Environment Monitoring IoT Device

CS3242

Micro-controllers and Applications

D. M. G. C. Dassanayake
170097P

Contents

1. Scope of the project

1.1 Task

To design and develop an environment monitoring IoT device with a data logger server application.
The device is capable of monitoring,

- Temperature
- Humidity
- Barometric pressure
- Ambient light level

It also is capable of operating in unreliable wifi connectivity by buffering its data and sending them once the connection is re-established.

1.2 Description

- The IoT device is composed of a NodeMCU32s and sensor modules which are battery powered.
- The server is developed with flask(a python framework) running on a PC based environment.
- The persistence layer is an sqlite database.

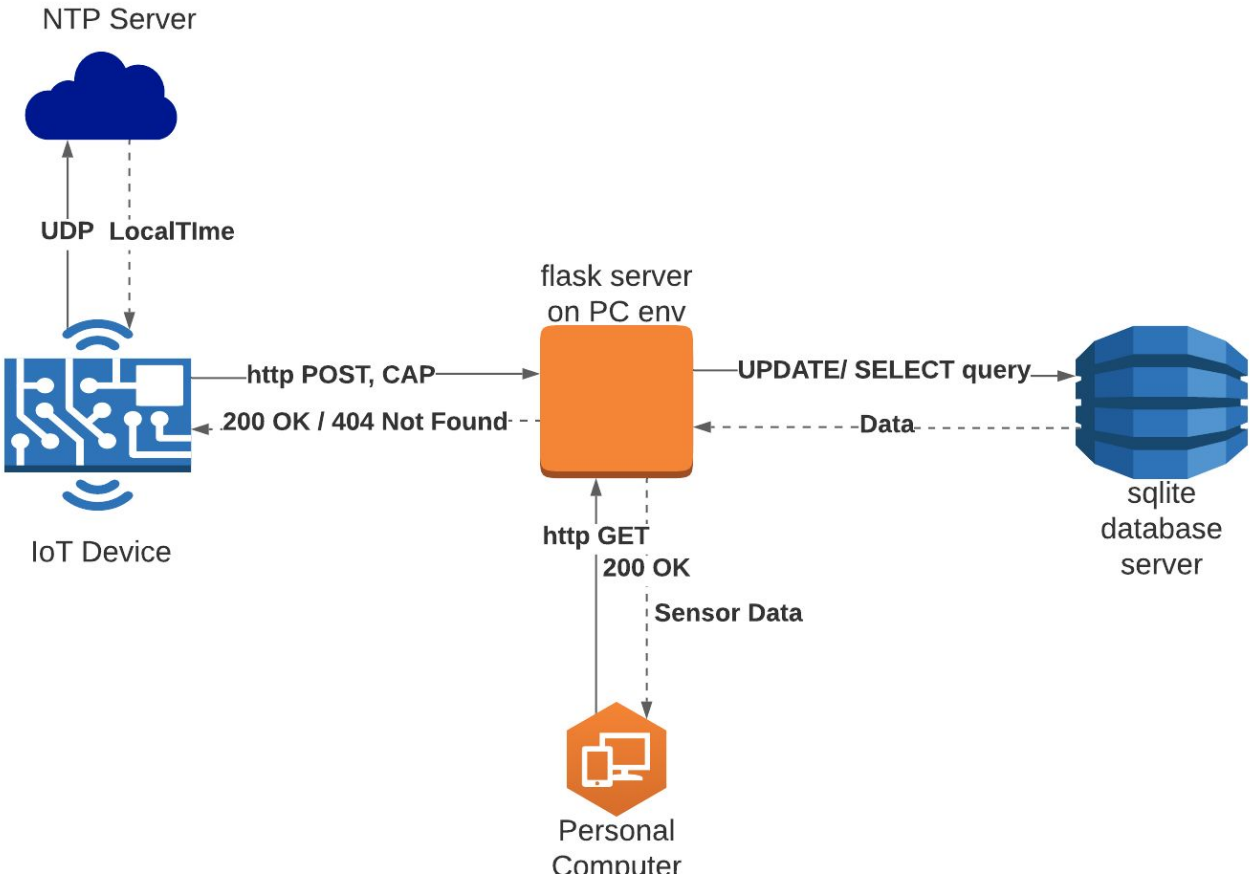
- The NodeMCU collects sensor data for a time period of 15 minutes and updates the server with the mean and the standard deviation of the sensor data along with a time stamp. The MCU connects to the internet through a WiFi connection, and sends an HTTP POST request to the server. The sensor data is encapsulated in Common Alert Protocol(CAP) in the request body.

- The server receives the request from the MCU, parses and stores data in the sqlite database server. Then the server responds with 200, OK to the NodeMCU.

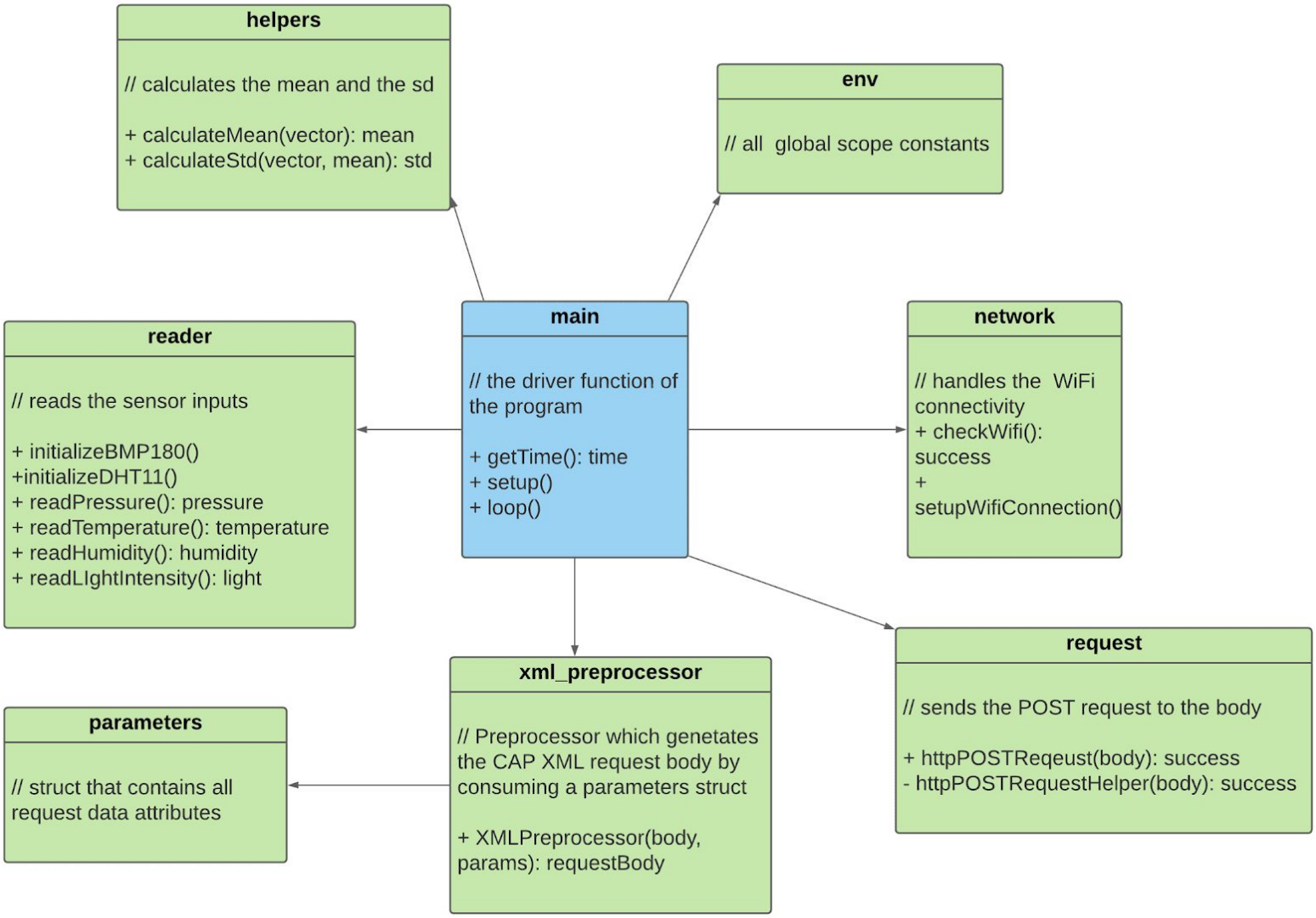
- The server also contains a GET endpoint that can be used to list the stored sensory information in the database.

2. High-level design

2.1 Overall Design

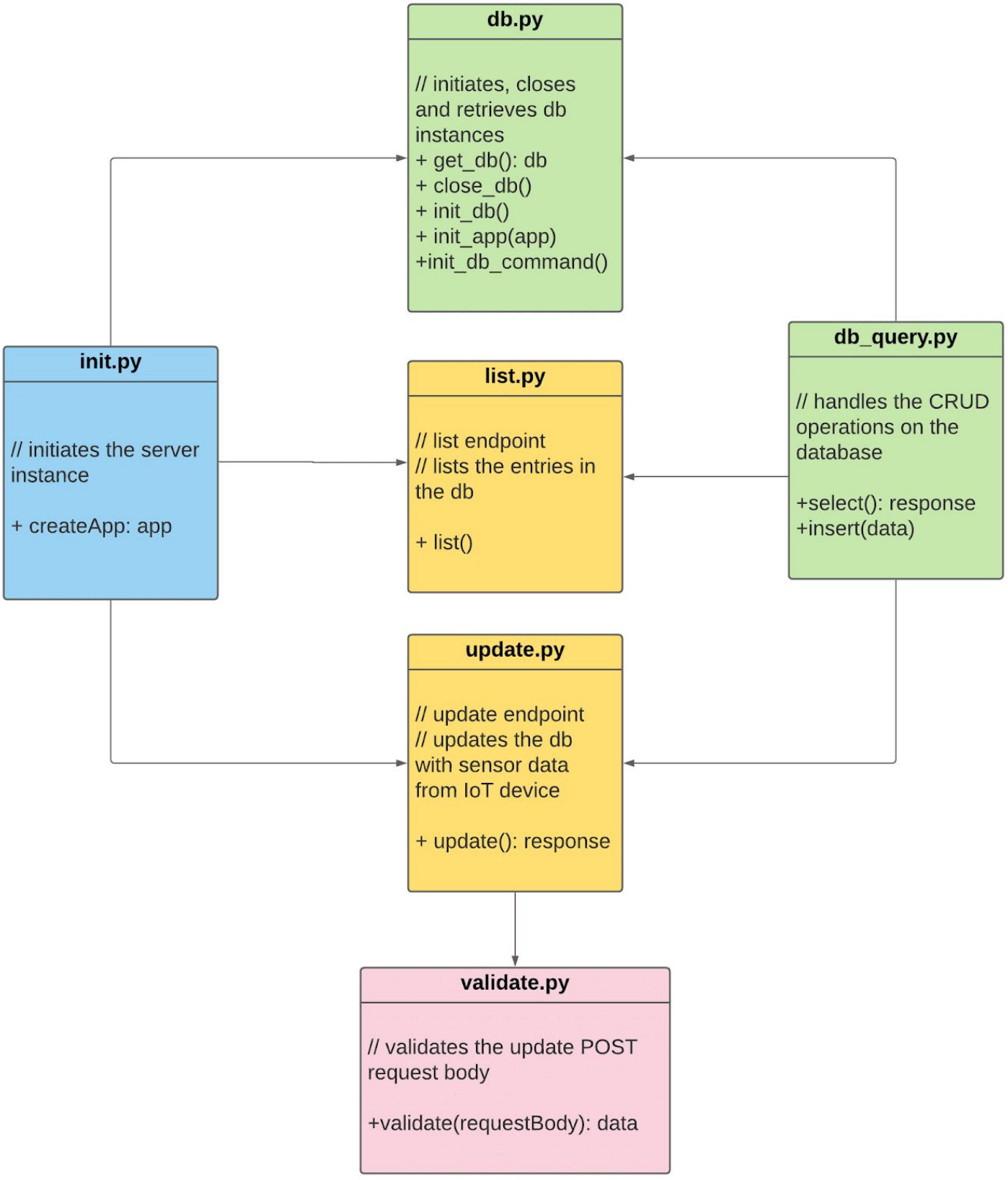


2.2 IoT Device



Some minor dependencies were ignored for simplicity

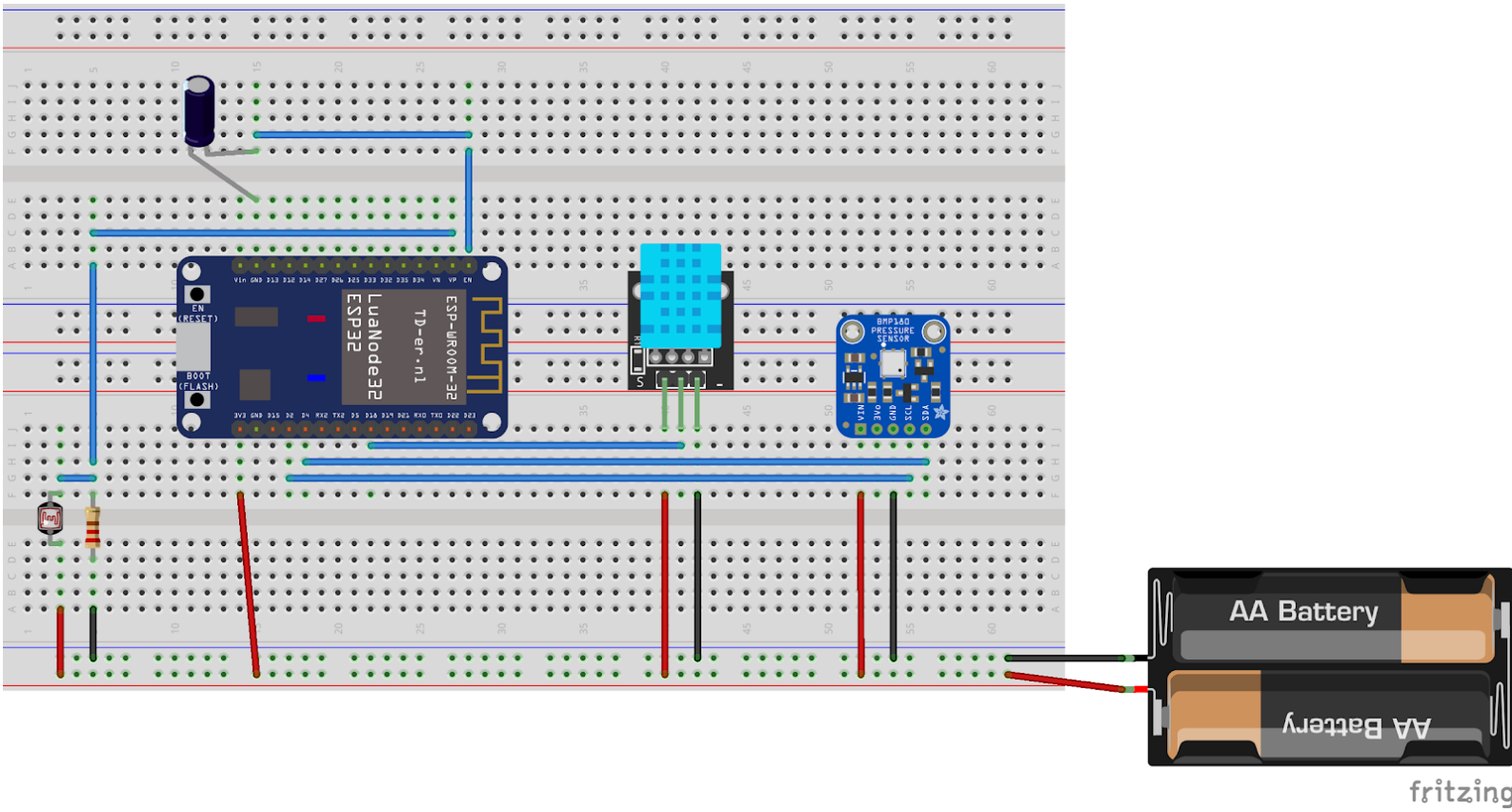
2.3 Server


db.py
// initiates, closes and retrieves db instances
+ get_db(): db
+ close_db()
+ init_db()
+ init_app(app)
+init_db_command()
init.py
// initiates the server instance
+ createApp: app
list.py
// list endpoint
// lists the entries in the db
+ list()
db_query.py
// handles the CRUD operations on the database
+select(): response
+insert(data)
update.py
// update endpoint
// updates the db with sensor data from IoT device
+ update(): response
validate.py
// validates the update POST request body
+validate(requestBody): data

3. List of components and their cost

	Component	Price (Rs.)
1	BMP180 4-pin 1.8V to 6V Digital Barometric Pressure Temperature and AltitudeSensor for Arduino	185.00
2	LDR 4mm	10.00
3	5k Variable Resistor Preset	10.00
4	2N3904 NPN General Purpose Transistor	4.00
5	NodeMCU ESP-32S WiFi Bluetooth Dual	1,050.00
6	DHT11 Temperature and Relative Humidity Sensor Module	250.00
7	Breadboard	285.00
Total		1, 794.00

4. Schematic Diagram



5. Description on how fault recovery options are implemented

5.1 WiFi Connection Drop

- The WiFi connection is established at the initial setup of the device.
- But the connection can be lost at any time.
- Hence a queue is introduced to store the mean and standard deviation of sensor details.
- Then the queue is processed one by one and if the WiFi connection is down it tries to connect to the wifi several times.
- If it fails to do so, it will keep the data in the queue and move on to calculating sensor values for the next iteration.
- If it succeeds to connect to WiFi the request is sent and the queue is emptied.

5.2 HTTP request failure

- If the HTTP request is failing the device will try several times to get a 200, OK response by repeating the request.
- Even then if it fails to send the request successfully, the queue containing the sensor data will not be emptied.
- It will try again in the next iterations until the request is successfully sent.
- Once the request is successfully sent the queue is emptied.

6. Algorithms

6.1 Device Pseudocode

```
BEGIN

    DEFINE PARAMETER_QUEUE, SENSOR_VALUE_ARRAYS, TIMER, SAMPLER,
                REQUEST_BODY

    START SETUP
        SETUP_WIFI_CONNECTION()
        CONFIGURE_NTP_TIME()
        INITIALIZE_SENSORS()
        INITIALIZE timer, sampler
    END SETUP

    LOOP
        IF CURRENT_TIME - TIMER > 15 MINUTES
            TIMER = CURRENT_TIME
            CALCULATE MEAN, STANDARD_DEVIATION OF SENSOR READS
            CLEAR SENSOR_VALUE_ARRAYS
            UPDATE_TIME = GET_TIME_FROM_NTP_SERVER()
            PARAMETER = NEW PARAMETER(MEAN, STANDARD_DEVIATION,
                        UPDATE_TIME)
            PARAMETER_QUEUE.ADD(PARAMETER)
        END IF

        IF(CURRENT_TIME - SAMPLER > 30 SECONDS)
            SAMPLER = CURRENT_TIME
            READ SENSOR_VALUES
            SENSOR_VALUE_ARRAYS.PUSH(SENSOR_VALUES)
        END IF

        WHILE(PARAMETER_QUEUE IS NOT EMPTY)
            IF WIFI NOT CONNECTED
                RESPONSE = CONNECT_WIFI
            END IF
```

```
                    IF RESPONSE != SUCCESS
                        BREAK
                    END IF
                    XML_REQUEST = XMLPREPROCESSOR(REQUEST_BODY,
                                    PARAMETER_QUEUE.FRONT)
                    RESPONSE = SEND_HTTP_REQUEST(XML_REQUEST)
                    IF RESPONSE == SUCCESS
                        PARAMETER_QUEUE.POP()
                    ELSE
                        BREAK
                    END IF
                END WHILE
            END LOOP
END
```

6.2 Server Pseudocode

```
BEGIN
    SETUP

    CREATE APP INSTANCE
    INITIALIZE APP INSTANCE
    IMPORT DB CONNECTION
    INITIALIZE UPDATE, LIST ENDPOINTS

    END SETUP

    WHILE TRUE
        LISTEN FOR REQUESTS

        IF REQUEST
            IF REQUEST == UPDATE
                VALIDATED_BODY = VALIDATE(REQUEST.BODY)
                INSERT_TO_THE_DATABASE(VALIDATED_BODY)
                IF ERROR
                    RETURN 404
                RETURN 200
                END IF
            ELSE IF REQUEST == LIST
                SENSOR_DATA  = SELECT_DATA_FROM_DATABASE()
                IF ERROR
                    RETURN 404
                RETURN 200
                END IF
            END IF
        END IF
    END WHILE
END
```

7. References

7.1 Github repository

https://github.com/gayaldassanayake/microcontrollers

7.2 IoT Device

1. Hardware specification
   ESP8266

2. C++ references
   2.12 — Header guards

   How should I declare global variables in my C++ project?

3. ESP32S WiFi Connectivity
   ESP32: Connecting to a WiFi network

4. HTTP POST request from ESP32S
   ESP32 HTTP GET and HTTP POST with Arduino IDE
5. Libraries for sensors
   pressure
   BMP180 by Felix Rusu · Libraries · PlatformIO
   Humidity and temperature
   DHT sensor library by Adafruit · Libraries · PlatformIO
   LDR
   NodeMCU With LDR : 4 Steps (with Pictures)

   ESP32 Analog Input with Arduino IDE

6. NTP Server
   Getting Date & Time From NTP Server With ESP32

7.3 Server

7. Flask official documentation
   https://flask.palletsprojects.com/en/1.1.x/quickstart/#
8. CAP specification
   https://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html

9. CAP Parsing library

   https://pypi.org/project/capparselib/

8. Full source code (an Annexure)

Server

```
|-- ./src
        |--  ./database
                |-- db_quety.py
                |-- db.py
        |--  ./routes
                |-- list.py
                |-- update.py
        __init__.py
        validate.py
        schema.sql
```

__init__.py

```
import os

from flask import Flask


def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
    )

    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)
    else:
        # load the test config if passed in
        app.config.from_mapping(test_config)

    # ensure the instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    from flaskr.database import db
    db.init_app(app)

    from flaskr.routes import update, list
    app.register_blueprint(update.bp)
    app.register_blueprint(list.bp)

    return app
```

validate.py

```
from capparselib.parsers import CAPParser


def validateRequest(body):
    try:
        alert_list = CAPParser(body).as_dict()
        alert = alert_list[0]

        parameters = alert['cap_info'][0]['cap_parameter']

        for parameter in parameters:
            if(parameter['valueName'] == 'temperature_mean'):
                temperature_mean = float(parameter['value'])
            elif(parameter['valueName'] == 'temperature_std'):
                temperature_std = float(parameter['value'])
            elif(parameter['valueName'] == 'humidity_mean'):
                humidity_mean = float(parameter['value'])
            elif(parameter['valueName'] == 'humidity_std'):
                humidity_std = float(parameter['value'])
            elif(parameter['valueName'] == 'pressure_mean'):
                pressure_mean = float(parameter['value'])
            elif(parameter['valueName'] == 'pressure_std'):
                pressure_std = float(parameter['value'])
            elif(parameter['valueName'] == 'light_mean'):
                light_mean = float(parameter['value'])
            elif(parameter['valueName'] == 'light_std'):
                light_std = float(parameter['value'])
            elif(parameter['valueName'] == 'update_time'):
                update_time = str(parameter['value'])

        if(not temperature_mean or not humidity_mean or not pressure_mean or
not light_mean
           or not temperature_std or not humidity_std or not pressure_std or
not light_std or not update_time):
            raise ValueError()

        return (temperature_mean, humidity_mean, pressure_mean, light_mean,
temperature_std, humidity_std, pressure_std, light_std, update_time)

    except:
        return ValueError()
```

database/db.py

```
import sqlite3

import click
from flask import current_app, g
from flask.cli import with_appcontext


def get_db():
    if 'db' not in g:
        g.db = sqlite3.connect(
            current_app.config['DATABASE'],
            detect_types=sqlite3.PARSE_DECLTYPES
        )
        g.db.row_factory = sqlite3.Row

    return g.db


def close_db(e=None):
    db = g.pop('db', None)

    if db is not None:
        db.close()

def init_db():
    db = get_db()

    with current_app.open_resource('schema.sql') as f:
        db.executescript(f.read().decode('utf8'))


@click.command('init-db')
@with_appcontext
def init_db_command():
    """Clear the existing data and create new tables."""
    init_db()
    click.echo('Initialized the database.')

def init_app(app):
    app.teardown_appcontext(close_db)
    app.cli.add_command(init_db_command)
```

database/db_query.py

```
from flaskr.database.db import get_db
from flask import jsonify


def insert(temperature_mean, humidity_mean, pressure_mean, light_mean,
           temperature_std, humidity_std, pressure_std, light_std,
           update_time):
    try:
        db = get_db()
        db.execute(
            'INSERT INTO status(temperature_mean, humidity_mean,
pressure,light_mean,'
             'temperature_std,humidity_std,pressure_std,light_std,
update_time) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)',
            (temperature_mean, humidity_mean, pressure_mean, light_mean,
             temperature_std, humidity_std, pressure_std, light_std,
             update_time)
        )
        db.commit()
    except Exception as e:
        print(e)
        raise e


def select():
    try:
        db = get_db()
        data = db.execute(
            'SELECT update_time,
temperature_mean,humidity_mean,pressure_mean,light_mean,'
            'temperature_std,humidity_std,pressure_std,light_std FROM status
order by update_time').fetchall()
        response = []

        for row in data:
            entry = {}

            entry['temperature_mean'] = row[1]
            entry['humidity_mean'] = row[2]
            entry['pressure_mean'] = row[3]
            entry['light_mean'] = row[4]
            entry['temperature_mean'] = row[1]
            entry['humidity_mean'] = row[2]
```

```
            entry['pressure_mean'] = row[3]
            entry['light_mean'] = row[4]
            entry['update_time'] = row[0]

            response.append(entry)

        jsonResponse = jsonify(response)
        return jsonResponse

    except Exception as e:
        print(e)
        raise e
```

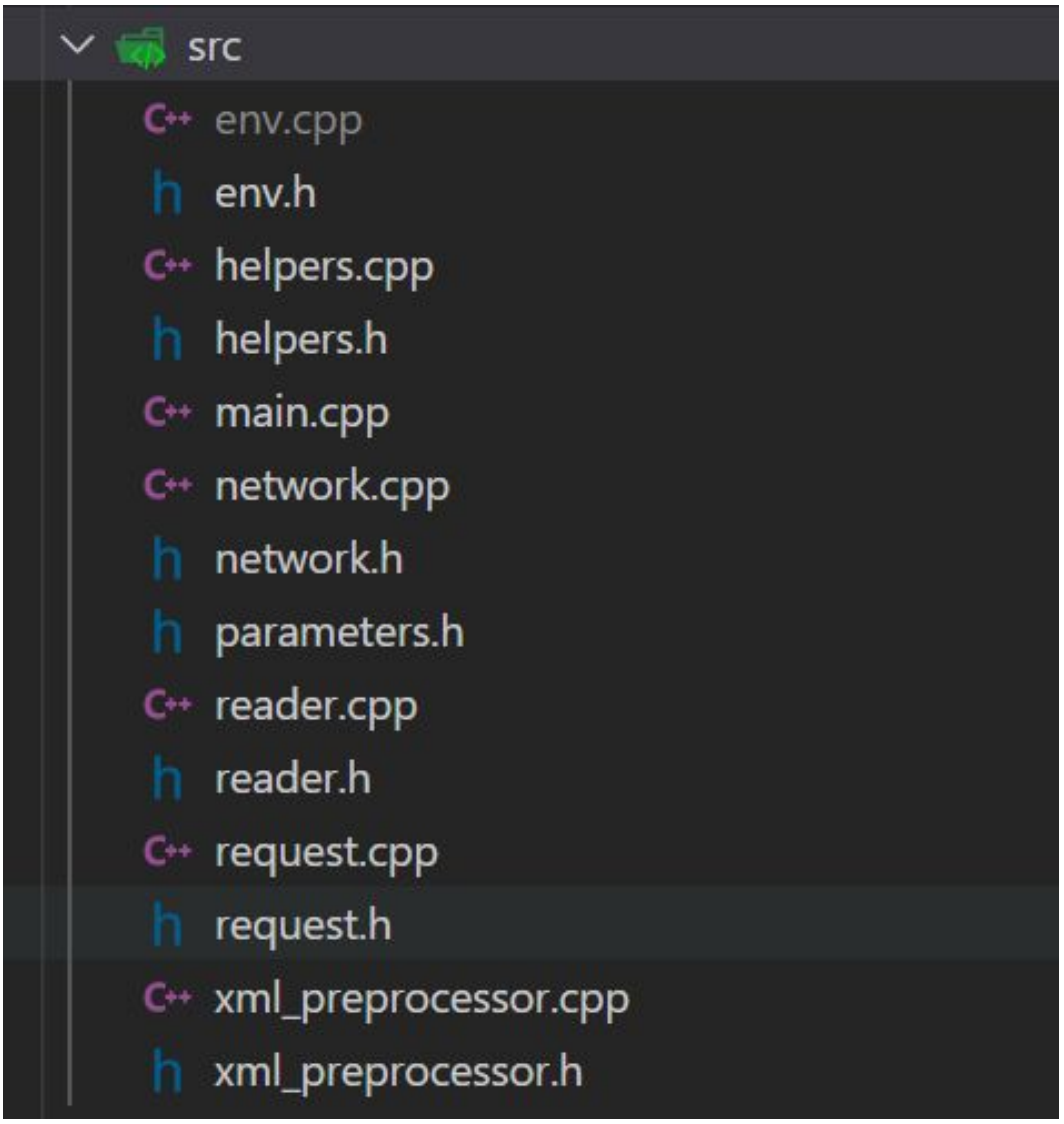
routes/list.py

```
import functools

from flask import (Blueprint, request)

from flaskr.database.db_query import select

bp = Blueprint('list', __name__, )


@bp.route('/list', methods=['GET'])
def list():
    if request.method == 'GET':
        try:
            response = select()
            return response, 200

        except:
            return 'Error while listing', 404

    return '', 404
```

routes/update

```
import functools

from flask import (Blueprint, request)

from flaskr.validate import validateRequest

from flaskr.database.db_query import insert

bp = Blueprint('update', __name__, )


@bp.route('/update', methods=['POST'])
def update():
    if request.method == 'POST':
        try:
            body = request.get_data().decode('utf-8')

            temperature_mean, humidity_mean, pressure_mean, light_mean,
            temperature_std, humidity_std, pressure_std, light_std,
            update_time = validateRequest(body)

            insert(temperature_mean, humidity_mean, pressure_mean,
light_mean,
                   temperature_std, humidity_std, pressure_std, light_std,
                   update_time)

            return '', 200

        except:
            return 'Invalid message body format', 404

    return '', 404
```

IoT Device


src
env.cpp
env.h
helpers.cpp
helpers.h
main.cpp
network.cpp
network.h
parameters.h
reader.cpp
reader.h
request.cpp
request.h
xml_preprocessor.cpp
xml_preprocessor.h

main.cpp

```
#include <Arduino.h>
#include <queue>
#include <WiFi.h>
#include <vector>
#include "network.h"
#include "request.h"
#include "time.h"
#include "reader.h"
#include "env.h"
#include "parameters.h"
#include "helpers.h"
#include "xml_preprocessor.h"


using std::queue;
using std::vector;

const char *ntpServer = "pool.ntp.org";
const long gmtOffset_sec = (int)(5.5 * 60 * 60);
const int daylightOffset_sec = 0;

queue<Parameters> parameterList;

vector<float> temperatureList;
vector<float> humidityList;
vector<float> pressureList;
vector<float> lightList;

int timer, sampler;
int identifier = 0;
char requestBody[1000];

void getTime(char *update_time)
{
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo))
  {
    Serial.println("Failed to obtain time");
    return;
  }
  char buffer[20];
  strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", &timeinfo);
  sprintf(update_time, buffer);
```

```
}

void setup()
{
  Serial.begin(115200);
  delay(100);

  setupWifiConnection();

  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

  initializeBMP180();
  initializeDHT11();

  timer = millis();
  sampler = millis();

  Serial.println("Setup done");
}

void loop()
{

  if (millis() - timer > TIMER_DELAY)
  {
    timer = millis();

    Parameters param;
    param.humidity_mean = calculateMean(humidityList);
    param.temperature_mean = calculateMean(temperatureList);
    param.pressure_mean = calculateMean(pressureList);
    param.light_mean = calculateMean(lightList);

    param.humidity_std = calculateStd(humidityList, param.humidity_mean);
    param.temperature_std = calculateStd(temperatureList,
param.temperature_mean);
    param.pressure_std = calculateStd(pressureList, param.pressure_mean);
    param.light_std = calculateStd(lightList, param.light_mean);

    humidityList.clear();
    temperatureList.clear();
    pressureList.clear();
    lightList.clear();

    param.id = identifier;
```

```
    identifier++;

    getTime(param.update_time);

    parameterList.push(param);
  }

  if (millis() - sampler > SAMPLING_RATE)
  {
    sampler = millis();
    temperatureList.push_back( readTemperature());
    humidityList.push_back( readHumidity());
    pressureList.push_back( readPressure());
    lightList.push_back( readLightIntensity());

  }

  while(parameterList.size()>0 && checkWifi()){
    XMLPreprocessor(requestBody, parameterList.front());

    if(httpPOSTRequest(requestBody)){
      parameterList.pop();
    }
    else{
      break;
    }
  }
}
```

helpers.h

```
#ifndef HELPERS_H
#define HELPERS_H
#include <vector>
using std::vector;

float calculateMean(vector<float> data);

float calculateStd(vector<float> data, float mean);

#endif
```

helpers.cpp

```
#include <Arduino.h>
#include <vector>

using std::vector;

float calculateMean(vector<float> data){
    int length = data.size();
    float total = 0;
    for(int i =0; i< length; i++){
        total+=data.at(i);
    }

    float mean = total/ length;
    return mean;
}

float calculateStd(vector<float> data, float mean){
    int length = data.size();
    float sumOfSquares = 0;
    for(int i =0; i< length; i++){
        sumOfSquares+= pow(data.at(i), 2);
    }

    float std =  pow(sumOfSquares/length - pow(mean, 2)*length, 0.5);
    return std;
}
```

network.h

```
#ifndef NETWORK_H
#define NETWORK_H
void setupWifiConnection();
bool checkWifi();
#endif
```

network.cpp

```
#include "WiFi.h"
#include "env.h"

void setupWifiConnection()
{
    Serial.begin(SERIAL_RATE);
    WiFi.begin(SSID, PASSWORD);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print("Connecting to WiFi: ");
        Serial.println(SSID);
    }
    Serial.println("Connected to the WIFi network!");
}

bool checkWifi()
{
    if (WiFi.status() == WL_CONNECTED)
    {
        return true;
    }
    for (int i = 0; i < 5; i++)
    {
        WiFi.begin(SSID, PASSWORD);
        delay(100);
        if (WiFi.status() == WL_CONNECTED)
        {
            return true;
        }
    }
    return false;
}
```

parameters.h

```
#ifndef PARAMETERS_H
#define PARAMETERS_H

struct Parameters
{
    int id;
    float temperature_mean;
    float humidity_mean;
    float pressure_mean;
    float light_mean;

    float temperature_std;
    float humidity_std;
    float pressure_std;
    float light_std;

    char update_time[20] = {};
};

#endif
```

reader.h

```
#ifndef READER_H
#define READER_H

void initializeBMP180();
void initializeDHT11();
float readPressure();
float readTemperature();
float readHumidity();
float readLightIntensity();

#endif
```

reader.cpp

```
/**
 * Sample code Written by Limor Fried, Ladyada and,
 * Example testing sketch for various DHT humidity/temperature sensors
 * Written by ladyada for Adafruit Industries
 * was used when constructing the following code.
 * Adafruit invests time and resources providing this open source code,
 * please support Adafruit and open-source hardware by purchasing
 * products from Adafruit!
 */

#include <Wire.h>
#include <Adafruit_BMP085.h>
#include "DHT.h"

#include "env.h"

#define DHTTYPE DHT18

Adafruit_BMP085 bmp;

#define DHTPIN 2      // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

void initializeBMP180()
{
    if (!bmp.begin())
    {
        Serial.println("Could not find a valid BMP085 sensor, check
wiring!");
        while (1)
        {
        }
    }
    Serial.println("BMP085 Initialized!");
}

void initializeDHT11()
{
    dht.begin();
    Serial.println("DHT11 initialized!");
```

```
}

float readPressure()
{
    float pressure = (float)bmp.readPressure();
    if (isnan(pressure))
    {
        Serial.println("Failed to read temperature from DHT sensor!");
        return -1;
    }
    return pressure;
}

float readTemperature()
{
    float temperature = dht.readTemperature();  // temperature in celcius
    if (isnan(temperature))
    {
        Serial.println("Failed to read temperature from DHT sensor!");
        return -1;
    }
    return temperature;
}

float readHumidity()
{
    float humidity = dht.readHumidity();    // humidity as a percentage
    if (isnan(humidity))
    {
        Serial.println("Failed to read temperature from DHT sensor!");
        return -1;
    }
    return humidity;
}

float readLightIntensity()
{
    int LDRValue = analogRead(LDR_PIN);
    float lightIntensity = LDRValue * (100.0 / 4095.0);   // Intensity as a
value in [0.0,1,0]
    return lightIntensity;
}
```

request.h

```
#ifndef REQUEST_H
#define REQUEST_H

bool httpPOSTRequest(char* body);

#endif
```

request.cpp

```
#include <Arduino.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include "env.h"

bool httpPOSTRequestHelper(char *body)
{
    if (WiFi.status() == WL_CONNECTED)
    {
        HTTPClient http;

        String endpoint = (String)SERVERNAME + "/update";
        // String endpoint = (String)SERVERNAME + "/post";

        http.begin(endpoint);

        http.addHeader("Content-Type", "application/xml");
        // http.addHeader("Content-Type", "text/plain");

        int httpResponseCode = http.POST((String)body);

        http.end();

        if (httpResponseCode == HTTP_CODE_OK)
        {
            Serial.println("Request sent.");
            return true;
        }

        Serial.print("HTTP Error:");
        Serial.println("httpResponseCode");
        return false;
    }
```

```
    Serial.println("WiFi Disconnected");
    return false;
}

bool httpPOSTRequest(char *body)
{
    for (int i = 0; i < 5; i++)
    {
        if (httpPOSTRequestHelper(body))
        {
            return true;
        }
    }

    return false;
}
```

xml_preprocessor.h

```
#include "parameters.h"

#ifndef XML_PREPROCESSOR
#define XML_PREPROCESSOR

void XMLPreprocessor(char *body, Parameters params);

#endif
```

xml_preprocessor.cpp

```
#include <Arduino.h>

#include "parameters.h"

void XMLPreprocessor(char *body, Parameters params)
{
    String id = (String)params.id;

    String temperature_mean = (String)params.temperature_mean;
    String temperature_std = (String)params.temperature_std;

    String humidity_mean = (String)params.humidity_mean;
    String humidity_std = (String)params.humidity_std;

    String pressure_mean = (String)params.pressure_mean;
    String pressure_std = (String)params.pressure_std;

    String light_mean = (String)params.light_mean;
    String light_std = (String)params.light_std;

    String update_time = (String)params.update_time;

    sprintf(
        body,
        "<?xml version = '1.0' encoding = 'UTF-8' standalone = 'yes'?>"
        "< alert xmlns = 'urn:oasis:names:tc:emergency:cap:1.1' >"
            "<identifier>%s</identifier>"
            "<sender>iotdevice</sender>"
            "<sent>%s</sent>"
            "<status>Actual</status>"
            "<msgType>Alert</msgType>"
            "<scope>Public</scope>"
```

```
            "<info>"
                "<category>Env</category>"
                "<event>Update</event>"
                "<urgency>Expected</urgency>"
                "<severity>Minor</severity>"
                "<certainty>Likely</certainty>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value>"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s< / value >"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value>"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value >"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value>"
                "</parameter>"
                 "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value>"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s< / value >"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value>"
                "</parameter>"
                "<parameter>"
                    "<valueName>%s</valueName>"
                    "<value>%s</value >"
                "</parameter>"
            "</info>"
        "</alert>",
        id.c_str(),
```

```
        update_time.c_str(),

        "update_time", update_time.c_str(),

        "temperature_mean",temperature_mean.c_str(),
        "temperature_std", temperature_std.c_str(),

        "pressure_mean",pressure_mean.c_str(),
        "pressure_std", pressure_std.c_str(),

        "humidity_mean",humidity_mean.c_str(),
        "humidity_std", humidity_std.c_str(),

        "light_mean",light_mean.c_str(),
        "light_std", light_std.c_str()

    );
}
```

env.h

```
#ifndef ENV_H
#define ENV_H

extern const char *SSID;
extern const char *PASSWORD;
extern const int SERIAL_RATE;
extern const char *SERVERNAME;
extern const int TIMER_DELAY;
extern const int LDR_PIN;
extern const int SAMPLING_RATE;

#endif
```

Env.cpp - not included due to sensitivity of data