Sprint-1

Team ID	PNT2022TMID11493
Project Title	Signs with smart connectivity for better road safety
Date	12/11/22

Problem Statement

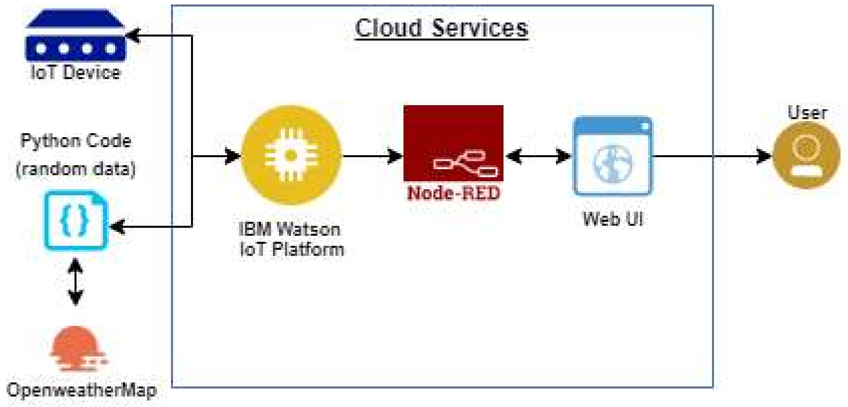
This project will replace the static boards to smart boards that will change the speed limits according to the weather climate and show diversion messages if there are accidents in the road and alert messages if there are hospitals,schools or any roadworks.

Proposed Solution

This project aims to collect information such as Weather conditions, Location data, other road information, etc and process them accordingly to make changes in the sign boards and speed control appropriately.The corresponding information is shared as a message to the User Interface.

Theoretical Analysis

To implement this solution the following architecture will be followed ▭

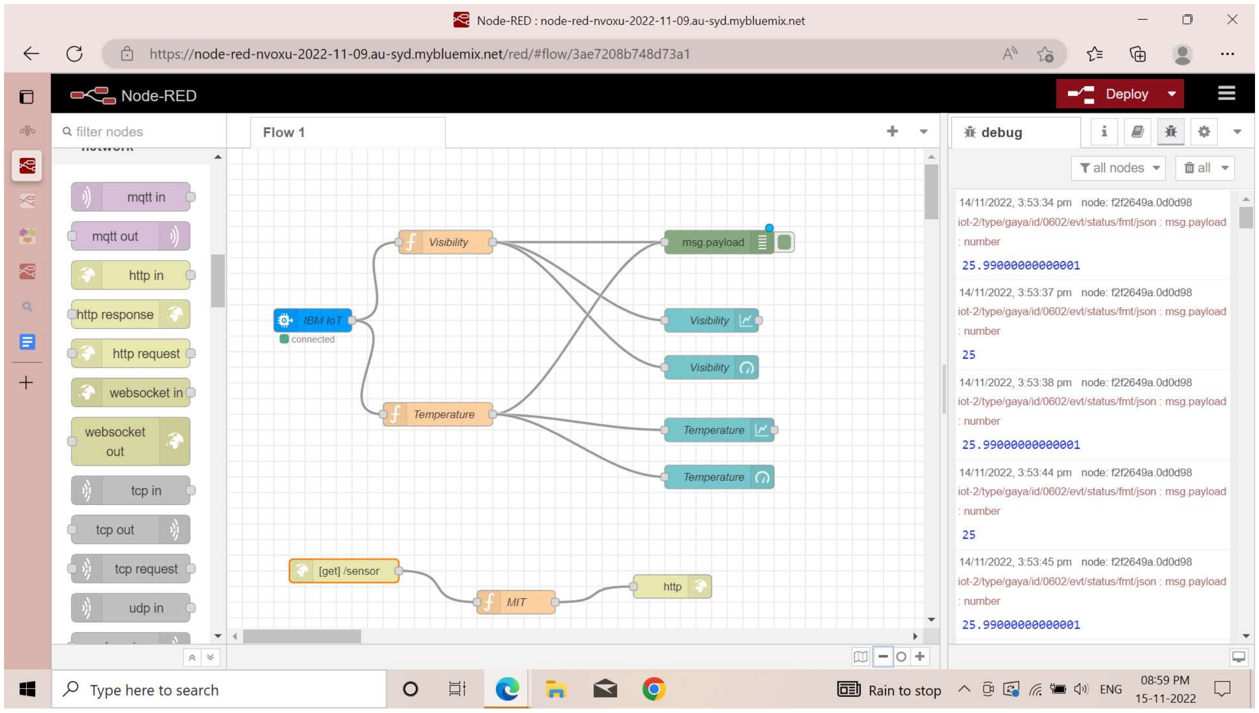


Required Software Installation :

Node-Red

It is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions.



```
14/11/2022, 3:53:34 pm   node: f2f2649a.0d0d98
iot-2/type/gaya/id/0602/evt/status/fmt/json : msg.payload
: number
25.99000000000001

14/11/2022, 3:53:37 pm   node: f2f2649a.0d0d98
iot-2/type/gaya/id/0602/evt/status/fmt/json : msg.payload
: number
25

14/11/2022, 3:53:38 pm   node: f2f2649a.0d0d98
iot-2/type/gaya/id/0602/evt/status/fmt/json : msg.payload
: number
25.99000000000001

14/11/2022, 3:53:44 pm   node: f2f2649a.0d0d98
iot-2/type/gaya/id/0602/evt/status/fmt/json : msg.payload
: number
25

14/11/2022, 3:53:45 pm   node: f2f2649a.0d0d98
iot-2/type/gaya/id/0602/evt/status/fmt/json : msg.payload
: number
25.99000000000001
```

Installation of IBM IoT and Dashboard nodes for Node-Red

In order to connect to IBM Watson IoT platform and create the Web App UI these nodes are required

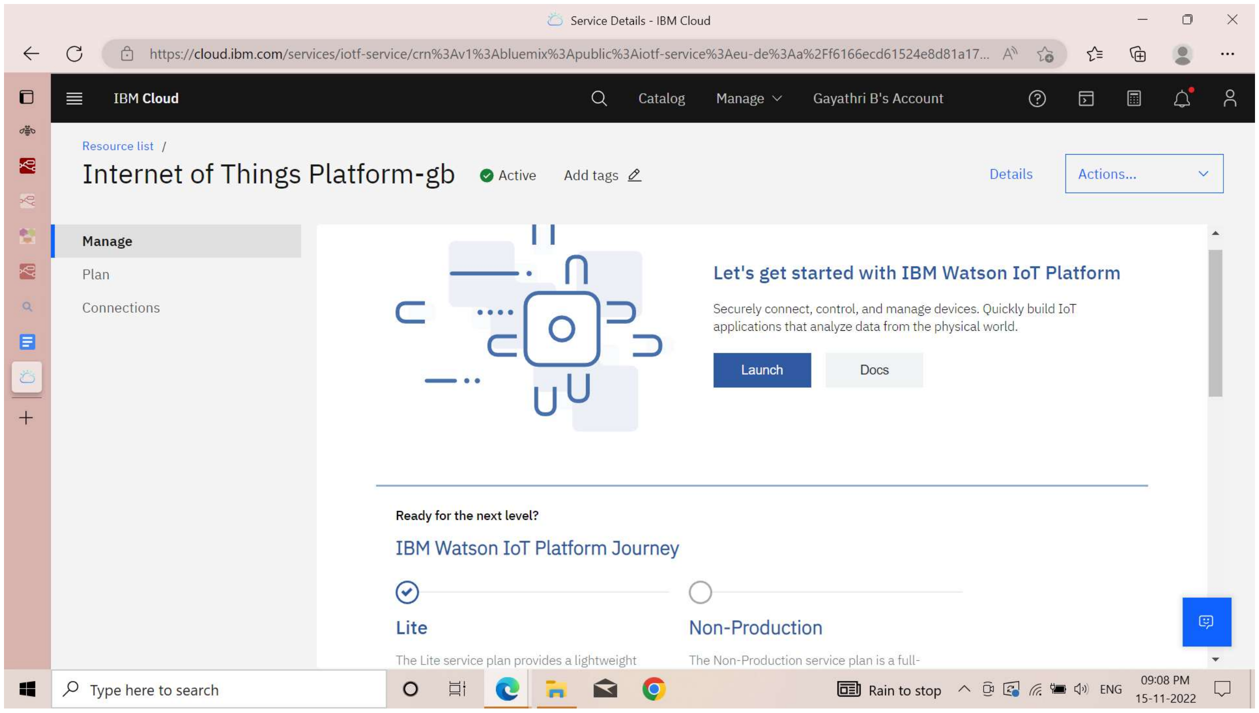
1. IBM IoT node
2. Dashboard node
2. Dashboard node

IBM Watson IoT Platform

A fully managed, cloud-hosted service with capabilities for device registration, connectivity,control, rapid visualization and data storage. IBM Watson IoT Platform Is a managed, cloud hosted service designed to make it simple to derive value from your IoT devices.

Steps to configure:

- Create an account in IBM cloud using your email ID
- Create IBM Watson Platform in services in your IBM cloud account
- Launch the IBM Watson IoT Platform
- Create a new device
- Give credentials like device type, device ID, Auth. Token
- Create API key and store API key and token elsewhere.


Service Details - IBM Cloud
https://cloud.ibm.com/services/iotf-service/crn%3Av1%3Abluemix%3Apublic%3Aiotf-service%3Aeu-de%3Aa%2Ff6166ecd61524e8d81a17...
IBM Cloud
Catalog
Manage
Gayathri B's Account
Resource list /
Internet of Things Platform-gb
Active
Add tags
Details
Actions...
Manage
Plan
Connections
Launch
Docs
Type here to search
Rain to stop
ENG
09:08 PM
15-11-2022

Let's get started with IBM Watson IoT Platform

Securely connect, control, and manage devices. Quickly build IoT applications that analyze data from the physical world.

Ready for the next level?

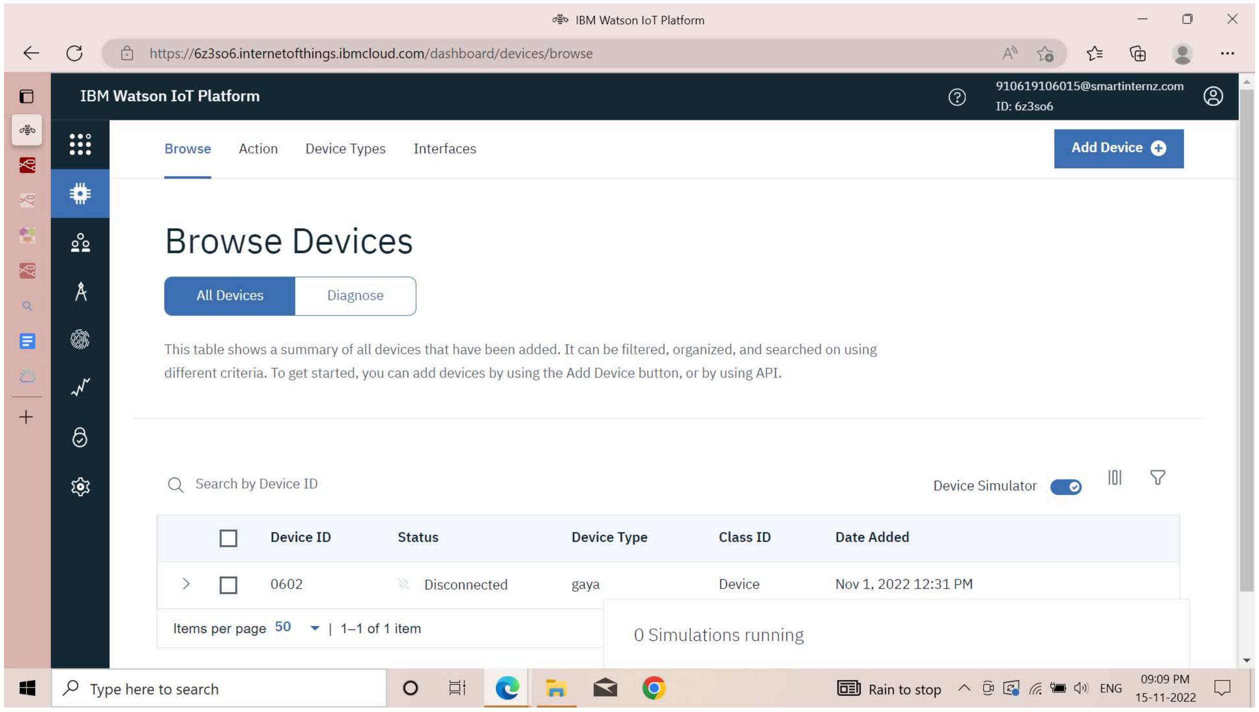
IBM Watson IoT Platform Journey

Lite

The Lite service plan provides a lightweight

Non-Production

The Non-Production service plan is a full-



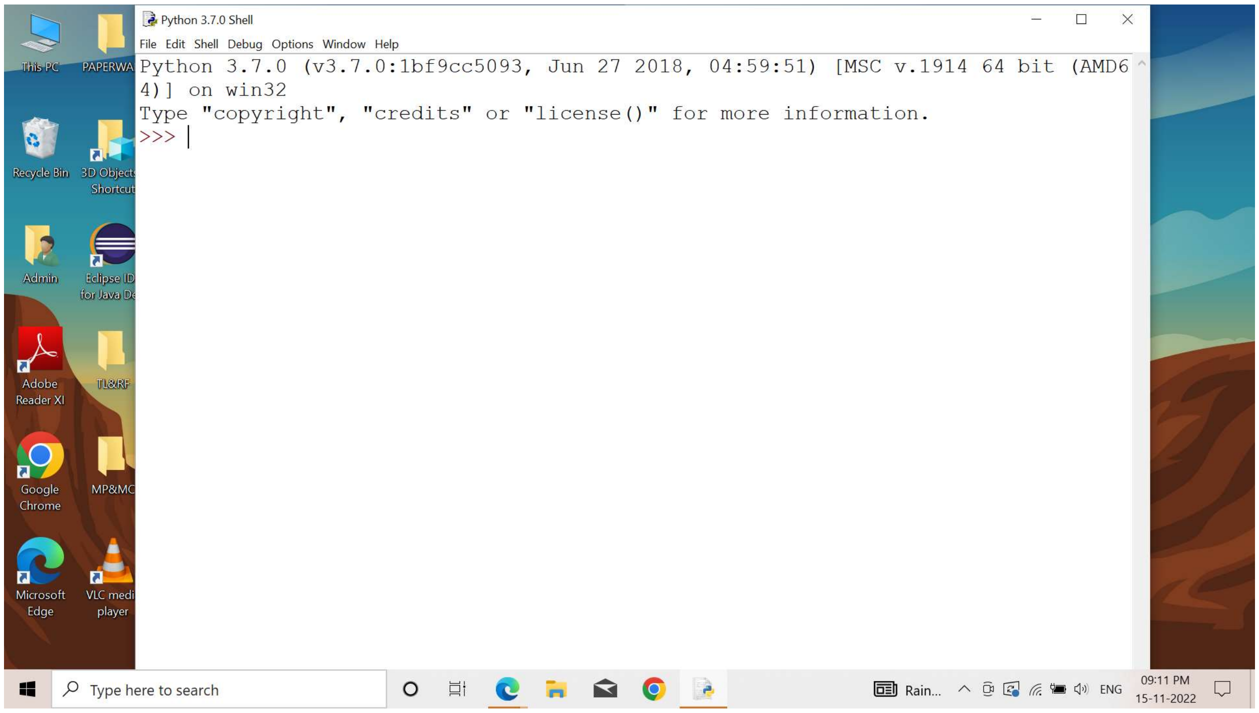
Browse Devices

This table shows a summary of all devices that have been added. It can be filtered, organized, and searched on using different criteria. To get started, you can add devices by using the Add Device button, or by using API.

Device ID	Status	Device Type	Class ID	Date Added
0602	Disconnected	gaya	Device	Nov 1, 2022 12:31 PM

Items per page 50 | 1–1 of 1 item

Python IDE 3.7



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Python code:

```
#Python code
# IMPORT SECTION STARTS
import weather
from datetime import datetime as dt
# IMPORT SECTION ENDS
# -----------------------------------------------
# UTILITY LOGIC SECTION STARTS
def processConditions(myLocation,APIKEY,localityInfo):
 weatherData = weather.get(myLocation,APIKEY)
 finalSpeed = localityInfo["usualSpeedLimit"] if "rain" not in weatherData
else  localityInfo["usualSpeedLimit"]/2
 finalSpeed = finalSpeed if weatherData["visibility"]>35 else finalSpeed/2
 if(localityInfo["hospitalsNearby"]):
 # hospital zone
   doNotHonk = True
 else:
   if(localityInfo["schools"]["schoolZone"]==False):
 # neither school nor hospital zone
    doNotHonk = False
   else:
 # school zone
    now = [dt.now().hour,dt.now().minute]
 activeTime = [list(map(int,_.split(":"))) for _ in
localityInfo["schools"]["activeTime"]]
 doNotHonk = activeTime[0][0]<=now[0]<=activeTime[1][0] and
activeTime[0][1]<=now[1]<=activeTime[1][1]
 return({
 "speed" : finalSpeed,
 "doNotHonk" : doNotHonk
 })
# UTILITY LOGIC SECTION ENDS
```

```
# Python code
# IMPORT SECTION STARTS
import brain
# IMPORT SECTION ENDS
# -----------------------------------------------
# USER INPUT SECTION STARTS
myLocation = "Chennai,IN"
APIKEY = "c7388b7d0d823ee0ee0be65c6fd40411"
```

```
localityInfo = {
  "schools": {
    "schoolZone": True,
    "activeTime": ["7:00", "17:30"]  # schools active from 7 AM till 5:30
PM
  },
  "hospitalsNearby": False,
  "usualSpeedLimit": 40  # in km/hr
}
# USER INPUT SECTION ENDS
# -----------------------------------------------
# MICRO-CONTROLLER CODE STARTS
print(brain.processConditions(myLocation, APIKEY, localityInfo))
'''
MICRO CONTROLLER CODE WILL BE ADDED IN SPRINT 2 AS PER OUR PLANNED SPRINT
SCHEDULE
'''
# MICRO-CONTROLLER CODE END
```

```
# Python code
import requests as reqs


def get(myLocation, APIKEY):
  apiURL =
f"https://api.openweathermap.org/data/2.5/weather?q={myLocation}&appid={AP
IKEY}"
  responseJSON = (reqs.get(apiURL)).json()
  returnObject = {
    "temperature":
    responseJSON['main']['temp'] - 273.15,
    "weather": [
      responseJSON['weather'][_]['main'].lower()
      for _ in range(len(responseJSON['weather']))
    ],
    "visibility":
    responseJSON['visibility'] /
    100,  # visibility in percentage where 10km is 100% and 0km is 0%
  }
  if ("rain" in responseJSON):
    returnObject["rain"] = [
      responseJSON["rain"][key] for key in responseJSON["rain"]
    ]
  return (returnObject)
```