

Technical Design

Healthcare system for the Elderly

CyberCare

Max Thielen 497441

Maurits Hameter 504233

Mihhail Tšulinda 498046

Mohammed Alghaithi 519740

Ladan Dehghantarzejani 501619

Muhammad Arief Kurniawan 495831

Table of Contents

Introduction.....	1
System Overview	1
Computing Unit Selection.....	2
System Unit Component Summary Tables	3
Wearable Unit Overview	3
Pill Dispenser Unit Overview	3
System Aspects	4
Mechanical.....	4
Pill Dispenser	4
Wearable	9
Electronic	11
Component List.....	11
Wearable Schematic:.....	13
Wearable 3D Model	20
Wearable PCB Layout:	22
Pill Dispenser Schematic	24
Pill Dispenser 3D Model.....	25
Pill Dispenser PCB Layout	25
Software	26
Wearable Unit	26
Pill Dispenser Unit	31
Server Unit	36
Web Interface.....	44

TABLE OF FIGURES

Figure 1 - System Block Diagram.....	1
Figure 2 - Wearable Unit Component Summary Table	3
Figure 3 - Pill Dispenser Unit Component Summary table	3
Figure 4 - 3D Design of Pill Dispenser.....	4
Figure 5 - Pill Dispenser Base Drawing.....	5
Figure 6 - Pill Dispenser Cover Drawing.....	6
Figure 7 - Pill Dispenser Pillbox Drawing.....	7
Figure 8 - Pill Dispenser Slot Drawing.....	8
Figure 9 - 3D Design of Wearable	9
Figure 10 - Wearable Base Drawing.....	10
Figure 11 - Wearable Schematic.....	13
Figure 12 - ESP32 Chip	14
Figure 13 - FT232RL Chip for USB - UART Communication	15
Figure 14 - Voltage Regulator Schematic.....	16
Figure 15 - Battery Management Schematic.....	17
Figure 16 - Protection IC Schematic.....	17
Figure 17 - Switching Transistors Schematic	18
Figure 18 - Pull Up Resistors Schematic	18
Figure 19 - Headers Schematic	19
Figure 20 - SD Card Schematic	19
Figure 21 - Wearable 3D Design Top View	20
Figure 22 - Wearable 3D Design Bottom View.....	21
Figure 23 - Pill Dispenser LEDs Schematic	24
Figure 24 - 3D Design Pill Dispenser LEDs Top View.....	25
Figure 25 - 3D Design Pill Dispenser LEDs Bottom View	25
Figure 26 - 3D Design Pill Dispenser LEDs Isometric View	25
Figure 27 - Wearable State Diagram.....	26
Figure 28 - Wearable UML Class Diagram.....	27
Figure 29 - Sending Sensor Readings Flowchart.....	29
Figure 30 - Receiving Message or Alert from Server	30
Figure 31 - Pill Dispenser State Diagram	31
Figure 32 - Pill Dispenser UML Class Diagram.....	32
Figure 33 - Alert Pill Taking Flowchart.....	33
Figure 34 - Receiving Message/Command/Alert from Server.....	35
Figure 35 - Database UML Class Diagram.....	36
Figure 36 - Concept Web Interface.....	44
Figure 37 - Web Interface Functional Flowchart.....	45

TABLE OF TABLES

Table 1 - Computing Unit Comparison Table	2
Table 2 - Wearable Unit Component List.....	12
Table 3 - Pill Dispenser Component List.....	12
Table 4 - Software Library Dependencies	28

Introduction

The purpose of this document is to provide a realisation of the system's final design. This consists of highlighting the design documents, schematics, simulations, measurements, and dependencies. This document references the development team's system requirements and functional design documents for reasoning and elaboration. With the system technical design, the team has produced pre-defined building procedures for the different system units. The scope of the procedures is fragmented into the various aspects of the system. The outcome of this document is that the manufacturing team can manufacture the intended system by simply reviewing the technical design.

System Overview

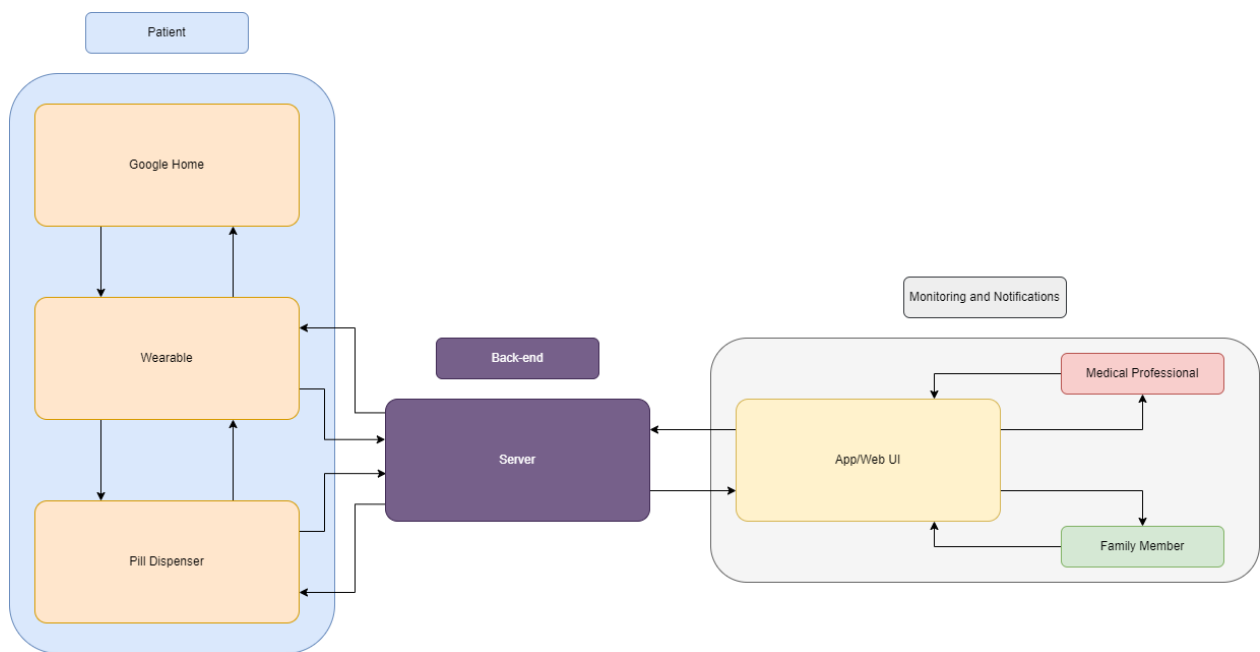


Figure 1 - System Block Diagram

The system consists of entities that are bound to different physical environments. The physical environments are patient and cloud (web-based) bound. The figure above illustrates that the wearable, pill dispenser, and google home units are patient bound and the latter devices are bound to the internet. The monitoring system can be accessed by the medical professional and if given permission, the family member can access it as well. The system server provides the back-end functionality of the system. The system server serves as a bridge for the entities to communicate with one another.

Computing Unit Selection

	Raspberry Pi	ESP32	Arduino ATmega328P
GPIO pins	40	34	19
Integrated graphics	yes	limited	no
Integrated audio	yes	no	no
Integrated wireless communication	Wi-Fi and Bluetooth	Wi-Fi and Bluetooth	no
Power consumption	medium	low	low
Type of machine	computer	microcontroller	microcontroller

Table 1 - Computing Unit Comparison Table

Wearable Unit

The ESP32 chip is chosen for the computing unit of the wearable due to its computational and networking capabilities. The ESP32 can be programmed using C++, Python, and the existing Arduino framework. The ESP32 provides an ample amount of GPIO pins for the integration of sensor modules. The ESP32 can perform on low power mode to preserve the battery life of the wearable unit.

Pill Dispenser Unit

The RaspberryPi is chosen as the computing unit of the pill dispenser. The RaspberryPi can be programmed in multiple programming languages including C++ and Python. This is important as the pill dispenser is required to communicate to both the server and the wearable. The RaspberryPi can provide the required networking capabilities for the system. The RaspberryPi has built-in graphic components that display graphics. This is useful as the pill dispenser unit is required to display information to the patient via a monitor. Furthermore, the RaspberryPi is equipped with sufficient GPIO pins for connecting LEDs of the pill segments.

System Unit Component Summary Tables

From the functional design, the team reviewed the possible components that the concept design should be equipped with. This section summarises the chosen technical components for the design of the wearable and pill dispenser units. The server and web interface are not summarised as these entities do not require physical components to function. The patient-specific sensors for both the wearable and pill dispenser units are not shown in the component summary tables as it is mentioned in detail in the Electronic section of this document.

Wearable Unit Overview








Wearable	Functions						
	Housing	Computing	Battery	Strap	User Input	Monitor	Feedback
Chosen Option	 3D Printed	 Custom PCB with ESP32	 Removable Battery	 Cotton	 Push Buttons	 LCD	 Text

Figure 2 - Wearable Unit Component Summary Table

The housing of the wearable unit is made from 3D printed material. The design houses a removable circular battery for power input, pushbuttons for user interaction, and an LCD monitor that can display the system responses. The patient can wear the wearable using the cotton strap attached to the edge of the 3D printed housing. The wearable design is accustomed to a custom PCB that houses the ESP32 chip.

Pill Dispenser Unit Overview







Pill Dispenser	Functions					
	Housing	Computing	Power	Feedback	User Input	Monitor
Chosen Option	 3D Printed	 RaspberryPi	 Main Electricity	 Voice	 Push Buttons	 LCD Monitor

Figure 3 - Pill Dispenser Unit Component Summary table

Like the wearable unit, the housing of the pill dispenser is 3D printed. The pill dispenser is powered by main electricity and uses pushbuttons for user interaction. The pill dispenser can respond either through voice output from the Google Home integration or display system responses on an LCD monitor.

System Aspects

The purpose of this chapter is to take the system overview and approach it from the three disciplines contributing to this project. First, the physical components' housing is elaborated on, including the final 3D designs together with all measurements. Next, each of the custom circuitry and necessary power calculations is discussed. This also includes the final component list necessary to assemble the final design. Lastly, the software is split into four sections starting with the wearable device and pill dispenser logic. Following the hardware software, the server functions are fleshed out before ending on the main functions of the web interface.

Mechanical

Pill Dispenser

In this section, we will elaborate on the chosen concept for the pill dispenser. The dispenser consists of 3 main parts: base, top cover, and pill slots. All the parts are going to be 3D printed with a 100% fill option to increase structural integrity.

3D render of the assembled unit:

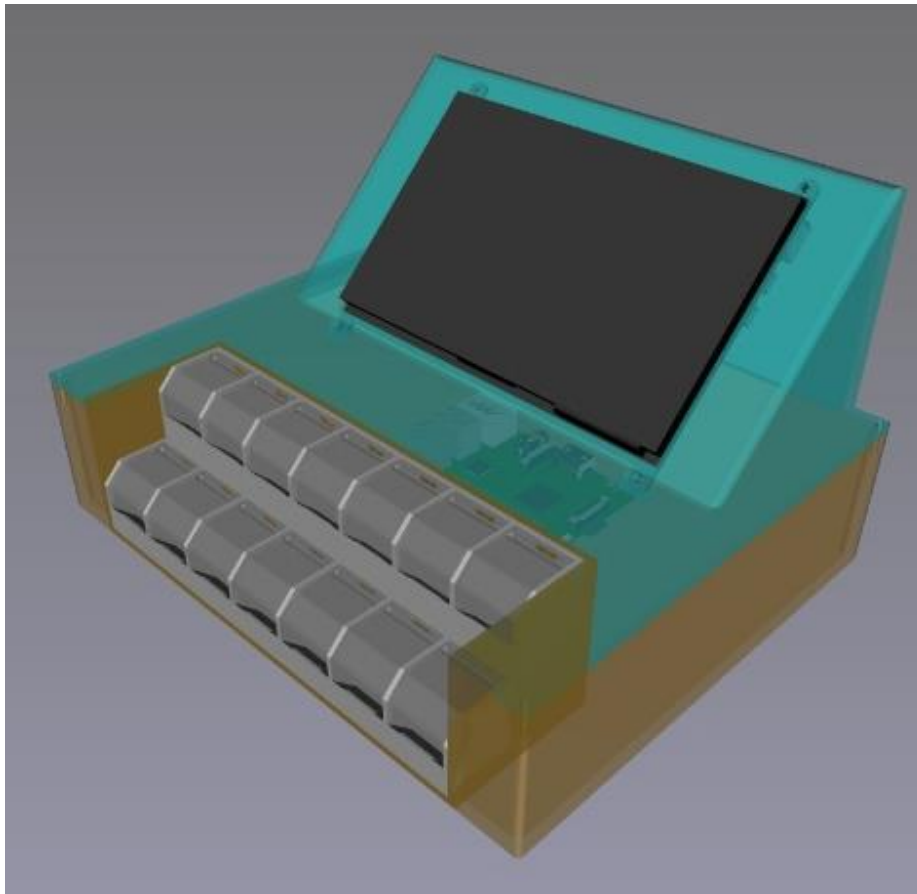


Figure 4 - 3D Design of Pill Dispenser

The base part acts as a container for the RaspberryPi, as well as a core structural element. In the front, there are shelves to house the pillbox with cut-outs for each container's LED. On the rear of the model, we have a passthrough for the power cable.

Technical drawing of the base part:

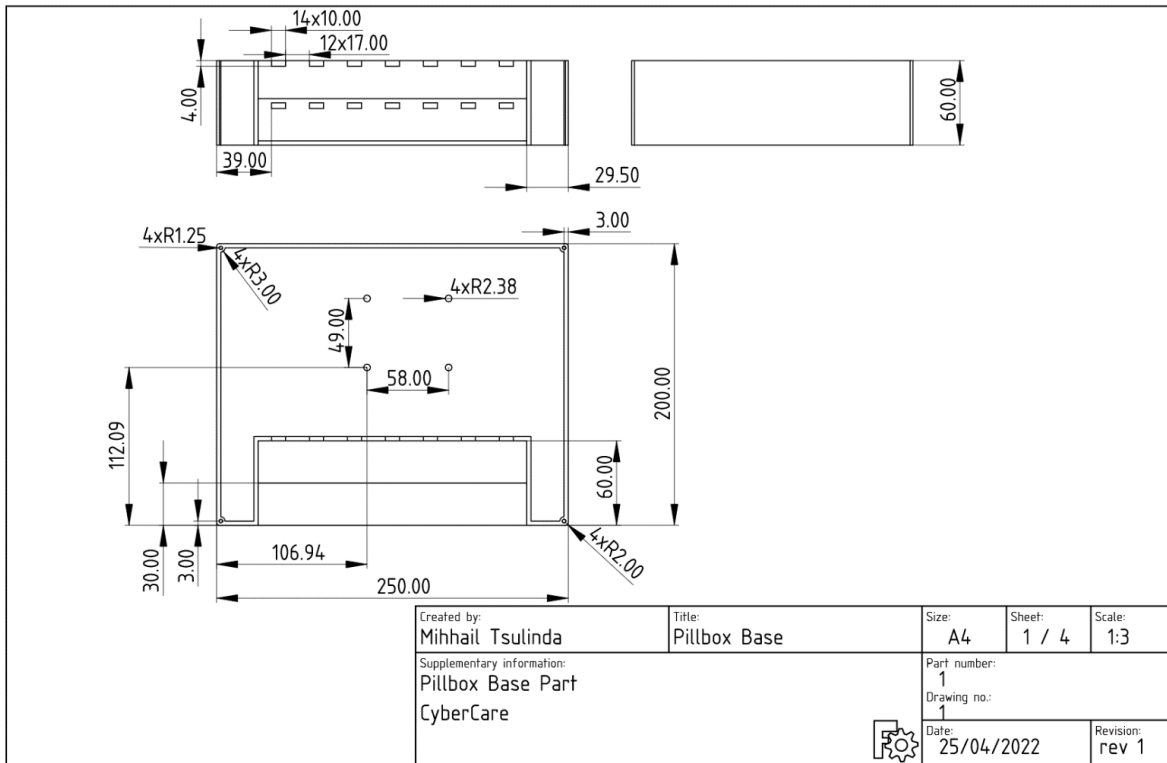


Figure 5 - Pill Dispenser Base Drawing

The cover sits on top of the base and houses our "7inch Capacitive Touch Screen LCD (C)". The display is angled towards the user to improve readability and ease of touchscreen use.

Technical drawing of the cover part:

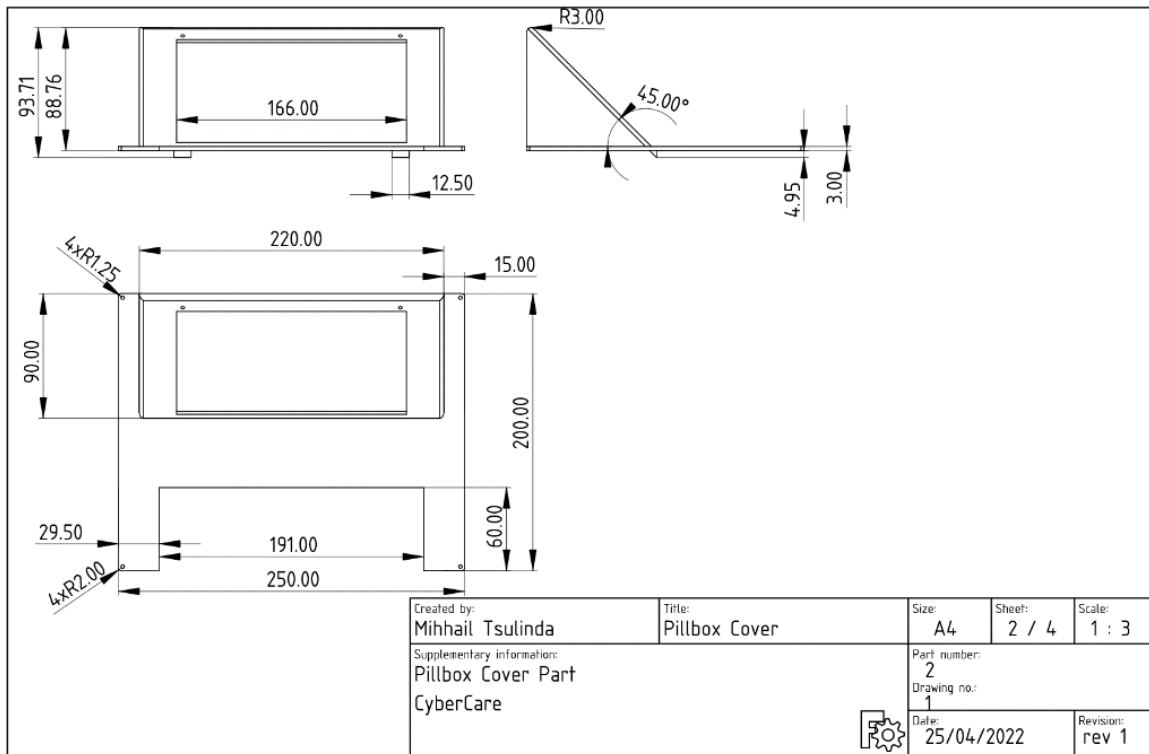


Figure 6 - Pill Dispenser Cover Drawing

The pillbox itself consists of seven slots; each slot has a cut-out for the LED. Slots were designed to be as ergonomic as possible to help elderly people. Inside the slot, there is a sloped surface to aid in the retrieval of pills. In the final assembly, we have two pillboxes above each other, both of which are removable for a simpler refill.

Technical drawing of the pillbox slots:

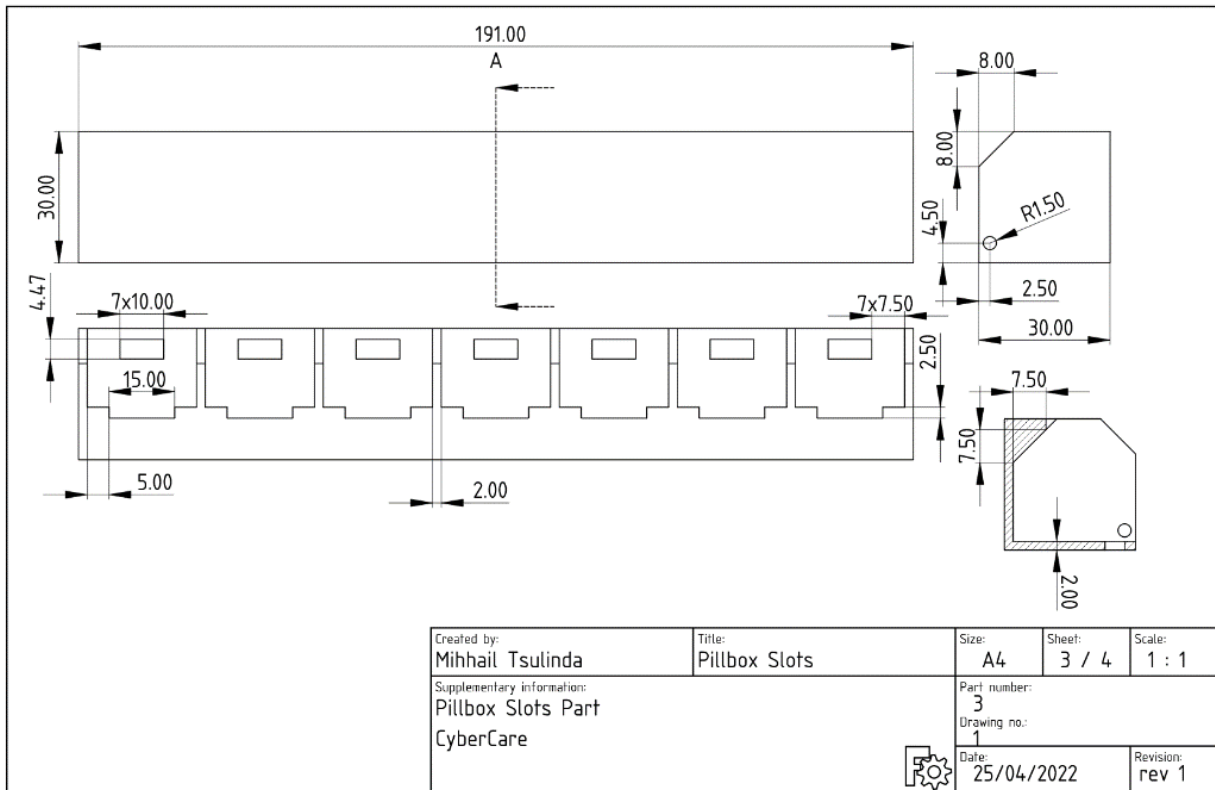


Figure 7 - Pill Dispenser Pillbox Drawing

On top of each separate slot, there is a transparent pivoting cover. The covers are held in place with a press fit. The final assembly has a total of fourteen of them.

Technical drawing of the slot's cover:

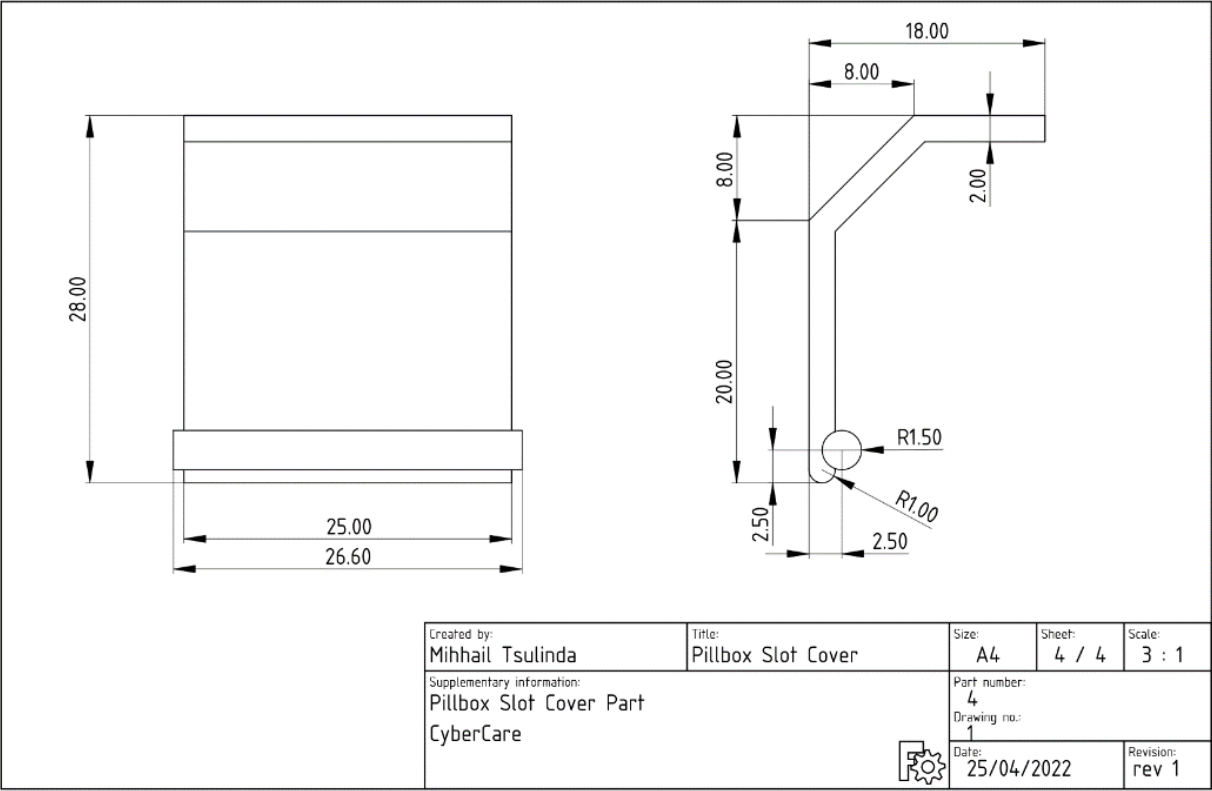


Figure 8 - Pill Dispenser Slot Drawing

Wearable

For the wearable, we have decided to go with a workbench style design for the first iteration of the product. This will allow us to test the fitting and placement of all the electronics while having a capable prototype. Our findings will aid us in refining the final design.

3D render of the wearable:

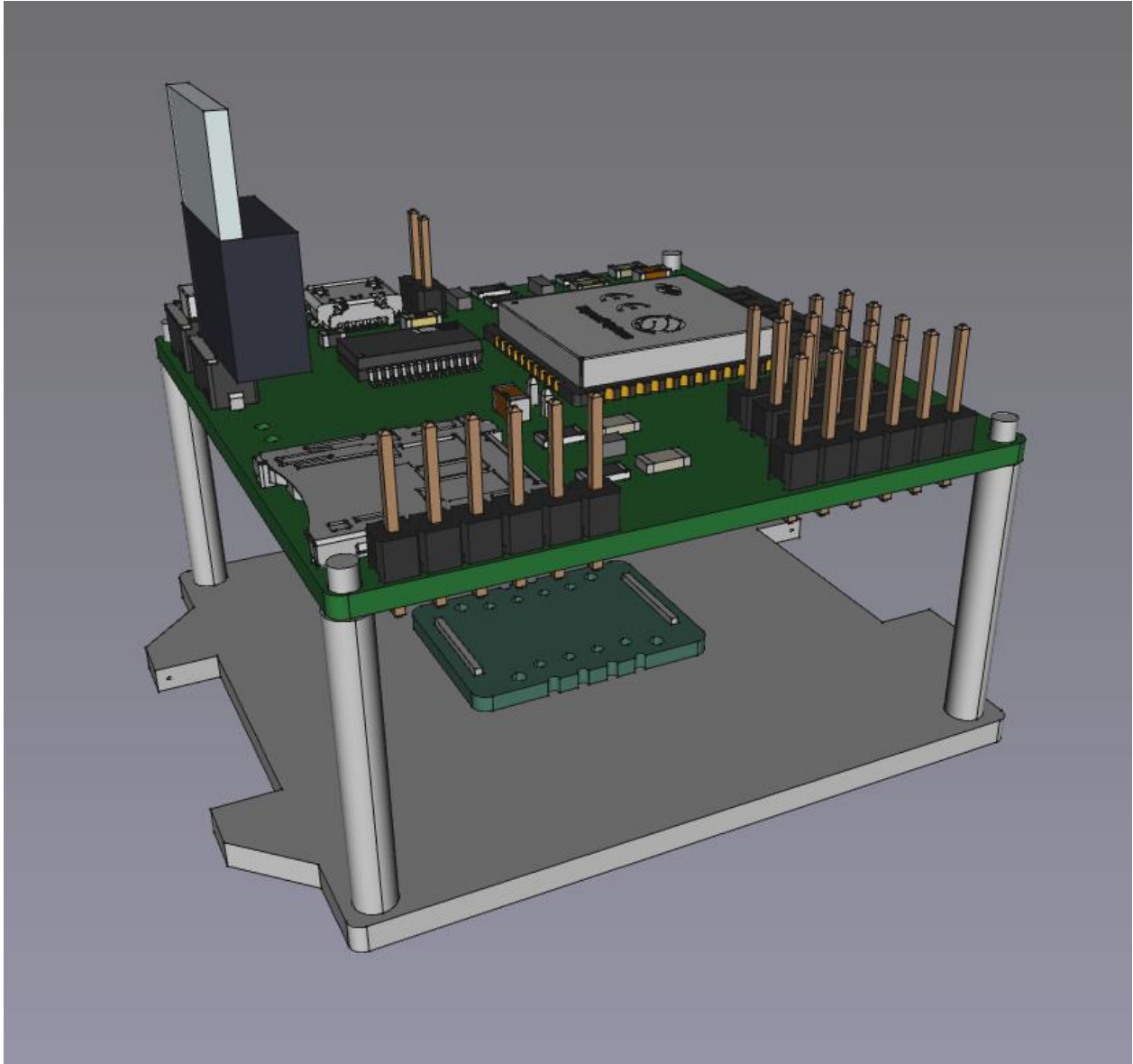


Figure 9 - 3D Design of Wearable

Two opposing sides have band lugs of somewhat standard size. For the band itself, we will be using an already acquired band. In the future, we will have in-place printed watchbands.

Technical drawing of the base part:

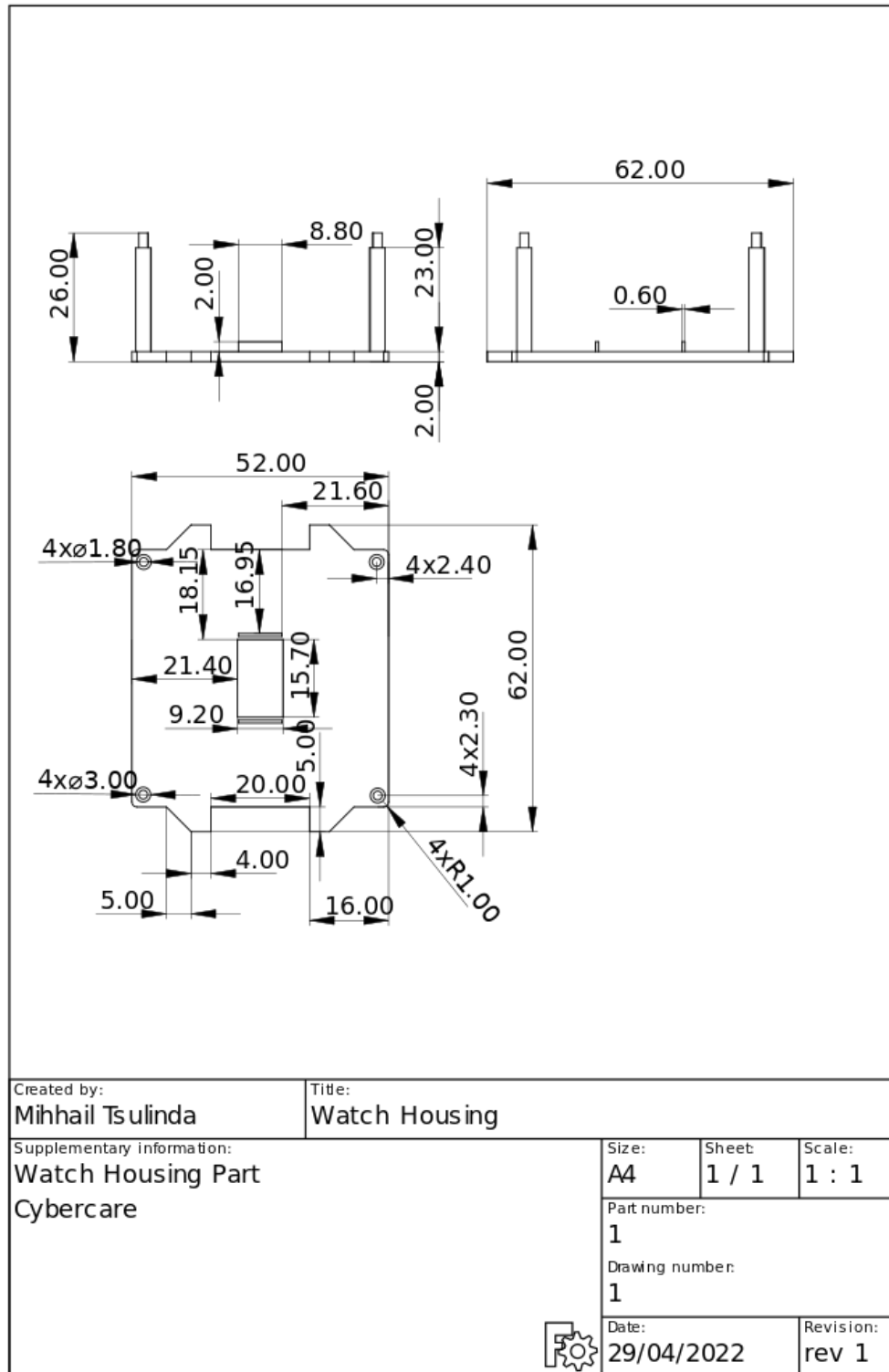


Figure 10 - Wearable Base Drawing

Electronic

This section covers the electronical aspects of the system. This consists of reviewing the component list of each system unit to provide realisation for the design of the schematic and the PCB layout.

Component List

In this section, the components of the physical system units are displayed. This consists of highlighting the components of the wearable and pill dispenser. The features of server and web-interface units are reviewed in the sections Server Unit and Web Interface respectively. With a worldwide chip shortage, the team has chosen the mentioned components due to their availability and estimation to produce minimal implications integrating into the system.

Wearable Unit

The wearable unit is battery powered. This battery is rechargeable using a micro-USB charger. Hence the micro-UBS can power the wearable as well. The wearable use case is to monitor patient vitals using sensor modules. The sensors collectively monitor the heartbeat, and blood oxygen, and detect abrupt movement for fall detection. Sensor monitoring is the top priority for wearable functionality. The wearable is equipped with an OLED monitor to display notifications from the pill dispenser. All data associated with sensor monitoring is stored on a micro-SD card. Refer to figure Table 2 - Wearable Unit Component List for the wearable component list.

Function	Component	Reasoning
Heart Rate	MH-ET LIVE MAX30102	LED-Reflective solution Small dimensions (5.6mm x 3.3mm x 1.55mm) Programmable Sample Rate Low-power Heart-Rate monitor draws (< 1mW) I2C compatible Programmable with Arduino Framework
Blood Oxygen	MH-ET LIVE MAX30102	LED-Reflective solution Small dimensions (5.6mm x 3.3mm x 1.55mm) Programmable Sample Rate Low-power Heart-Rate monitor draws (< 1mW) I2C compatible Programmable with Arduino Framework
Fall Detection	MPU 6050	3-Axis Gyroscope 3-Axis Accelerometer Embedded algorithms for compass calibration in a library. Supports Interrupts I2C and SPI compatible Operating Current 3.8mA (full power, gyro at all rates, accel at 1kHz sample rate)
User Input	Push Button	Small dimensions Simple implementation
Monitor	I2C OLED Display GM009606	Small dimension (27.8 x 27.3 x 4.3 mm) (l x h x w) Ample screen resolution (128*64 pixels) Low power input (3.3V – 5V) I2C Compatible Programmable with Arduino Framework
Feedback	I2C OLED Display	Small dimension (27.8 x 27.3 x 4.3 mm) (l x h x w) Ample screen resolution (128*64 pixels) Low power input (3.3V – 5V)

		I2C Compatible Programmable with Arduino Framework
Power	PiJuice Zero 1000mAh	Dimension: 8mm x 20mm x 40mm Volts: 3.7 Type: Li-Polymer

Table 2 - Wearable Unit Component List

Pill Dispenser

The pill dispenser is powered by mains electricity. The pill dispenser is equipped with simple circuitry using LEDs and an OLED monitor to provide information to the patient. The feedback of the pill dispenser uses the LEDs and the monitor to display notifications received from the server. Refer to figure Table 3 - Pill Dispenser Component List for the full component list.

Function	Component	Reasoning
User Input	Push Button	Small dimensions Simple implementation
Monitor	Waveshare 7inch Capacitive Touch Screen LCD	1024x600 IPS screen Capacitive touchscreen HDMI connection
Feedback	Google Home	Communication over WIFI or Bluetooth+ Built-in microphone Speech recognising capable Text-To-Speech capable Compatible with multiple languages
Feedback	RGB LEDs	Small dimensions Simple implementation
Power	12.5W Micro USB Power Supply	Output voltage: +5.1V DC Minimum load current: 0A Nominal load current: 3.0A Maximum power: 15.3W 2.5A output 1.5m lead Short circuit, over-current and over-voltage protection

Table 3 - Pill Dispenser Component List

Wearable Schematic:

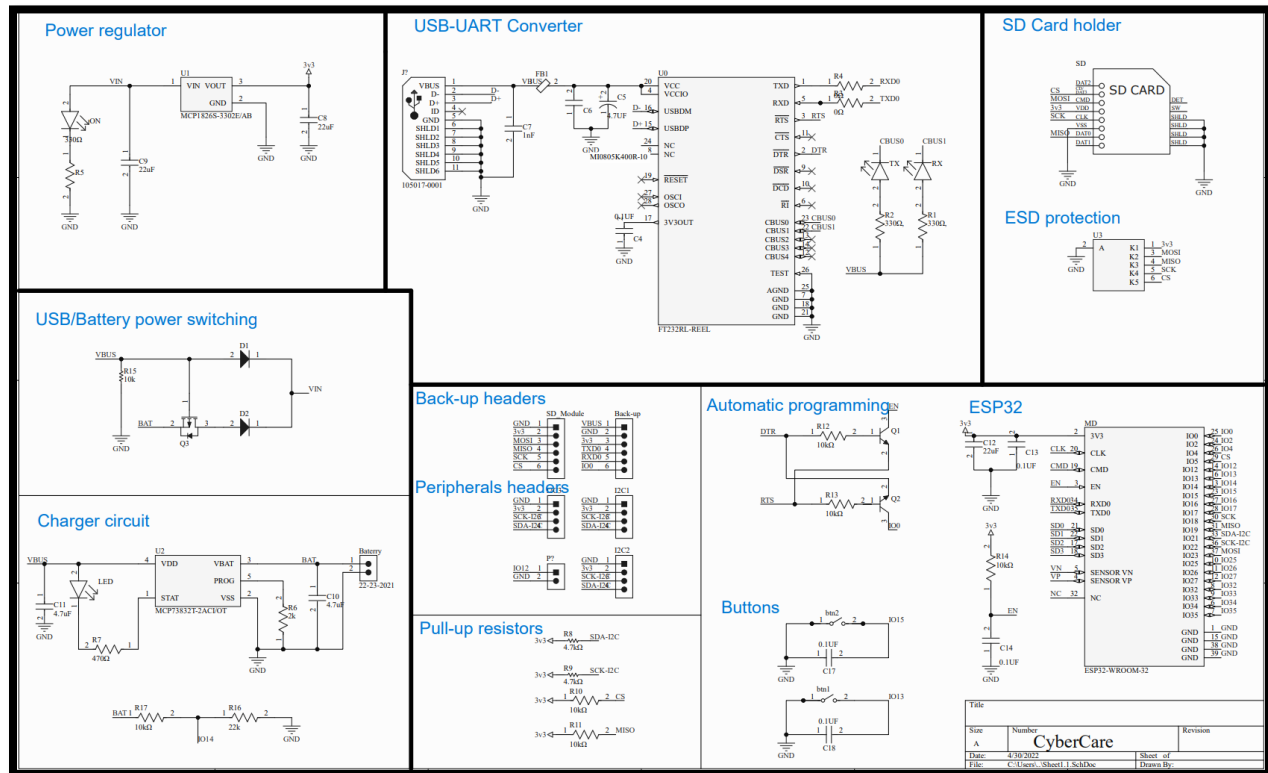


Figure 11 - Wearable Schematic

ESP-32 chip:

This ESP-32 is the brain of this board. It is a microcontroller that has a lot of features. Along with being a low-cost and low-power chip, it has integrated Wi-Fi and dual-mode Bluetooth.

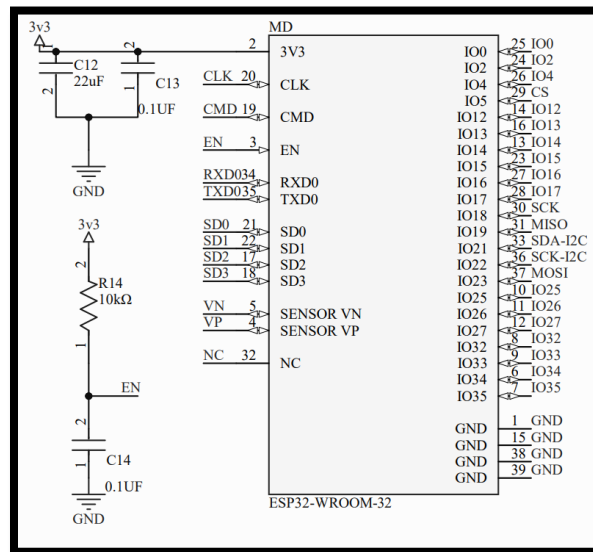


Figure 12 - ESP32 Chip

The purpose of the decoupling capacitor here is to stabilize the input voltage of the chip. If the input voltage drops, then a decoupling capacitor will be able to provide enough power to the chip to keep the voltage stable. If the voltage increases, then a decoupling capacitor will be able to absorb the excess energy trying to flow through to the chip, which again keeps the voltage stable. The values of the capacitors C1 and C2 are 0.1 μF and 22 μF, respectively.

A 10 KΩ pull up resistor is also used to drive the enable pin high. The capacitor connected to the enable pin is also used for filtering reasons and its value is 0.1μF. The values have been selected from the datasheets.

USB to UART Communications:


FT232RL chip is an FTDI chip that works as an intermediate between the USB protocols of the computers and the UART protocols of the microcontrollers. With this chip programming, the ESP-32 is possible.


FT232RL chips specification:

USB Full Speed to Serial UART IC, Includes Oscillator and EEPROM, SSOP-28


USB 2.0 Slave to UART Converter


 **Data Rates:** 3MBaud

 **USB Transfer Modes:** Bulk

 **USB Host:** No

 **Channels:** 1


 **USB Class:** Vendor


 **Operating Temperature:** -40°C to +85°C

 **USB Speed:** Full Speed (12Mbps)

 **Interfaces:** UART with 4 GPIO pins

 **Packages:** 28-pin SSOP

 **I/O Voltage:** 1.8V to 5V

 **Virtual Com Port:** Yes

One of the main reasons for this chip to be selected for this assignment is the integrated oscillator. There are also the hands-free programming features that this chip provides. Refer to the Switches section for more information.

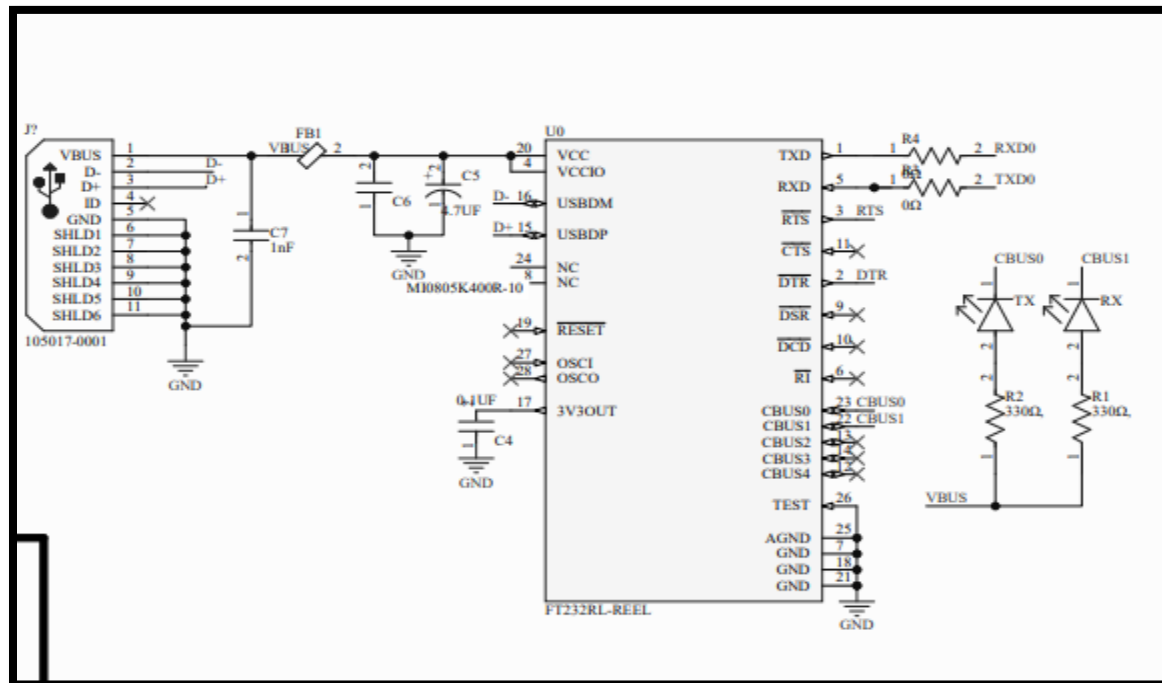


Figure 13 - FT232RL Chip for USB - UART Communication

In the schematic above you can see the main connection from the USB connector to the FT chip. The capacitor is used here also for decoupling purposes. The values for C6, C7 and C14 are 4.7uF, 0.1 uF and

1000pF, respectively. Two $0\ \Omega$ resistors are used for the transmitting and receiving lines. There are also 2 LEDs TX and RX to indicate if there is data being transferred.

Voltage regulator:

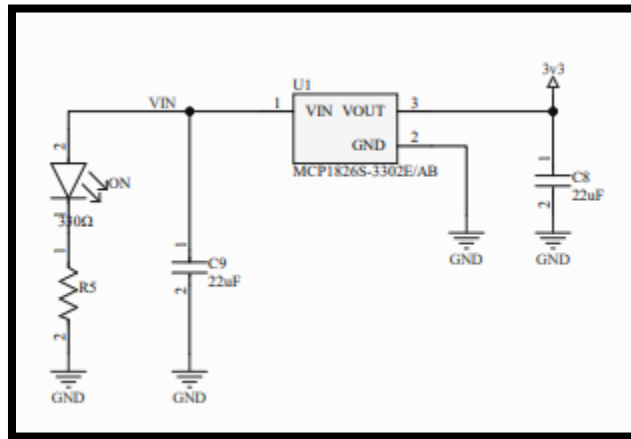


Figure 14 - Voltage Regulator Schematic

The MCP1826 is a Low Dropout (LDO) linear regulator that provides high-current and low-output voltages. The device can provide fixed output voltages of 3.3V with only two decoupling capacitors at both the input and the output. Almost all the peripherals including the ESP-32 chip can be powered with 3.3V.

The MCP1826 can provide the board with a maximum current of 1000 mA which is more than enough for the ESP-32 and its peripheral. The ESP32 in the active mode (Wi-Fi is on and using some peripheral devices) needs around 240mA.

Battery management ICs:

Charger IC:

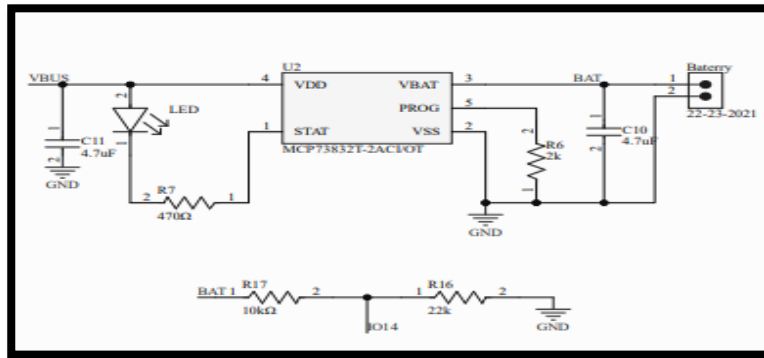


Figure 15 - Battery Management Schematic

To safely charge the battery a charger circuit along with a protections circuit must be included especially for the Li-Ion or Li-Polymer type of batteries.

The mcp73832 is a charge management controller, that employs a constant-current/constant-voltage charge algorithm. One of the advantages of this chip is the low number of external components required.

To determine the constant charging current a 2k ohm resistor is connected to pin 5 (PROG). A voltage divider is applied with two resistors to detect the battery charge level.

Protection IC:

The protection IC is included in the battery. This protection IC will switch off the voltage during the charging process when reaching a specified threshold. It will also turn off the discharge of the battery pack when it is discharging too quickly or if the discharge levels become too low.

Switching logic:

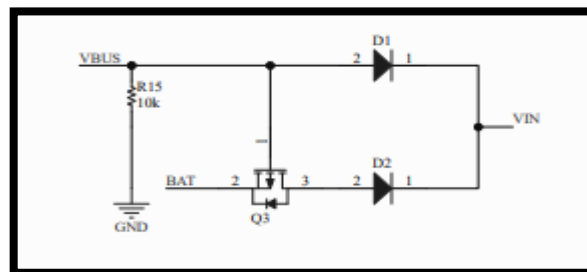


Figure 16 - Protection IC Schematic

The two Schottky diodes along with the PFET create a switching circuit that will allow the wearable to work even though the battery is charging.

When the USB cable is plugged in to charge the battery, it will provide the VBUS line with 5 volts. This will close the channel between the terminals of the PFET thus disconnecting the battery. The VBUS line is connected through a diode to the power input pin of the power regulator to provide power while the battery is charging.

If the USB cable is not connected, the VBUS line will be pulled down by the resistor and the channel of the PFET will be opened allowing the current to flow from the battery to feed the power regulator.

Switches:

Transistor switches:

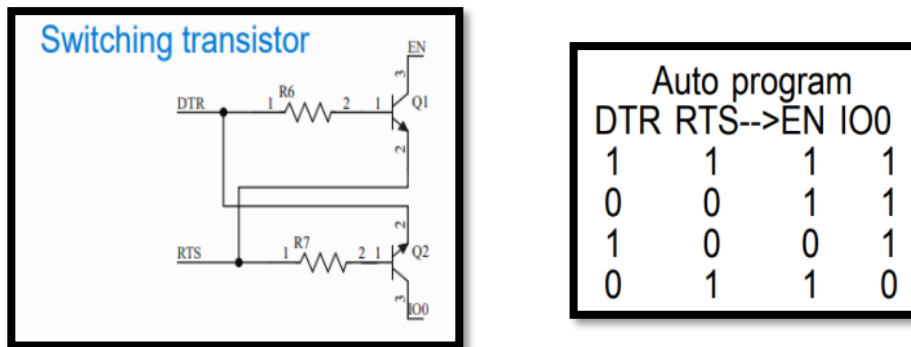


Figure 17 - Switching Transistors Schematic

The two transistors here provide the board with a Hands-Free programming feature. In the truth table, you can see the possibilities that These two transistors can produce. The most important part is EN transitions from 0 to 1 whilst IO0 is low, resetting into the bootloader.

Physical switches:

There are two buttons connected to the I/O pins to interact and control the wearable.

Pull-up resistors:

Five Main pull resistors have been added to the board. Two of them are for I2C protocols for SDA and SCK lines. The I2c will not work without connecting those resistors. Another one for the MISO line in the SPI communication protocols is because this line is only driven by a slave when its slave select line is asserted. There is also one for the Enable pin to enable the ESP-32.

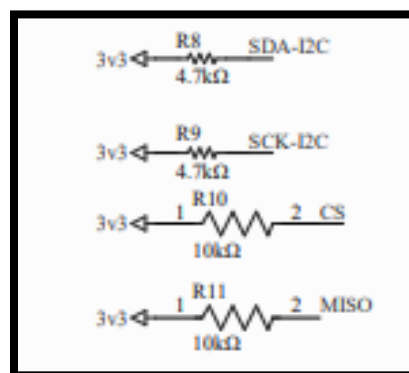


Figure 18 - Pull Up Resistors Schematic

Headers:

Headers are also included in this design to connect additional modules to the ESP32. There is also a backup header that includes the power and the serial communications pins. Manual programming can be done using this backup header.

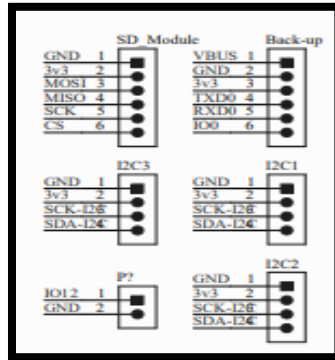


Figure 19 - Headers Schematic

SD Card:

The schematic highlights the SD cardholder as well. The SD Card will store the data from the sensors to be uploaded to the server every fixed amount of time.

ESD protection is considered of helping to guard the SD card against the potentially damaging effects of electrostatic discharge.

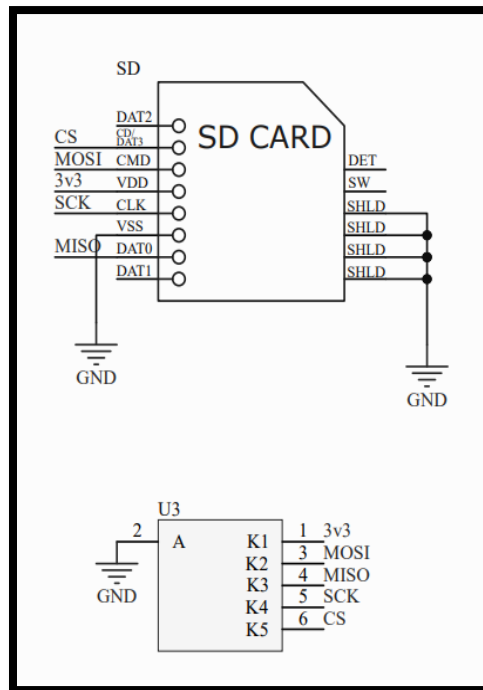


Figure 20 - SD Card Schematic

Wearable 3D Model

TOP:

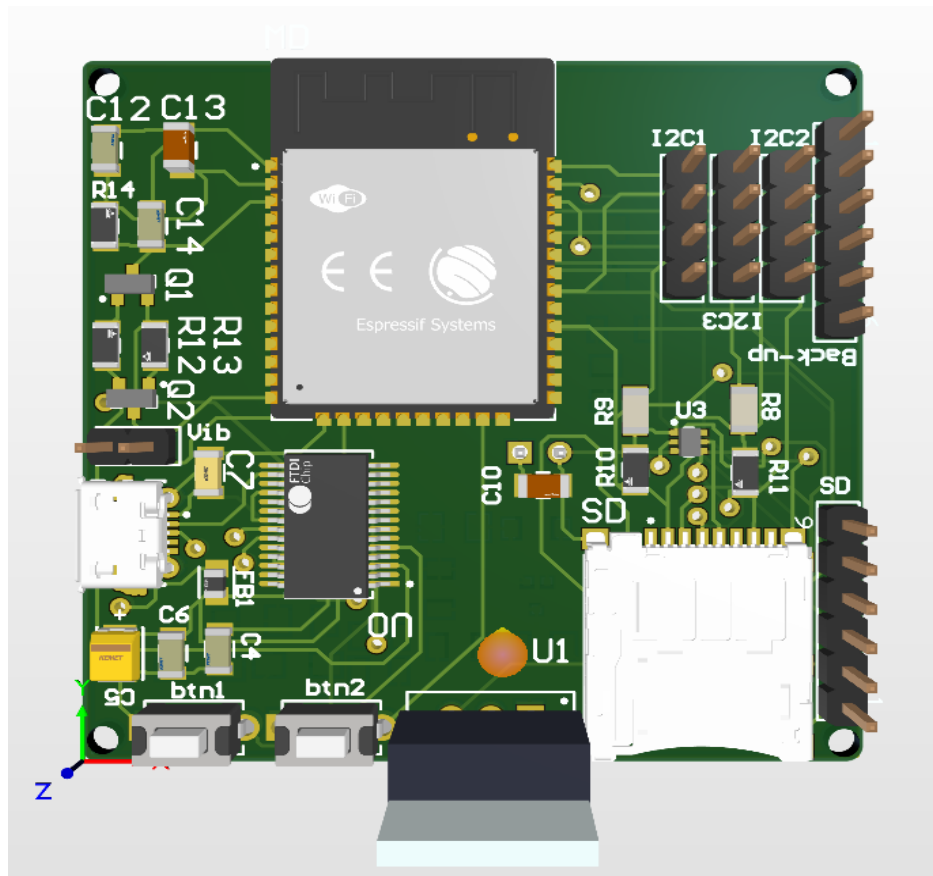


Figure 21 - Wearable 3D Design Top View

Bottom:

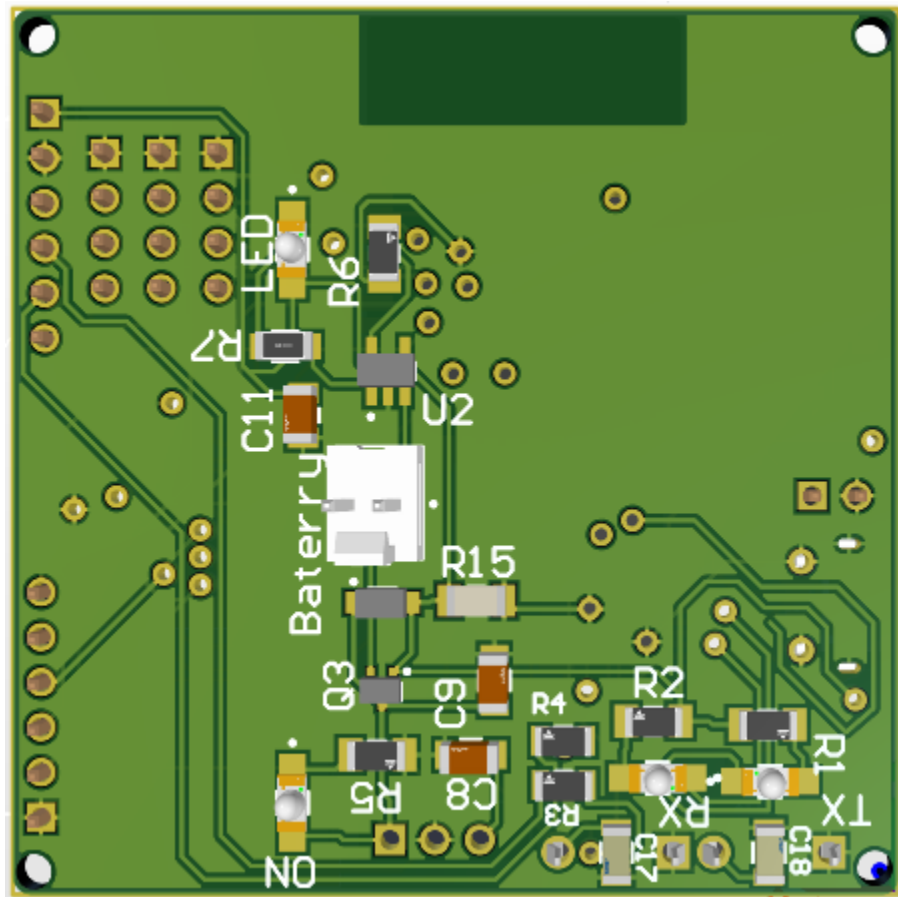
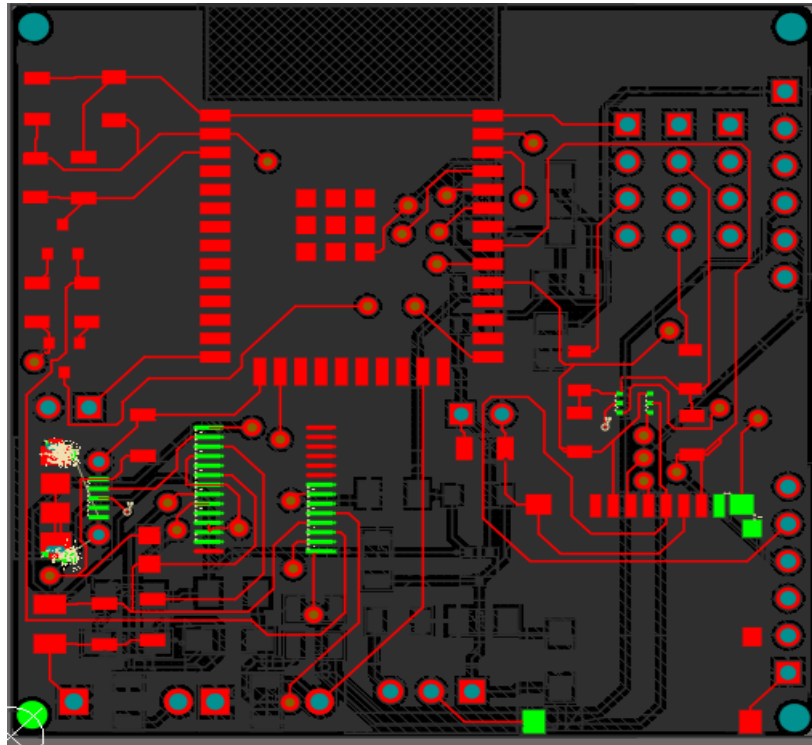
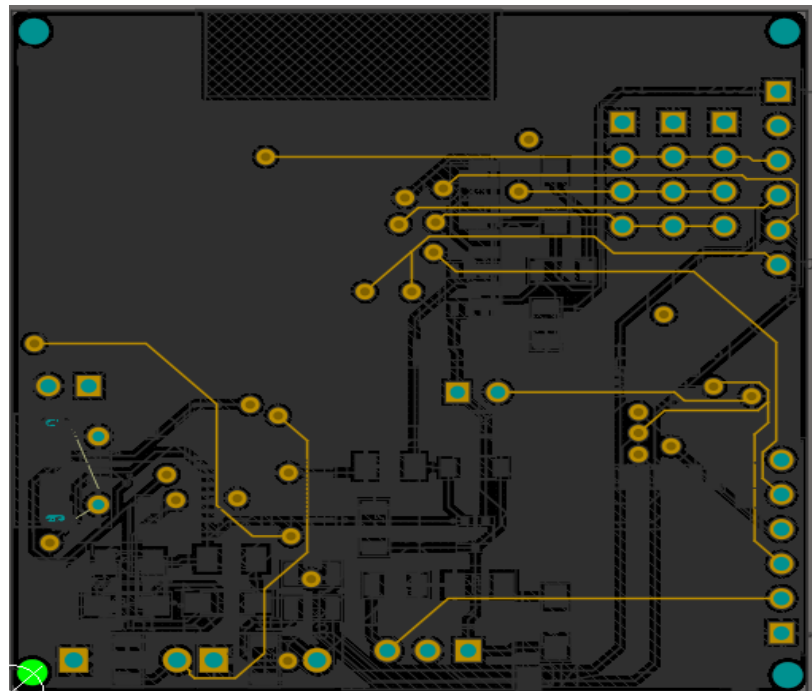


Figure 22 - Wearable 3D Design Bottom View

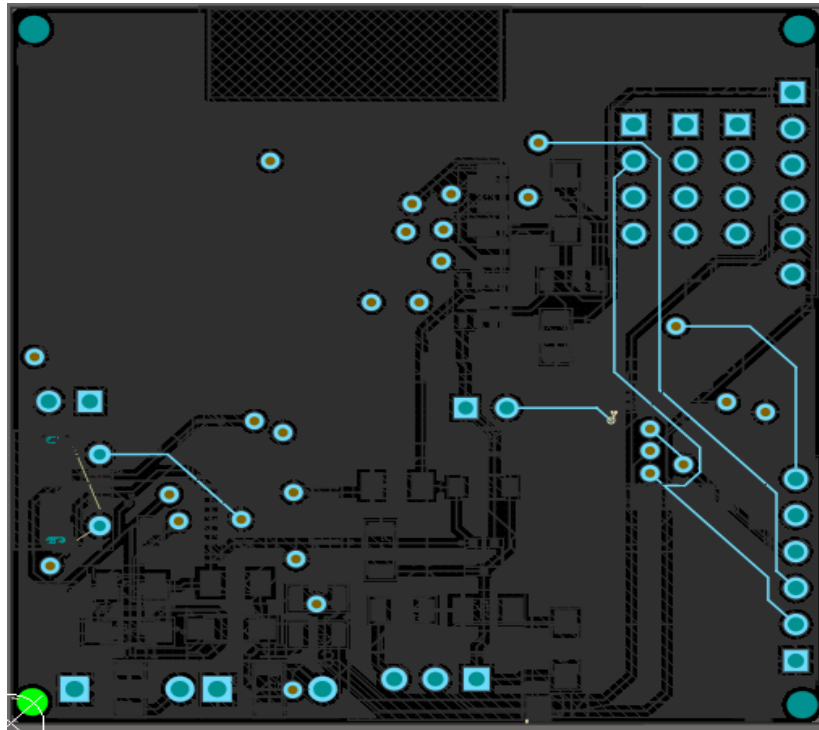
Wearable PCB Layout:
Top Layer:



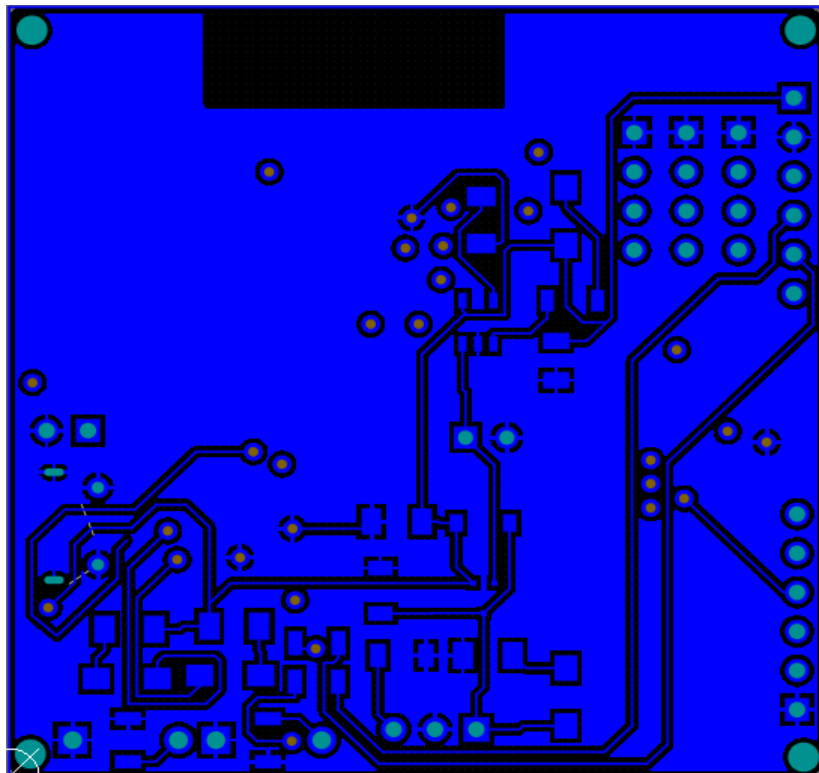
Layer 1:



Layer 2:

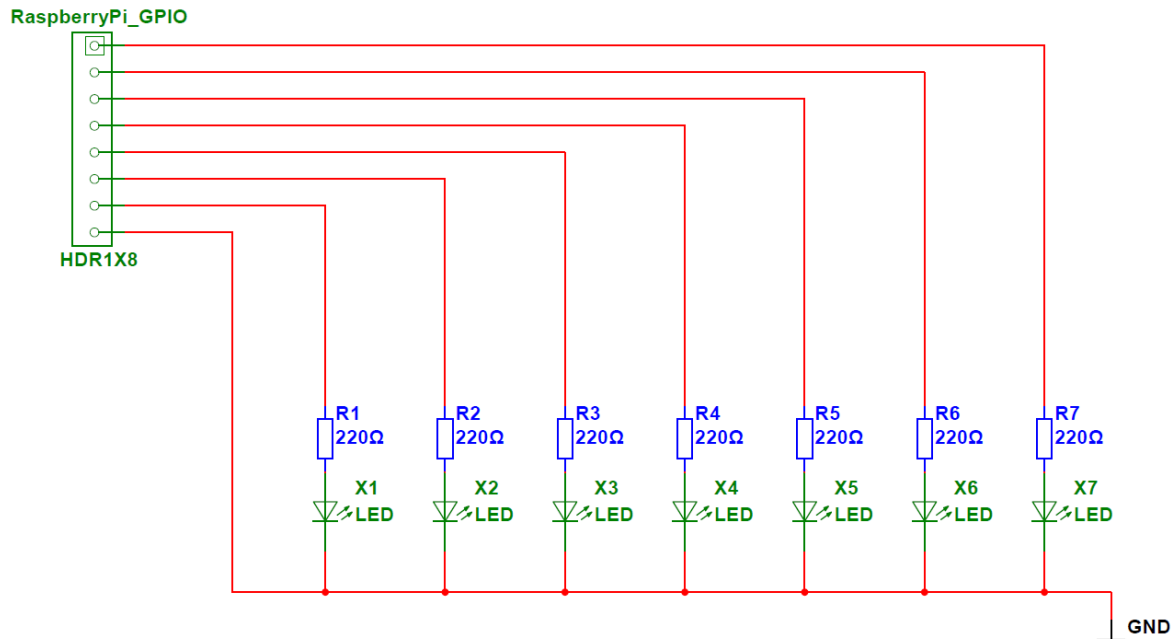


Bottom Layer:



Pill Dispenser Schematic

The pill dispenser unit significantly relies on the processing power of the RaspberryPi. As mentioned in the functional requirements of the pill dispenser, the patient is notified of the specific pills to take by the guidance of lighting LEDs in the appropriate pill segments. With this requirement, the pill dispenser uses a custom-built PCB for the lighting of the LEDs.



Title: CyberCare: Pill Dispenser Schematic		
Schematic of patient pill dispenser		
Designed by: CyberCare	Document N:	Revision: 1.0
Checked by:	Date: 27/04/2022	Size: Custom
Approved by:	Sheet 1 of 1	

Figure 23 - Pill Dispenser LEDs Schematic

The schematic highlights a simple design. GPIO pins from the RaspberryPi are used to provide an appropriate signal to toggle the LEDs. The LEDs are provided resistors to prevent LED blowing.

Pill Dispenser 3D Model

Top:

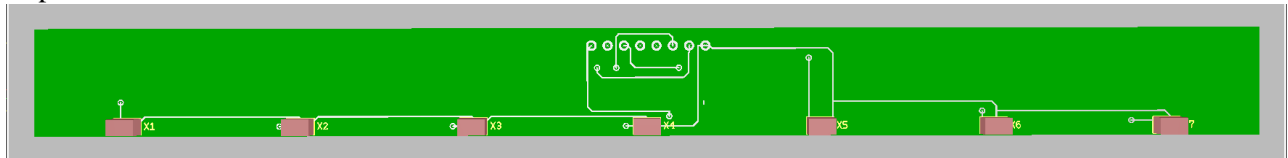


Figure 24 - 3D Design Pill Dispenser LEDs Top View

Bottom:

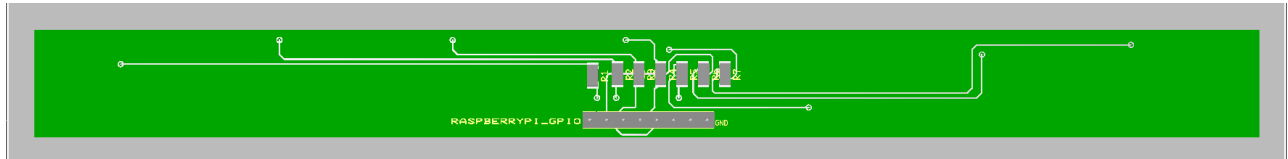


Figure 25 - 3D Design Pill Dispenser LEDs Bottom View

Isometric:

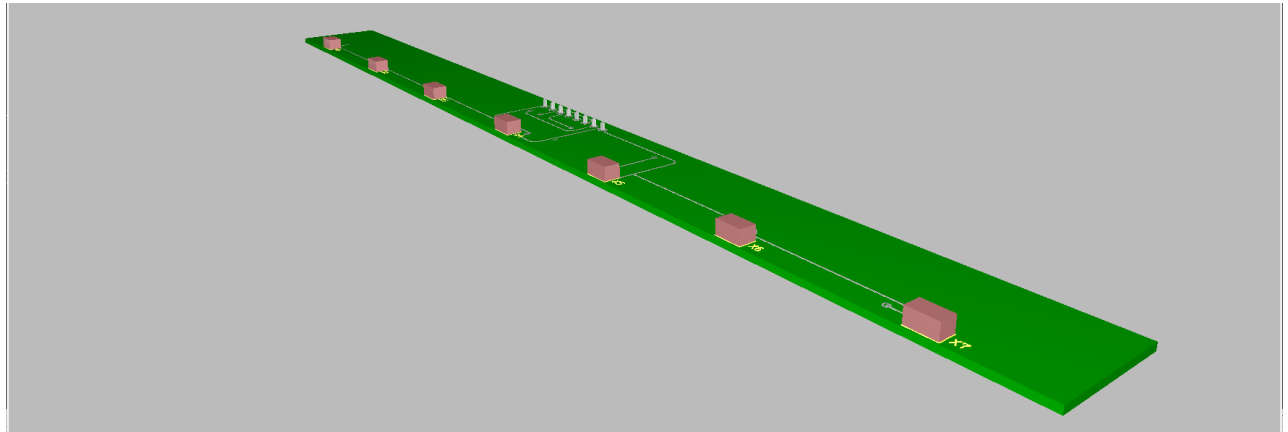
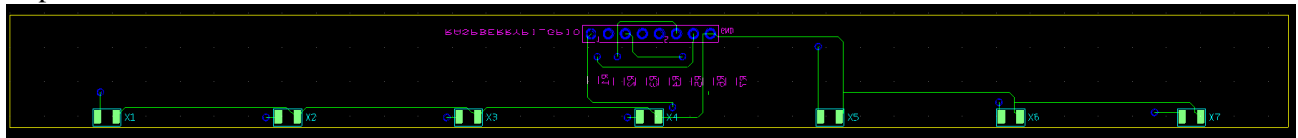


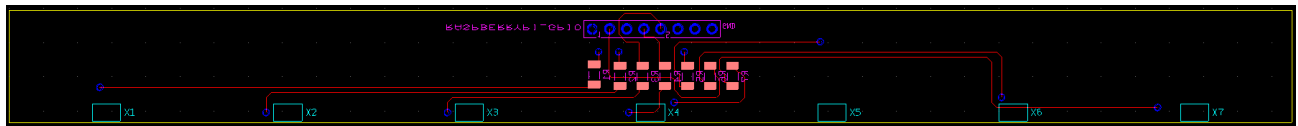
Figure 26 - 3D Design Pill Dispenser LEDs Isometric View

Pill Dispenser PCB Layout

Top:



Bottom:



Software

In this section, the software aspects of the system units are reviewed. Principal processes such as sending, receiving, reminding, and updating are mentioned in the coming sub-sections. This section evaluates the mentioned processes from the wearable, pill dispenser, server, and web-interface perspective, respectively. Flowcharts and diagrams are used to visualise the processes.

Wearable Unit

The function of the wearable is to monitor sensor readings, send sensor data, receive incoming notifications from the pill dispenser, and make appropriate changes if required. The state of the wearable is assigned to one of 4 states. The states are ready, idle, busy, and unavailable. These 4 states define the priorities that the wearable abides by. Refer to the Wearable State Diagram and Wearable UML Diagram sections.

Wearable State Diagram

The 4 states define the behaviour of the system at a point in time. The system start-up process starts with a signal and the system is set to the idle state.



Figure 27 - Wearable State Diagram

In the idle state, the system constantly listens for alerts, commands, or messages sent from the pill dispenser. If an input is absent from the pill dispenser, the system continuously displays sensor data to the display. In the case that an alert, command, or message is present the system state is set to ready. The system is prone to communication loss with the pill dispenser unit. In this case, the system state is set to unavailable, and the system will attempt to reconnect with the pill dispenser unit by performing a system reset.

In the ready state, the system attempts to recognise the input. If the system recognises the input, the system will attempt to act. The system state is set to unavailable when the input is not recognised. When the system acts, the system state is set to busy.

In the busy state, the system performs appropriate action-specific verification methods to ensure the action is successful. In the positive case that the action is managed by the system, the system state is set back to the ready state. If the system is unable to oversee the action, the system state is set to unavailable.

In the unavailable state, the system can retry to perform a certain action to prevent unnecessary system resets or perform a system reset due to the connection loss with the pill dispenser unit.

Wearable UML Diagram

The team has produced the UML diagram within the designing phase of the system. The team understands that the structure is bound to change in the implementation stage of the system development. However, the UML diagram highlights the intended principal aspects of the wearable unit. The wearable is controlled with a single class that is equipped with different object handlers. The object handlers consist of device wearables, system flow objects, and external device dependencies. With this structure, the intention is to be able to control the various devices in a single class providing a modular and straightforward approach.

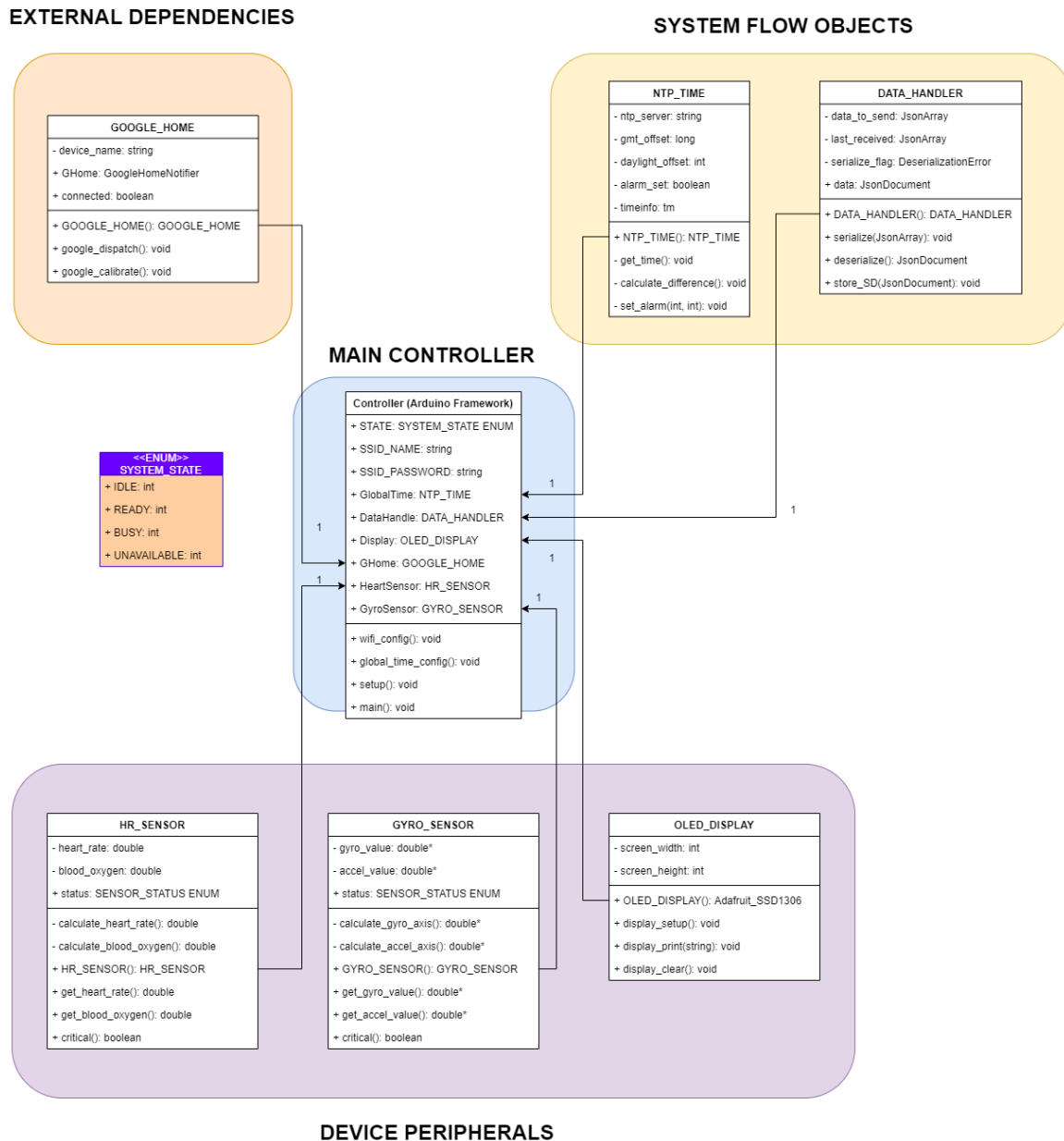


Figure 28 - Wearable UML Class Diagram

About the UML diagram, the external libraries that the unit is dependent upon can be recognised in the table below.

CLASS	LIBRARIES	REASONING
OLED_DISPLAY	Adafruit_SSD1306	Existing interface functions
DATA_HANDLER	ArduinoJSON	JSON capabilities for Arduino framework compatible boards
GOOGLE_HOME	GoogleHomeNotifier	Existing API to communicate with Google Home Compatible with an ESP32 board

Table 4 - Software Library Dependencies

For the wearable to function smoothly, system flow objects are created to provide guidance and information to and from the system. In this category, there are two classes. The classes are NTP_TIMER and DATA_HANDLER. The NTP_TIMER provides the system with real-world time information. The NTP_TIMER aids the system to perform time-specific actions such as providing reminders to the patient or providing timestamps for data logging. The purpose of the DATA_HANDLER is to manage all types of data that is to be received or sent by the wearable. The tasks assigned to the DATA_HANDLER consist of handling data that is received from the other system units and storing data on the installed SD card.

As mentioned in the wearable component list, the sensors' heart rate, gyroscope, and OLED display are installed onto the design. Each device module is assigned a class. Sensor configurations and other appropriate methods and functions explicit to the device are defined and implemented within these classes. In the controller class, each device is assigned a single instance of each device's peripheral class. The HR_SENSOR class defines and implements the behaviour of the heart-rate sensor. Similarly, the GYRO_SENSOR defines and implements the behaviour of the gyroscope. The sole purpose of the OLED_DISPLAY class is to monitor and control the intended information to display to the patient.

The wearable depends on the external google home device to provide voice input from the patient and to broadcast received information from sensor data and other system units. The GOOGLE_HOME class provides the wearable with an interface to communicate with the google home device.

Sending Sensor Readings

Each sensor is assigned a timer value. This timer value is used to define the frequency at which the system attempts to receive data from a specific sensor. When it is time to retrieve data from a specific sensor, the system refers to the device class instance in the controller. The device class instance obtains the sensor value and checks for abnormalities. In the positive case that abnormal values are absent, the data is returned to the controller class where it can be sent to the DATA_HANDLER class to format the data into JSON values and send it to other system units. In the opposite case where abnormal values are present, the abnormal or extreme data flag is set. With the abnormal or extreme flag, the server recognises the abnormality and can provide an appropriate action to counter.

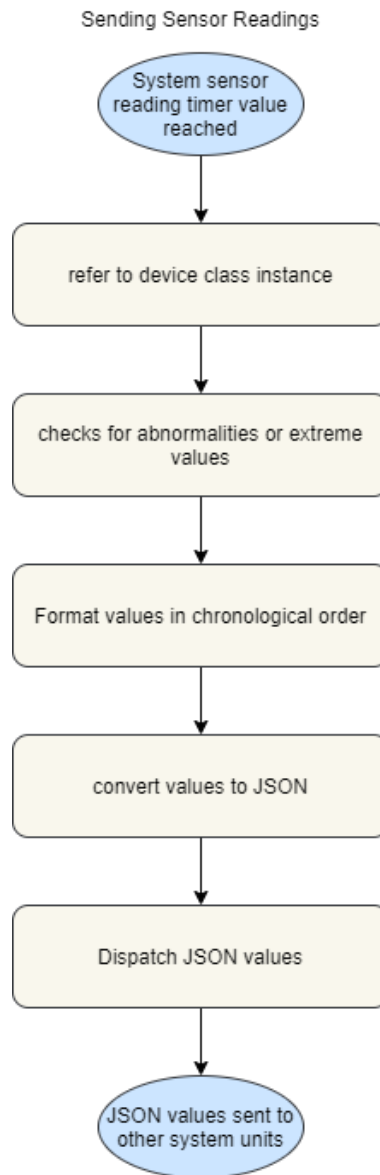


Figure 29 - Sending Sensor Readings Flowchart

Receiving Message/Alert from Pill Dispenser

In principle, one of the three data types is received and managed at a time. The DATA_HANDLER class recognises the input data types of alerts, messages, and commands. The system attempts to perform an appropriate action to the input data. For alerts and messages, the information is displayed in a flashing manner on the monitor using the OLED_DISPLAY device peripheral class. Commands are not displayed on the monitor. The speed at which the display monitor flashes differ on the significance of the input data. In the case that the input data is an alert, the monitor flashes twice per second. This is to persuade the patient to notice the monitor display informing that an alert is present. For the latter case, the information is simply displayed with the monitor flashing occurring every second. The system waits for an acknowledgement by waiting for the input signal from the push-button mechanism. The system will stop displaying the message once an acknowledgement signal is received from the push-button mechanism.

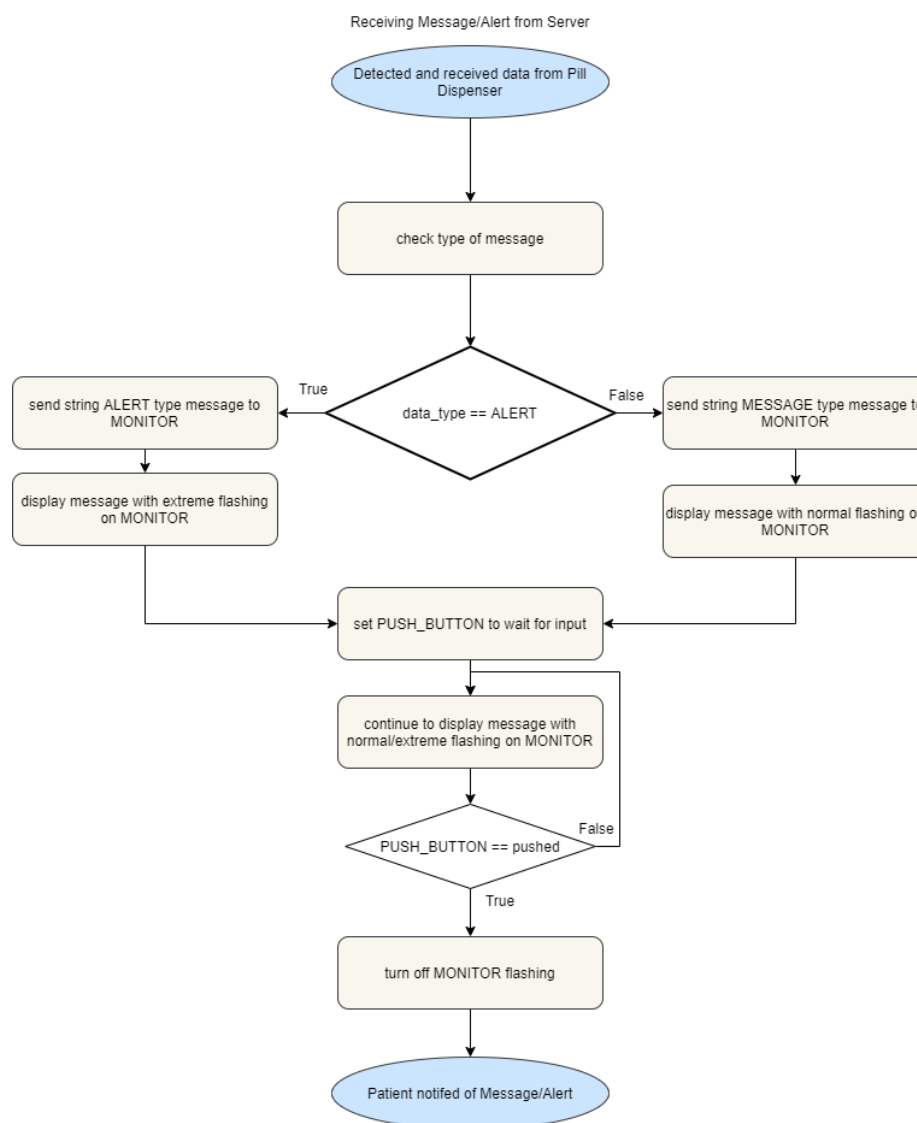


Figure 30 - Receiving Message or Alert from Server

Pill Dispenser Unit

The purpose of the pill dispenser is to guide the scheduling of the patient's pill intake. Furthermore, the pill dispenser can receive notifications or commands from the server. Commands from the server can alter the scheduling of patient pill intake or modify the pill dispenser system settings. Like the Wearable Unit, the pill dispenser has 4 states. The states are ready, idle, busy, and unavailable. The priorities of the pill dispenser differ from the wearable. Refer to the sections Pill Dispenser State Diagram and Pill Dispenser UML Diagram.

Pill Dispenser State Diagram

Although like the wearable state diagram, the pill dispenser alternates between three instead of four states. The focus is on whether the connection to the server is present. Regardless of the internet connection, the patient needs to be able to receive reminders to take the pills so when the server is not connected the pill dispenser relies on the schedule stored internally. Whenever the connection to the server is restored, the internal schedule is synchronised with the one on the server to ensure any changes made by the medical professional or family members are considered. Whenever a notification or command is sent or triggered on the pill dispenser the state shifts to waiting for feedback to ensure the patient has registered the notification. Any fault in the system is dealt with by shifting to the waiting state until the system reboots to deal with the issue.

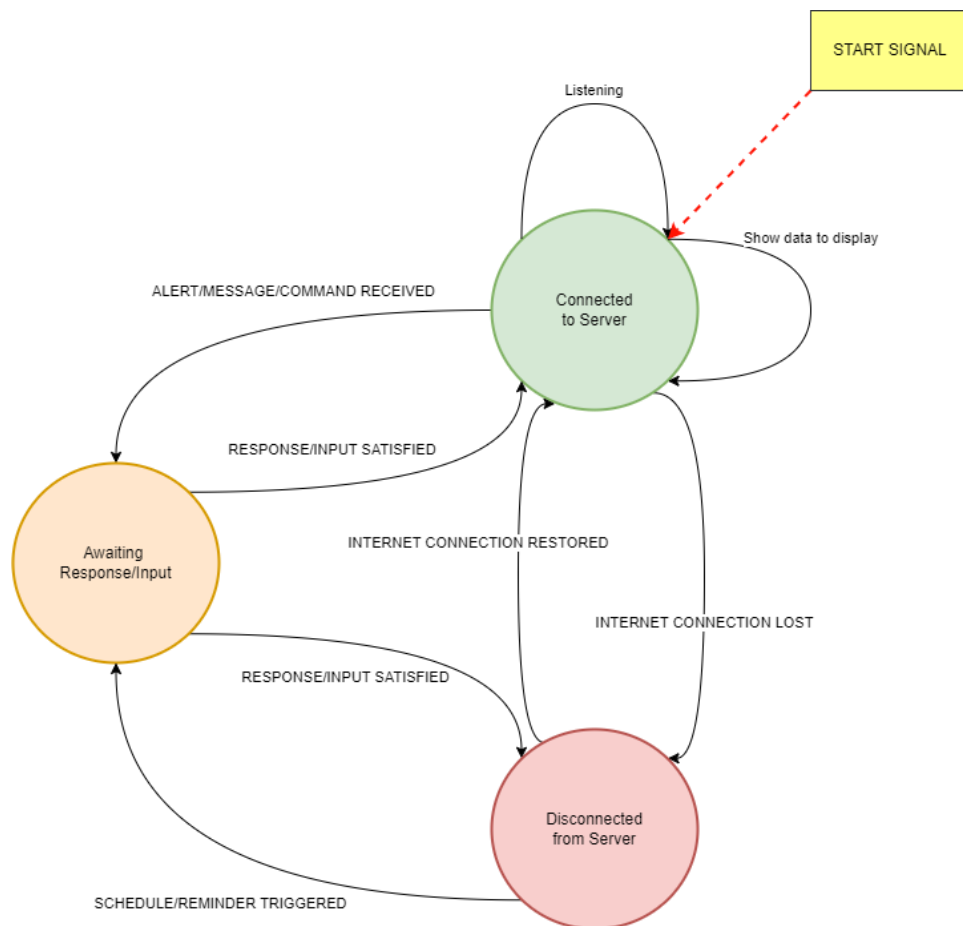


Figure 31 - Pill Dispenser State Diagram

Pill Dispenser UML Diagram

The UML diagram for the Pill Dispenser follows the same structure as the wearable's UML diagram. Just like before, the system flow objects ensure that the JSON data gets interpreted properly and that time calculations are made easier. The main differences found here, are that the main controller is built on a Linux framework instead of the wearable's Arduino and that the device holds a schedule type. The schedule is currently defined loosely with a simple pointer holding an array of reminders/alerts.

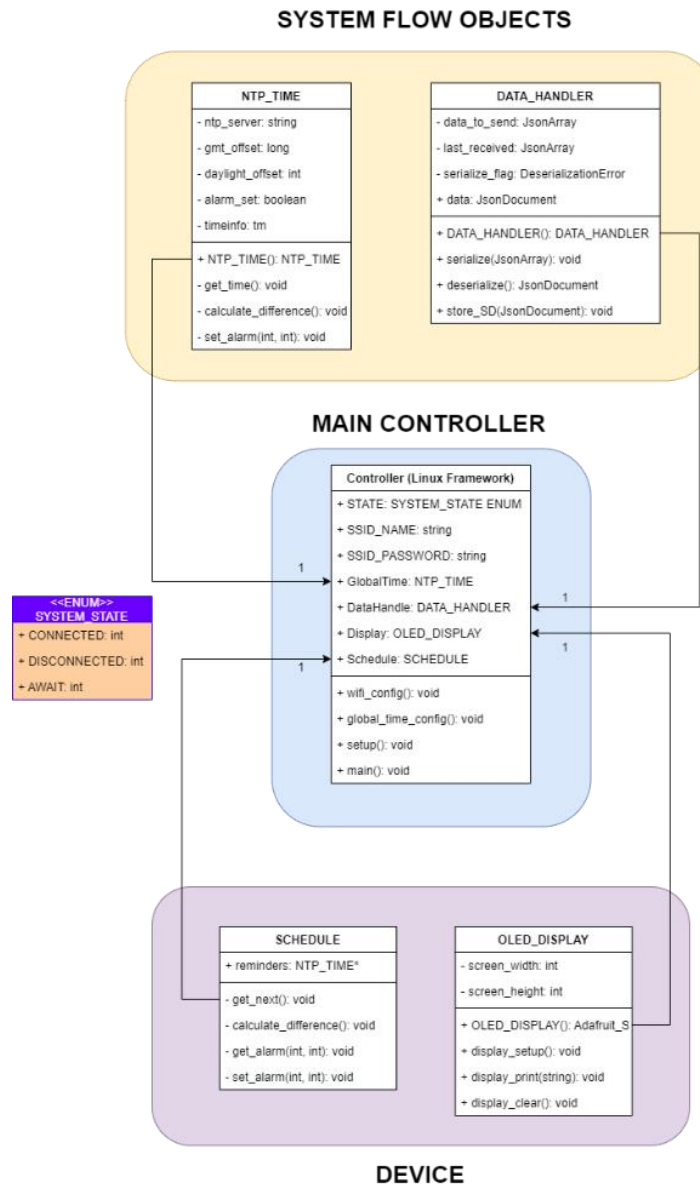


Figure 32 - Pill Dispenser UML Class Diagram

Alert Scheduled Pill Taking

To make it as easy as possible for the patient to know exactly which grouping of pills needs to be taken when an alert gets triggered, the pill dispenser contains LEDs. The process for turning on the correct LED for a scheduled event is a simple while loop. When the system receives the alert, the required sensor is passed with it. The sensor is stored and compared to the known sensors of the system until a match is found. Finally, upon finding the match the LED is triggered and the patient can be sure that he/she is taking the correct pill.

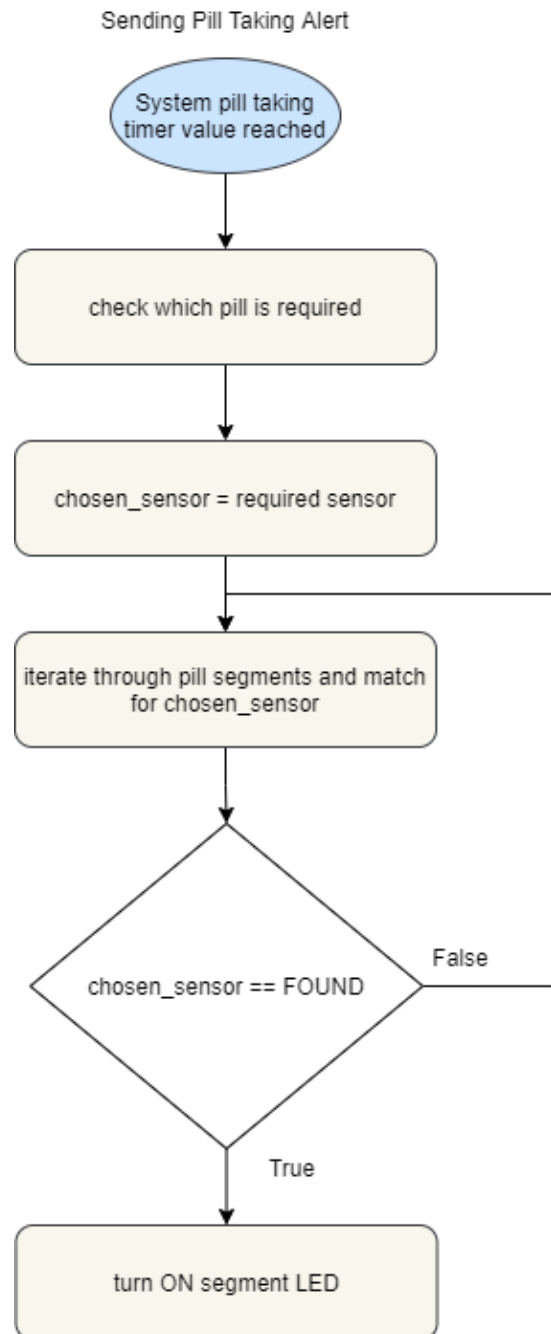


Figure 33 – Alert Pill Taking Flowchart

Receiving Message/Alert/Command from Server

When communicating with the server, the pill dispenser can receive three types of data. In the case the received data is an alert, the pill dispenser can display it on the built-in OLED screen as well as forward it to the Google Home to notify the patient verbally. If the data received is a message, the response is very similar to an alert except since the message does not have as high of precedence, the Google Home is not notified. The message is simply displayed on the OLED screen. Lastly, the data received could be a command, in this case, the system must first check if the command is meant to alter the system settings or if it is meant to alter the patient's schedule. In either event, the modification is made, and the patient is alerted with a message on the screen. With every message or alert, the system waits for a button press response from the patient until removing the message or alteration from the screen to ensure the patient has been notified.

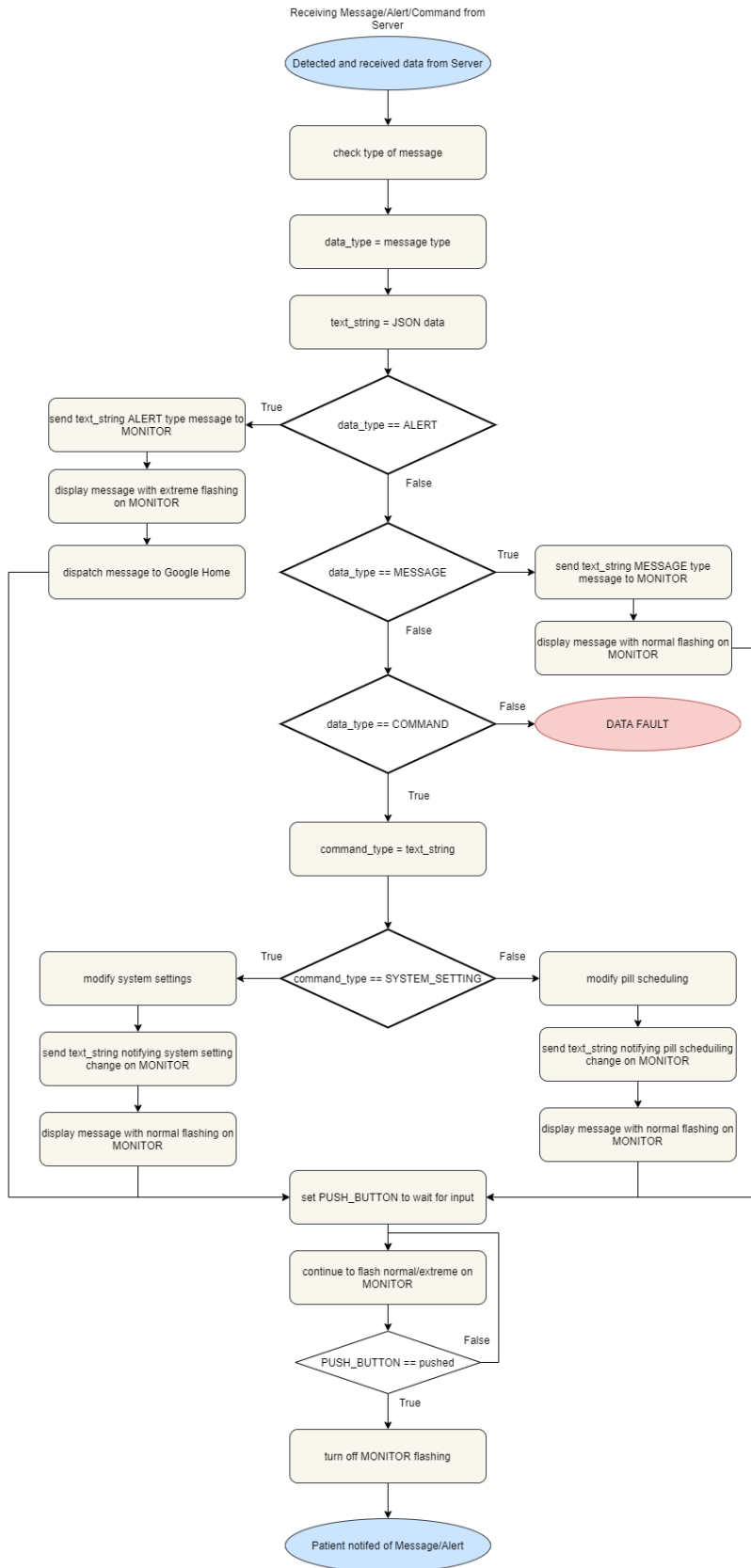


Figure 34 - Receiving Message/Command/Alert from Server

Server Unit

On the server side, there is a database and an API handler. This part of the document contains a UML diagram with the database schematics and a first draft of the API documentation. This will help us understand what and how we need to develop the server in the next stage of the project.

Database

The database will contain all the data that needs to be stored. This could be data that is collected with sensors in the watch. But also, information for the pill dispenser and user data should be stored in a database. We created a UML diagram to visualize what data should be stored in what table.

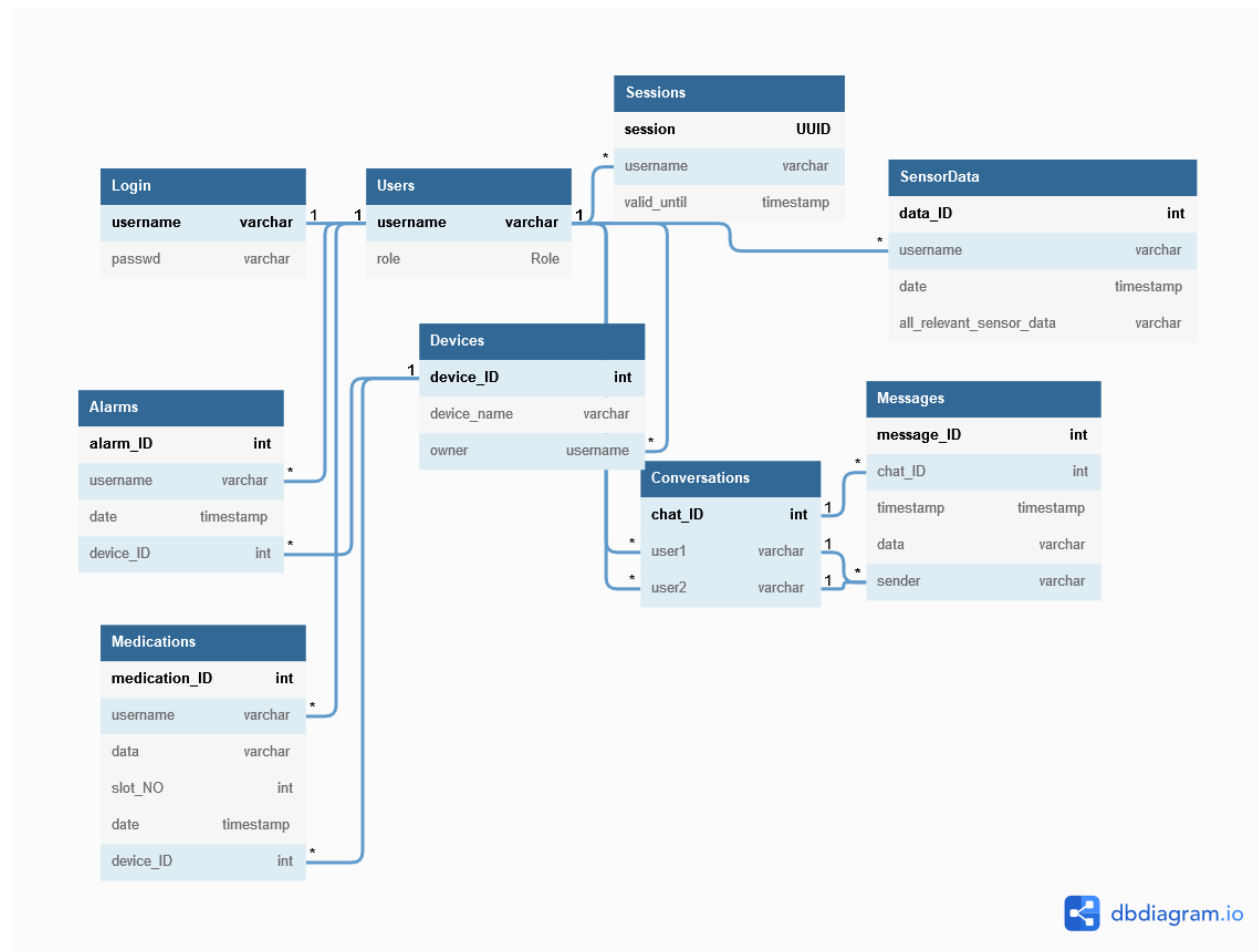


Figure 35 - Database UML Class Diagram

The figure is the first draft of our database diagram. It is subject to change over time, but it gives a rough indication of how and what. Central in this database is the user functionality. In the Users table, we store the username and the corresponding role. This role could be the patient, a family member, a medical professional, or an administrator. All roles have different rights. For example, only the medical professional is allowed to change medications, but the patient might still access the table to review his current medications.

All tables are connected to a user. The data in these tables are connected by referencing the username.

The database software that we intend to use has not been chosen yet. Options are PostgreSQL or MySQL. The language of this different database software is the same at its core. However, they are different in efficiency and way to store the actual data.

API documentation

This document explains what APIs are available to use. It will explain the instructions of the API call and what the expected output will be.

The API methods are used by calling `/API/ [function name]`. All API methods with exception of the login methods need to authenticate with a session token. This is a cookie that is generated by the server after login. This session token needs to be included in the HTTP header.

The API method always returns a JSON with the attribute `request`. The value will either be `OK` or `NOK`. In the case of `NOK`, something went wrong. That means either a wrong parameter was entered, or an invalid session token was included in the HTTP header.

Unless otherwise stated, the API method uses the HTTP-POST requests. This means that any parameters should be added to the form-data field of the request.

login

`login` is an HTTP-POST request. It is called to authenticate a user. If the authentication is complete and correct, a cookie is distributed. This cookie can be used for authentication during a session. This will prevent the user from sending his password multiple times.

Parameters

for this HTTP request, two different parameters are passed. They are passed as form data in the body of the HTTP request. If either one of those parameters is not in the form of data, the request will fail. If the parameters are available, the request will not fail. However, only if the username and password match the database, a cookie is created and set.

`Username` is the username of the user.

`Password` is the password of the user.

logout

logout This request will invalidate the session cookie. After the user will be redirected to the home page. The specific details of this request will be worked out later. [TODO]

getConversations

This method is called with an HTTP-POST request. It returns all conversation_IDs of the conversations that the current user is in. This request may only be called with a valid session ID in the HTTP header.

parameters

recipient: Only conversation_IDs with conversations between logged in user and recipient are shown. This parameter is [OPTIONAL]

examples

/API/getConversations returns all conversations that the current logged in user takes place.

In this example, the session ID in the HTTP header is linked back to Bob. All chat_IDs where Bob took place are returned:

```
{
  "request": "OK",
  "conversation_ID": [
    {
      "id": 0,
      "user1": "Alice",
      "user2": "Bob"
    },
    {
      "id": 2,
      "user1": "Bob",
      "user2": "Charlie"
    },
    {
      "id": 5,
      "user1": "Bob",
      "user2": "Danny"
    }
  ]
}
```

The next example has a valid session ID in the HTTP header. However, this user did not take part in any conversations. Therefore, chat_id has an empty list:

```
{
  "request": "OK",
  "conversation_ID": []
}
```

recipient=Charlie returns all conversation_IDs with currently logged in user Bob and the recipient Charlie:

```
{
  "request": "OK",
  "conversation_ID": [
```

```
{
  "id": 2,
  "user1": "Bob",
  "user2": "Charlie"
}
]
```

newConversation

This method is called with an HTTP-POST request. It will create a new conversation with a user known by the system. On success, a JSON will be returned with the newly generated conversation_ID.

parameters

recipient: This specifies the user_ID with whom a new conversation should be started.

examples

recipient=Charlie. A new conversation between the logged in user and Charlie will be created.

On success, a JSON will be returned containing "request": "OK" and "conversation_ID": [int]

```
{
  "request": "OK",
  "conversation_ID": 8
}
```

getMessages

This method is called with an HTTP-POST request. It returns all messages with a specific `conversation_ID`. This parameter should be added in the form-data field of the request and it's mandatory.

The session ID of the logged-in user should be added to the HTTP header for authentication purposes.

parameters

`conversation_ID`: Mandatory parameter that is used to get all messages with that specific `conversation_ID`

examples

`conversation_ID=2` returns all messages with conversation ID number 2. In this example that would be all messages between Bob and Charlie. The

In this example, the session ID in the HTTP header is linked back to Bob. All chat_IDs where Bob took place are returned:

```
{
  "request": "OK",
  "messages": [
    {
      "id": 4,
      "timestamp": 1650294015,
      "data": "Hey Charlie!",
      "sender": "Bob"
    },
    {
      "id": 9,
      "timestamp": 1650294915,
      "data": "Hey Bob!",
      "sender": "Charlie"
    },
    {
      "id": 10,
      "timestamp": 1650294925,
      "data": "Are you ok?",
      "sender": "Charlie"
    },
    {
      "id": 13,
      "timestamp": 1650295015,
      "data": "Yeah everything is ok!!",
      "sender": "Bob"
    }
  ]
}
```

sendMessage

/api/sendMessage is an HTTP-POST request. The request is used to send a message in a specific conversation. Authentication by session_ID is mandatory.

parameters

Message contains the information of the message sent. The assumption is made that only text can be sent. Chat_ID is the ID of the conversation in which the message should be sent.

examples

A valid post-request returns a JSON with the following keys. The first key returns that the request was executed without errors. The message_ID that got assigned to the message will also be returned.

```
{
  "request": "OK",
  "message_ID": 19
}
```

getAlarms

getAlarms is an HTTP-POST request. The request is used to get all alarms set for the current user. A user with elevated access is also able to get the alarms of other users. This may be the medical professional for example.

setAlarm

This request is used to set a specific alarm. The alarm can be set for different devices. Users with elevated access can set alarms for users other than themselves.

getMedications

getMedications is an HTTP-POST request. The request is used to get medications set for the current user. A user with elevated access is also able to get the alarms of other users. This may be the medical professional for example.

setMedication

This request is used to set a medication rule. Only a medical professional would be able to use this function.

getSensorData

This request is used to get sensor data. Different parameters can be passed to edit what kind of data should be returned. This request will be worked out at a later stage.

Web Interface

The team deems it important to not only be able to interact with the system through the patient's hardware such as the google home and wearable. To streamline the edits made to the patient's schedule, access patient data logs, and contact the patient as a family member or medical professional a web portal will be set up as a user interface. Initially, the user is met with a login/sign up screen to ensure that the user access only concerning to them and can only make changes within their right. The user accounts will make it easy to limit the application features when necessary while also adding a layer of security easing especially the patient's mind on security.

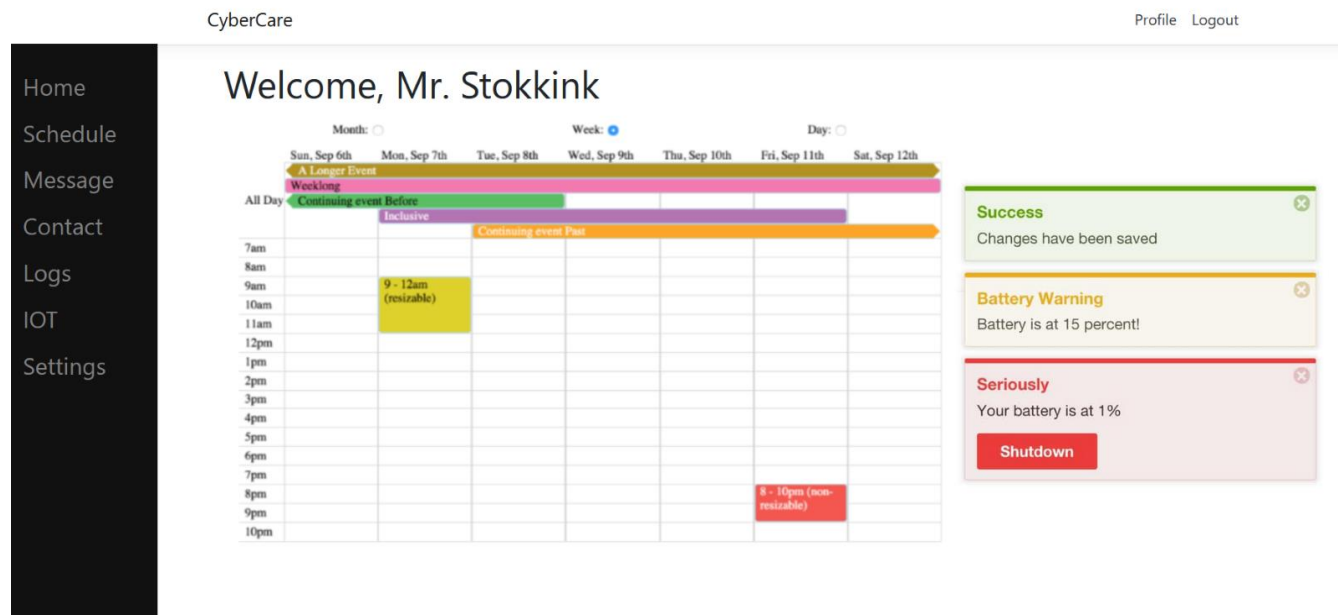


Figure 36 - Concept Web Interface

Regarding permissions for each account, the medical professional will have root access to the application being able to hand out permissions to the family members and patient based on the request of the client. As a default setting, the patient can modify the system settings and schedule through the pill dispenser hosting the web portal. The family member will be restricted from sending messages, viewing logs, and adding events to the schedule. All versions of the web portal will be receiving the relevant notifications and alerts.

Considering some of the more backend functionality of the web portal, a big focus lies with getting commands from the web portal to the greater system. When the user makes a change to the schedule or system settings this must be reflected across the entire system. The web portal's responsibility in this process is to get the change to the central server from where it will reach the affected devices. Since the server stores everything from the schedule to messages in a database, making a scheduling edit is as simple as sending a SQL update query to the API with the relevant information. To ensure the web portal is synced with the version of the schedule on the database, it is a safe idea to first query the database, wait for the JSON response, and compare it before making a change and sending the update query. This does eat up more time, but it avoids running into errors in the long run since the web portal is not the only member of the system that edits the database.

Not so different from sending commands to the greater system, the process for displaying the patient's vital sign data is a matter of querying the database. The trick is querying on the necessary data, especially when

spanning over a greater period by adding a sampling frequency to the query to ensure that the relations between duration and frequency stay balanced. With that issue taken care of going from a JSON file to a graph is straightforward. Loading all timestamps and data points into parallel arrays and representing them as axes.

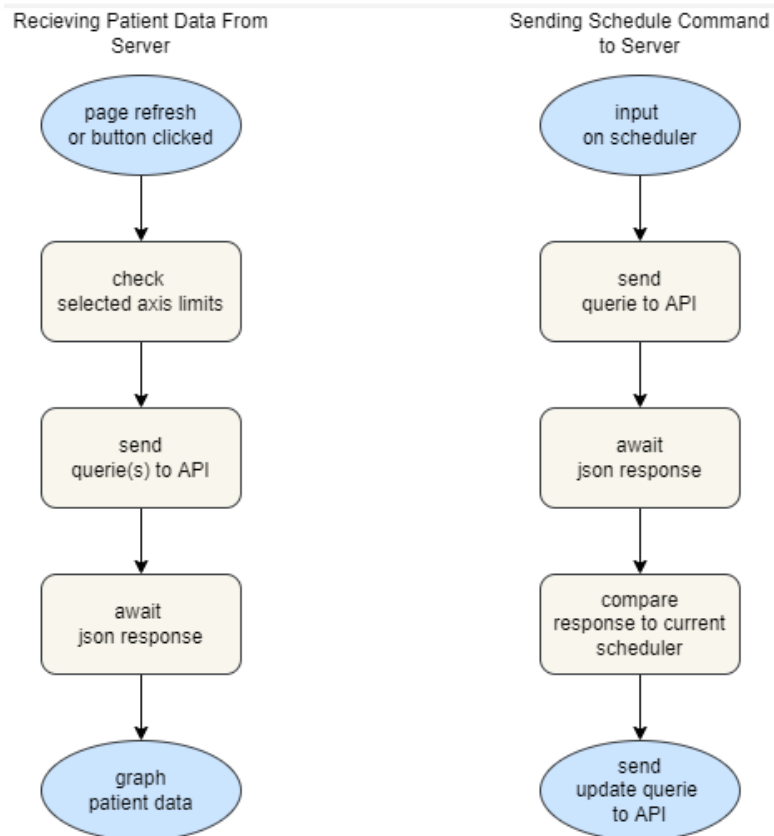


Figure 37 - Web Interface Functional Flowchart