Year 1 Semester 2

Course Title  
Software Design with Artificial Intelligence for Cloud Computing

|  |  |
| --- | --- |
| Individual reflection  29/03/2022 | Project title  Build a connected system between Raspberry PI, MIT App Inventor and Flask  Natalia Palej  Mobile Apps & Connected Devices |

# Introduction

In the final project I was tasked to build a connected system between Raspberry PI and MIT App Inventor. We can achieve that by using Python build-in library called Flask. The objectives were to create a phone app with buttons, which after pressing, respond to Raspberry PI sense-hat 8x8 display and reads its sensors.

# Specs

I created 11 buttons and one text box. Four buttons change colour of all 64 sense hat pixels, but prior to the colour change, it clears the screen to make sure there is nothing displayed on the screen. One button displays a message of smiley face, another button has a function which displays an image of heart, three other buttons are used to read Raspberry PI sensors (temeprature, humidity and pressure), one button resets sense-hat display. Last button moves to a second screen which consists of group members information.

The aim of additional buttons (heart, smiley face, team information) is to show knowledge that I have gained throughout first year of college by using different methods to call desired messages. By that, I show the capability to define a function along with successfully calling and displaying it. Throughout the year, we were using Python build-in commands to show\_message on the sense-hat, which was also implemented in the code. This shows the potential and how powerful Python programming language is.

# Python Code

At the very start, I have commented out libriaries that I am not using in our program. Since it is a template that I have re-used from previous classes, I have decided to leave them in, to show my ability of re-coding existing files and keeping only what is being used.

Text, letter

Description automatically generated

Afterwards, I created instances of SenseHat and Flask. Once that was done, I decided to declare the colours that will be used in our program.

Text

Description automatically generated

Then, I proceesed to created app routes for temperature, humidity, and pressure. In each of them, I am initializing the variable to read appropiate sensor value, and then rounding it to two decimal places. Since we are also displaying the equivalent number of pixels on sense-hat, I created temp\_value which rounds the temperature to no decimal places. At this point, we figured the pixels will go out of boundry if the temperature exceedes 64 – which is the maximum of sense-hat display. This made us realize, we must implement appropiate calculations and apply range in our code.

Following that, I initialized the sense-hat display to light up red with the equivalent number of pixels to temperature and the rest of the display to white. Next, I am using set\_pixels build-in command, which will light up the sense-hat. Lastly, I am creating return message that will be displayed in text box. To achieve this, sensor values must be casted to String. That line of code informs the user of the temperature that is rounded up to two decimal places.

Text

Description automatically generated

After that, I proceeded with app route for humidity. Here, I had to implement extra line of code which calculates the value accurately to the 8x8 display. Since humidity is represented in percentage, we are multiplying the number of available pixels (64) by the humidity value and then diving it by 100. Later, just like with temperature, I am setting pixels to light up blue with equivalent number of pixels to humidity and white for the remaining range. After setting the pixels, I have added the return message to appear in text box, which informs the user of humidity and clearly shows it is present in percentage.

Text, letter

Description automatically generated

The very same steps have been implemented for pressure function. The only difference would be how is the sense-hat range calculated. By using Trinket, it was noticed that the maximum hPa pressure available on the emulator is 1260. To make it work on 8x8 matrix display, we divided it by 20, which returns 63. That means, the pressure value will never go out of sense-hat boundries. I have set pressure pixels to light up green and the remaining of range white.

Text, letter

Description automatically generated

Afterwards, I am defining app route to accept multiple actions. These if statements are checking which button have been pressed and what action should be displayed. To each if-statement I have added return message with appropiate action. I have done that to avoid long html code in text box, that otherwise was being displayed. It is also a good practise for testing purposes, since the programmer can check whether correct action is being ran after pressing the buttons.

A picture containing graphical user interface

Description automatically generated

Originally, I had else-if statements for temperature, pressure, and humidity actions. Once the testing was completed and successful, I realised these are unnecessary and redundand. The program was never going into any of these else-if statement, since it had its own, dedicated app routes, which were delacred above.

Timeline

Description automatically generated

To prove my point, I have printed a message in both, app routes and action route. This not only showed which route the program goes to, but also confirmed one of them, in fact, is never being used. Once that was tested, I commented out the print messages.

Graphical user interface, text

Description automatically generated

As a final step in the Python code, to run the code directly from executed code, I have added app.run that runs by using Flask. Port number may vary and was often needed to be changed when the code was stopped and ran again.



# Graphical user interface, text, application Description automatically generatedA screenshot of a cell phone Description automatically generated with medium confidenceMIT APP Inventor Interface

# MIT APP Inventor blocks

To be able to connect Raspberry Pi with the MIT APP Inventor, I needed to connect it to a local network (in this case, hotspot). I initialized global variable IP to the IP address, followed with port number (which was declared in Python code and must be the same in the code, as well as in the blocks).



By initializing our global IP, I did not have to change the port number multiple times, in every single block. Instead, when a button is clicked, I set the website to get an url of the global IP and after slash, the action is specified. We can achieve it by join text block. Once that is done, I call the Web1 to get the data.

A picture containing timeline

Description automatically generated

In the example above, the URL will look like this: http://192.168.43.171:5002/red, which will light up sense-hat display to red.

The below block sets the TextBox to appropiate message (it runs the “return” statement which was implemented to Python code). In the “red” example, it simply would display “Red” in the text box.

Graphical user interface, application, website

Description automatically generated

Text

Description automatically generated with medium confidence

Excatly the same steps were implemented for all other actions, with the difference of what shows up after slash symbol.

Timeline

Description automatically generated

I have created additional button that was desired to go to different screen and display team members infromation. The block below allows user to change between screens after pressing the button.

Chart

Description automatically generated with low confidence

# Chart Description automatically generatedA picture containing text, whiteboard Description automatically generatedTesting

# Conclusion

In this project I have learnt how to build a connected system between Raspberry Pi, Python and Flask. In fact, it was one of my favourite projects I have done in First Year. It was extremly interesting and fun to create.