Smart Mirror

Student name: *Andy Vuong, Daniel Sarni, Erick Hernandez, Erik Szayna, Evan Luu, Serjo Barron*

Course: *CSE123A*
Due date: *June 13, 2024*

1 Executive Summary

1.1 Project Description:

A hands-free mirror that provides relevant information to consumers during their morning routines without the need to use their phones.

1.2 Target Audience:

Busy professionals or students seeking a way to stay digitally connected while doing their morning routines without the need to grab their phones (and experience the negative health effects) or other devices and stay hands-free.

1.3 Design Objectives:

1. Modularized Information Windows: Show the user the information they seek in modules
    - Integrate API calls, such as Google, for modules such as calendars, weather, etc.
    - Display the modules on the Mirror and make it configurable within the app.
2. User-Friendly Mobile App: Create a mobile app that can be accessed by any mobile device for setup before mirror usage.
    - Allow the app to edit the specific modules so that the users can edit their mirror configurations to their liking.
    - Allow the app to help enroll verification for users.
3. Straightforward Verification: Allow the users to be able to easily add their specific verification to access their mirror.
    - Allow the user to seamlessly verify themselves with their mirror.
    - Load the user's profile after they verify themselves.
    - Implement a way for users to register their fingerprints once and directly verify on the mirror itself after registering.

1.4 Ethics Statement

Statement for Ethics: Our product and team will provide services ethically and honestly, prioritizing our customers' safety. Our product will only be used in the way we advertise it to be used and any further updates will immediately be communicated truthfully and objectively. We believe that conducting ourselves in an honorable, responsible, ethical, and lawful manner is critical to the success and honor of our product. We understand

2 Table of Contents

Contents

3 Introduction

3.1 Need and Goal Statements

1. **Problem**: Every morning, the first thing many people reach for is their phone to access crucial information such as weather, texts, and time, which can lead to an inefficient morning routine and negatively impact mental health. Our specific targets are as follows:

    - Those who want to use their phones less in the morning.
    - Those who want a more seamless method of catching up on data during their morning routines without needing to use their hands to get such data.
    - Those who want a productive start to their mornings, free from their phone's distractions or attention traps.

2. **Need**:

    - We need a product that can seamlessly integrate into a morning routine, efficiently and digitally provide desired up-to-date information, and does not negatively impact the user's time and mental health, and is mostly hands-free.

3. **Goal**:

    - Our Smart Mirror should seamlessly integrate into a morning routine, efficiently and digitally provide desired up-to-date information, positively impact the user's time and mental health, and be mostly hands-free.

3.2 Personas

1. **Persona 1:**

    (a) **Name:** Peter Cox

    (b) **Age:** 23

    (c) **Job:** Software Developer

    (d) **Lifestyle:** Young professional part of a fast-paced tech company who lives alone.

    (e) **Bio:** Peter is always on the move, and his mornings are rushed due to work deadlines and other commitments. To leave for work on time, he wants his morning routines to be more efficient.

2. **Persona 2:**

    (a) **Name:** Erica Finger

    (b) **Age:** 28

    (c) **Job:** Marketing Specialist

    (d) **Lifestyle:** Lives with boyfriend in an urban neighborhood, works demanding jobs and is passionate about fitness & health.

(e) **Bio:** Erica's mornings involve preparing for work and workout routines. She wants to be able to see the news and her calendar but wants a method to not need to immediately stare at her phone in the morning.

3. **Persona 3:**

   (a) **Name:** Skylar Sky

   (b) **Age:** 27

   (c) **Job:** Graphic Designer

   (d) **Lifestyle:** Lives in a shared apartment with a roommate. Enjoys an artistic lifestyle and balanced morning routine.

   (e) **Bio:** Likes to plan her day while leaving room for hobbies, while also staying connected to what's going on in the world. She wants to be able to keep up to date but doesn't like using her phone. She wants more personalization with her morning routine without the need for her mobile device.

4. **Persona 4:**

   (a) **Name:** John Bob

   (b) **Age:** 20

   (c) **Job:** College Student

   (d) **Lifestyle:** Lives in an off-campus apartment in a single room. Enjoys watching TikTok.

   (e) **Bio:** As a student, John is busy with his studies. However, his classes tend to be later in the afternoon so he ends up staying in bed for a while after he wakes up as he tends to immediately open TikTok on wakeup. He wants his mornings to be productive and wants it to be distraction-free so he can spend mornings not on his phone in bed.

3.3 Research into existing designs and/or products that meet / partially meet the need

Other available mirrors are too expensive or have a lot of unnecessary features. Our mirror is much cheaper, just as simple to use, and does not have features that detract from the goal: nondistracting access to desired information.

3.4 Sustainability Statement

Our Smart Mirror is committed to sustainability and environmental responsibility. While we acknowledge that not all components used in our product are directly renewable, we strive to incorporate as many sustainable practices as possible. The two-way glass (acrylic panel) is a renewable component and we will continue to explore ways to increase the renewable content of our product's used materials. In terms of disposability, we have selected components with a high lifespan that are repairable, thus reducing the need for frequent replacements. The monitor display, Raspberry Pi 3, and two-way glass are all recyclable, ensuring

that at the end of their lifespan, they can still be disposed of in an environmentally friendly manner. We will furthermore encourage our customers to recycle these components appropriately to minimize environmental impact.

4 Design

4.1 Physical Design

To showcase our Physical design, we created a CAD model. It was created in Blender. It's almost exactly what our actual design ended up being since there are not many ways you can change up the shape of the mirror. The only difference is the inclusion of a fingerprint scanner in our final product, along with the Raspberry Pi on the back of the mirror itself.

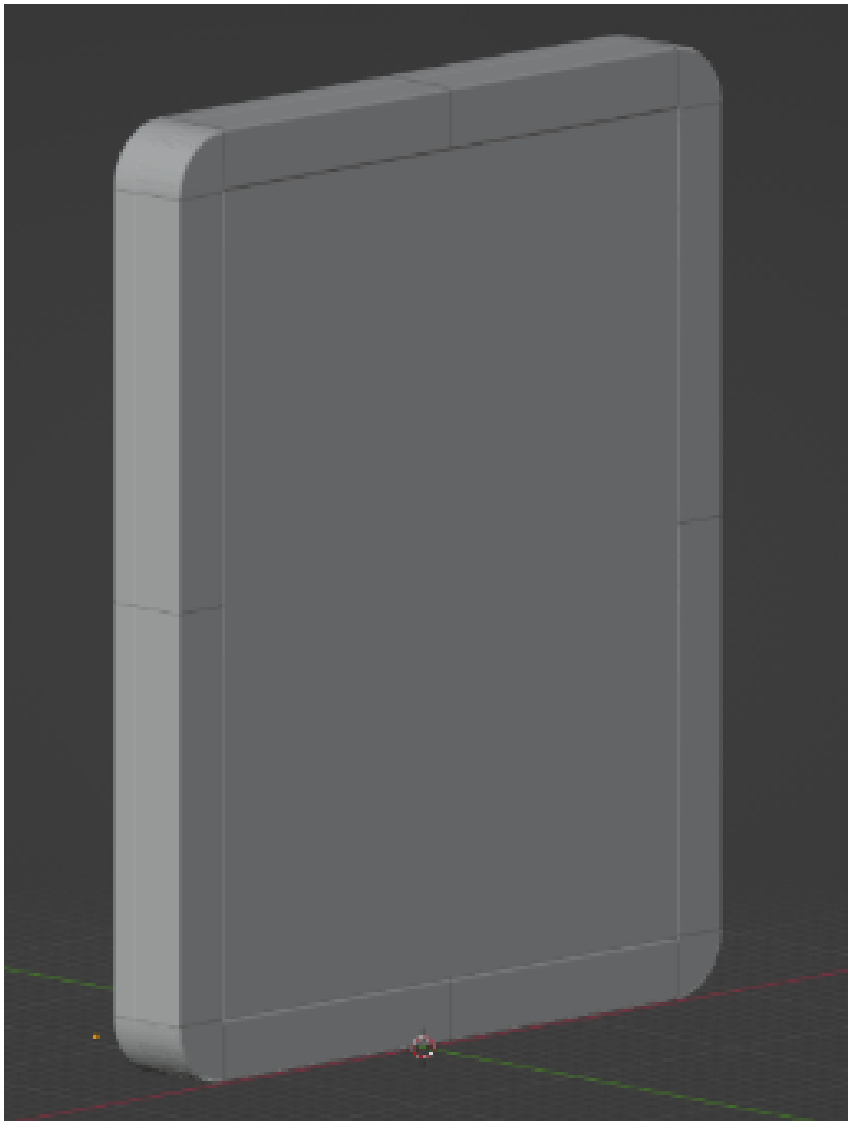


Figure 1: CAD model of our physical design.

4.2 Design for Manufacture, Assembly, and Maintenance

4.2.1 Manufacture

1. **Computer to Run Software**:
    - **Material**: Raspberry Pi
    - **Source**: For the final product, we will buy from vendors. For the prototype, we had one in our inventory to be able to use for this project.
2. **Fingerprint Scanner**:
    - **Material**: Adafruit Fingerprint Scanner
    - **Source**: Purchased directly from Adafruit themselves.
3. **Speaker**:
    - **Material**: Conduction Speaker
    - **Source**: Purchased directly from DigiKey.
4. **Display**:
    - **Material**: Monitor Display
    - **Source**: For our final product, we would provide a thin display, purchased from a vendor. For our prototype, we acquired one from an electronic waste station.
5. **Frame**:
    - **Material**: Mirror Frame
    - **Source**: For the final product, it would be a wooden frame where the wood would be purchased and we would create the frame itself in production. For prototype, it was a cardboard frame.
6. **Reflective Product**:
    - **Material**: Mirror Glass
    - **Source**: For the final product, it would be a one-way reflective glass mirror that would be able to shine the display's content but also reflect whatever is in front of the mirror, purchased by a vendor. For prototype, was a reflective acrylic piece.
7. **Mounting Equipment**:
    - **Material**: Mirror Adhesive
    - **Source**: This is to mount the mirror to the wall, the source would be from vendors that sell the adhesive.

4.2.2 Assembly

The assembly of our project is very straightforward. The usage of a wooden frame allows it to be durable but not heavy, allowing for easy transportation and installation. The choice of a thin display was to not only save space but also allow for a thinner mirror design that would fit in most spots. The choice of reflective glass is similar to the display but it ensures that there are no wrinkles on the actual display itself compared to acrylic or reflective tape. The choices also protect structural integrity, if a user accidentally knocks over their mirror, the wooden frame is strong enough to absorb most of the impact and not break apart.

While the final product will consist of a wooden frame, thin display, and thin glass, our prototype will consist of a cardboard frame, a computer monitor, and a reflective acrylic sheet. The decision for our prototype to use this was to ensure we can create this project at a cheap price, as well as store the project in the lab without fear of damage. We also believed that this did not ultimately result in any major difference in the form or function of the final product.

For the final product, users would not need to worry about assembling anything out of the box as we plan on shipping the product fully assembled. All users would need to do is mount the mirror onto their wall using the provided adhesive. Users would apply the adhesive to the back of the mirror and stick it to the wall.

4.2.3 Maintenance

A goal of our Smart Mirror is that it requires minimal maintenance. It is designed to not require regular maintenance in any capacity. If any part of it is damaged, it is possible to fix or replace that part and reassemble the product together. The wooden frame of our Smart Mirror will be covered in a polyurethane finish to protect the wooden frame in the moist and humid environment of a bathroom. If the Raspberry Pi or the fingerprint scanner experiences any difficulties, the Raspberry Pi can just be restarted.

4.3 Electron App and Configuration Files

The main electron app is run off the open-source smart mirror platform, Magic Mirror[2]. It was used as the main framework for the Electron app itself, with our logic built into it. It doesn't natively support profiles, so we added our functionality to allow for the creation of users, and to allow them to be swapped around as needed.

This was done by creating a new file, 'userId.js', wherein the current user and all existing users are stored. If userId is ever edited, the electron.js app will refresh the app, thus loading in the new user's respective configuration file. Essentially, the electron.js file will first check if the currentUser variable in userId exists - meaning they're currently already in that same file array of existingUsers, then a new file will be created. This file will be named after the user, so 'user1.js' for example, should the userId be user1. Afterwards, the default template of the mirror will be pasted into their configuration file, and the electron app will refresh itself and now load in that user. They will then be added to the existingUsers array, ensuring a new file for them is not created in the future. The electron app will also watch the currentUsers respective configuration files, so if they ever make a change, it'll immediately refresh the app to show this change.

4.4 Fingerprint Scanner

The fingerprint scanner we used was Adafruit's Ultraslim Fingerprint Scanner. The fingerprint sensor's main objective was to act as the verification method of the mirror. As per Adafruit's website, the sensor itself can store up to 150 fingerprints.

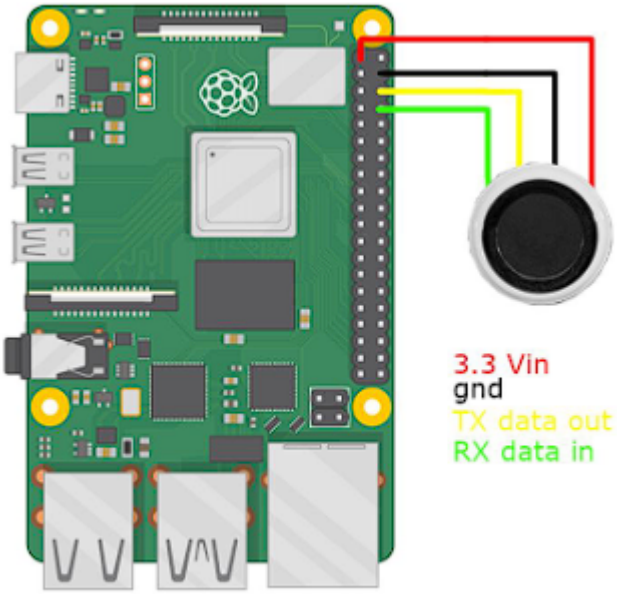


Figure 2: Wiring Diagram of Fingerprint Scanner

Above is the wiring of the fingerprint scanner. It was powered by the GPIO pins of the Raspberry Pi (specifically the Vin was a 3.3V pin, ground to GND pin, and communication was the TX and RX pins). We chose Python for the program to enroll, verify, and store our fingerprints as it was the language with the most support from Adafruit.

From there we were able to communicate directly to the Electron App.

4.5 Custom BLE Profile and Fields

4.5.1 Overview

For communication between the mirror and the mobile app, we created a custom BLE service encapsulating multiple characteristics. An overview of this can be seen in Figure 3. Additional justification for the design of this custom service can be found in section 6.5.1.

4.5.2 Module Data Fields and Serialization

The information transmitted over Bluetooth is the contents of the configuration file for one user, as described in section 4.3. This includes position and enable information for each of the user's apps. In the user configuration JSON files, each position and enable/disable value is stored as a string. On the BLE central and peripheral side, these values are converted to a corresponding integer, and then converted back upon being received.

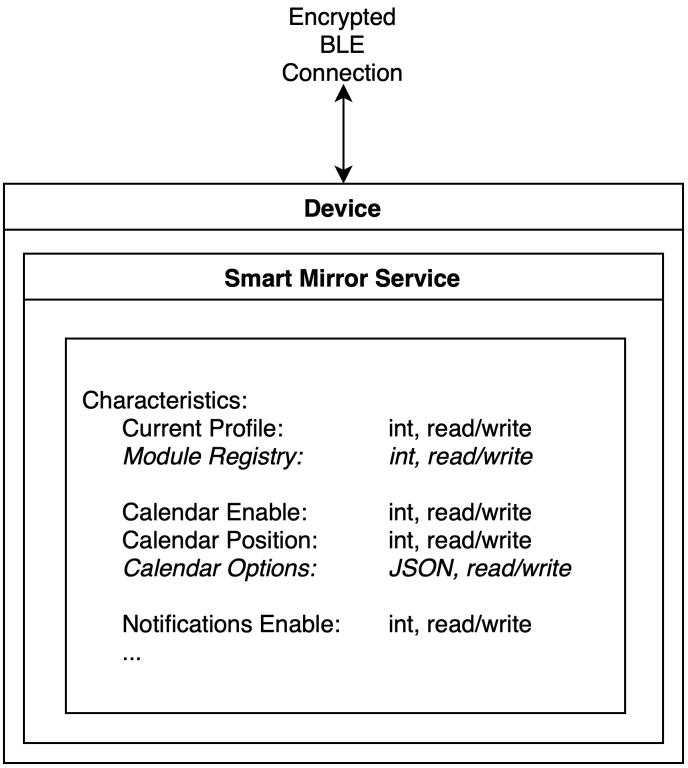


Figure 3: Overview of the custom BLE service for interacting with the mirror. Italicized items are not currently implemented in the functional prototype.

The exception to this integer-only data transmission is any additional app-specific information (like authentication information, weather location, etc). That is transmitted as JSON serialized to an integer array.

4.5.3 User and Modules List Characteristics

In addition to the app-specific characteristics, there is a characteristic that displays which user is currently signed in to the mirror. As a backup to the fingerprint scanner, this characteristic can be written to (via a page in the app UI), to manually change the current logged-in user on the mirror. Additionally, this characteristic can be read via the app, to display a user-specific mirror configuration to the app user.

There is also a module registry characteristic, which holds information about which modules are available on the mirror. This is a series of flags corresponding to which apps are available on the mirror.

4.5.4 Pairing and Security

To pair a phone to the mirror, if there is no phone currently connected, the mirror advertises via BLE GAP. Any phone can initiate a connection to the mirror at this time. Upon a BLE central attempting to pair to the mirror, a popup with a code will appear on the mirror side and phone side. Both sides will need to acknowledge with an "OK" (a fingerprint press on the mirror side) for the bonding to be successful.

Once a phone is bonded to the mirror, it is assumed to be a trusted user. From this point, any bonded phone app can modify any user configuration on the user side. BLE data communication over GATT is encrypted, via the underlying BLE stacks on the phone side and the Linux side.

4.6 BLE Central (Mobile Phone App)

4.6.1 Overview

The companion mobile app is used to configure a smart mirror's settings. The app is not intended for daily use, but just for setup and configuration tasks. Specifically, the user is able to hide and show apps, move apps around, provide authentication to third-party apps (Google Calendar, etc), and change the system settings (language, units) of the mirror.

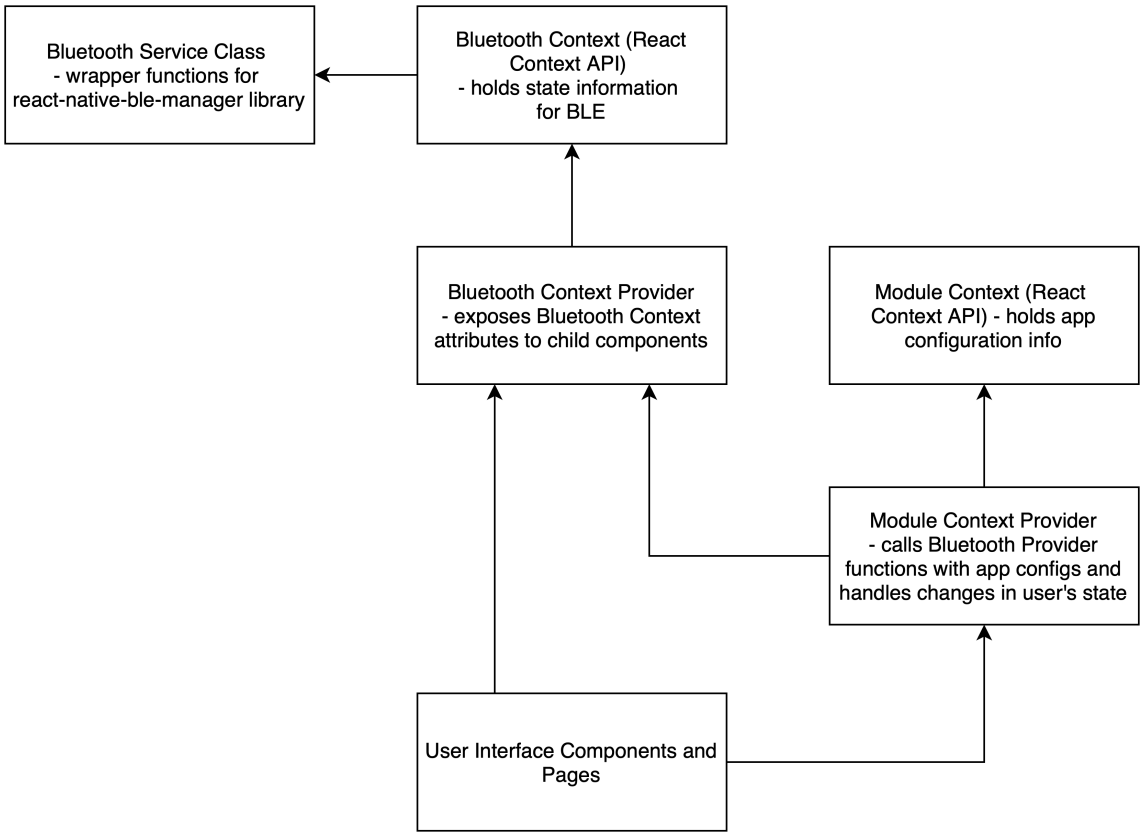


Figure 4: Overview of the React Native mobile app used to configure the mirror.

The mobile app is built with React Native, Typescript, and the react-native-ble-manager library. An overview of the structure of the app can be seen in Figure 4. The full rationale behind these technology choices can be found in the appendix, section 6.5.2.

4.6.2 State Management

To manage the state across the app, the React Context API and context providers were used. The top-level encapsulating component is a Bluetooth context provider, which provides functionality for connecting, reading, and writing to a device, as well as holding onto UUIDs. The next child component is a "module" context provider, which holds onto a user's preferred mirror state (window positions, etc). Finally, as a child of both those context providers, are the individual screens.

4.6.3 Pairing

To bond to the mirror device, the user uses their phone's system settings to initiate the bonding. From there, they press a "connect" button in the app (implemented as 4 sequential buttons to press in the prototype, instead of 1). This "connection" step encompasses getting list of bonded devices back from the phone's BLE history, connecting to it (if it has become disconnected), and populating information from it into the BLE context.

In the BLE context, there is a mapping of app names and attributes (ex. clock enable) to characteristic UUIDs. This map is populated during the "get services" step of connecting, based on the descriptions attached to each characteristic.

4.6.4 Writing to the Mirror

On the app side, an update of the user's configuration (of window positions, etc) gets written to whichever current user is logged in on the mirror. See justification in 6.5.1 for why the phone is trusted to write to any user.

Inside of module context is a field for which the user is currently logged in. Upon reading the "current user" characteristic or setting it manually, the corresponding field is updated in module context. Based on that field, one of the multiple saved configurations is displayed to the user in the configuration form.

4.7 BLE Peripheral

4.7.1 Backend Implementation - Python

Python was chosen due to its ease of use, readability, and extensive libraries available for development use. It allowed rapid development and prototyping while also having strong community support making it easier to troubleshoot any BLE issues.

4.7.2 Bluetooth Stack and Communication

BlueZ is the official Linux Bluetooth stack and was used due to its role in managing Bluetooth devices and handling low-level operations. It provides support for core Bluetooth layers and protocols which allows to create/manage GATT services and characteristics. Overall BlueZ was used to control Bluetooth devices, manage connections, and perform any required BLE operations.

D-Bus is an inter-process communication (IPC) that was also used to enable communication between processes running concurrently on the same machine (the Raspberry Pi). D-Bus is used to interact with BlueZ to facilitate communication between the Python script

(peripheral) and Bluetooth stack. This also allows for the peripheral to send commands to BlueZ to handle any asynchronous events like device connections and/or disconnections.

4.7.3 GATT Application

GATT defines how 2 BLE devices transfer data using services and characteristics. In our implementation, the peripheral defines these services and characteristics which is what the client will be able to interact with. Each GATT service is identified by a unique service UUID, and each characteristic within a service has its own characteristic UUID.

The implementation involves defining methods ReadValue and WriteValue for each modifiable characteristic. These methods handle the data transfer between the peripheral and the client device (BLE central).

4.7.4 BLE Advertising

BLE advertising is a process by which a BLE peripheral broadcasts data packets to nearby devices, indicating its presence and capabilities. In our implementation, advertising is crucial for making the peripheral discoverable to client devices. As for the data included in advertising (advertisement data), it includes the device name and available services.

4.7.5 Overall Implementation

1. Initialize D-Bus connection and set up BlueZ service interface.

2. Define GATT services and characteristics alongside their respective ReadValue and WriteValue methods, allowing the characteristics data to be modifiable.

3. Register the GATT application with BlueZ, configure the advertising data, and then start BLE advertising to make the peripheral discoverable to other devices (the BLE central on a mobile device).

4. Once connected to connection and disconnection events are handled as well as any read/write requests to characteristics received by the peripheral.

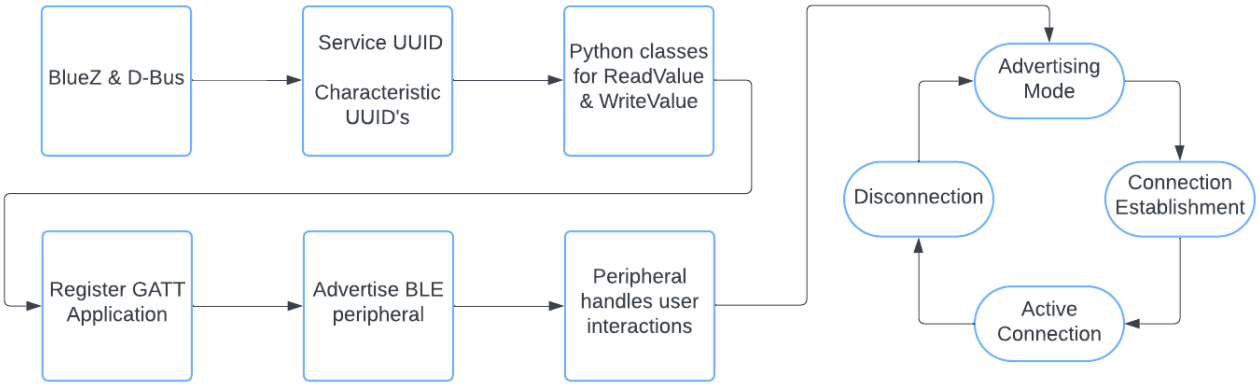


Figure 5: Running BLE Peripheral Script Flowchart

5 Evaluation

5.1 Functional Prototype

5.1.1 Components

We had a multitude of components for our functional prototype. We had a 24" 1920x1200 monitor in order to display all of our content to the user, and as a foundation to hold everything together. Next, we had the reflective, mirror-like finish acrylic piece that would rest flush against the display monitor. For our computer, we used a Raspberry Pi 3 for running our mirror interface as well as our back-end processes. Additionally, we had a fingerprint sensor and audio exciter.

5.1.2 Hardware

For the hardware of this project, we used the display as a foundation to mount all of our other hardware pieces together. We first had to cut the acrylic piece to shape the size of the monitor. After we cut the acrylic, we mounted it to the monitor using adhesive tape on the rim to give it a frame-less look. After, we mounted the Raspberry Pi to the back of the monitor to hide it from the front. We did this because we were unable to print our 3d frame as we originally planned. The 3d printed frame would have been responsible for housing the components safely.

After we mounted the Pi computer to the monitor using more adhesive tape, we connected the audio exciter to the Pi computer and taped the audio exciter to the back center of the monitor so that it would vibrate the entire chassis of the monitor to create audio that the user would be able to hear, such as when playing music, listening to a podcast, or listening to the news. Additionally, we soldered a fingerprint sensor to the GPIO pins of the pi, specifically the 5V, and Ground. We then mounted the sensor to the side of the monitor since it was the easiest method for the user to interact with it.

5.1.3 Software

For our front-end of the display, we had the open source software MagicMirror be our launch-pad for our custom modules. This was responsible to display modules to the user and respond to user input. On the back-end, the software was running our custom modules that allows for easy profile switching whenever a registered user authenticates themselves using the fingerprint sensor. We modified the provided python code that uses Adafruit's APIs to interface with the fingerprint sensor so that it would always scan for fingerprints. If it found a user, it will print to stdout the user it found, or any other errors it encounters. Our electron nodejs module would then read and parse this output using regex and react accordingly.

For instance, if a user was found, the module would load the user configuration stored in an array of all registered users, and load said profile. If the user is not found twice in a row, it is assumed the user would like to register as a new user so that the modules would be the onboard and enrollment process for a new fingerprint.

Figure 6: Electron app default showcase on the mirror. Greets the user and showcases the time, news, and weather of their location.

Figure 7: Fingerprint scanner next to the mirror. Used to swap between profiles by touching your finger on it.

5.1.4 Mobile App

Our mobile app handles the configuration of the mirror. Specifically, since the mirror's only way of accepting feedback is the fingerprint scanner, we needed a way to move around windows and hide/show apps as needed.

If a phone is not connected to the mirror, the mirror is always BLE advertising. The user is able to connect to the mirror via the app and get information about which user is currently logged in. This can be seen in Figures 8 and 9.

Upon connecting, the user can update their window positions and hide/show apps. Upon the user pressing "send", this is updated in real-time on the mirror. This can be seen in Figure 10.

The entire process of app connection and module configuration can also be seen in this video recording: https://youtu.be/XPClvklzuRY?si=BpTFww$_5$48$qMrg2v$

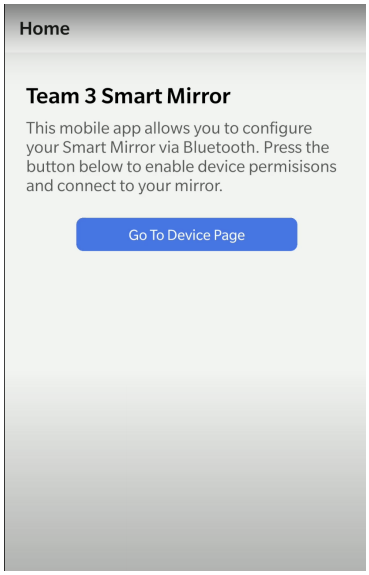


Figure 8: App home screen. Upon navigating to the next screen from this page, any required Bluetooth and location permissions are triggered.

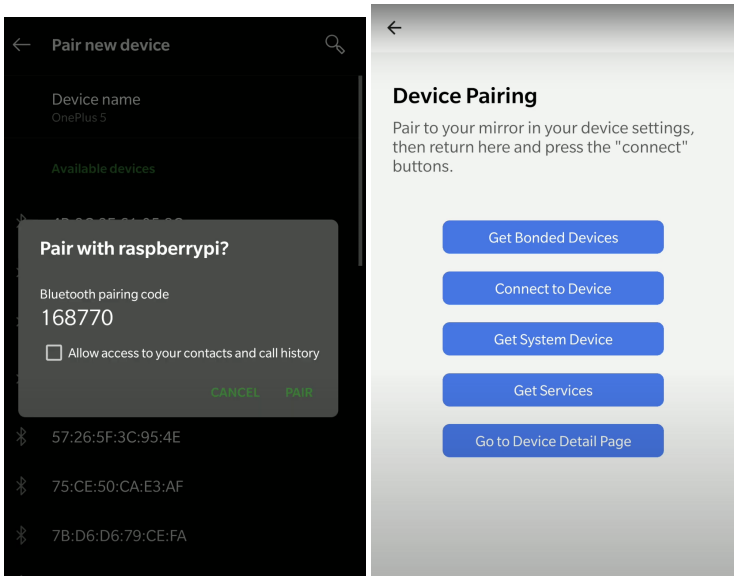


Figure 9: App Mirror Pairing. After the device is BLE bonded in the phone's settings, the connecting is completed by pressing the first 4 buttons sequentially.

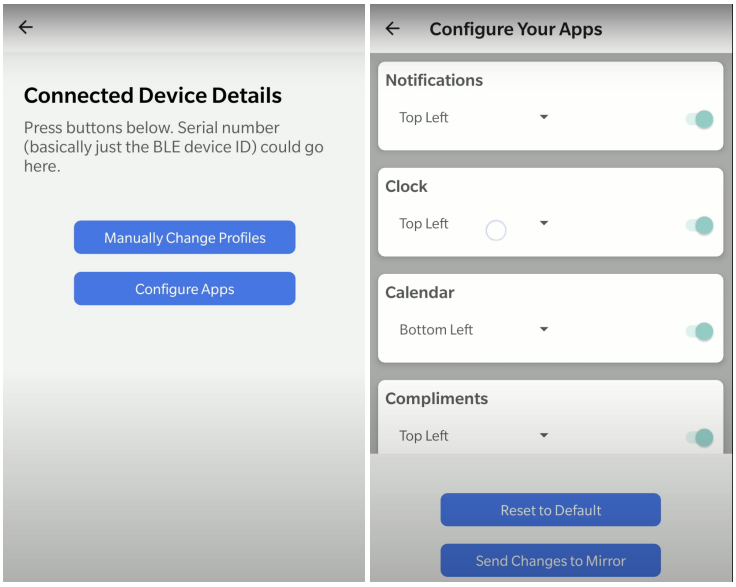


Figure 10: App Mirror Configuration Screens. A page to optionally manually switch profiles is available, as well as the main module configuration screen.

5.2 Testing

The objective of our testing was to measure aspects of the prototype's functionality in various conditions. Our full test plan and results can be viewed in the appendix, section 6.3.1.

We measured the visibility of the screen and the mirror at set distances of two feet, four feet, and eight feet. We repeated this visibility test with both a brightly lit environment and a dark environment. The screen and mirror had good visibility throughout the tests, proving its usability in normal household environments.

We measured the reliability of the fingerprint scanner by measuring its response time and success rate in different finger placement positions. We had five different finger placement angles: flat, up-, down-, left-, and right-leaning. The tests had response times of 1.1 seconds on average, which we were satisfied with. There was also great success in the different finger placement angles.

We also measured the time taken by the Smart Mirror to fetch data based on the module, the number of modules, and small and large changes. The time taken is consistently within 50ms, which is very good.

6 Appendices

6.1 Appendix 1 - Problem Formulation

6.1.1 Conceptualisations

1. **Mind Map**:

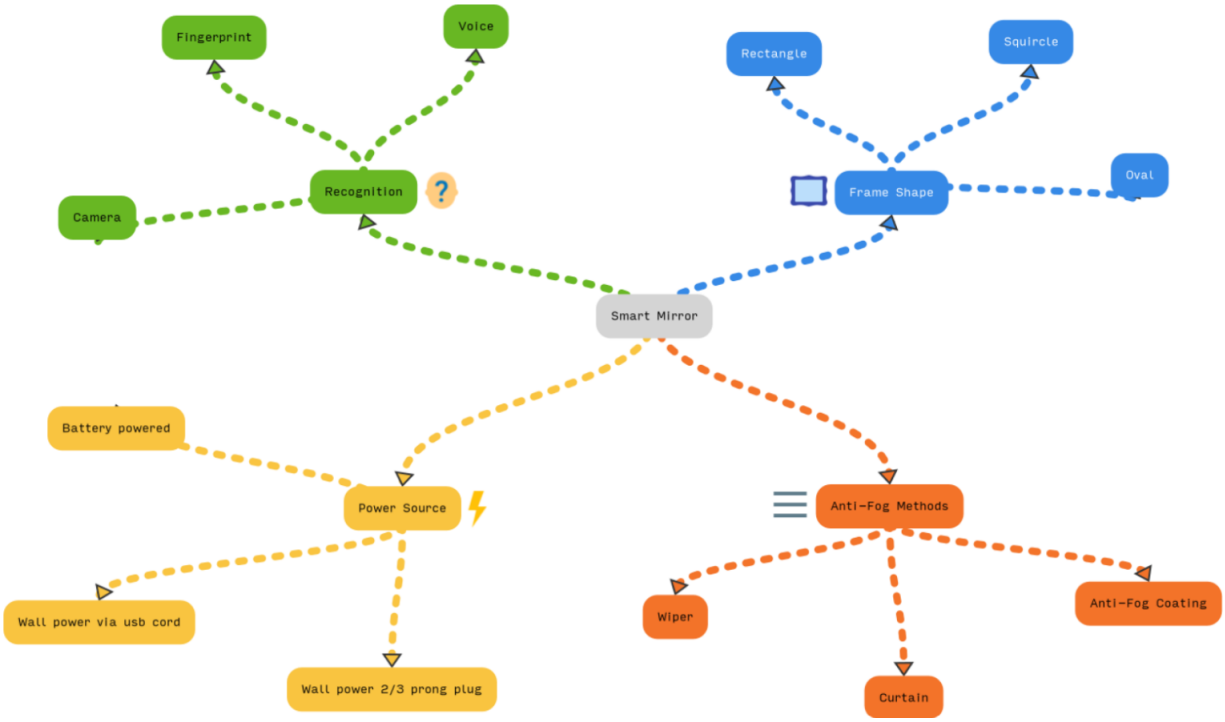


Figure 11: Mind Map

6.1.2 Brainstorming

1. **Touchless Recognition:**
   - Ensure users can keep up to date with relevant information while doing other things, such as doing their makeup or morning routine
2. **Anti Fog:**
   - Prevent the mirror from fogging up if, for example, a user is showering and the room is steamy
3. **Multiple User Support:**
   - Allows multiple family households to have different profiles for each member so each member can edit their profiles to their liking
4. **Efficient Power Usage:**
   - Allows a cost-efficient power method for users to be able to use their mirror whenever they need without using up a lot of power
5. **Organized Display:**
   - Information a user wants to see is nicely organized
6. **Bluetooth Speaker:**
   - Play music, alarms, or sound cues

6.1.3 Decision Tables

1. **Design 1 Description:** wall power via USB; uses wiper for anti-fog; vertical rectangle; voice commands to control.
2. **Design 2 Description:** wall power via $\frac{2}{3}$ cord; uses film for anti-fog; horizontal rectangle; camera recognition.
3. **Design 3 Description:** battery power; curtain for anti-fogging; square mirror; fingerprint scanner.

Criteria:	**Design 1:**	**Design 2:**	**Design 3:**
Cost:	Balanced	Cheap	Balanced
Ease of setup:	Easy	Hard	Easy
Reliability:	Balanced	Least	Most
Lifespan:	Fair	Fair	Poor
Recyclability:	Medium	High	Low

Table 1: Decision Table

Cost Evaluation Scale:	
Dollar Amount:	**Points:**
>300	1
200-300	2
100-200	3
50-100	4
<50	5

Table 2: Cost Evaluation Scale

Ease of Setup Scale:	
Time Taken for Setup (minutes):	**Points:**
>60	1
30-60	2
15-30	3
5-15	4
<5	5

Table 3: Ease of Setup Scale

Reliability Scale:	
Time Taken before Action Retry Needed (minutes):	**Points:**
<1	1
1-5	2
5-15	3
15-45	4
>45	5

Table 4: Reliability Scale

Lifespan Scale:	
Time Taken before Repair Needed (months):	**Points:**
<1	1
1-2	2
3-4	3
5	4
>6	5

Table 5: Lifespan Scale

Recyclability Scale:	
Cost Required to Recycle (USD):	**Points:**
<10	1
10-20	2
20-50	3
50-100	4
>100	5

Table 6: Recyclability Scale

Criteria:	**Weight:**	**Design 1 Raw:**	**Design 1 Weighted:**	**Design 2 Raw:**	**Design 2 Weighted:**	**Design 3 Raw:**	**Design 3 Weighted:**
Cost:	40	3	120	2	80	3	120
Ease of Setup:	15	4	60	3	45	5	75
Reliability:	20	3	60	3	60	4	80
Lifespan:	15	3	45	3	45	2	30
Recyclability:	10	3	30	3	30	1	10
Total:	100	16	315	14	260	15	315

Table 7: Full Decision Table

6.1.4 Morphological Charts

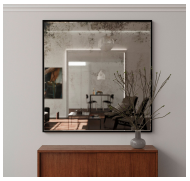
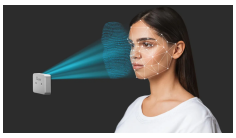
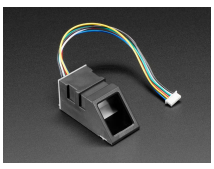
	Option 1:	**Option 2:**	**Option 3:**	**Option 4:**
Power Source:	 Wall power via 2/3 power cord	 Wall power via USB-cord	 Batteries	 Battery and solar
Anti-fog mirror methods:	 Mirror wiper	 Curtain	 Film	 Heated Mirror
Mirror & frame shape	 Horizontal Rectangle	 Vertical Rectangle	 Oval	 Square
Recognition	 Camera recognition	 Fingerprint sensor	 Voice activation	 Bluetooth

Table 8: Morphological Chart

6.2 Appendix 2 - Planning

6.2.1 Basic Plan/Gantt Chart

1. **Project Overview**:

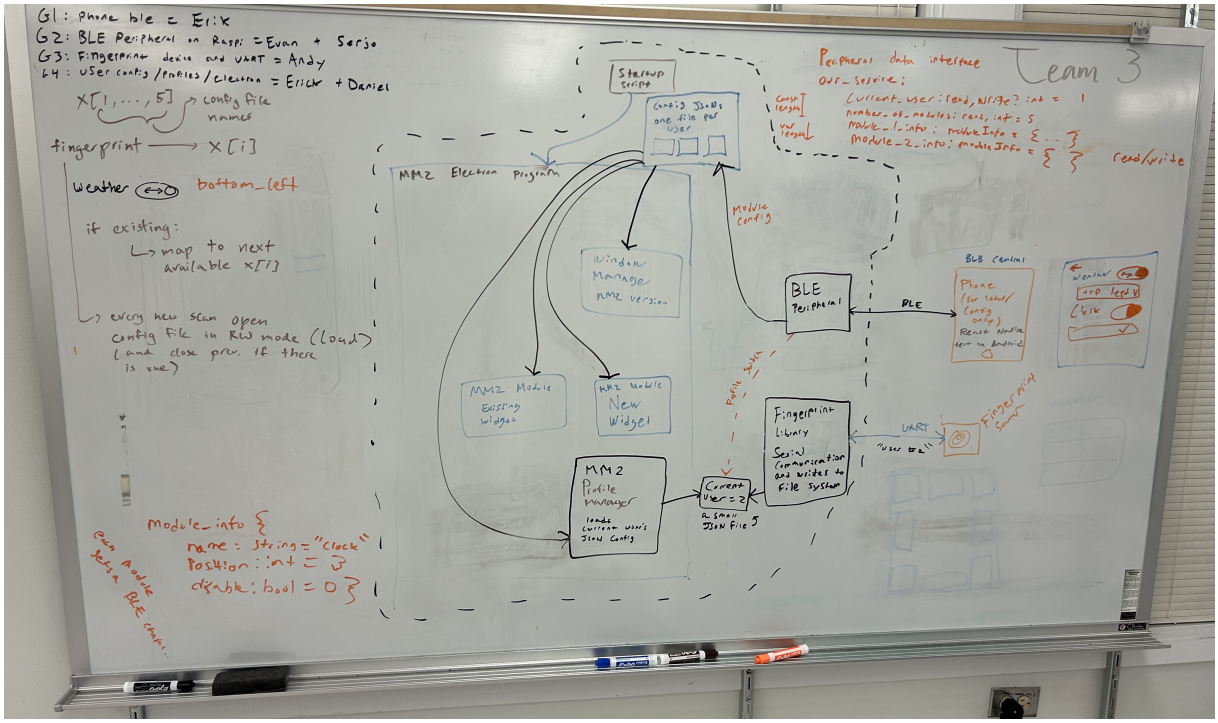


Figure 12: Whiteboard Design Planning

1. **Gantt Chart**

   - Here is is a link to the Gantt chart due to poor visibility.

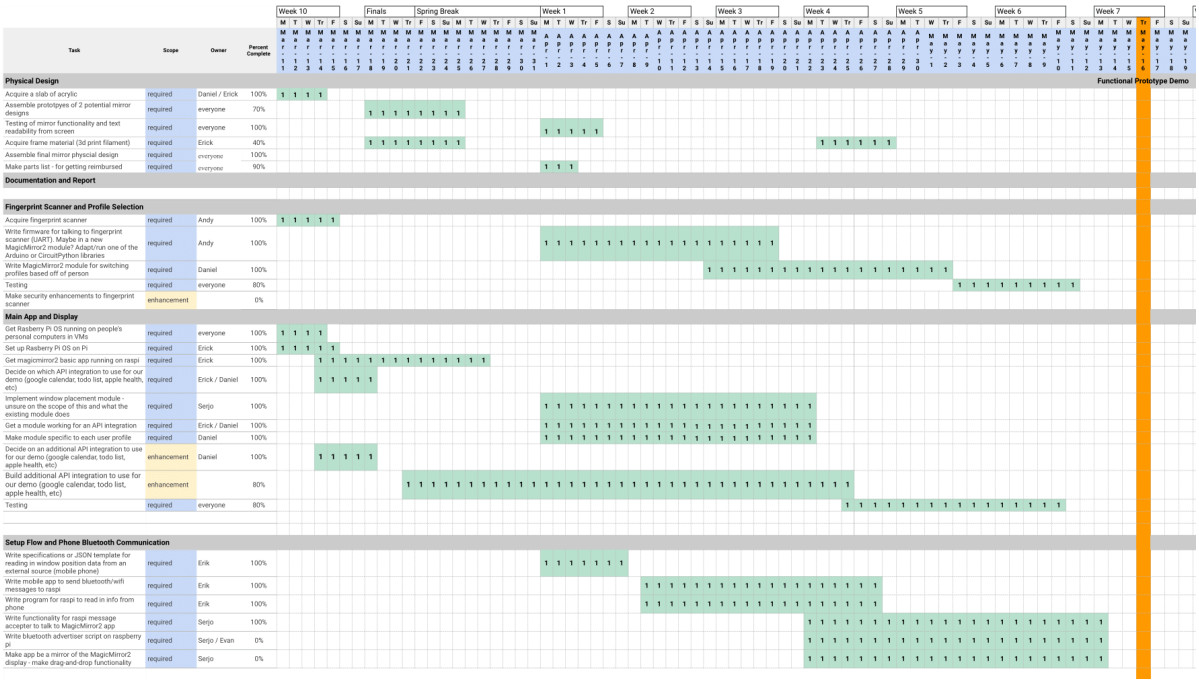


Figure 13: Gantt Chart

6.2.2 CPM & PERT Analysis

1. **CPM Chart**:

Figure 14: CPM Chart

1. **PERT Analysis**:

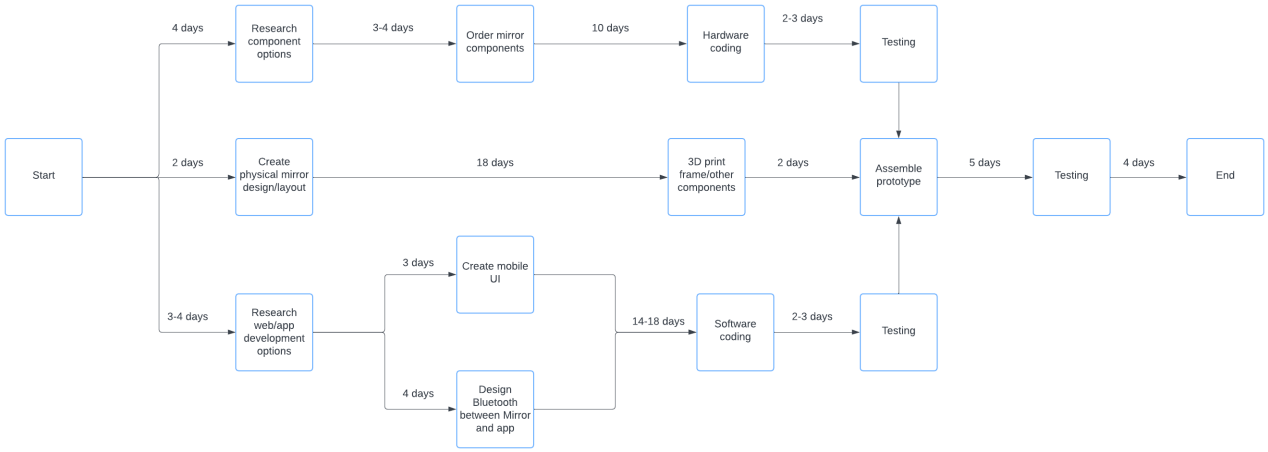


Figure 15: PERT Analysis Chart

6.2.3 Division of labor during prototyping phase

Our team was split into three major subteams: Fingerprint, Electron, and Bluetooth. Each team was delegated a task related to their expertise.

1. Fingerprint:
    - The fingerprint team was comprised of Andy, and his job was to set up the fingerprint sensor (with the Raspberry Pi via wiring) and create code for it to properly enroll and verify fingerprints.
2. Electron:
    - The Electron app team was comprised of Daniel and Erick, their jobs were to set up the Electron app as well as to write new modules to be incorporated with the actual mirror UI.
3. Bluetooth (BLE):
    - The Bluetooth team was comprised of Erik, Serjo, and Evan. Erik made the mobile app (BLE central) which would be used to update the mirror's configuration over BLE. Serjo and Evan created the BLE peripheral which served as a gateway point for the BLE peripheral to communicate to and modify the necessary config file and thus update the mirror UI in real-time. All three of them came up with the custom Bluetooth service.

6.2.4 Collaboration

We used one single Github repo for the entire team, and separate folders in it for each subaspect of the project. We divided up into teams based on personal interest and were each responsible for our piece of the project. For documentation, we each wrote about our respective parts, and those who were less busy with code wrote more of the need statement/persona/overview content of the reports.

6.2.5 Libraries Used

Magic Mirror[2]:

1. We used the Magic Mirror[2] repository as a blank template.
2. It can be found here: https://github.com/MagicMirrorOrg/MagicMirror
3. Although used as a base electron app, it was heavily modified in our end (prototype) product.

Adafruit Fingerprint Scanner:

1. We used Adafruit's fingerprint scanner repository.
2. It can be found here: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library
3. We used it as it was Adafruit's library, but we made it so the fingerprint scanning was autonomous (with the rest of our project) by writing our own code.

GATT Server Example:

1. We used a GATT server example.
2. It can be found here: https://github.com/Douglas6/cputemp
3. It was used as a base template due to its included GATT server scripts which helped with initializing & starting the server. However, the entire actual peripheral code was written entirely from scratch.

react-native-ble-manager

1. The mobile app was created from a blank create-react-native-app project.
2. For handling Bluetooth, the react-native-ble-manager library was used. It can be found here: http://innoveit.github.io/react-native-ble-manager/methods/
3. The entire mobile app was written from scratch, but I did use the library's example project for reference.

6.3 Appendix 3 - Test Plan & Results

6.3.1 Test Plan

1. **LED Screen**
   (a) **Goal/Purpose**: Testing LED screen visibility for integrated weather, news, and clock display behind an acrylic mirror.
   (b) **Parameters**:
      i. **Brightness**: Affects the readability of displayed content under varying brightness levels.

ii. **Darkness**: Affects readability and influences the contrast of displayed content.
iii. **Temperature**: This can affect the performance of the LED display depending on extreme conditions.
iv. **Condensation**: This can occur on the surface of an acrylic mirror, affecting visibility.

(c) **Expectation**: The LED screen will be less visible in brighter and/or condensated (humid) environments.

(d) **Date, Location, Who**:
i. **Date**: March 2024
ii. **Location**:
A. Baskin Lab 250
iii. **Who**: Everyone

(e) **Design**:
i. **Testing Method**: The LED panel will be on while the mirror is placed in different environments (bright room, dark room, and condensated room).
ii. **Significance**: Each environment will serve as a testing space for the mirror to test visibility across different distances.
iii. **Independent variable**:
A. Distance
B. Brightness level
C. Darkness level
iv. **Dependent variable**: Visibility
v. **Samples**: Dark and bright environments will be tested twice (at short, medium, and long distances) at 2 brightness/darkness levels.

(f) **Procedure**:
i. Place the mirror at a relatively eye-level height (about 5 feet above the floor).
ii. Set up the environment by either turning off the lights in the room (or dimming) or turning all lights on in a room (or brightening).
iii. Test LED screen visibility at 2, 4, and 8 feet distances and qualitatively record visibility observations.

2. **Fingerprint Scanner**

(a) **Goal/Purpose**: Testing fingerprint scanning accuracy for user authentication.

(b) **Parameters**:
i. **Angled Finger Placement**: Finger positioning can affect scanning accuracy.
ii. **Condensation**: This can affect the scanner's ability to accurately read fingerprints.

(c) **Expectation**: Condensation on the scanner or finger will trouble the scanner with recognizing the user. If the majority of the fingerprint is not on the surface area of the scanner or angled, the scanner will have trouble recognizing the user.

(d) **Date, Location, Who**:

i. **Date**: April 2024

ii. **Location**:

A. Baskin Lab 250

iii. **Who**: Everyone

(e) **Design**:

i. **Testing Method**: The mirror will be placed in a standard bathroom that is susceptible to condensation and wetness.

ii. **Significance**: Bathrooms allow us to forcibly create a condensation environment (with abundant amounts of steam emitted via hot water).

iii. **Independent variable**:

A. Condensation level

B. Finger Placement

iv. **Dependent variable**:

A. Recognition Speed

B. Fingerprint Accuracy

v. **Samples**: Fingerprint recognition will be tested in a condensated environment, and then in a dry environment. Angled finger placement (flat, right, left, up, down) will be tested in the aforementioned environments.

(f) **Procedure**:

i. Place a mirror in the bathroom (height placement is negligible)

ii. Set up the environment by either condensating the room by taking a shower, or applying small amounts of water directly to the scanner.

iii. Test the finger applied flat with the scanner having visible condensation, and record whether it fails or succeeds and the recognition time.

iv. Set up the dry environment with a completely dry scanner.

v. Test finger angles against the scanner. Record failure or success and recognition time.

3. **Lag**

(a) **Goal/Purpose**: Evaluate the response time of the mirror in fetching and displaying news, weather, and health updates.

(b) **Parameters**:

i. **Response Time**: Provides a quantitative for the time it takes the mirror to fetch information and visually update.

ii. **Efficiency**: Assess the mirror system's ability to handle requests (single or multiple) without major performance (CPU, memory) usage.

(c) **Expectation**: The smart mirror should have a quick response time (under a few seconds), for fetching and displaying new and current information.

(d) **Date, Location, Who**:

i. **Date**: April 2024
ii. **Location**:
    A. Baskin Lab 250
iii. **Who**: Everyone

(e) **Design**:

i. **Testing Method**: The mirror is tested under different loads by updating small or large amounts of data at a time.
ii. **Significance**: Will help simulate real-world usage scenarios when fetching data under different loads.
iii. **Independent variable**:
    A. Number of Requests
    B. Type of information being requested
iv. **Dependent variable**: Response Time
v. **Samples**: Updates will be tested by isolation in small or large fetching increments (i.e. only testing news updates). Updates will also be tested together (i.e. news, weather, or calendar) in small or large fetching increments as well. 2 tests will be done for each (news, weather, or calendar) isolated data fetch in both small or large increments for a total of 12 tests. 2 final tests will be done when requesting/fetching data from all 3 sources simultaneously.

(f) **Procedure**:

i. Set up the mirror and make sure it's connected to WiFi
ii. Test fetching small or large amounts of information under news, weather, and health updates separately.
iii. Record/measure the response time when pulling X amount of data from which source.
iv. Assess performance by tracking CPU usage and memory utilization.
v. Test fetching from the 3 sources (news, weather, health) simultaneously in small or large fetch amounts.
vi. Record/measure the response time when pulling data simultaneously.
vii. Also assess performance by tracking CPU usage and memory utilization.

6.3.2 Results

1. **LED Screen Visibility Test**:

(a) **Goal**: Test the monitor's (LED screen's) visibility for integrated weather, news, and clock display UI with a reflective acrylic panel on top.

(b) **Method**: Take qualitative assessments of visibility from varying distances in different environments.

(c) **Parameters**:

i. **Independent variable**: Distance and environment light level
ii. **Dependent variable**: Visibility

iii. **Other factors/considerations**:

(d) **Trial/Sample Test Results**:

i. **Sample 1**: Lights On (Bright)
- Distance: 2 feet
- Visibility: Good

ii. **Sample 2**: Lights On (Bright)
- Distance: 4 feet
- Visibility: Good

iii. **Sample 3**: Lights On (Bright)
- Distance: 8 feet
- Visibility: Satisfactory

iv. **Sample 4**: Lights Off (Dark)
- Distance: 2 feet
- Visibility: Good

v. **Sample 5**: Lights Off (Dark)
- Distance: 4 feet
- Visibility: Good

vi. **Sample 6**: Lights Off (Dark)
- Distance: 8 feet
- Visibility: Good

(e) **Analysis/Thoughts**:

i. This test was intended to ensure that our actual mirror UI was still visible with the reflective acrylic panel placed on top. We found that under bright conditions in the lab (the lights switched on), the visibility was still quite good but started to fall off once at far distances due to the white color of the text on a black screen. Under darker conditions though the overall visibility of the UI/text popped out more and was more adaptive to farther viewing distances.

2. **Fingerprint Scanner Accuracy Test**:

(a) **Goal**: Test fingerprint scanners accuracy for user authentication/recognition.

(b) **Method**: Take quantitative assessments on user recognition response time and qualitative assessments on success/failure of user authentication.

(c) **Parameters**:

i. **Independent variable**: Finger placement (angle on fingerpad)

ii. **Dependent variable**: Accuracy and recognition time

iii. **Other factors/considerations**:

(d) **Trial/Sample Test Results**:

i. **Sample 1**:
- Finger Angle Placement: Flat

- Recognized/Registered: Success, Success, Success, Success
- Response Times: 1.15s, 1.19s, 1.14s, 1.12s
- Average Response Time: 1.15s

ii. **Sample 2**:
- Finger Angle Placement: Up
- Recognized/Registered: Success, Success, Success, Success
- Response Times: 1.19s, 1.2s, 1.13s, 1.11s
- Average Response Time: 1.16s

iii. **Sample 3**:
- Finger Angle Placement: Down
- Recognized/Registered: Failure, Success, Success, Success
- Response Times: 1.05s, 1.1s, 1.14s, 1.13s
- Average Response Time: 1.1s

iv. **Sample 4**:
- Finger Angle Placement: Left
- Recognized/Registered: Success, Success, Failure, Success
- Response Times: 1.17s, 1.15s, 1.12s, 1.1s
- Average Response Time: 1.1s

v. **Sample 5**:
- Finger Angle Placement: Right
- Recognized/Registered: Failure, Success, Success, Success
- Response Times: 1.13s, 1.19s, 1.12s, 1.22s
- Average Response Time: 1.17s

(e) **Analysis/Thoughts**:
- This test verified that the fingerprint scanner worked relatively fast (between 1 to 1.2 seconds) in recognizing a fingerprint. We found that the angle of the fingerprint was somewhat important as it determined the surface area of our finger making contact with the actual scanner. Despite the fingerprint scanner blinking red in the case of not recognizing a fingerprint, we found that once registered the scanner would be less likely to fail. We also decided to not test in a wet environment due to how our purchased scanner works, which could have led to damage.

3. **Data Fetch Time (Lag) Test**:

(a) **Goal**: Evaluate the response time of the Raspberry Pi 3 in fetching data and consequently displaying the updated UI (news, weather, calendar, etc.) on the mirror.

(b) **Method**: Test the mirror under different loads by sending small (1-2) configuration changes to larger changes (entire modifiable thing in the configuration file).

(c) **Parameters**:

i. **Independent variable**: Number of requests

ii. **Dependent variable**: Response time

iii. **Other factors/considerations**:

(d) **Trial/Sample Test Results**:

i. **Sample 1**: Small Transfer
   - Module: News
   - Response Times: 25ms, 22ms, 27ms
   - Average Response Time: 24.7ms

ii. **Sample 2**: Small Transfer
   - Module: Weather
   - Response Times: 29ms, 21ms, 19ms
   - Average Response Time: 23ms

iii. **Sample 3**: Small Transfer
   - Module: Calendar
   - Response Times: 22ms, 30ms, 24ms
   - Average Response Time: 25.3ms

iv. **Sample 4**: Large Transfer
   - Module: News
   - Response Times: 47ms, 53ms, 50ms
   - Average Response Time: 50ms

v. **Sample 5**: Large Transfer
   - Module: Weather
   - Response Times: 48ms, 49ms, 55ms
   - Average Response Time: 50.7ms

vi. **Sample 6**: Large Transfer
   - Module: Calendar
   - Response Times: 53ms, 50ms, 44ms
   - Average Response Time: 49ms

vii. **Sample 7**: Small Transfer (all 3 modules)
   - Modules: News, Weather, & Calendar
   - Response Times: 44ms, 50ms, 48ms
   - Average Response Time: 47.3ms

viii. **Sample 8**: Large Transfer (all 3 modules)
   - Modules: News, Weather, & Calendar
   - Response Times: 88ms, 95ms, 92ms
   - Average Response Time: 91.7ms

(e) **Analysis/Thoughts**:

i. This test was intended to measure performance between our BLE central and BLE peripheral. Since we programmed the communication in a way that requires the data being sent to be very minimal (i.e. integers), this allowed the packet sizes to be relatively small. The results showed us that

as expected smaller transfers required less time, while the larger transfers required more. Then in the scenario in which multiple modules were being written, that also increased in response time. Overall though, we are happy with the response times despite the increase due to more characteristic writing. Visually though the mirror does take a bit longer to display these changes, but this time was not included when recording the response time data above.

6.4 Appendix 4 - Reflection

6.4.1 Daniel

The electron app was fun to work on. After changing up the Magic Mirror app, adding our own feature was a lot of fun. If I had to do something different, I'd commit to the Magic Mirror framework from the start. Initially, we began designing our own Electron app, and that took a lot of time before switching. Other than that I wouldn't change anything.

6.4.2 Erick

I enjoyed the challenge of reverse engineering the fingerprint sensor code to work as we would like it to for our program. Even though it took some time, it taught me how to engineer something without any external help. This was also my first time using electron, so it was a bit of a learning curve on how some code was interpreted since it caused complications in my written code scripts. This was the longest bit, reading outputs and finding edge cases that would break the main electron app. It was very time-consuming, but in the end, we worked it out.

I also made assumptions before writing code on portions of the code I thought would work without much effort, but I was wrong once I started actually coding. In hindsight, I am confident we would have been able to implement more features, in a nicer package, if we started earlier on the software side.

Working with these complications and hurdles that we came across as a team made for some interesting and fun interactions, such as figuring out how to package everything without the 3d printed frame since the printers stopped working. In the end, I am happy with what we all worked on together and how our final design was implemented given the circumstances.

6.4.3 Erik

This was quite a challenging project. I underestimated the scope of building an entire Bluetooth app from scratch. With all of the state management and UI needed, the app required about 2800 lines of original code. Despite starting this over spring break, this took me a significant amount of time throughout the quarter.

I'm not sure I'd use React Native for this again. On the upside, I learned a ton about Bluetooth and React and got quite comfortable in Typescript (I'd never want to write that kind of complex logic without static typing). However, RN's main BLE library didn't have a ton of examples or documentation, so that made it challenging.

As a team, I think we all could have started coding earlier. It felt like quite late in the quarter when we all had our parts working. However, we got along well, and I think the final design we came up with was a good one.

6.4.4 Andy

For what went well, we definitely were a functional team that worked together and achieved a lot as a group. We definitely made good progress on our prototype compared to what we envision this project to be and I believe that is something that I myself am very proud of. We also had a strong overall design. For what could've done better, we could've started doing harder work earlier and made sure we made very clear deadlines. We also could've delegated better and communicated to one another whether we needed help so we didn't fall behind.

6.4.5 Serjo

I initially started by looking into potential window managers to be used for this project but then decided to move to Bluetooth due to it being a more urgent manner. Along with my other teammates I have very little experience with Bluetooth and only had previous experience in client/server code as well as networking knowledge. Figuring out what to exactly build the peripheral on was a pain but once we found an example that worked on our machine, the rest of actually writing the peripheral script went relatively smoothly. With how we designed our BLE device and BLE central to communicate only certain types and sizes of data, we were able to optimize on not worrying about long response times. If we were to do this project again, we should be more on top of setting harder/quicker deadlines for compatibility testing as well as meeting more often for the entire teams, and more specifically subteams as that significantly increased our development for the project.

6.4.6 Evan

The Bluetooth Peripheral was difficult to make - none of us had prior experience with Bluetooth. It took a lot of trial and error to figure out what examples worked and were what I wanted, and more to learn from the examples to make our own code. A lot of results from Google searches were outdated, irrelevant, or not functional. In the end, it worked out. I wouldn't change anything.

6.5 Appendix 5 - Additional Design Justifications

6.5.1 Custom BLE Service

For the custom BLE service, we decided to split it up into one characteristic per configurable aspect of the user module configuration (enable, position, etc). This was because we were concerned about the MTU of the Linux Bluetooth, and we worried that trying to send the entire JSON user configuration at once would be error-prone. Sending anything larger than an integer was untested in our design. Any other way of splitting up the data communication (sending the entire user config JSON at once, or all of the fields for one module) could reduce the complexity of mapping characteristics to specific module attributes. However, we did not test this, and this would require retesting of all Bluetooth aspects of the mirror and app.

Our rationale for allowing the app to write to any user's config on the mirror is that this is a one-per-household device. Our reasoning was that if someone paired to the mirror from inside your bathroom (as ensured by the "OK" popup on both sides while pairing), then they are a trusted user. Having individual log-ins for the mobile app would introduce unnecessary complexity for what we deemed an irrational security risk.

6.5.2 Mobile App

We opted to use React Native for the mobile app. We did consider writing the app natively, either in Swift or Kotlin. The upside of this would have been being "closer" to the actual phone Bluetooth stack - and thus likely better performance, error reporting, and stability. However, due to the expected quicker time to bring the product to market with a cross-platform framework (as opposed to writing a separate iOS and Android app), we went for the cross-platform approach. Additionally, the app does not require any low latency or high-performance functionality. Also, both Flutter and React Native give the option to call native code if needed, so we had that option in case a cross-platform BLE library wasn't working as expected. We specifically settled on React Native due to having some prior experience with it.

For state management, we opted to use the Context API. Alternative approaches would have been to use a specific state management library (Redux), or a wrapper component. However, We estimated that our app was not complex enough to warrant Redux and that the prop passing associated with an encapsulating component would be ugly.

6.5.3 Magic Mirror^2

We chose to use Magic Mirror^2 as a magic mirror framework because it had most of what we already needed - a modular electron app that allows for the use of a configuration file to display certain modules onto the screen. It's essentially an efficient display manager that allows us to seamlessly integrate required modules for the user onto our monitor. Although we'd be able to make our own Electron app, it'd essentially result in the same thing - a base electron app that displays modules we make.