**Topic:** Smart Home Office

**Author:** Benchley André (安本林)

**Student ID:** 0616336

**Objective**

* LED turns on when you’re in the desk vicinity. You can toggle the LED from a website, a desktop app, and with an on-site switch.
* The first time the LED is turned on during the day, a notification will be sent to the user letting him know what the weather will be like during the day.
* Fan turns on when the temperature exceeds a preset temperature value.
* An automatically generated email with the top headlines.
* A full control over the IOT system from the website and desktop app.

**Specs**

The Smart Home Office counts a total of 4 automations. The first one is the LED turning on or off depending on how close the user gets to the sensor. This automation can also be deactivated, letting the user manually turning the LED on or off. The second automation is a notification sent to the user, the first time the light turns on during the day. We have to note the it does not matter if the light was turned on automatically or manually. The third automation is the fan turning on or off depending on the temperature in the room. The fan used for this project is experimental, hence the size. A bigger fan can be used, given an appropriate power source is used as well. The last automation is a periodically sent email containing the top headlines in whatever the user is interested in.

The set of hardware devices used for this project is as followed:

* Laptop (for the Desktop app), smartphone (for the MCS mobile application)
* Raspberry Pi 4b
* HC-SR04 (Ultrasonic Ranging Module)
* DHT22 temperature and humidity sensor
* DC5V Fan
* LED x1, Resistor x1, Push button x1, NPN transistor x1, Wires

**System Design**

The smart home office system comprises of 3 main parts. The first part is the device where the LED, the resistor, the push button, the wires and the sensor are connected to the raspberry pi. The second part of the system is the cloud, where the LED state is displayed and controllers can be used to change the LED state on the device site. The third part is the desktop app which allows the user to control the state of the LED, and make some specifications about how the notification is sent if it is to be sent.

Desktop App

Device

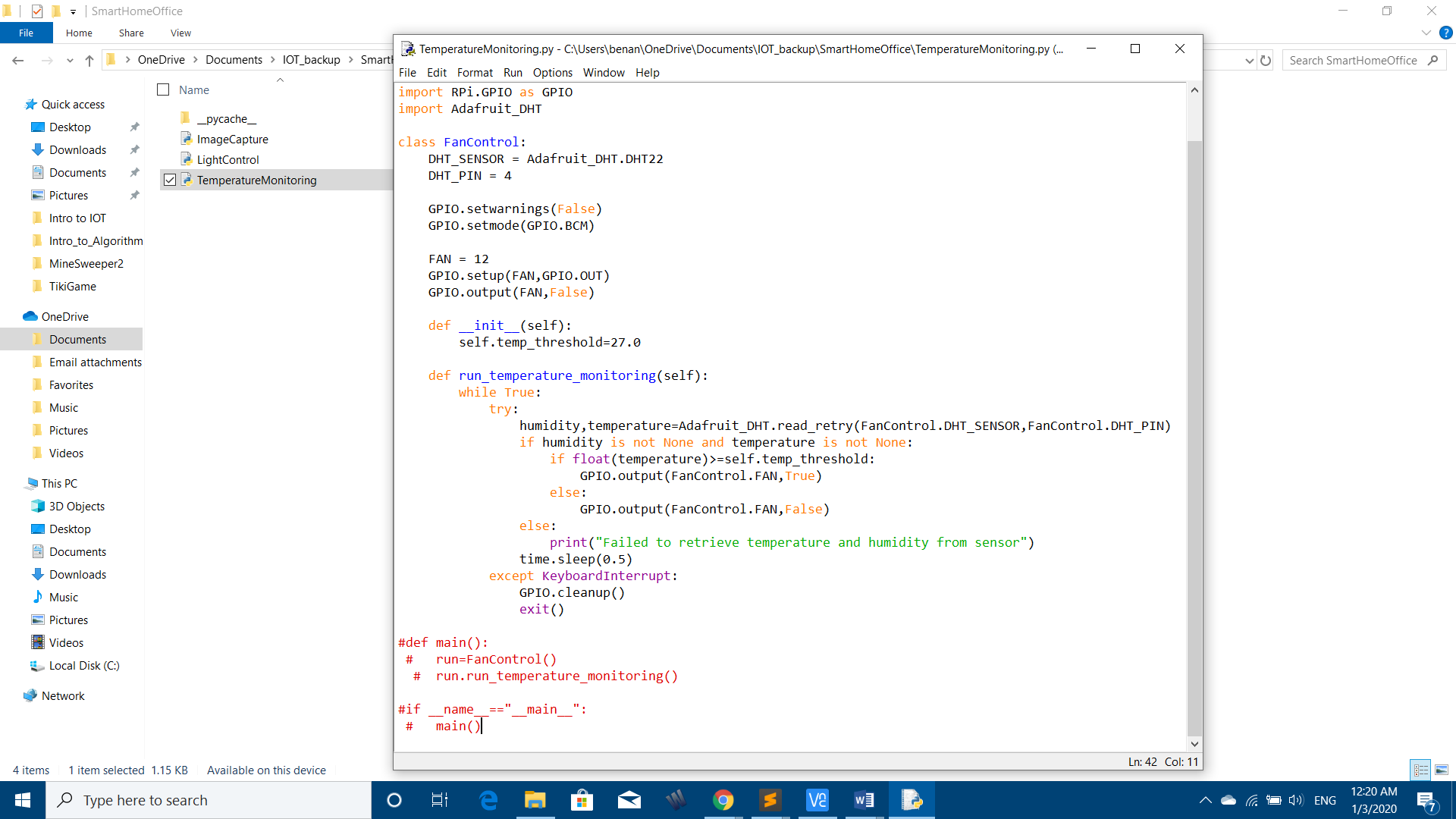
MCS

The above graphical representation shows how each part of the system can change values of other parts of the system while the device is being used. However, while the device updates its state every few seconds using a while loop, the desktop app only updates manually using a refresh button. One of the reasons why this design is used is because the MCS cannot handle a lot of requests for a certain amount of time. Consequently, the desktop application doesn’t update every time a value is changed on the MCS. Also, the light on-delay automatically get its value from MCS whenever the settings window is opened on the desktop app. The desktop app can modify values on the MCS and as a result on the device. For example, the click of a button on the desktop app sends a request to the cloud to change one of the controller’s state, which will turn the light on, as the device constantly sends request to the cloud to get the controllers’ states.

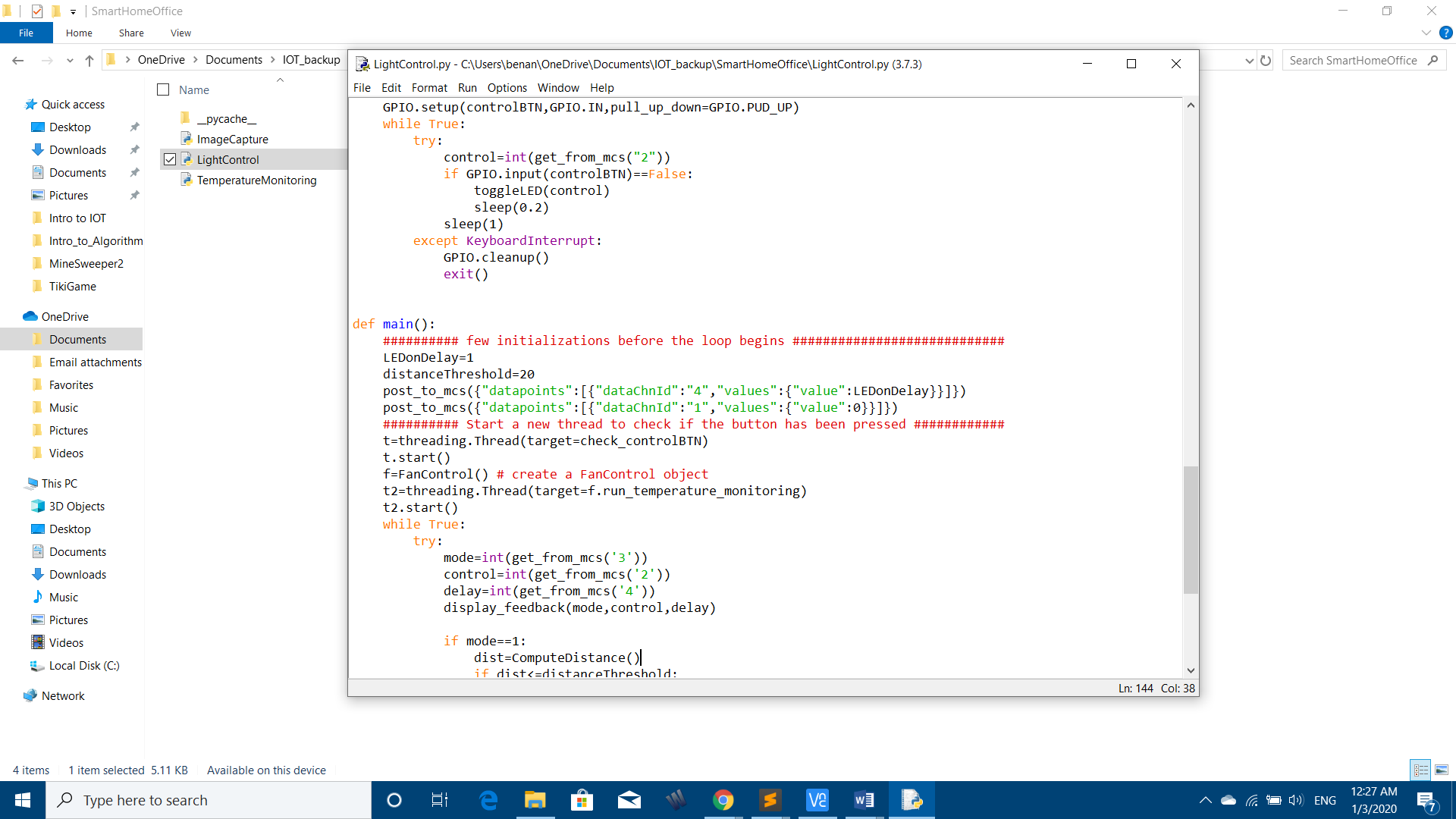
However not all the programs on the device communicate with the cloud. The fan control program remains purely on the device. It does neither get nor post anything to the cloud. The main reason is because the programming is light enough so the device can do the computing by itself, and also the value of the temperature and humidity need not to be stored.

**Module Implementation**

On the device side, we use two python scripts, “LightControl.py” and “TemperatureMonitoring.py”. In “LightControl.py”, we import all the necessary modules, including “TemperatureMonitoring” from which we use the “FanControl” class. That class implementation is as followed.



An instantiation of that class is ran on a different thread alongside 2 other threads. The main one, which is responsible for the LED toggling, and the other one responsible for the toggling of the push button.

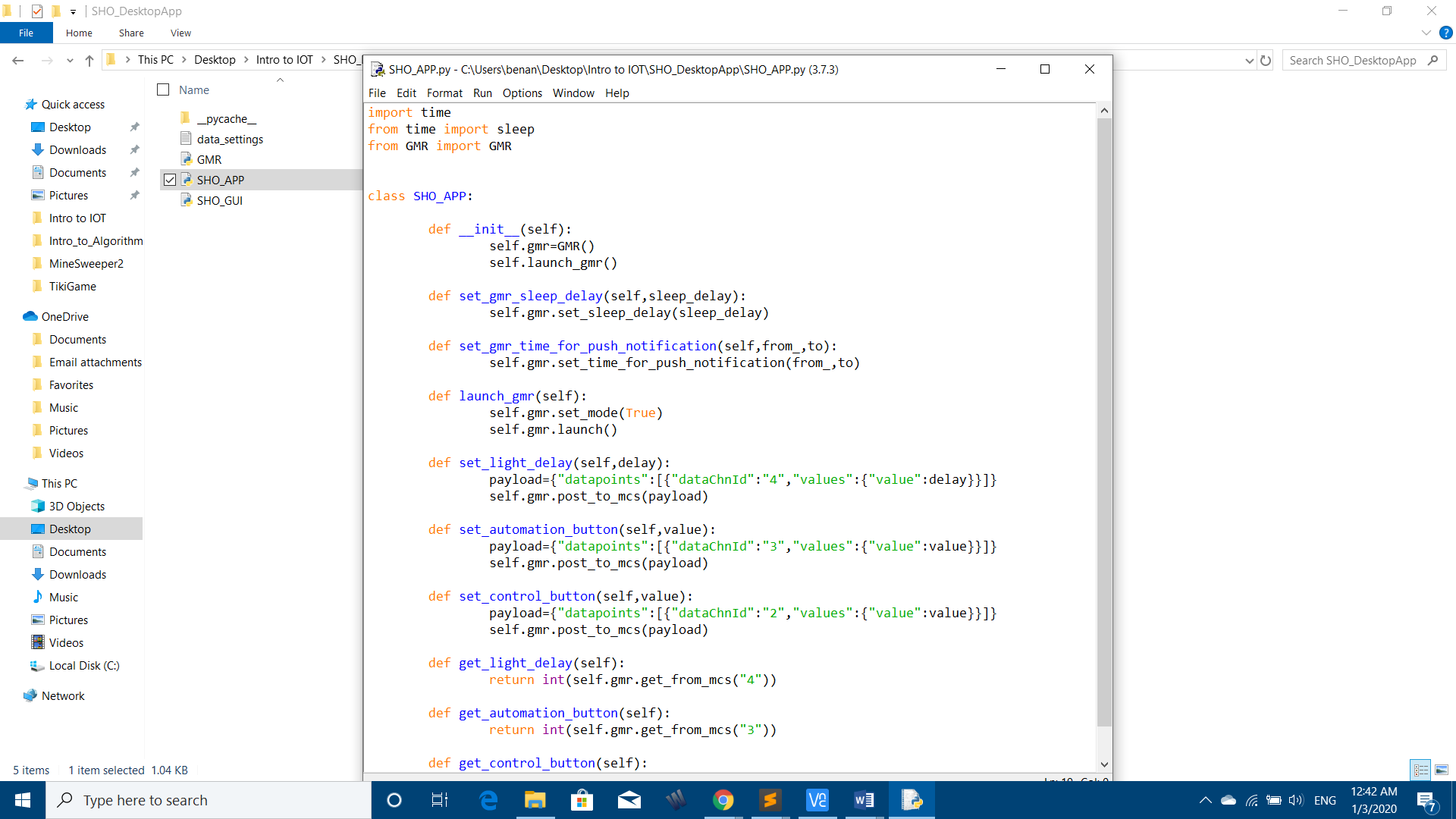


For the desktop app, we created a module called GMR (short for Good Morning Routine, as the original idea behind the notification is that it occurs essentially in the morning). That module contains a class GMR which sends a string to the MCS when some criteria are met. Another module SHO\_APP.py, instantiates an object from the class GMR in other to make further operations easier. SHO\_APP.py will be used in our desktop application SHO\_GUI.py, to send specific requests to the MCS and modify specifications of the GMR object.

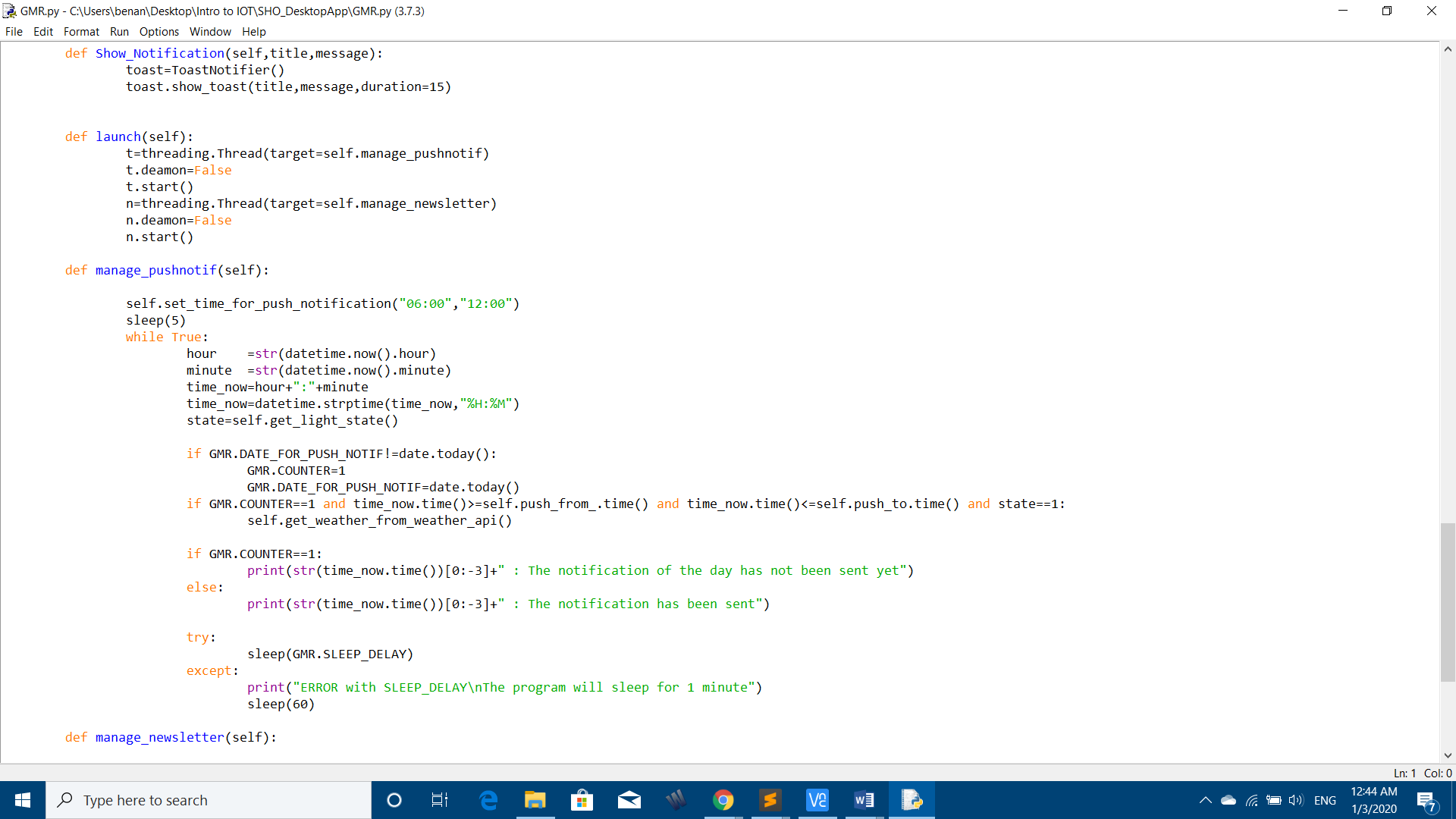
The GMR class send requests to 3 APIs, MCS, Open Weather and News API. The main purpose of this program is scheduling the push notifications and the emails sent by MCS. The push notification and the email parts are handled in different threads.

When a SHO\_APP object is created, it automatically creates a GMR object that launches the 2 threads.

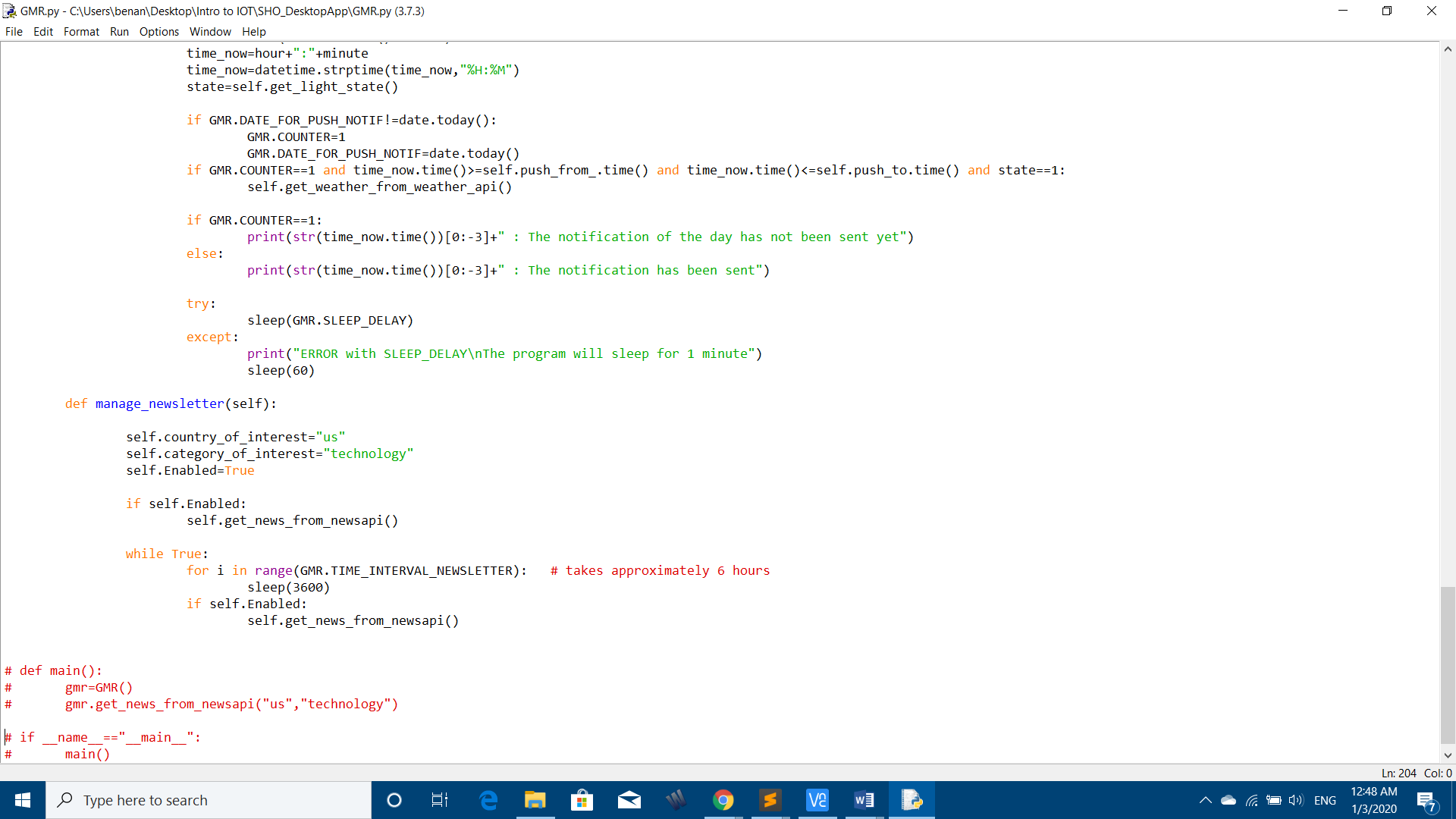
In SHO\_APP.py



In GMR.py



Handles the push notifications

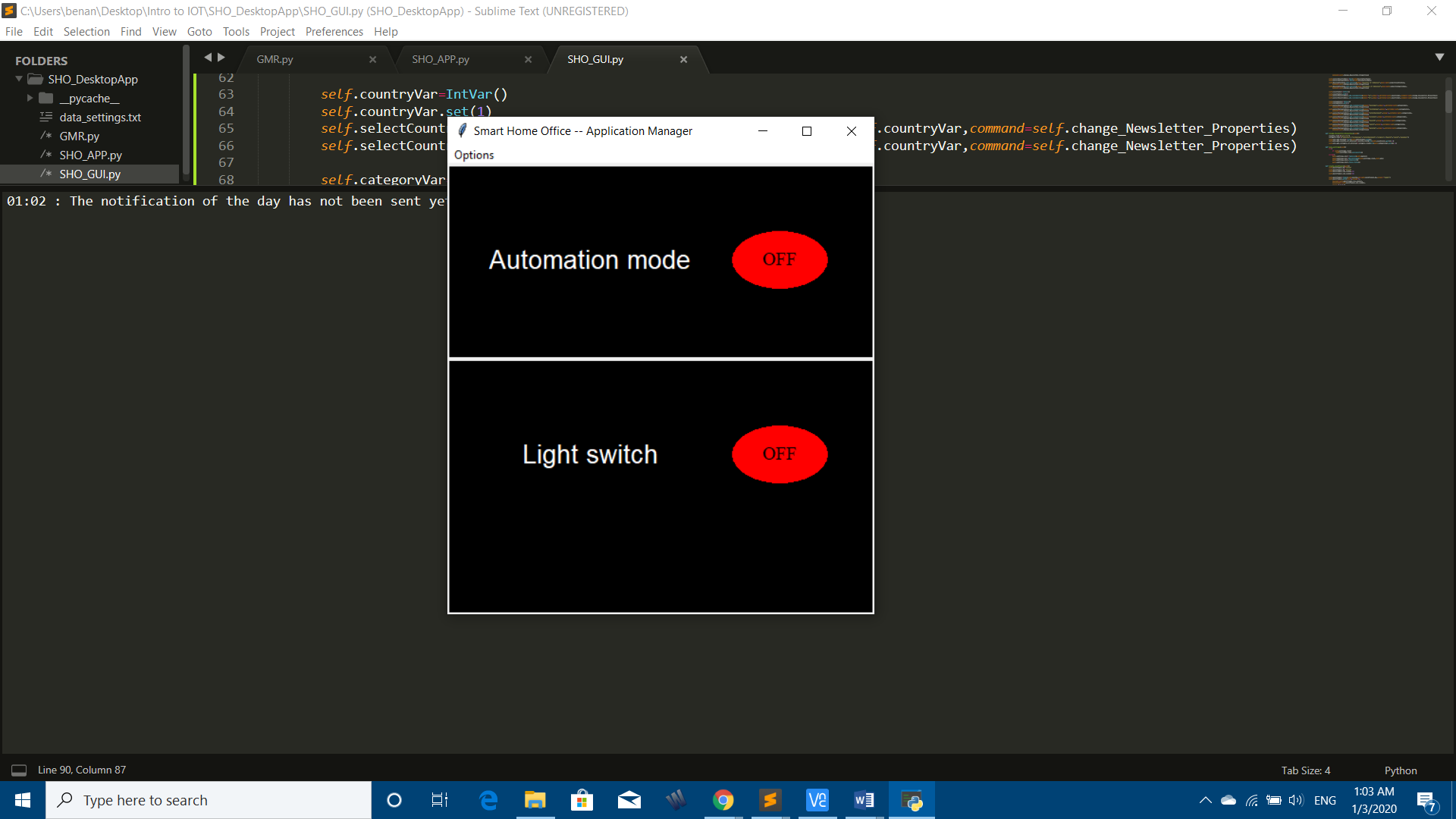
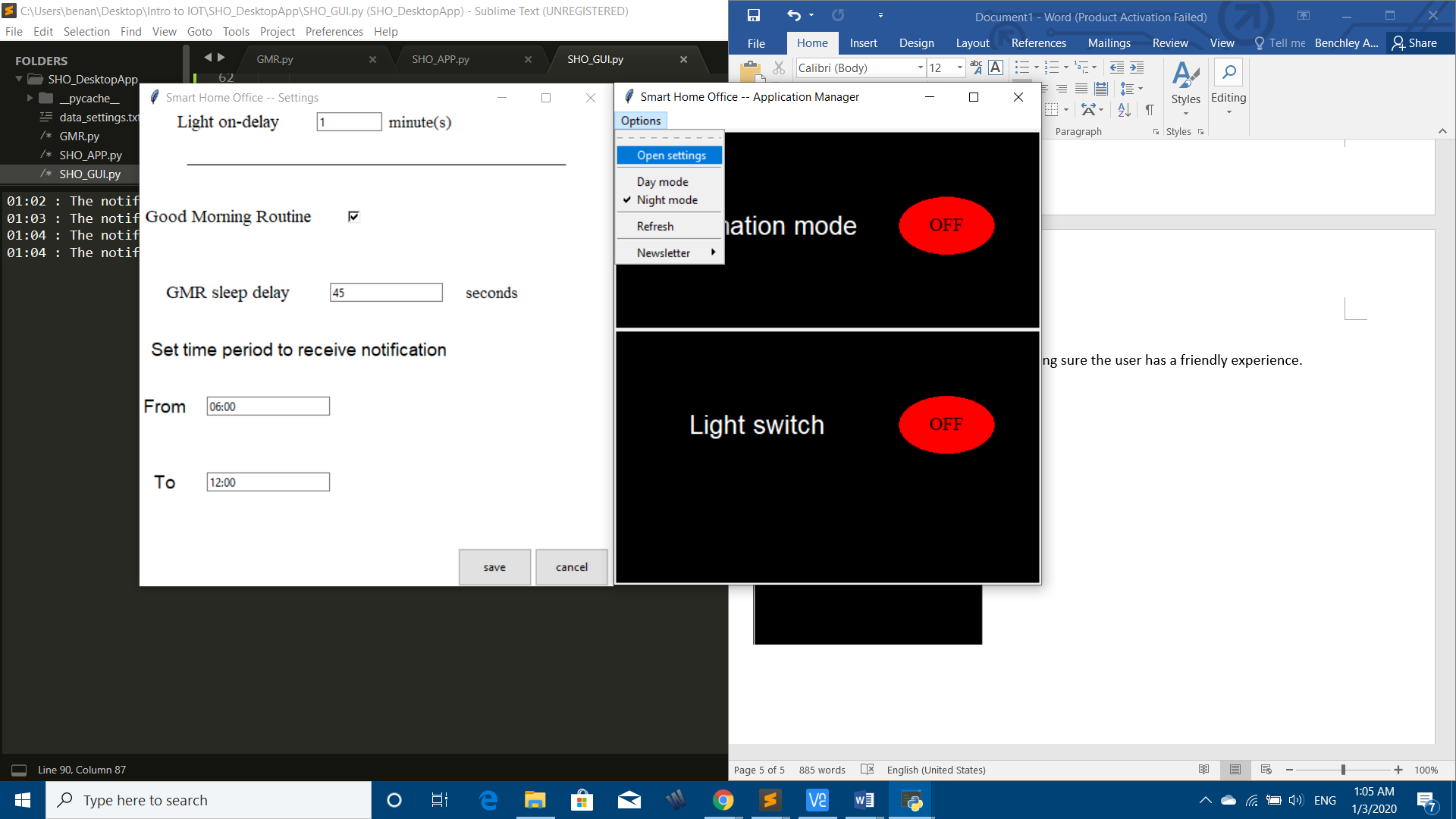


Handles the email forwarding

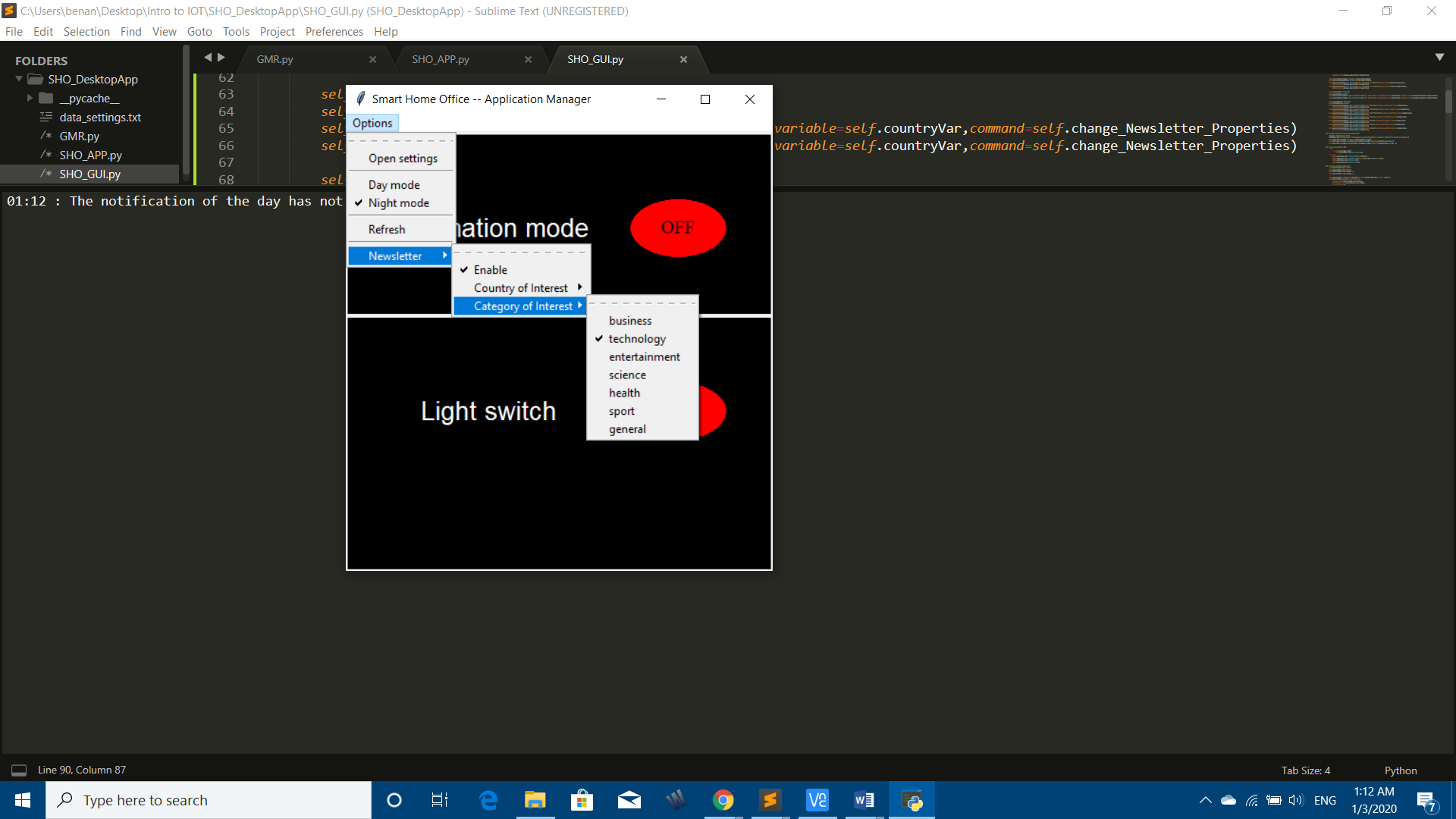
**Demonstration**

The desktop app has a graphical interface making sure the user has a friendly experience.

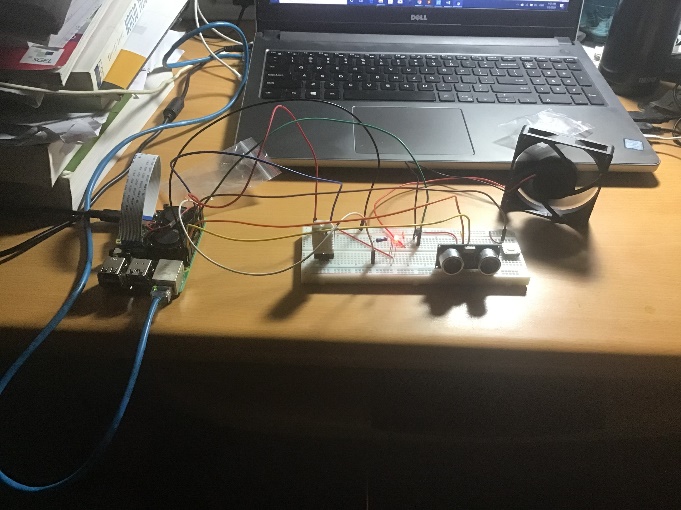
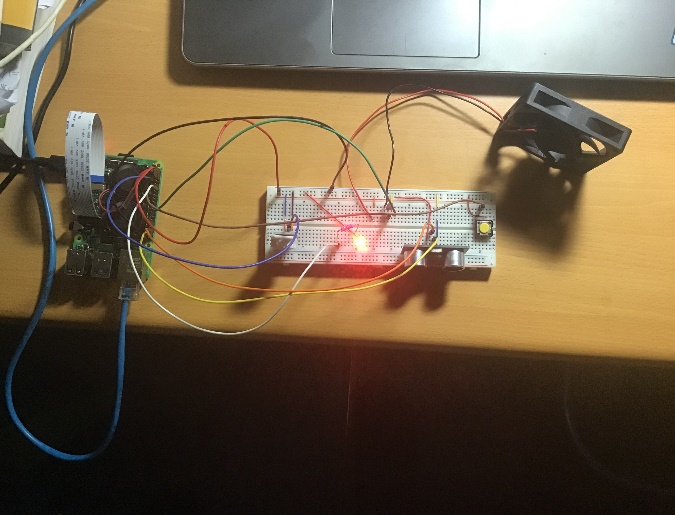
Open the settings window

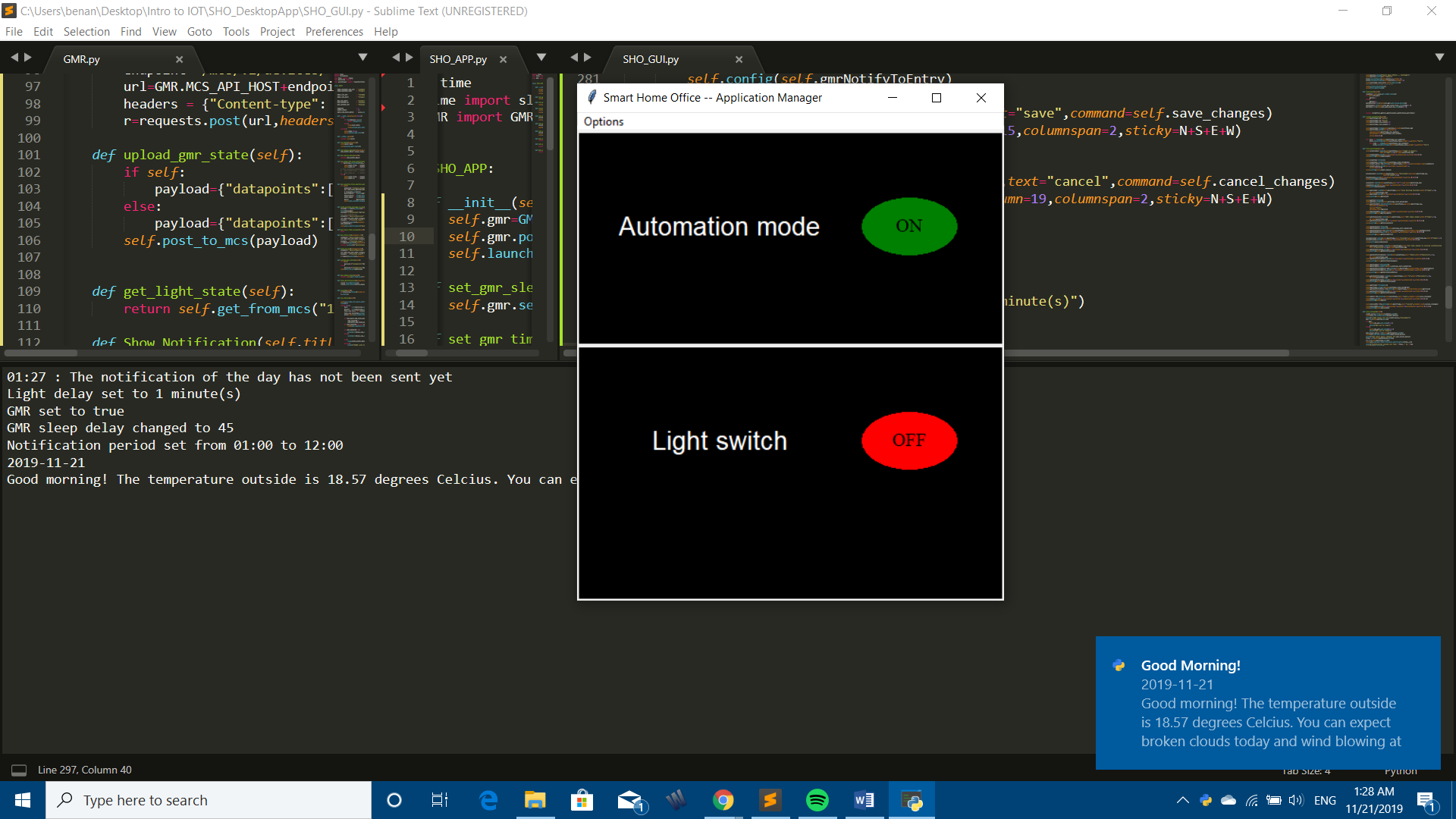
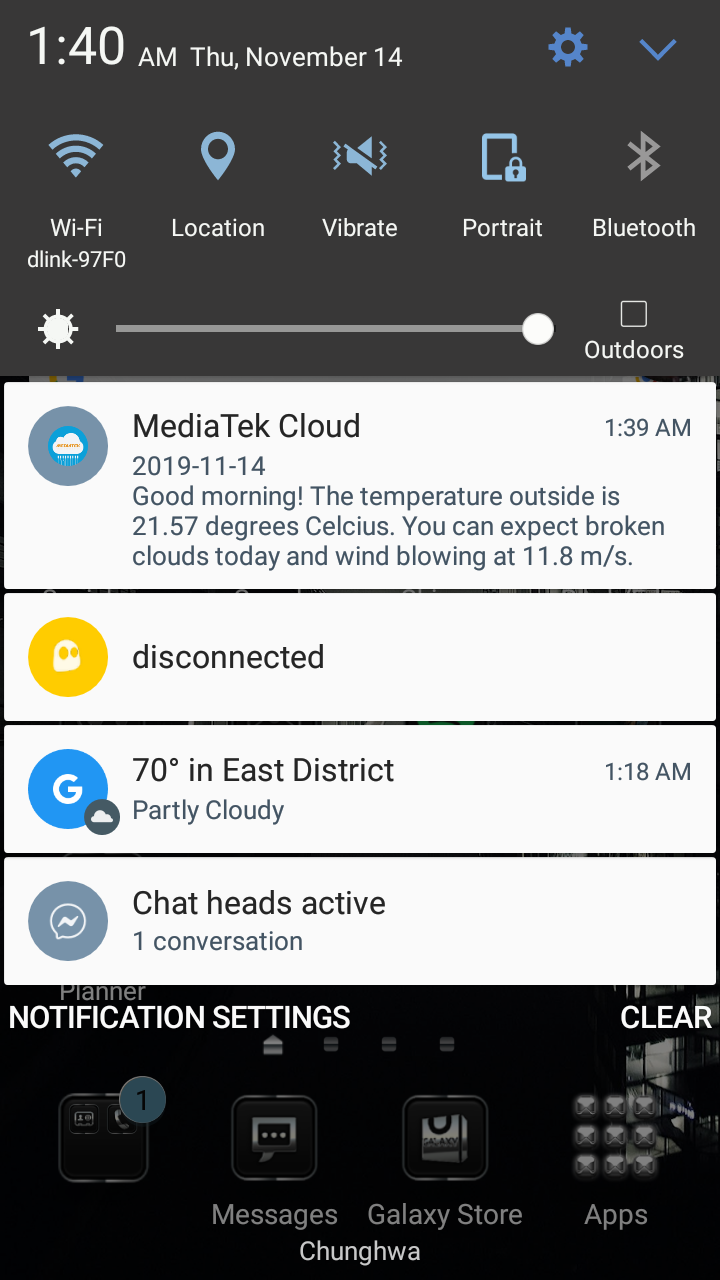
Change preferences for the news email

As we can see, the user can go ahead and select his country and category of interest. The push notifications are sent on a daily basis while an email is sent approximatively every 6 hours.

The whole system works perfectly. Below are a few photos proving the successful implementation of the IOT system.

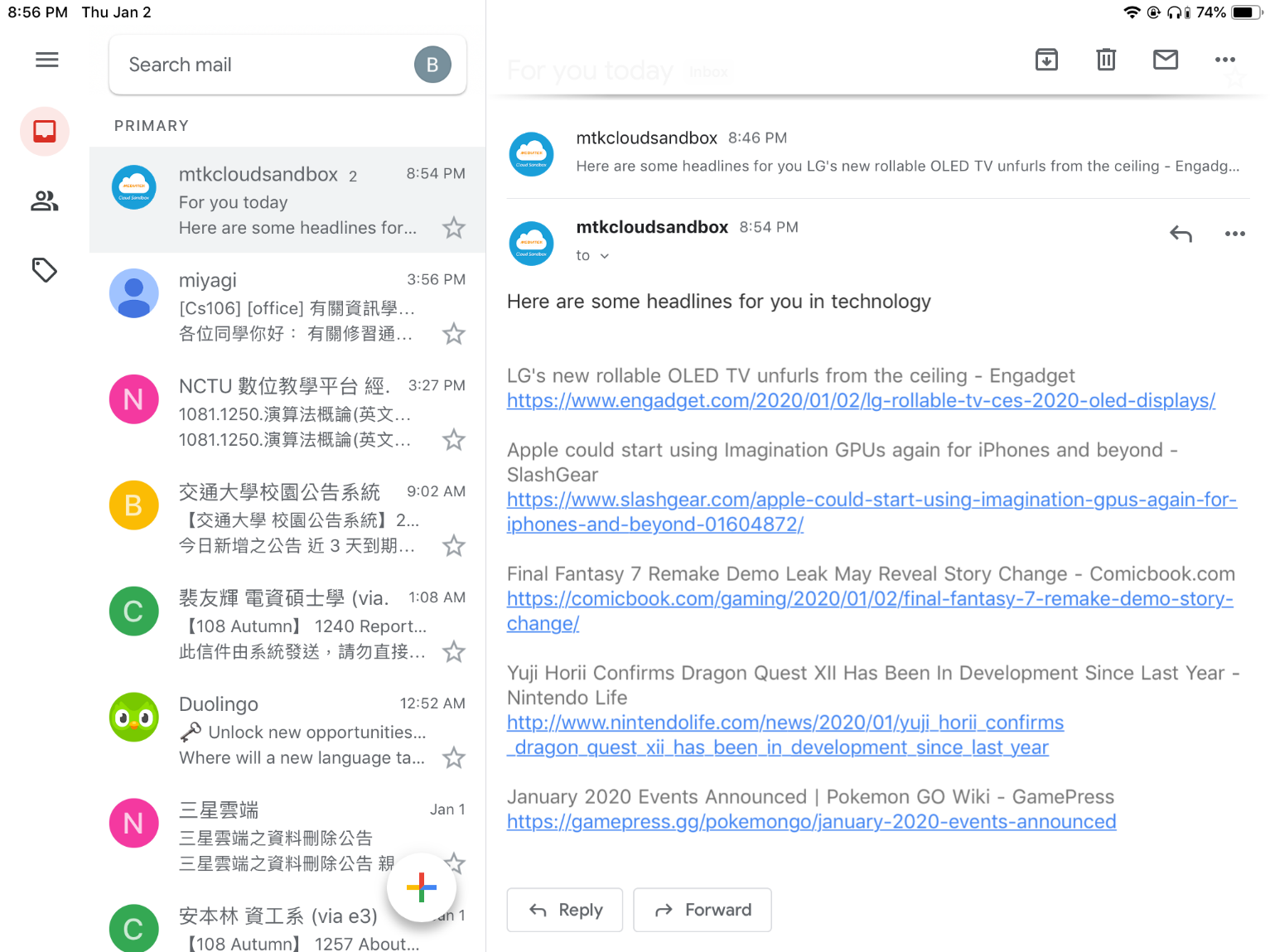


In the photo above, the LED is turned on automatically first, and in the second picture, using the desktop application. When the temperature exceeds a predefined threshold, the fan also turns on.



On the left is the notification sent on the user’s smartphone, and on the right side, is the one accompanying the desktop application.

Below is the email sent to the user containing a few headlines. The user can then click on the url if he’s interesting in any particular article.



**Reference**

* [www.sunfounder.com](http://www.sunfounder.com) (for the setup of the ultrasonic range module, and the python piece of code computing the distance of an object to the sensor).
* Source code provided by the TAs.