IoT-based Medication Reminder and Dispenser

Report for Group Project

Internet of Things: CITS5506
University of Western Australia
Semester 02, 2022

Group 06 Team Members:

- Nishan Devkar (23762917)
- Charlie Hu (23217014)
- Ame Liu (22910358)
- Alexander Turner (23423442)
- Feifan Wu (22951266)

ABSTRACT

Medication adherence is necessary for the health and wellbeing of any individual taking prescription medication, however correctly managing medication can prove to be a stressful day-to-day challenge. This report outlines decisions made in the design of an Internet of Things (IoT)-based Medication Reminder and Dispenser system that is able to accurately dispense pills according to a schedule set by a user, detect whether a user has adhered to their medication plan, automatically remind patients of upcoming medication schedules, and notify caregivers when medication has been missed. In this report, the limitations of other medication dispensing and reminding systems are first assessed, and key aims and requirements are outlined for the proposed system. The system design is then comprehensively described. A complete overview of components involved in the system, a breakdown of production costs, the process of design, technical challenges encountered, and a description of how these components work together are detailed. An analysis of tests on the system is then presented and the design of the system is assessed. Finally, the significance of the design's accomplishments and shortcomings are discussed, and future developments and ethical considerations are proposed. This device is comprised of: a motor-based dispensing system; an LDR-based sensing system for detecting the presence of pills; a WIFI-enabled gateway for communicating information over a network; a web-based user interface for users to input their specific medication requirements and contact details; and a server for storing user details and sending out email reminders and notifications.

INTRODUCTION

Statement of Purpose

It is estimated that around 35% of all Australians take prescription medicine daily [1]. For many patients and members of their care network, medication adherence is a stressful day-to-day challenge as mismanaging or forgetting to take the correct dosage at the correct time can cause serious health complications. In Australia alone, an estimated 250,000 people are hospitalised each year due to problems caused by their medication [2] whilst in the United States, medication non-adherence may cost an individual between $949 to $44,190, and has been estimated to cost the US healthcare system upwards of $290 billion per annum [3].

Individuals and health care organisations have increasingly turned to technologies to achieve better health outcomes through patient engagement and smart medication devices have shown to significantly help with medication adherence [4]. Specific focus of this effort has been directed towards elderly adults as they are both major consumers of prescription medications and at high health risks of non adherence. One of the main purposes of these devices is to increase the patient's ability to comply with their own medication routines without unnecessary supervision and therefore they are often designed with seniors in mind.

Current Medication Device Research and Analysis

Manual based medication dispensing systems are often unable to support the needs of elderly adults due to the lack of proactive prompts or reduced manual dexterity to operate.

Pill dispensers with automated features vary in their capability which can be matched to the individual's care requirement. Common features include:

- In-built reminder mechanisms to prompt the user to take their medication. Many of these use amplifier components built into the hardware [5]. For these devices, a user has the potential to miss their medication reminder if they are hard of hearing or if they are not in the vicinity of the device at the time the reminder is set.
- Automated medication dispensing where the mediation is presented to the patient at the required time. This approach is preferred to reduce the risk of a patient taking repeat doses unintentionally. This mechanism is often implemented using rotation-based motion using motors or simple linear motion based motion using springs and solenoids. The design of these mechanisms is often complex in order to accurately dispense one pill at a time, be adjustable to different pill sizes, and as to not damage the medication or other device components in the dispensing process.
- Medication adherence tracking. This capability is targeted at home-use where the device can identify if medication has been removed and raise alerts after a period of time. The tracking capability often relies on infrared sensors to detect the motion of the individual's body [6]. This creates a potential for error, where a person may move in front of the sensor without moving the medication.
- Medication adherence alerting is the ability for the device to alert a caregiver where there is an issue with medical adherence or the overall health of the device. This is considered a premium feature and is often implemented using a subscription model.

Capsules and tablets are the most common types of oral medication, however these pills come in varied forms, shapes and sizes. Pills also have tolerance (±mm) in their dimensions as a natural result of their manufacturing [7]. Designing a dispensing system that is able to reliably function with different types of medication is a challenge.

Smart medication devices can be expensive, MedaCube charges an upfront cost of $1,399 USD for their product, and other companies charge monthly fees for use of their software [8]. A smart medication device is often used by multiple individuals in an aged care home, so the features of these devices must be highly flexible in order to support varying needs. However this cost creates a barrier for entry for many elderly adults who live alone, and are often on low-income pension wages. Few affordable smart medication devices are currently available that are able to track medication adherence and automatically update caregivers.

Project Statement

In order to improve the quality of life and safety of the individual, this device is designed to be able to accurately dispense precise medication at correct times, remind the individual when they are to take the medication, and notify a caregiver when the patient hasn't taken their medication. Our IoT-based medication reminder and dispenser device aims to promote medication adherence while still taking into account their current level of functioning, values and preferences.

Taking into account the digital literacy of seniors, our goal was to make both the hardware device and software interface as simple as possible to set up and use. The reliability and usability of this device is particularly important given the serious health implications and associated risk related to failure.

Given that smart mobile phones are commonplace, portable and often kept on hand, the device developed as part of this study uses an email system to handle both the reminder and adherence alerting functionality. This has the additional benefit of reducing hardware costs for the construction when compared to local reminder-based alternatives.

To track medication adherence in a non-invasive manner, the device uses a photoresistor (LDR) sensor to detect whether medication has been taken to overcome the identified limitations of the IR model used by many market devices.

It was noted that many market designs included removable pill holders. This could be problematic, as the individual could misplace the pill holder. Using dispensable pill holders is not an environmentally friendly nor self-contained, sustainable solution and so was discounted as a potential option.

The device was designed to be affordable with low upfront costs and no subscription fees for software use.

DESIGN SUMMARY

The hardware component consists of two subsystems. The first is the storage and dispensing system, designed to store the medicine and dispense the medicine when the hardware receives a dispense message from the web server. The second is the sensing and communicating system, designed to perceive the state of the medicine (whether present or not), receive messages from the web server, react according to the message, and send messages to the web server.

The database is stored on a cloud server. The user details are stored in this database. The web application front end runs on the cloud virtual machine. Communication between the hardware and server is enabled via WiFi technology.

The system's flow of data between the device user, software and hardware is presented in Figure 1. The system's sequence for registration, medication plan creation and medication reminding is presented in Figure 2.

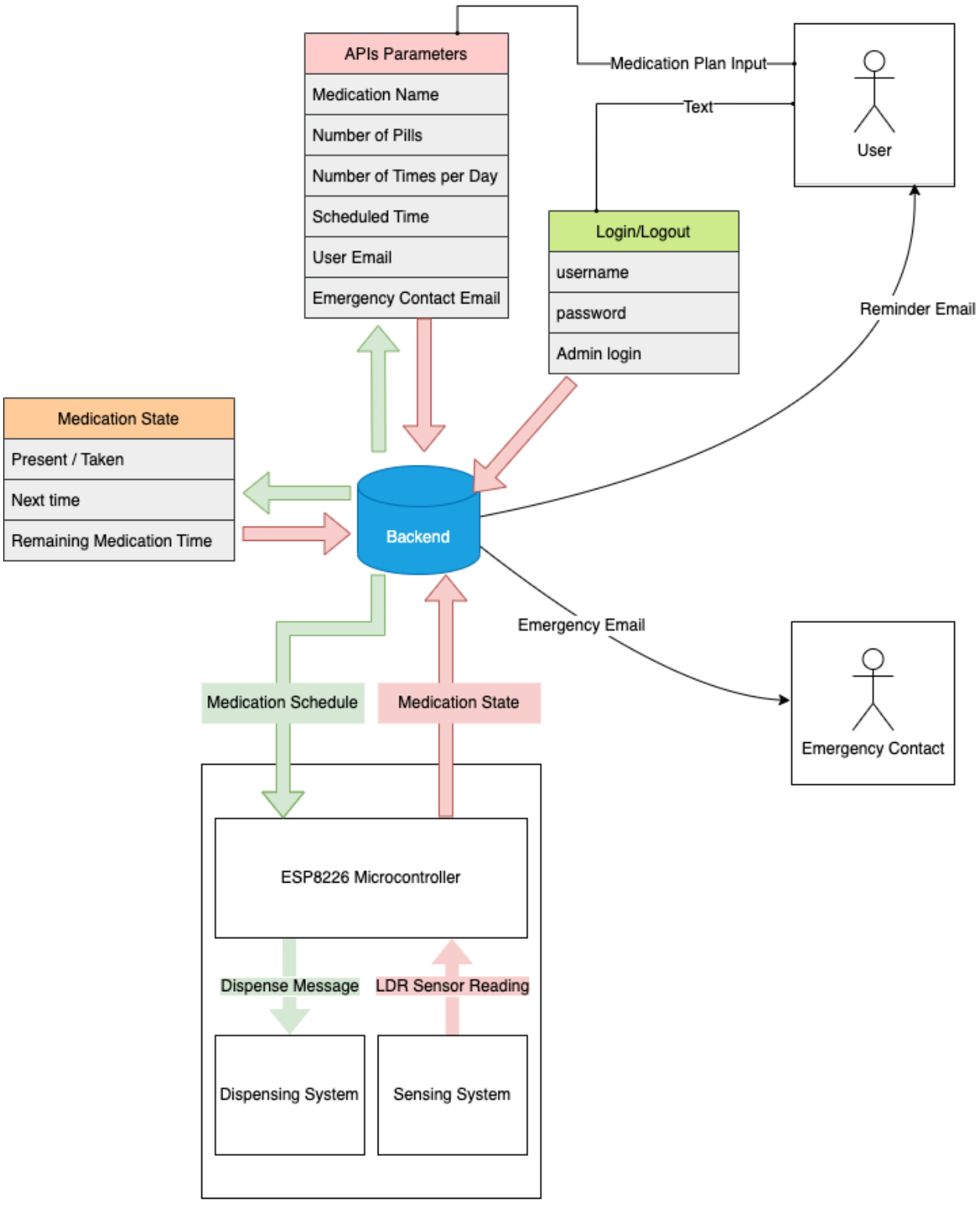


Figure 1: Data flow

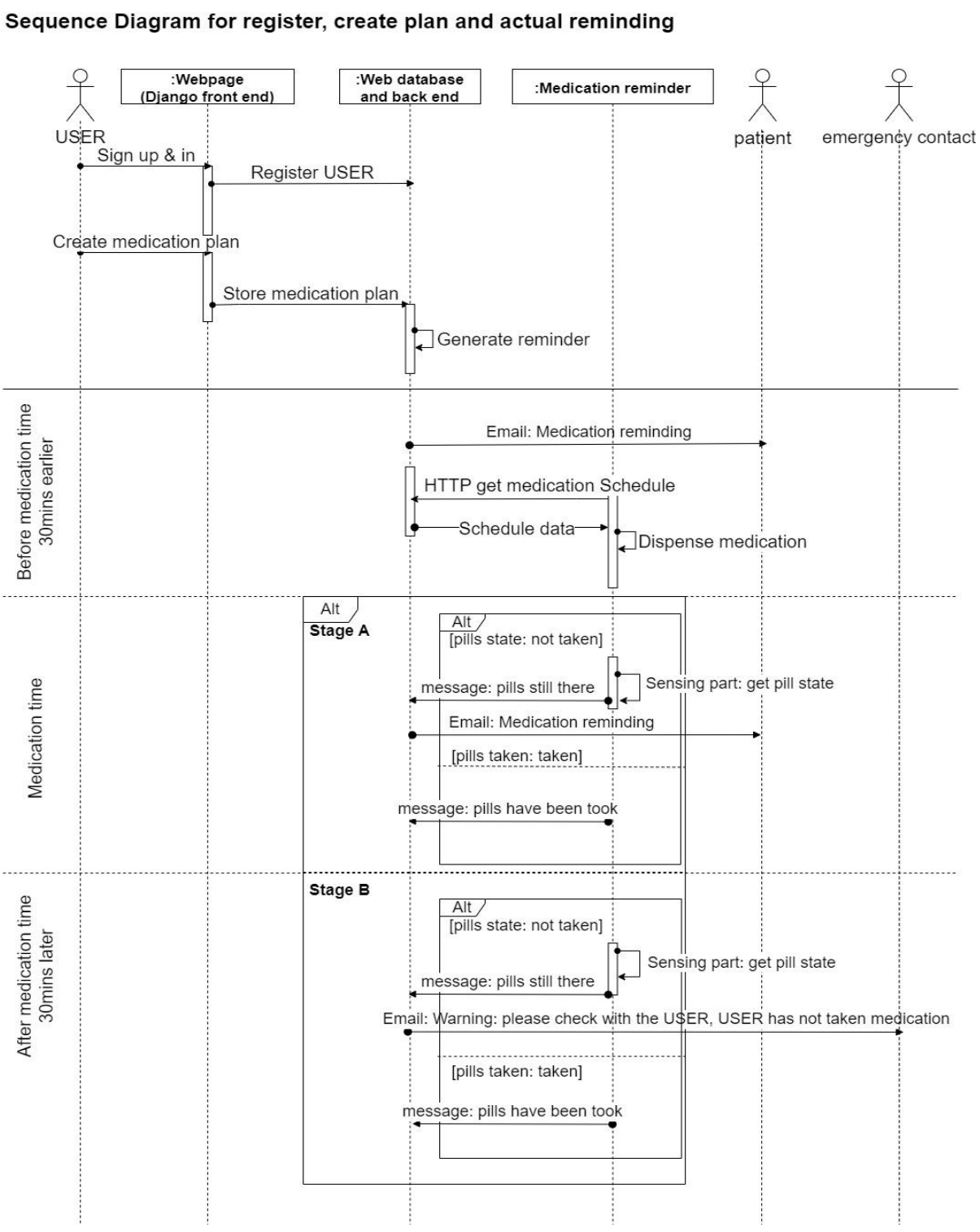


Figure 2: System sequence diagram for registration, creating plans and reminding

PROTOTYPE DEVELOPMENT

Required Components

Hardware Electrical Components and Tools

Hardware components were purchased from Jaycar Electronics store [9] and Altronics Electronics store [10]. The cost per unit, quantity required, and function of purchased components are included in Table 1. Materials that were sourced free of charge from the UWA Makers Club and are included in Table 2. The materials that we sourced are readily available from electronics stores such as Jaycar and Altronics. The total budget for this project was $44.80.

Table 1: Components and supplies purchased

Item Description	Quantity	Unit Cost	Store CODE and Web Address
WiFi ESP8266-ESP-12F Development Board	1	$21.95	Altronics (Z6381)
Arduino Compatible Photosensitive LDR Sensor Module	1	$5.95	Jaycar (XC4446)
Arduino Compatible Mini Breadboard with 170 Tie Points	1	$4.95	Jaycar (PB8817)
Arduino Compatible 9G Micro Servo Motor	1	$11.95	Jaycar (YM2758)

Table 2: Sourced components and supplies

Item Description	Quantity
1.5kΩ resistor	1
5mm white LED	1
Jumper leads (Plug-Socket and Socket-Socket)	8
Black electrical tape	1
Micro USB cable	1
USB wall plug adapter	1
3D printer filament (PLA)	~ 200g

Hardware Structural Components

All structural components required for this project are included in Table 3.

Table 3: description of structural components required

Figure	Label	Component	Component Description and Dimensions
Figure 3	1	Dispenser structural support	To provide support to the dispenser system.
	2	Rotating dispensing disk	3D printed semicircle. 3mm height, 30mm diameter.
	3	Storage container	Where the medication is stored. 5mm diameter, 32mm length.
	4	Base plate	To support the pill transportation. Statically positioned underneath the rotating dispenser disk.
	5	Delivery track	To deliver the pill from the rotation disk to the point of collection.
Figure 4	1	Sensing support	To provide structural support for the LED.
	2	LED	To provide the LDR with a constant light source.
	3	Pill holder	The point of collection, where the user takes the pill from.
	4	LDR container	To prevent ambient light from affecting the LDR sensor.
	5	LDR	To detect the presence of the pill.
Figure 5	1	Central structural support	To provide structural support for the microcontroller.
	2	Breadboard	To provide circuit connections with the microcontroller.
	3	ESP8266	Microcontroller that processes sensor readings, receives and sends messages.
	4	MicroUSB	To supply power to the microcontroller.
	5	Device casing	To house the device.

The design of the storage and dispensing system with labelled structural components can be seen in Figure 3. The red line indicates the movement of the pill when dispensed and the blue dashed line indicates the movement of the rotation disk after receiving a message to dispense. The design of the sensing system with labelled structural components can be seen in Figure 4, with the orange dashed line indicating how light travels from the LED to the LDR. Figure 5 shows the structural components involved in the communication system.

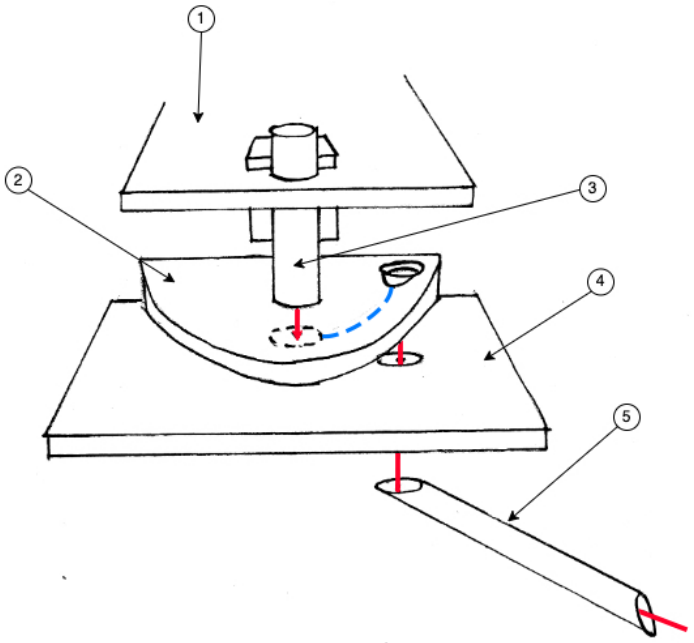


Figure 3: Design of the dispensing system

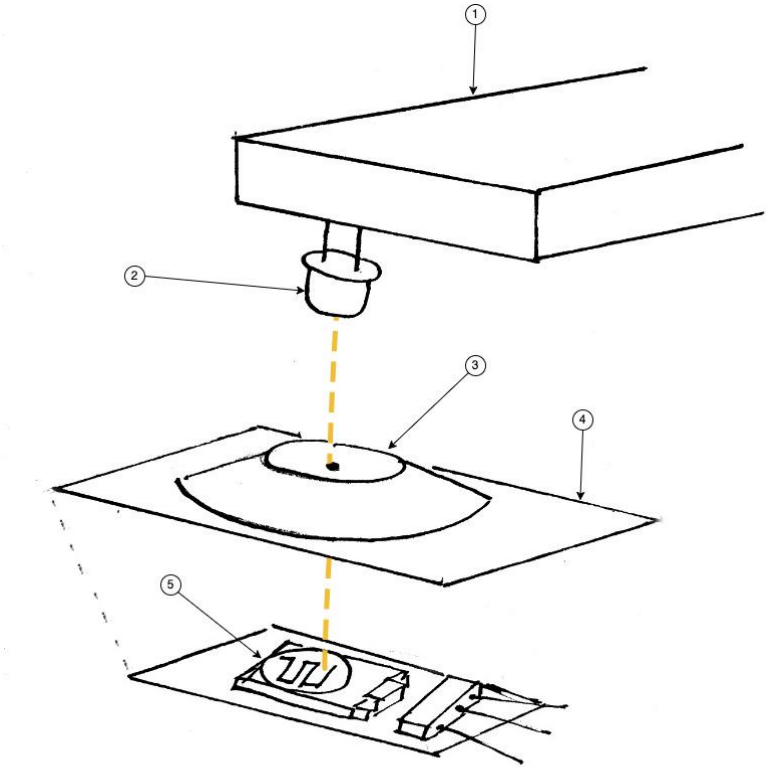


Figure 4: Design of the Sensing System

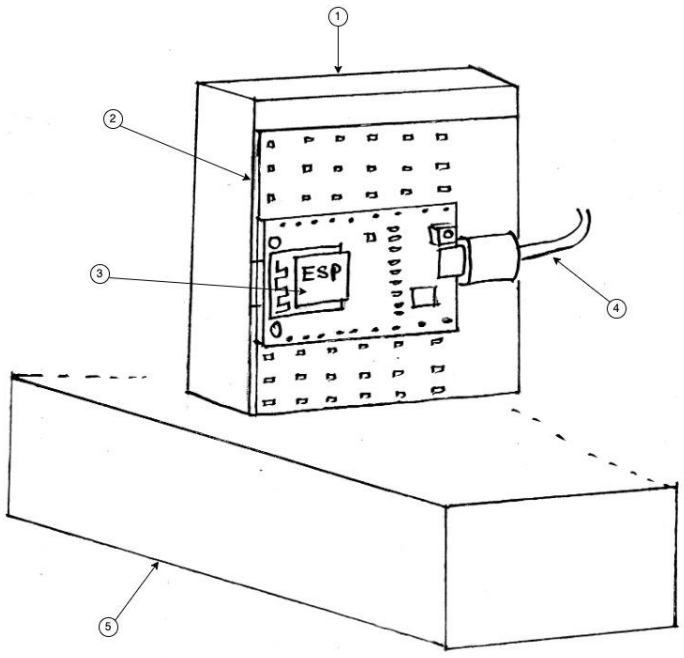


Figure 5: Structural Components of the Communication System.

Hardware Manufacturing Specifications

Our device was built to work with a pill of fixed size and shape, 6mm diameter and 3mm height. This pill dimension was chosen as it was similar to that of an aspirin tablet, a commonly prescribed medicine for seniors [11]. All dimensions and geometrical aspects of the hardware components were adapted to suit this pill size. Initially, we thought that most of our structural components would have to be printed, given the specific dimensions of the medication tablets. Throughout the design process however, we ended up sourcing a lot of materials from around the UWA Makers laboratory that were able to successfully prove our concept.

Printed components were created using the Ultimaker 2+ 3D printer available at the UWA Makers laboratory [12] [13]. Tinkercad web application [14] was used to create and edit the designs of 3D printed components. We used Ultimaker Cura 5.1.1 [15] for the slicing process and generating STL files for printing. For all printed components, we decided to use Polylactic Acid (PLA) filament as it was conveniently accessible to us, though Polyethylene terephthalate glycol-modified (PET-G) was also a potential filament option. In order to work the 3D printer, we applied a thin and even layer of glue from a glue stick before the hot bed started heating up. This ensured that the components remained sturdy throughout the printing process. We used a 0.4mm nozzle when printing, as it typically provides a good balance between speed and quality. Given the specific dimensions required for the medicine to fit through the track, we didn't want the print quality to be too low. For the sake of time and cost and strength of the print, our printer settings were set to print the first two layers at 100% fill rate, following layers at 20% fill rate, and the final two layers at 100% fill rate.

Hardware Design

Microcontroller

We programmed the ESP8266 using the Micropython Firmware version 1.19.1 [16] and uPyCraft [17] developing environment. See **Appendix C** for a detailed process of coding the microcontroller. Once the code and the firmware is burnt into the chip, we are able to power the microprocessor using a microUSB plugged into a wall socket. We double checked that this would not damage any of the electronic components by testing it with the prototype.

Hardware Circuit Design

The hardware circuit design for the device is presented in Figure 6. The sensing circuit design was adapted from a guide written by CodeChamp [18]. The dispensing circuit design was adapted from a guide written by rahuladitya303 [19].

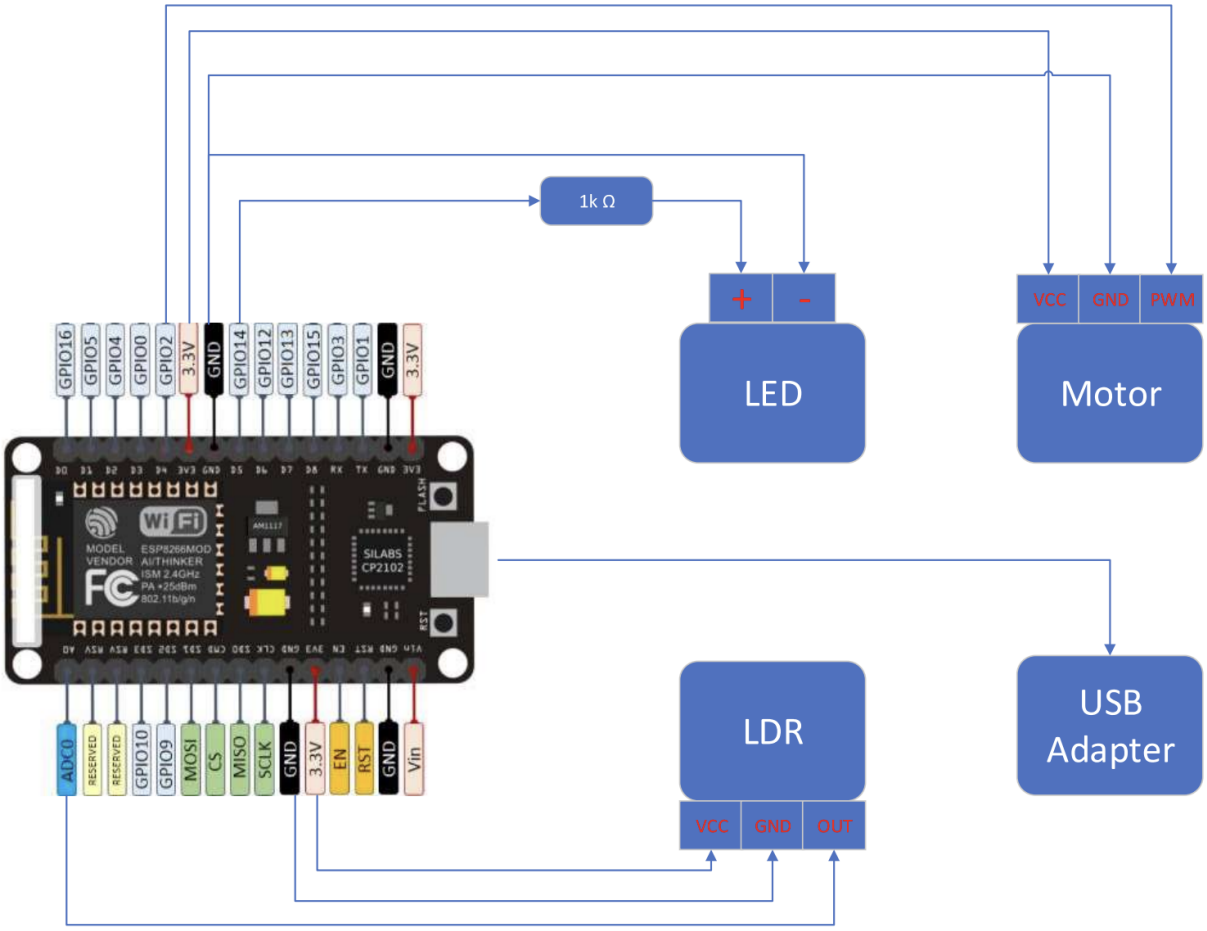


Figure 6: circuit design to enable sensing and dispensing functionalities

Dispensing System Summary

We began by identifying the requirements of the dispensing system:

1. It should retrieve one pill at a time from the storage container
2. The dispensing process should not damage the medication
3. It should be able to dispense as per the schedule

Storage Container

The initial design for the storage container component was a simple funnel shape (see Figure 7). In this design, the pills stacked together at the base of this printed funnel shape. We lightly shook the storage component to assess whether a vibration device attached to the storage container would be a viable solution, however vibration was unreliable in dislodging any jammed pills. For our second design, we printed a hollow cylindrical container. For this version, the diameter of the cylinder was large enough for the pills to still be able to get jammed vertically. This did not occur if the pills were stored into the container carefully, so we decided to revise the dimensions of the cylinder. We determined that its diameter would need to be slightly larger than that of the pill (5mm). Whilst making changes to the 3D model design, we found that the outer shell of a pen had the precise diameter that we required. This cylinder storage container was robust to the problems identified with prior designs. We noted however that the cylindrical shape increased the complexity of the refilling process.

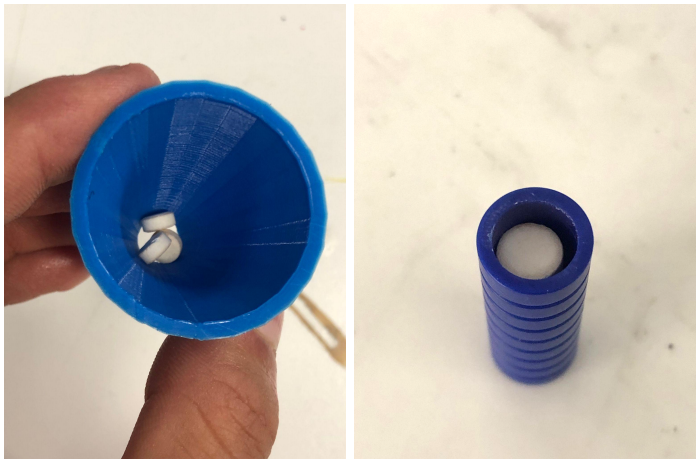


Figure 7: Pills stuck in a funnel version of the storage component (left) and pills successfully stored in the cylindrical storage component (right).

Dispensing System design

When the microcontroller receives the message to dispense, it sends two messages to a motor. The first message moves a rotation disk from the zero position to the drop position. At the drop position, medication is transferred from the rotation disk to the medication track. The second message moves the rotation disk back to its zero position, where it collects another pill from the storage unit.

A single hole in the rotation disk is required to collect the pill from the storage container. In order for only a single pill to be dispensed, the pill needed to be able to reliably drop and fit into this hole. The height of

the rotating plate was designed to match the height of the medication. The diameter of the hole in the rotating plate was designed to be slightly larger than the diameter of the medication (6mm).

Despite considering these precise measurements in our initial rotation disk print, a second pill was still able to drop down slightly into the rotation disk from the storage container. With the pill tolerance in mind, adjusting the height of the rotating disk to match the height of a specific pill would not be appropriate. We solved this problem by shaving a slope to the edges of the hole on the rotating disk using smooth sandpaper. Any rigidness could potentially cause problems with dispensing, therefore, if the prototype were to be replicated, we would suggest building this slope into the 3D model. This slight gradient allows the pill to slide instead of getting caught. The impact of this slope may not be noticeable for capsule pills or tablets with significantly curvature, so we were fortunate to catch this dispensing issue early through testing the system with a flat tablet pill. The thickness of the rotating disk was changed throughout the design process from 4mm (1mm larger than the pill) to 3mm (the approximate height of the pill).

A static plate was required to transport the medicine from the rotating disk to the medicine track. The rotating disk needed to be mounted precisely on the static plate. A large gap between the plate and the disk resulted in a second pill being dispensed. When the static plate was too close to the disk, it prevented the motor from turning and dispensing the pill.

We measured the minimum and maximum angle of the motor rotation to determine the limit of the motor's rotation. We identified the zero position of the motor and fixed the rotating plate to the motor at this zero position. After a series of rotations however, the motor kept rotating but the disk remained stationary. The motor did not precisely fit the design of the rotating disk, resulting in the rotating disk becoming unstuck. First, we tried using superglue to secure the join between these two components. Eventually, we filed down the servo arms on the motor and used screws to join the components. The precise degree of rotation was calculated once all other structural components had been set in place.

Pill Holder and Medication Track

The pill holder had to have an inclination such that the pill reaches the bottom of the pill holder and covers the pinhole above the LDR. If the pill holder is too deep, then the pill may become difficult to retrieve. The medication track had to be designed and positioned in a way such that it was not obstructing the rotation of the motor and rotation disk, nor obstructing the light from the LED from reaching the LDR.

For the first version of the pill holder component, we printed a simple hollow hemisphere shape. We transferred a pill into the pill holder down a makeshift medicine track to assess the effectiveness of this print. Increasing the angle of the medicine track from the ground increased the speed of the pill but decreased the horizontal projection of the pill. An angle of approximately 30degrees appeared to be optimal; at an angle greater or less than 30degrees, the pill bounced out of the pill holder, and at very low angles, the pill didn't always reach the pill holder.

The pill also regularly became caught on the sides of the pill holder, not slipping into the bottom as we had hoped. As the base of the pill holder was printed first, at a relatively fast speed, the coarser side of the 3D filament was on the inside of the pill holder. We hypothesised that the rough surface of the filament could be increasing friction. We considered reprinting the component, however found a speaker cone with a similar shape and dimension, as well as surfaces to easily attach to other components.

Sensing and Communication System

The tasks taken to achieve a proof of concept for the sensing system involved: building a mock enclosure, designing the photoresistor sensor circuit with the ESP8226 microcontroller; calibrating the sensor (see Figure 8) such that if the pill is obstructing the sensor, the sensor value should be less than the base sensor value (threshold value set by the LED light).

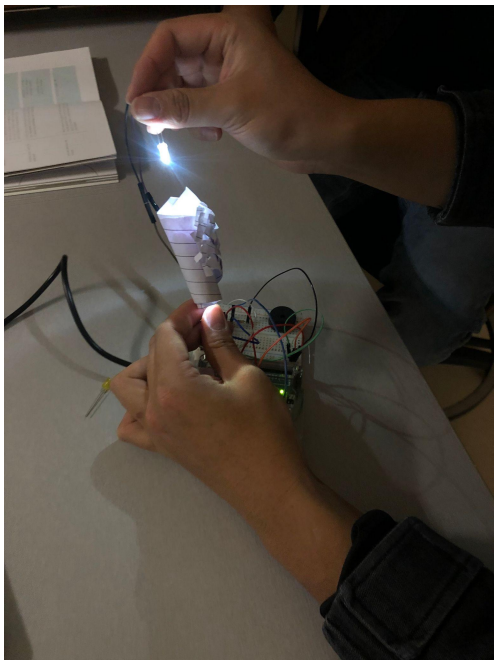


Figure 8: Initial tests to calibrate the LDR under different lighting conditions.

For the medication sensing system, we conducted a number of tests to assess: the proximity of the LED from the sensor; the size of the LED/LEDs; the quantity of LEDs used; dimensions and properties of the pill holder.

We started by building the container to house the LDR sensor. This container has a small pin-hole in the centre to expose the LDR to the outside environment. The LDR must be fixed in position within this box, to prevent it from moving and impacting the detection of the pill. It was essential to prevent all other ambient environmental light from entering this container, so the container was then taped once the LDR was fixed in place. We tested the LDR in a dark environment and, as expected, the LDR wasn't sensitive enough to distinguish the difference between a dark room and when it was being covered by a pill. This confirmed that a consistent light source was necessary, so we opted to use an LED due to its small size and brightness.

We required the LED to be at least 4cm above the pill holder in order for there to be enough room for the individual to easily remove the pill. We started testing the sensing system with a 2.5mm LED from this distance, however this did not produce enough light for the photoresistor to measure accurately. Specifically, the values reported by the LDR ranged from approximately 8 to 12 when the LDR when the pinhole was covered and exposed respectively. Also, despite the pill remaining static, the LDR readings weren't stable. To account for the fluctuation in readings, we programmed the LDR to calculate the average of 5 readings taken over the course of 5 seconds. Even if the pill was removed during the time

taken to complete this process, enough light enters the black box to skew the average. This was then decreased to 5 readings taken over the course of 1 second, to decrease the chance of miscalculating the presence on the pill based on unintentional movements of the container.

We changed to a larger 5mm LED. The LED was very bright without any resistor in the circuit. It generated enough heat to burn the breadboard, and would die very quickly, so we added a 1.5 KΩ resistor. To further increase the lifetime of the LED, instead of having the LED remain constantly on, we programmed it to turn on just prior to each sensing process, and to turn off after the completion of each sensing process. This was achieved by reconnecting the LED to a GPIO pin on the ESP8226.

Housing Container

The housing container for the device was designed last to match the dimensions of the hardware. The housing container is a box with dimensions of approximately 6cm in length, 12cm in width and 11cm in height. The final design of the project can be seen in Figure 9.

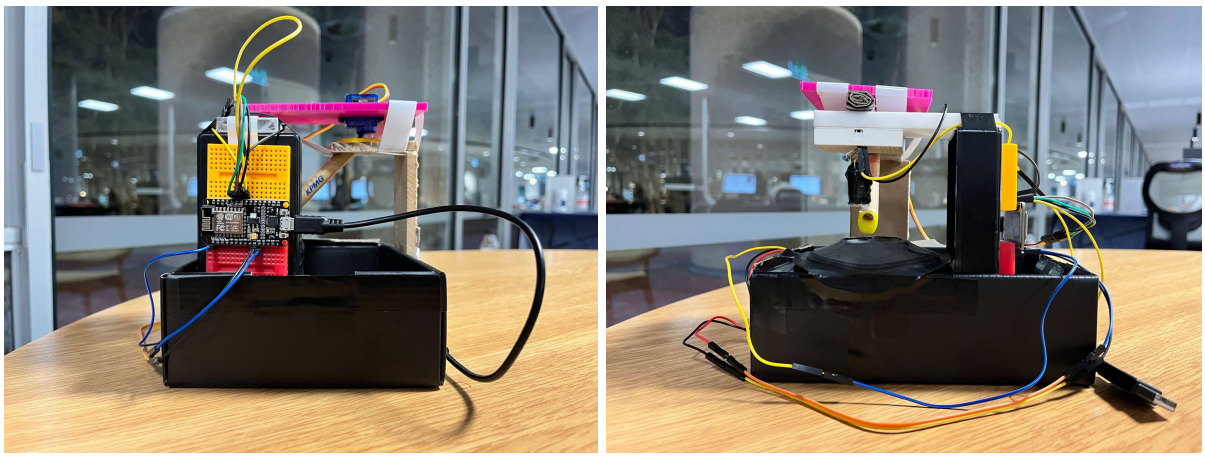


Figure 9: front view of the final design (left), side view of the final design (right).

Software Design

Software Requirements

We started the software design process by identifying the key requirements of the software components:

- Privacy and security: patient data should be stored and accessed in a secure manner
- Accessibility: content and design should be simple to understand, and should consider the needs of people living with disability
- Scalable: the interface should be able to support multiple users
- Reliable: the interface should function at an efficient speed, and be robust to bugs
- Responsive: the interface automatically adapt to the viewports and screen sizes of different devices
- Smart: the software should automatically and reliably send reminders to the user and notifications to the caregiver based on user medication adherence

Software Summary

After identifying key components of the user interface we built a wireframe (see Figure 10) to outline the structure of the web application.

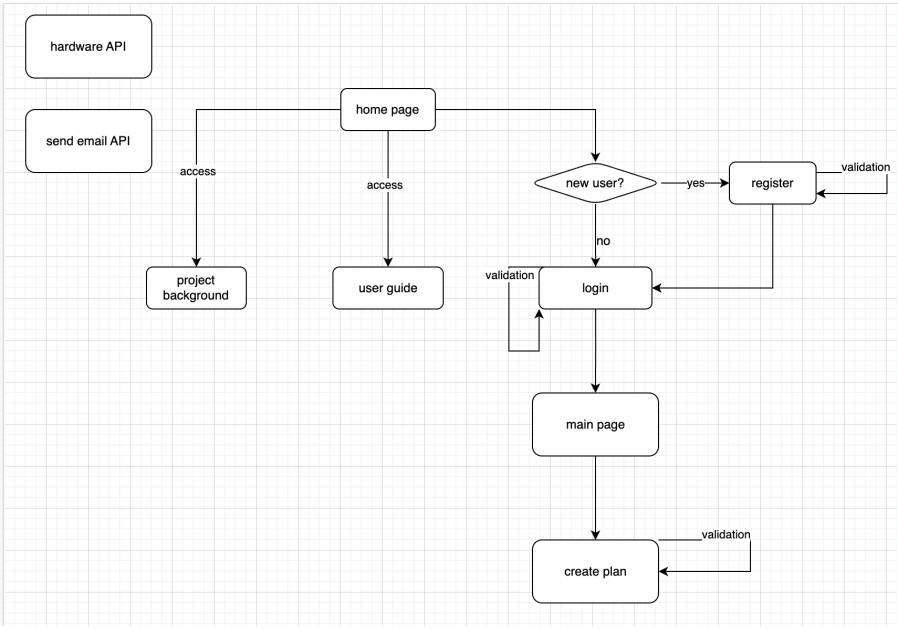


Figure 10: web application wireframe

The web application was set up as a front and back-end model. Jinja2 was used to fetch the back-end data from front-end requests made by the user. In order for the web application to interact with the hardware, we used the Django framework and the Django REST module to provide communication between the back-end and the hardware. The static web application components (HTML, CSS, JS) were written and compiled with Django in the Visual Studio Code IDE. This Python-based web framework was used as it is easy to maintain, easily scalable, and integrates well with the backend software. We developed the frontend of the web application using a mobile-first practice with Bootstrap CSS framework in order to ensure responsive design. Changes made to the web application code were tracked using Github and can be viewed by visiting: https://github.com/Charlie-Hu/djangoturtle. We used a database created with MySQL [20] in order to securely store and access patient information and account details.

On the homepage of the web application, the user is able to read a brief introduction on the device and the goals of the device, read a user guide that provides a simple step-by-step guide on how to use the application; and register / login to the web application. When registering, the user provides their contact details. After successfully registering an account and logging in, the user is able to create a medication plan to set up the device based on their current prescription medication requirements. Multiple medication plans may be created. The medication plan requires the medication details (name of medication), number of pills, number of times per day (that the pills are dispensed), scheduled time (time for the medication to be dispensed, inputted in 12hr time), emergency email (for a member of their care network).

Local Deployment

In order to set up the local deployment for development of the web application, we: installed Pip3 package manager, Django framework, and Python virtual environment; deployed the code from GitHub; installed

MySQL and created a database; ran the project in the Python virtual environment. This local deployment process was implemented using the Mac OS command line and Linux virtual environment in AWS. See Appendix A for the detailed approach taken for local deployment.

In order to construct the user interface, we created functions to: render the URL from the backend; provide login, logout and registration functionalities; access user data stored in the backend based on requests made by the user on the frontend; create and delete medication plans; send user data to the hardware; retrieve sensor data from the hardware; and send notifications to the user and caregivers. The process taken to achieve each of these steps is detailed in Appendix B. User identification (login and logout operations) were achieved with the User module available in the Django framework [21]. Jinja2 was used to access the status of the login from the front-end user interaction [22].

To ensure simplicity and reduce cognitive load, our web application only displays essential information. Background information on the product, and a brief guide to using the product are stored within modal boxes and are accessed via buttons. We centred and enlarged these two buttons to draw the focus of the user. We utilised colours (pink and red) and symbols (supporting hands and heart shapes) commonly associated with health and care to provide visual aids to make the application more intuitive to use (see Figure 11).

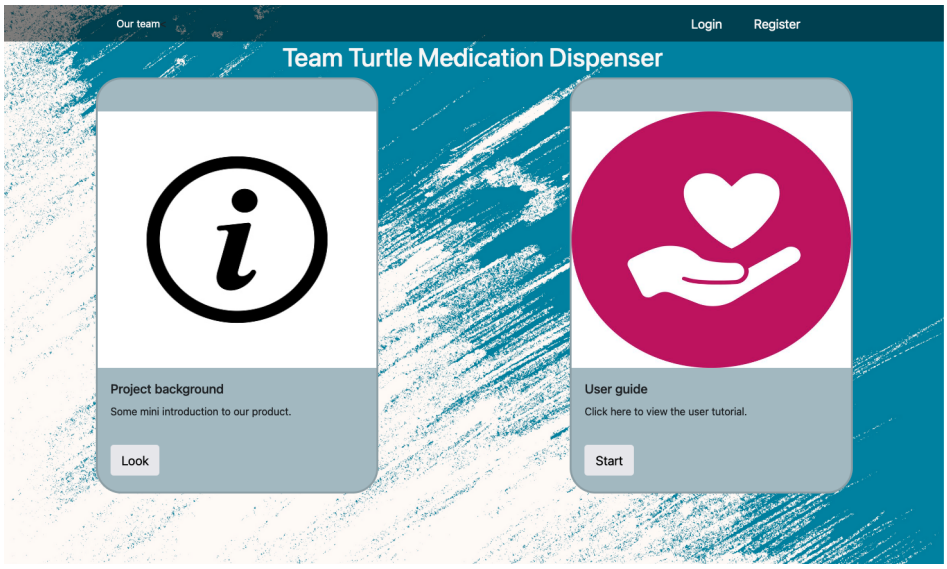


Figure 11: Web application home page front-end design (viewed from a laptop browser).

Accessibility was considered throughout the front-end design process. This process involved: deciding upon legible fonts; revising the size of elements and colour contrast; using alternative text for images; declaring the document language in the head of each HTML page; using aria-labels when necessary for hyperlink tags; and ensuring that semantic HTML was used for HTML elements when possible.

Communication with Hardware

The next step was setting up the communication between the software and the hardware interface. To achieve this, the hardware needs to access the data from the URL and parse the data by adding parameters to the header of the URL. The results of this can be seen by visiting: http://3.26.197.0/hardware/?username=Charlie&password=123.

The final stage of the software development involved deploying the web application. This application was deployed using Amazon Elastic Compute Cloud (AWS EC2), proxying with Nginx [23]. The current-version project can be accessed via: http://3.26.197.0.

The smart-reminder component of our project relies on Simple Mail Transfer Protocol (SMTP) to automatically send out email reminders based on the individual's medication adherence. The first notification will always be sent 30 minutes in advance of the individual taking their medication. The second reminder is sent at the time that the medication is dispensed. If the pill has not been taken by the individual after a given 30 minutes, the SMTP will notify caregiver/s.

We then tested the device to ensure that emails are successfully sent and received (see Figure 12)

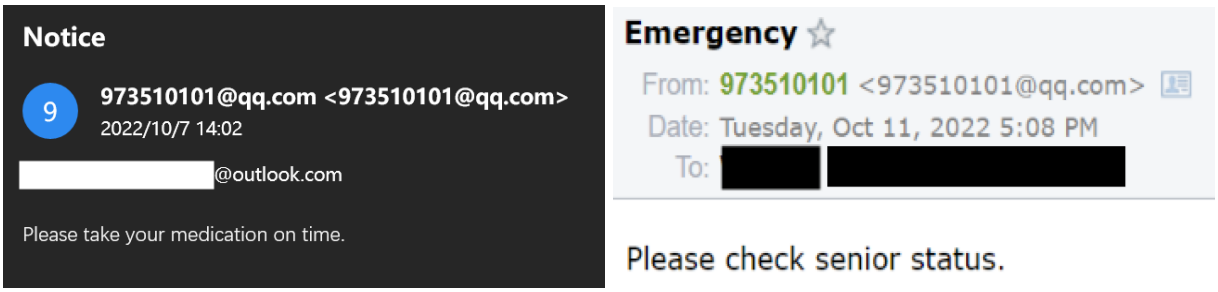


Figure 12: Email reminder received by user prior to the dispensing of their medication (left) and email received by caregiver if user misses their medication (right).

DATA ANALYSIS

Comprehensive testing and optimisations were conducted to identify, record and resolve bugs, anomalies, and errors in the hardware and software components. We tested the system under four different usage scenarios to test the system stability and collect data.

Experiments Summary

Experiment 1: analyse the responsiveness of the device. The purpose of this test is to assess whether the device can remind the user to take the medicine on time.

Experiment 2: test the stability of the device for long-term operation. The purpose of this test is to test whether it can be guaranteed to be reliable for a long time.

Experiment 3: count the number of tablets delivered by the device at a time. The purpose of this test is to assess the accuracy of the dispensing system in order to ensure the safety of the user.

Experiment 4: test the responsiveness of the web application in different situations. This test is designed to ensure that the website application can always guarantee the reliability of its functionality in different network environments.

Results of Experiments

The CSV files containing data for each experiment can be downloaded from:
https://github.com/alexrturner/IoT-based-Medication-and-Reminder-System.git

Experiment 1

For a reminder to be accurate and reliable, the user must receive the reminder within the allocated medication reminder time ±2mins. This test is designed to measure the timing of the prescribed alarm clock 10 times, such as 8:10 8:20 8:30 ~ 9:40 for a total of 10 moments, to see how many times it has been successfully prompted.

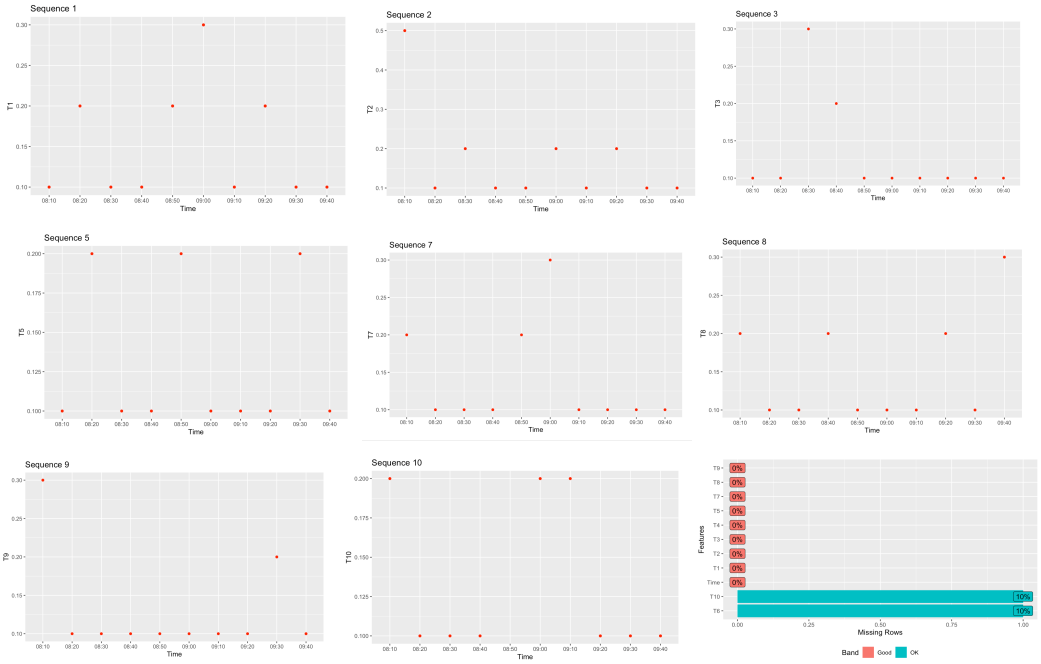


Figure 13: Results of experiment 1

The test results of experiment 1 (see figure 13) show that only 10% of reminders (10/100) didn't meet the experiment criteria. The overall pass rate is 90%.

Experiment 2

The device was run over a long period (72 hours). Every 24 hours, the device was tested to assess whether it functioned as expected. Test 5 times per time point.

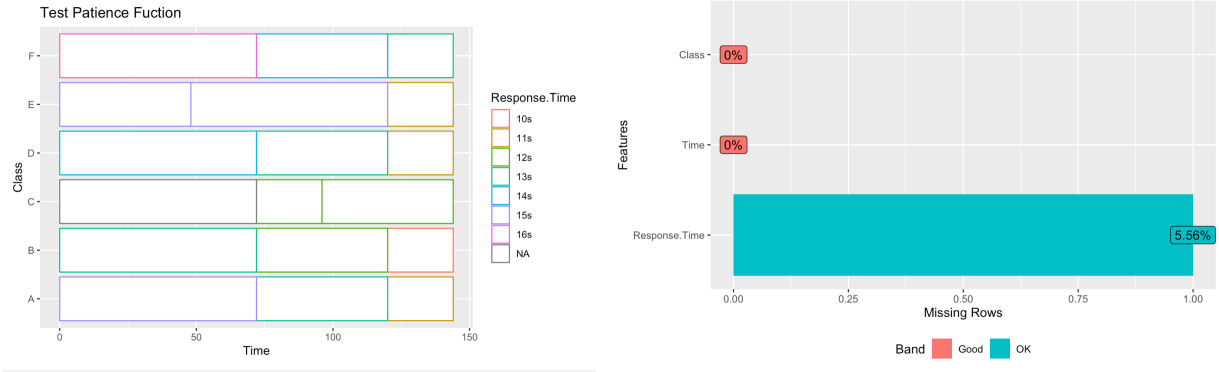


Figure 14: Results of experiment 2

The results of experiment 2 (see Figure 14) indicate that only one of the 5 tests after 72 hours was unsuccessful. The overall reliability of the 24-hour operation is very high.

Experiment 3

Test the tablets 100 times to see if the tablets delivered by the device are accurate. The results of experiment 3 are displayed in Figure 16. On two occasions, less tablets were dispensed than required, and on one occasion, the device dispensed one more tablet than required. Based on 100 trials, the error rate of the dispensing system is 3%.

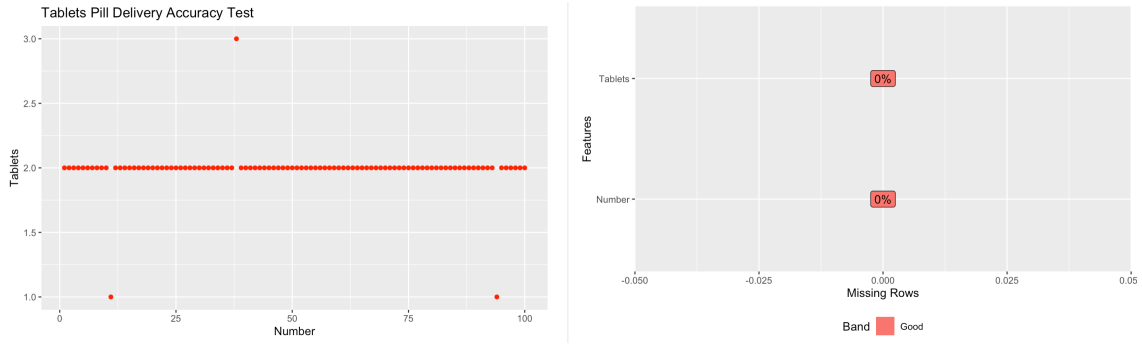


Figure 15: Results of experiment 3

The results of experiment 3 (see Figure 15) indicate that only three tablets were placed once in 100 placement operations, and 1 pill was placed twice, which means that only 3 of the 100 placement problems occurred, and the reliability reached 97%.

Experiment 4

Test to see if the page will function properly whilst 10 people visit the login page at the same time. A total of 20 tests were conducted.

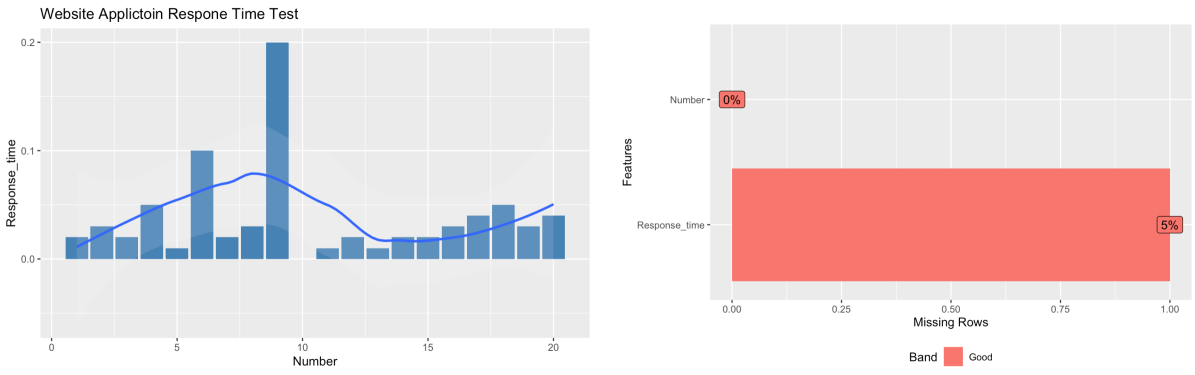


Figure 16: Results of experiment 4

Assessing the results of experiment 4 (see Figure 16), we found that only 1 time did the web application fail to load. The response time in the remaining 19 experiments was controlled within 0.2s. The pass rate is as high as 95%.

CONCLUSION

Final Design Assessment

Overall, the results of the experimental data analysis were positive. The device software is stable and reliable, with the device able to support many users visiting the web application at the same time. By the end of the project, a successful working prototype was made. The end result is a fully functional proof of concept that met most of the objectives laid out in the statement of purpose section. The device is self-contained in the sense that it can function as expected anywhere with a general power outlet and WIFI, and it is able to carry out dispensing and reminding operations based on the user's inputted medication plan. A video recording of the final product successfully reminding, dispensing, sensing and sending notifications can be viewed via: https://www.youtube.com/watch?v=p9gR4HVszik

Final Design Discussion and Future Considerations

Most of the device problems can be attributed to the complex design required for the dispensing system. Specifically, this model requires pills to be inserted one-by-one into the storage container. The precision required to ensure that the pills don't stack vertically on top of each other may pose a particular challenge for individuals who lack fine motor skills, which is common for the target demographic of this project.

Initially, we planned to use Fusion 360 [24] to design all of our 3D components, however, with little prior knowledge, we severely underestimated the time required to effectively design and edit 3D printed components. In future developments of this design, we would spend significantly more time researching similar components and devices listed under the Creative Commons Attribution licence. Specifically, the iterative process involved in designing and positioning structural components to achieve a functioning dispensing system proved to be a significantly complex engineering problem. Getting the correct tolerances of this device took a significant amount of time, which is a shame, given the fact that it can't scale easily to work with other pill dimensions. At the end of this process, we have more respect and admiration for open source communities and designs. We would definitely lean more on the prior work of others actively designing medication dispensing devices and look to a hardware structural design that doesn't require such precise measurements.

Though the software is able to support an individual who takes multiple medications, the hardware component of our design is limited by two major factors. Firstly, the design of components is fixed to a specific size of pill, and secondly, the storage component is only suitable for one type of pill. Multiple types of medication could be supported relatively easily by increasing the number of storage containers, rotation disks and medication tracks. This dispensing system design would follow a similar structure to that of the MedsGenie pill dispenser created by Wojtek Siudzinski [25]. This solution is still not easy to assemble or refill. It also isn't able to dispense different sizes of medication.

A potential solution could involve angling a large storage container such that pills collect in one position, and having a rotation disk that sits underneath the pills in order to collect and dislodge the pills. This would follow a similar solution to that provided in by R, Astronaut [26]. Interestingly, the design of this device is not as accurate as ours at dispensing pills first try. It uses a vibration sensor fixed to the pill container to detect if the pill was successfully dispensed, and if dispensing is unsuccessful, the dispensing process is tried again. Though a modular storage component was not a requirement that we initially identified, having a device designed such as this (that allows for storage components to be easily removed and added) would solve many of our problems.

Hardware Considerations

We experienced many challenges when testing how hardware components interacted with each other. Given that we frequently needed to make only minor adjustments to the positions of our components, we avoided printing components with fixed connection points for the sake of time and cost. As a result, we often held components together loosely with tape. If a single component was pushed or moved slightly from its desired position however, the whole system suffered. Dealing with such precise measurements often made it difficult to identify the exact source of the problem. We could solve this by reassessing how components are joined, and implement this by redesigning 3D printed components for simple assembly and strong connections. Snap fit joints (studs, hooks or beads) would be easy to assemble and strong, however we may have difficulties designing them, as it would again be an iterative process to achieve the correct tolerances [27]. Specifically, it would have been better to use hot glue to provide a strong but less permanent bond instead of tape for this iterative design process.

Dispensing and Storage Considerations

One of the largest challenges that we faced involved designing the dispensing system, specifically the issue of how to accurately dispense one pill at a time. Pills require to be dispensed one at a time - to separate the pills when they are stored was quite difficult.

Sensing System Considerations

When using a single LED, if it breaks, then the entire sensing system would stop working. Using multiple supplemental LEDs would not only provide more light to the LDR and therefore increase the accuracy of the pill detection system, but also be robust to situations where one LED breaks. In this system, an email notification that addresses replacing a broken LED could be sent to the caregiver if the LDR value dropped below a certain point. Another notification system could also be set up to contact the caregiver if the device regularly loses WIFI connection.

This project makes the assumption that the caregivers/users have a smartphone with email push notifications available. We are aware that email notifications may not be the most reliable method to receive important information. SMS is a potential alternative solution of communication, however an android application would be perhaps best suited if using smartphones to deliver notifications.

Communication and Notification System Considerations

In this design, the code, URL, User ID and Password are burnt into the microprocessor. If we were to develop this into a sellable product, we would need to approach software and hardware communication differently. This could potentially be achieved by developing an Android application, as the user would be able to input their user ID password etc manually by connecting to the ESP microprocessor via WIFI. The labour required to develop an android application was outside the scope of this project.

Currently, the reminder and notifications are fixed times, set at 30 minutes. It would be good to make this time dynamic for the user to better suit their needs. Where medication is necessary to be taken within a stricter time frame, we should provide an option for the user to decrease the time before the emergency contact is notified. A secondary input could also be included in the ‘create medication plan’ page in order to ensure that the correct emergency contact email address has been inputted.

Business Considerations

This device is affordable Costing less than $43 to produce, this device is affordable. The costs of materials could be significantly reduced by ordering hardware components from wholesale or international distributors. 3D printing each structural component is a requirement for the scalability of this product. It would also add structural durability and integrity to the device. Durability could further be increased by soldering hardware circuit connections where applicable.

Given the software was developed with Python and hosted via the AWS cloud server, it is easily scalable and hosting plans are priced depending on the requirements of the application.

Experimental Survey Considerations

We could further assess the device qualitatively through experimental design. Specifically, this could involve conducting a survey on a sample of seniors that used the device. The results from this survey would provide valuable feedback on the usability of the device and prompt further considerations for future revisions. We would need to research the ethics, legalities and safety precautions required to conduct this experiment, however an example of questions that we would consider asking in the survey are included in Appendix D.

Significance of Results

The proportion of the world's population over 60 years old is expected to nearly double from 12% to 22% by 2050 [28]. As a result, there will be an ever increasing need for devices that are cheap, easy to use, and automatically able to remind the user of their medication schedules in order to help them live healthier, happier lives. Monitoring medication adherence in an non-invasive manner is able to provide family members and caregivers peace of mind for the health and safety of seniors that live alone. This is particularly relevant for those who aren't able to physically visit an individual at high health risk, as many of us have recently experienced throughout the COVID-19 pandemic.

REFERENCES

1. H. Marton, “Take medicines seriously,” *healthdirect*, 22-Aug-2018. [Online]. Available: https://www.healthdirect.gov.au/blog/take-medicines-seriously. [Accessed: 15-Oct-2022].
2. R. Lim, L. M. Ellett, S. Semple, and E. E. Roughead, “The extent of medication-related hospital admissions in Australia: A review from 1988 to 2021,” *Drug Safety*, vol. 45, no. 3, pp. 249–257, 2022.
3. R. L. Cutler, F. Fernandez-Llimos, M. Frommer, C. Benrimoj, and V. Garcia-Cardenas, “Economic impact of medication non-adherence by disease groups: A systematic review,” *BMJ Open*, vol. 8, no. 1, 2018.
4. “Health Policy Brief: Patient Engagement,” *Health Affairs*, Feb. 2013.
5. “ESP 8266 based Smart Medicine Box Project using Google Firebase,” *YouTube*, 30-Jan-2022. [Online]. Available: https://www.youtube.com/watch?v=DsUOp-knoMY. [Accessed: 16-Oct-2022].
6. Anmolino and Instructables, “Automatic Pill Dispenser,” *Instructables*, 04-Jan-2019. [Online]. Available: https://www.instructables.com/AUTOMATIC-PILL-DISPENSER/. [Accessed: 16-Oct-2022].
7. “Capsule size guide: Interactive specification chart,” *LFA Capsule Fillers*. [Online]. Available: https://www.lfacapsulefillers.com/capsule-size-chart. [Accessed: 15-Oct-2022].
8. “Comparison: Automated medication dispensers,” *The Senior List*, 17-Jan-2022. [Online]. Available: https://www.theseniorlist.com/medication/dispensers/. [Accessed: 16-Oct-2022].
9. “Jaycar electronics: Components, connectors, switches, power, and more,” *jaycar*. [Online]. Available: https://www.jaycar.com.au/. [Accessed: 16-Oct-2022].
10. “Altronics | Electronic Components, AV, CCTV, Power & More.” [Online]. Available: https://www.altronics.com.au/. [Accessed: 16-Oct-2022].
11. Drugreport, “The 50 most commonly prescribed drugs in America and their average price,” *DrugReport.com - Reporting on Dangerous Drugs, Defective Medical Devices & Harmful Products*, 03-Dec-2020. [Online]. Available: https://www.drugreport.com/50-commonly-prescribed-drugs-in-america/. [Accessed: 16-Oct-2022].
12. UWA Makers, “UWA makers 3D printers: Background information,” *Google Docs*. [Online]. Available: https://docs.google.com/document/d/1ri6oIVJbi4fweyfGjLuVqj2zysziEXirhSPpi1MR2Dg/edit. [Accessed: 16-Oct-2022].
13. Ultimaker3D, “Ultimaker: How to quick start ultimaker cura 3.0,” *YouTube*, 17-Oct-2017. [Online]. Available: https://www.youtube.com/watch?v=2FgljwgF0ec. [Accessed: 16-Oct-2022].
14. “From mind to design in minutes,” *Tinkercad*. [Online]. Available: https://www.tinkercad.com/. [Accessed: 16-Oct-2022].
15. “Ultimaker Cura 5.1 arrives with metal FFF printing, better supports, and improved surface quality!,” *https://ultimaker.com*. [Online]. Available: https://ultimaker.com/learn/ultimaker-cura-5-1-stable-release. [Accessed: 16-Oct-2022].
16. “Python for microcontrollers,” *MicroPython*. [Online]. Available: https://micropython.org/download/esp8266/. [Accessed: 16-Oct-2022].
17. J. T. Cullins, J. Young, S. Santos, J. Clarke, John, Johan, R. Santos, S. Kristensson, H. Kohlsdorf, Martin, Rome, Eiot, B. E, and Bernard, “Install upycraft IDE - windows PC,” *Random Nerd Tutorials*, 02-Apr-2019. [Online]. Available: https://randomnerdtutorials.com/install-upycraft-ide-windows-pc-instructions/. [Accessed: 16-Oct-2022].
18. CodeChamp and Instructables, “Nodemcu with LDR,” *Instructables*, 13-Jul-2017. [Online]. Available: https://www.instructables.com/NodeMCU-With-LDR/. [Accessed: 16-Oct-2022].

19. rahuladitya303, “ESP8266 Servo Controller,” *Arduino Project Hub*, 02-May-2020. [Online].Available:https://create.arduino.cc/projecthub/rahuladitya303/esp8266-servo-controller-3966bc. [Accessed: 16-Oct-2022].
20. *MySQL*. [Online]. Available: https://www.mysql.com/. [Accessed: 16-Oct-2022].
21. “User authentication in Django,” *Django*. [Online]. Available: https://docs.djangoproject.com/en/4.1/topics/auth/. [Accessed: 16-Oct-2022].
22. *Jinja*. [Online]. Available: https://jinja.palletsprojects.com/en/3.1.x/. [Accessed: 16-Oct-2022].
23. “Nginx proxy manager,” *Nginx Proxy Manager*. [Online]. Available: https://nginxproxymanager.com/. [Accessed: 16-Oct-2022].
24. “Fusion 360: 3D CAD, CAM, CAE, & PCB cloud-based software,” *Autodesk*, 12-Oct-2022. [Online]. Available:https://www.autodesk.com/products/fusion-360/overview. [Accessed: 16-Oct-2022].
25. Suda, “Automatic Pill Dispenser by Suda,” *Thingiverse*. [Online]. Available: https://www.thingiverse.com/thing:457321. [Accessed: 16-Oct-2022].
26. R. Astronaut, “Automated Pill Dispenser - finally finished,” *Mellowfire.com*, 10-Sep-2022. [Online]. Available: https://www.mellowfire.com/post/automated-pill-dispenser-finally-finished. [Accessed: 16-Oct-2022].
27. “3D printed joinery: Simplifying Assembly,” *Markforged*. [Online]. Available: https://markforged.com/resources/blog/joinery-onyx. [Accessed: 16-Oct-2022].
28. “Who: Number of people over 60 years set to double by 2050; major societal changes required,” *World Health Organization*. [Online]. Available: https://www.who.int/news/item/30-09-2015-who-number-of-people-over-60-years-set-to-double-by-2050-major-societal-changes-required. [Accessed: 16-Oct-2022].

APPENDICES

Appendix A: Local Deployment Code

The first and second code blocks were run using the Mac OS command line. It should be noted that this code may differ if using another operating system.

1. *Installing pip3, Django, and Python virtual environment*

Install updates for each outdated package and dependency on your system.

```
sudo apt update
sudo apt upgrade
```

Install pip3 (the official package manager and pip command for Python 3)

```
sudo apt install python3-pip
```

install the Django framework

```
pip3 install django
```

install the Python virtual environment

```
sudo apt install python3-venv
```

1. *Deploy code from GitHub*

enter the project root directory

```
git clone git@github.com:Charlie-Hu/djangoturtle.git
```

create a virtual environment in the root directory

```
cd djangoturtle/
```

#enter and activate the virtual environment

```
python3 -m venv venv
source ./venv/bin/activate
```

install the requirements for the virtual environment

```
pip3 install -r requirements.txt
```

1. *Install MySQL*

open the command line in the AWS ec2 linux system

install mysql service

```
sudo apt-get install mysql-server
```

check if the mysql is installed successfully.

```
mysql --version
```

#Change the password of the root user to 123456.

```
sudo mysql
```

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '123456'
If you can enter mysql with the root name and the new password, you successfully change the password.

```
mysql -u root -p
```

Create the database named team_turtle in mysql.

```
create database team_turtle;
```

1. *Run the project*

After installing MySQL and creating the database, you can run the project in the virtual environment.

Enter the virtual environment.

```
cd djangoturtle
source ./venv/bin/activate
```

Database migration

```
python3 manage.py makemigrations app
python3 manage.py migrate
```

Run the server

```
python3 manage.py runserver 127.0.0.1:50005
```

open the HTML page in the static directory with any browser

Appendix B: User Interface Design Process

Rendering URL

When users visit the URL, the front-end renders the home page content stored in the back end.

Login and logout functionality

Determine the request method: if the user clicks on the login button from the main page, or clicks on the login button from the login page without inputting anything in the login form, a GET request is sent which renders the login page. If the front-end form is filled out, then the request method is POST. This triggers the request.POST function, which looks at the data in the front-end input form, and checks the database to see whether this data corresponds to the data of an existing user. The login operation is performed. An error message is displayed if the user details entered in the form aren't found in the database. If the user already exists in the database, the login is successful. The logout method uses the logout method directly and renders the login page.

```
Charlie-Hu +1
def log_in(request):
    if request.method == 'GET':
        return render(request, "login.html")
    username = request.POST["user"]
    password = request.POST["pwd"]
    user = authenticate(username=username, password=password)
    if user is not None:
        login(request, user)
        return redirect('/main/')
    return render(request, 'login.html', {"error_info": "invalid username or password"})


Charlie-Hu
def log_out(request):
    logout(request)
    return render(request, 'login.html')
```

Registration

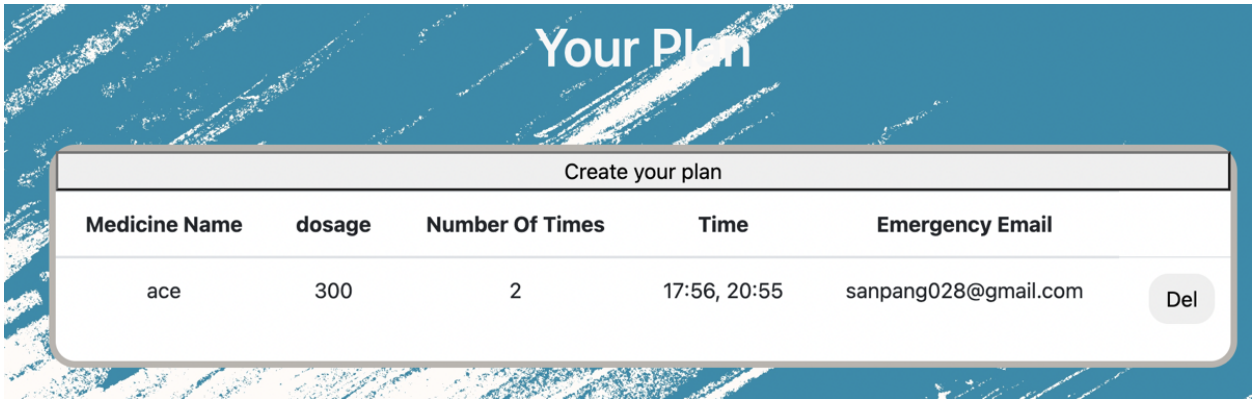
The registration component followed a similar principle to the login component. The major difference is that it needs to determine the data entered by the user. If the validation passes, the data is saved in the database. It then renders the jump page, which displays that the registration was successful. The user is automatically transferred from the jump page to the login page after a few seconds.

```
Charlie-Hu +1
def register(request):
    if request.method == 'GET':
        return render(request, "register.html")
    password = request.POST.get('pwd')
    email = request.POST.get('email')
    username = request.POST.get('user')
    # print(password)
    if len(password) < 1 or len(email) < 1 or len(username) < 1:
        return render(request, "register.html", {'state1': 'input detail can not be empty'})
    elif User.objects.filter(username=username):
        return render(request, "register.html", {'state2': 'user_exist'})
    else:
        new_user = User.objects.create_user(username=username, password=password, email=email)
        new_user.save()
        return render(request, "jump.html")


Charlie-Hu
def jump(request):
    return render(request, "jump.html")
```

```
<script>
    onload = function () {
        setInterval(go, 1000)
    };
    var x = 3;

    function go() {

        if (x >= 0) {
            document.getElementById("sp").innerText = x;
        } else {
            location.href = "/login";
        }
        x--;
    }
</script>
```

Set login permissions for the main page and render the data from the database to the main page.

```
Charlie-Hu +1
@login_required
def main(request):
    if request.method == 'GET':
        user_plan = Userplan.objects.filter(name=request.user.username).values()
        return render(request, "main.html", {"user_plan": user_plan})
```



The delete function removes a plan from the database. This function is triggered when the user clicks on the ‘Del’ button of a specific plan.

```
Charlie-Hu *
@csrf_exempt
@login_required
def plan_delete(request):
    items_to_delete = request.GET.get('id')
    Userplan.objects.filter(id=items_to_delete).delete()
    # print('1')
    return HttpResponse('delete successful')
```

Create Medication Plan

When the create plan button is clicked on the frontend, this create plan function is run. The user input is obtained via the POST request and the current user is obtained via the user form. The user's data inputted in the form is then validated and stored in the database when it passes.

```
Charlie-Hu
@login_required
def plan(request):
    if request.method == 'GET':
        return render(request, "plan.html")
    username = request.user.username
    med_name = request.POST.get("med_name")
    dosage = request.POST.get("dosage")
    times = request.POST.get("times")
    num_time = list()
    for i in range(0, int(times)):
        per_num_time = request.POST.get('num_time{num}'.format(num=i))
        num_time.append(per_num_time)
    em_email = request.POST.get("email")
    print(len(num_time))
    if med_name is None or dosage is None or float(dosage) < 0 or times is None or num_time[0] is None or em_email is None:
        return render(request, "plan.html", {"state1": "input detail can not be empty"})
    new_time = ',  '.join(num_time)
    Userplan.objects.create(name=username, medicine_name=med_name, dosage=dosage, times=times, num_time=new_time,
                            email=em_email)
    return redirect("/main/")
```

Interaction with Hardware

Two interfaces are used to facilitate communication between the hardware and software.

The Django REST framework is used to get the parameters from the URL header that have been passed in by the hardware, and then perform the login verification. If the login verification is successful, the serializer is used to parse the requested data to the URL in JSON format.

```python
class Hardware_View(APIView):

    Charlie-Hu
    def get(self, request, *args, **kwargs):
        queryset = Userplan.objects.all()
        username = request.GET.get('username')
        psw = request.GET.get('password')
        user = authenticate(username=username, password=psw)
        self_email = User.objects.filter(username=user).values("email").first()
        print(psw, username)
        ser = HardwareDataSerializer(instance=queryset, many=True)
        fl_data = dict()
        if user is not None:
            for item in ser.data:
                if username == item.get("name"):
                    if item.get("name") not in fl_data:
                        fl_data[item.get("name")] = list()
                        fl_data["self_email"] = self_email["email"]
                    fl_data[item.get("name")].append(item)

            return Response(fl_data)
        return HttpResponse('invalid username or password')
```

The second interface is an email interface that was built using the native Django send_email function. This function first takes three parameters from the hardware: medicine_status (whether the medicine has been taken), b_email (the emergency contact email), and s_email (the user's email). If the medicine is taken, the user is reminded to take the medicine. If the medication is not taken, the emergency contact is alerted to check the user's status. Tencent email is used as the host email sender, because it is difficult to get email access to Gmail.

Charlie-Hu

```python
@csrf_exempt
def send_email(request):
    medicine_status = request.GET.get("medicine_status")
    print(type(medicine_status))
    em_email = request.GET.get("b_email")
    self_email = request.GET.get("s_email")
    if medicine_status == '1':
        send_mail(
            'Emergency',
            'Please check senior status.',
            settings.EMAIL_HOST_USER,
            [em_email],
            fail_silently=False,
        )
        return HttpResponse('send email')
    elif medicine_status == '0':
        send_mail(
            'Notice',
            'Please take your medication on time.',
            settings.EMAIL_HOST_USER,
            [self_email],
            fail_silently=False,
        )
        return HttpResponse('send email')
    return HttpResponse('unknown status.')
```

Appendix C: ESP8266 Code Sheet and Experiment Setting

Boot.py

Function: setting up WiFi connection and set local RTC with network time protocol

```
import gc
import os
import network
import time
from machine import Pin
led=Pin(2,Pin.OUT)          #create LED object from pin2,Set
Pin2 to output
gc.collect()
ssid = "ROGM16"
password = "wu92387268"
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()                             # Scan for available
access points
print("Connecting to: " + ssid)
sta_if.connect(ssid, password)    # Connect to an AP
while sta_if.isconnected()!=0:
  led.value(1)              #turn off
  time.sleep(0.2)
  led.value(0)              #turn on
  time.sleep(0.2)
if sta_if.isconnected()==0:
  print("Connected to: " + ssid)# Check for successful
connection
  for x in range(5):
    led.value(0)              #turn on
    time.sleep(0.5)
    led.value(1)              #turn off
    time.sleep(0.5)
  #LED blink 5 times indicate that WIFI connected
successfully
import ntptime
def sync_ntp():
    ntptime.NTP_DELTA = 3155644800   # UTC+8
    ntptime.host = 'pool.ntp.org'  # ntp server
default"pool.ntp.org"
    ntptime.settime()
sync_ntp()
```

Main.py
Function init:
Getting and processing schedules from website via http get request.

```
def init(url):
  timeseq,dosage,bEmail,sEmail = getSch(url)

  print("Dosage is: "+str(dosage))

  print(timeseq)

  for times in timeseq:
    print("Timing is: "+str(times))

  mins = []

  for times in timeseq:
    tmp = times.split(":")
    min = int(tmp[0])*60+int(tmp[1])
    mins.append(min)

  print(mins)
  return timeseq,int(dosage),str(bEmail),str(sEmail),mins
```

Dispensing System

Function s1:
Phase 1 for dispensing system

```
#drop pills and sense&email to notice, led:on->off
def s1(dosage,emailUrl):
  motorsp(int(dosage))
  print("motor done")
  setLed(1)
  print("led on")
  readLdr()
  print("ldr done")
  send(0,emailUrl)
  print("email done")
  setLed(0)
  print("led off")
```

Function s2:
Phase 2 for dispensing system

```
#detect pills and send email to notice, led:on
def s2(emailUrl0):
  setLed(1)
  print("led on")
  ldr = readLdr()
  if ldr==1:
    print("ldr done")
    send(0,emailUrl0)
    print("email done")
```

Function s3:
Phase 3 for dispensing system

```
#detect pills and send emergency email to notice, led:on
def s3(emailUrl1):
  setLed(1)
  print("led on")
  ldr = readLdr()
  if ldr==1:
    print("ldr done")
    send(1,emailUrl1)
    print("email done")
```

Function medicationReminder:
Entire script for medication dispensing and reminding system

```
def medicationReminder():
  rtc = RTC()
  while True:
    #gc.collect()
    #scenario testing localtime
    localTime = rtc.datetime()
    print(str(localTime[4])+":"+str(localTime[5]))
    localMins = localTime[4]*60+localTime[5]
    print(localMins)

    timeseq,dosage,bEmail,sEmail,mins = init(url)
    for min in mins:
      print(min)
      print(min - localMins)
      if min - localMins == timeRange:
        print("USER email is:"+sEmail)
        print("Emergency contact email is: "+bEmail)
        #1==pill there check elders 0==pill there plz take
        emailUrl1 = emailPrefix+"1&b_email="+bEmail+"&s_email="+sEmail
        emailUrl0 = emailPrefix+"0&b_email="+bEmail+"&s_email="+sEmail
        #phase 1:
        #  1, dispensing
        #  2, sending email to notice USER to take pills
        s1(dosage,emailUrl0)
        #wait for 60*timerange seconds to next phase
        sleep(60*timeRange)
        #phase 2:
        #  1, sensing whether pills have been took
        #  2, sending email to notice USER to take pills
        s2(emailUrl0)
        #wait for 60*timerange seconds to next phase
        sleep(60*timeRange)
        #phase 3:
        #  1, sensing whether pills have been took
        #  2, sending email to emergency contact to check USER status
        s3(emailUrl1)
        setLed(0)
```

apiGetSch.py

Function: Get schedule from website and process time sequence, dosage for medication, user email address, emergency contact email address

```
emailPrefix = "http://3.26.197.0/send_email/?medicine_status="
api1prefix = "http://3.26.197.0/hardware/?format=json"
username = "test123456"
password = "123456"
url = api1prefix+"&password="+password+"&username="+username

def getSch(url):
  response = urequests.get(url)
  parsed = response.json()
  time = parsed[username][0]['num_time'].replace(" ","")
  timeseq = time.split(",")
  dosage = parsed[username][0]['dosage']
  bEmail = parsed[username][0]['email']
  sEmail = parsed['self_email']
  return timeseq,dosage,bEmail,sEmail
```

Emailesp.py

Function: Sending medication status and email address to website backend to call email sending api.

```
#1==pill there check elders 0==pill there plz take
def send(medicine_status,emailUrl):
  if medicine_status == 1:
    urequests.get(emailUrl)
  if medicine_status == 0:
    urequests.get(emailUrl)
  print("email sent code: "+str(medicine_status))
```

Ldresp.py

Function: Initialise LDR module and collecting illumination value of direct LED light source above the LDR module

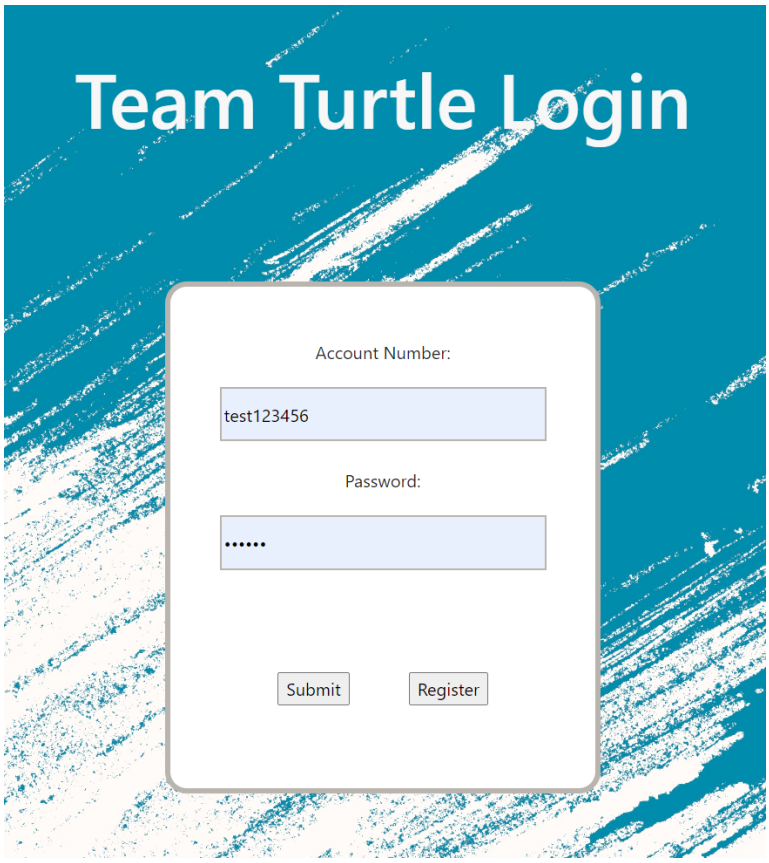
```
from machine import Pin, ADC
from time import sleep

pot = ADC(0)


def readLdr():
  sleep(1)
  pot_value = 0
  pot_value_tt = 0
  for i in range(10):
    pot_value = pot.read()
    print(pot_value)
    pot_value_tt = pot_value_tt + pot_value
    sleep(0.5)

  pot_value_tt = pot_value_tt/10
  print(pot_value_tt)

  if pot_value_tt<50:
    return 1
  else:
    return 0
```

Motoresp.py

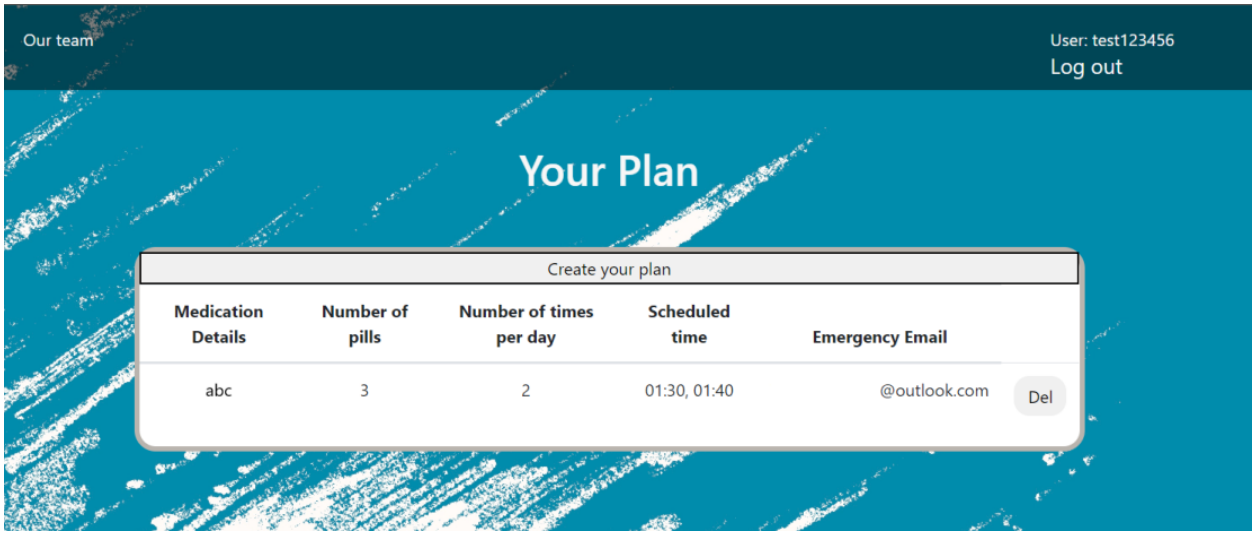
Function: Initialise servo motor module and spin the dispensing disk attach to spinning arm of motor according to dosage to deliver specific pills amount.

```
import machine
from time import sleep
from machine import Pin

# set machine GPIO2 as p2
p2 = machine.Pin(2)
# configure PWM on pin p2
pwm2 = machine.PWM(p2)
# set the PWM frequency as 50Hz
# the frequency must be between 1Hz and 1kHz.
pwm2.freq(50)

def motorsp(dosage):
  for i in range(dosage):

    sleep(1)
    pwm2.duty(50)
    sleep(1)
    pwm2.duty(98)
    sleep(1)
    pwm2.duty(50)
```

Website setting:
Login account: test123456



Medication plan specification:
Medication name: abc
Dosage : 3
Scheduled time: 1:30 AM and 1:40 AM
Emergency contact email: xxxx@outlook.com



Appendix D: Potential Survey Questionnaire

1. Have you experienced taking medication on a daily basis?
2. Have you experienced difficulty taking medication frequently?
3. Have you considered using a smart medication dispenser? Why / why not?
4. Was this device helpful to you? Why / why not?
5. Did you notice the presence of the device in any way?
6. Did you experience any difficulties setting up your medication plan via the website application?
7. Did you experience any difficulties using the device?
8. Do you have any recommendations as to how the web application may be easier for you to use?
9. Do you have any recommendations as to how the device may be easier for you to use?