Isabella Phung
June 5, 2025

Lab 7 Report

7.1:

For this section, I followed the professor's tutorial in lecture and managed to reproduce the same results.

7.2:

I had to rely heavily on ChatGPT for the rest of this project since I'm unfamiliar with creating web servers or working with JSON data. There was some back and forth, testing different ways I could make a web server before settling on Flask, since it seemed to handle POSTing and it seemed fairly simple. I did face some difficulty with actually being able to access the webserver at first, but after checking possible issues, it seemed like the firewall just needed to allow tcp to 1234 which I fixed using the ufw command.

After ChatGPT generated the code for the ESP32C3, I found that it used the "example_connect" function which was located in the esp-idf/examples folder. I wasn't sure how to include it as the ChatGPT code didn't do so and any examples seemed to have it somehow included in the build folders in a way that I admittedly didn't really understand. ChatGPT showed me code that might remove the need for example_connect, but I was worried that a variety of errors might crop up as a result. What I decided to try was copy one of the example folders that utilized this folder, copy my .c files into the main folder and adjust the CMakeList as needed, which happened to work to my relief.

After this, I tweaked my webpage thanks to ChatGPT's suggestions such that it would display the posted data. There was some back and forth as I didn't understand how the html and Javascript actually addressed data from the stored keys. After ensuring that my esp32c3 was sending *both* the humidity and the temperature data, this section worked as intended.

7.3:

I tried to ask ChatGPT for code on how to perform a GET request; the code for the webserver worked fine, but I found that the ESP32C3 would not return any values whatsoever. I searched around on the various examples provided and followed the "esp_http_client" example, specifically the http_rest_with_url function. Reading the code, I was confused as to how it actually stored data into the buffer. After many long hours, I got help from Professor Renau and we looked at another function in the same file, http_rest_with_hostname_path. He showed how it was the event handler that actually performed the specifics of the request, much to my surprise. I thought that the event handler was only for error checking. With this, I managed to get the location data from the web server onto the ESP32C3.

Next hurdle was integrating 7.2 into this project. I copied over the files from 7.2 and moved the app_main commands over to the 7.3 main file. When I compiled this project, I found that the binary for the project exceeded the default allocated partition space, so I checked the documentation on how to create a custom partition table, changed menuconfig to read the new file, and it worked fine after that.

The final part of this was majoritively string manipulation using strchr and strcat, in that the location data I received from the web server appeared like this: "{"location" : "Santa Cruz"}".

Once I got it into the proper format, I needed to send this value to the https reading function to create the proper url. I had some difficulty with constant NULL errors that would make my board restart. The error was a stack protection fault regarding the buffer variable I passed, which was exceptionally unusual as it worked fine before I started passing the location data into the function. Initially my buffer was allocated as size 512, but after reducing it to about size 64, the stack protection fault disappeared. This buffer was allocated in stack memory, so I theorize that the size of the buffer exceeded the allocated stack space for the function. With this, I made some small adjustments, ensuring that the values were passed and displayed correctly and my project was complete.