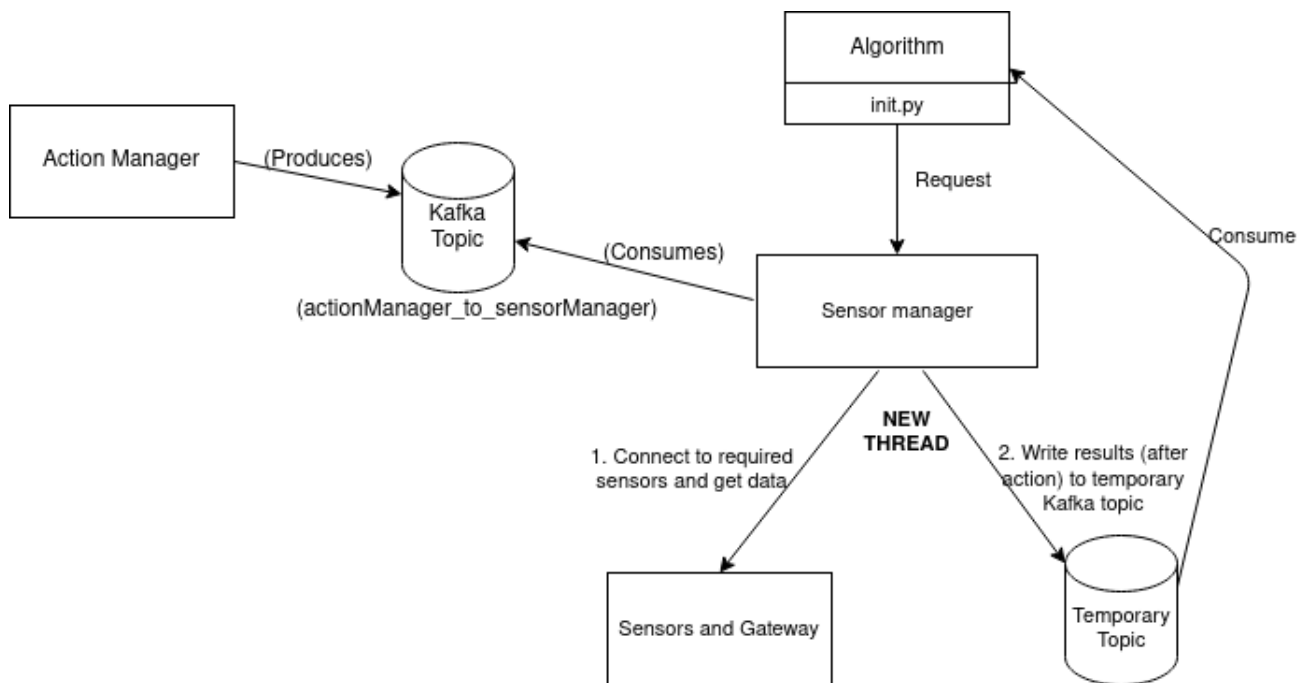
The raw data from the sensor is published over kafka topics which are mentioned at the time of sensor instance registration itself. Hence the sensor manager is able to map any sensor to its respective kafka ip and topic and consume its raw data from that point. This data in turn is then sent to the service that requires it

- **With Deployer (Node)**

Node will request the sensor manager with an API hit. This API will take application config along with details of the service required.

After parsing this JSON, the sensor manager will get the location and the type of the sensors required and map with the nearest sensor\_id available in that location.

Mapping the required type of sensor available in that location, sensor manager will create a temporary topic to publish data of that sensor over to this topic. The temporary\_topic name is sent back in the request and thus the service can consume the data published over this topic.



## 4 Technologies used

- **MongoDB** We use mongodb for data base as the information provided by the sensor configurer is not always complete and there can be many missing fields. Also MongoDB is easily scalable and has auto-sharding.
- **Flask** Flask is used for API creation as it has high flexibility and provides ease to create APIs. The request response is in terms of JSON objects.
- **Kafka** Kafka is a distributed event streaming platform that aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.