FOSSEE Summer Internship

Indian Institute Of Technology, Bombay

Generic IoT Platform

**Under the guidance of
Prof. Kannan M Moudgalya,
FOSSEE, IIT Bombay**

Sr. Project Manager

Rajesh Kushalkar

Authors

Nikitha Kondapalli (FOSSEE)
Pradeep B (FOSSEE)
Priyanka Kurkure

Anjali Dhabaria

Meghna Bhave

Mentor

Manas Ranjan Das

Amruta Deshpande

Anuj Khare

PROJECT APPROVAL CERTIFICATE

FOSSEE Summer Internship 2017

The project entitled '**Generic IOT Platform**' submitted by Mr. Anuj Khare, Ms. Anjali Dhabaria, Ms. Priyanka Kurkure, Ms. Meghna Bhave, Ms.Amruta Deshpande, Ms. Nikitha Kondapalli and Mr. Pradeep B. is approved for summer internship 2017 at FOSSEE, IIT Bombay.

_____

Mr. Rajesh Kushalkar
Sr. Project Manager

Place: IIT Bombay
Date: July 4, 2017

Acknowledgement

We would like to express our thanks to all those who helped us in accomplishing our project and achieving the desired outcomes. We would also like to express our sincere thanks to Prof. D. B. Phatak, Prof. Kannan M Moudgalya and Sr.Project Manager Mr. Rajesh Kushalkar for providing us with this opportunity and having faith in our ability. We are thankful to our mentor Mr. Manas Ranjan Das for his constant support and giving us the broader perspective about the project. His constant endeavour and guidance helped us complete our project before deadline and explore new things and also our special thanks to the administrative team of IIT-Bombay for helping and making our working environment comfortable.

Table Of Contents

Introduction

Internet of Things (IoT) is the setup of all the physical devices that we can imagine in our day to day life for collection and transmission of data through various sensors. This makes life easy with minimal human intervention. This project aims at creating a platform for data transmission and reception from various sensors and plotting the graphs on a dashboard developed using javascript. It also includes the process to setup a Smart plug in which a device can be plugged in and controlled using a dashboard application after authentication. The data transmission is secured so that no external user can see the information that is being exchanged between an authenticated client and server.

The data exchange is done using MQTT protocol and ESP8266. ESP8266 is the WiFi module which includes a microcontroller unit that can be programmed using Arduino IDE. ESP8266 connects to the active WiFi connection and supports the data transmission using MQTT. Various libraries have been included in the project and corresponding links have been provided in this documentation.

The project is divided into two sections

- **Hardware section:**

Hardware section involves arranging the setup and making the circuits required for the project. It also includes setting up the ESP8266 module and programming it using Arduino IDE.

- **Software section:**

The Software section includes creating a Web Application for the user using NodeJs and WebViewer in Android Studio. It provides many functionalities to the user like adding a device, controlling it etc.

Purpose

Internet, since its inception has redefined the ways in which the world interacts today, countless gadgets communicate with each other to make our life easy and to make this world a better place to live.

Today the world has turned to a place where each and everything needs to be connected. A generic IoT platform helps us to get the data from various setups and conversely control the devices from distant places using internet. The readings of temperature and humidity sensor can be used to control an air conditioner.

Conservation of resources drives the development of new technologies. A smart plug can help you save and monitor your energy consumption by showing you energy statistics. Moreover it will help you regulate the intensity at which your device operates which will indeed help you save power.

This project is beneficial for budding IoT enthusiasts as it would help them to gain an in-depth understanding of IoT and make a project which they can implement in their daily lives.

Project Flow
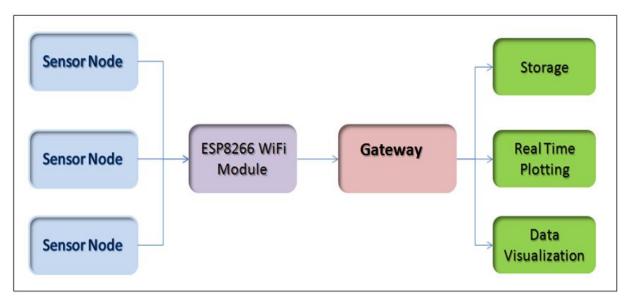
Fig. 1: Generic IoT Platform for Sensors flow diagram
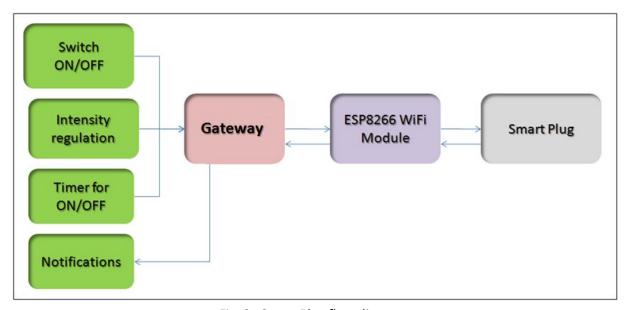
Fig. 2: Smart Plug flow diagram

Hardware Section :

Prerequisites:

Hardware Requirement

Generic IOT platform
1. Sensors
   a. DHT11
   b. ADXL335
   c. Light Intensity Sensor
2. Arduino Uno
3. NodeMCU (ESP8266-12E)
4. Relay
5. Motor
6. Tungsten Bulb

Smart Plug
1. NodeMCU (ESP8266-12E)
2. Transistor (BC547)
3. Solid State Relay
4. Capacitors
5. LED Resistors
6. SPDT
7. Intensity Control Module
   a. Optocoupler 4N35
   b. Resistors: 33kΩ , 22kΩ , 220Ω
   c. Capacitors : 2.2 µF, 220nF
   d. Bridge Rectifier
   e. Diodes : 1N4007, zener(10V, 4W)
   f. AC supply 230V
   g. Lamp
8. AC to 5V DC Buck Converter

Software Requirements

1. **Adding NodeMCU/ESP8266 board to Arduino IDE :**

Step 1: Goto https://github.com/esp8266/Arduino and scroll down to installing with boards manager and copy the .json link (http://arduino.esp8266.com/stable/package_esp8266com_index.json)

Step 2: Open Arduino IDE and goto File->Preferences. In the preferences window paste the link in the Additional Boards Manager URLs.
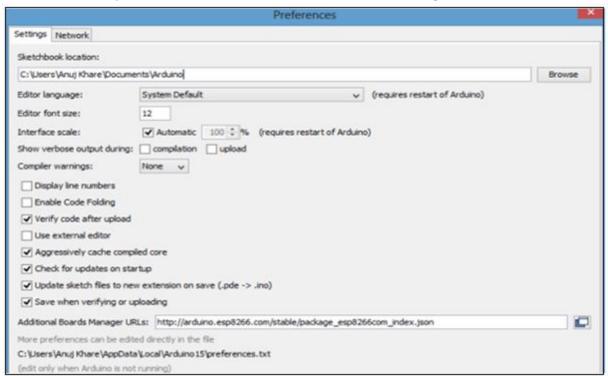
Fig. 3: Adding NodeMCU/ESP8266 board to Arduino IDE

Step 3: Go to **Tools->Board->Boards Manager** and search for ESP8266 and then select the version and install it. After installing move down in the boards list and you will see NodeMCU 1.0(12-E Module).

2. **Installing Mosquitto Broker for MQTT in Ubuntu :**

Mosquitto is an open source broker that implements MQTT protocol.

Step 1: Enter **$ sudo apt-get update** on the terminal and press enter key.

Step 2: Next enter **$ sudo apt-get install mosquitto** on the terminal to install the
mosquitto broker and press enter.

Fig. 4: Installing mosquitto broker on Ubuntu

Step 3: Enter **$ sudo apt-get install mosquito-clients**

Fig. 5: Installing mosquitto client

Step 4: Using Mosquitto
To Subscribe : Terminal 1 :- **$ mosquito_sub -v -t 'check'**
To Publish : Terminal 2 :- **$ mosquito_pub -t 'check' -m 'hello world'**
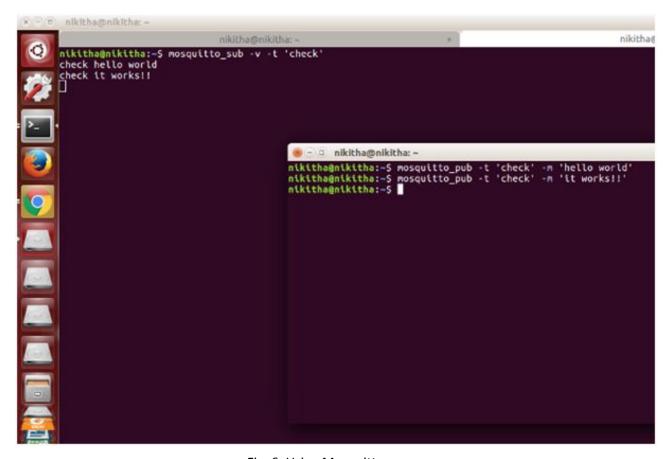
Fig. 6: Using Mosquitto

3. Installing node.js for Node-Red :

Node-Red is built on node.js so to install node-red first you need to install node.js.

Step 1: $ sudo apt- get update

Step 2 : $ sudo apt-get install node.js to install node.js in your system.

Fig. 7: Installing node.js

Step 3: Install npm to install applications related to node red
   Enter **$ sudo apt-get install npm** on your terminal and press enter.

Fig. 8: Installing npm

After installing node.js and npm , now node-red can be installed.

4. **Installing Node-Red**
    Type: **$ sudo npm install --unsafe-perm node-red -g.** Once the Installation is completed, you can now run it by typing **$ node-red** on the terminal.

Fig. 9: Running node-red

Visit the link displayed on the terminal (highlighted above). Generally it is
127.0.0.1:1880

5. **Installing UI/Dashboard in node Red :**

   Once you install node-red it enables you to use buttons, switches etc.
   To install dashboard, go to **'Manage Palette'** by clicking on the drop down
   menu present in the top right corner. Then select node-red-dashboard under
   install option and click on install button. Once installed, you can see nodes
   such as switch, gauge etc. on your palette.

Fig. 10: Node-red with dashboard nodes in palette

About Raspberry Pi and ESP8266

Raspberry Pi:

The Raspberry Pi is a low cost computer and can function as a proper monitor. Raspberry Pi can be used to build smart devices where it acts as a Gateway. Raspberry Pi is usually slow than our normal laptop, but is still a Linux computer. Though Raspberry is slow it is preferred over computers as it's power consumption is quite less. Pi 1, Pi 2,Pi 3 indicate the generation of the model. In our project we are using Raspberry Pi 3 Model.

[1]**Specifications of Raspberry Pi 3 :**
- **Processor :** Broadcom BCM2837
- **CPU Core :** Quad Core ARM Cortex - A53,64 bit
- **RAM :** 1GB with a clock speed of 1.2GHz
- **Power Supply :** 5V,2.5A via a micro USB
- **GPIO :** 40 Pin Header
- **USB Ports :** 4 x USB2.0 with an output current of 1.2A
- **Networking :**
  - 1 x 10/100 Ethernet
  - 802.11n Wireless LAN(WiFi) and Bluetooth 4.1
- **Low Level Peripherals :** UART, I2C bus
- **Ports :** HDMI Port, 3.5mm Audio jack and video composite, Camera Interface(CSI), Display Interface(DSI)
- Micro SD Card Slot

---

[1]http://www.mouser.in/new/raspberry-pi/raspberry-pi-3/?gclid=CjwKEAjws-LKBRDCk9v6_cnBgjISJAADkzXe627xf8eqSuFCUFuYQvrZKOyPuzNPo6amPoyTehkdqxoCSyXw_wcB
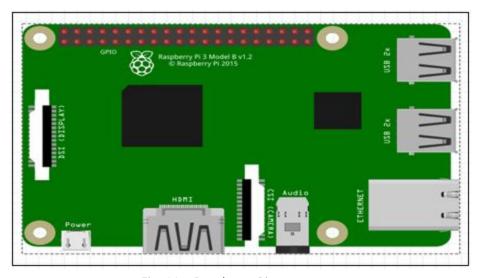
Fig. 11: Raspberry Pi

ESP8266

NodeMCU development board is Open Source IoT platform. NodeMCU is a small board,based on ESP-12E WiFi Module from AI-Thinker,itself containing a single-chip ESP8266 WiFi-SoC. NodeMCU development has ESP12E,a Microcontroller, AMS117. Power Supply to NodeMCU and uploading the programs to Flash Memory of NodeMCU is done via USB Port.

Features of NodeMCU include :

- Acting as WiFi access point (WiFiManager)
- Acting as a client
- Socket connections
- Built-in MQTT
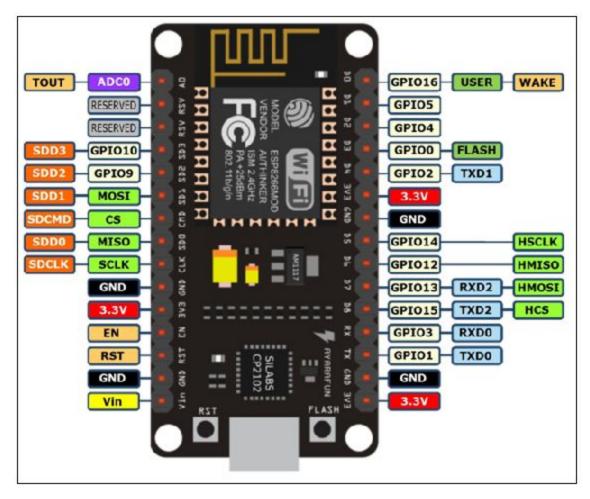
Fig. 12: NodeMCU pin configuration

About Sensors

1. DHT11 : Digital Temperature and Humidity Sensor

   a. OUTPUT: Temperature in degree Celsius and relative humidity in percentage.
   b. Data pin gives a digital output hence it is connected to a digital pin.
   c. Capacitive humidity sensor to measure relative humidity and thermistor to measure temperature of the surroundings.
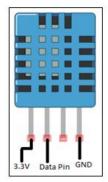
Fig. 13: DHT11 sensor

2. ADXL335 – Accelerometer

   a. Three axis (x,y,z) sensing.
   b. OUTPUT: Signal conditioned voltage outputs for each axis proportional to the acceleration.
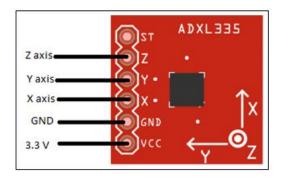
Fig. 14: ADXL335 Sensor

3. Light Intensity Sensor

   a. Voltage proportional to light intensity in the output, use analogRead to read it.
   b. As light intensity increases resistance of LDR decreases therefore voltage on analog pin would also decrease as light intensity increases.
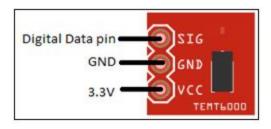
Fig. 15: Light intensity sensor

About RFID

Radio-frequency identification (*RFID*) uses electromagnetic fields to automatically identify and track tags attached to objects. The tags contain electronically stored information. The tags are further classified into two types.

a. **Passive Tags**: Operate from short distances as they collect energy from a nearby RFID reader interrogating radio waves.

b. **Active Tags:** They have a local power source such as a battery and may operate at hundreds of meters from the RFID reader.

RFID can be used for security purposes. It can be implemented such that a user having valid tag can only access a device. RFID is better than other scanning mechanisms due to following reasons:

1. The tag need not be on the surface of the object.
2. The read time is typically less than 100 milliseconds.
3. Large numbers of tags can be read at once.
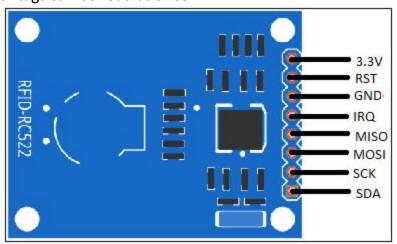
Fig. 16: RFID MFRC522 Reader

MQTT Protocol

MQTT (Message Queue Telemetry Transport) is a lightweight publish subscribe protocol for IoT based application. MQTT is preferred over HTTP protocol because HTTP protocol requires handshaking for every request whereas, on other hand, MQTT is a simple protocol where data gets published and subscriber receives it by subscribing to that particular topic. Also in HTTP the client has to pull the information from the server, conversely, in MQTT, broker pushes the information to the client. Main elements required for MQTT protocol implementation are :

a. Publisher
b. Broker
c. Subscriber
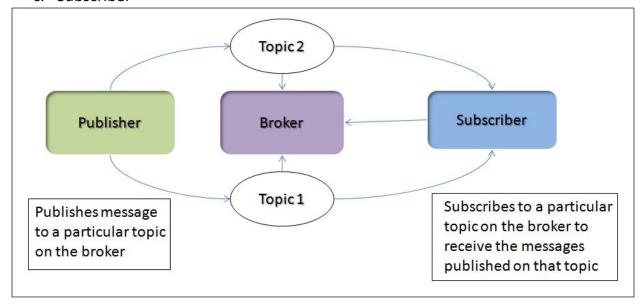
Fig. 17:   MQTT protocol

When a client is connected to the broker, it can publish messages which are filtered by the topic. A client publishes a message to a particular topic and the broker pushes the message to the client who is interested in receiving the data published on that topic. The publisher as well as the subscriber must have the same topic in order to communicate with each other. MQTT makes it easy for a client to publish and allows multiple clients to access the information by subscribing to a particular topic.
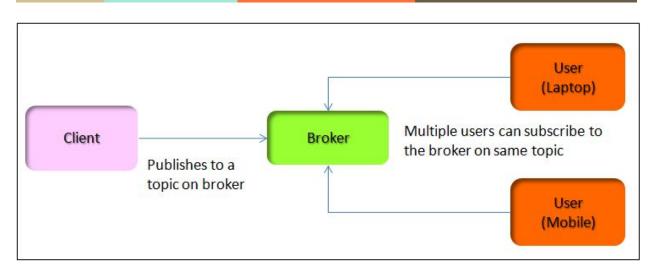
Fig. 18: MQTT - Publish and Subscribe

Important note :
- While using two or more ESPs, MQTT client ID for each ESP should be unique.

Encryption and Decryption

It is important to secure the messages that are being sent and received at application level. Since we are using MQTT protocol for communication between a client and server, two methods can be used to secure the payload. They are as follows:
1. SSL / TLS (Network layer Security)
2. Payload Encryption and Decryption (Application layer Security)

Since our project is application based, we chose encryption and decryption to secure the payload. Encryption-Decryption follows a simple algorithm. In order to avoid a third party from viewing the messages that you are publishing to a particular topic, we need to hide the messages or in other way, display them such that it makes no sense to the third party and only the authorized user can decrypt and get the correct message. This doesn't allow the third party to view or change the messages.

To encrypt a message

1. A randomly generated integer (called the **random key**) is added to each character of the payload string. This random key is actually added to the ASCII value corresponding to that payload character. The character corresponding to this sum (in ASCII table) is stored in a character array. This character array is now the encrypted payload.
2. The random key is also sent with payload to decrypt the message at other end. To do so, a constant key (called the **private key**) which is known only to the two parties authorized to communicate is used. This secret key is called private key and is different for different setups.
3. The random key generated in first step is encrypted using the private key (using the same algorithm used to encrypt the payload) .
4. The encrypted random key and encrypted payload strings are then concatenated with a special character separating them. This helps the decryptor to identify the random key and payload separation. Eg. : "Encrypted random key" +"/"+"Encrypted payload"
5. This encrypted string is now secure and can be published to the required topic.

Message Format

There are some limitations on the messages that can be sent by encrypting them with the above scheme. These are:

1. Messages should have characters below the ASCII value of 91(including 91).
2. Extended ASCII is not supported. All encrypted messages will also have characters of normal ASCII.
3. Private and random keys can be any number between 1 to 36.
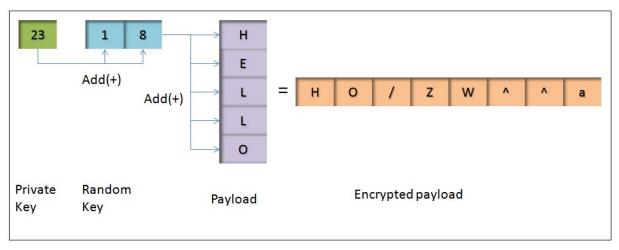
Fig. 19: Encrypted message example

To decrypt a message

1. Scan the encrypted message till the special character separating the encrypted random key and encrypted payload (in the example, "/"). Subtract the private key from ASCII value of all the characters of encrypted message until the special character ("/ "). The difference gives the ASCII value corresponding to the random key.
2. Convert the character array corresponding to this difference into an integer. This integer is now the random key.
3. The decrypted random key is now used to decrypt the encrypted payload . This is done by subtracting the random key from each character of the encrypted payload . The characters corresponding to this difference (ASCII table) give you the decrypted payload.
4. The decrypted payload can be further used in application by the decrypting party.
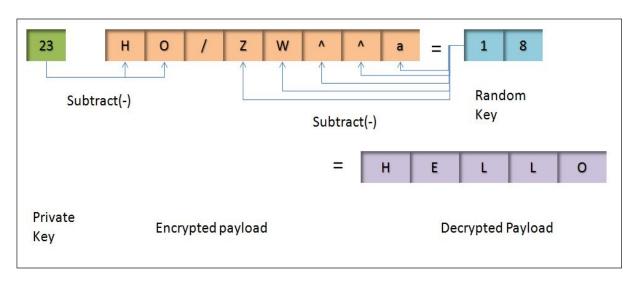
Fig. 20: Decrypted message example

Generic IoT Platform for Sensors

In today's IoT applications fundamentally focus is on collecting data from the sensors and performing analysis.

Schematic for NodeMCU and sensors
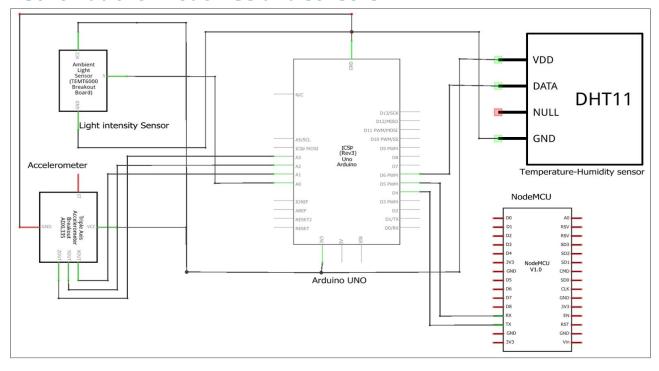
Fig. 21: Schematic for sensors setup

Algorithm for Generic IoT Platform

Upload the codes to Arduino-UNO and NodeMCU.

Data Received by Arduino:

1. Temperature and Humidity - by functions from DHT Library

2. Light Intensity, Accelerometer - AnalogRead and then mapping between 0 and 255 from 0 to 1024.

Serial communication between Arduino and NodeMCU starts and data is sent continuously using Serial Communication

NodeMCU – connected to Wi-Fi and Mosquitto Broker (RPi)

Data Received by NodeMCU is Encrypted

The Encrypted data is published by client to the broker and the other client subscribes the data from the broker.

Visualization of Data on Dashboard.

Fig. 22 : Flow for sensors

Flow in Node-Red
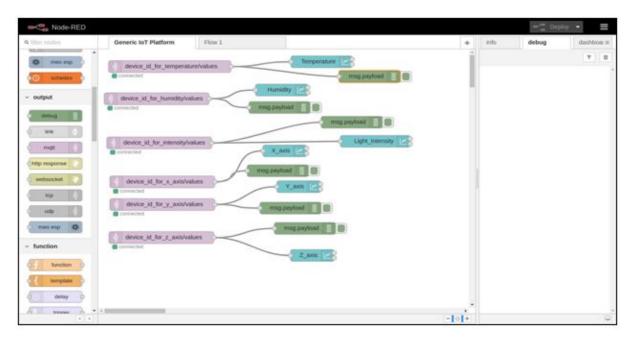
Fig. 23: Node Red flow for sensors

Results obtained on Node-Red
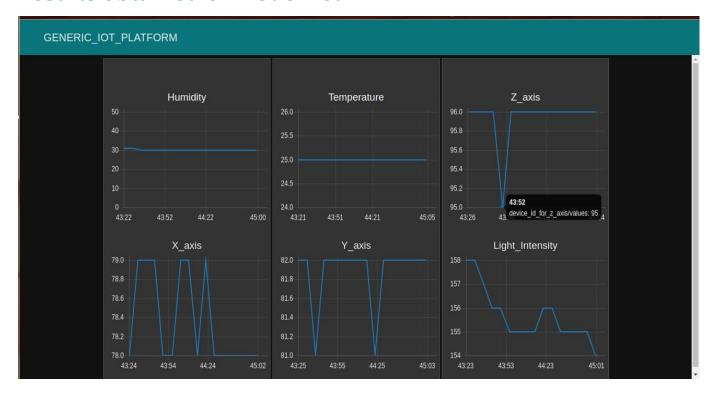
Fig. 24: Results obtained on node-red Dashboard

Explanation

Consider the case of collecting data from Light Intensity Sensor.

- Declare pinMode of A0 to be INPUT using the command in the setup function. pinMode(A0,INPUT);
- Declaring of the functions for reading and sending the data

```
This function reads the Light Intensity using analogRead and maps between 0 and 255

        void intensity(){
        /*
        *   Read data from Analog Pin and map them from 0 to 255
        */
        inten=map(analogRead(A0),0,1024,0,255);
        Serial.println("Light Intensity is:");
        Serial.println(inten);
        }

This function sends the data to NodeMCU using Serial Communication

        void writedata(){
            serial.write(temp);
            serial.write(hum);
            serial.write(inten);
            serial.write(x);
            serial.write(y);
            serial.write(z);
        }
```

Fig. 25: Function declaration

- Continuously sending the data to NodeMCU and then publishing from NodeMCU.

```
void loop()
{
    dht1();
    intensity();
    acc();
    writedata();
}
```

Fig. 26: Calling sensor functions

```
while(Serial.available())
    {
        x=(Serial.read());
        snprintf (msg, 75, "%d",x);

        if(i ==0)
        {
            encryptMessage(msg,"594a0139ad19f00b086a7b73/values");
        }
```

Fig. 27: Data transmission

RFID and Servo Motor

Schematic for RFID and Servo Motor :
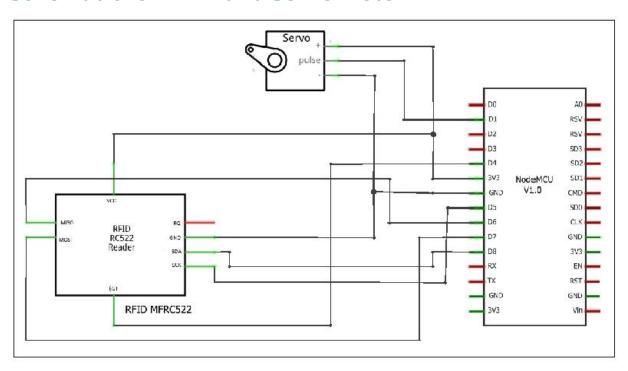
Fig. 28: RFID Servo Schematic

Algorithm for RFID and Servo
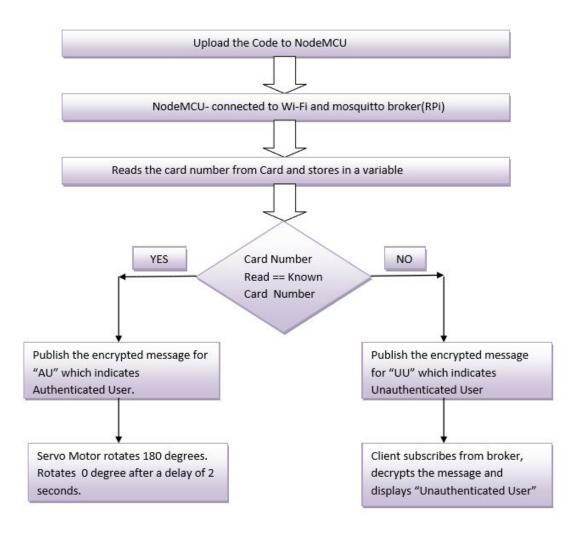
Fig. 29 : Flow for RFID and ServoMotor

Smart Plug

This WiFi Smart Plug allows to switch devices on/off with just one touch via IoT.
It is a great alternative for making smart homes and even Industrial Iot Projects. It has it's own firmware and mobile app. It is based on ESP8266, making it compatible with arduino and easy to use and develop. And It It can work as a normal switch too.

A. ON/OFF Devices (~ 100W) from
    i. Mobile App
    ii. DashBoard
    iii. Manually
B. Regulation -
    i. Speed
    ii. Intensity
C. Set a timer for turning the device ON/OFF.
D. Energy Consumption
E. Indicates when ESP is
    i. Connected
    ii. Connecting
    iii. Disconnected
F. Allows to Re-configure ESP

Schematic for Smart Plug
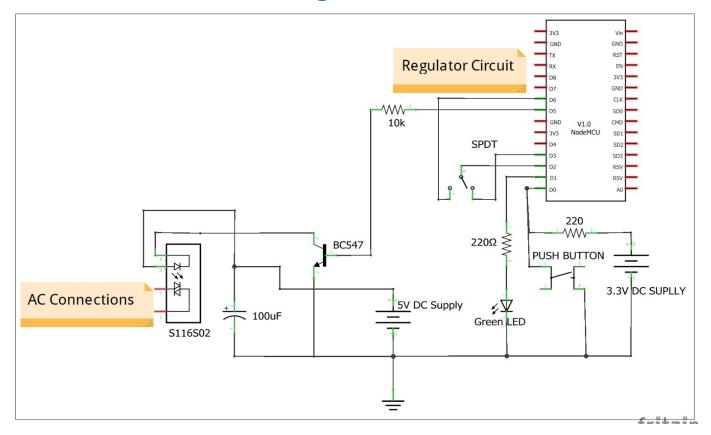
Fig. 30: Smart Plug Schematic
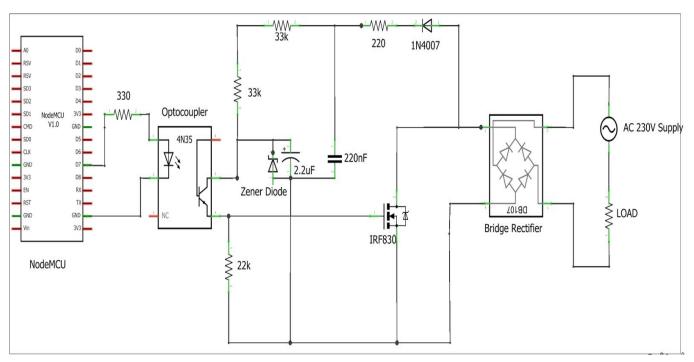
Fig. 31: Schematic for intensity regulation

Algorithm for Smart Plug:

Connect ESP to the WiFi : Using WifiManager library configure it to join any available network.

Define Pin Modes for ESP Pins that will hold LED, SPDT Switch, Pushbutton, Solid State Relay and Intensity Regulator Module.

LED indicate ESP state DISCONNECTED, CONNECTING (or) CONNECTED.

SPDT Switch to manually turn ON/OFF the smart Plug

Pushbutton to reset Settings and re-configure ESP

Solid State Relay to turn ON/OFF the smart Plug

Intensity Regulator Module to regulation

Connect ESP to Mosquito Broker (here Raspberry Pi)

00

Once both connections hold true (WiFi and Mqtt) then communication between ESP(Smart Plug) and Gateway begins

MQTT :

PUBLISH
- Notifications such as Device already On/Off
- Intensity
- Energy consumed by the device

SUBSCRIBE
- ON/OFF
- Intensity
- Time after which device turn on/off

Take action on the subscribed topic values such as turn the device on/off at a particular intensity and also initiate a timer based on the timer value subscribed

The published values are encrypted. Similarly, the subscribed values are decrypted. Hence the communication is secure.

Fig. 32:Flow for SmartPlug

Flow in Node RED
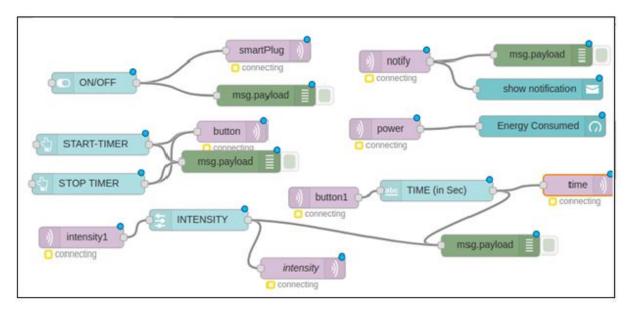
Fig. 33: Smart Plug Flow in Node Red

Node-Red Dashboard
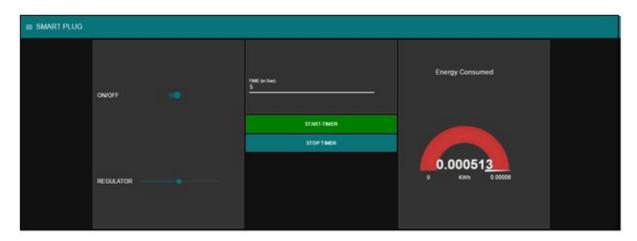
Fig. 34: Dashboard on Node-red for Smart Plug

Software Section

Overview

The project aims at providing an **open source web and mobile platform** for all IoT based data analysis. The platform allows users to send data to the platform for **real time data visualisation**. Users can also **control the devices** through the platform. **Power consumption** of every device can also be viewed through this platform.

Real-time Data Streaming:

- The user can view the current reading of the sensors through the gauge chart.
- Time vs Value data of the device is plotted through line chart in real time.
- The user can see day-to-day analysis of the device readings through a line chart.
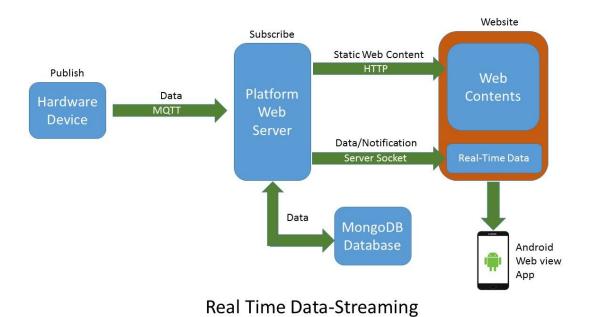
Fig. 35: Data flow

Controlling of the Device:

- The user can **turn ON/OFF** the device through a toggle switch.
- The user can **schedule a timer** for turning ON/OFF the device depending upon his requirements.
- The user can also **change the intensity** of the device through intensity slider.

Controlling the Device
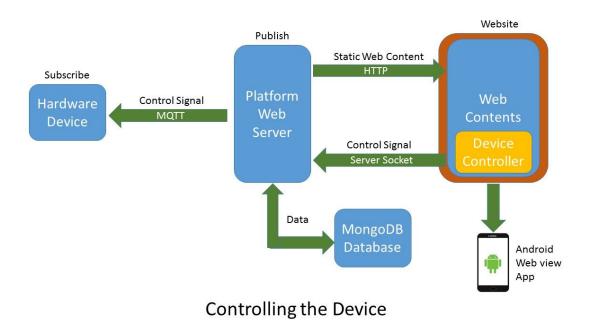
Fig. 36: Controlling of devices

Energy Consumption:

- The user can view the **energy consumed** by the device in a day

Project Creation

- The user can also create Projects and **aggregate different devices together** that have similar functionalities associated with the project.

Installation

Pre Requisites:

Node.js : https://nodejs.org/en/ version 6 and above.

MongoDB: https://www.mongodb.com/what-is-mongodb

NODE-RED: https://nodered.org/

Operating System: Ubuntu : 16.04 and above.

Steps to Install:

1. Download/ Clone the repository in the folder

    git clone <link>

2. To install all dependencies, go to the folder the repository is in and run

    npm install

3. To start mongodb, open a new terminal and run the command

    sudo mongod

    (If it doesn't work, try running 'sudo service mongod stop' and run the command again)

4. To run the server, use the command

    node app

5. The website will run on http://localhost:3000

6. To test the device visuals, open another tab and run :
    a. **Through HTTP GET Request:**
        http://localhost:3000/api/users?device_id=<device_id>&value=<value>
    b. **Through MQTT Protocol:**
        i. Open a new terminal and type

```
mqtt pub -t <device_id>/values -h <host_address> -m <value>
```

where <u>device_id </u>is the device id of the device where you are plotting.

<u>value </u>is the value you want to plot.

<u>host_address</u> is the IP address where the MQTT server is hosted.

Goals

V2.0 has the added functionality :

1. Allowing the user to **add various devices** based on his requirements.

2. Allowing the user to **create Projects** and add multiple devices under each project.

3. **Real time data streaming** on gauge and line charts.

4. **Real time energy analysis** on gauge graph.

5. **Date to date analysis** of device data based on starting and ending date set by the user on a line chart.

6. **Controlling remote devices** from the dashboard. (Activating or deactivating devices through the switch or by setting the timer).

7. Controlling the intensity of the devices from the dashboard.

8. Creating a mobile application encompassing all the features of the web application.

V3.0 will have the added functionality :

1. Adding data analytics techniques.
2. More sophisticated encryption/decryption protocol.
3. Login via facebook, twitter, and google.

Specifications

Technology Used

1. Bootstrap
2. Mongo DB
3. Canvas Gauges
4. JQuery: 3.1.1
5. Node JS
   - bcrypt-nodejs: 0.0.3,
   - body-parser": "^1.17.2",
   - canvas-gauges: ^2.1.4,
   - chart.js: ^2.5.0,
   - chartjs: ^0.3.24,
   - connect-flash: ^0.1.1,
   - cookie-parser:^1.4.3,
   - cookieparser: ^0.1.0,
   - email-verification: ^0.4.6,
   - epoch-charting: ^0.8.4,
   - express: ^4.15.2,
   - express-session: ^1.15.2,
   - format: ^0.2.2,
   - logger: 0.0.1,
   - mongoose :^4.9.8,
   - morgan: ^1.8.1,
   - mqtt: ^2.8.2,
   - nodemailer: ^4.0.1,
   - passport: ^0.3.2,
   - passport-local: ^1.0.0,
   - sleep: ^5.1.1,
   - socket.io: ^2.0.1,
   - ejs: ^2.5.6

6. Node-RED
7. Mosquitto

NodeJS

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. It is an open source server framework that allows us to use javascript on the server.Node.js runs the script server side allowing the user to create dynamic web pages.It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node version should be 6 and above.

For more information,visit:https://nodejs.org/
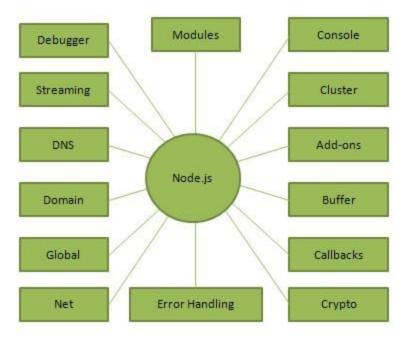
Fig. 37: Functionalities of node.js

MongoDB

MongoDB is a document based, NoSQL , free and open-source, published under a combination of the <u>GNU Affero General Public License</u> and the <u>Apache License</u>.database written in C++.MongoDb is based on documents instead of tables which provides it an added flexibility. In Mongodb one collection holds different documents whose number of fields, size etc. can vary.

The data in the database is stored in the form of JSON files.

Following are the basic commands to get you started on MongoDB

To use the database

    use platform2

To see the collections

    db.<collection_name>.find().pretty()

To drop the database

    db.dropDatabase()

For more information,visit: <u>https://www.mongodb.com/what-is-mongodb</u>

Bootstrap

Bootstrap is a free and open-source front-end web framework for designing responsive websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only.

For more information,visit: <u>http://getbootstrap.com/</u>

Canvas Gauges

Canvas gauges are open source minimalist HTML5-based components for web applications.

For more information,visit: https://canvas-gauges.com/

JQuery

JQuery is one of the most popular javascript library used to make the client-side scripting of HTML easier.It makes it easier to create Document Object Model(DOM) based elements,handle page navigations and create plugins.

For more information,visit:https://jquery.com/

Node-RED

Node-Red is a software tool built on Node.js developed by IBM which makes it easier to wire together hardware devices. It provides simple browser based editing facility which makes it easier to create flows just by a simple drag of the mouse. The flows can then be deployed at runtime.

For more information,visit:https://nodered.org/

Mosquitto

Eclipse Mosquitto is an open source message broker which implements the MQTT protocol of communication. A broker is primarily responsible for receiving messages from the client,filtering them and sending them to appropriate subscribed clients.It is the counterpart of a MQTT client.

MQTT provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for "Internet of Things" messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers like the Arduino.

For more information,visit:https://mosquitto.org/

Important Note :
- Supports Eclipse mosquitto version above 1.4.7.
- Compatible with Ubuntu OS version 16.04, Windows, Ubuntu Mate.

MongoDB Docs

Mongoose Documentation: http://mongoosejs.com/index.html

Passport Documentation: http://passport.org

Auth Schema (Collection: authSchema)

This schema is used for the user verification through email address. Whenever a new user is created the details are stored here and the state is assigned to false. After the email verification the state is changed to true and the same details are stored in user schema.

var authSchema = mongoose.Schema({

```
  local      : {
    api_key    : String,
        a unique string to identify each user(generated server-side)
    email      : String,
        email id of the user: verification email will be sent on this mail
    password   : String,
        password of the user: for logging in after verification, hashed
    url        : String,
        url generated server-side to be sent to the email for verification
    state      : Boolean
        If verified: state = true -> schema is deleted from collection, else false
```

State	Meaning	Action

true	The user email is verified	Document deleted from auth Schema and added to the userSchema
false	The user email is still unverified (default setting)	-NA-

```
    },
        //for future login functionality using facebook, google and twitter

    facebook      : {
       api_key     : String,
       id          : String,
       token       : String,
       email       : String,
       name        : String
    },
    twitter       : {
       api_key     : String,
       id          : String,
       token       : String,
       displayName : String,
       username    : String
    },
    google        : {
       api_key     : String,
       id          : String,
       token       : String,
       email       : String,
       name        : String
    }

},
{
  collection:'authSchema',
```

```
  safe:true
});
```

User Schema (Collection: users)

This schema is used storing the user details that is required for login. Data is stored in this schema after the verification of the email address.

```
var userSchema = mongoose.Schema({

    local       : {

        api_key     : String,
            a unique string to identify each user(generated server-side)

        email       : String,
            email id of the user: verification email will be sent on this mail


        password    : String,
            password of the user: for logging in after verification, hashed using b-crypt
node

    },

        //for future login functionality using facebook, google and twitter
    facebook        : {

        api_key     : String,

        id          : String,

        token       : String,

        email       : String,

        name        : String

    },

    twitter         : {
```

```
    api_key    : String,
    id         : String,
    token      : String,
    displayName  : String,
    username    : String
  },
  google       : {
    api_key    : String,
    id         : String,
    token      : String,
    email      : String,
    name       : String
  }

});
```

User Details Schema (Collection: userDetails)

This schema is used to store the details of the users. It contains unique api_key which is used to identify the user. It also contains list of devices and projects the user is working on.

```
var userDetailsSchema = mongoose.Schema({


 api_key : {type:String, unique:true},
        Unique string to identify each user

 email   : String,
        Email of the user
```

```
    devices : Array,
```
Array of device ids added by the given user
```
    projects : Array
```
Array of projects ids created by the given user
```
},
{
    collection:'userDetails',  //Stored in collection userDetails
    safe:true
});
```

Notification Schema

This schema is used for storing the notification details of the events according to the time.The notifications are sent by the device. Some examples are: Already On/off, Device is switched on/off, Device is connected/disconnected etc. These notifications are displayed in the notifications panel on the device visuals page.

```
var notification = mongoose.Schema({
    time : {type : Date, default: Date.now},
        //time of receiving the notification from the device
    message: String
        // message conveyed by the notification

});
```

Notification sent by Device	Message	Meaning
AO@12	Device was already on at intensity 12%	A signal was sent from the application to switch on the device, but the device was already on.

AF@0	Device was already off.	A signal was sent from the application to switch off the device, but the device was already off.
ON@12	Device is switched on at intensity 12%	The device was switched on at intensity 12%
OF@0	Device switched off	The device was switched off.
AU	Authenticated User Scanned	The user is authenticated by scanning an RFID card.
UU	Unauthenticated User Scanned	The RFID card scanned by the user is not valid.
CN	Device Connected	MQTT connection has been established between the device and the application.
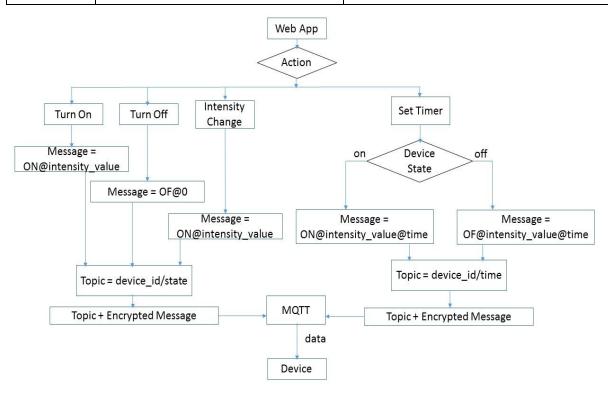
Fig. 38: Actions and corresponding messages.

Project Schema (Collection: projectData)

This schema contains the details of the project the user is working on. It contains list of all the devices that are used in the project.

```
var projectData = mongoose.Schema({

  project_name :String,
        // a user inputted string to identify the project

  project_id : String,
        // a server generated string to uniquely identify a project.

  description: String,
        // a user inputted description of the project

  device_id: Array
        // array of device_ids corresponding to the devices in the project


},
{

  collection:'projectData',

  safe:true

});
```

Sensor Schema (Collection: sensor)

This schema is used to store the device details. It also contains list of project IDs where the sensor is associated with.

```
var sensorSchema = mongoose.Schema({

  device_name : String,
        // user inputted string to identify the device

  device_id   : { type: String, unique: true },
        // server generated string to uniquely identify a device
```

description : String,
// user inputted description of the device

last_access : {type: Date, default: Date.now},
// time of creation of the device (name is misleading)

device_type : String,
// user inputted : type of device

timer   : {type:String,default:"false"},
// boolean, if timer is set or unset

timerDate   : String,
// if timer is set, the date to which it is set to

project_id   : {type:String,default:""},
// if the device belongs to a project, this field stores the id of the project it belongs to, else stores nothing

intensity   : {type:String,default:"0"},
// last set intensity of the device

state   : {type: String, default: "false"},
// state is true if device is on, else false

State	Meaning
True	Device is on
False	Device is off

activity   : [notification],
// array of all the notifications sent by the given device to the application.

values : [sensorDataSchema],
// stores all the time-value pairs used to plot the data sent by the device

energy : [ ]

// Array of numbers, stores the energy values for the last 30 days.

// The indices of the array correspond to the days of a month.

```
},{
  collection:'sensor',
  safe:true
});
```

Sensor Data Schema

This schema is used to store the sensor data values and the time when the device has to be turned on/off.

```
var sensorData = mongoose.Schema({
  time : {type : Date, default: Date.now},
        // timestamp of the instant when the data value to be plotted is received by
the application
  value : Number
        // data value to be plotted, sent by the device
});
```
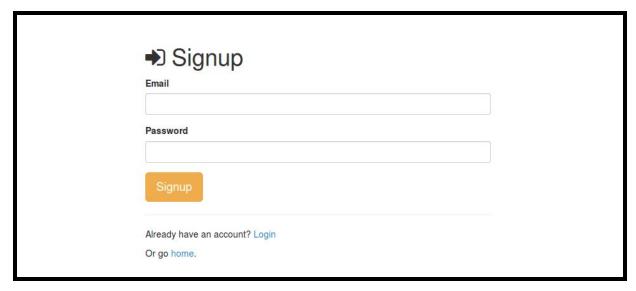
Usage

I.   Sign Up Page

Fig. 39: The signup page

This page is used for signing up of new users to access the platform. The user needs to enter his email ID and password which is used for user authentication during the login to the platform. To authenticate the user, an email is sent to the registered email. On clicking on the link provided in the email, the user account setup is completed and the user can then login to the application.

Frontend Checks:

1. Email is invalid: asked to enter again.
2. Both email and password are required.

Backend Checks

1. Email exists in database: user already exists, shows message.
2. Email does not exist in database: create a new entry in the authschema in the database, hash the password and send email to the registered email account for account verification.Email is sent through the email - iot.fossee@gmail.com.
3. If the user clicks on the sent link, delete user entry from the authSchema and enter in the userSchema. User account verification is successful.
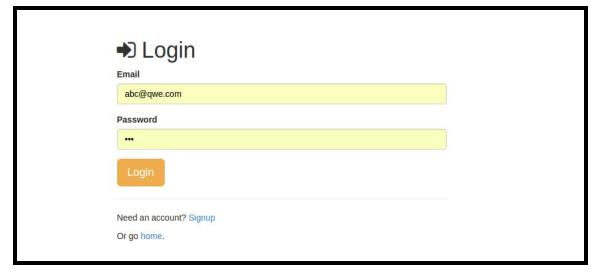
II.   Login Page

Fig. 40: The login page

Logs in the user if email-password pair is present in the users Schema and gives access to the platform to the users.If the user authentication is not completed(If user has not clicked on the link set through email), access is denied.

Frontend Checks:

1. Email is invalid: asked to enter again.
2. Both email and password are required.

Backend Checks:

1. Checks if the entered email is registered in the platform by checking the userSchema. If it is not present shows "User does not exist".
2. Checks if the entered password matches to the email from the userSchema. If the password is not matched it shows "Oops! Wrong password"

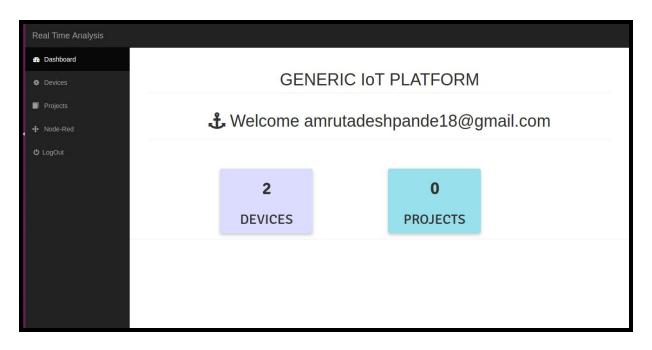## III. Profile Page/ Dashboard

Fig. 41: User Dashboard. Shows the number of devices added and the number of projects created

The dashboard or the profile page of the user shows the number of devices added by the user and the number of projects added by the user.

The side navigation bar has links to the following:

1. **Devices :** Displaying all the devices added by the user and a button for adding more devices.
2. **Projects :** Displaying the list of all the projects created by the user and a button for creating more projects.
3. **Node-Red:** Node-Red is a programming tool for adding together various hardware devices.
4. **LogOut :** For logging out of the account.

IV.   Devices Page
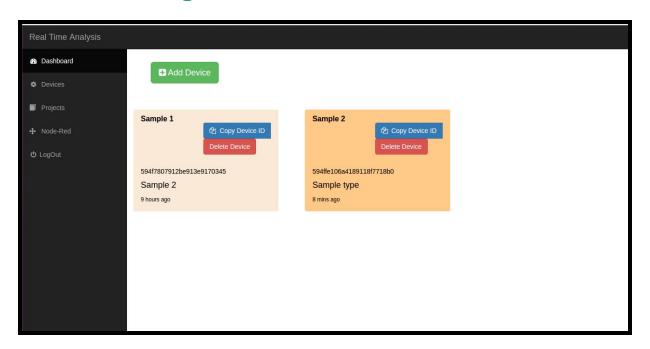

Fig. 42: Page displaying all the added devices

1. Shows all the devices added by the user in the form of attractively colored cards.The cards take up random colors each time the page is refreshed.
2. The cards display device name,id and device type.

3. The cards are clickable, on clicking the card, the visual page for the device opens.
4. The card also has **'Copy Device ID'** button which allows the user to copy the device id for use in the hardware code.
5. The **'Delete Device'** button allows the user to delete the particular device.
   a. A modal window pops up on clicking the 'Delete Device' button, confirming the deletion.
6. The **'Add Device'** button allows the user to add new device.On clicking the button, a modal form appears which looks like this:
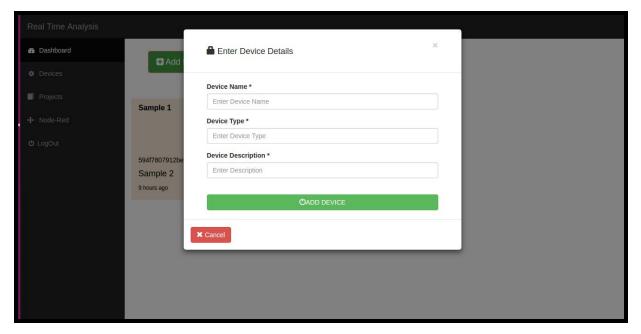
Fig. 43: The modal window for adding a device. Opens when the 'Add Device' button is clicked.

V. Node-RED
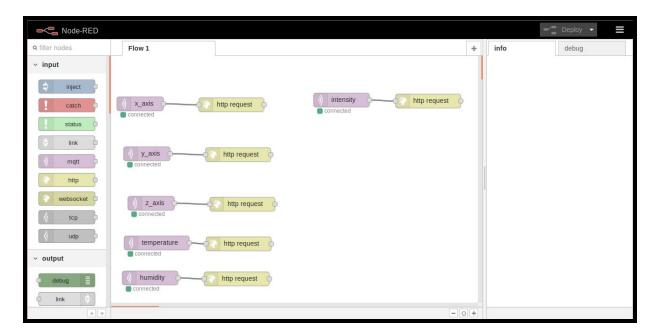
Fig. 44: Node-RED

<u>Node-RED</u> :Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.
It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.
The user can use this tool to experiment with various devices by sending and receiving the data over MQTT protocol.

VI. Visuals

The visual pages of the device is shown in the following screenshots.
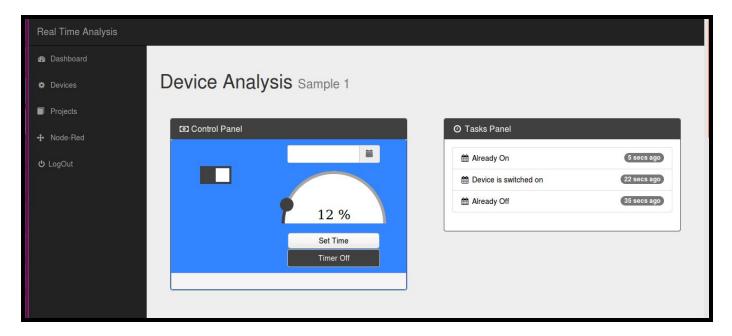
Fig. 45: The upper portion of the visuals page showing the control panel and the task panel

The top portion of the visual page shows controls for switching the device ON/OFF and controlling the intensity of the device.

The user can control the device directly by the switch or can set the timer.

When the timer is set then the datepicker is disabled. On expiration of the timer an expire message is displayed on the screen. To set the timer again or control the device press the 'Timer Off' button. The right side shows a task panel which displays the notifications sent by the device.

Fig. 44: Line chart plotting real time data

The above screenshot shows the middle portion of the visual page displaying the line chart. The line chart shows the real time data sent by the device over MQTT.

The line chart is made using the morris.js library.
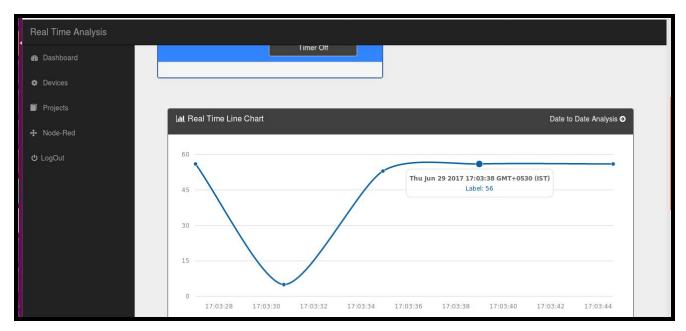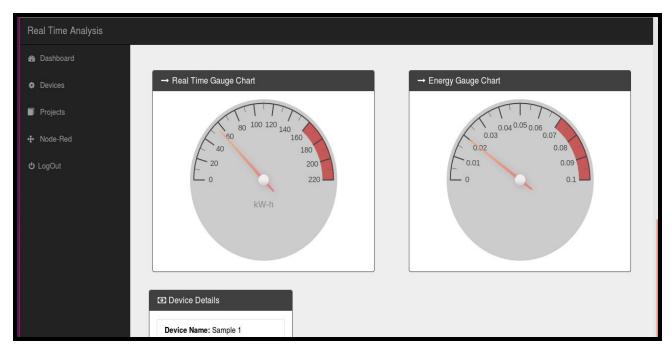

Fig. 46: Energy and gauge chart plotting real time data and energy

The above screenshot shows the bottom portion of the visual page.

It shows a **real time gauge graph** plotting real time device data. The **energy gauge graph** int the middle portion shows energy consumption of the device till that time. The bottom right side shows the **device details.**

VII.   Date to Date Data Analysis
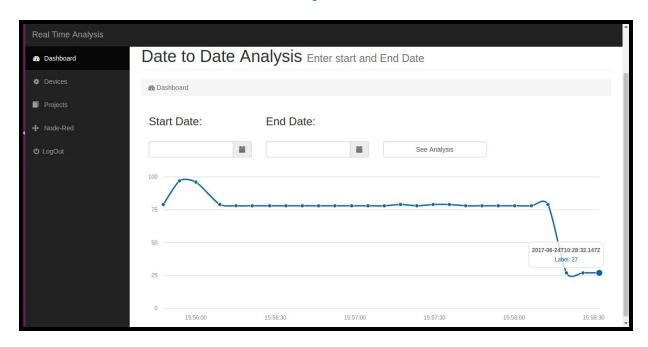
Fig. 47: Date to date Analysis

This page shows the analysis of data from a starting date to an ending date by plotting a line chart of the data between that period.

The user can enter the starting and ending date and the appropriate values are taken from the database and plotted on the chart.

Data of the same date as current cannot be displayed if the sensors are sending data to plot real-time on the visuals page.

VIII.   Projects
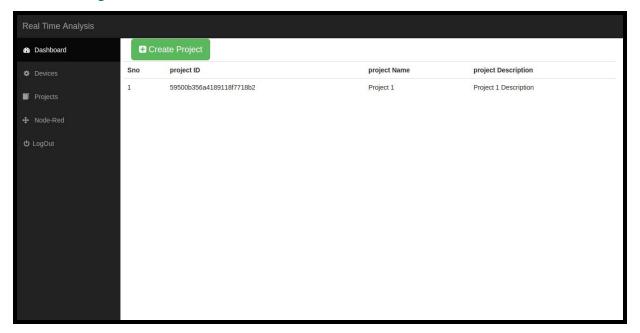
Fig. 48: Page showing list of all created projects

The page shows a list of projects added by the user,clicking on the project shows the devices added under that project.

The Projects Section can be used when some devices have similar uses.

The devices are grouped together under a common project, can be seen together under one project name.

These devices will also be visible individually in the devices section.

Clicking on a device leads to a page like the Visuals page. It gives the details and controls of that device.

MQTT Communication

All the communication between the device and the application is through the MQTT protocol. MQTT[1] (MQ Telemetry Transport or Message Queue Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922)[2]publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol. It is designed for

connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on the topic of a message.

All the messages sent on the topic are encrypted using the encryption protocols described in the Encryption/Decryption section.

The following are the details of the messages being sent and received between the application and the device:

Topic	Message	Meaning	Published/ Subscribed
device_id/state	T@12 F@0	T@12: Device is switched on at intensity 12% from the application.	published
device_id/time	T@12@123	Device is to be switched on at intensity 12% after 123 seconds. (Set Timer at the application)	published
device_id/notify	ON@12	Device is switched on at intensity 12%. (More details below)	subscribe
device_id/values	45	Value to be plotted, sent by device	subscribe
device_id/energy	30	30 kW-h of energy has been consumed by the device from start of the day till current instant(the instant when the notification is received by the application.	subscribe

➔ The /state topic :

◆ **Data Published** : ON/OFF signal with intensity

◆ **Format :** <state>@<intensity>

- state :
  - ○ T => device is switched on
  - ○ F => device is switched off

- Intensity: intensity at which device is to be switched on.

◆ **Function :**

- When the device is switched on/off from the device dashboard (Visuals Page) a message is sent to the actual device on this topic.

- User can set the intensity at which the device is to be switched on.

- The signal sent on this topic is executed by the device instantaneously.

→ The /timer topic:

◆ **Data Published** : ON/OFF signal with intensity and timer

◆ **Format :** <state>@<intensity>@<time>

- state :
  - T => device is switched on
  - F => device is switched off

- Intensity: intensity at which device is to be switched on.

- Time: Scheduling time after which the state has to change

◆ **Function:**

- When the timer to switch the device on/off is set from the device dashboard (Visuals Page) a message is sent to the actual device on this topic.

- User can set the intensity at which the device is to be switched on.

- The signal sent on this topic is executed by the device after the specified time interval.

→ The /value topic :

◆ **Data Subscribed :** value sent by the device to be plotted on the device dashboard.

◆ **Format :** <value>

- Value : value to be plotted on the graphs on the device dashboard.

◆ **Function :**

- A value is sent by the sensors connected on the device to the dashboard.

- This value is instantaneously published on the /value topic by the device, and subscribed by the server when the device visuals page is open. Values are not subscribed if the device visuals page is not open.

- The value is sent through a socket to the graphs (Line and Gauge Chart) where it is instantaneously plotted.

- The value of the device is stored in the MongoDB server.

- The stored values can be replotted by entering the start and end date in the Date to Date Analysis section. (Link over the line chart)

→ The /energy topic:

◆ **Data Published :** Energy consumed by the device from the start of the day till the current instant

◆ **Format :** <energy>

- Energy: The value of the energy consumed by the device.

◆ **Function :**

- Shows the energy consumed by the device on that day.

- The energy is plotted on the Energy gauge on the device visuals page.

- The energy of the past month is stored in the mongoDB database.

- An array of 31 elements is created in the sensor collection of the MongoDB database.
- The date on which the energy is sent is found using Date().getDate() function.
- The energy value is stored at the array index corresponding to the (date -1)
- In future versions, these can be accessed and plotted on a line graph.

➔ The /notify topic:

◆ **Data Published :** The messages about the actual device state from the device are received on the /notify topic.

◆ **Format :** Various messages have various formats. All messages start with the message code explaining the meaning and usage of the message.

Notification sent by Device	Format	Message	Meaning
AO@12	AO@<intensity>	Device was already on at intensity 12%	A signal was sent from the application to switch on the device, but the device was already on.
AF@0	AF@0	Device was already off.	A signal was sent from the application to switch off the device, but the device was already off.
ON@12	ON@<intensity>	Device is switched on at intensity 12%	The device was switched on at intensity 12%

OF@0	OF@0	Device switched off	The device was switched off.
AU	AU	Authenticated User Scanned	The user is authenticated by scanning an RFID card.
UU	UU	Unauthenticated User Scanned	The RFID card scanned by the user is not valid.
CN	CN	Device Connected	MQTT connection has been established between the device and the application.

◆ **Function :**

- For switching a device on and off, the notify acts as a handshake between the device and the application.

- For example, to switch on the device, the user can use the slider switch on the visuals page. Switching it on publishes on the "state" topic a message of the format "ON@intensity" when intensity is the current intensity set on the intensity slider. When such a notification is received by the device, it switches the device on and publishes on the "notify" topic that the device was switched on in the format "ON@intensity". This is especially useful if the device fails to switch on the device, or some error has occurred, and hence it fails to publish on the "notify" topic. Since the handshake was never completed, the state of the device remains unchanged in the database and no changes are reflected on the application.

- All the messages are stored in the notifications array in the sensor collection. The most recent notifications are shown under the 'Tasks Panel' on the device visuals page along with the time.

Android Mobile Application

The Android app for the platform is created by using webview. A "webview" is a program packaged within a portable application delivering a hybrid application. Utilizing a webview permits versatile applications to be constructed utilizing Web advancements (HTML, JavaScript, CSS, and so on). Since the platform is mobile responsive webview is used to make the android app.

Whenever the app is opened the login page is visible to the user, where he needs to authenticate his identity. The rest of the pages are exactly same as that of the web application. This mobile app provides all the functionalities of the web application. The user need not even update the app in the future because the update in the server also changes the interface in the application.

Technologies Used

1. <u>Android Studio</u> - It is the official IDE for android which provides fast tools for building apps on every type of android device.

   How to install Android-studio?

   Follow the guidelines in the following link:
   https://developer.android.com/studio/install.html

2. WebView - WebView is used to run web application inside the android app. Steps for building the webview app is given in the following link below https://developer.android.com/guide/webapps/webview.html

   The content of the webview is set to the login page using the following code
   WebView myWebView = (WebView) findViewById(R.id.webview);

   myWebView.loadUrl("http://<web address of the platform>");

References:

1. Adding NodeMCU board to Arduino IDE:
   https://github.com/esp8266/Arduino
2. Installing and Testing Mosquitto:
   http://wingsquare.com/blog/setting-up-mqtt-mosquitto-broker-in-ubuntu-linux/
3. Installing Node-Red:
   http://nodered.org/docs/getting-started/installation.html
4. RFID:
   https://en.wikipedia.org/wiki/Radio-frequency_identification'
5. MQTT:
   Mqtt.org

6. Node JS:
   https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm
7. Bootstrap:
   https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)
8. Mosquitto broker:
   Mosquitto: https://mosquitto.org/
9. Node-Red:
   NODE-RED: https://nodered.org/
10. MQTT:
    Wikipedia: https://en.wikipedia.org/wiki/MQTT
11. Intensity Regulator:
    https://diyhacking.com/arduino-lamp-dimmer/