Kenneth Miller

CIS 331

Mr. Garrison

2021-11-23

Engineering Log

**Schematics:**

![A picture containing text, device

Description automatically generated]()

(Had to use a different model board and sensor due to TinkerCad limitations)

**Work Log:**

11/23 – 1.25 hours

Initial setup of ESP8266 and plugin installation.

Attempted to install Android Studio, failed and will not execute correctly.

Decided to use ESP8266 webserver extension instead.

Added needed headers to file, and setup ESP8266 board in code.

11/30 – 1.25 hours

WiFi attempts:

Tried to upload sketch, but was failing, found out that pins could not be connected to d3/d4 in order to upload sketch.

Tried using static IP/gateway to connect to webapp, but not able to connect.

12/2 – 1.25 hours

WiFi attempts:

Board was not connecting to my WiFi, had to remove the static ip address setup, and let it assign a dynamic ip.

Board now connecting but not showing html webpage correctly. Its keeping the connection closed

Switched to different server commands using ESP8266WebServer header file

12/9 – 1 hour

Using the mDNS responder for Esp8266 and get/post forms for the led toggle allowed functionality of the html page.

Added pushbutton to wiring, but html function to handle pushbutton will not post the state. Suspecting the wiring is wrong, using a pinout that is used for data transmission.

12/11 – 1.5 hours

Fixed button pinout, switched from D1 to D5 and it now works.

Removed ability to turn off LED from webapp.

Added magnetic sensor to wiring, webapp status and functionality to trigger LED’s.

Added magnetic sensor state serial printing if triggered.

Only button can turn off LED’s, added serial print for if button is pressed.

Finalized wiring diagram.

**Project Summary:**

The project started off rough with Android Studio not installing or running correctly. After two failed reinstallation attempts, I decided the best course of action would just be to attempt to use Arduino extensions for the ESP8266 board to create a webapp server and use CSS styling in HTML to design the page. This required server.send() commands within the Arduino code, but was a simpler solution to the issues I was having. It resulted in a less aesthetically pleasing app, but a functional one. Future improvements will include CSS styling improvements, as well as device-conditional width of the app to match all screen sizes. In addition, there will be added functionality to light control, and an additional stage of security where audio alarms will sound if prompted through the webapp. In researching LED control through ESP8266 web app functions, I found three websites that provided information on how to send http requests through clients and the server. Randomnerdtutorials and lastminuteengineers proved to be incapable of handling the http requests correctly, and it wasn’t until the mDNS responder was added that I was able to correctly send and receive HTTP requests. This allowed me to adequately communicate through the webapp and the ESP8266 board for hardware control.

Works Cited

<https://randomnerdtutorials.com/esp8266-web-server/>

<https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/>

https://tttapa.github.io/ESP8266/Chap10%20-%20Simple%20Web%20Server.html