PG5501 Spring, candidate: 1017

The code runs through ***Fish_Feeder.ino***.

ssd and pass, must be filled for the project to work.

Case: Automatic fish feeder with Apple Siri integration!

Please note that during the video presentations the feeding cycle is set to run in an infinite loop. This is purely for demonstration purpose and the code for this is removed, as this operation only runs 1-3 times a day as I will further explain. Watch ***demo_using_siri*** a demonstration.

Purpose.

The main purpose of this device is to feed fish in an aquarium based on daily time intervals, let's say twice a day which is the default interval. This default interval can be overwritten by the user at any time. This is a handy device for when one is away from the resident for a couple of days, no longer need to ask anyone to come over just to feed the fish.

Parts used.

- **1 x Bottle + release mechanism e.g., a pen.**
- **1 x ESP32**
- **1 x Arduino UNO**
- **1 x Servo (The actuator)**
- **1 x TFT**
- **4 x Pushbuttons**
- **1 x 5v Relay**
- **1 x Lamp**
- **2 x LDR (The sensors) One to measure food amount in the bottle, another for the lamp.**
- **6 x 10Ω resistors, 2 for the LDR 4 for the buttons**

The Device:

The device consist of a TFT screen used to as an interface to display current weather/location, feeding process and setup.

When device boots up the device for the first time it connects to the internet and start feeding the fish twice a day from that moment each feeding cycle has a pre-defined duration of 30 seconds. After the feeding cycle the interface displays the weather, this is the "main" interface. Watch ***demo_boot*** for the demonstration. *Note* in the video the first feeding cycle is disabled, this is just to make the video shorter.

The mechanism of the device is simple, I used the servo from the kit as the actuator the servo is powered by 5V from an Arduino UNO as I realised 3.3V from the ESP32 was not

enough. The servo is placed in a bottle I cut open; this is the container for the fish food. I then attached a hollow tube from an old percolator as the release mechanism. I used an LDR sensor to measure if the bottle is empty or not as an example, I know the bottle is empty when a substantial amount of light is detected meaning that there is no more food in the bottle, so then the servo should not run. Watch ***demo_food_sensor*** for a demonstration.

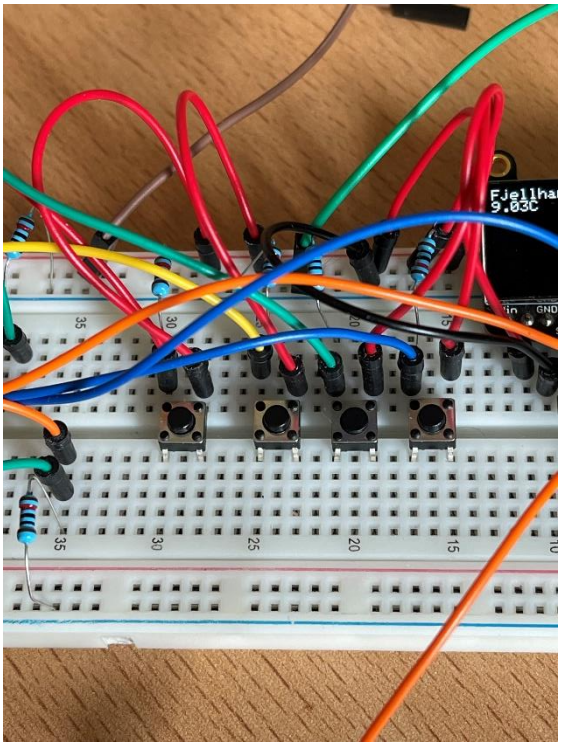
This method is not entirely flawless, as depending on the time of the day, light would still penetrate through the grains of fish food and cause inconsistent readings meaning the bottle is empty, when it's not. This could be "fixed" by taking the time of the day into the equation, let's say its midday and sunny then the sensor should be less sensitive. But given its imperfections I still found it to be the best solution for the problem among the sensor I had available.

Another LDR sensor is used to controll the aqurium lamp with a relay. This sensor is only active when feeding the fish, and is measuring light each second within the feeding cycle if to much light is detected it relay turns of and vice versa, watch ***demo_lamp_sensor*** for a demonstration.

The release arm and bottle container is empty,

LDR sensor to check whether the bottle neck.

The sensor has later been placed lower down



Button description from the left:

1. Allows for “on-demand” feeding.
2. Decreases the interval, without subceeding 1 which is the minimum daily interval.
3. Hold down for 3 seconds to enter “setup-mode”
4. Increases the interval, without exceeding 3 which is the maximum daily interval.

Note button 2 and 3 can only be used when in “setup-mode”.

The first prototype of the project used a rotary encoder to enter the setup and increase/decrease the value, but it was very inconsistent, tried using interrupts to no avail, tried again with a potentiometer without success, that’s why I ended up using buttons.

Watch ***demo_setup*** for a demonstration.

Internet.

I spend some time scouting the web for some interesting AP`is, I came across the NASA AP'is made a quick prototype of an Asteroid detection system but I kind of met a roadblock as it was not practical and functionality was limited, also tried the FBI API but same story, not practical.

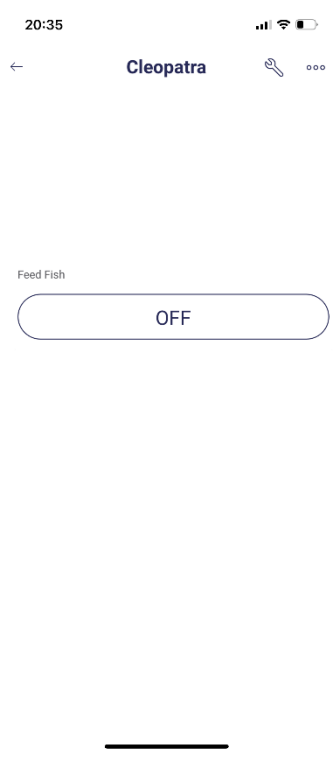
I finally just decided to stick to the openweathermap API to get weather information "the hard way". This device uses two AP'is, one to get geolocation data and another one to get the weather based on the geolocation.

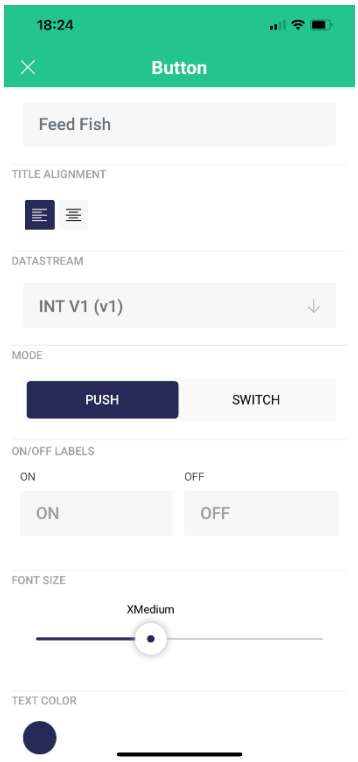
In addition to a standard API-key the openweathermap API requires *latitude* and *longitude* parameters within the URL. I looked in to setting up the geolocation API from Google so I could get geolocation data through that service, but this was a bit of a hassle, so after some browsing, I came across ip-api which is an API that returns geolocation data based on a given IP address. The problem was getting a public IP address from the EPSP32, the ***localIP()*** method from the Wi-Fi library could not be used since it's a private address, but after iterating through the other Wi-Fi methods from the library, I stumbled upon the ***dnsIP()*** method witch returned an DNS IP address. I then tested this address by sending a GET request using the Insomnia API client to fetch data from the geolocation API when this worked, I went back to the code and implemented the functionality to combine both AP'is.

The App/Siri-integration

Originally, I wanted to use the **Arduino Cloud IoT** to create the app since the free plan has more features than Blynk's free plan, but it seemed as I was bound to use the online code editor which is a dealbreaker for me.

The device uses Blynk as an alternative to feed the fish (Button one from previous picture), only one button is implemented for now, see picture below. I selected the PUSH option which resets the button back off after pressing, which is the response I needed.





The main interface

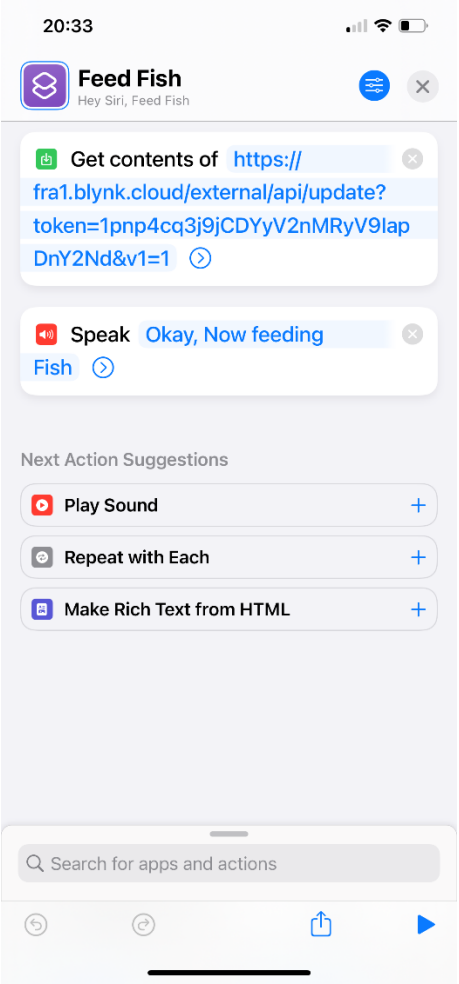
The button settings

I wanted to be able to control my device using my iPhone preferably through Apples HomeKit environment, I found Arduino libraries such as HomeSpan, and HomeKit-ESP32 which allows direct implementation of Apples HomeKit Protocol, but the documentation was a bit tedious to read for the amount of time I had available. Will definitely use HomeSpan for my next project!

So, for the Siri integration I came across IFTT which allows you to run trigger webhooks using so-called *applets* e.g. “feed fish” would be considered an applet, the downside is that the free IFTT plan only allows for 5 applets, which would have more than enough for the current stage of this project. But I figured out that its possible to access Blynk devices directly through URL, after testing this notion using once again, Insomnia to send GET request I came up with the following URL pattern:

GET https://fra1.blynk.cloud/external/api/update?token={YOUR_TOKEN}&{PIN_NUMBER}={VALUE}

I then used Apples Shortcuts application to create a shortcut that sends this request to the Blynk server. With this I was now able to use Siri to manually feed the fish.



The request to virtual pin 1 to feed fish.

The Code:

The code consists of 6 classes. In the order of the execution flow in ***Fish-Feeder.ino*** , could also have used a class diagram but...

1. **WiFiManager.cpp** Used establish Wi-Fi connections and to get the JSON data.

- **- HTTP Client**
    - Holds the state of the client requests

- **-payload**
    - Stores the payload/contents of a URL.

- + **setup()**
    - Setup Wi-Fi connection using *ssid* and *password* passed as parameter
- + **connected()**
    - Returns a boolean whether the Wi-Fi is connected
- + **getIP()**
    - Returns a string the IP address needed to for the openweather API
- + **getJSON()**
    - Returns a DynamicJsonDocument, used this over StaticJsonDocument as it was advised in the documentations. Takes an URL reference as parameter.
    - Used ArduinoJson Assistant to calculate the buffer, as advised in the docs.
- + **setJson()**
    - A helper for the getJSON(), takes the document and payload as reference parameter, deserializes the Json and checks for potential errors, like the buffer is insufficient.

2. **Location.cpp** Simple Struct that stores Geolocation.
    - **- m_countrycode, m_city, m_lat, m_lon.**
    - + **setup()**
        - Sets values to the fields.

3. **MyServo.cpp** contains the servo logic
    - + **MyServo()**
        - Sets the **pinNumber**
    - - **m_pinNumber**
        - Stores the pin number passed as param from constructor
    - - **m_servo**
        - Holds a Servo object found in the *ESP32Servo* library
    - + **operate()**
        - Runs the servo quicky twice from 0-45 degrees, this is to create a "shaking" effect in case any grains of food get stuck. I detach the servo, otherwise it will keep holding at 45 degrees, which is unnecessary.

4. **Button.cpp** stores the button states e.g., on/off
    - - **m_pinNumber**
    - - **currentSate**
        - Stores the current state of the button, HIGH/LOW
    - - **prevState**

    - Stores the previous state of the button, HIGH/LOW
- + **Button()**
    - Sets the fields currentState will be the digitalRead of the button creation, while previoState is initialized to 0.
- + **isPressed()**
    - Returns a boolean to the caller, a button is pressed when the state has changed, and current state is HIGH.

- + **stateChanged()**
    - Returns if the currentState is different from the previous, if it is then a change has occurred.
- + **readState()**
    - Updates the currentState
- + **updateState()**
    - Updates the previous state, with the current one.

5. **Light.cpp** switches on and off lights, the only thing special is that HIGH means turn off, and LOW means turn on.

6. **Sensor.cpp**
    - **- m_pinNumber**
    - **+ Sensor()**
        - Sets the fields
    - + **reading()**
        - Compresses the analog values from 0-4095 down to 0-255 and returns the value to the caller. 4095 is the analog range for the ESP32, while 1023 is the default for an Arduino UNO.

Fish_Feeder.uno: Where it all boils down

I am fully aware there is a decent amount of stuff going on this file, it might seem that I should have made some additional classes to handle this flow, but in a weird way I found it more organized to keep all the functionality in this one file, and share the work across multiple helper functions.

I will only comment on the content in the loop function, as this is most important, functions like ***setLocation, drawToScreen*** are self-explanatory and won't need any further explanation, same goes for variable names.

The first instruction in the loop is to run Blynk which was setup in the setup function, I then start counting the **currentTime** using **millis()**, this function is used to count the duration of time the board has been on (goes up to approximately 50 days, before resetting back to 0), this is useful to keep loops "tight", meaning further instructions can be executed

simultaneously as in multitasking, this is a better alternative than using **delay()** which is a code "blocking" function.

- **showWeather()**
  - Uses the **currentTime** to check weather it is time to **getWeather**() from the API, this is checked every 30 seconds, this function will run once after booting, feeding the fish or changing the intervals. Otherwise, the user would have to wait 30 seconds before receiving the weather data.

- **monitoFeedingInterval()**
  - checks whether its time to feed the fish, here I use **currentTime** again to check whether its time to feed the fish, this function is run once upon booting the device and changing the intervals.

- **monitTorButtons()**
  - monitors (reads/updates) the four buttons previously shown

  feedFishButton()

  - when this button gets pressed it manually feeds the fish, while keeping the daily interval. The **feedFish()** is another loop that runs the servo once, but keeps checking the sensors (**foodSensor,lampSensor**) once a second for 30 seconds, here again **millis()** is used instead of **delay()**.

  setFoodIntervalButton()

  - Hold down for three seconds to enter the **setFeedInterval()** function. **setFeedInterval()** allows user to change the daily feeding interval, using **decreaseFeedIntervalButton/increaseFeedIntervalButton.**
    Once the time is set by again pressing the **setFoodIntervalButton(),**
    The function **getFeedIntervalMills()** is invoked, this function takes the hour as a parameter and does calculation that divides a day (24h) to the hour, multiplies the result by 360000, witch is an hour in seconds and returns the value, this value will be the new **dailyFeed**.

Libraries used.

ArduinoJson.h

Adafruit_GFX.h

Adafruit_ST7789.h

SPI.h

Adafruit_I2CDevice.h

BlynkSimpleEsp32.h

ESP32Servo.h

WiFi.h

HTTPClient.h

ArduinoJson.h

Sources.

Used this one as a main guide to the ESP32 pinouts, found it easier to follow than a plain datasheet.

https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Resources provided by the lecturer on Canvas.