***CMPE 280(Spring 2020)***

***Firefighter Dashboard Project Report***

The proposed project (Firefighter dashboard) shall be used for the firefighter’s safety precautions. In the current scenario, firefighters are not embedded with any sensor device, hence it becomes difficult to keep track of all the firefighters by the team lead.

The aim of this project is to get all the telemetry data of the firefighter team members and send it to the team lead by displaying it in the front end HTML5-CSS-JavaScript based dashboard/website. The technological hurdles that shall be associated with this project is interfacing hardware boards by STMicroelectronics containing onboard sensors with Google Cloud and Firebase Backend.

After successful completion of this project, the firefighting team lead shall monitor the telemetry data and accordingly provide instructions to its team members preventing any life casualty/injury. The primary area of application shall be for the Firefighting department, however, with some medication and adding other sensors, this project application shall be extended to military operations or any sport adventures like camping. The project report shall be divided into 3 parts

1. STMicroelectronics Hardware Description, sensor interfacing and communication with the Google Cloud(IoT)
2. Front end website based on HTML5-CSS-JavaScript, node js, socket.io and react API
3. Back End using Firebase

***STMicroelectronics Hardware Integration with Google Cloud***

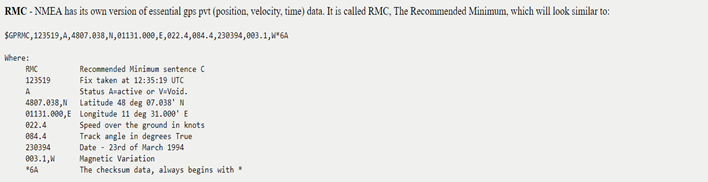
* The hardware selected for this project is STMicroelectronics STM32 IoT Discovery Kit (B-L475E-IOTA) since it has integrated support of all the telemetry sensors such as temperature, pressure, humidity, etc along with Wifi module to send the data over the cloud.



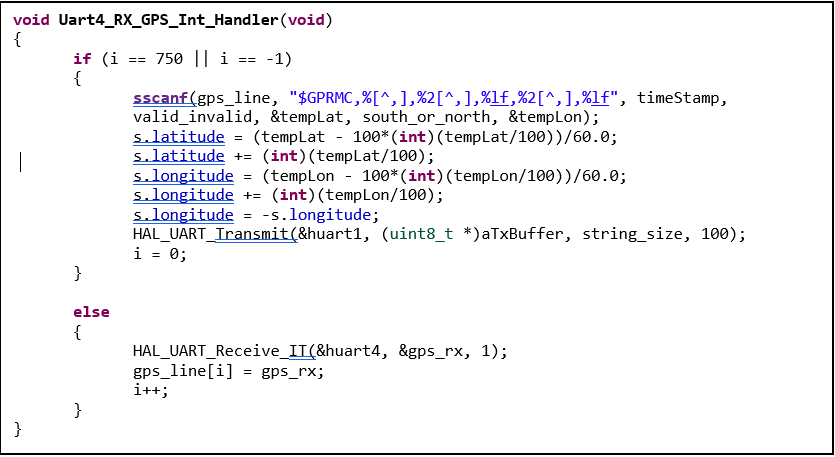
***Sensor Overview***

1. **GPS Sensor**

* The GPS module used in this project is <https://www.mikroe.com/gnss-4-click> . It works on UART communication and requires a GPS parsing algorithm to extract latitude and longitude.
* We connected the GPS module to a GPS antenna with RX and TX pins connected to UART4 pins (PA0 to RX and PA1 to TX). The module is communicating at the baud rate of 9600 bps.
* After setting the baud rate, we get GPRMC data over the output. The data format for GPRMC is as follows:



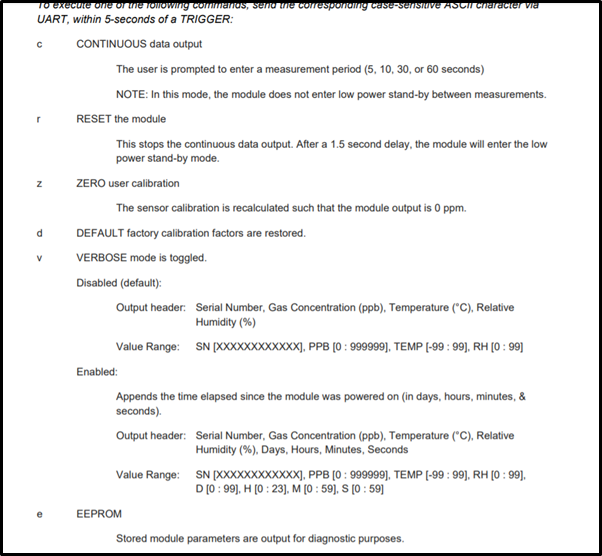
* From the format, we can see that latitude and longitude is the only information that needs to be parsed. The location value in this format is a combination of actual latitude and longitude as well as the time. So, we implemented a parsing logic that scans the values as per format, stores in variables and converts to actual latitude and longitude.



* The above lines of code represent the GPS parsing logic in UART4 interrupt callback function. We define a buffer of 750 and collect the characters and store them in a character array till the buffer is full. Once it is full, we scan the array to find the characters starting with “$GPRMC” and store the values, convert to int and extracts the actual gps latitude and longitude.

1. **CO Sensor**

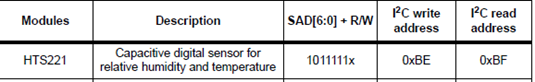
* CO Sensor is provided by Spec sensor(<https://www.spec-sensors.com/product/iot-co-1000-digital-co-sensor-module/>) which works on UART communication. We have various modes that can be configured to get the CO sensor data like Gas Concentration(PPB), humidity and temperature.
* Below screenshot explains the characters to be sent over UART to get the sensor data as well as the output format (Reference datasheet:<https://www.spec-sensors.com/wp-content/uploads/2017/01/DG-SDK-968-045_9-6-17.pdf>)



* The gas concentration of CO Sensor ranges from **800PPB to 2000PPB** for normal environmental conditions, since the natural concentration of carbon monoxide in air is around **0.5 to 1 ppm**, and that amount is not harmful to humans.
* CO sensors are normally very stable. They are cross sensitive to a couple of potential interfering gases, however.
* **Usually negative readings** are the result of the CO sensor being fresh air adjusted while in the presence of a detectable interfering gas, or when the sensor is fresh air adjusted before it has completed recovering from a prior exposure to an interfering gas.
* The CO sensor is always producing a signal as long as the instrument is turned on. In the presence of CO the signal goes up. In fresh air, you still get a signal, but the value is lower. When you fresh air zero the instrument you are telling the instrument to use the signal from the sensor at that moment as the point of comparison.

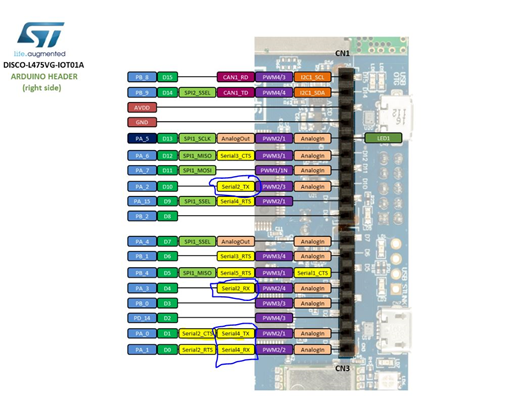
1. **HTS221 On board sensor in STM32 IoT Discovery Kit**

* The HTS221 is an ultra-compact sensor for relative humidity and temperature. It includes a sensing element and a mixed signal ASIC to provide the measurement information through digital serial interfaces.
* The sensing element consists of a polymer dielectric planar capacitor structure capable of detecting relative humidity variations and it is manufactured using a dedicated ST process.
* On the STM32L4 Discovery kit for IoT nodes, the I2C2 bus from STM32L475VG is used.
* STM32Cube\_FW\_L4\_V1.12.0 contains the sample project that measures the temperature from onboard HTS221. Since it uses I2C, slave address for HTS221 to read the temperature is



***Hardware connections:***

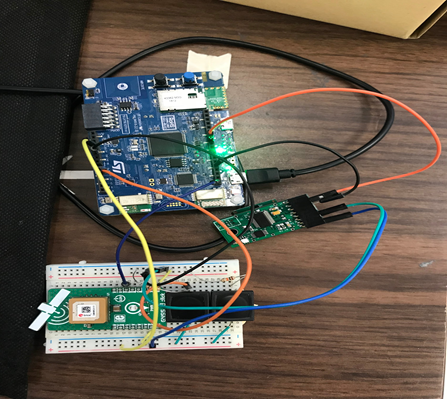
* We use B-L475E-IOTA board to communicate with the sensors and publish the data over the google cloud. We will be using UART2 and UART4 pins of the board to connect to the sensors. Below are the pin connections:



* The below table is the pin connection details from board to the sensors:

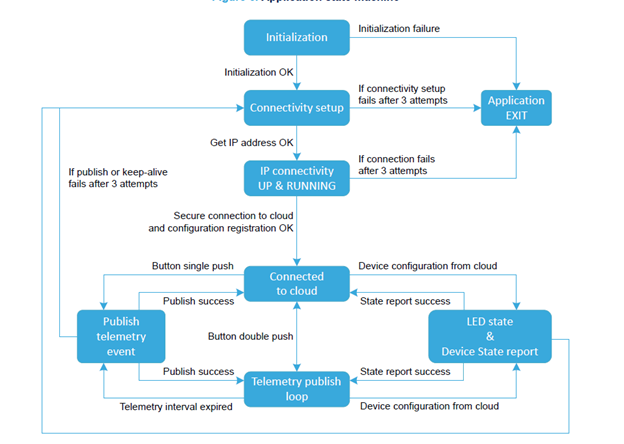
|  |  |
| --- | --- |
| **B-L475E-IOT01A pins** | **Sensor pins** |
| PA\_0 (D1) | ‘RXD’ pin of GPS Module |
| PA\_1 (D0) | ‘TXD’ pin of GPS Module |
| PA\_3(D4) | Pin 3(TX) of CO Sensor |
| PA\_2(D10) | Pin 2(RX) of CO Sensor |
| 3.3V | ‘3.3V’ pin of GPS module  Pin 8 of CO Sensor |
| GND | ‘GND’ pin of GPS Module  Pin 6 of CO Sensor |

* The actual hardware connections look like this:

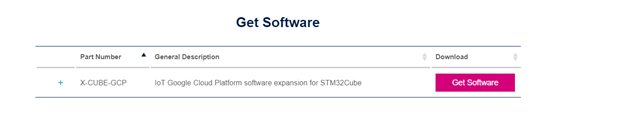


***Google Cloud Configuration:***

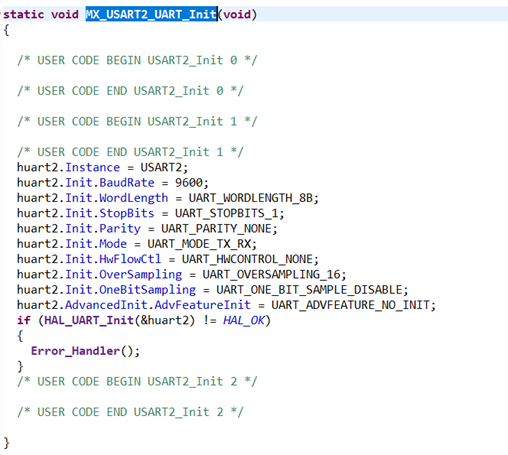
* In order to publish the data to google cloud, we have to make a free google cloud account, register and enable the APIs. The steps are mentioned in section 5.2 of GCP User Manual (<https://www.st.com/content/ccc/resource/technical/document/user_manual/group1/e9/8a/9b/73/5c/ff/4d/10/DM00522079/files/DM00522079.pdf/jcr:content/translations/en.DM00522079.pdf>)
* The Application flow for entire code shall be as follows:



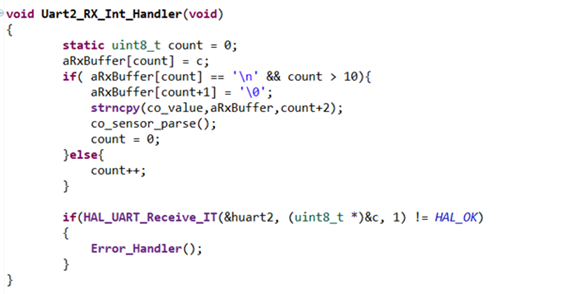
* Download the Google cloud package (<https://www.st.com/en/embedded-software/x-cube-gcp.html>) was used to send the sensor data over the google firebase
* The GCP package was installed using the steps mentioned in sections 5.2 to 5.5 of the user manual (<https://www.st.com/content/ccc/resource/technical/document/user_manual/group1/e9/8a/9b/73/5c/ff/4d/10/DM00522079/files/DM00522079.pdf/jcr:content/translations/en.DM00522079.pdf>)



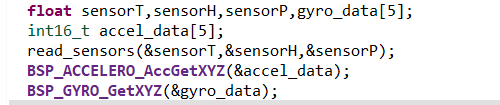
* After unzipping the package, I imported the project in Eclipse with the given sample project “B-L475E-IOT01\_GoogleIoT”
* Created the UART2 instance for the CO sensor and enabled the interrupt to send command and receive data from the sensor.



* Once the gcp function starts, the UART2 and UART4 interrupts are triggered and UART\_RXCallback() was executed, where it started storing the sensor data as string.



* In addition to sending the gps latitude and longitude and CO sensor’s gas concentration and relative humidity, I sent the temperature, pressure, accelerometer and gyroscope data to the cloud. The GCP package has an API built in to get the readings from the onboard temperature sensor, accelerometer and gyroscope.



* Once the data was present in the variable, next step was to send data over the google cloud. We stored the data format into a character buffer using snprintf() and strcat() and converted to the JSON format.



**Terminal Output:**

In teraterm, go to terminal->receive->AUTO, transmit->LF, Enable Local Echo. Go to serial port-->Baud rate: 115200

Perform the following steps as the output is shown in teraterm

1) Enter Wifi SSID, Password and security type. **Make sure to use your Mobile Hotspot to establish data connection.**

2) Enter the TLS certifications by adding the public key and device key generated while configuring Google cloud in the terminal

3) The public key can be found inside “comodo\_google.pem” found here: **STM32CubeExpansion\_Cloud\_GCP\_V1.0.0\Projects\Common\GoogleIoT**

4) The private key is the one used to configure the google cloud topic.

5) Enter the Google cloud core connection server address (project id, device id, registry id, region). The following format is accepted in the terminal:

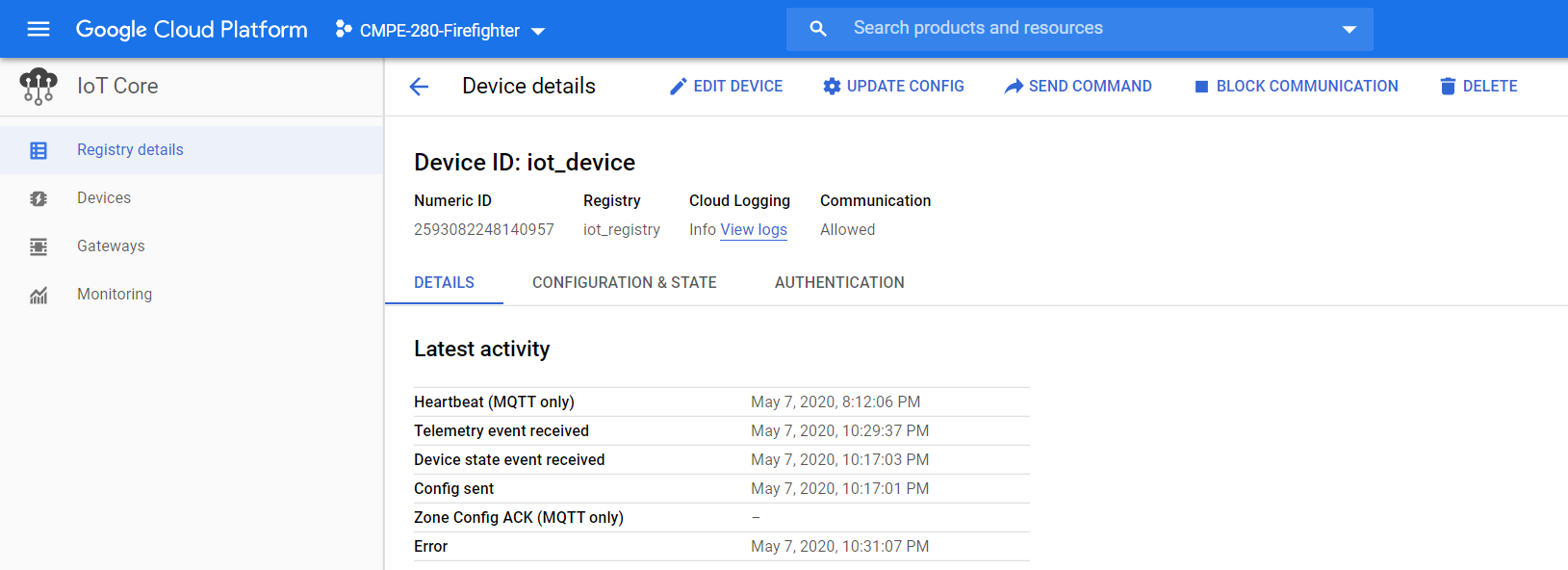


6) After socket connection is established, press the blue button twice to send data over the google cloud. Basically, it will subscribe to a pub/sub topic and publish the data over that topic.

7) Below is the output format going in the JSON format



* Basically, it will subscribe to a pub/sub topic and publish the data over that topic. After successful sending of data, the google cloud indicates the date and time of data published.



* Once the data was published in the google cloud, the next step was to send the data from Google cloud to Firebase.

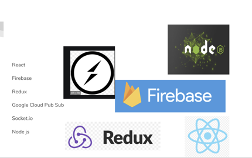
**Web Application Technology:**

Backend Services are created in Nodejs. Unit test cases were written using the Jest framework. Load Testing was performed using jMeter. A javascript module of Pub Sub is used to subscribe to the endpoint which listens to the various services.

The major endpoints created were Login, Signup, List all pin points, get details of single points, show analytics on historical data. Since the database acts as the endpoint, all details get filled in it.

We used a fire store-schema node javascript module to talk to the database. The data in the fire store is stored as documents. A group of documents make a collection.

Front end application was created using React, Redux and react-google-maps module. Redux was used to maintain the states easily.



**BACKEND**

There are 6 major services:

· Login

· Sign Up

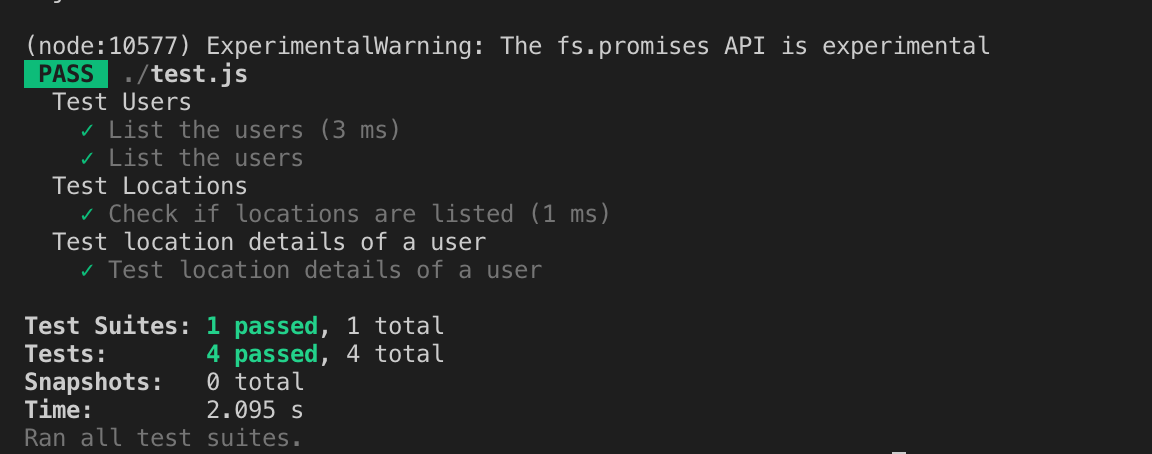
· Show all geographical pins

· Display details on a single pin

· Display analytics on historical data.

The users collection had details about all the users – these details included first name, last name, age, location, temperature. The sensor collection includes details like humidity, temperature, pressure etc. It also shows the user, whose sensor data is being displayed. Based on this, we were able to query the database and display relevant data.

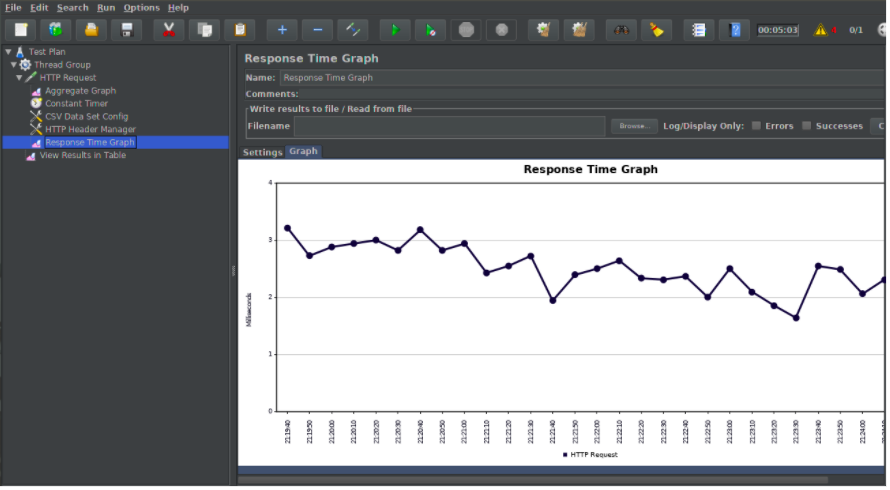
The tests are written using jest framework.We run the command npm test to see a comprehensive list :



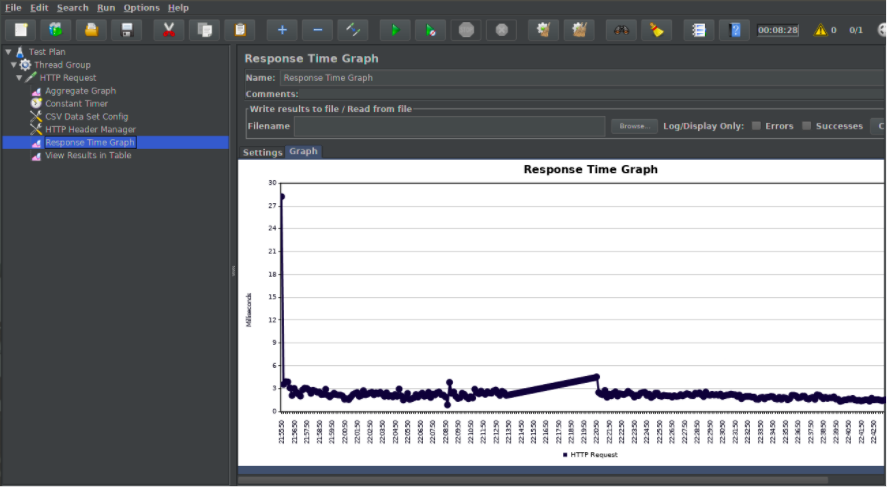
JMETER TESTING:

In order to test how our services scale, we used jMeter testing. We have attached the performance of our API through these screenshots.

For 100 concurrent users:



For 500 concurrent users

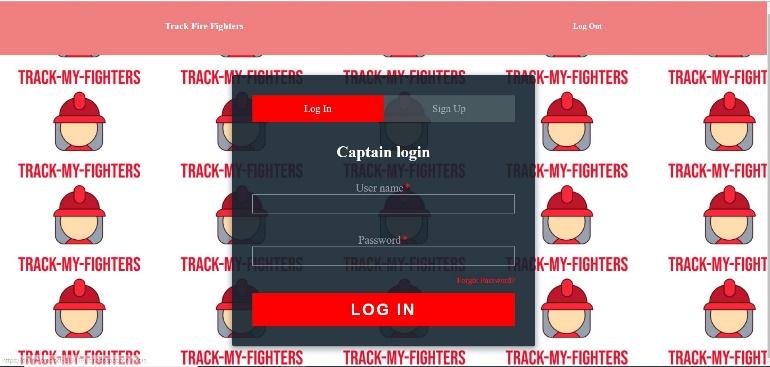


**FRONT END**

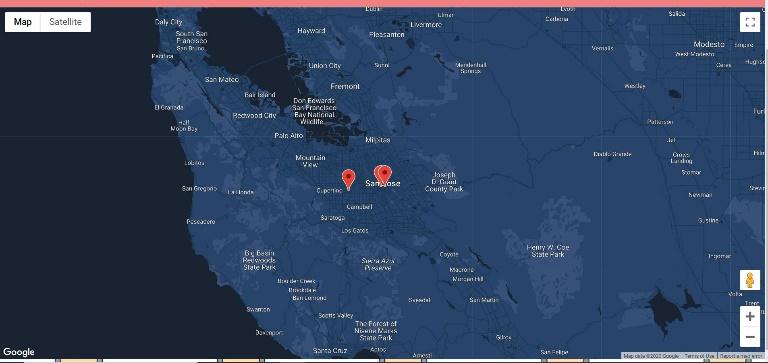
The web application was created using React-Redux framework.It talks to the backend server through https protocol.

Attaching some of the screenshots from the app:

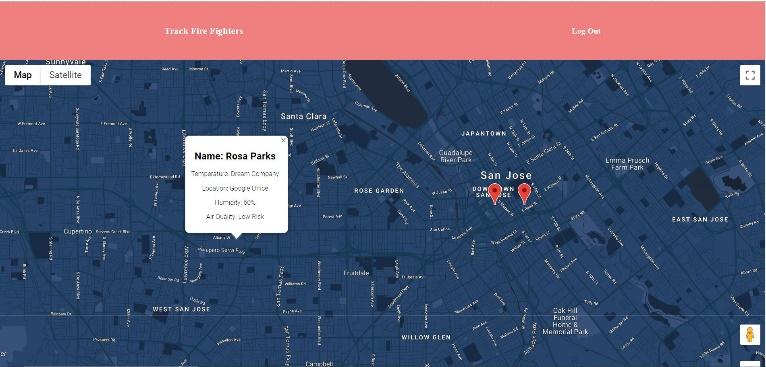
Login Screen/Sign up Screen:

**

FireFigher Details Screen

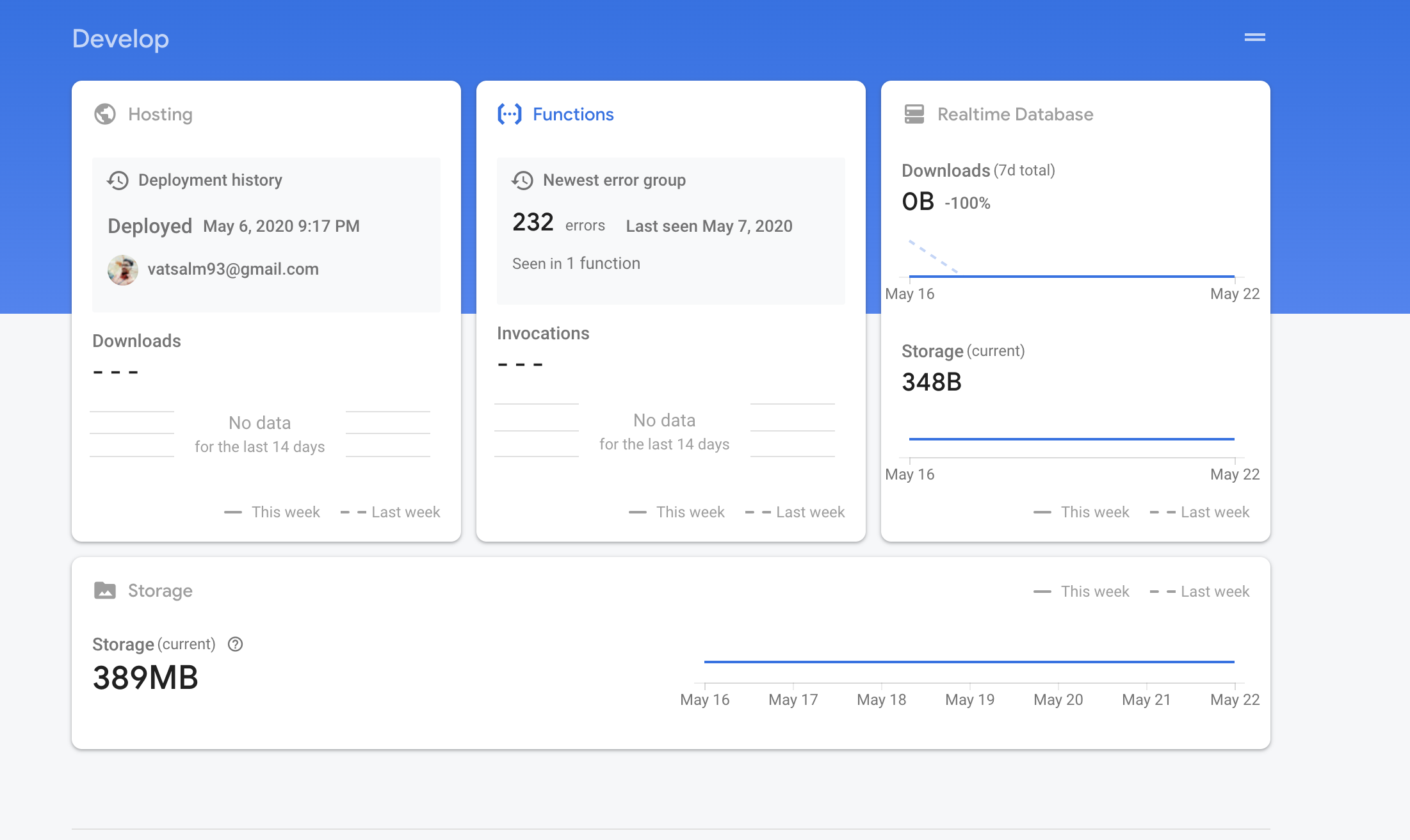
**

Firefighting details

**

**HOSTING**

The frontend and backend is hosted in Google Cloud system.This is a snapshot of our usage of the google cloud.



Once hosted, we can monitor our usage through

