Milestone 2

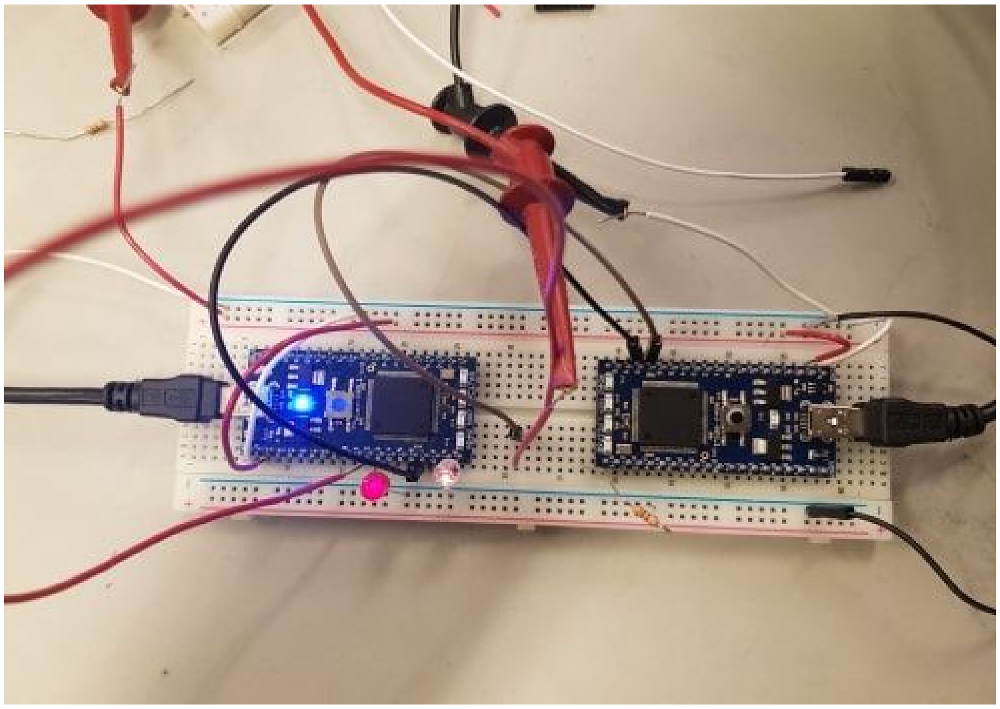
(Muhammad Fahad Asghar (masghar), Max Brown(maxbrown),

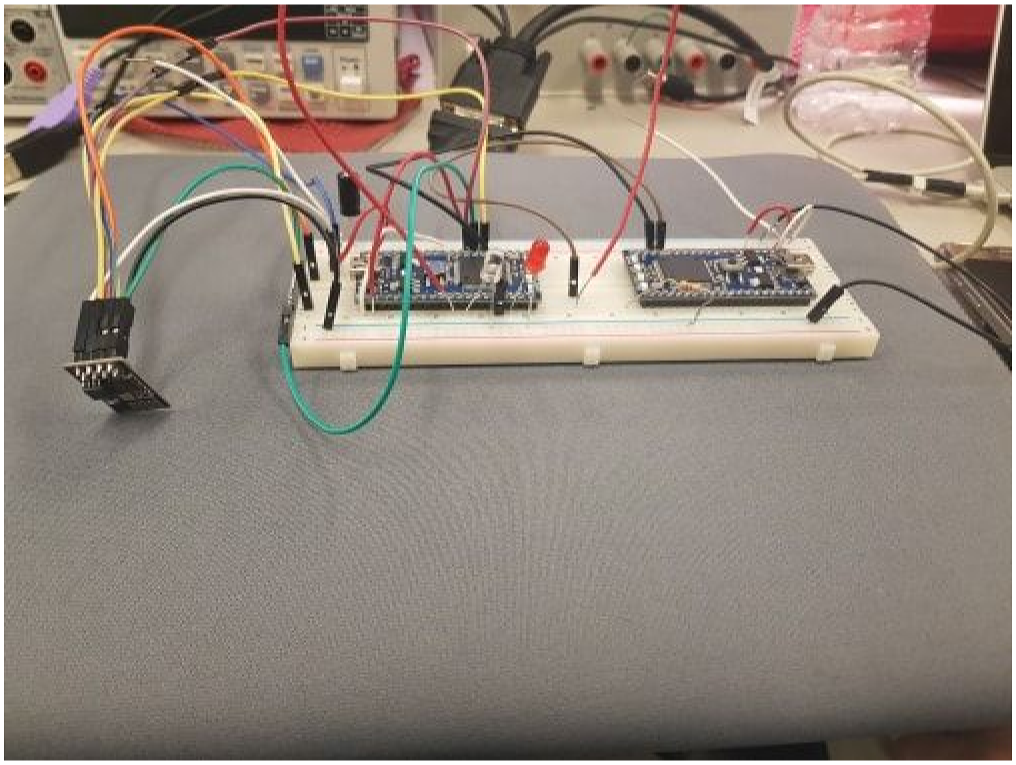
Henry Garant(henrygar), Kwame Owusu(owusua))

General overview

In this milestone we do a model based software implementations by following:

1. Designing the heart model and pacemaker along with the heart monitor model using threads on the same mbed from our UPPAAL model after verifying queries.
2. We publish the average heartbeats and alarms on the cloud using MQTT. The cloud application stores them and show them on a webpage (explained below).





Since we are also monitoring two abnormal heartbeat scenarios, we look at them using the two LEDs which indicate if any of the two scenarios have taken place. LED1(red) for the first scenario and LED2(transparent) for the second scenario.

Pacemaker and the Heart Monitor

In this file we generate the code for the pacemaker from our Uppaal model alongside the heart monitor. We start off by setting the default rate interval as the LRI value. This is followed by the declaration of guards for the VRP, pacemaker function to start and the LEDs that indicate the abnormal heart-beat scenarios. Setting all the values that we had in our Uppaal model, eg Sense, Pace count etc as 0 to start the working of the pacemaker. After this we declare the semaphores to prevent deadlocks and so that the resources are not being used simultaneously.
This is followed by the declaration of RTOS timers for the waiting of VRP, intervals, the heart rate monitor and resetting our statistics. After this we initialize our polling timer to check if the heart had sent a ‘sense’ signal. This is followed by the method *poll_input* to check if the heart mbed had sent out a sense signal. This is followed by declaration of different methods to calculate the heart rate for the given window. This is called every PERIOD seconds which we in our code are specifying as 60 seconds. This method for the monitor sends the data to the MQTT server and gives us a signal on the mbed. Within tis we check for conditions such as if the pacemaker is being called two often we send an alarm and set the red Led as on and if the heart is beating too fast, ie rate is greater than the URL (window/1.8). Here 1.8 is our maximum pulse width. If this condition holds true we again Alert and send a signal to the Led 2,

which is the transparent LED. Following this we define a method, *pace_timer,* which is called once our timer expires (when maximum wait time defined above is exceeded) and we need to pace the heart. In this we check if the if we are waiting for the VRP or not. If we are not waiting we send a signal to the heart, start the VRP timer and clear the pace signal and update our statistics that we defined in the start (Look at the code). This is followed by another method *wait_VRP_timer(),* which is called when the VRP expires and we have to start a new heart cycle. It sets *waitVRP* and *paceStarted* to false and by using the semaphore, set *stopVRP* to true and increment the heart beats

The while loop has two purposes, and it executes only when waitVRP time has been exceeded, that is, the pacemaker is now ready to wait for a heartbeat or if enough time has elapsed, send a pace signal to the heart. The other role is two flash the alarms if an irregularity has occurred, such as the heart beating too quickly or a high frequency of pace signals were sent.

<u>Heart Model</u>

We start off by defining a boolean value - *beatAgain* to make the heart beat again and set the RTOS timers again to track the polling pace and our heartbeat. This is followed by the declaration of a method *getHeartBeatTime()* that gives us a randomly generated heartbeat interval between the minimum 100ms and the max 1900ms values defined. After this we are defining a polling method *poll_pace* that checks to see if the pacemaker has sent a signal every *POLLING_INTERVAL*ms. After this the *heartBeat* method sends a signal to the pacemaker telling it that the heart is beating normally. We set the beat vale to 1 which we send out to the pacemaker via DigitalOut and then sets the guards setting the *beatAgain* as true and then releases it.

The heart's main loop continuously checks whether beatAgain is true. If true, then the heart is ready to beat. A function returns a random time between 100ms - 1900ms and starts the heartbeat timer with that interval. beatAgain is set to false, ensuring that there is only one beat per interval. Finally, after a quick wait, the beat voltage is set back to 0.

<u>Connecting the Two Mbeds (Communication)</u>

We were able to make the two mbeds communicate using the *DigitalIn pace* and *DigitalOut beat* on both Mbeds. This process is also defined along the way in the code explanations and the general overview of the different parts of the code above.

Cloud application

Architecture Flow

Data Collector: Mbed Heart Rate Monitor

Data Ingestor: Google Cloud Pub/Sub for MQTT

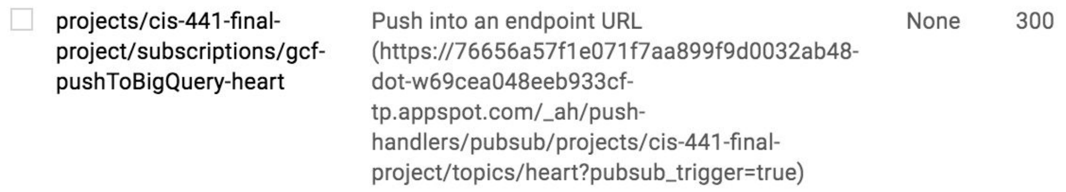
Data Pipeline: Google Cloud Pub/Sub → Google Cloud Function →
Google Cloud BigQuery → Google Data Studio(frontend)

Frontend: Google Data Studio -> https://cis-441-final-project.appspot.com/

Architecture Explained

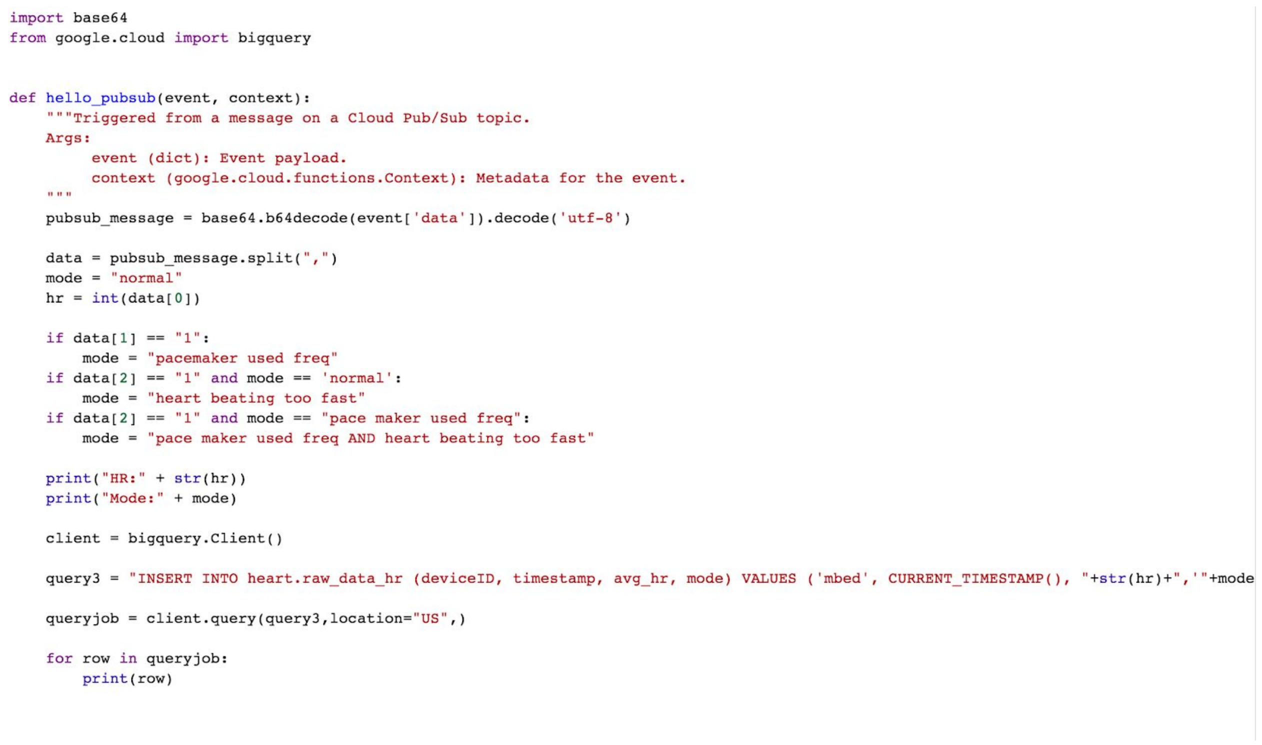
Data Collector Overview: Average Heart Rate data and pacemaker mode are sent from the heart rate monitor simulator running on mbed. The data is sent via MQTT publishing to the data ingestor.

Data Ingestor Overview: Google Cloud Pub/Sub for MQTT was used to create a MQTT broker. From there, we create a topic to publish and subscribe to monitor messages. The topic is *cis-441-final-project.* In this model, the data gets published to the topic via MQTT. Then, a subscriber(Google Cloud Function) receives an MQTT message.

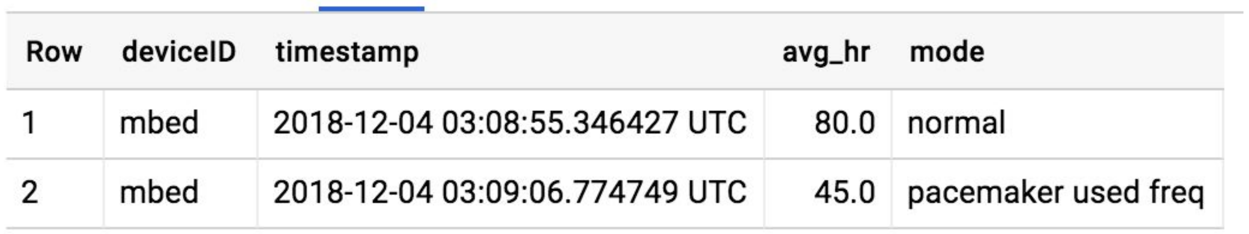


☐	projects/cis-441-final-project/subscriptions/gcf-pushToBigQuery-heart	Push into an endpoint URL (https://76656a57f1e071f7aa899f9d0032ab48-dot-w69cea048eeb933cf-tp.appspot.com/_ah/push-handlers/pubsub/projects/cis-441-final-project/topics/heart?pubsub_trigger=true)	None	300

Data Pipeline Overview: Google Cloud Pub/Sub receives an MQTT message and triggers a Google Cloud Function. The Google Cloud function parses the message and uploads relevant data to storage, BigQuery. The structure of data for the table in BigQuery is also below. Once the data is updated from the MQTT message, the data is presented in Google Data Studio. The embedded graph of Google Data studio can be found on the frontend at https://cis-441-final-project.appspot.com/.

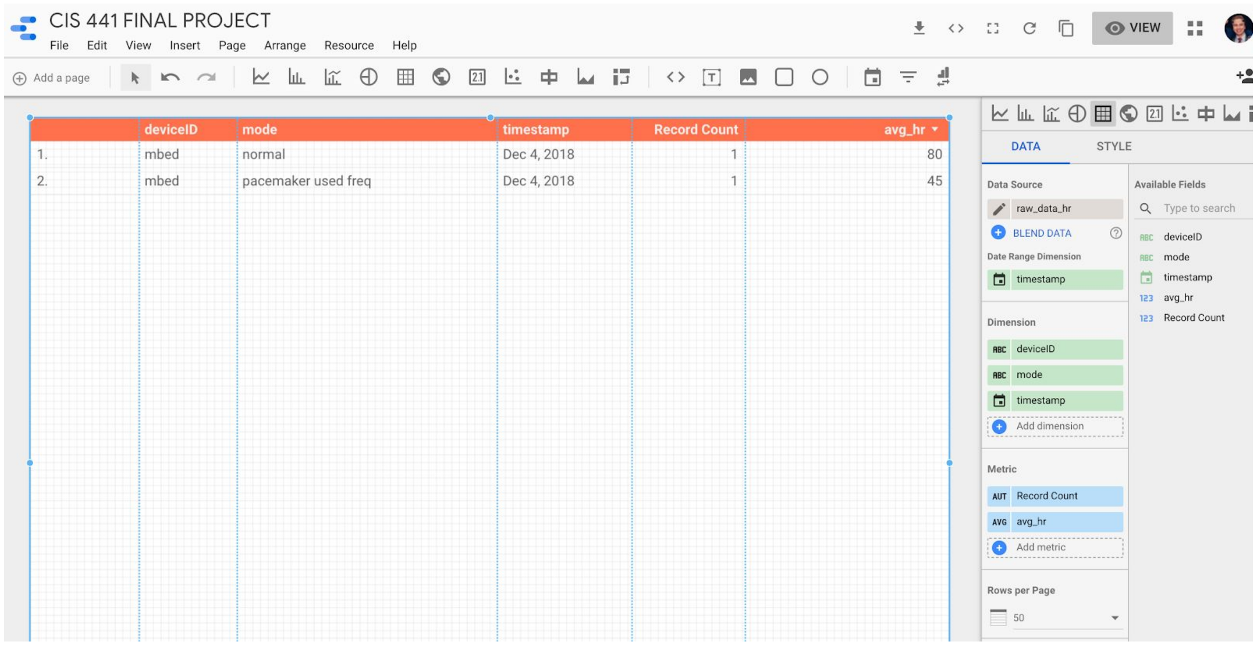


```python
import base64
from google.cloud import bigquery


def hello_pubsub(event, context):
    """Triggered from a message on a Cloud Pub/Sub topic.
    Args:
         event (dict): Event payload.
         context (google.cloud.functions.Context): Metadata for the event.
    """
    pubsub_message = base64.b64decode(event['data']).decode('utf-8')

    data = pubsub_message.split(",")
    mode = "normal"
    hr = int(data[0])

    if data[1] == "1":
        mode = "pacemaker used freq"
    if data[2] == "1" and mode == 'normal':
        mode = "heart beating too fast"
    if data[2] == "1" and mode == "pace maker used freq":
        mode = "pace maker used freq AND heart beating too fast"

    print("HR:" + str(hr))
    print("Mode:" + mode)

    client = bigquery.Client()

    query3 = "INSERT INTO heart.raw_data_hr (deviceID, timestamp, avg_hr, mode) VALUES ('mbed', CURRENT_TIMESTAMP(), "+str(hr)+",'"+mode

    queryjob = client.query(query3,location="US",)

    for row in queryjob:
        print(row)
```



Row	deviceID	timestamp	avg_hr	mode
1	mbed	2018-12-04 03:08:55.346427 UTC	80.0	normal
2	mbed	2018-12-04 03:09:06.774749 UTC	45.0	pacemaker used freq

Frontend: Google Data Studio allows easy graphing of BigQuery table data. Using Data Studio, an embedded graph is created that can be displayed on any webpage. The frontend web page is served using Python 3.7 and Flask framework.

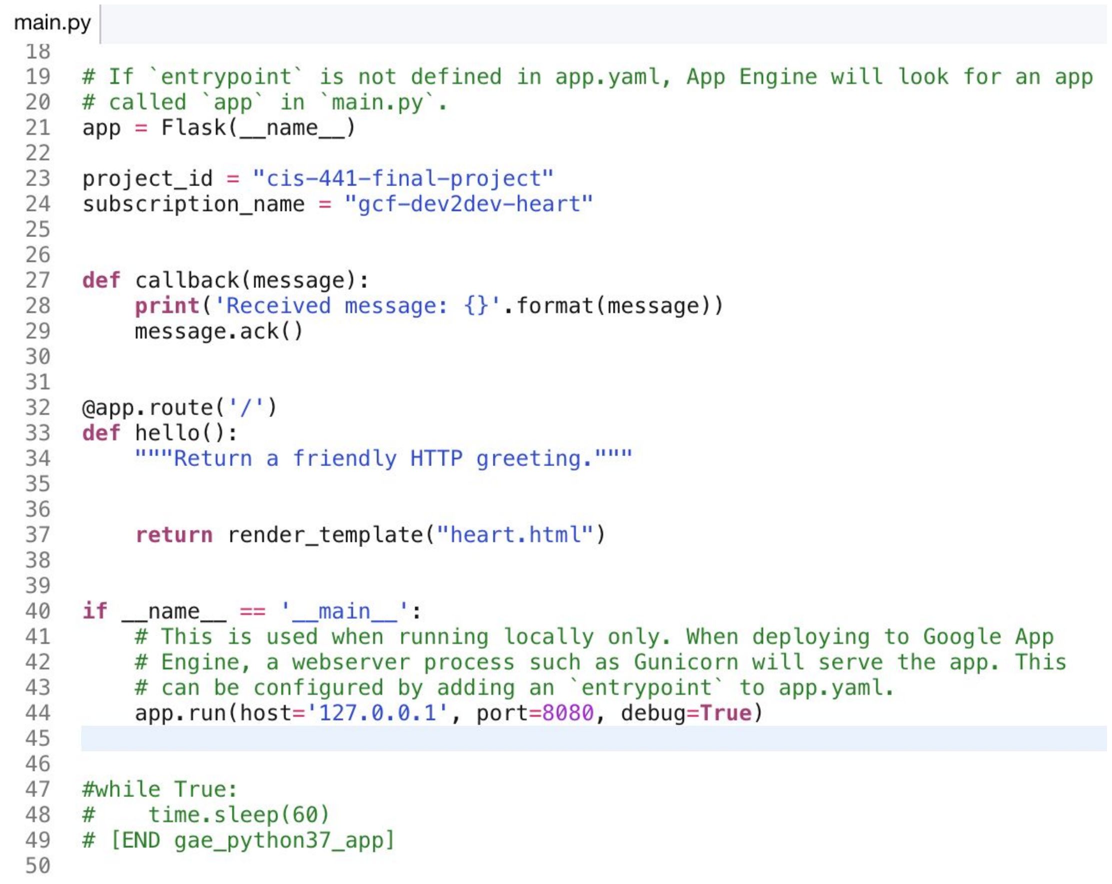


	deviceID	mode	timestamp	Record Count	avg_hr
1.	mbed	normal	Dec 4, 2018	1	80
2.	mbed	pacemaker used freq	Dec 4, 2018	1	45

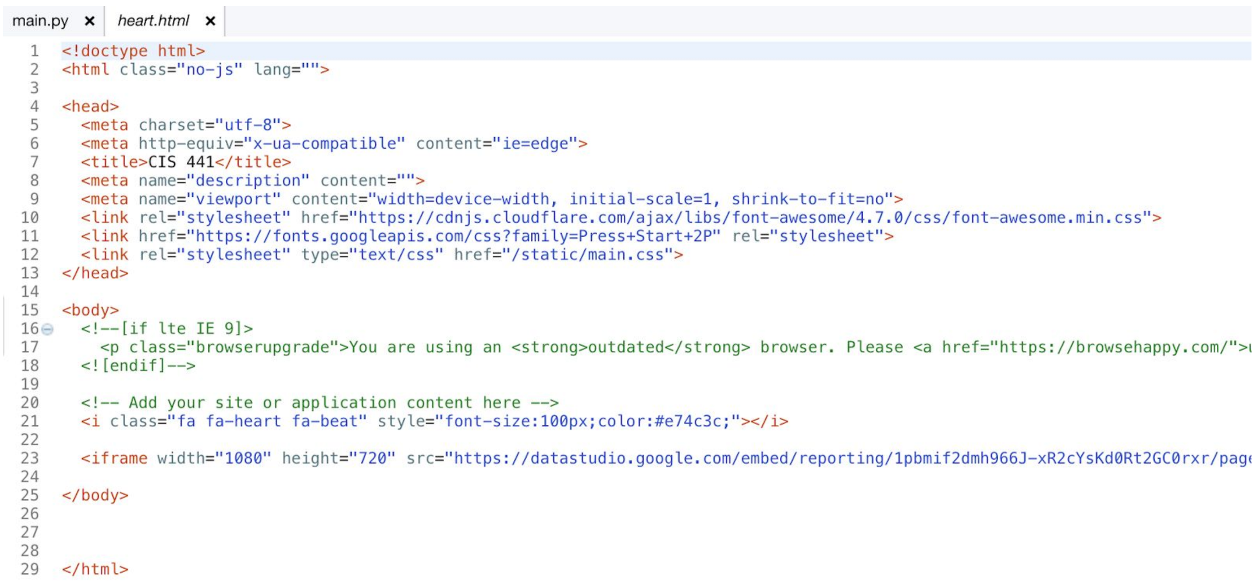
Data Source: raw_data_hr
BLEND DATA
Date Range Dimension: timestamp
Dimension: deviceID, mode, timestamp
Add dimension
Metric: AUT Record Count, AVG avg_hr
Add metric
Rows per Page: 50

Available Fields: deviceID, mode, timestamp, avg_hr, Record Count

Frontend Code:

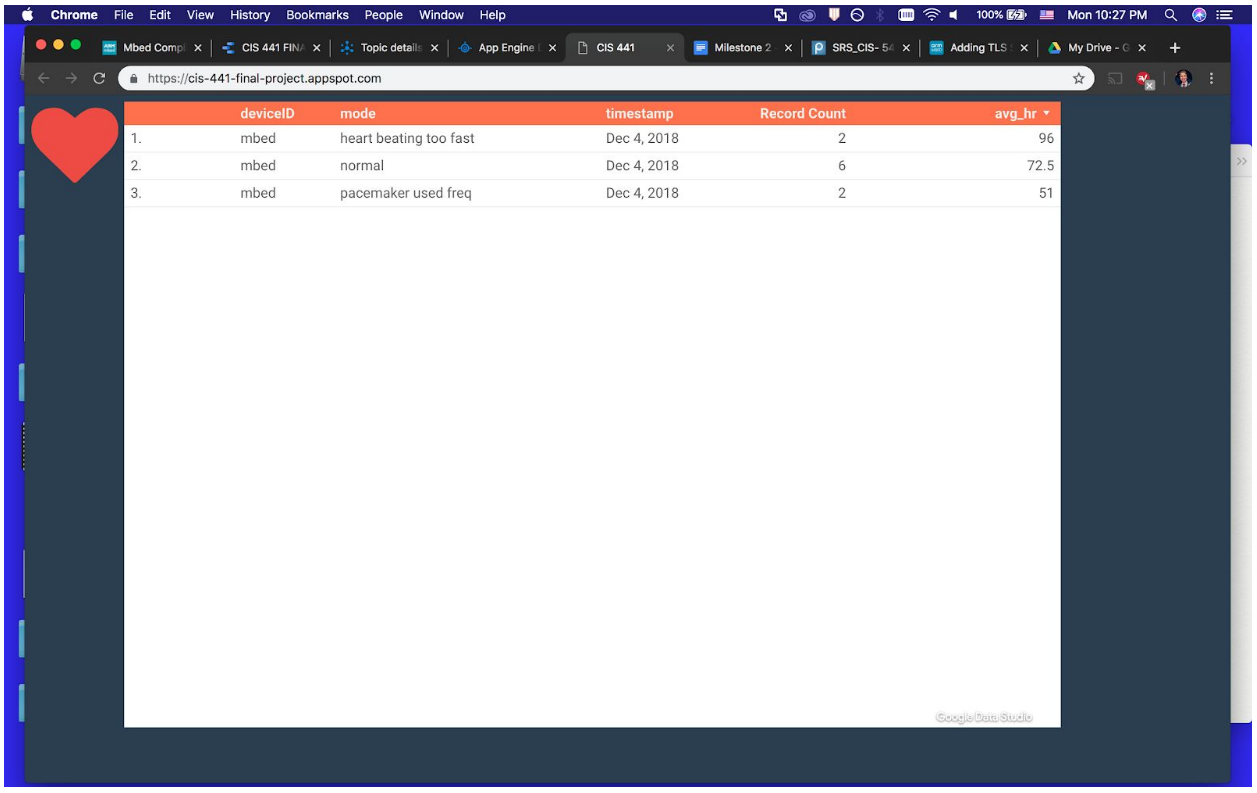


```python

# If `entrypoint` is not defined in app.yaml, App Engine will look for an app
# called `app` in `main.py`.
app = Flask(__name__)

project_id = "cis-441-final-project"
subscription_name = "gcf-dev2dev-heart"


def callback(message):
    print('Received message: {}'.format(message))
    message.ack()


@app.route('/')
def hello():
    """Return a friendly HTTP greeting."""


    return render_template("heart.html")


if __name__ == '__main__':
    # This is used when running locally only. When deploying to Google App
    # Engine, a webserver process such as Gunicorn will serve the app. This
    # can be configured by adding an `entrypoint` to app.yaml.
    app.run(host='127.0.0.1', port=8080, debug=True)


#while True:
#    time.sleep(60)
# [END gae_python37_app]

```



```html
<!doctype html>
<html class="no-js" lang="">

<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <title>CIS 441</title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <link href="https://fonts.googleapis.com/css?family=Press+Start+2P" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="/static/main.css">
</head>

<body>
  <!--[if lte IE 9]>
    <p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="https://browsehappy.com/">
  <![endif]-->

  <!-- Add your site or application content here -->
  <i class="fa fa-heart fa-beat" style="font-size:100px;color:#e74c3c;"></i>

  <iframe width="1080" height="720" src="https://datastudio.google.com/embed/reporting/1pbmif2dmh966J-xR2cYsKd0Rt2GC0rxr/pag

</body>



</html>
```

Final Result:



	deviceID	mode	timestamp	Record Count	avg_hr ▾
1.	mbed	heart beating too fast	Dec 4, 2018	2	96
2.	mbed	normal	Dec 4, 2018	6	72.5
3.	mbed	pacemaker used freq	Dec 4, 2018	2	51

Observe:

2 Alerts for Heart beating too fast

2 Alerts for PaceMaker required too frequently

6 normal heartbeats averaging to 72.5bpm