Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Safe View Camera Monitoring System

Project Engineering

Year 4

Kenneth Mc Hugh

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2024/2025

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

___Kenneth Mc Hugh____

NOTE: The Project Title and Author Name appear in the header of each page. You must edit these “Document Properties” using the File > Info menu!

Acknowledgements

I liked to thank my supervisors Paul Lennon and Pat Hurney for their time and effort. I'd also wish to thank Michele Lynch, Ben Kinsella, Niall O Keefe, Brian O Shea Dave Newell for their help and advice during the project year.

Table of Contents

1 Summary

The goal of this final-year project was to design and develop a real-time monitoring system aimed at enhancing the safety and peace of mind for individuals with specific health conditions such as dementia, epilepsy, heart issues, or those at risk of falls while alone. The system uses a camera module along with various sensors to monitor a person's vital signs. In the event of an incident, the camera is automatically triggered, and an alert is sent to a designated family member or carer. The carer can then access a web page to assess the situation and intervene swiftly if needed. Additionally, the system stores this data for future medical review.

The system was initially designed for a residential bedroom, with potential scalability to other areas of a home, residential care facilities, or medical environments. Key features include real-time video observation, sensor-based monitoring, and software to handle data processing and alert thresholds.

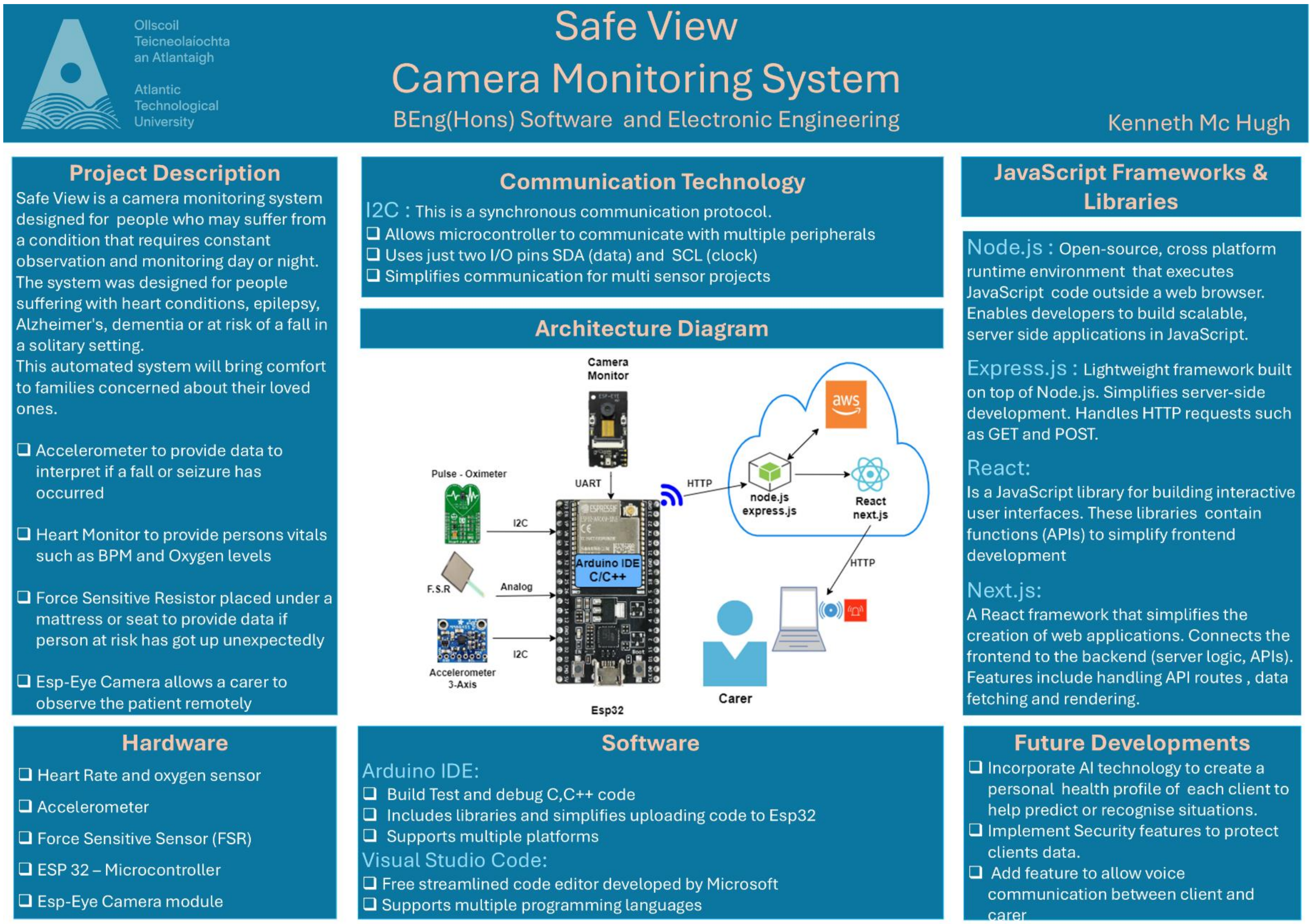
The project was structured in two phases:

- Semester One: focused on hardware development, aiming to have most components connected and basic functionality established by the end of term.
- Semester Two: shifted to software development, including testing the sensors to calibrate accurate threshold settings for alerts.

The software stack included Node.js for the backend and React for the frontend interface. The ESP32 microcontroller ran firmware developed in Arduino IDE (C++), with I2C protocol used for sensor data communication.

Although the full project was not completed, I successfully established communication between the sensors, camera, and the server. This experience taught me valuable lessons in project management particularly the importance of not spending excessive time on a single issue. In future projects, I've learned that when working with inexpensive components, it's much quicker to replace them if causing an issue and shift focus to another part of the project, rather than spending too much time trying to fix one issue

2 Poster



Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Safe View

Camera Monitoring System

BEng(Hons) Software and Electronic Engineering

Kenneth Mc Hugh

Project Description

Safe View is a camera monitoring system designed for people who may suffer from a condition that requires constant observation and monitoring day or night. The system was designed for people suffering with heart conditions, epilepsy, Alzheimer's, dementia or at risk of a fall in a solitary setting.

This automated system will bring comfort to families concerned about their loved ones.

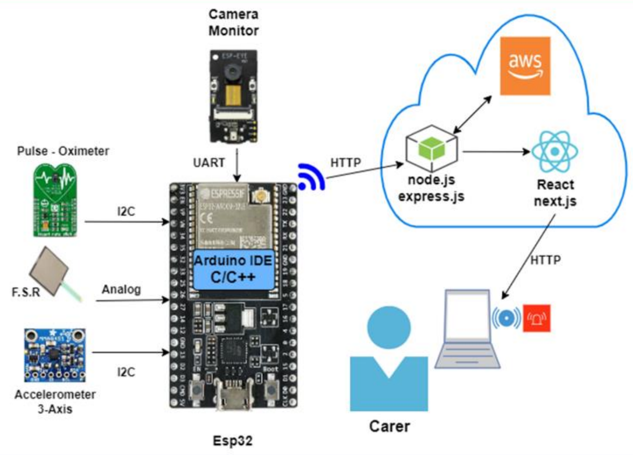
- ❑ Accelerometer to provide data to interpret if a fall or seizure has occurred
- ❑ Heart Monitor to provide persons vitals such as BPM and Oxygen levels
- ❑ Force Sensitive Resistor placed under a mattress or seat to provide data if person at risk has got up unexpectedly
- ❑ Esp-Eye Camera allows a carer to observe the patient remotely

Communication Technology

I2C : This is a synchronous communication protocol.

- ❑ Allows microcontroller to communicate with multiple peripherals
- ❑ Uses just two I/O pins SDA (data) and SCL (clock)
- ❑ Simplifies communication for multi sensor projects

Architecture Diagram



JavaScript Frameworks & Libraries

Node.js : Open-source, cross platform runtime environment that executes JavaScript code outside a web browser. Enables developers to build scalable, server side applications in JavaScript.

Express.js : Lightweight framework built on top of Node.js. Simplifies server-side development. Handles HTTP requests such as GET and POST.

React:
Is a JavaScript library for building interactive user interfaces. These libraries contain functions (APIs) to simplify frontend development

Next.js:
A React framework that simplifies the creation of web applications. Connects the frontend to the backend (server logic, APIs). Features include handling API routes , data fetching and rendering.

Hardware

- ❑ Heart Rate and oxygen sensor
- ❑ Accelerometer
- ❑ Force Sensitive Sensor (FSR)
- ❑ ESP 32 – Microcontroller
- ❑ Esp-Eye Camera module

Software

Arduino IDE:
- ❑ Build Test and debug C,C++ code
- ❑ Includes libraries and simplifies uploading code to Esp32
- ❑ Supports multiple platforms

Visual Studio Code:
- ❑ Free streamlined code editor developed by Microsoft
- ❑ Supports multiple programming languages

Future Developments

- ❑ Incorporate AI technology to create a personal health profile of each client to help predict or recognise situations.
- ❑ Implement Security features to protect clients data.
- ❑ Add feature to allow voice communication between client and carer

3 Introduction

The project aim was to design and develop a real-time monitoring system with an alert function for individuals who are vulnerable due to certain medical conditions such as dementia, epilepsy, heart issues, or those at risk of falling. The primary goal was to enhance safety and provide peace of mind to both the individuals and their families or carers.

My motivation was driven by a couple of aspects. I wanted to help alleviate some of the worry that family members feel for their loved ones who are vulnerable due to health conditions. In facilities such as residential care homes as an extra safety feature which could support staff. My own experience with epilepsy also influenced the development of certain aspects of the system. Additionally, the system was to log footage and sensor data to a database which could be used for medical reviews. I've learnt from personal experience that "Picture speaks a thousand words" and if a doctor has actual data and footage of incidents this can be invaluable for diagnosing or identifying early signs of underlying issues.

The system is designed for a single room and was a proof-of-concept project with the potential for future expansion throughout a residence or care facility. The report will outline the development of the system over two semesters. This report outlines the development process over two semesters, detailing the hardware components used, the software developed, programming platforms (IDEs), and a brief overview of the project management tools. A final section will explore possible directions for future development.

4 Background

You should change the title of this section to suit your own project subject. The aim of this section is to introduce to the reader any relevant background information that is required for your project.
You may have multiple 'background' sections. Think of any of the questions you had to answer during the research phases of your project – these likely should be addressed in a section like this.

5 Project Architecture

Your diagram should be self-documenting. Use subsequent sections in your report to elaborate on technologies / software / hardware in your diagram.

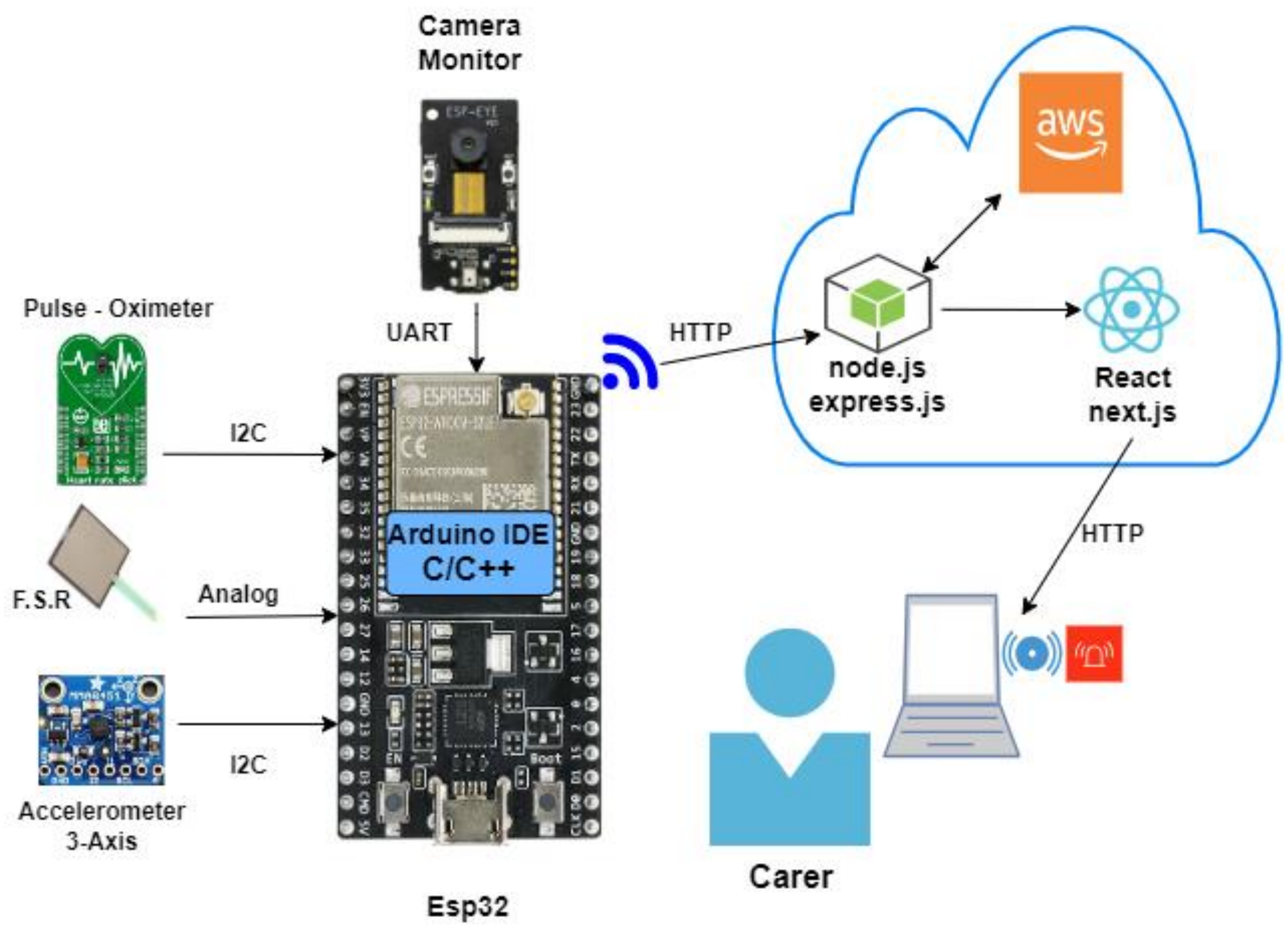
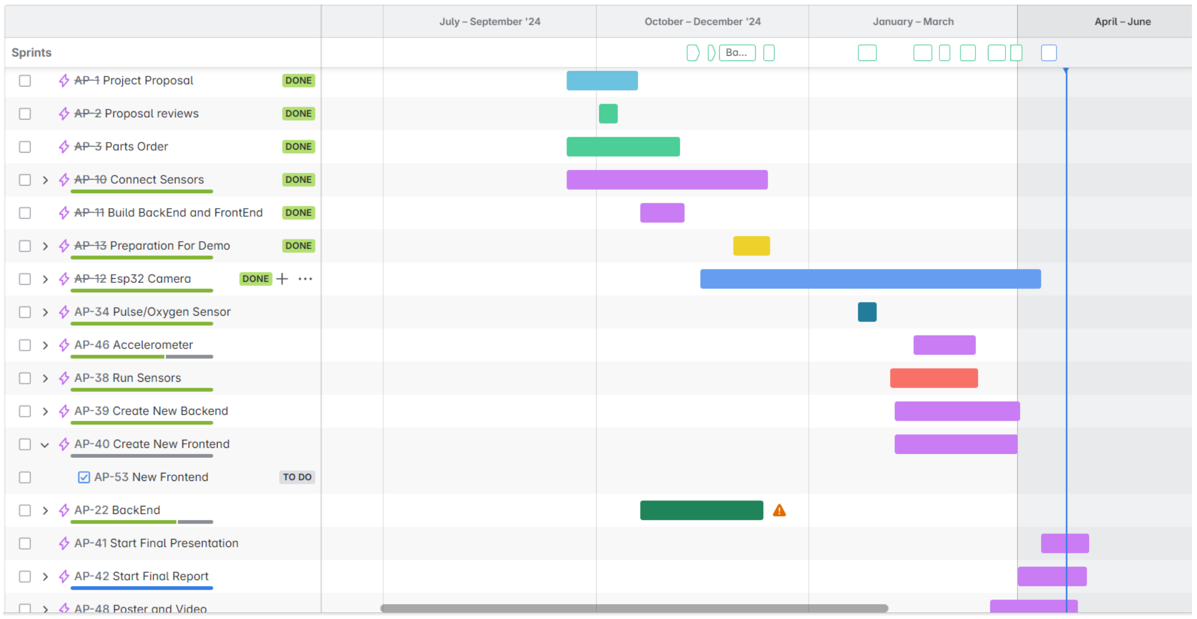


Figure 5-1 Architecture Diagram

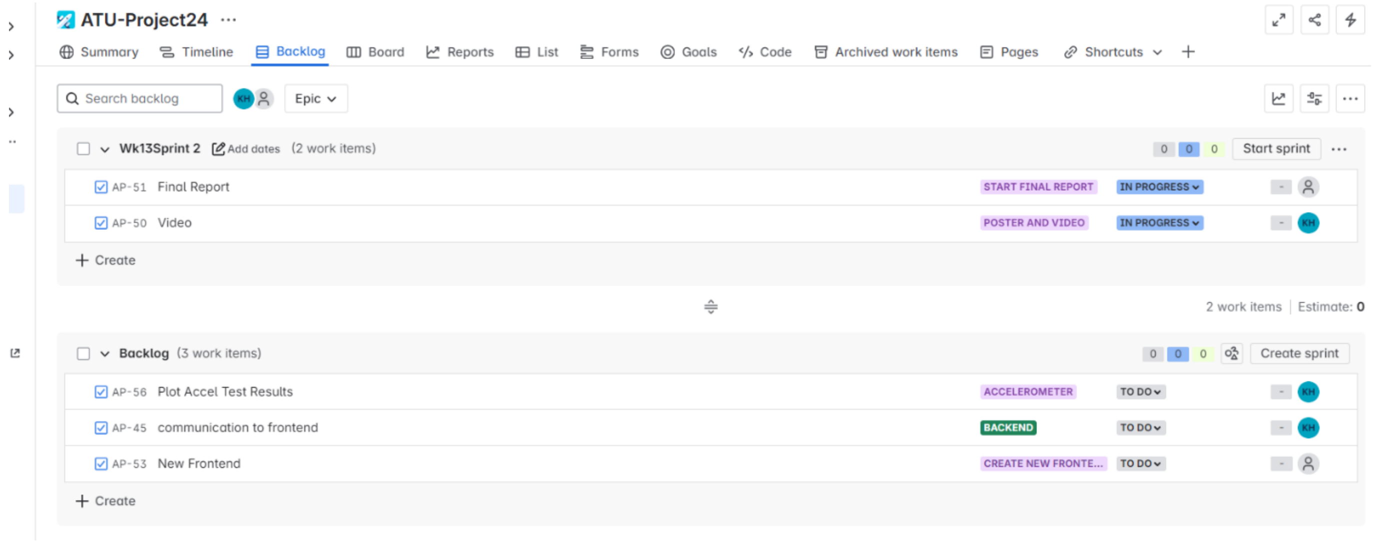
6 Project Plan

This is Jira the project management tool used. The first image is my timeline layout with the parent objectives which then could be broken into child objectives. Example would be “connect sensors” as the parent objective and two child objectives would be the accelerometer and the pulse sensor.



~~AP-1~~ Project Proposal	DONE	
~~AP-2~~ Proposal reviews	DONE	
~~AP-3~~ Parts Order	DONE	
~~AP-10~~ Connect Sensors	DONE	
~~AP-11~~ Build BackEnd and FrontEnd	DONE	
~~AP-13~~ Preparation For Demo	DONE	
~~AP-12~~ Esp32 Camera	DONE	
AP-34 Pulse/Oxygen Sensor		
AP-46 Accelerometer		
AP-38 Run Sensors		
AP-39 Create New Backend		
AP-40 Create New Frontend		
AP-53 New Frontend	TO DO	
AP-22 BackEnd		
AP-41 Start Final Presentation		
AP-42 Start Final Report		
AP-48 Poster and Video		

The image below is the backlog page with tasks to be completed. Here tasks can be placed in sprints to complete within a chosen time frame.



Wk13Sprint 2 Add dates (2 work items) 0 0 0

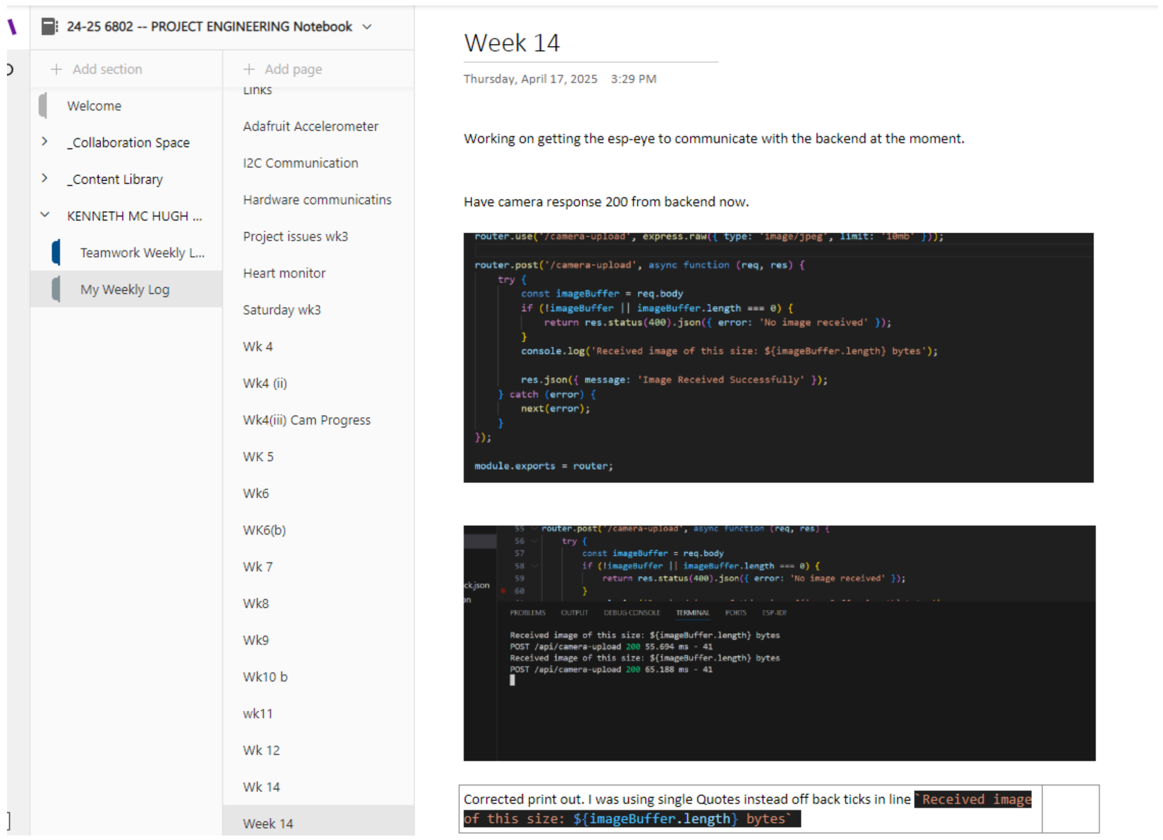
AP-51	Final Report	START FINAL REPORT	IN PROGRESS
AP-50	Video	POSTER AND VIDEO	IN PROGRESS

2 work items | Estimate: 0

Backlog (3 work items) 0 0 0

AP-56	Plot Accel Test Results	ACCELEROMETER	TO DO
AP-45	communication to frontend	BACKEND	TO DO
AP-53	New Frontend	CREATE NEW FRONTE...	TO DO

Also used notebook to log weekly project progress as well as attended weekly standup meetings.



Week 14

Thursday, April 17, 2025 3:29 PM

Working on getting the esp-eye to communicate with the backend at the moment.

Have camera response 200 from backend now.

```
router.use('/camera-upload', express.raw({ type: 'image/jpeg', limit: '10mb' }));

router.post('/camera-upload', async function (req, res) {
    try {
        const imageBuffer = req.body
        if (!imageBuffer || imageBuffer.length === 0) {
            return res.status(400).json({ error: 'No image received' });
        }
        console.log('Received image of this size: ${imageBuffer.length} bytes');

        res.json({ message: 'Image Received Successfully' });
    } catch (error) {
        next(error);
    }
});

module.exports = router;
```

```
router.post('/camera-upload', async function (req, res) {
    try {
        const imageBuffer = req.body
        if (!imageBuffer || imageBuffer.length === 0) {
            return res.status(400).json({ error: 'No image received' });
        }
```

```
Received image of this size: ${imageBuffer.length} bytes
POST /api/camera-upload 200 55.694 ms - 41
Received image of this size: ${imageBuffer.length} bytes
POST /api/camera-upload 200 65.188 ms - 41
```

Corrected print out. I was using single Quotes instead off back ticks in line `` `Received image of this size: ${imageBuffer.length} bytes` ``

7 Hardware and Software Technologies

7.1 Hardware

7.1.1 Esp 32

The ESP32 is part of a family of low-cost microcontrollers manufactured by Espressif Systems (ESP32, n.d.). The development board used for this project is the ESP32-DevKitC V4, which features the WROOM-32E module. This module has dual core processing and integrated WI-FI and Bluetooth connectivity. The development board is very versatile in its use from simple IOT projects to more complex industrial applications [1] .This is a 38-pin development board. Certain pins are restricted to input only (GPIO 34-39) and restrictions on pins GPIO6-11 used to flash [2] .Peripherals can be connected using jumper wires or the ESP32-DevKitC V4 can be mounted on a breadboard [3] .This development board was a good choice for the project due to its low cost, processing power, integrated WIFI for setting up the network, I2C capabilities and a wide range of peripheral interfaces [4] .For this project the ESP32 was mounted on a breadboard to facilitate the force sensitive resistor and sensor interfacing. I interfaced the accelerometer, heart rate sensor and FSR to the ESP32. The accelerometer (MMA8451) and heart rate sensor (MAX30100) were interfaced with the ESP32 via I2C communication, using the dedicated I2C pins — GPIO 21 (SDA) and GPIO 22 (SCL). The FSR was connected to analog input pin **X** (*update with actual pin used*), allowing for measurement of applied force through voltage level changes. Firmware for the ESP32 was developed and uploaded using the Arduino IDE. The code includes the following libraries:

- Adafruit_MMA8451.h and Adafruit_Sensor.h for accelerometer data acquisition
- MAX30100_PulseOximeter.h for heart rate and SpO2 monitoring
- Wire.h for I2C communication
- WiFi.h to establish Wi-Fi connection
- HTTPClient.h to facilitate HTTP requests and data transmission to the Node.js backend server

Together, these libraries enable real-time sensor interfacing, WiFi communication, and HTTP data transfer functionalities on the ESP32. Overall, the ESP32 served as a reliable and efficient development board for sensor integration and network communication throughout the project.

7.1.2 Accelerometer

For this project I used the Adafruit MMA8451 Accelerometer Breakout. It is based on the Freescale Xtrinsic MMA8451Q Tri-axis, capacitive accelerometer IC [5]. I chose this sensor due to its low cost but high precision. The MMA8451 is the most precise model in a family of accelerometers and features a built in 14-bit ADC (analogue to digital converter) [6]. This Sensor can be used to detect motion, tilt and basic orientation and is compatible with both Arduino and the ESP32 microcontroller [7].

A Tri-axis accelerometer means the sensor can measure acceleration in three orthogonal directions [8] meaning three spatial directions at right angles to each other (X,Y,Z) [9]. This allows for detection of movement in multiple directions which is important for monitoring dynamic motion.

Capacitive sensors work by altering their capacitance in response to the acceleration they experience. Two capacitive plates have a diaphragm placed between them, when the sensor experiences acceleration the diaphragm can slightly move which changes the distance between the plates. The change in distance of the plates changes the capacitance of the sensor [10]. Since the change in capacitance is proportional to the applied acceleration, it can be measured and converted into an acceleration value. The accelerometer was interfaced with the ESP32, and sensor data was transmitted via the I^2C communication protocol

7.1.3 Heartrate

For the project, I chose to use the Heart Rate Click board manufactured by MIKROE. The Heart Rate Click features the MAX30100, which is an integrated heart rate sensor and pulse oximeter IC. The advantages of using this click board include its low cost, low power consumption, and built-in ADC (Analog-to-Digital Converter). Both heart rate and blood oxygen saturation levels (SpO2) can be read via the Inter-Integrated Circuit (I2C) interface. Another useful feature is Ambient Light Cancellation (ALC), which ensures that ambient light does not interfere with the signal, keeping the readings clean and accurate [11].

The MAX30100 sensor has two integrated LEDs, one red and one infrared (IR). Light is detected by a red/IR photo detector built into the sensor. As the heart pumps blood round the body the blood vessels expand and contract. When you place your finger on the sensor the amount of light detected increases and decreases depending on the amount of blood passing through the blood vessels of the finger. The increase and decrease of light the photo detector receive can be measured as the amount of light absorbed is proportional to the concentration of the light absorbing substance (Beer's law) [12]

The IR LED is mainly used for measuring pulse, as it penetrates deeper into tissue. As blood volume changes with each heartbeat, the amount of IR light detected by the sensor also changes, allowing the signal to be measured. The oxygen level reading works on the same principle but uses both the IR and red LEDs. Oxygenated blood, which contains more hemoglobin (Hb), absorbs more infrared light, while deoxygenated blood absorbs more red light. By comparing the ratio of absorbed red to infrared light, the oxygen saturation level (SpO_2) can be calculated [13].

7.1.4 ESP-EYE Camera

The Camera module used in this project is the ESP_EYE V2.2 DevKit used for image recognition and speech signal processing. The ESP- EYE is recommended for Artificial Intelligence of Things (AIOT), allowing for the combination of AI technology with Internet of Things (IOT) applications. The camera is a good choice as it is easily integrated with this project. The ESP-EYE V2.2 is a self-contained development board which has its own ESP32 chip integrated. The ESP-EYE features face detection and recognition capabilities and a digital microphone for voice control functions. It is simple and straight forward to use with lots of support online.

7.2 Software

This section will be broken down into two areas. The C++ embedded code developed in the Arduino IDE (Integrated Development Environment) for the ESP32 and the ESP-EYE Devkit. The second section will cover node.js backend.

7.3 Arduino IDE

I will explain some of the key parts of the logic developed for the accelerometer (MMA8451) and the Heart Rate Click Board (MAX30100 sensor). These libraries which provide the code to allow the use of Wifi connection, I2C, sending HTTP requests and controlling the sensors. Libraries save time and simplify development. I've also included the pins defined for Serial data and serial clock on the ESP32 for I2C communication.

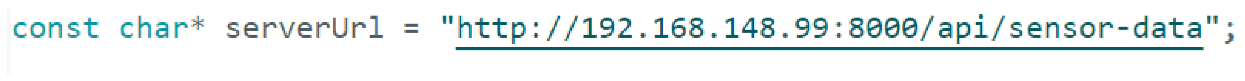
```
#include <Wire.h>                     // Enables I2C communication
#include <WiFi.h>                     // Enables WIfi capabilities
#include <HTTPClient.h>               // Library for sending HTTP requests (GET, POST) to web server
#include <Adafruit_MMA8451.h>         // Library with extra support for MMA8451 accelerometer (sensor)
#include <Adafruit_Sensor.h>          // Library used for Adafruit sensors
#include "MAX30100_PulseOximeter.h"   // Library for MAX30100 (Heart rate and oxygen)

#define SDA_I2C 21  // defining SDA  pin 21    Default pins of the esp32 for I2C communication
#define SCL_I2C 22  // defining SCL  pin 22
```

Here I have created MMA8451 accelerometer object to access sensor functions like reading acceleration.

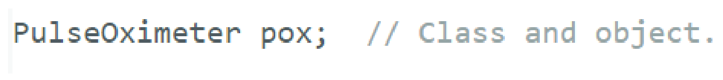
```
Adafruit_MMA8451 mma = Adafruit_MMA8451();
```

A constant pointer to a character array which holds the Server URL where sensor data will be sent. Contains the IP address of server port number the server is listening on for incoming connections and the API endpoint where images are been sent.



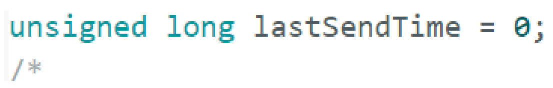
```
const char* serverUrl = "http://192.168.148.99:8000/api/sensor-data";
```

Creates an object of the PulseOximeter class. “pox” is the object (instance) created of the PulseOximeter class. “pox” will be used to call the functions from the PulseOximeter class -e.g pox.getHeartRate() or pox.begin().



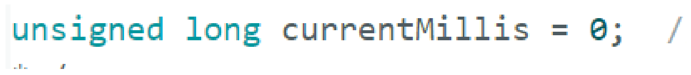
```
PulseOximeter pox;  // Class and object.
```

Stores the timestamp of the last time data was sent. Used later to check if enough time (e.g., 5 seconds) has passed before sending new data to server.



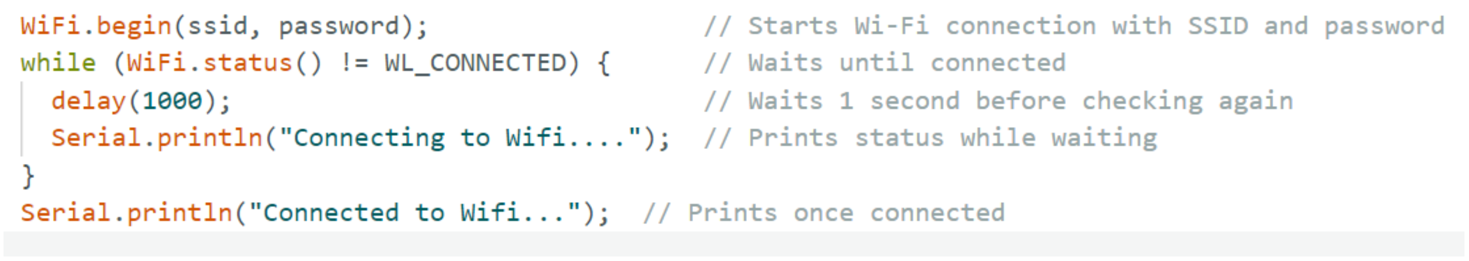
```
unsigned long lastSendTime = 0;
/*
```

Holds the current time in milliseconds since the program started, updated each loop. Used later to track time intervals for sending data to server.



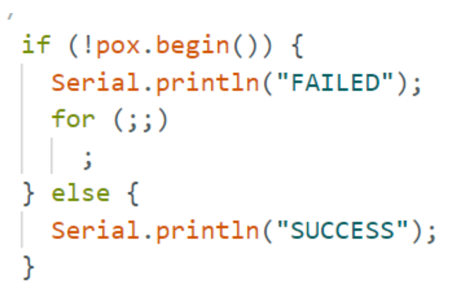
```
unsigned long currentMillis = 0;  /
```

Connects to Wi-Fi using the provided SSID and password. Waits until the connection is established, printing status updates.



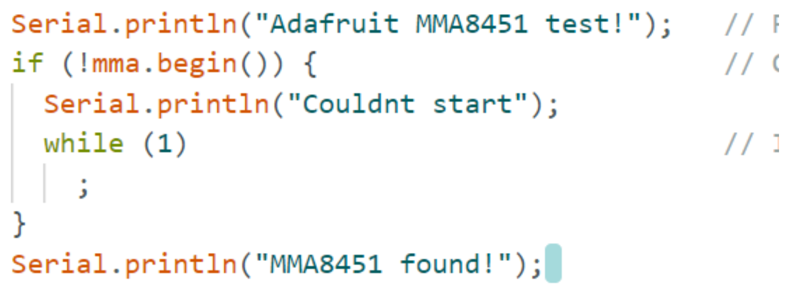
```
WiFi.begin(ssid, password);                    // Starts Wi-Fi connection with SSID and password
while (WiFi.status() != WL_CONNECTED) {        // Waits until connected
  delay(1000);                                 // Waits 1 second before checking again
  Serial.println("Connecting to Wifi....");   // Prints status while waiting
}
Serial.println("Connected to Wifi...");  // Prints once connected
```

Attempts to initialize the pulse oximeter sensor. If initialization fails pox.begin() returns false, prints "FAILED" and halts the program with an infinite loop. Otherwise, pox.begin() (returns true) and prints "SUCCESS". Infinite loop prevents running faulty logic or crash later in logic



```
if (!pox.begin()) {
  Serial.println("FAILED");
  for (;;)
    ;
} else {
  Serial.println("SUCCESS");
}
```

Initializes the MMA8451 accelerometer sensor. If initialization fails mma.begin() returns false, then prints message "Couldn't start" and halts the program with an infinite loop. If successful mma.begin() returns false, prints message "MMA8451 found!".

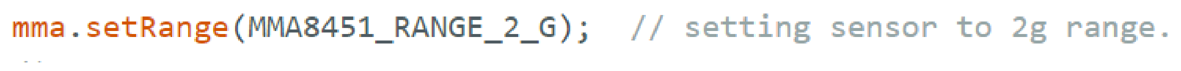


```
Serial.println("Adafruit MMA8451 test!");    // F
if (!mma.begin()) {                           // (
  Serial.println("Couldnt start");
  while (1)                                   // I
    ;
}
Serial.println("MMA8451 found!");
```

The accelerometer sensor range is set to ±2g below. The MMA8451 supports ranges of +-2g, +-4g, +-8g. A "g" refers to the acceleration due to gravity at Earth's surface (approx: 9.80665 m/s^2).

In +-2g mode the MMA8451 acts as a 14-bit sensor, outputting raw values between -8192 and +8191 which corresponds to -2^{13} to $2^{13}-1$. Each axis (X, Y, Z) provides a raw value within this range. The sensor outputs 4096 counts per g in +-2g mode because the full 14-bit output range (from -8192 to +8191) covers the entire +-2g acceleration range (from -2g to +2g).

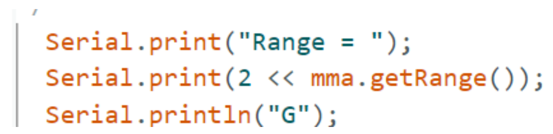
+-2g mode is more sensitive to smaller movements in acceleration. Better for fall detection and shaking. +-4, +-8g best for measuring higher accelerations but less sensitive to smaller movements in acceleration.



```
mma.setRange(MMA8451_RANGE_2_G);  // setting sensor to 2g range.
```

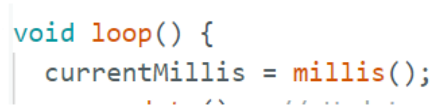
mma.getRange returns an integer value that represents the range setting (0 => +-2G, 1 => +4G,

2 => +-8G). 2 << mma.getRange() use's bitwise left shift (<<) operator. Shifts binary representation of a number to the left by set number of positions (X << N) X left by N number of bits or X * 2.



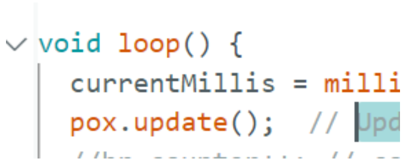
```
Serial.print("Range = ");
Serial.print(2 << mma.getRange());
Serial.println("G");
```

I have used a if statement to send data to server every 5 seconds to prevent overloading the backend as part of trying to solve an issue with heart rate sensor freezing and not sending new values. Here I have assigned millis() which is the time since program started to currentMillis at the start of the loop.



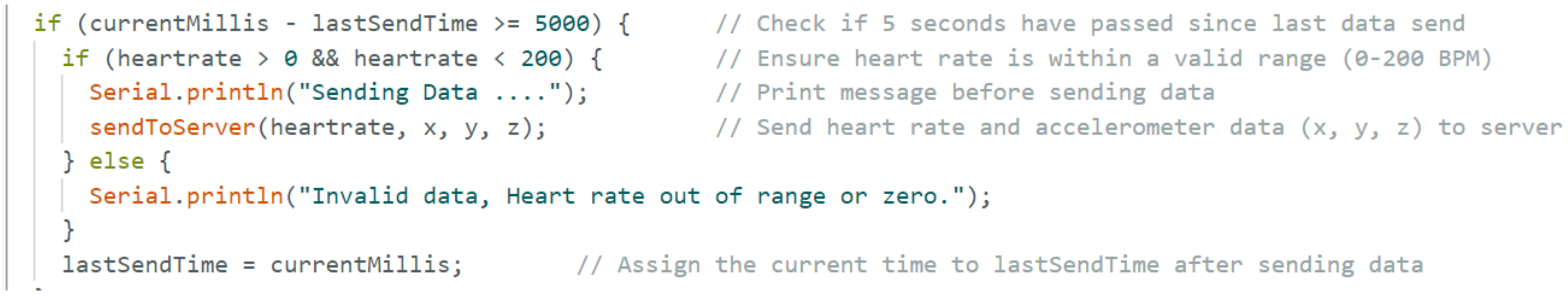
```
void loop() {
  currentMillis = millis();
```

pox.update(),updates heart rate and pulse oximeter readings and checks for new data (e.g., heartbeats) every loop.



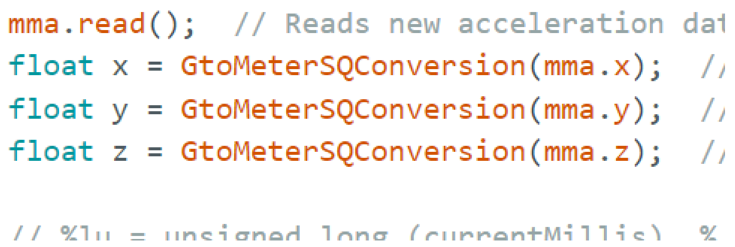
```
void loop() {
  currentMillis = milli
  pox.update();  // Upd
```

This if statement was to prevent overloading the server with requests possible issue with sensor freezing. millis() at start of loop returns the current time in milliseconds since the program started. currentMillis = millis(); updates currentMillis with the current time. currentMillis holds the current time. lastSendTime stores the time when the data was last sent. Then the if statement compares the current time (currentMillis) with the last sent time (lastSendTime) to check if 5 seconds has passed. If it's 5 seconds (5000ms) or more the data is sent to the server. Once data is sent currentmillis gets assigned to lastSendTime and loop continues till data can be sent again.



```
if (currentMillis - lastSendTime >= 5000) {      // Check if 5 seconds have passed since last data send
  if (heartrate > 0 && heartrate < 200) {        // Ensure heart rate is within a valid range (0-200 BPM)
    Serial.println("Sending Data ....");         // Print message before sending data
    sendToServer(heartrate, x, y, z);            // Send heart rate and accelerometer data (x, y, z) to server
  } else {
    Serial.println("Invalid data, Heart rate out of range or zero.");
  }
  lastSendTime = currentMillis;        // Assign the current time to lastSendTime after sending data
```

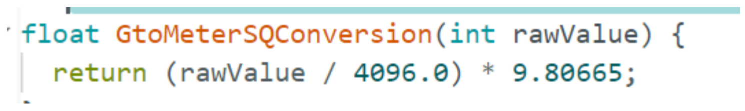
The function call to read new accelerometer data from the MMA8451 sensor and the function calls for conversion to meters per second squared which are assigned to type floats variables x, y, z. Values used in payload.



```
mma.read();  // Reads new acceleration dat
float x = GtoMeterSQConversion(mma.x);  //
float y = GtoMeterSQConversion(mma.y);  //
float z = GtoMeterSQConversion(mma.z);  //

// %lu = unsigned long (currentMillis)  %
```

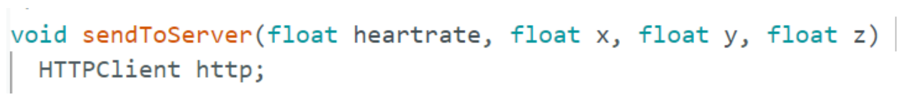
Function for conversion of rawValues to meters per second squared 1G = 9.80665 m/s^2



```
float GtoMeterSQConversion(int rawValue) {
  return (rawValue / 4096.0) * 9.80665;
```

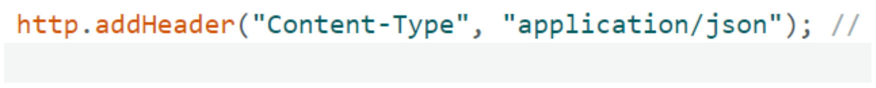
Function to send heart rate and accelerometer data to the server (non-blocking). Function will send data to the server without blocking other tasks.

HTTPClient is used for HTTP requests. Creating an object of the class HTTPClient http; allows the use of methods and properties provided by the HTTPClient class to make HTTP requests. The object is used to initialize and manage the HTTP connection, Set headers, timeouts and send data to the server and handle the response from the server.



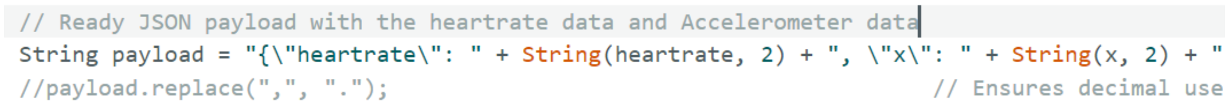
```
void sendToServer(float heartrate, float x, float y, float z)
  HTTPClient http;
```

Prepares the HTTP request by setting the appropriate header for sending data in JSON format. Tells the server the data being sent is in JSON format. REST APIs commonly use JSON as the data format for requests and responses. Specifying this header tells the server that the body of the HTTP request will contain JSON-formatted data (string). The server will then know how to properly parse and interpret the data sent in the request. Parsing allows the server to extract the actual values from raw JSON data and convert them to usable variables and objects which the server can work such as save to a data base or send a response to client.

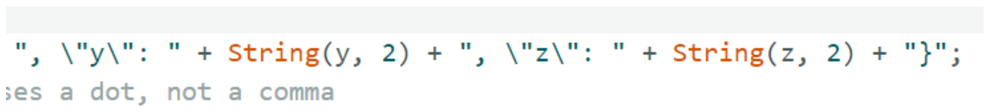


```
http.addHeader("Content-Type", "application/json"); //
```

JSON payload with the heartrate data and accelerometer data to be sent to the node.js server.

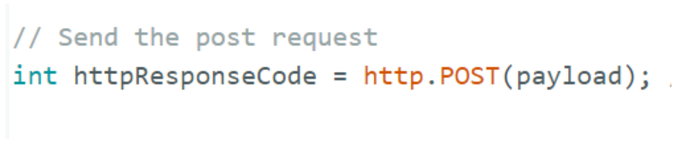


```
// Ready JSON payload with the heartrate data and Accelerometer data
String payload = "{\"heartrate\": " + String(heartrate, 2) + ", \"x\": " + String(x, 2) + "
//payload.replace(",", ".");                                          // Ensures decimal use
```



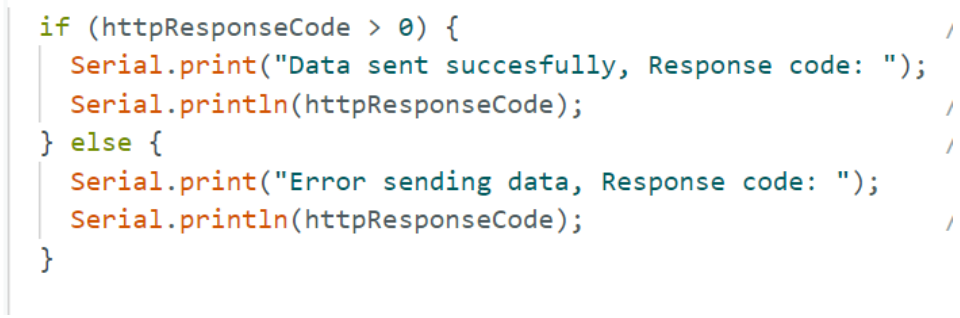
```
", \"y\": " + String(y, 2) + ", \"z\": " + String(z, 2) + "}";
ses a dot, not a comma
```

Sends the HTTP POST request with the payload (data) to the server



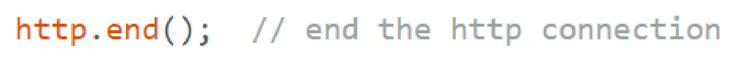
```
// Send the post request
int httpResponseCode = http.POST(payload);
```

HTTP response code returned by the server will be > 0 meaning success response (HTTP 200) or < 0 which means an error response (HTTP 404 or 500). Checks if the server response is greater than 0, indicating success. Prints the success message and the HTTP response code (200). If the server response is less than 0 there was an error sending data. Prints the error message and the HTTP response code (404 or 500)



```
if (httpResponseCode > 0) {                               /
  Serial.print("Data sent succesfully, Response code: ");
  Serial.println(httpResponseCode);                       /
} else {                                                  /
  Serial.print("Error sending data, Response code: ");
  Serial.println(httpResponseCode);                       /
}
```

Ends connection



```
http.end();  // end the http connection
```

7.4 Node.js

Node.js is an open-source, cross platform runtime environment that executes JavaScript code outside a web browser. It’s built on Chrome’s V8 JavaScript engine allowing developers to build scalable, server-side applications in JavaScript [14]. Node.js is used to power the backend server, handling requests and providing API endpoints for the application.

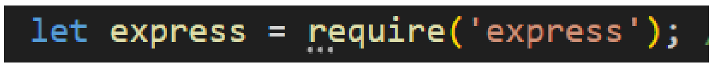
Express JS was also used to build the backend. Express.js is a lightweight framework built on top of Node.js. Simplifies the server-side development, the building web applications, APIs and provides tools and utilities for handling HTTP requests and responses. Supports middleware which handles HTTP requests such as POST, and errors [15].

7.5 Backend

When the sensor payload is sent to the backend, Express.js recognizes that the incoming data is in JSON format, thanks to the Content-Type: application/json header. Using its middleware (express.json()), Express.js automatically parses the JSON string into a JavaScript object.

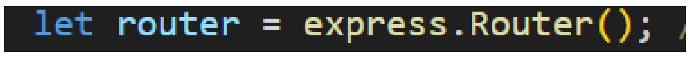
The router.post('/sensor-data') defines the /sensor-data URL, which listens for POST HTTP requests. When sensor data from an ESP32 is sent to the endpoint, the route logic extracts the heartrate, x, y, and z values from the parsed data, validates that all required fields are present, logs the received sensor readings, and sends a success response back to the ESP32.

Imports Express framework to create routes and handle requests.



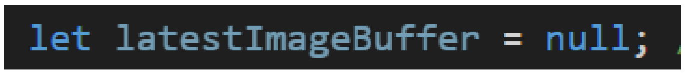
```
let express = require('express');
```

Creates a mini router which can be used to define API endpoints (GET, POST).



```
let router = express.Router();
```

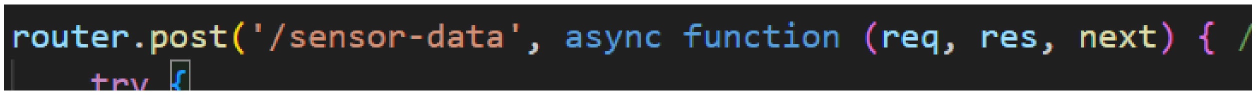
This is dynamically updated with latest image from my camera.



```
let latestImageBuffer = null;
```

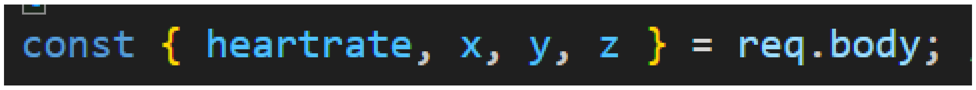
Backend Sensor Router Post:

Route to receive sensor data (acceleromter & heart rate). Defines post route at "/sensor-data". async keyword, allows handling of asynchronous operations (e.g database queries).



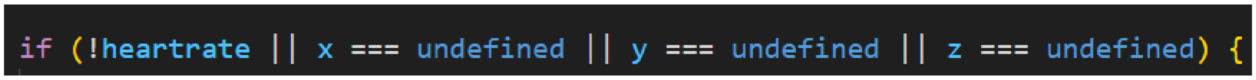
```
router.post('/sensor-data', async function (req, res, next) { /
    try {
```

Extracts values (heart rate) from the incoming JSON body of the request.



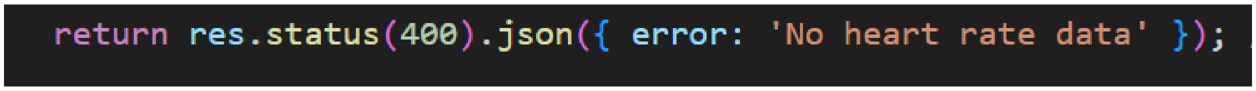
```
const { heartrate, x, y, z } = req.body;
```

Condition to check if heartrate is missing or null. !accelerometer returns true if the accelerometer data is missing and !heartrate returns true if heart rate data is missing.



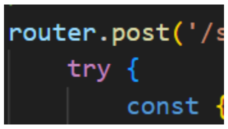
```
if (!heartrate || x === undefined || y === undefined || z === undefined) {
```

Stops request and sends 400 status code Bad Request error. Sends JSON response with error message

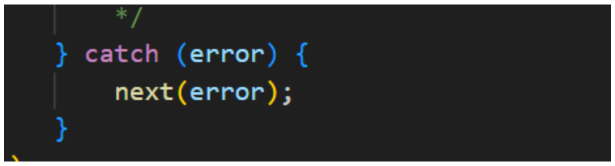


```
  return res.status(400).json({ error: 'No heart rate data' });
```

Part of Try catch statement to handle errors inside async function. If an error occurs in the try block, the catch block catches it and passes the error to Express's built-in error handler using next(error). Express.js automatically processes it and sends a standard error response.



```
router.post('/s
    try {
        const {
```



```
        */
    } catch (error) {
        next(error);
    }
```

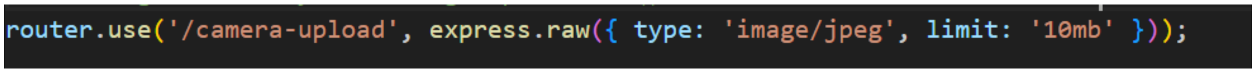
Backend Middleware router use:

Middleware to parse incoming JPEG image data for /camera-upload as a raw buffer up to 10MB in size express.raw() tells Express.js not to parse the body as JSON or text so keeps it as a raw binary buffer. Type: 'image/jpeg' means it only applies this raw parsing to requests where the Content-Type is image/jpeg. limit: '10mb' prevents uploading huge images beyond 10 megabytes.

I'm using a raw binary buffer because the ESP32 sends the image data as binary, not as JSON or text. express.raw() allows the capture of the full image exactly as it is without trying to parse or modify it. This allows the image to be safely handled and processed without corrupting the data. If I sent it as JSON I would have to encode the image making it slower to transmit.

Before the router post express.raw() middleware is set up to handle the raw image data and ensures it's available in req.body in the correct format when the router.post() logic runs.

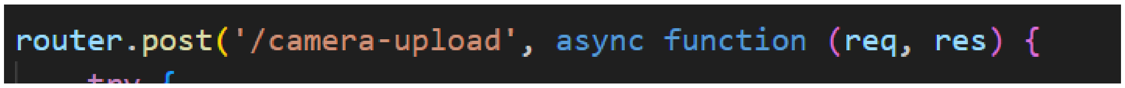
The router.use() sets up the rules (middleware) for handling requests. router.post() processes the actual route logic after the middleware has done its job.



```
router.use('/camera-upload', express.raw({ type: 'image/jpeg', limit: '10mb' }));
```

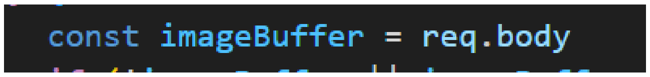
ob

Post route at URL /camera-upload. The req (request) object contains the contains the incoming data (image). The res (response) object will be used to send the response to the ESP32.



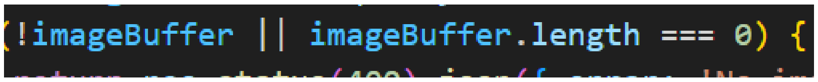
```
router.post('/camera-upload', async function (req, res) {
```

Extracts the raw image buffer data from the request body



```
  const imageBuffer = req.body
```

Checks if no image data was received (empty or undefined)



```
(!imageBuffer || imageBuffer.length === 0) {
```

8 Communication Protocol

8.1 Inter Integrated Communication (I2C)

I2C is a two-wire serial communication protocol using a serial data line (SDA) and a serial clock line (SCL) [16]. I2C allows multiple Slaves (e.g sensors) can be connected to a master (e.g esp32) or multiple masters can control multiple slaves. Data is transferred and received bit by bit on the single wire of the serial data line (SDA) [17]. The serial clock line (SDA) sets the timing for the data transfer, this tells the devices when to read or when to write a bit. I2C is synchronous so the clock keeps everything in sync. Clock signal is always controlled by the master [18]. Each slave has its own address (Hex number) which allows the master (e.g. esp32) to communicate with the slaves (e.g. sensors).

Data is transferred in messages. These messages are broken down into data frames. The messages contain an address frame containing the binary address of the slave, a frame or more with the data being transmitted. Message also includes start stop conditions, read/write bits and ACK/NACK between each data frame [19]. ACK (Acknowledge) and NACK (Not Acknowledge) are used to answer back to the master with a bit. The Master receives an ACK (sent 0) to when slave receives the sent data and slave sends a NACK (sent 1) if something wrong or to stop [20].

The steps for I2C to transmit data are as follows. Master sends start condition to all slaves by changing the SDA line from high to low, then changes the SCL line from high to low. Master sends the address of slave it wants to communicate with plus a read/write bit. Each slave compares sent address from master with its own address. Whichever slave the address matches returns an ACK (0) by pulling the SDA line low for a one bit. All other slaves ignore and leave SDA line high. Master sends or receives data frame. When each data frame is transferred the slave returns a ACK bit to acknowledge receiving frame successfully. To stop transmission master sends stop condition by switching SCL high and then switching SDA high [21].

I2C is an open drain system (Devices can only pull the bus low, not push high). For I2C communication to work, pull up resistors are needed. Devices pull the communication path (I2C bus) low (connect to ground) to represent a 0. The pull up resistor then pulls the bus back up to 3.3V or 5V because devices cannot push the bus high to represent a 1.

This setup allows multiple devices (sensors, ESP32) to safely share the same bus (communication path) without competing. The pull-up resistor ensures the bus stays

high when no device is pulling it low (idle state) or if device needs to send a 1 (bit) [22].

9 Ethics

The Safe View Monitoring System would raise many ethical concerns that would have to be carefully addressed. The fact that users and carers will place a lot of trust in the system to ensure a person's safety, it would be important that the system did not claim to perform functions that it could not. This would lead to serious risks for vulnerable individuals using this system.

The system would be classified as a medical or assistive device so rigorous testing would have to be conducted to ensure that the system performed as intended. Failure in its functionality could place the vulnerable person using the system at high risk. Ensuring reliability, accuracy, and safety of the system would have to be top priority.

Security of the system would also be a major concern with clear risks of cyber attacks like recent ransomware frauds on medical facilities. Stored medical data or medical data been transferred would have to be given the highest cyber encrypted security possible. Access to the system and records would have to be strictly limited to authorized personnel. The system hardware been hacked and malfunctioning putting user again at risk would have to be secured against unfortunately in today's world. Compliance with relevant data privacy regulations, for example GDPR would have to be adhered to. Also, affordability of system would have to come into the ethical consideration as the system could not exclude users due to their financial situation. Should be accessible by as many users as possible without losing quality and top functionality.

10 Conclusion

This project set out to create a real-time monitoring system aimed at improving safety for individuals with health conditions that put them at risk. Although the full system was not completed, the core goal of establishing communication between the sensors, camera, and server was successfully achieved. Throughout the project, I gained valuable experience in both hardware and software development, as well as important lessons in project management, problem solving and critical thinking. Overall, the project has been good experience of the challenges when undertaking a project incorporating hardware, software, design, development, problem solving, project management and adhering to deadlines.

The system proved the concept of real-time monitoring with integrated sensor data and video, laying a foundation for future improvements. After a completed prototype is available and working, the opportunities for further development would include incorporating AI to oversee certain system functions and assist with pattern recognition. This pattern recognition could play a role helping to diagnose patients who may have underlying or unrecognized issues, such as heart condition or sleep apnea. Training the camera function through AI tools to recognize footage, such as a person in distress or after having a fall, could enhance the system's responsiveness. However, human oversight would always be retained, remaining a key part of its operation.

11 References

[1] “ESP32 WROOM 32E,” SunFounder, [Online]. Available: https://docs.sunfounder.com/projects/esp32-starter-kit/en/latest/components/component_esp32_extension.html. [Accessed 24 April 2025].

[2] "ChatGPT," OpenAI, [Online]. Available: https://chatgpt.com/. [Accessed 24 April 2025].

[3] "ESP32-DevKitC V4," Espressif Systems, [Online]. Available: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html. [Accessed 24 April 2025].

[4] "ESP32 WROOM 32E," SunFounder, [Online]. Available: https://docs.sunfounder.com/projects/esp32-starter-kit/en/latest/components/component_esp32_extension.html. [Accessed 24 April 2025].

[5] "ADAFRUIT INDUSTRIES 3-axis Sensor, Module, I2C, 16-Pin," RS components, [Online]. Available: https://ie.rs-online.com/web/p/motion-sensor-ics/9054665?searchId=52712cfe-7318-42d1-9b6a-ae3341895429&gb=s. [Accessed 24 April 2025].

[6] "Adafruit MMA8451 Accelerometer Breakout," Adafruit, [Online]. Available: https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout/overview. [Accessed 24 April 2025].

[7] "Adafruit MMA8451 Accelerometer Breakout," adafruit, [Online]. Available: https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout/overview. [Accessed 24 April 2025].

[8] "How to Measure Acceleration?," DWYEROMEGA, [Online]. Available: https://www.dwyeromega.com/en-us/resources/accelerometers. [Accessed 24 April 2025].

[9] "ChatGPT orthogonal," OpenAI, [Online]. Available: https://openai.com/index/chatgpt/. [Accessed 24 April 2025].

[10] "How to Measure Acceleration?," DYWEROMEGA, [Online]. Available: https://www.dwyeromega.com/en-us/resources/accelerometers. [Accessed 24 April 2025].

[11] "HEART RATE CLICK," MIKROE, [Online]. Available: https://www.mikroe.com/heart-rate-click?srsltid=AfmBOopt_7qTjN2-58Ukygn3jMGFLO26i59-YXsQlr92riEXTReWlF5H. [Accessed 24 April 2025].

[12] "How pulse oximeters work explained simply," How Equipment Works, [Online]. Available: https://www.howequipmentworks.com/pulse_oximeter/. [Accessed 24 April 2025].

[13] "Chat GPT," openAI, [Online]. Available: https://chatgpt.com/c/680b8ab7-0ea4-8011-abd7-fdb2b6ecea4b. [Accessed 24 April 2025].

[14] "Introduction to Node.js," OpenJS Foundation, [Online]. Available: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs#introduction-to-nodejs. [Accessed 24 April 2025].

[15] "Express.js & Node.js For Server Side Development," Medium, [Online]. Available: https://medium.com/@ravinduperera1229/express-js-node-js-for-server-side-development-99f489baf4d6. [Accessed 24 April 2025].

[22] “ESP32,” [Online]. Available: https://en.wikipedia.org/wiki/ESP32.

[23] "Adafruit MMA8451 Accelerometer Breakout," Adafruit, [Online]. Available: https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout/overview. [Accessed 24 April 2025].

[24] "How to Measure Acceleration?," DWYEROMEGA, [Online]. Available: https://www.dwyeromega.com/en-us/resources/accelerometers. [Accessed 24 April 2025].

[25] "Chat GPT," openAI, [Online]. Available: https://chatgpt.com/c/680b8ab7-0ea4-8011-abd7-fdb2b6ecea4b. [Accessed 24 April 2025].