Criterion C: Development

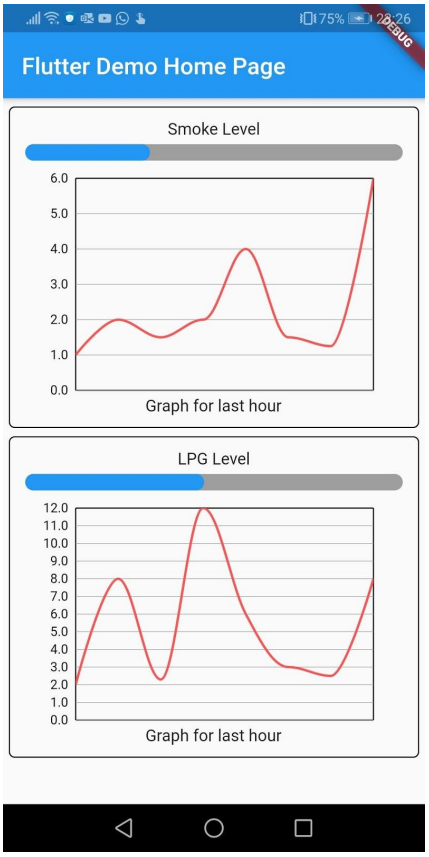
Techniques used:

- Dart (Flutter) coding
- Python coding
- Networking
- Databases
- Firebase server configurations

Dart Coding

For the Android App part of my project, the code was written using Google's Flutter framework for the Dart language, using Android Studio as the IDE. The app was then tested on a Huawei Honor 7X. GUIs were created by hand, as WYSIWYG editors for Flutter are not easily found.

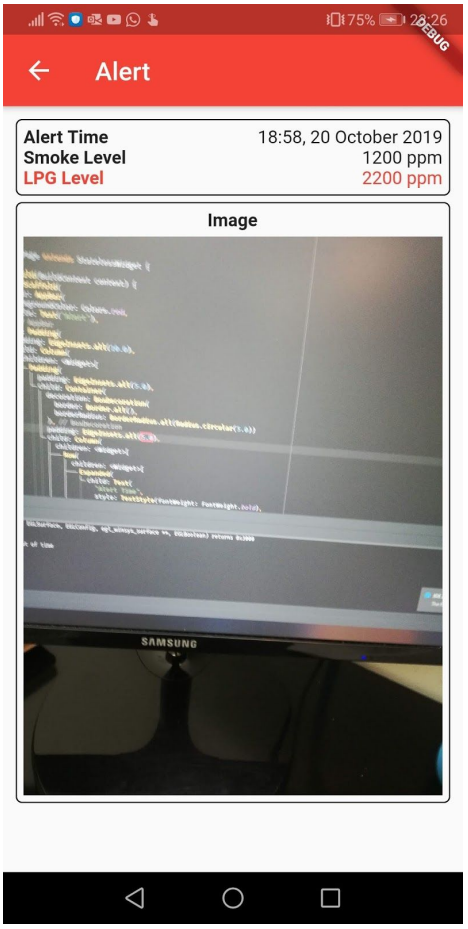
Main Screen



The main screen of the app simply provides graphs for the levels of Smoke and LPG in the room from the last 10 minutes (This was shortened from an hour as it would allow for one measurement to be taken every second as opposed to every minute, without storing too much data on the database), with the data being pulled from the database (on the Firebase servers) whenever the app is opened.

Alert Page

The alert page of the app shows up whenever the device receives a notification of high Smoke or LPG levels. This page gives the numerical values of those, and also provides a picture taken on the Raspberry Pi.



Python Coding

The code running on the Raspberry PI is written in Python. This code mainly serves to obtain data from the sensors connected to the device, and push it to the database. The data is pushed using a Python library that allows for Firebase Databases to be manipulated, after providing the required credentials. This also includes the code that checks if the levels of Smoke and/or LPG are unsafe and, if they are, instructs the Firebase servers to send a notification to the phone (using a similar library), causing the alert screen to show up. At this point, an image from the attached camera is also taken, which is then uploaded to the Firebase storage. The alert screen on the app can then pull the image from there.

The primary difference between the Python code that was planned and the code that was actually used was that the used code ended up being mainly procedural due to the format in which data was obtained from the sensors, and the format in which data had to be sent to

the server, as well as the fact that it turned out to be the best way to handle the requirements of the device.

Networking

The device, servers, and companion app all connect to each other over the internet, which is what allows the device to work in a way that the user is notified wherever they are, as long as they have an internet connection. The app can also utilize push notifications because of this, which means that instead of the user periodically needing to check or refresh the page, the app will simply notify the user of unsafe levels LPG/Smoke in their home.

Code pushing notification

```
if value > 135 or value2 > 135:#This block sends a notification to the client when High levels of LPG are detected, and uploads a photo as well
    if value > 135 and value2 > 135:
        title = "High Smoke and LPG detected"
        body = "Smoke level of {}\n".format(value) + "LPG level of {} detected".format(value)
    elif value > 135:
        title = "High Smoke detected"
        body = "Smoke level of {} detected".format(value)
    else:
        title = "High LPG detected"
        body = "LPG level of {} detected".format(value2)

    result = service.notify_topic_subscribers(topic_name="global", message_body=body, message_title = title)
```

Code uploading image:

```
camera.start_preview()
    time.sleep(2)#Camera needs time to startup
    camera.capture("image.jpg")

    bucket = storage.bucket()#This block uploads image

    img_dat = open("image.jpg", "rb").read()
    blob = bucket.blob("lastalert.jpg")

    blob.upload_from_string(
        img_dat,
        content_type = "image/jpg"
    )
```

Databases

The measurements taken by the device are stored in a database on the Firebase servers. They are split into 3 collections. The first one stores all measurements from the previous 10 minutes - used to create the graphs on the main screen. The second collection stores only the last measurement taken - used for the current level indicators above their respective graphs, and the third stores the last alert sent out, so that it can quickly be accessed as soon as the user opens the app.

Code to get data from database

```
Future<List<List>> getData() async{//Gets data from database
 final databaseReference = Firestore.instance;
 List<List> lst = [<FlSpot>[],<FlSpot>[]];

 var stuff = await databaseReference.collection("datapoints").orderBy("time").getDocuments();
 stuff.documents.forEach((f){
   Timestamp time = f.data["time"];
   print(time);
   int smoke = f.data["Smoke"];
   int lpg = f.data["LPG"];

   int diff = (now.seconds-time.seconds);
   lst[0].add(FlSpot(diff.toDouble(), smoke.toDouble()));//adds point to graph
   lst[1].add(FlSpot(diff.toDouble(), lpg.toDouble()));//adds point to graph
 });
 lst.add(await finalDP());
 return lst;
}
```

Code to push data to database

```
db = firestore.client()
collection = db.collection(u'datapoints')#Database collection storing the last 10 minutes' data

othercol = db.collection(u'laststuf')#stores final datapoint

alertcol = db.collection(u'lastalert')#stores last alert

counter = 0;

bus = smbus.SMBus(1)
while True:
    bus.write_byte(address,A1)
    value = bus.read_byte(address)
    print("Smoke", value)#Obtains Smoke value

    bus.write_byte(address, A0)
    value2 = bus.read_byte(address)
    print(value2)#Obtains LPG Value

    collection.document(str(counter%600)).set({
        u'Smoke' : value,
        u'LPG' : value2,
        u'time' : datetime.now()
```

```
})#Adds value to database, overwriting the value from ten minutes ago

othercol.document(u'main').set({
    u'Smoke' : value,
    u'LPG' : value2
})#Sets last datapoint to same
```

Structure of product

The reason the system with one Raspberry Pi detection device connected to a server and a database which in turn connects to a companion app was chosen to allow for a connection between the user's phone and device indirectly that would work no matter where the user was, as explained in Criterion B.

It may also be helpful to summarize what is happening every time the device takes a readout from the sensors, especially as during the design process it was found that the solution could run more efficiently by following a slightly different order of operations to that proposed in the flowchart from Criterion B. The sensors picked were the LPG and and Smoke sensing ones from the MQ series of gas sensors. These provide a continuous analog output based on the level of their respective substance they detect, which was then run through an analog to digital converter to allow the Raspberry Pi to interpret it. The Raspberry Pi then takes this data and uploads it to the database on Google's Firebase platform. If the readouts are below the threshold values, the process stops there. However, if it detects a high level of either LPG or smoke, a photo is taken with the camera, which is then uploaded to a preset storage URL on the Firebase servers. The device then sends a request to the server instructing it to send a notification to the client's phone with a specific message.

When the phone app is opened, it requests all previous readings from the database and generates the two graphs on the main screen. If there is an alert, when the alert page is opened it pulls the last alert readout from the database and displays those readings, and also pulls the image from the preset storage URL and displays that.