(20102337) Kamil Pawlowski

20102337@mail.wit.ie

Abstract

The following discusses my assignment on my plant monitoring system, how it works, various results and challenges faced.

iot standards & protocols

Assignment Report

Contents

[Introduction 2](#_Toc207394978)

[System architecture diagrams 3](#_Toc207394979)

[Pictorial (block level) representation of project 3](#_Toc207394980)

[Informational flow diagram 4](#_Toc207394981)

[Flow chart of algorithm 5](#_Toc207394982)

[Code snippets 6](#_Toc207394983)

[Hardware Manager 6](#_Toc207394984)

[App 9](#_Toc207394985)

[Other features 16](#_Toc207394986)

[MS Azure 16](#_Toc207394987)

[Custom Vision 16](#_Toc207394988)

[Blynk 17](#_Toc207394989)

[Data collected 19](#_Toc207394990)

[Challenges faced 21](#_Toc207394991)

# Introduction

My assignment focused on creating an IOT device to help a user monitor and take care of a houseplant.

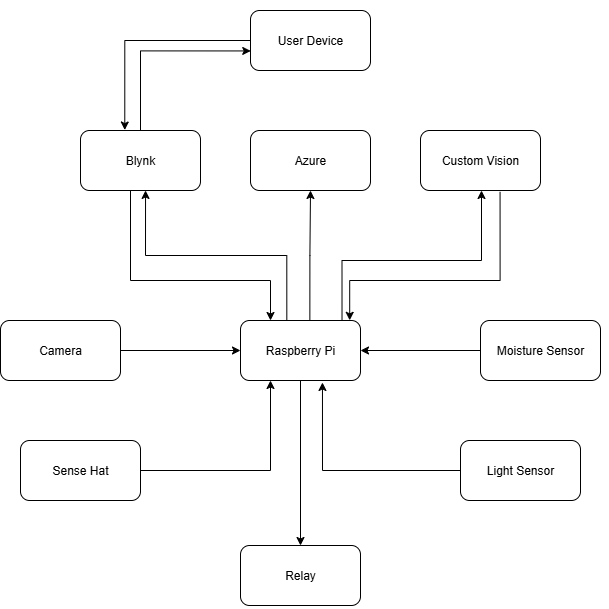
The idea came to me because I got a lemon tree at the start of the year and found there are many little tasks I need to keep track of to take good care of it. Knowing when to water and mist it, making sure it’s getting enough light and rotating it (it’s indoors so this helps spread the light out evenly) etc.

The goal isn’t to replace what a person would need to do to take care of a plant, rather to help make it more manageable through tracking information and sending notifications for tasks. The only thing that would be done automatically is a UV lamp would be turned on (represented by the relay), should there not be enough light for the plant.

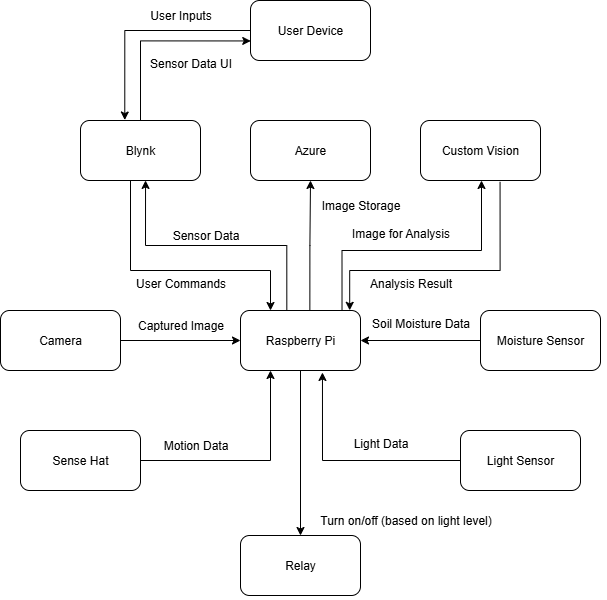
Another issue I ran into is various family members, as well as my dog, bumping into the tree because of how it’s positioned in my room. My room doesn’t have much space so there’s no where I can move it to, so instead I added a camera to snap pictures of whoever bumped into the tree and used AI to tell you who it thinks did it. However, because I cannot demonstrate on real people in class, I instead used plush toys as a proof of concept.

# System architecture diagrams

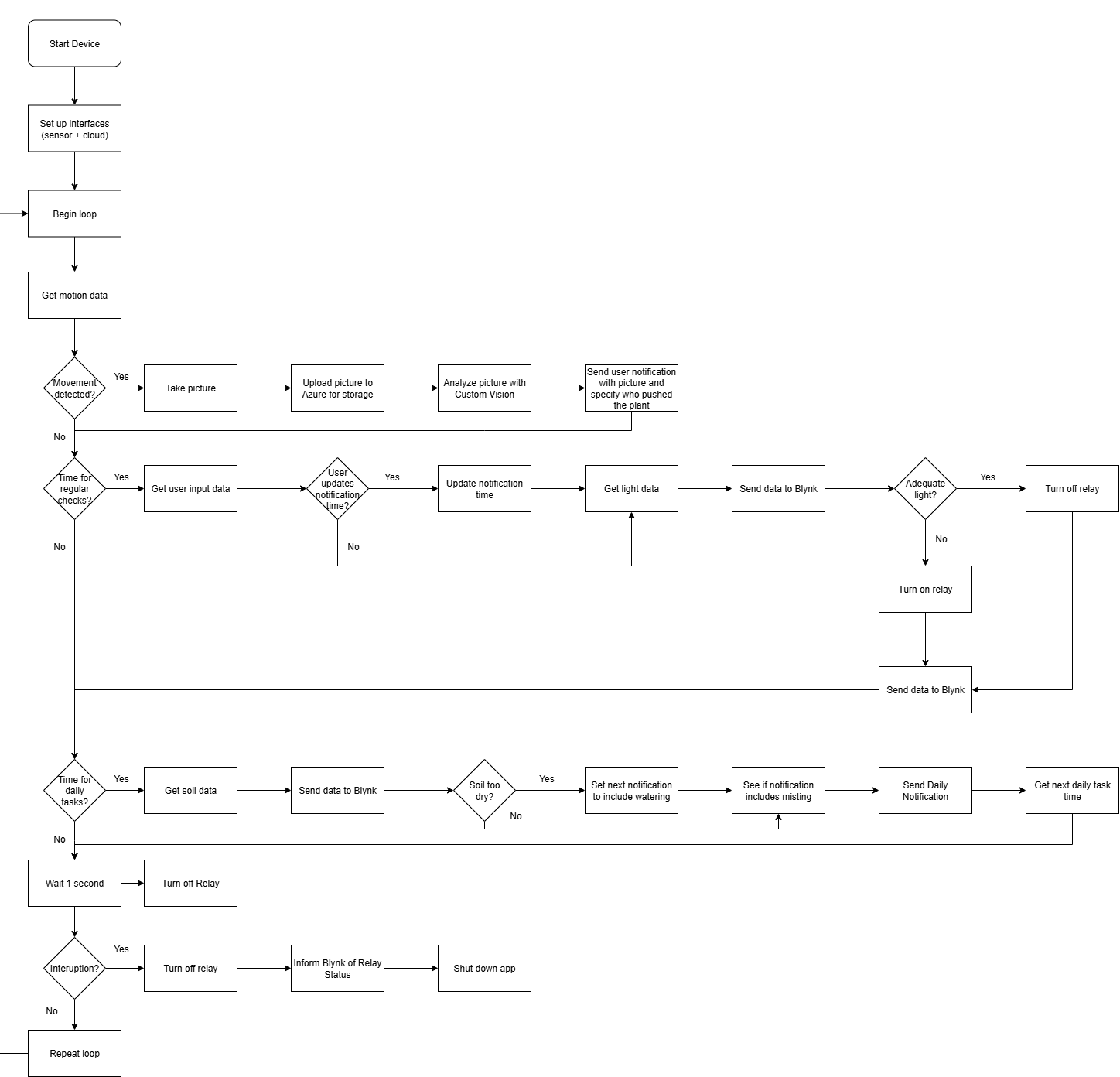
## Pictorial (block level) representation of project



## Informational flow diagram



## Flow chart of algorithm



# Code snippets

My code is divided into 2 sections: hardware\_manager.py and app.py

## Hardware Manager

The purpose of this class is to create an interface for interacting with the different sensors and actuators on my raspberry pi

A screen shot of a computer program

AI-generated content may be incorrect.

Note: I needed to correct the ADC for the light and moisture sensor as the one set automatically was incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

All sensor data would be automatically logged to a file (with a timestamp) and printed in the console

A screen shot of a computer code

AI-generated content may be incorrect.

Motion is checked for by calculating the magnitude of the force the accelerometer of the Sense HAT is experiencing. If it’s not moving, then it should be experiencing 1g (only gravities force), so if it’s experiencing any other amount that’ll mean it’s moving.

In theory it could go perfectly opposed to gravity and that would also give a reading of 1 however it’s unreasonable for it to reach such a specific speed and direction within a tenth of a second (that’s how often motion is checked) and remain perfectly consistent so that edge case and others like it can be ignored.

There is also a threshold within which motion isn’t counted, just to account for minor imprecision in the accelerometer.

Data is of course logged and printed.

A computer screen shot of text

AI-generated content may be incorrect.

Light and moisture sensor data functions. Data is logged and printed.

A screen shot of a computer program

AI-generated content may be incorrect.

Image capture function. Image from the pi camera is saved to a file, and it’s logged that an image was taken.

A computer screen shot of text

AI-generated content may be incorrect.

Code to control relay.

## App

The purpose of app.py is to manage various interfaces, time management and generally running the app.

A computer screen shot of a program

AI-generated content may be incorrect.

Various delays are implemented to help with efficiency. Motion is checked for every tenth of a second and on success there is a 2 second cooldown before checking again. Light data is checked every minute.

A black screen with white text

AI-generated content may be incorrect.

Various thresholds are set to decide when certain actions need to be taken and what’s considered desirable ranges.

A black screen with white text

AI-generated content may be incorrect.

Various interfaces are used to interact with different things. Hardware manager has been already discussed, the rest allow the app to interact with Blynk, MS Azure blob storage and custom vision AI.

These are instanced later in the code to allow for error handling.



The user can customise which hour of the day they receive their daily notification.

A black screen with white text

AI-generated content may be incorrect.

Various timestamps are used for the code to know when to run certain blocks.

A black background with white text

AI-generated content may be incorrect.

Variables tracked for notifications.

A screen shot of a computer code

AI-generated content may be incorrect.

This function attempts to run an input function (it’s used for functions that create the previously mentioned interfaces) 3 times and if all 3 fail it exits the app.

A computer screen shot of text

AI-generated content may be incorrect.

Functions to create the interfaces. As up until these are ran Blynk is equal to “None”, instead of using annotations I needed to run the function within here to make it react to the daily hour change from the Blynk dashboard.

A screen shot of a computer program

AI-generated content may be incorrect.

Light data is gotten from the light sensor (through the hardware manager) and sent to Blynk.

If it is determined to be too dark, the relay (representing a UV light) is turned on, otherwise it’s turned off. This too is reported to Blynk.

A screen shot of a computer program

AI-generated content may be incorrect.

Moisture data is gotten, reported to Blynk and then if the moisture reading is too high (it’s measured in mV, and the higher the value the more dryness) then set a Boolean to make sure we remind the user to water the next time they get a notification

A computer screen with colorful text

AI-generated content may be incorrect.

Get’s the next time to do the daily tasks (which would be measuring soil moisture and sending the user their daily notification) by getting the desired hours timestamp for today, and if we’re already past it then add a day to get that hour tomorrow.

A screenshot of a computer program

AI-generated content may be incorrect.

This code runs if the user interacts with the Blynk dashboard to change the hour that daily tasks are done.

A computer screen with text

AI-generated content may be incorrect.

Determines the contents of and sends the daily notification to the user. Misting is done on every third day; the watering Boolean is covered earlier by checking the soil moisture and every day you want to rotate the plant.

A screen shot of a computer program

AI-generated content may be incorrect.

Uploads a captured image to Azure blob storage. This way it can be viewed through the Blynk dashboard using the gallery widget.

A screen shot of a computer code

AI-generated content may be incorrect.

Sends a picture to the Custom Vision AI I’ve trained to determine which plush toy is in the picture. The results are printed, and the user receives a notification telling them who touched their plant.

A screen shot of a computer code

AI-generated content may be incorrect.

This is the first part of the actual app running. Here all the interfaces are created with error handling and should any fail after 3 attempts the app is shut down.

A computer screen shot of a program code

AI-generated content may be incorrect.

The main loops runs indefinitely unless there’s an error or the user manually interrupts it.

First thing is checking if we’re allowed to look for motion and then seeing if there is any motion.

On success motion checks are banned for 2 seconds, an image is taken, uploaded to MS Azure and Custom Vision AI is used to determine which toy is in the picture and the user is notified through Blynk.

A computer screen shot of a program

AI-generated content may be incorrect.

Every minute light data as well as Blynk events are processed.

Once a day daily task (processing soil data and sending the daily notification) are processed.

The two timestamps are updated.

The app sleeps for 0.1 seconds before repeating the loop again.

A screen shot of a computer program

AI-generated content may be incorrect.

If the app is manually shut down, no errors are shown, and it ends.

No matter if purposefully or through error, if the app shuts down the relay is always turned off and Blynk is updated.

# Other features

As previously mentioned, external features were needed. This section will discuss them.

## MS Azure

MS Azure stores the image taken by the camera. There is only one ever stored and it gets overwritten by new ones.

## Custom Vision

The Custom Vision model was trained on approximately 80 images and 2 iterations. It was trained to look using the general domain and was consistent at determining which plush it’s looking at.

A screenshot of a video game

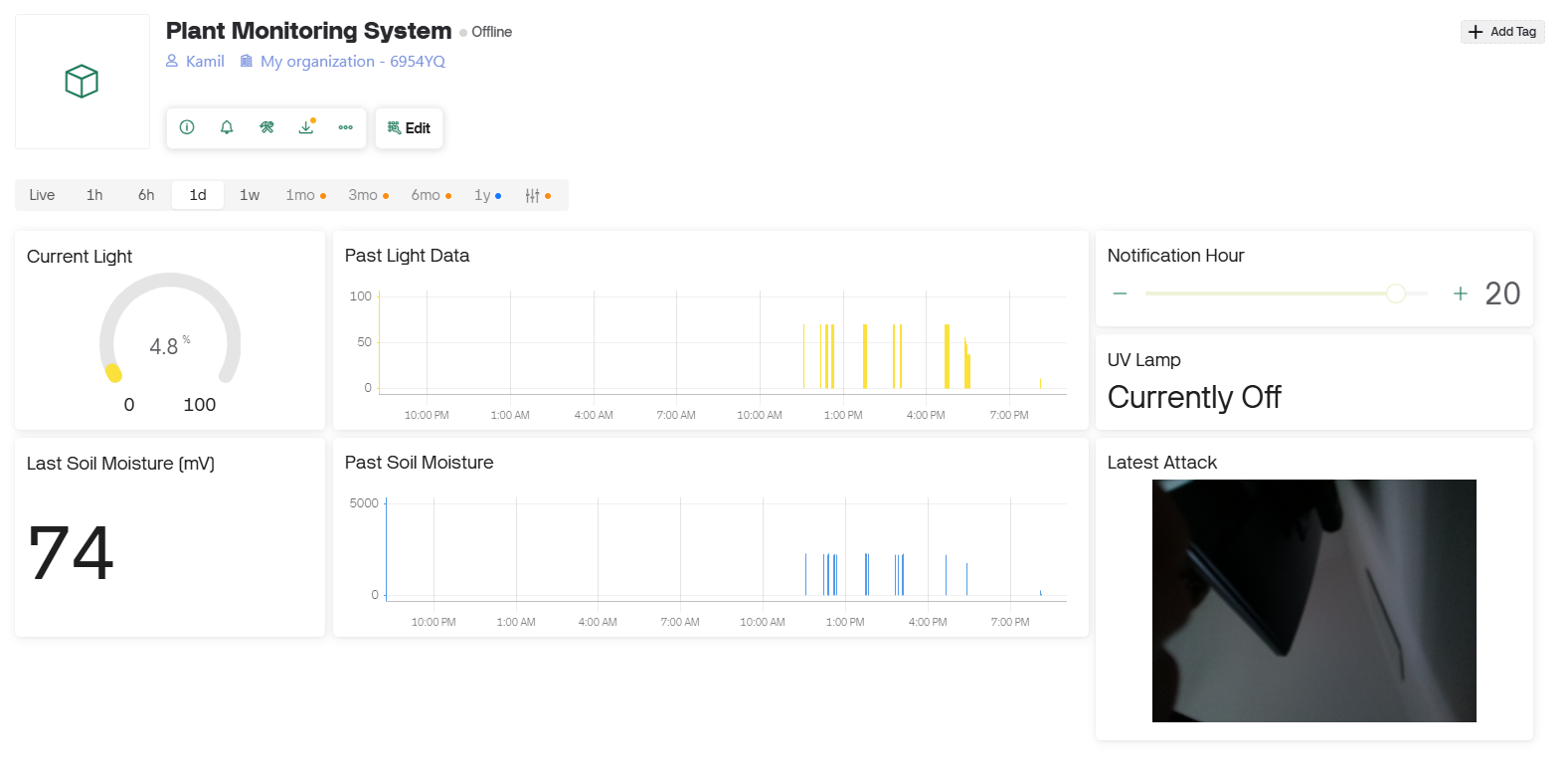
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

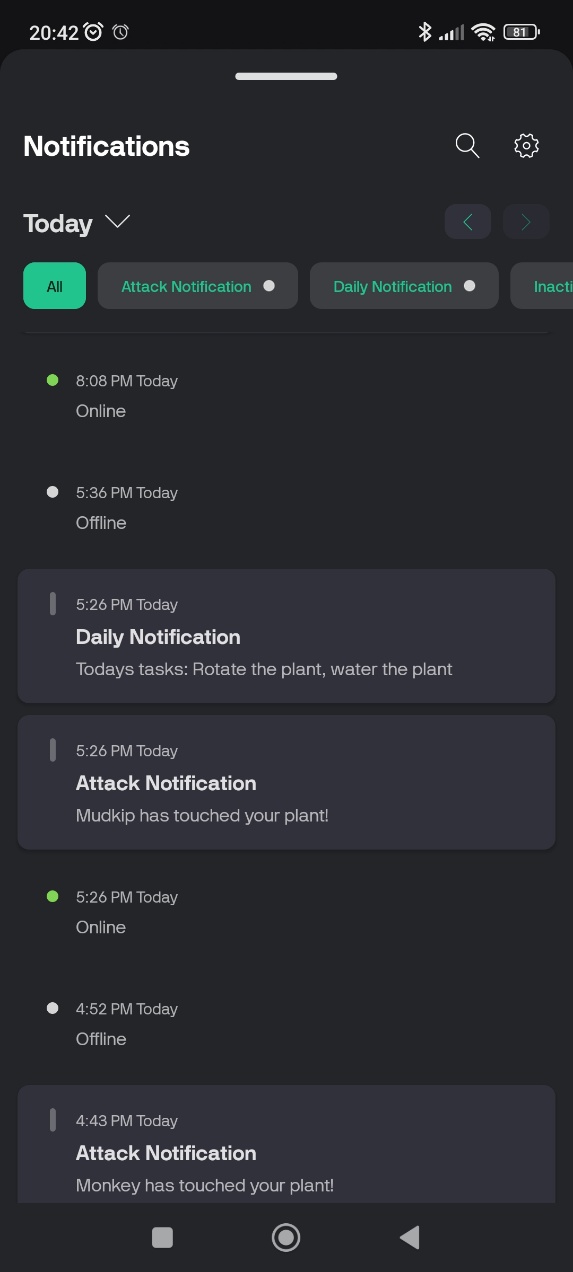
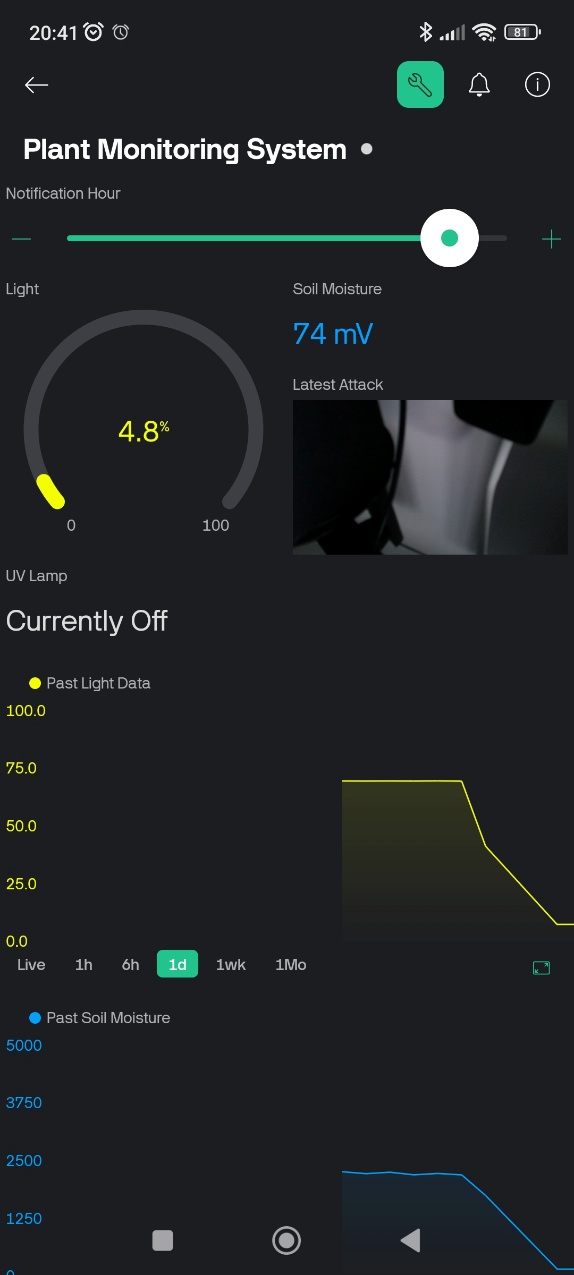
## Blynk

Blynk allowed me to handle notifications as well as displaying data from my IOT device. I made a dashboard for my computer as well as one for the mobile app. Both works identically and can handle notifications.



A screenshot of a phone

AI-generated content may be incorrect.



# Data collected

Data collected is shown in 3 different ways. It is printed to the console while the program runs, saved to a text file (and timestamped) every time new data is gotten and uploaded up to Blynk.

A screenshot of a computer program

AI-generated content may be incorrect.

Light and soil data written in console.

A screenshot of a computer program

AI-generated content may be incorrect.

Data logged to text file.

A screenshot of a computer

AI-generated content may be incorrect.

Data uploaded to Blynk (with visualisation).

# Challenges faced

As with any new project there were many things I didn’t know how to do at first and needed to overcome.

One of the first was making sure all my sensors work so that I could code. I didn’t include this code in here (though it is on GitHub) as it isn’t used anywhere but early on, I made a sensor test script that let’s me quickly check if all of them are working how I’d expect. It essentially let me run though whichever sensor or actuator I wanted to quickly without needing to deal with Blynk or Azure. This was useful throughout development and especially mid updates.

A screen shot of a computer program

AI-generated content may be incorrect.

Another is I wasn’t sure how to use the accelerometer to detect movement. I thought it would just have a value of 0 unless it’s actively moving. After research I learned it’s closer to telling you the forces it’s experiencing. I then had to find a way to take the 3 forces it has and have a singular force. This is something I’d need to do in game development quite often, so once I realised it’s a matter of getting the magnitude of the vector I knew where to go from there. It’s like using Pythagoras theorem in 2 dimensions, only here I’m doing 3 dimensions.

The only issue after that was that it kept going off because the readings were never exactly 1 (due to imprecision I think) and it was very sensitive, so I added a threshold, so it only activated with a notable enough movement. This felt like the right decision in this context as if someone just walks past the plant it should have minor movements, but the device should only detect whenever the plant pot is touched, not walked past.

I wasn’t sure how to manage time management at some point in the early development of the app. I couldn’t just use sleep the whole time because every tenth of a second something did need to happen. By using timestamps though I could still sleep the app when it truly didn’t need to do anything, and it would still skip running most the code most of the time (as other parts needed to be ran once a minute and once a day) which helped improve efficiency and not make the device overheat for example.

The attempt\_interface\_creation() function took me some time to make as well. Originally, I had a very large code block that looked very messy. Each interface was wrapped in essentially the same code, however they each needed to run different code for setting them up. I did find that python (like Kotlin and other languages) does let you use callables as parameters, which let me clean up a lot of the code.