

FINAL REPORT TEAM -1

Team Members:

B Sanjana

Yogesh

Shreyas

Shalaka

Aditya C

Meghna Kashyap

Tejas C

Yash Kumar Gupta

Problem Statement: An apartment manager is not sure when there is no water available in the water pump. He needs to get an alert every time water reduces by 10% below a certain threshold.

TRADE STUDY

| Parameters | Piezoresistive | Submersible | Ultrasonic |
|-------------------|-----------------------|--------------------|-------------------|
| Reliability | 4 | 4 | 3 |
| Cost | 3 | 3 | 5 |

| | | | |
|-----------------------|-----------|-----------|-----------|
| Ease of installation | 3 | 4 | 4 |
| Measuring range | 3 | 3 | 1 |
| Weighted score | 40 | 43 | 33 |

| | | | | | |
|--------------------|-----------|-----------|------|------|------|
| Reliability | Excellent | Very good | Good | Fair | Poor |
| Scale | 5 | 4 | 3 | 2 | 1 |

| | | | | | |
|-----------------|-----------|-----------|-----------|------------|-------------|
| Cost(Rs) | 1000-3000 | 3001-6000 | 6001-9000 | 9001-12000 | 12001-15000 |
| Scale | 5 | 4 | 3 | 2 | 1 |

| | | | | | |
|-----------------------------|-----------|------|----------|-----------|----------------|
| Ease of installation | Very easy | Easy | Moderate | Difficult | Very difficult |
| Scale | 5 | 4 | 3 | 2 | 1 |

| | | | | | |
|---------------------------|-------|---------|---------|---------|------|
| Measuring range(m) | >=400 | 300-399 | 200-299 | 100-199 | 0-99 |
|---------------------------|-------|---------|---------|---------|------|

| | | | | | |
|--------------|---|---|---|---|---|
| Scale | 5 | 4 | 3 | 2 | 1 |
|--------------|---|---|---|---|---|

| Factor | Weight |
|----------------------|---------------|
| Reliability | 4 |
| Cost | 2 |
| Ease of installation | 3 |
| Measuring range | 3 |

Solution ranking:

- 1. Submersible pressure sensor**
- 2. Piezoresistive pressure sensor**
- 3. Ultrasonic sensor**

SUBMERSIBLE PRESSURE SENSOR

WORKING:

The submersible pressure sensor, sometimes also referred to as a level probe, is a special type of pressure sensor for hydrostatic level measurement in tanks, wells, shafts and boreholes. For this, the submersible pressure sensor is submerged directly in the liquid to be measured and positioned as close as possible to the bottom.

The measuring cell or the sensor of a submersible pressure transmitter is used for measuring the hydrostatic pressure at the measuring point. There is a diaphragm in the transmitter that works as a pressure sensor. Even under the most severe conditions, in the measurement of media which is dirty, containing particulates or fibres, with abrasive contents or sludge, the measuring cell of a level probe must ensure accurate and reliable measurement. We will put this transmitter into the bottom of a tank and we have a cable that goes up to the surface. The submersible pressure transmitter will send a 4-20 mA signal corresponding to the pressure. To get the information about the pressure we need to know the depth of the tank and the length of the cable.

In principle a submersible pressure transmitter can be divided into five different design elements: the sensor or measuring cell, the electronics, the housing, the cable entry and the cable itself.

Basic Info.

| | |
|---------------------|--------------------------------|
| Model NO. | ASP2002CL |
| Certification | ISO9001 |
| Customized | OEM & ODM Are Accepted |
| Operating Voltage | 2.8V~3.6V |
| Output | Uart/Modbus/RS232 |
| Current Consumption | Standby: Less Than 2ua@3.3VDC; |
| Digital Output | Modbus Communication |
| Electrical Damping | 0.1~16s |
| Measuring Range | 0.1~200m |

Product Introduction

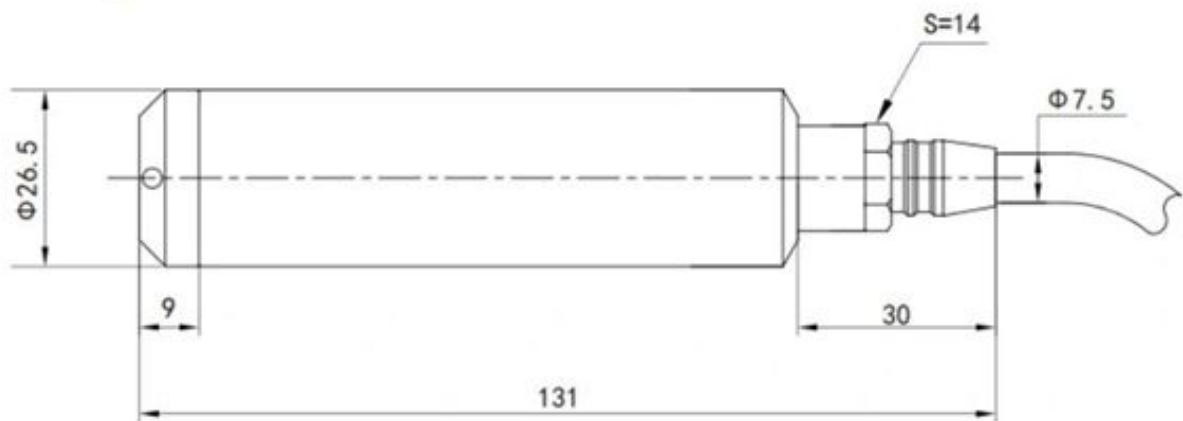
Based on the principle that the measured hydrostatic pressure is proportional to the liquid height, the input type level transmitter uses the electro-resistive effect of diffused silicon or ceramic sensitive elements to convert static pressure into electrical signals. After temperature compensation and linearity correction, convert it into 4-20 mA DC standard current signal output. The sensor part of the input type level transmitter can be directly put into the medium, while the transmitter part can fix by flange and bracket, so it's convenient to install and use.

Measuring range

Lower range limit: -100%URL ($\geq -0.1\% \text{MPa}$)~ +100%URL-Span

Upper range limit: $-100\%URL + \text{Span}$ ($\geq -0.1\%MPa + \text{Span}$) $\sim +100\%URL$

Structural Drawing



ASP2002CL Smart Input type Liquid Level Transmitter outline dimensional drawing (Unit:mm)



DATA SHEET

| Specifications | Technical data |
|--------------------------------|---|
| Input data | |
| Measuring ranges | 0~100kPa ~70 MPa |
| Overload pressures | 150%F. S |
| Burst pressures | 300%F. S |
| Operating voltage | 2.8V ~ 3.6V |
| Output data | |
| Communication | UART (9600 bps; 8N1; Customize) |
| Accuracy | 0.25% |
| Response time | ≤ 300 ms |
| Long term drift | ≤±0.1 % F.S typ. / year |
| Environmental condition | |
| Current consumption | Stand-by: <2 uA@3.3VDC Communication: <1.2 mA@3.3VDC |
| Operating temperature range: | -20 ~ +80 °C |
| Storage temperature range: | -30 ~ +85°C |
| EMC Performance: | EN 61000 - 6 - 2 / 3 / 4 / 6 |
| Vibration: | 10g, 20Hz~2kHz (IEC 60068-2-6) |
| Shock: | 50g, 11ms (IEC 60068-2-27) |
| Free fall: | 1m (IEC 60068-2-32) |
| Insulation resistance: | >100 MΩ (100 VDC) |
| Dielectric strength: | AC1000V/2mA(1min) |

Sending data from Rpi to IoT Hub and storing it in Table storage.

Initially, our task was to send data from **NodeMCU** to the **IoT hub**. Trying numerous methods and codes, we were getting **a lot of errors** but meanwhile. We tried to send data from NodeMCU to MQTT client which was our first successful try to send the data to a server. **Akshay sir** has helped us by sending the data from NodeMCU to IoT hub. He suggested that using NodeMCU things can get much more complex and complicated (especially the code part), he also **suggested us to use** either Raspberry pi or esp32. We had a Raspberry pi simulator and a Raspberry pi 3 hardware. So we sent data from **Raspberry pi** to **IoT hub**, we sent data both from **python IDE** and **Raspberry hardware**. We were able to send data both from Rpi to **Event Hub** and as well as Rpi to **IoT hub**, but as suggested by Akshay sir, that using IoT hub in our case is the **best option**, because event hub is to send a larger amount of data (which we are not sending) also it doesn't lie in the free tier section and it **does not** allow **bi-directional communication** between device and cloud. We sent the data to the IoT hub and stored it in **Table storage**. As we didn't have the **credentials** of the paid version of Microsoft Azure, So, for now, we have sent the data from the IoT hub to **MySQL(localhost)**.

We have completed all our tasks.

A detailed explanation of sending data from python to IoT hub and storing into table storage (*code and a ppt*) is given in this **GitHub repo**: <https://github.com/yovikas1730/Sending-data-to-Azure>

Software/Application used:

- 1)Microsoft Azure
- 2)Raspbian
- 3)Python IDE
- 4)MySQL
- 5)Arduino IDE
- 6)MQTT Box

Azure Resources used :

- 1)IoT Hub
- 2)Event Hub
- 3)Stream Analytics Job
- 4)Storage Account

Hardware used:

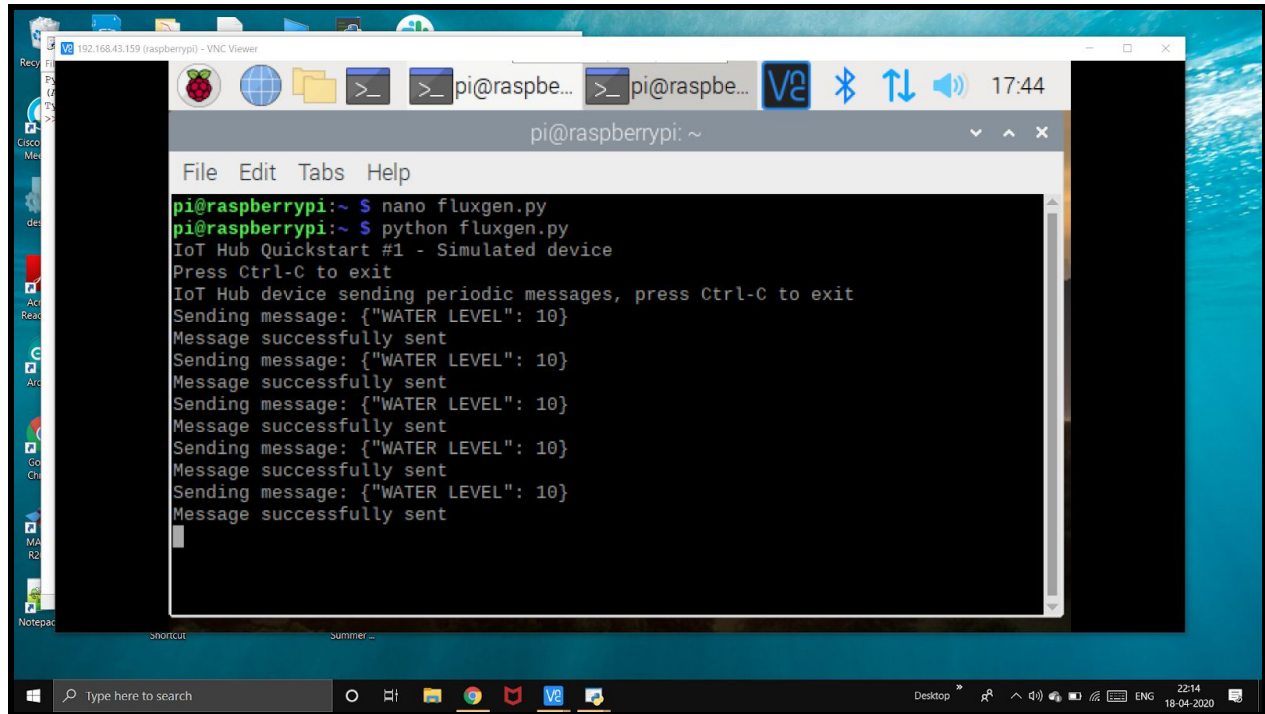
- 1)Raspberry pi 3 B+
- 2)NodeMCU

Future Requirements:

- 1)A paid azure account,so that we can send data from IoT hub to Azure's SQL database.
- 2)We Require Hardware to get actual data from the sensors.

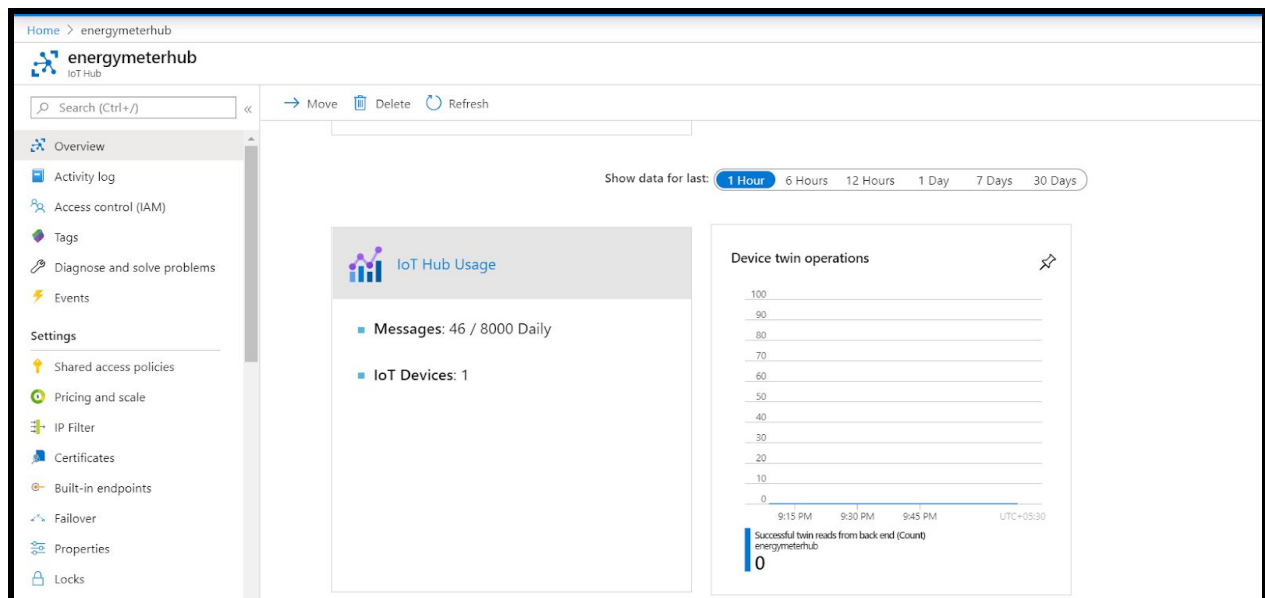
Sending messages from R-Pi

The below picture shows the **water level(dummy data)** being sent to **IoT Hub**. We have used dummy data to send to IoT Hub due to the **lack of hardware** with us.



Getting the Messages on IoT Hub

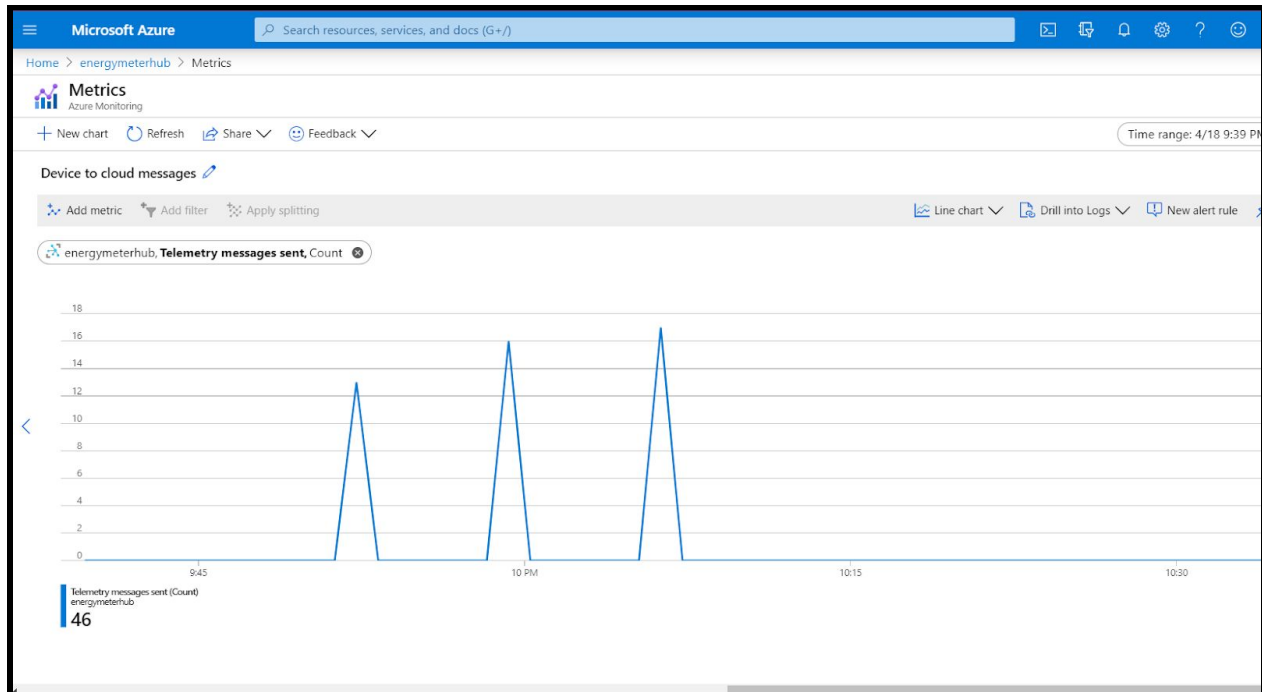
From the above image, we can see the message count being updated in



IoT Hub usage which shows that the messages are being sent to IoT Hub successfully.

Spike in the Messages Graph

From the below image we can see the spike in the messages graph after successfully sending the messages to **IoT Hub**.



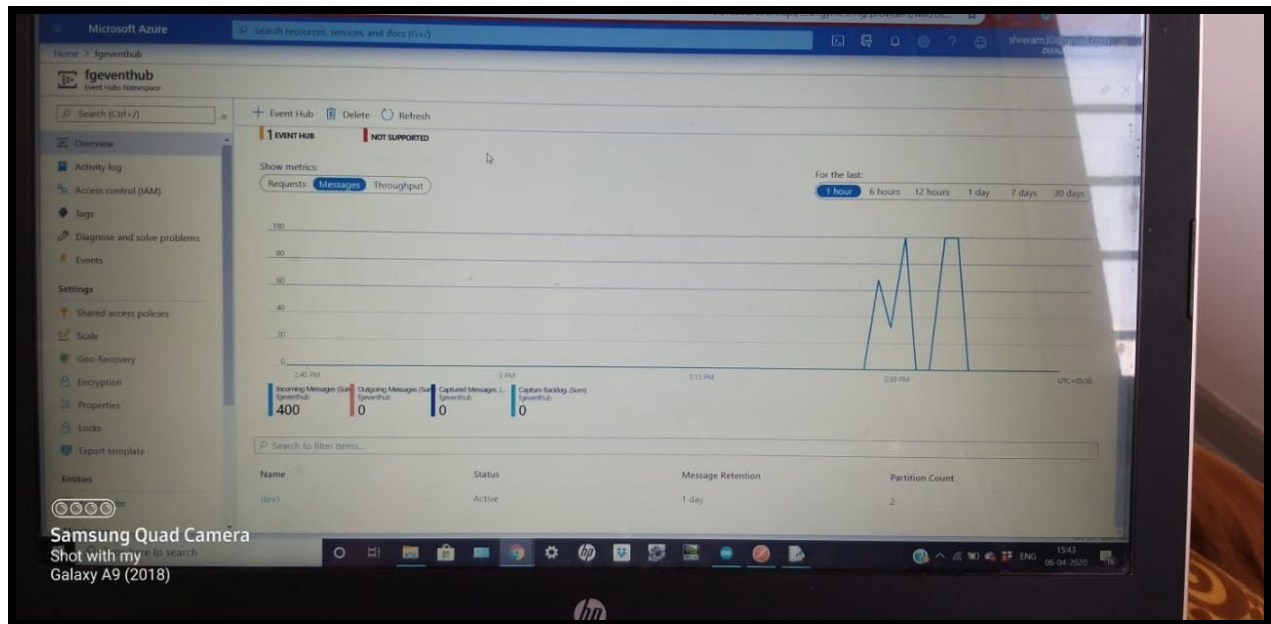
Data stored in Table Storage

In the below image we can see the messages which were sent from **R-Pi** to **IoT Hub** being stored in **Table Storage** using Stream Analytics Job. We can see the water level being stored in the table.

The screenshot shows the Microsoft Azure Storage Explorer interface. The breadcrumb navigation indicates the path: Home > fgstrage | Storage Explorer (preview). The page title is 'fgstrage | Storage Explorer (preview)'. The left sidebar shows the 'Storage Explorer (preview)' view. The main area displays a table named 'table1' with the following columns: TIMESTAMP, EVENTQUEUEDUTCTIME, EVENTPROCESSEDUTCTIME, IOTHUB, PARTITIONID, WATER_LEVEL, and ABC. The table contains 20 rows of data.

| | TIMESTAMP | EVENTQUEUEDUTCTIME | EVENTPROCESSEDUTCTIME | IOTHUB | PARTITIONID | WATER_LEVEL | ABC |
|-----------|------------------------------|--------------------------|------------------------------|--------|-------------|-------------|--------------------------|
| 9100000Z | 2020-04-18T16:35:39.1255073Z | 2020-04-18T16:35:32.91Z | 2020-04-18T16:35:33.0802162Z | Record | 0 | 10 | 2020-04-18T16:35:32.91Z |
| .0250000Z | 2020-04-18T16:35:39.9172658Z | 2020-04-18T16:35:34.025Z | 2020-04-18T16:35:34.1433787Z | Record | 0 | 10 | 2020-04-18T16:35:34.025Z |
| .1810000Z | 2020-04-18T16:35:41.1214207Z | 2020-04-18T16:35:35.181Z | 2020-04-18T16:35:35.238804Z | Record | 0 | 10 | 2020-04-18T16:35:35.181Z |
| .3060000Z | 2020-04-18T16:35:42.3295789Z | 2020-04-18T16:35:36.306Z | 2020-04-18T16:35:36.3375375Z | Record | 0 | 10 | 2020-04-18T16:35:36.306Z |
| .4470000Z | 2020-04-18T16:35:43.5327325Z | 2020-04-18T16:35:37.447Z | 2020-04-18T16:35:37.5373159Z | Record | 0 | 10 | 2020-04-18T16:35:37.447Z |
| .5720000Z | 2020-04-18T16:35:44.7368869Z | 2020-04-18T16:35:38.572Z | 2020-04-18T16:35:38.6316269Z | Record | 0 | 10 | 2020-04-18T16:35:38.572Z |
| .7130000Z | 2020-04-18T16:35:45.9460463Z | 2020-04-18T16:35:39.713Z | 2020-04-18T16:35:39.7911004Z | Record | 0 | 10 | 2020-04-18T16:35:39.713Z |
| .8580000Z | 2020-04-18T16:35:47.1552057Z | 2020-04-18T16:35:40.858Z | 2020-04-18T16:35:40.8879096Z | Record | 0 | 10 | 2020-04-18T16:35:40.858Z |
| .0140000Z | 2020-04-18T16:35:48.3513524Z | 2020-04-18T16:35:42.014Z | 2020-04-18T16:35:42.0965723Z | Record | 0 | 10 | 2020-04-18T16:35:42.014Z |
| .1560000Z | 2020-04-18T16:35:49.5575064Z | 2020-04-18T16:35:43.156Z | 2020-04-18T16:35:43.1909611Z | Record | 0 | 10 | 2020-04-18T16:35:43.156Z |
| .2490000Z | 2020-04-18T16:35:49.5715197Z | 2020-04-18T16:35:44.249Z | 2020-04-18T16:35:44.2853099Z | Record | 0 | 10 | 2020-04-18T16:35:44.249Z |
| .3900000Z | 2020-04-18T16:35:50.7616574Z | 2020-04-18T16:35:45.39Z | 2020-04-18T16:35:45.4915349Z | Record | 0 | 10 | 2020-04-18T16:35:45.39Z |
| .5150000Z | 2020-04-18T16:35:51.9718184Z | 2020-04-18T16:35:46.515Z | 2020-04-18T16:35:46.5910066Z | Record | 0 | 10 | 2020-04-18T16:35:46.515Z |
| .6250000Z | 2020-04-18T16:35:53.1769735Z | 2020-04-18T16:35:47.625Z | 2020-04-18T16:35:47.6789891Z | Record | 0 | 10 | 2020-04-18T16:35:47.625Z |
| .7830000Z | 2020-04-18T16:35:54.379126Z | 2020-04-18T16:35:48.783Z | 2020-04-18T16:35:48.8862339Z | Record | 0 | 10 | 2020-04-18T16:35:48.783Z |
| .9090000Z | 2020-04-18T16:35:55.5862835Z | 2020-04-18T16:35:49.909Z | 2020-04-18T16:35:49.9631929Z | Record | 0 | 10 | 2020-04-18T16:35:49.909Z |
| .9880000Z | 2020-04-18T16:35:56.7924402Z | 2020-04-18T16:35:50.988Z | 2020-04-18T16:35:51.078421Z | Record | 0 | 10 | 2020-04-18T16:35:50.988Z |

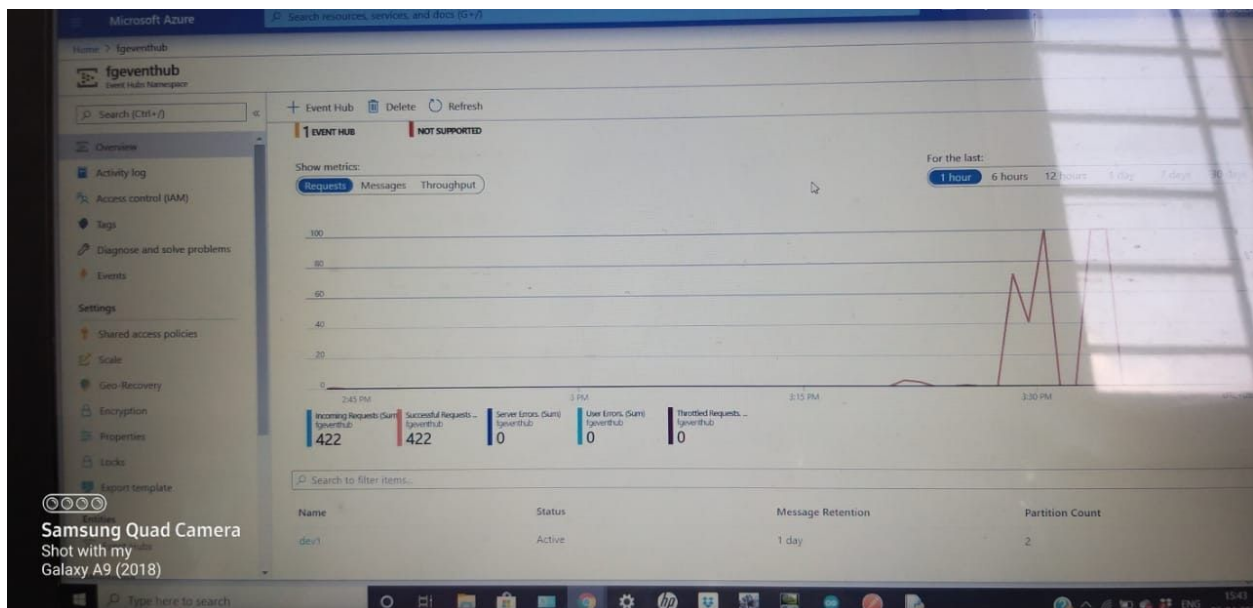
Messages sent to Event Hub



In the above image we can see the spike in the messages graph after successfully sending messages to **Event Hub**.

Requests sent to Event Hub

In the below image we can see that requests were made to **Event Hub** successfully.



Messages stored in MySQL(Local Host)

From the below image we can see the messages being stored in MySQL(**localhost**) as **suggested by Deepak Sir** since we did not have azure credentials to store in the SQL database on **Azure**.

```
mysql> USE fluxgen
Database changed
mysql> select * from WaterLevel
-> ;
+-----+-----+
| Id | WaterLevel |
+-----+-----+
| 1 | 25 |
+-----+-----+
1 row in set (0.00 sec)

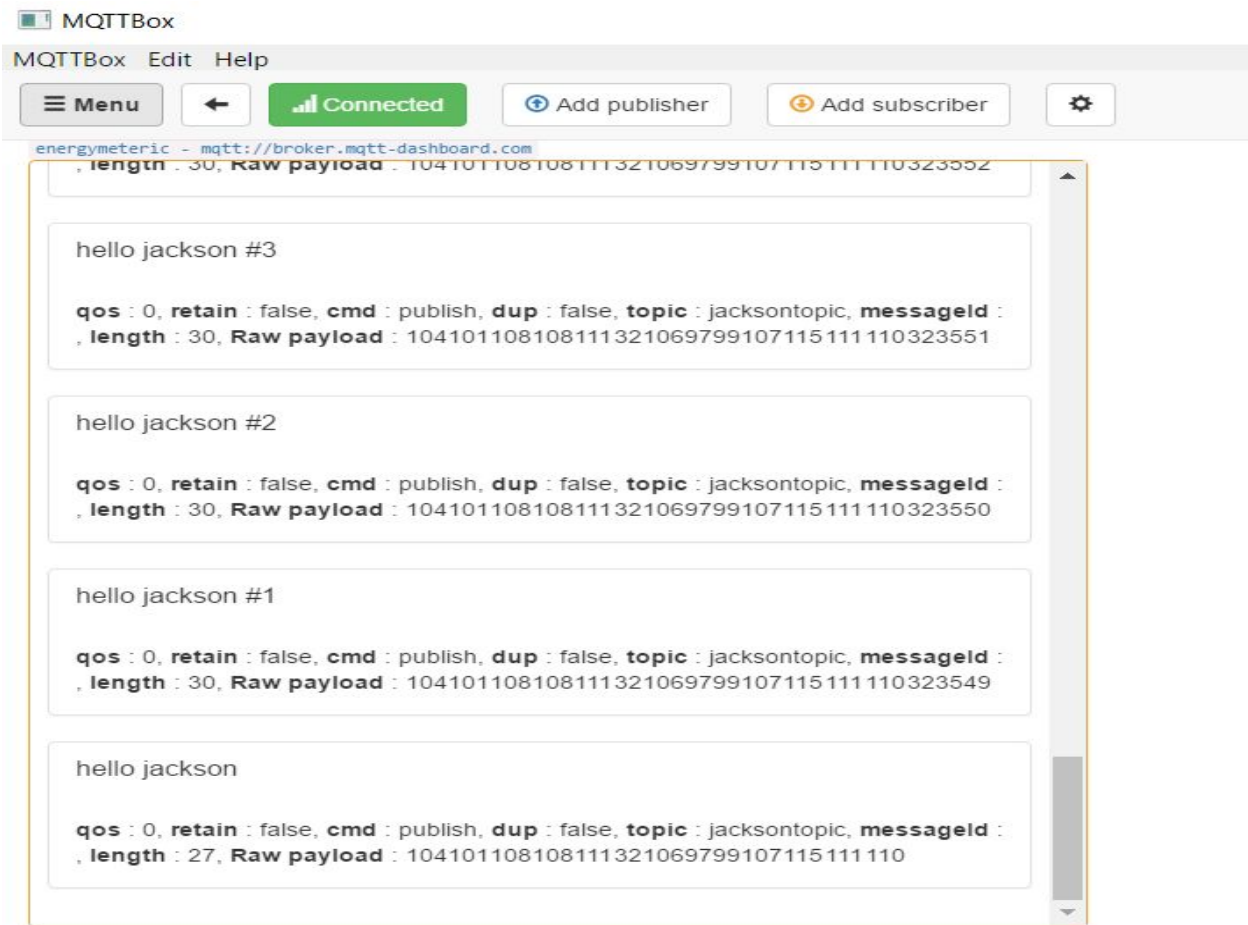
mysql>
```

Sending messages from NodeMCU to MQTT Box using MQTT

The image below is an image of the serial monitor which shows images being sent from NodeMCU to MQTT Box using MQTT.

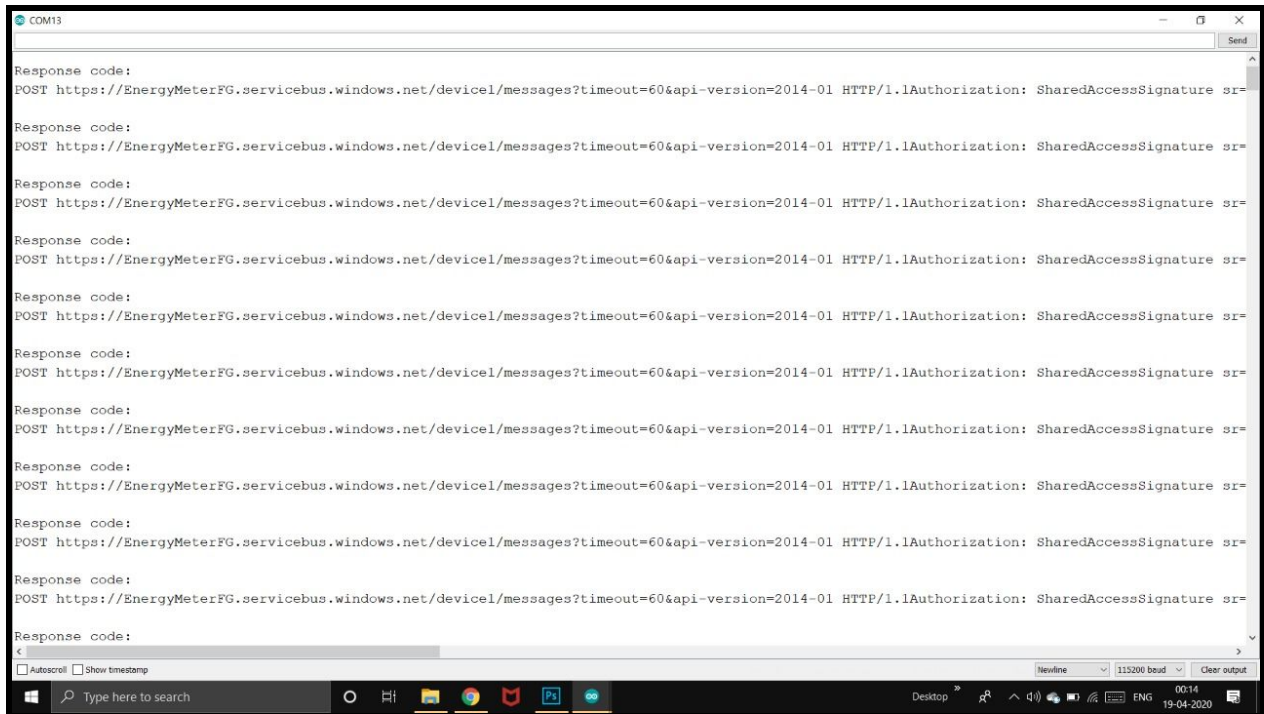
```
COM4
.....
WiFi connected
IP address:
192.168.0.114
Attempting MQTT connection...connected
Publish message: hello jackson #1
Message arrived [inTopic] 1
Publish message: hello jackson #2
Publish message: hello jackson #3
Publish message: hello jackson #4
Publish message: hello jackson #5
Publish message: hello jackson #6
```

Messages sent to MQTT Box



In the above image, we can see messages being displayed in MQTT Box

Sending messages to Event Hub using NodeMCU



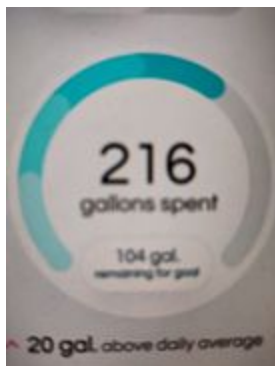
In the above image(image of serial monitor) we can see messages being sent to Event Hub from NodeMCU using Arduino IDE.

Visualization for mobile and web application for the apartment manager to manage the water availability.

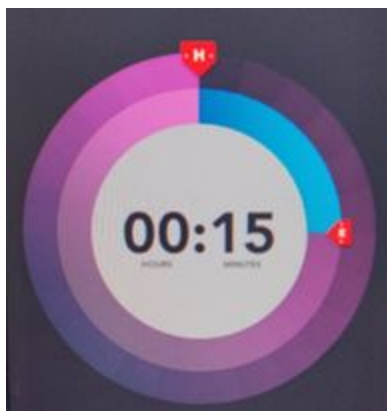
Tools used: **Python IDE**

Initially the task was to come up with a way of **visualizing data** in such a way that all the important information is communicated effectively. After the initial few discussions, we decided that we need to put up the current water level, the amount of water consumed and the estimated time of alert. On discarding the initial few sketches, we agreed on the following two.

1) This is the 1st page where the amount of water consumed is shown as well as the total water remaining in the tank. The blue outer circle shows the portion of the tank filled



2) The next visualisation is a timer, which would show an estimate to when the alert will be sent. The outer circle depicts the current water level.

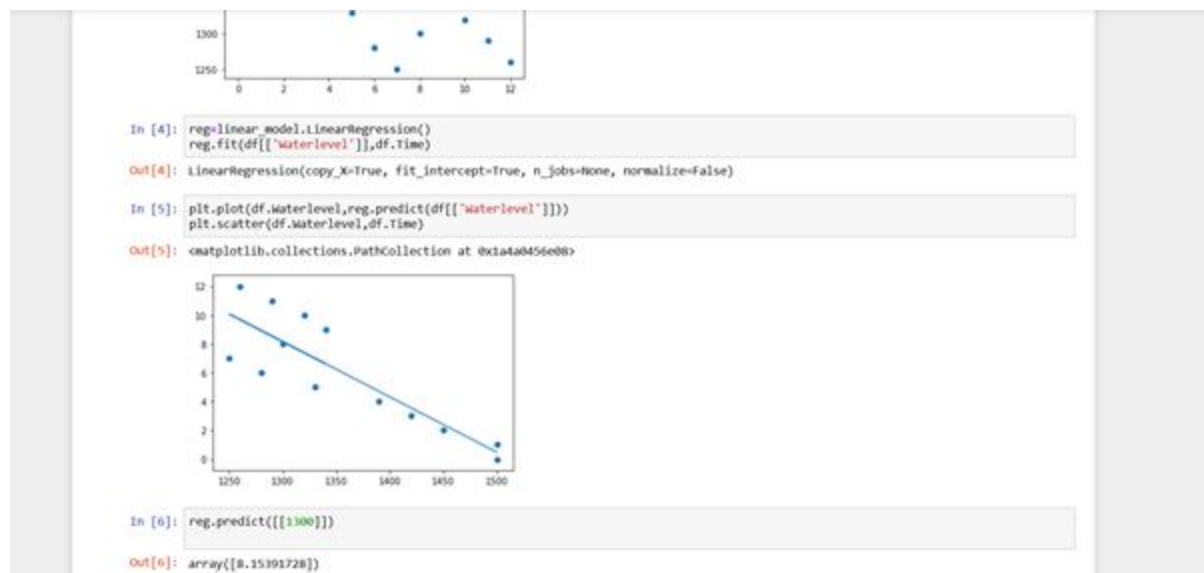


The transition was supposed to be a slide over and an extra Consumption vs Time graph for the Web app.

So the current water level data could be directly accessed but we needed to calculate the estimated time of alert which was not assigned to anybody, so we had a discussion about it and I took it up.

Started with asking a few friends and seniors(Aditya, Harsh and Adithya) about how I can go about this, learned implementing a Regression model could do the job. Went through some codebasics YouTube tutorials and some papers that showed how this could be done.

First, implemented **Linear Regression**.



This gave an output but wasn't accurate. Moreover studying the outcome and the dataset made it clear that linear wasn't appropriate for the purpose.

So next learnt and implemented about **logistic regression**.

```
In [13]: X_train, X_test, y_train, y_test = train_test_split(df[['waterlevel']],df.Time)

In [14]: from sklearn.linear_model import LogisticRegression

In [61]: model = LogisticRegression(max_iter =20)

In [ ]:

In [63]: model.fit(X_train,y_train)

C:\Users\Shalaka-Deshpande\Anacondapython\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
  https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
  https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
Out[63]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=20,
multi_class='auto', n_jobs=None, penalty='l2',
random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
warm_start=False)

In [64]: model.predict([[1228]])

Out[64]: array([12], dtype=int64)

In [69]: plt.plot(train.waterlevel,logmodel.predict(train[['waterlevel']]))
plt.scatter(train.waterlevel,train.Time)
```

This again gave a value but the model was over fitting because the dummy data set used was very small. On requesting Deepak sir for a sample data set, we got a relevant data set.

We learnt to figure out how to use the time on x-axis as the format in which it was given was (date,time). This could be done treating the cell as an array and only using relevant data to create a new column of minutes and using it.

Objectives that are being worked on at the moment.

Understanding the code part of the above given solution and implement:
2-3 days

Mobile application to show the data received by the API to display the data as visually designed

Click to view prototype:

<https://x.thunkable.com/copy/a87042cd806596c196f7f59d0b22f324>

The app is built using thunkable software. Thunkable is a cross-platform app builder that allows anyone to build their own native mobile apps. You can easily design beautiful apps, program powerful functionality with drag & drop blocks and then upload your apps to the Google Play Store and App Store. It is easy to connect thunkable to a database , which is the most important factor

1) UI of the app:

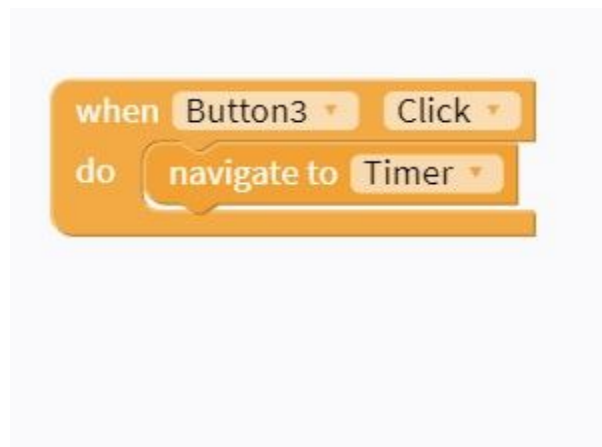
1) The opening page of the app is like a splash screen which will last for 2 secs and will automatically navigate to the next screen. The page and the code block is shown below.



The next page of the app displays the real time water level in the tank of the apartment. It also consists of a button which leads to the next page , i.e. the timer.



The code block for the button is as follows:



The next page is the timer. It displays the amount of time for water to reduce to 10% of the volume of the tank. The page also contains 2 buttons which navigate to the previous page , i.e. waterlevel and the next page too. Here is the code block for navigation between pages pages through the button.



The next page displays the rate of flow water coming out of the tank. This page consists of two buttons again which navigate to the timer and waterlevel pages.

The code block for the buttons is shown below.



2)Database used:

The database used in developing this mobile app is firebase. Firebase is google's mobile platform which helps you develop high quality apps. Firebase provides a free real time database which can be easily connected to thunkable hence we have made use of this in our app.

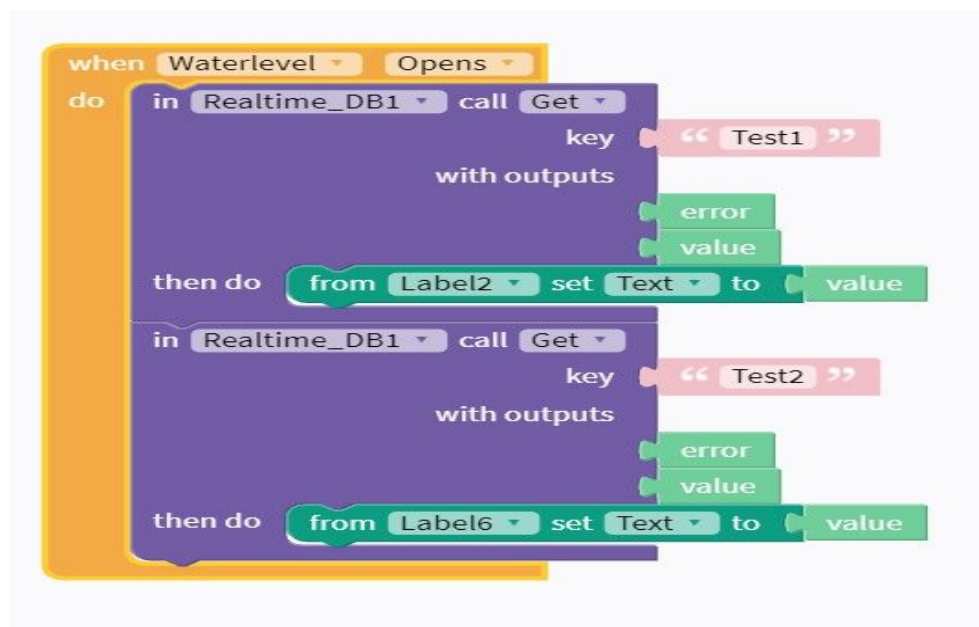
3)Connecting firebase to thunkable:

The first step is to create a firebase account and access its real time database. While doing so we will be provided with an API key and a database URL which must be copied to the thunkable app.

The image shows a 'Firebase Settings' dialog box. It has two input fields. The first is labeled 'API Key :' and contains the text 'AlzaSyCLUNqE9_4BjkmG6JrC'. The second is labeled 'Database URL :' and contains the text 'https://fluxgenproblem1.firebaseio.com/'.

4)Retrieving data from the firebase:

This can be done by dragging and dropping the real time database component in thinkable and modifying the code blocks as follows .



The 'key' should be the same as the key in the database.

Here is a picture of the database key -

 <https://fluxgenproblem1.firebaseio.com/>

fluxgenproblem1

└─ Test1: "ON"
└─ Test2: "3L"

The app is almost complete, the only thing remaining is receiving the actual data from the firebase instead of the dummy data. This will be easily done when the real data is stored in the firebase.

Web App to show the data received by the API from task2 to display the data as visually designed in task 3.

Please find attached the following codes:

- 1) Frontend
- 2) Backend
- 3) Code for inculcating Gauge-meter
- 4) Code for inculcating charts for each apartment. (Sample charts in this folder)

Github link: <https://github.com/meghna-sk/FluxGen-Problem1>

Languages used:

The MERN stack to create this webapp.

Once we obtain Azure credentials, a shift will be made to the SQL database.

React JS is used for creating the Frontend.

Node and Express JS for creating the server.

MongoDB Atlas (on the cloud) is used for the database. This database is temporary until Azure SQL database is ready.

Objectives successfully completed:

- 1) Server and Database are connected and work perfectly.
- 2) Front end is made and is connected to the backend.
- 3) The webapp works as a whole with few minute details to be put up so that it caters better to the problem statement.

Objectives that are being worked on at the moment.

- 1) Inculcating the gauge meter to show the threshold value and water level. Code is ready.

Requirements:

- Need real time data.
Simply need to incorporate code in the main frontend and connect to the database. In this way alert is scheduled.
- 2) Inculcate the rate vs time graph. The code is ready and needs real-time data.
 - 3) Fix tiny errors in the connection between frontend and backend.

The time required to fix all errors and complete web app: less than a week.

How to run the given code:

On terminal for both frontend and backend folders:

```
$npm install
```

```
$npm start
```

Working on the web app

The problem statement states that this web app is mainly used for a manager who is keeping tabs on apartments.

The web app works in such a way that there exists a signup page and a login page for a new manager to be created or an old manager to check on his/her existing apartments.

Sign up and login can be found in the Authenticate Navlink in the Navbar.

Sign up:

The signup page contains the name of the manager, Email ID and Password (minimum 6 characters long). This data is then stored in the database under the 'manager' document.

When a new manager signs up a new manager_id which is unique for the manager is created.

FluxGen: Apartment-Water-Management

All Managers Authenticate

Login Required

Your Name

E-Mail

Password

SIGNUP

SWITCH TO LOGIN

Waiting for localhost...

Login:

The login page requires the manager to give in his/her email ID and password which they used for signing up. This data is checked if it exists in the database or an appropriate error message occurs.

FluxGen: Apartment-Water-Management

All Managers Authenticate

Login Required

E-Mail

Password

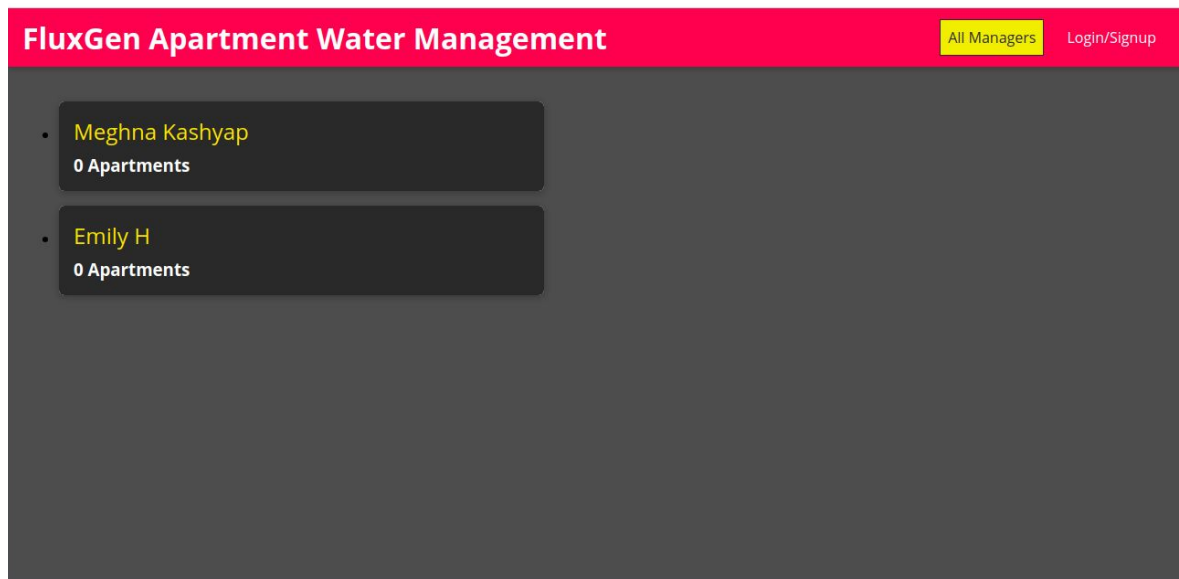
LOGIN

SWITCH TO SIGNUP

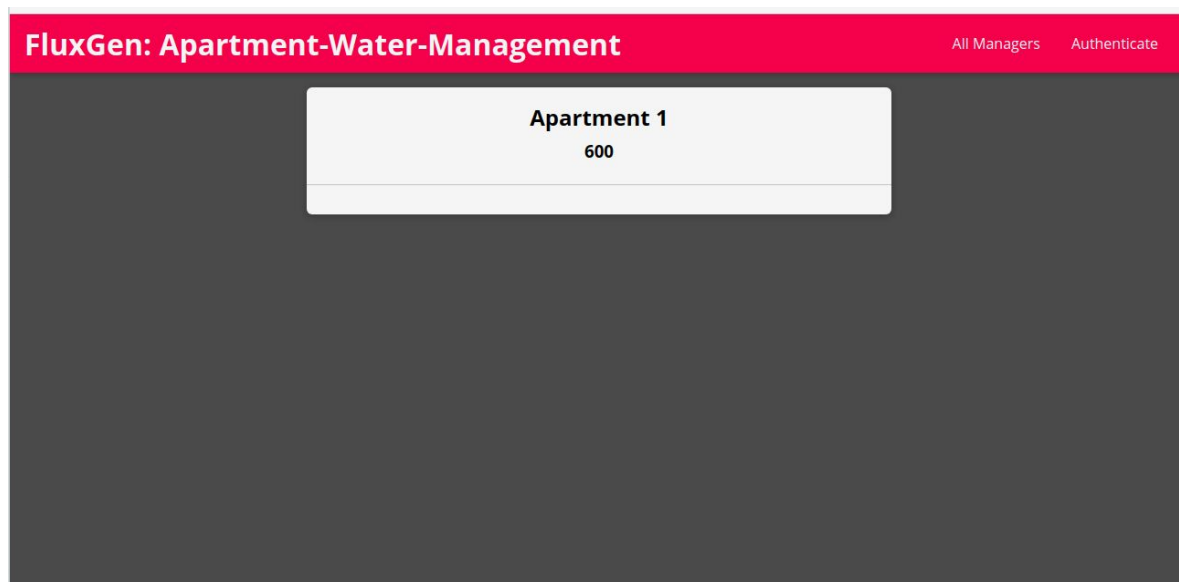
Waiting for localhost...

All Managers

The All Managers tab contains the list of all the managers who have an account. This is shown along with their name and the number of apartments they are seen to manage.



Here we can see two managers with 0 apartments assigned to them.



On clicking the manager whose apartments you wish to see, we get this page.

This page contains the Apartment name and its threshold value for the tank.

Here we have given the dummy threshold value as 600.

Another point to inculcate here is the gauge meter and the rate vs time graph. Each apartment will have its own gauge meter and rate vs time graph and the code needs to simply be incorporated.

The gauge meter to be included:



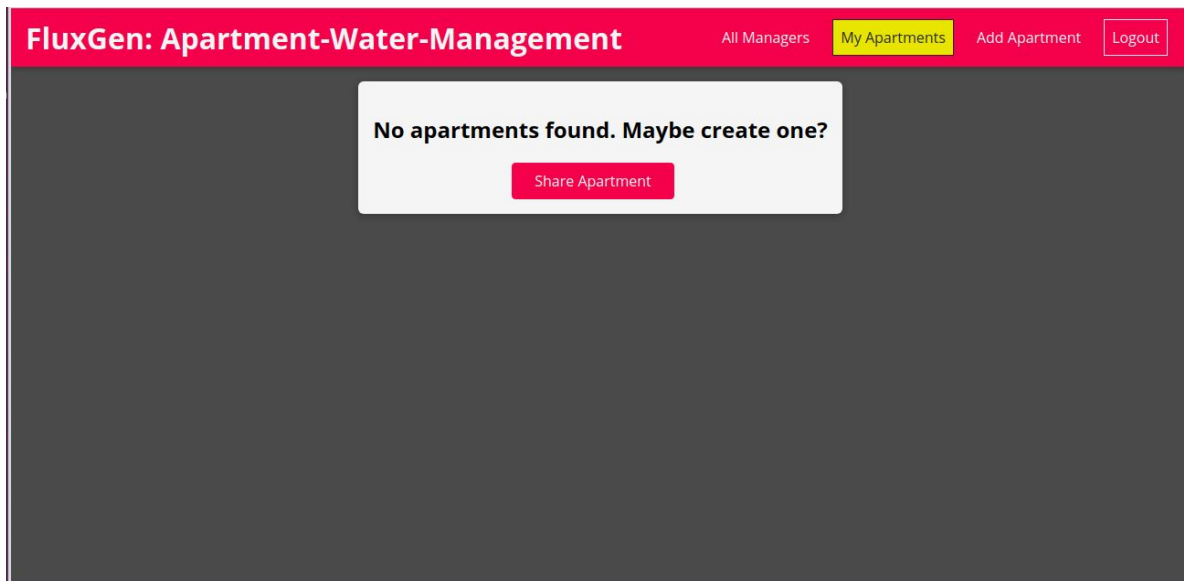
The code for the given gauge meter: (gauge.js)

```
<GaugeChart id="gauge-chart3"
  nrOfLevels={30}
  colors={['#FF5F6D', '#FFC371']}
  arcWidth={0.3}
  percent={0.37} //We get this from the real time database having water level
  alert={0.1* apartment.threshold}
/>
```

The code for the rate vs time graph can be found in the Charts folder.
(sample charts)

The percentage will be converted to how many liters of water are left in the tank. The alert will pop up when the meter shows 10% below the threshold value.

If there are no apartments for the given manager account, the following screen is shown:



Please note the Logout button at the side as well, showing that we are in a manager account right now.

As there are no apartments in the given account, we can click 'Share Apartment' to create a new one.

In order to create a new apartment and add its details, click on 'Add Apartment'.

In both cases, the following screen is shown and here I'm adding an apartment titled 'Apartment 2' with a dummy threshold value of 20.

FluxGen: Apartment-Water-Management

All ManagersMy ApartmentsAdd ApartmentLogout

Title

Apartment 2

Threshold Value

20

ADD APARTMENT

One can update or delete an apartment as well.
Update feature allows updating the title and threshold

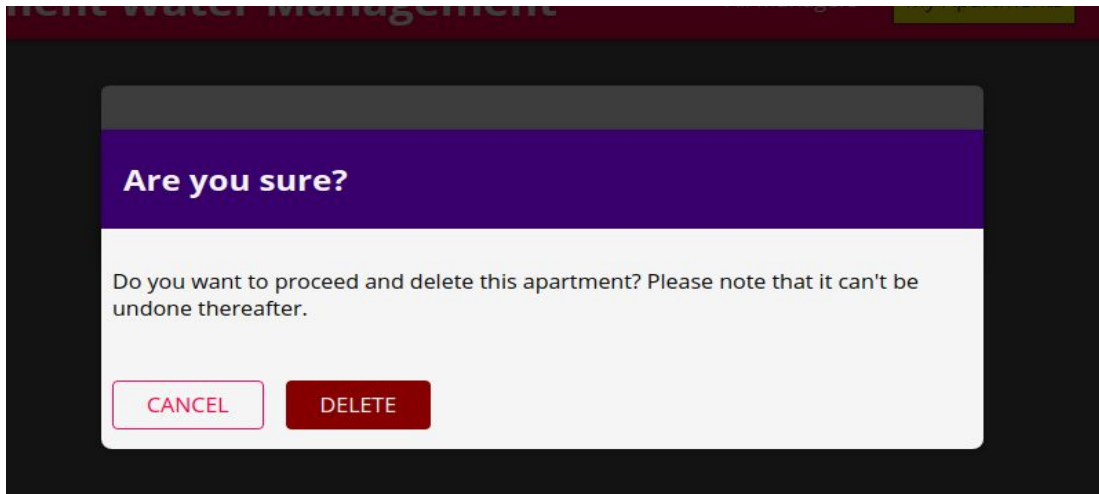
Delete allows you to delete the apartment.

FluxGen Apartment Water Management

All ManagersMy ApartmentsAdd ApartmentLogout

Apartment 5

DELETE



All changes reflect on the database as well.

Backend

A MANAGER CAN MANAGE MULTIPLE APARTMENTS BUT AN APARTMENT CAN HAVE ONLY ONE MANAGER

The schema for both manager and apartment can be found in apartment.js and manager.js

```
const Schema = mongoose.Schema;

const apartmentSchema = new Schema({
  title:{type: String, required: true},
  manager_id:{type: mongoose.Types.ObjectId, required: true, ref: 'Manager'},
  threshold:{type: Number, required: true}
});
```

```
const Schema= mongoose.Schema;

const managerSchema= new Schema({
  name:{type: String, required: true},
  email:{type: String, required: true, unique: true},
  password:{type: String, required: true},
  apartments: [{type: mongoose.Types.ObjectId, required: true, ref: 'Apartment'}]
});
```

Each Manager has certain attributes to himself/herself.

Manager attributes:

name,

email,

password (min 6 characters long),

Apartments (number of apartments he manages)

```
const createdManager= new Manager({
  name,
  email,
  password,
  apartments:[]
});
```

Similarly, each apartment has its own attributes

title,

manager_id

threshold

```
const createdApartment = new Apartment({
  title,
  manager_id,
  threshold
});
```



When a manager is created, a unique manger_id is created or the same. When an apartment is created, a unique apartment id is created as well.

manager_id is mainly used to see how many apartments a manager manages and which apartment is controlled by which manager.

The main backend codes for the working of the web app is in the controllers folder, apartment-controllers.js, and managerscontrollers.js

Error Control

A separate folder is created for error control. A template is made and all different types of errors are raised using this template

```
models > js http-error.js >  HttpError
1  class HttpError extends Error {
2    constructor(message, errorCode) {
3      super(message); // Add a "message" property
4      this.code = errorCode; // Adds a "code" property
5    }
6  }
7
8  module.exports = HttpError;
```

All errors have been taken into consideration. Wrong inputs as well as server errors. The appropriate error message is given for every situation.

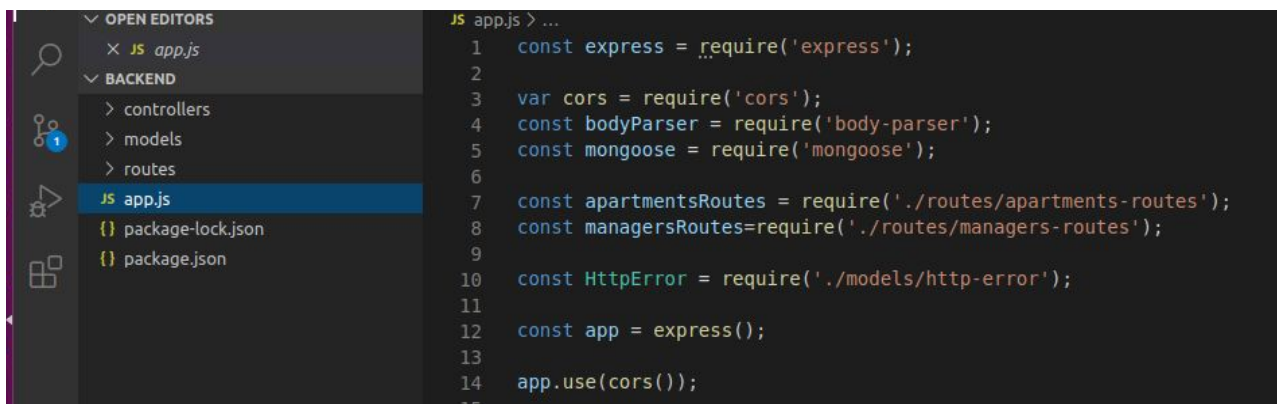
A common error faced while trying to access the backend is of 'CORS'. It's a cross platform error, to resolve it we have to set our response headers as follows.

```

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*')
  res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, Authorization')
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PATCH, DELETE')
  next()
})

```

An easier fix is to npm install cors and implement by the following method:
This code can be found in app.js in the backend folder



```

JS app.js > ...
1  const express = require('express');
2
3  var cors = require('cors');
4  const bodyParser = require('body-parser');
5  const mongoose = require('mongoose');
6
7  const apartmentsRoutes = require('./routes/apartments-routes');
8  const managersRoutes=require('./routes/managers-routes');
9
10 const HttpError = require('./models/http-error');
11
12 const app = express();
13
14 app.use(cors());
15

```

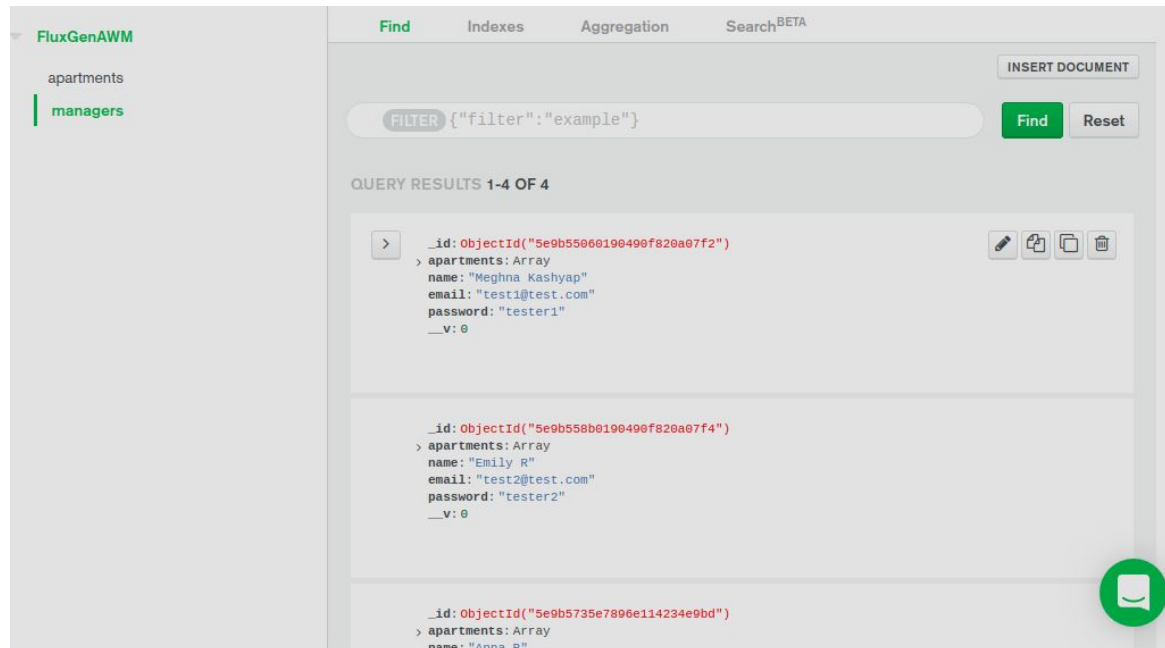
Database:

```

mongoose
  .connect('mongodb+srv://meghnak:12345@apartment-water-management1-3rjlm.azure.mongodb.net/FluxGenAWM?retryWrites=true&w=majority')
  .then(()=>{
    app.listen(5000);
  })

```

This link from mongoDB atlas helps connect the database to the server.
For managers:



```
_id: ObjectId("5e9b55060190490f820a07f2")
> apartments: Array
  name: "Meghna Kashyap"
  email: "test1@test.com"
  password: "tester1"
  __v: 0
```

NOTE: Unique ID for manager is generated in this screenshot.

The given screenshots are from MongoDB Atlas.

For apartments:



NOTE: Unique ID for apartment is generated in this screenshot and the `manager_id` of the manager is shown.

There are two tables here, 'apartments' and 'managers'. When the original database is obtained, real time water level will also be shown on the webapp using the gauge meter.

Rate vs Time graph will also be inculcated using real-time data. (Sample charts to be inculcated in the charts folder)