CSE123 Group 2
Schedule Companion

Mason Becker
marobeck@ucsc.edu

Sulaiman Islam
suaislam@ucsc.edu

Isabella Phung
itphung@ucsc.edu

Akanksha Rajagopalan
akrajago@ucsc.edu

Lennan Tuffy
ltuffy@ucsc.edu

June 12, 2025

1 Introduction

Time is an essential resource, and managing it throughout work and life is a constant balancing act. We aim to assist in sorting out these constant considerations and burdens through a system to organize time. Our approach utilizes technology that aims to provide support in time management or in hindering agents that waste time. Software that helps people adhere to their goals and promises, yet divorced from the busy screens that so easily distract us. A separate device, compatible with familiar scheduling services such as Google Calendar, yet capable of dynamically constructing a workflow for our user to follow. Static calendars do not address a key issue with abiding to a calendar: life is far too complicated to parse out into chunks without issue; interruptions and errors will always be present. Our device addresses this issue by fulfilling real time compensation from sudden interruptions to missed days of activity. Our separate device should always be present, reminding and encouraging the user to fulfill their goals and finding time to occupy with events they enjoy without worry.

Our device will adhere to association code ISO/IEC 27001 (Information Security Management Systems) and the Responsible Business Alliance (RBA) Code of Conduct.

Contents

2 Background

2.1 Need

Scheduling applications lack the medium to excercise greater precedence over other applications and lack organization between activities on the basis of necessity, urgency, and frequency.

2.2 Goal

Develop a system that utilizes a dedicated tablet device and a synchronous web application that organizes user activities on the basis of necessity, urgency, and frequency.

2.3 Existing Designs

There are a variety of existing digital planners available on the market, one of which is the Skylight calendar. It has a 15" touchscreen, 16GB of storage, and mobile app with third party integration from external scheduling applications such as Google Calendar and Outlook. It has todo list functionality, as well has up to date weather, calendar view, sleep mode, parental locking, many devices to one account as well as many users to one account. They have a monthly subscription plan with meal planning and phone screensaver functionality. However, while this device is very similar to ours by having a dedicated calendar device, it is not intended to be portable. It does not have a battery and it is unknown if the device operates without wifi connectivity. Unlike our device, the screen is large enough for an on screen keyboard when entering the wifi information, whereas ours will use an phone app for wifi provisioning. This device mainly serves as a digital calendar rather than providing guided focus sessions using timers as our devices does.

Additional existing products also include productivity apps available on any smartphone. TimeTune app has similar functionality to our device implementation such as time blocking, pop up notifications, a timer-based focus mode, and habit statistics. However, one of our key design objectives was to have a device separate from one's phone. A phone has countless distractions from messages, entertainment, and social media. So it is crucial that there exists a device that is separate from a person's smartphone while still remaining portable.

3 Why are we making this Device?

Productivity is a key factor in the success of both individuals and societies. In today's market, where media companies monetize our attention, staying focused and organized while prioritizing tasks can be challenging. Although scheduling applications help with organization, they often treat all tasks the same, making it hard to determine priorities and decide on which activity to work on and when. To address this, we identified the parameters of differentiation that would address activity priority and these were necessity, urgency, and frequency. Our application is designed around these criteria to help users prioritize more effectively.

3.1 Personas

The following are a few target audiences for our productivity device:

Camillia Anas

Info

- Married, with two children
- 41 years old
- Occupation: Cardiologist

Goals

- Needs to stay on top of her patients, but also wants to be present for her kids' events

Behaviors

- Works long shifts at the clinic
- Often takes vacations with the entire family
- Kids have many extracurriculars such as: piano lessons, tutoring, and sports meetups

Elijah Eiko

Info

- Has a partner, no children
- 26 years old
- Occupation: IT Help at a drone development company
- Recently adopted a puppy

Goals

- Recieves *many* emails and messages from within the company for issues and meetings to fix something
- Wants a way to keep track of all the tasks he gets in a day and prioritize them easily
- Sometimes meetings and appointments go for too long and needs to rearrange task list

Behaviors

- Works from home from time to time
- Working on building habits with his puppy

Linda Alfreda

Info

- Single, with no children
- 27 years old
- Occupation: Marketing and Sales

Goals

- Focused on her work and beginning to take on a managerial role at her company
- Needs to manage multiple meetings per day and plan tasks for others

Behaviors

- Sometimes forgets to take care of herself
- Still learning how to delegate tasks to others

Mary Torfinn

Info

- Single, no children
- 22 years old
- Occupation: University student

Goals

- Needs to manage multiple assignments, chores, and social tasks
- Wants to build better study habits, looking for something that will help her concentrate

Behaviors

- Finds herself sitting for long periods of time
- Has a hard time focusing when using other productivity apps since using her phone is distracting

Peyton Woodie

Info

- Has a partner, with one child
- 31 years old
- Occupation: Barista and bartender

Goals

- Wants to build a new habit of exercising regularly, but it's difficult to find time with a varying schedule

Behaviors

- On same days with double shifts, he is too exhausted to work out

3.2 Sustainability Statement

In designing this device, we wanted to create a sustainable product. Our device will be built using recyclable materials for its casing and internal structure, and we will work with suppliers who adhere to environmental and social governance (ESG) standards. We will also use sustainable packaging materials such as vegetable inks for printing, recycled cardboard, and bio-based plastic films. We are commited to minimizing the environmental impacts involved with the creation, usage, and disposal of our device.

4 Design

4.1 Entry Design

The schedule companion is designed to interact with the user through guided workflows based on the data entries they provide, enabling a streamlined approach to task and time management tailored to the user's daily routine and priorities.

4.1.1 Time Management

To function effectively, the device depends on accurate timekeeping, which it achieves through a Real Time Clock (RTC) in conjunction with optional network synchronization capabilities. Timekeeping is critical because the device emphasizes the *here and now*, presenting the user with only the most relevant current activity rather than attempting to manage a full time-blocked schedule. Traditional time-blocking assumes accurate future estimations from the user, which is often unrealistic. For instance, knowing how long a project will take to complete before starting it would be improbable. To circumvent this, the device uses the concept of a *work period*: a user-defined segment of time during which the device encourages tasks to be completed by the user.

Once a work period is initiated, the device identifies and presents the highest priority task to the user. Alternatively, through onboard controls, the user may browse other top-priority tasks. The work period continues until its predefined cutoff is reached, or the user ends it early. During this time, the device can engage a focus-oriented operating mode inspired by the Pomodoro technique, segmenting work and rest intervals. While in *focus mode*, distractions are minimized: the user interface displays only the task's name and remaining time until the next break, reinforcing attention on the task at hand.

The user maintains complete agency throughout the process. Work periods can be paused, concluded early, or marked complete at any time. Break periods follow a structured prompt system, offering the user three choices: to continue with the current task, to mark it as complete, or to conclude the session and potentially transition to another task. These structured yet flexible workflows help the user remain productive without overcommitting or becoming overwhelmed. For these features to operate meaningfully, the device must maintain a detailed understanding of the user's responsibilities and routines.

4.1.2 Data Management

To inform the device's behavior, user-supplied data is categorized into three primary entry types: **Tasks**, **Events**, and **Habits**, each fulfilling a specific role within the internal logic of the system and mirroring aspects of one's day-to-day experiences.

Tasks are defined objectives that must be completed by a given deadline. Each task entry includes a textual description, a due date stored as a Unix timestamp, and a priority parameter that conveys the size or effort required as seen by the user. The device uses these values, particularly the due date and priority, to determine task priority and ensure that more urgent or significant tasks are surfaced first during work periods.

Events represent fixed intervals of time and are recorded within the device's internal calendar. These may include meetings, appointments, or any other scheduled commitments. Events can be treated as *busy periods* and, if busy periods overlap with a work period, their start time is

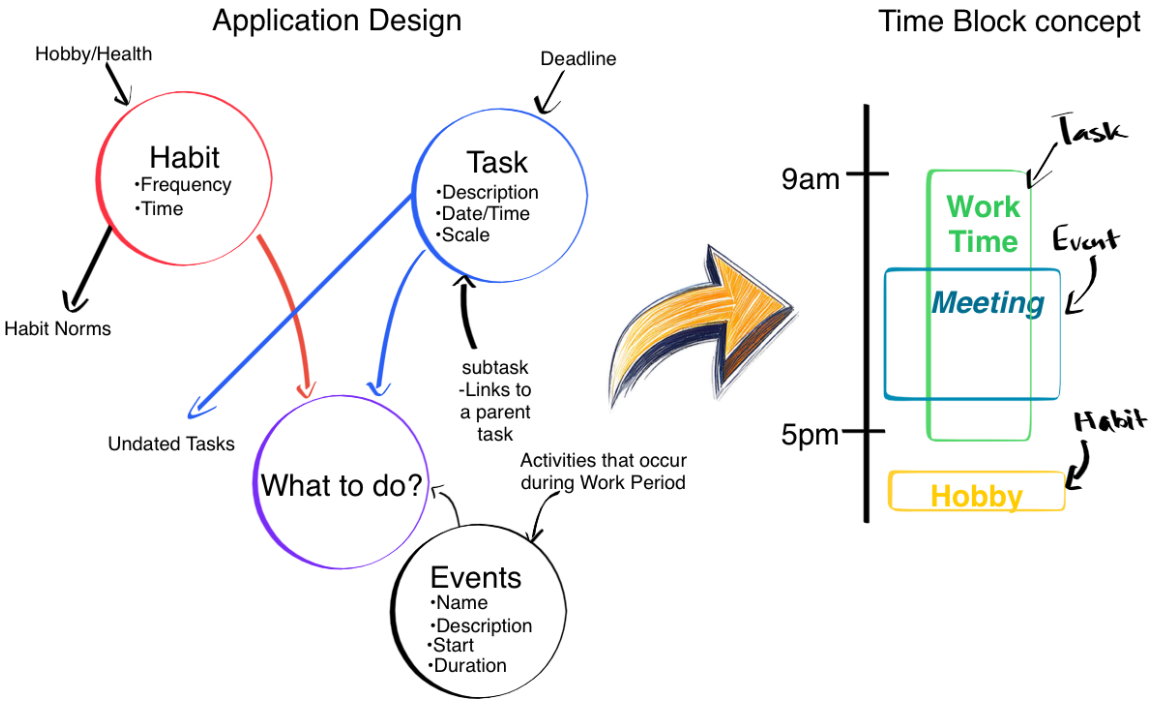


Figure 1: Data management and logic of the device

interpreted as a cutoff point. This allows the device to dynamically adjust work periods to avoid scheduling conflicts.

Habits, in contrast to tasks, represent repeated behaviors the user wishes to cultivate. Rather than relying on due dates, habits are defined by a *frequency* metric. This metric is internally represented using a binary bit field, where each of the seven least significant bits corresponds to a day of the week. If a given bit is set high, the habit is expected to be performed on that day. This compact representation avoids unnecessary space consumption and database reads/writes. Habits are displayed within a dedicated interface, frequently shown to remind the user of their daily responsibilities. The design ensures that habit tracking remains prominent but not intrusive.

There is also an auxiliary category visible in Figure 1 labeled *hobbies*. These are undated task entries, included as a means for users to track long-term personal projects or leisurely goals without the pressure of deadlines. By omitting due dates, hobbies serve as a gentle encouragement rather than an obligation, however the device will offer the same focus mode functionality for these low priority achievements as was introduced for work periods. The device would rely on the priority parameter dictated by the user to order the entries such that the user can readily choose a task that they find most pressing or prominent.

4.1.3 Database Integration

The relationships between tasks, habits, events, and their associated data are maintained using a relational SQL database. This model was chosen for its robustness in expressing complex data inter-relations and its efficiency in handling queries related to sorting, filtering, and retrieving contextual information.

Each primary data category: tasks, habits, events, and habit entries, is mapped to its own table within the database. These tables mirror the structure and logic described previously. For instance, task entries store a textual name and description, a Unix timestamp representing the due date, and a priority field. Unix time was selected as the time representation format because it enables consistent numerical operations, such as sorting by ascending time or computing time differences.

The task priority field plays a dual role: it captures the user's sense of urgency and informs the device's task sorting logic. Sorting is governed by a formula: $Date - Priority \times Multiplier$, where the multiplier is a constant that converts the priority value into a time offset. This transformation allows tasks with higher urgency to rise in importance even if their due dates are farther out.

Habit frequency, as mentioned earlier, is stored in a single numerical field interpreted as a 7-bit value, with each bit representing one day of the week. Since SQL lacks a dedicated bit-field datatype, this field is read and manipulated at the application level. As long as both the server and client interpret this integer consistently, the system can efficiently determine on which days a habit should be active, without requiring additional columns or redundant data.

Tracking user adherence to habits is managed through a separate *habit entry* table. Each entry records whether the user completed a habit on a specific day, using a timestamp truncated to the start of the day via the formula $date = time - (time \bmod 86400)$. This approach ensures that all entries for a given day are uniformly referenced, facilitating both retrieval and statistical analysis. Additional numerical fields may be included in the habit entry to represent progress toward more granular goals, such as step counts or hydration targets. The presence of a habit entry alone may suffice to indicate completion, while its removal serves as a mechanism for user correction to indicate incompletion.

This entry-per-day model offers a balance between precision and efficiency. It allows the device to track how consistently users meet their goals without bloating the database with unnecessary metadata. All of this contributes to a responsive, insightful interface that helps users stay on track while minimizing the cognitive overhead typically associated with time management tools.

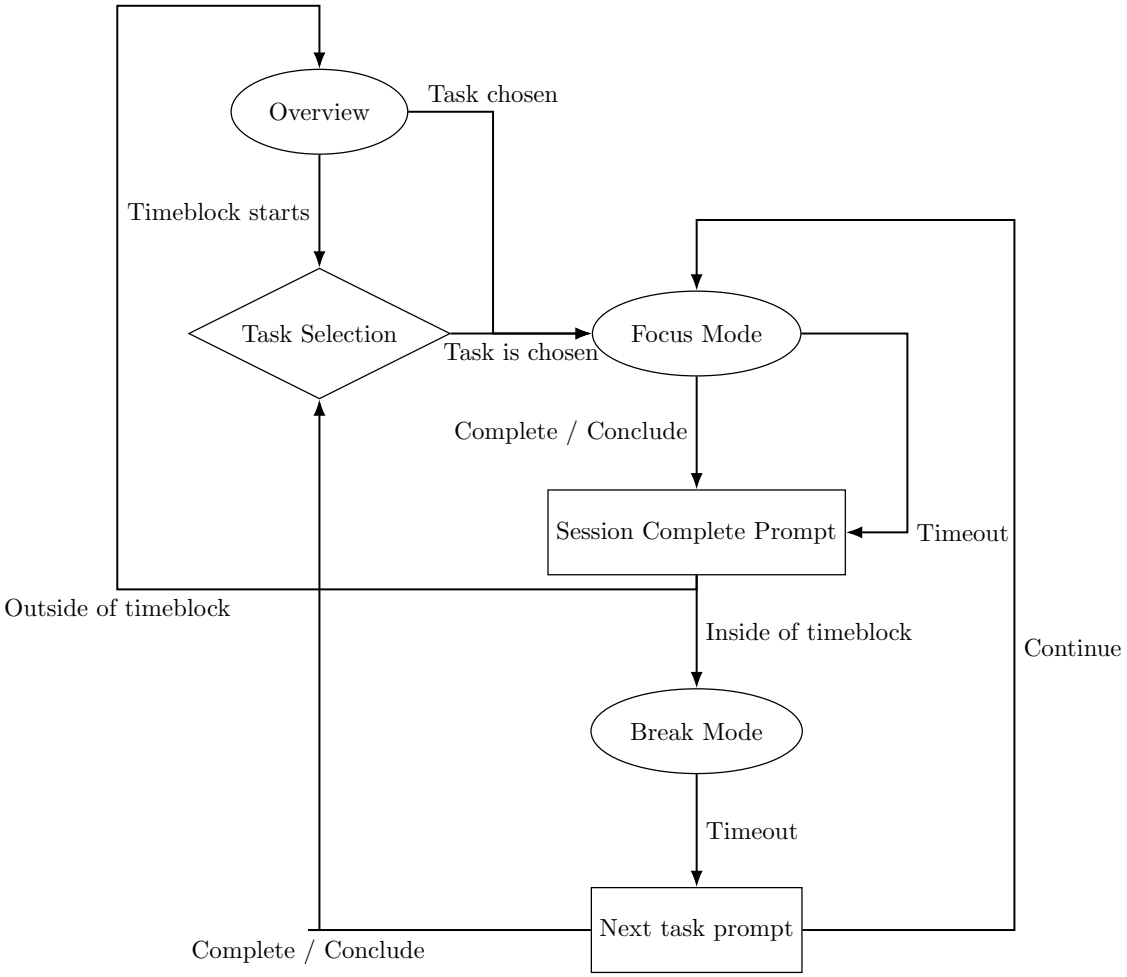


Figure 2: Flow of task focus and selection during a timeblock session

4.2 Hardware Design

4.2.1 Processor

Initial schedule companion prototype used an ESP32C3 board which was determined to be too limited in memory for this product's intended features. The ESP32C3 also uses RISCV which cannot operate with external RAM modules. A system with greater than 520 KB of RAM and an ARM based CPU is preferred for this device. The CPU of the ESP32C3 board ran at 160MHz but a

lower clock speed is acceptable, as the main bottleneck in the prototype was the limited memory. An ESP32-S3 unlike the C3, is capable of interfacing with external RAM modules, making it a potential alternative to the C3 microcontroller. However, ESP-IDF only supports Espressif branded external memory. A Teensy 4.1 with its 1MB of RAM and RAM expansion capabilities makes for another potential microcontroller candidate for this device, although it is limited to one core which may cause bottlenecks. STM32H7 series chips are another inexpensive option with 1MB of built in RAM, RAM expansion capabilities, and ARM based dual-cores, allowing for one to handle device cloud interactions, and the other for graphics display.

4.2.2 Additional Components

Device requires flash memory, which can be added as a separate module or as an IC chip on the device board. A separate module using an SD card is preferable to make the product more easily repairable. An external RTC module with a coin cell battery is also required to keep time and should be placed in the device housing such that the coin cell is easily replaceable in the event it runs out of charge. A small button motor and speaker must also be included in the productivity device for haptics. A button for powering on and restarting the device is also required. The device will use a USBC port placed on the back for power.

4.2.3 Power

The schedule companion, unless set to power saving mode, will keep its screen on until the user presses the power button briefly to return it to sleep mode. In power saving mode, it will automatically enter sleep mode after one minute of inactivity, but focus mode timers will continue to count down even in sleep mode. If the power button is long pressed, it will shut down the device. When the device is on very low battery, it will power off and display a low battery icon when the power button is pressed. If the battery is too low, the device will be unable to display any icon at all. Once the device has been recharged enough, the device will power on as per typical operation. A long press will make the device fully power off. The schedule companion is expected to operate offline using two 3500mAh 3.7V 18650 batteries wired in series in an easily removable battery holder. Rather than using a specific pouch battery, this would allow for the user to easily replace the batteries in the event of cell failure and make the device highly repairable. The power PCB will require step-down converter, as well as charge balancing circuitry, and a thermometer to monitor the internal device temperature to only allow charging from 0C to 45C as is recommended for lithium-ion batteries[1]. Although precise battery life cannot be calculated, rough estimations approximate minimum battery life to be 4.6 hours, assuming the microcontroller and its peripherals draws 200mA[2] and the screen uses 550mA[3] for a total of 750mA of current. Using a 1A charger with 85% charging efficiency will require about the same amount of time to charge as it does to discharge, about 4 hours, calculated using the following formula: Charging Time = (Battery Capacity / Charging Current) × (100 / Efficiency)[4]. The device is also expected to operate when plugged in with the batteries removed as well.

[1] https://web.archive.org/web/20090411024100/http://www.sony.com.cn/products/ed/battery/download.pdf
[2] https://www.emcraft.com/som/stm32h7/STM32H7-SOM-CURRENT_CONSUMPTION.pdf
[3] https://community.element14.com/products/raspberry-pi/f/forum/11117/official-rpi-7-touch-screen-power-requirements
[4] https://www.orbtronic.com/18650-li-ion-battery-charge-time-calculator-charging-chart/

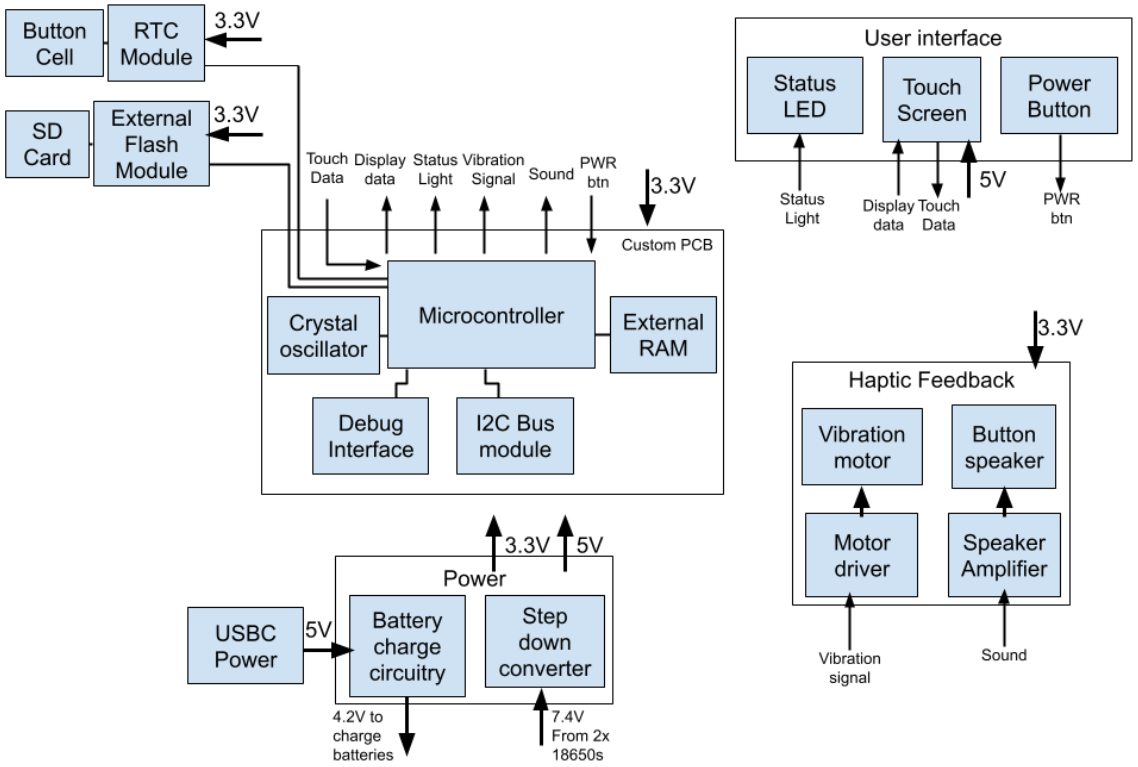


Figure 3: Manufactured PCB Diagram

4.2.4 PCB

As mentioned earlier, it is preferrable to keep the boards fairly modular, mounted into the PCB housing using screws. In the event the selected SOC chip does not have built in wifi and/or bluetooth, an additional module with a bluetooth and wifi antenna must be included on the PCB. The PCB should also include a debugging interface such as JTAG. All of the PCBs are secured inside of the housing using small screws.

4.3 Device UI Specifications

The device UI is developed with a similar style guide to the web app and phone app. Schedule units, whether they be tasks, habits, or events, must be formatted in similar manners across all devices for ease of use and continuity.

Tasks are displayed in a manner that is easy for the user to view and access them, preferably in the form of a list showing the due date with some way to differentiate between completed and incomplete tasks, either with color or a symbol.

When selected or tapped on, the task displays either a different menu or expands in some way for the user to view its details: the description, its priority, its due date, as well as be able to interact with the task by deleting or focusing the task. An example is shown in fig. 4.

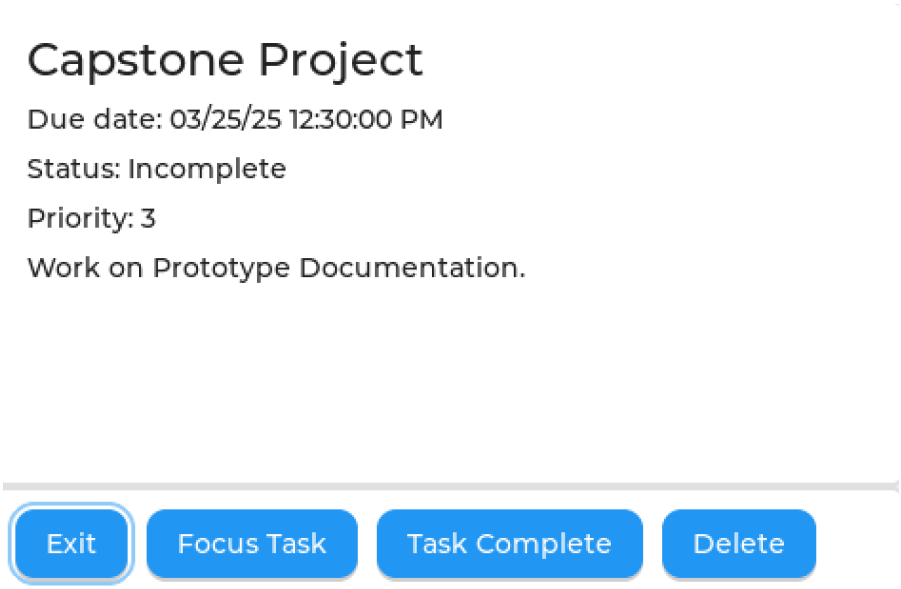


Figure 4: Task Detailed View

Entering focus mode halts all synchronization tasks so as to save on battery and displays the current date and time, a timer that counts down for the task, and options for the user to exit out of focus mode, restart the timer, or set the current focused task to complete. The timer for this focus mode can be changed in the user settings. An example of focus mode is shown in fig. 5

Completed tasks are displayed until some duration in which they are deleted from the server and the device. This duration can be adjusted according to user preference in the settings. If a user decides to delete a task, a confirmation screen appears to ensure the user has not missclicked. If a task is deleted, it is immediately removed from the device UI.

Events are also be displayed in an easy to view manner, preferably as a list. Events mirror tasks in that, when selected, they also display details regarding the event and options such as deleting the event. Events are not be displayed in the list if they have already passed. Similar with tasks, events show a deletion warning before confirmation and they are deleted immediately.

An example of the tasks and events menu is shown in fig. 6. While the task and event views were combined into one screen, they can be separated in to multiple screens.

An optional additional view with be a calendar that displays all of the events as well as tasks is also recommended.

Habits are displayed going right to left, with the rightmost being the current day and the left most being 6 days prior to the current day. This gives 6 days for the user to check off if they have performed a habit. The list of days is updated at midnight of the user's time. An example of the habits menu is shown in fig. 7. Different colors are used to differentiate between the days where the habit *should* be done and days when the habit *has been performed*.

Depending on the amount of available RAM in the device, schedule units can be displayed in groups at a time and updated when the user scrolls down. An alternative is loading a specific number of schedule units and using arrow buttons to load more tasks and events.

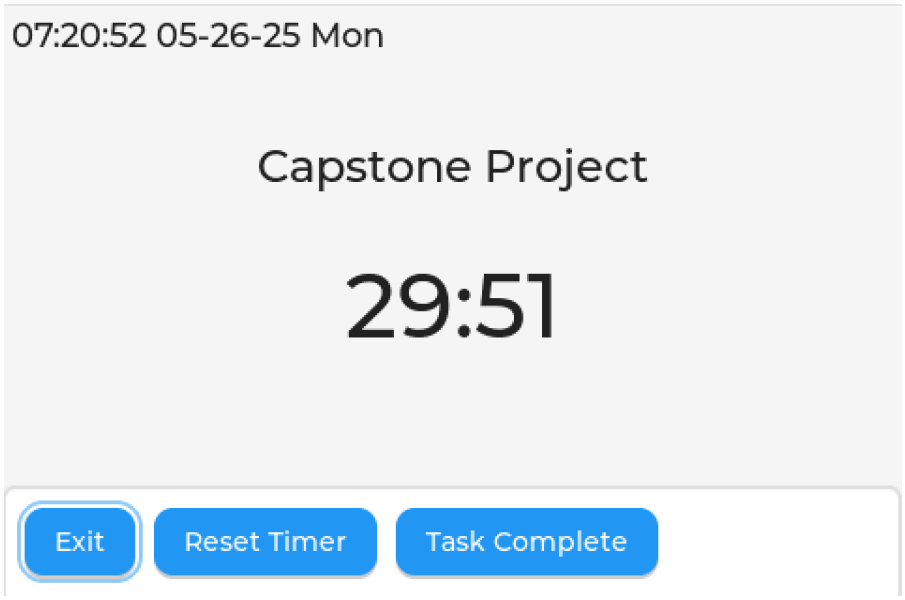


Figure 5: Focus Mode

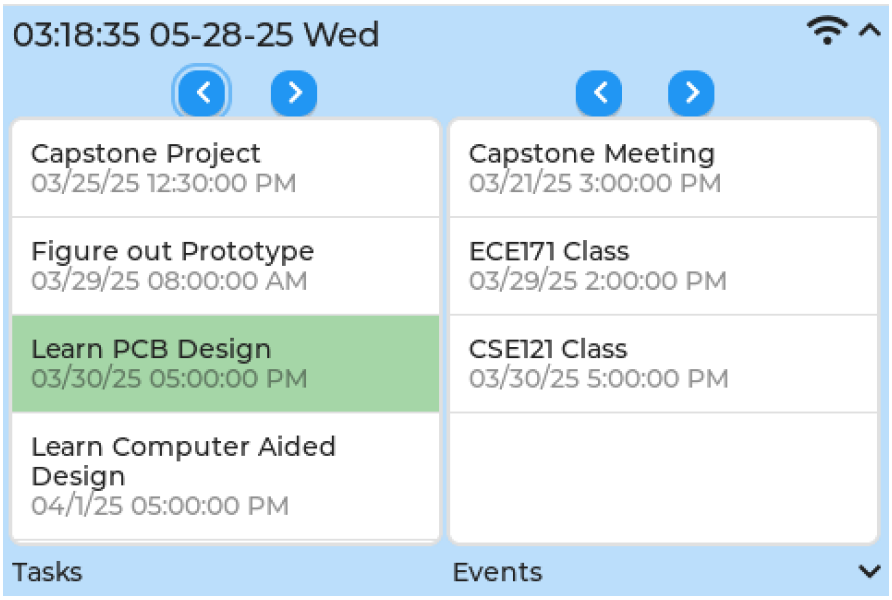


Figure 6: Task and Event Menu

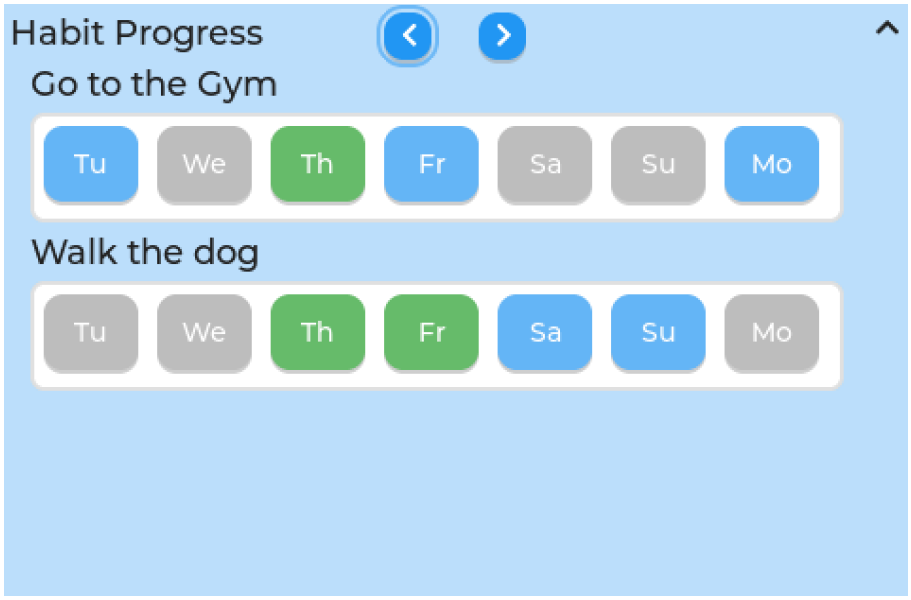


Figure 7: Habit Menu

When the device is being set up for the first time, a screen instructing the user to download the phone app and connect to the device via their phone's bluetooth is displayed, similar to fig. 8. This screen is only ever shown during first time setup out of the box or after a factory reset. After setup when the device loses internet connection, it will continually search for bluetooth connection while still allowing the user to continue using the device unless the user turns the device's wifi off in the settings menu.

The UI must display some manner of wifi symbol on the screen when the user is viewing their schedule to indicate when the device is connected. In addition, a synchronization symbol is displayed when the device is synchronizing with the server.

Pop ups appear on the lower area of the screen prompting the user if they want to enter focus mode for certain tasks if the user has enabled the guided time block system.

The UI must also adapt according to the user configuration settings. Color theme settings change the colors of the UI, text size settings scale the text, and changing the timezone and time display settings change the time values on the device appropriately.

Whenever a user tries to add a device to their account, a pop up will appear on the screen stating "User is trying to add this device to their account." with two buttons allowing the user to confirm that this is them or to reject the addition.

4.3.1 User Tutorial

In order to ensure the user fully understands how to utilize the device, a user tutorial should be provided in a similar fashion to shown in Fig. 9. Additional panels for user configuration settings should also be included for user clarity.

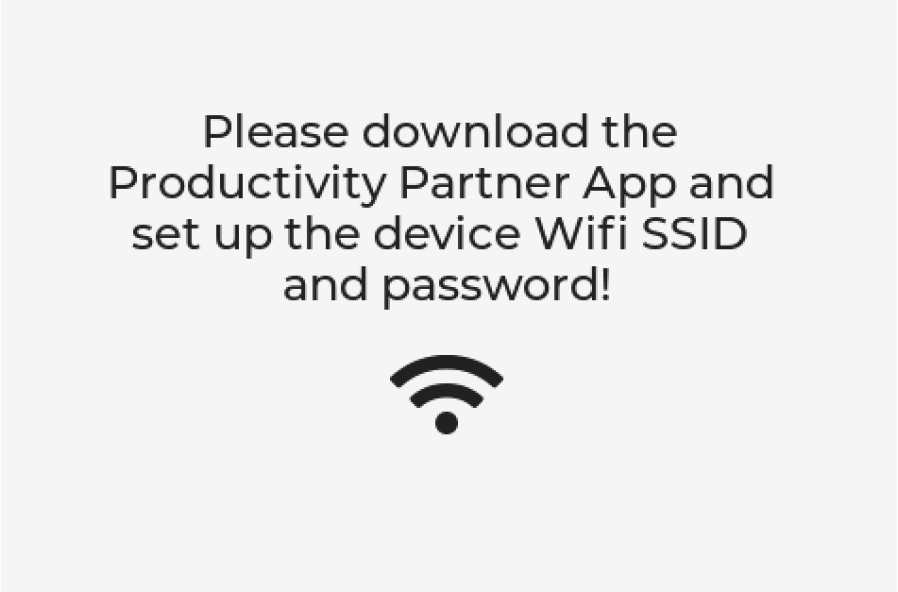
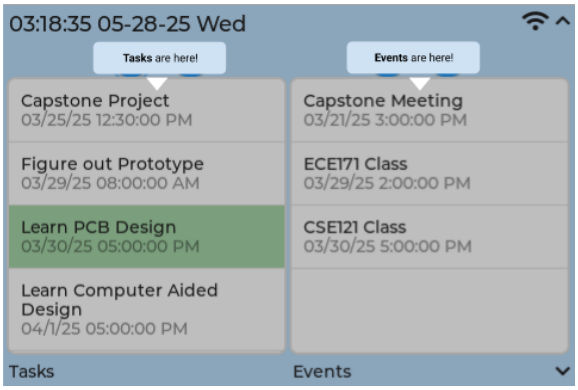


Figure 8: Wifi Setup

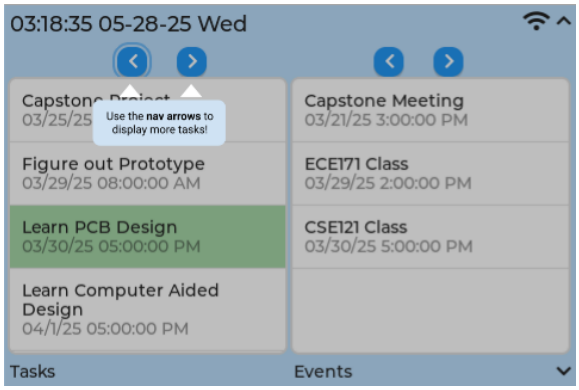
4.4 Configurable Settings

Some of the User Configurable settings are and values displayed on the web app and phone app are:

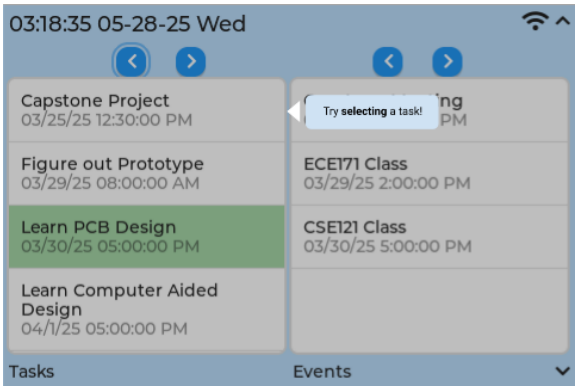
- Switch User/Log out
- Disconnect from Wifi/Airplane Mode
- Screen Brightness
- Power saving mode: When enabled, device will enter sleep mode after one minute of inactivity.
- Time format: 24 hr vs 12 hr, include or exclude seconds
- Timezone
- Text Size: small, medium, and large
- Date format: Change between date formats such as MDY, DMY, YMD.
- Color scheme: light mode or dark mode, additional personalized color schemes are also provided.
- Guided Time Block system enable: user can enable this for a guided time block system which will automatically prompt the user if they want to enter focus mode for different tasks. This can be disabled for users who would prefer to select which tasks they want to focus on by themselves.
- Focus mode timer duration
- Focus mode pomodoro setting: user can switch between a standard timer or a setting that will automatically begin a 15 minute break timer during focus mode
- Keep Completed Duration: User can select the duration with which completed tasks remain on their account: delete immediately, keep for 6hrs, keep for 12hrs, keep for 1 day, keep for 3 days, keep for 1 week.



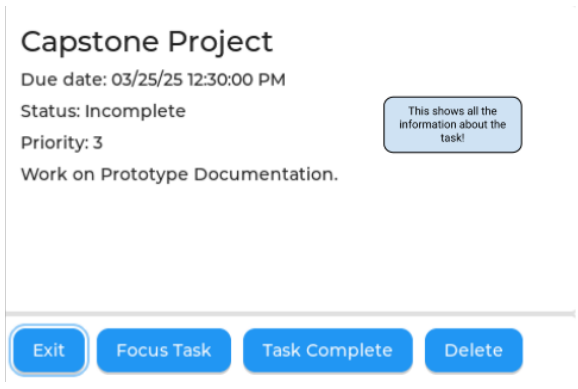
(a) Shows what the different columns mean



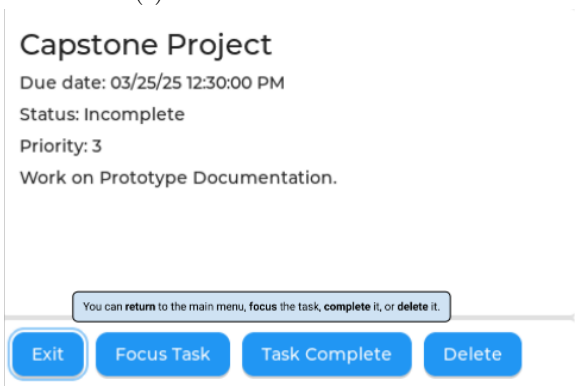
(b) Introduces navigation arrows



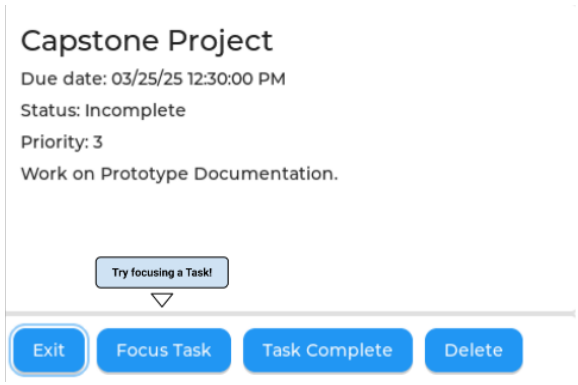
(c) Tells user to select a task



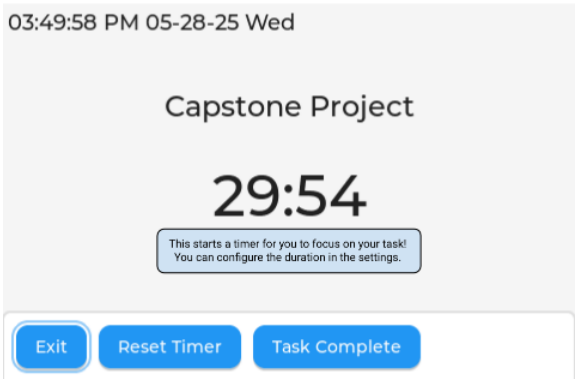
(d) Introduces task detail view



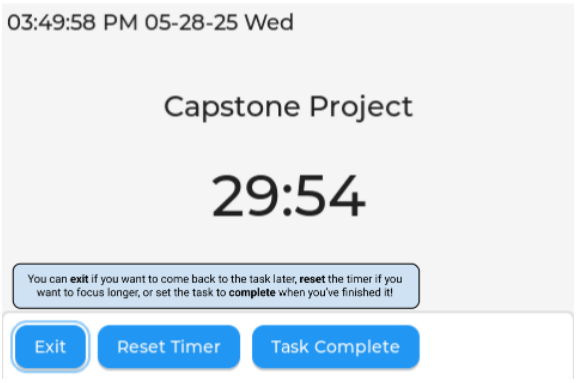
(e) Introduces task interaction buttons



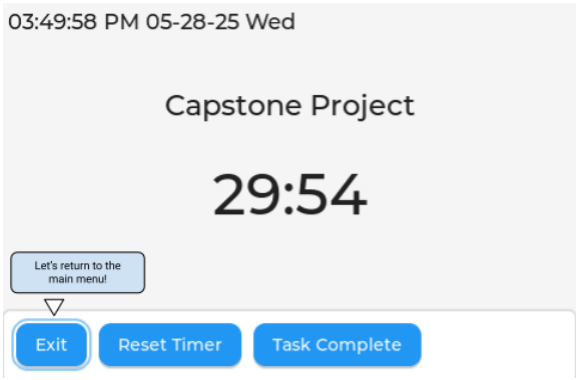
(f) Tells user to try focusing a task



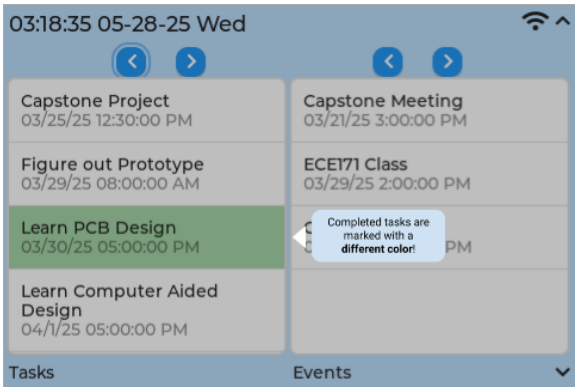
(g) Introduces focus mode



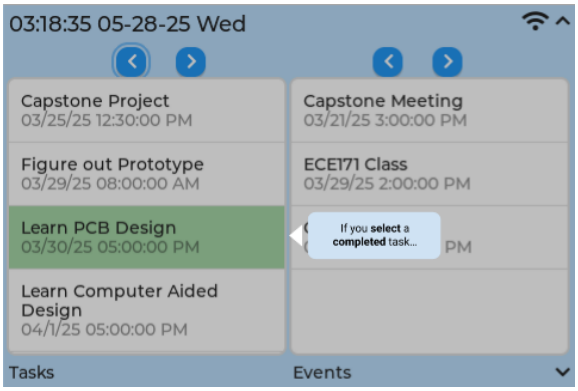
(h) Introduces focus interaction buttons



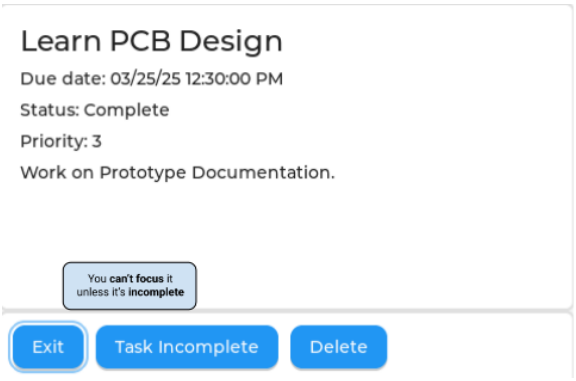
(i) Tells user to return to main menu



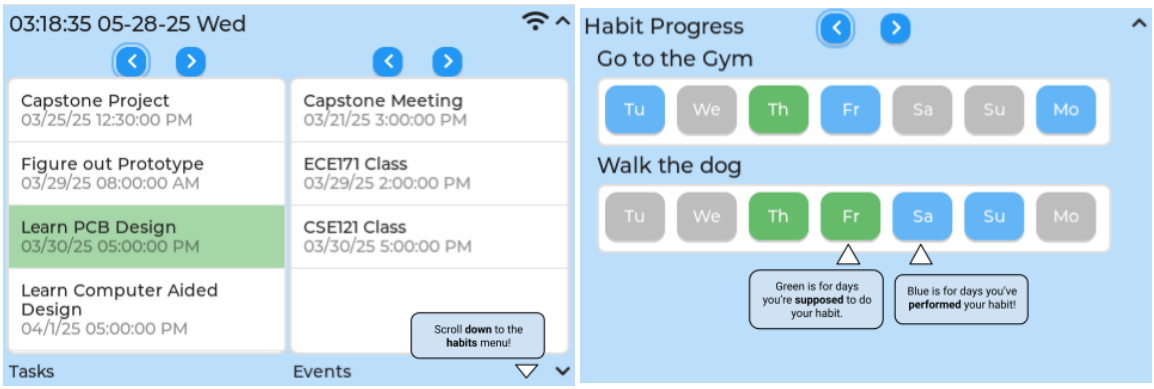
(j) Introduces completed tasks



(k) Tells user to select completed task



(l) Informs user they cannot focus completed tasks



(m) Instructs user to scroll to find habits menu

(n) Introduces color meanings for habits

Figure 9: User tutorial example

- Device ID
- Check for Updates: checks for available updates and asks the user if they want to update their device if an update exists
- Software version: lists current software version
- Factory reset

4.5 Device-server Communication

4.5.1 Entry Serialization

The server and device must agree on a set format for entries in order to communicate effectively. The three entry types — tasks, events, and habits — are designed, and serialized in a way that is most compatible with the different data structures that are in use. The JSON format should be used as it has widespread support. Thus, text-based protocols should be used.

All entry types have a unique identification and a name (limited to 32 characters). Tasks contain an UUID, name, description, completion status, priority, and due date. The description can be used in addition to the name to describe the task; its length is limited to 1024 characters. The completion status can be set as incomplete, complete, or deleted. When a task is set to be deleted, any changes made to it are to be ignored, and should not be displayed on any client. The priority ranges from 1 to 3, to be used as described in Section 4.1. The due date is encoded as a unix timestamp by when the task is to be completed.

Events contain an UUID, name, a start time, and a duration. The start time is a unix timestamp of when an event begins. The duration is the amount in seconds from the start time when the event ends.

Habits contain an UUID, name, goal, and completion status. The goal is encoded as a bit-mask of the 7 days of the week — the least significant bit corresponding to Saturday. For example, a habit goal of Tuesdays and Thursdays would be encoded as `0b0010100` or 20. The completion status is a list of unix timestamps for the day a habit was completed. To be consistent, the 0th second of that day is used.

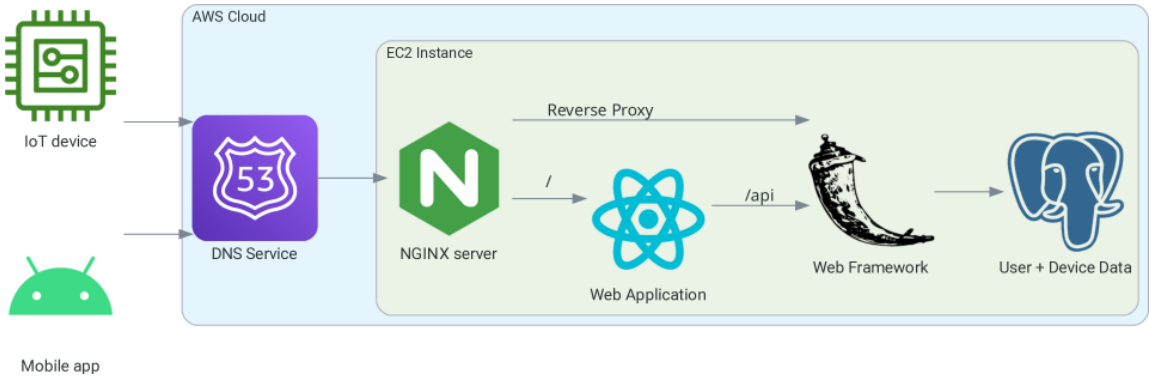


Figure 10: Example server architecture

4.5.2 Protocol

The data transferred between the device and the server must be encrypted to respect the privacy of the user. Several text-based protocols support this, such as HTTPS or MQTT. The device will make requests to the server to get the entire data backup, or incremental backups since a reference timestamp. Any modifications made on the device will be sent to the server with the corresponding UUID, and the modified fields. Packets being sent to or from the server should come with the entry serialization, as well as the following: source ID, action type, and data type. The server will have the ID of "server". The source ID is the UUID of the device, which is pre-configured at the manufacturing time. The action type can be either refresh, update, response, or ack. Requesting to the server for the full backup is done with the refresh action, which gets the response action back from the server. Requesting for changes to be made to entries is done with the update action, which gets the ack action back from the server.

4.6 Server design

A web server will facilitate the communication between the device and the cloud backup, as well as serve a web app for data entry and viewing. The server architecture is summarized in Figure 10.

4.6.1 Encryption

Security is a high priority when involving user data in a system involving the cloud. To ensure security of the data being transmitted between the user and the device, the data must be encrypted using Transport Layer Security (TLS). NGINX creates a secure connection by sending the ssl certificate to clients. NGINX serves requests to the server coming from the client or the device, and routes it to the appropriate endpoint.

4.6.2 Database Connection

User login and configuration information, as well as device backups are stored in a PostgreSQL database. This data storage is not directly accessible from the internet, and is accessed by making requests to the server API. Devices making requests for database refreshes or data modification

must make them to the "/api" url with an unexpired, valid API token. The react front-end fetches user and device data from the database similarly to the device.

4.6.3 Flask API

The Flask server has API endpoints which communicate with the database. This includes endpoints to verify login credentials, get tasks/events/habits for a specific device, and update tasks/events/habits for a device. The Flask server also has login and logout token endpoints to set and remove cookies containing JWT access tokens. If an access token for a user is expired or not found, the Flask app will redirect the user to the login page. Device selection is also controlled through the Flask server. When a user selects a device, the server sets a cookie with an API token for that device. The database endpoints are used by the web app and device, and the endpoints with cookies are used by the web app to keep track of the current user/device.

4.6.4 Device security

When associating devices to accounts, it's important to ensure that users aren't able to access devices which aren't theirs. In order to do this, each device will be shipped with a serial number (ID) and a code. When adding a device, the user must enter the ID and code, along with a name for the device. The server will then notify the device that a user is trying to link with it, and will ask for confirmation. Once the user clicks "confirm" on the device, the device ID will be linked with their account. This way, a user can't access a different device by mistyping or trying different serial numbers because the code won't match. When the user selects a device, the server creates an API token for the device by encrypting the account's username and selected device. This token is then sent to the server on all API requests. This prevents users without access to a device from getting data from the API since their token will fail to be decrypted successfully.

4.7 Webpage Design Philosophy

The schedule companion requires a web application that will provide the user with a medium to manage user-to-device settings, manage user data, and view or edit scheduled items. This section of the document provides the principles that are necessary to the design of this webpage. These principles revolve around the categories defined in entry design and their actionable functionality within the user interface. It also includes necessary applications for the user interface such as the login, device management, and calendar pages.

Authentication and User Management:

- User Authentication through login page
  - Sign up option within login page to register new user
    - Create account with full name, username, email, and password
      - Username uniqueness check
      - Password strength validation with visual feedback
      - Automatic user ID generation
  - Forgot password option to reset credentials
  - Prompts user for email and password
    - Credential validation against existing user

        * Session token management with CSRF protection
        * Secure login state across browser session
        * Includes persistent login state across browser sessions

Device management

- Device Registration
    - Add device with unique device ID
        * Device will utilize wifi to link to webserver
        * Option to link device to user account
    - Switch between user devices
    - Remove device from user directory
- Device State Persistence
    - Remember selected device across sessions
    - Handle device availability and connection status
    - User data linked to device

Main Page

- Access to tasks, habit, event, calendar, device managment, and login page
- Add, delete or complete tasks, habits or events
- Display of today's tasks, habits and events with their times
    - Times implemented with Unix timestamps
    - View task description, priority, completion and due times
    - View event duration and due date

Tasks

- Addition
    - Prompts user for task name, due date, due time, priority, and description
        * Set priority level using numeric values 1-3
            · Initialize completion status as 0 (incomplete)
        * Define due date and specific time (hour, minute, AM/PM)
            · Convert time selection to Unix timestamp for storage
- Management
    - Toggle completion status: 0 (incomplete) - 1 (complete)
    - Soft delete functionality: set completion status to 2 (deleted)
    - Maintain completion state persistence across sessions
- Display
    - Filter tasks by date range (today, specific dates)
    - Sort by completion status first, then priority level
    - Visual priority indicators with color coding

  - Expandable detail view showing full task information
  - Real-time timestamp display and conversion to readable format

Habits

- Creation
  - Prompts user with day of the week display that allows user to select days when habit should be fullfilled
  - Prompts user for habit name
  - Add and remove habit options
- Definition
  - Bitwise flags to define weekly schedule
  - Flag mapping: Sunday=0x40, Monday=0x20, Tuesday=0x10, Wednesday=0x08, Thursday=0x04, Friday=0x02, Saturday=0x01
  - Calculate combined flags for multiple day selection
- Completion Tracking
  - Track daily completion status per habit
  - Maintain completion history across different dates
  - Visual distinction between scheduled versus non-scheduled days
- Daily Habit Display
  - Provide full weekly grid view in dedicated habits page
  - Current day highlighting in weekly view
  - Completion status visualization with checkboxes

Events

- Creation
  - Prompts user for event name, event date, start time, priority, duration, and description
  - Define start date and specific time (hour, minute, AM/PM)
  - Convert time selection to Unix timestamp for storage
- Status Tracking
  - Determine "active" events (currently happening based on start time + duration)
  - Track event completion or cancellation
  - Soft delete functionality for events
- Display
  - Show events by date range
  - Display calculated duration and end times
  - Visual indicators for currently active events
  - Time-based sorting of events

Calendar

- Monthly Calendar Display
  - Generate calendar grid for any month/year
  - Properly handle month boundaries and leap years
  - Navigate between months with controls
- Highlight current day across all calendar views
  - Task-calendar integration
  - Display tasks on their scheduled dates within calendar
  - Limit visible tasks per day with overflow indicators
  - Handle multiple tasks on same date
  - Click-through functionality to detailed day view
- Day View Modal
  - Show all tasks and events for selected date
  - Sort items chronologically
  - Full detail expansion for each item
  - Filter by exact date match using timestamp ranges

4.8 Phone app

The phone app closely mirrors the web app and productivity device in terms of functionality and design. The app will require the user to sign in and just like with the web app, they can add a new device or select a device to view the dashboard of. The most important element being that it includes bluetooth functionality for wifi provisioning.

4.9 Third-Party Integration

Since many other popular applications use a similar system of entry keeping that our user could have already set up how they wanted, our server can retrieve data from these sites and parse them onto our device. Event entries that a standard calendar application would use must have the parameters present within our system to properly dictate an event, that being the start time, duration or ending time, and name. By establishing a proper tool chain for establishing a connection to the API of desired and eligible cloud based calendars and interpreting the entry data into our system, we enable users to link their calendar accounts to our server and seamlessly import their events. Tasks may also be imported into the system from these programs such as from Google Tasks, however since sites like these often lack the priority parameter, the default priority level must be set to a default value to be altered later by the user. Incoming tasks should be set to the lowest priority, which is without any offset, as we would want to avoid providing any unnecessary alterations to the user's work flow without their authority.

4.10 Device Casing

The triangular volume on the back of the device serves as a kickstand and where some of the PCB modules are located. Additionally, the triangular kickstand keeps the center of gravity low, making it difficult to tip over. This space is also necessary to fit two 3500mAh 18650 battery cells which we

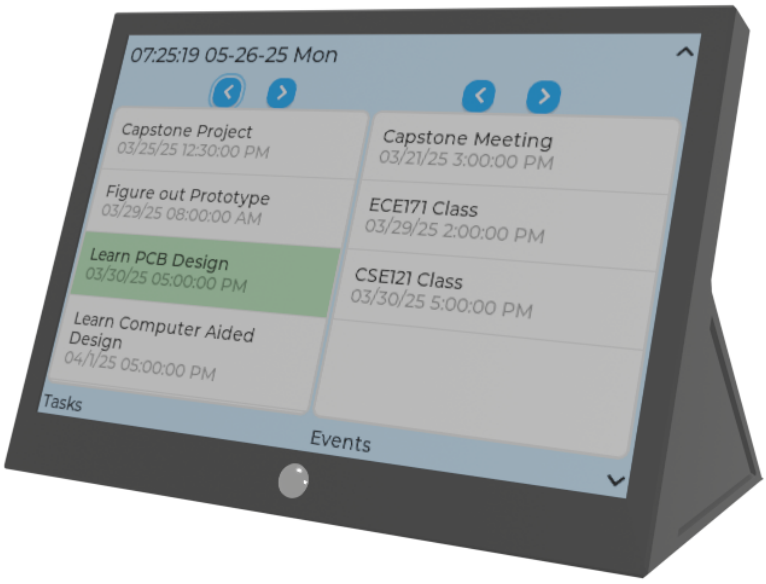


Figure 11: Front render of productivity device

Figure 12: Back render of productivity device

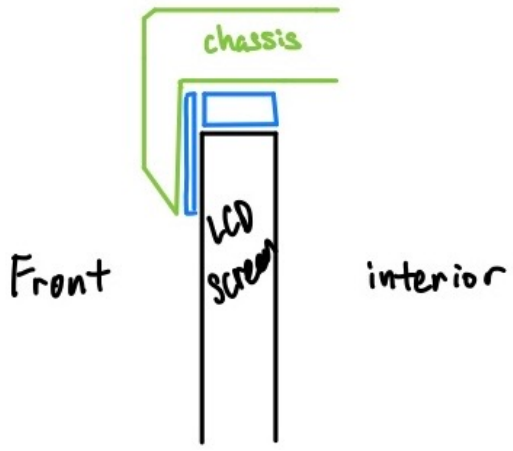


Figure 13: LCD neoprene spacer

selected over a specific pouch battery for repairability and providing room for easy assembly and disassembly.

The screen of the device is tilted at a comfortable viewing angle and includes two slits for the speakers, as well as a power button and the 7.5 inch LCD touchscreen.

As described in design for manufacture and assembly, the chassis will be made of a combination of recycled polymers, bioplastics, and recycled aluminum where metal is necessary. The device must have a screen protector applied to the LCD screen to protect the surface from scratches. Additionally, strips of neoprene are added around the edges of screen as well as the front edge of the screen to serve as a cushion for the screen in the event that the device is dropped. The blue areas shown in Figure 13 represent the neoprene strip placement around the LCD.

The chassis utilizes zinc-plated carbon steel screws to make dissassembly and repair easy. Threaded metal screw inserts will be melted into the plastic chassis as these are more durable and reusable in comparison to threading the screws directly into the plastic housing. The screw holes are covered with small rubber screw hole plugs to hide the screws, and reduce any possibility of dust or debris getting stuck in the screw holes.

4.11 Additional Requirements

Each device will be shipped with a unique serial number (ID) and an 8-digit code for device identification. Alongside the device, a printed start up manual must be included with the device. An example is shown below:

Welcome to your new productivity partner! Start with connecting your device to the provided power supply. When the device displays the following screen, the device is ready to be connected to the internet.

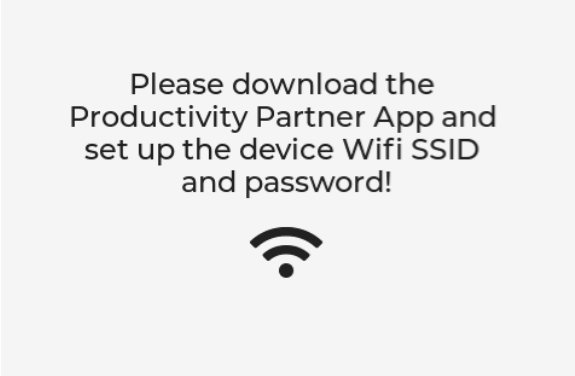


Figure 14: Wifi connection screen

Figure 15: scheduleapp.click QR Code

Download the Productivity Partner App and follow the instructions to connect to your device and enter in the wifi credentials.

Once complete, your productivity partner is ready for your schedule!

Be sure to log in or create an account at `scheduleapp.click` or you can scan this QR code!

Once logged in, you'll see the following screen. Follow the instructions on screen to add your device.

1. Power on your new device and ensure it's connected to Wi-Fi.
2. Find the Device ID on the back of your device or in the settings menu.
3. Click "Add Device" above and enter the Device ID.
4. Give your device a friendly name (e.g., "Kitchen Tablet").
5. Select the device from the list and click "Go to Dashboard" to start managing it.

You can now add tasks, habits, and events via the web app and they will show up on the device!

Be sure to customize your device's timezone and in the settings window of the website! Additional specifications for the manufactured design are described in section 5.2 and section 5.4.

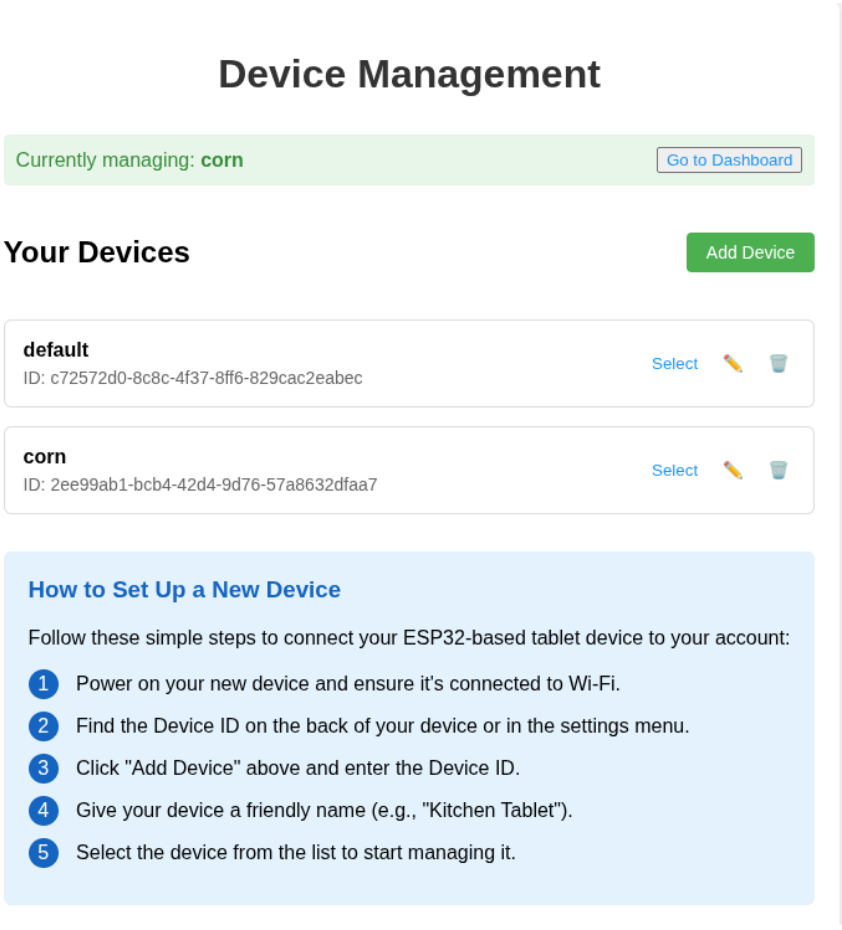


Figure 16: Device setup page

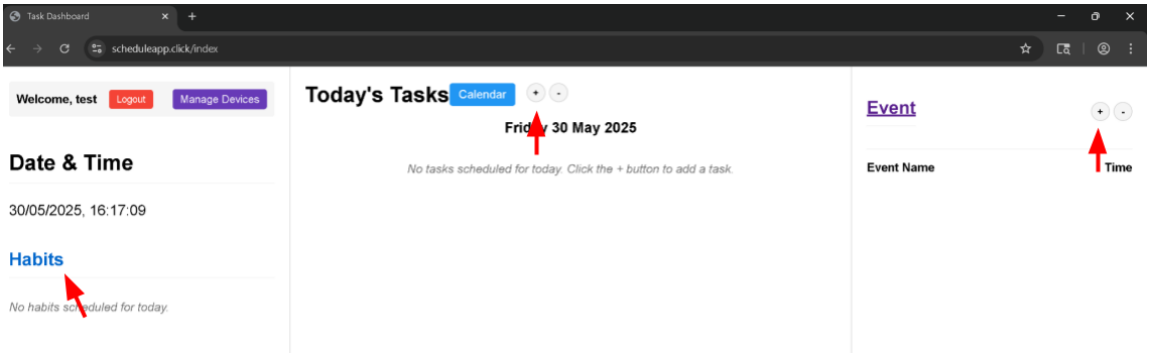


Figure 17: Device Dashboard

4.12 Design for Manufacture and Assembly

1. DFM Emphasis: Minimize part count, ensure ease of assembly, and use sustainable, readily sourced materials that adhere to ESG standards.

2. Component and Part Optimization Simplified System Architecture: Use a low-power SoC that provides enough performance for basic apps without unnecessary complexity. Limit the number of integrated peripherals to only those essential for Wi-Fi connectivity, Bluetooth, and basic I/O. Minimization of Part Count: Integrate functions wherever possible to reduce separate components. Use a unified PCB design that consolidates sensor circuits, display driver circuitry, and memory modules to simplify assembly.

3. Material and Process Selection External Casing: Use a blend of recycled polymers and bio-based sugarcane materials sourced from suppliers in Malaysia/Indonesia that meet ISO 14021 standards. Ensure the casing is designed with minimal thickness while maintaining durability. Internal Structure: Use recycled aluminum for internal structural supports, taking advantage of its low-energy reprocessing. Display Options: Provide two models: an LCD variant and an E-Ink variant for users who prioritize energy efficiency and long life cycles. Design the display mounting system to be common across models, with slight adjustments for backplane electronics.

4. Assembly and Integration Ease of Assembly: Design the device with self-orienting parts and clearly marked alignment features to facilitate automated or manual assembly. Use snap-fit connections and standardized fasteners that reduce the need for adhesives or complex bonding. Modular Design: Develop modules for the display, battery, and main logic board that can be independently assembled and later integrated. Allow for easy disassembly to facilitate repair or recycling. Interconnects and Tolerances: Define clear tolerances for PCB components, connectors, and display interfaces to minimize assembly errors. Include "keyed" connectors to prevent misalignment during the integration of sensors, memory, and driver circuits.

5. Process Considerations and Supplier Collaboration Manufacturing Process: Choose manufacturing techniques that are optimized for high-volume, low-cost production. Specify processes with low energy consumption and minimal waste. Supplier Standards: Work with suppliers who adhere to environmental and social governance (ESG) standards.

6. Quality Assurance and Testing Design for Testability: Integrate test points on the PCB for automated functional checks during assembly. Ensure that all critical interfaces are accessible for in-process quality assurance. Tolerance Verification: Create detailed CAD drawings with precise dimensions and tolerance specifications to be shared with suppliers and manufacturing partners. Implement in-line inspection procedures to verify material quality and assembly integrity.

7. Sustainability and End-of-Life Recyclability: Design the product for ease of disassembly, marking components clearly for recycling. Use materials such as recycled polymers, bio-based plastics, recycled aluminum that have established recycling processes.

4.13 Life Cycle Assessment

High Performance Electronics

- **Objects**: Semiconductor Chips, Memory Modules, Sensors, Display Driver Circuitry, Backplane Electronics
- **Sourcing**: Generally sourced from semiconductor fabs in Taiwan, South Korea, or China.
- **Sustainability**: Material extraction is correlated with high carbon emissions. Manufacturing requires high energy expenditure. Recycling is difficult and often costs a lot of money. We will work with suppliers who adhere to environmental and social governance (ESG) standards.

Device Display

- **Objects**: LCD Variant or E-Ink Variant will be supported as different models of the same product.
- **Sourcing**: LCD model panels are generally sourced in South Korea, Japan, or China. They are generally regulated to optimize for production energy efficiency and reduction of waste. E-Inks are generally sourced from Taiwan from companies such as E-Ink Holdings. These suppliers generally also focus on sustainability.
- **Sustainability**: LCDs have established separation streams that make for easier recycling. E-Ink panels are easier to recycle and have significantly longer life spans relative to LCDs. Both are significantly more energy-efficient and sustainable than OLED displays.

Device External Structure

- **Objects**: A blend of recycled polymers and bio-based sugarcane renewable source
- **Sourcing**: They are generally sourced from Malaysia and Indonesia, which specialize in recycled plastics and exercise ISO 14021 audits and similar standards to verify their recycled content and eco-friendly production processes.
- **Sustainability**: The plastic blend would require less energy and is engineered for easier recyclability.

Device Internal Structure

- **Objects**: Recycled Aluminum
- **Sourcing**: Generally sourced from certified European or American recycling plants.
- **Sustainability**: Reprocessing recycled aluminum requires about 5% of the energy expenditure of newly processed aluminum. This process lowers embodied carbon as well.

Prints On Packaging and Printed Materials

- **Objects**: Vegetable non-VOC Inks — these inks are made from renewable sources and contain minimal volatile organic compound (VOC) concentrations.
- **Sourcing**: These are generally sourced from eco-friendly chemical suppliers in Europe.
- **Sustainability**: Production requires fewer emissions and toxic waste and is easier to recycle.

Packaging

- **Objects**: FSC-Certified Recycled Cardboard (Forest Stewardship Council) and Minimal Bio-Based Plastic Films. The films are used for shrink wrapping in packaging.
- **Sourcing**: FSC-certified recycled cardboard comes from European suppliers, and the bio-based films are generally sourced from Southeast Asia, often from Indonesia and Malaysia.
- **Sustainability**: FSC-certified recycled cardboard boasts a lower carbon footprint. Both require less energy expenditure and are easily recycled.

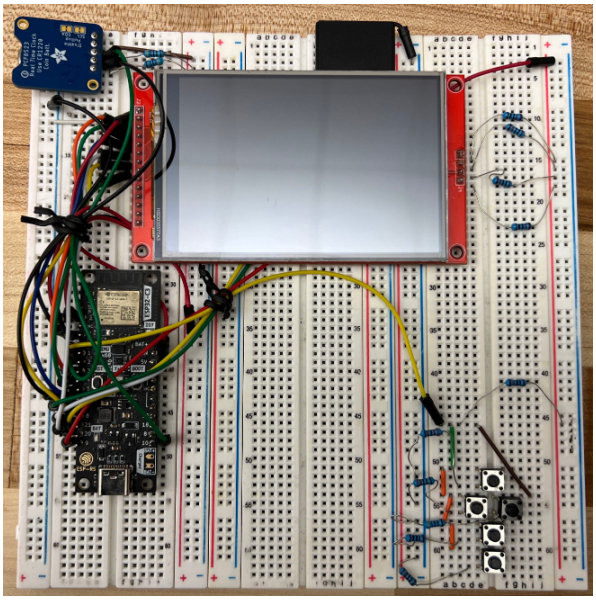


Figure 18: Prototype off

5 Evaluation

5.1 Prototype

5.1.1 Hardware

Images of the prototype are found in figures 18 to 25. The prototype consists of the microcontroller dev-board, an 3.5 inch LCD touch screen breakout board, a real-time clock breakout board, and several tactile buttons for navigating the UI of the device. The wiring diagram for the prototype is described by Figure 26

- **ESP32-C3-DevKit-RUST-1**: A microcontroller board featuring the ESP32-C3 SoC.
- **MSP3520**: An inexpensive, SPI TFT LCD touch screen that comes with an SD card reader.
- **Adafruit 3295**: A breakout board for the PCF8523 real time clock.

5.1.2 Software

The software for the ESP32-C3 was written in the C programming language, with the Espressif IoT Development Framework (esp-idf) for building, flashing, and testing the prototype.

The ESP32-C3 had three major roles to fill in our prototype: the graphical user interface (GUI), the database, and cloud synchronization. The GUI displays each respective type of entry within the database through a page system that the user can navigate using the directional buttons. The database maintains each entry on flash memory, manipulating entry data, and providing organized lists of entry data for the GUI to display. Cloud synchronization sends details about how the user

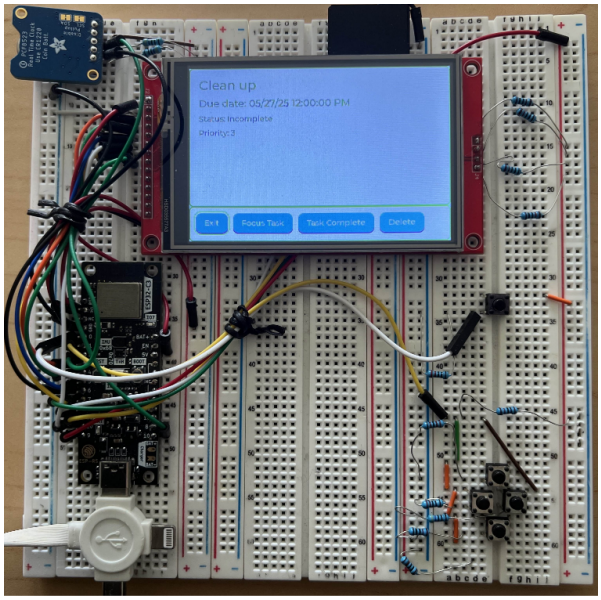


Figure 19: Prototype on

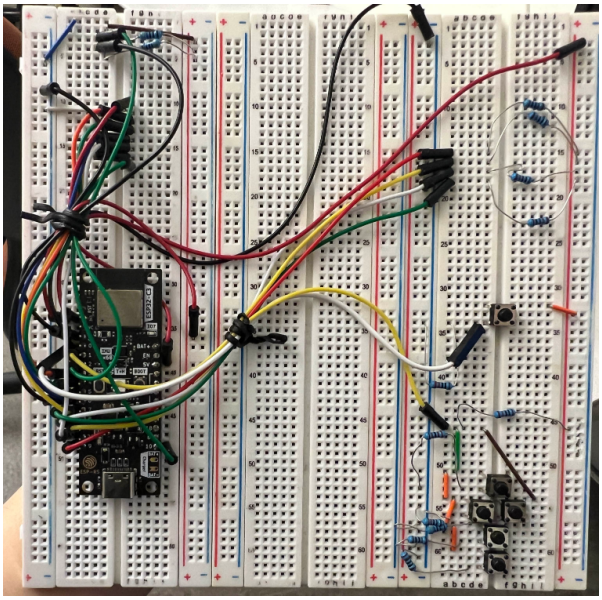


Figure 20: Prototype without screen and RTC

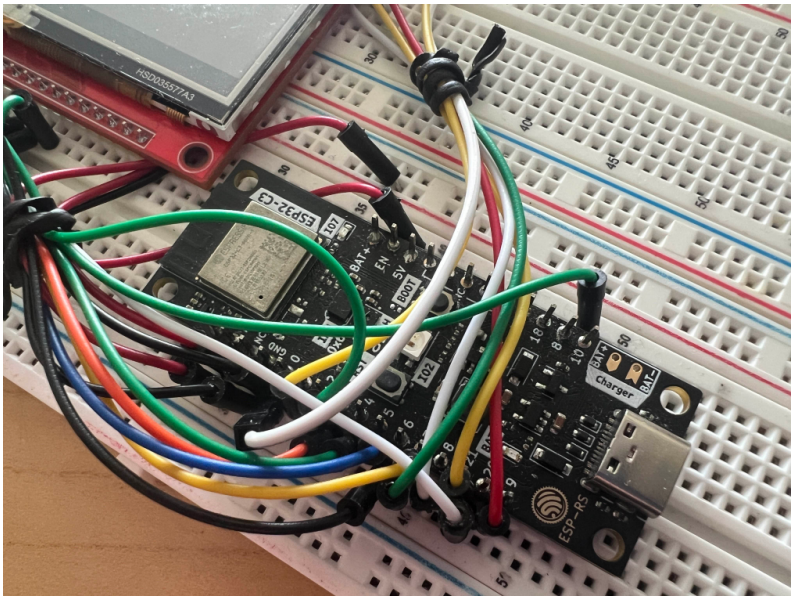


Figure 21: Prototype ESP32-C3 closeup

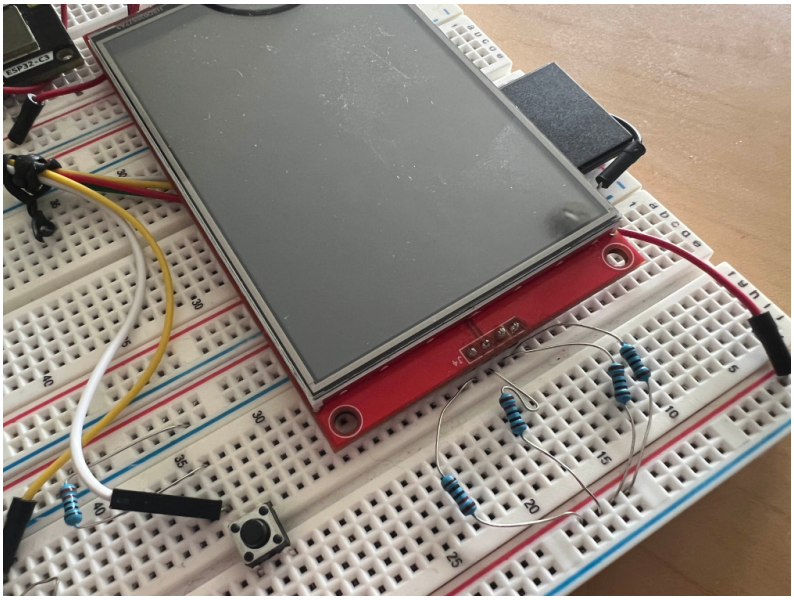


Figure 22: Prototype Screen and sync button closeup

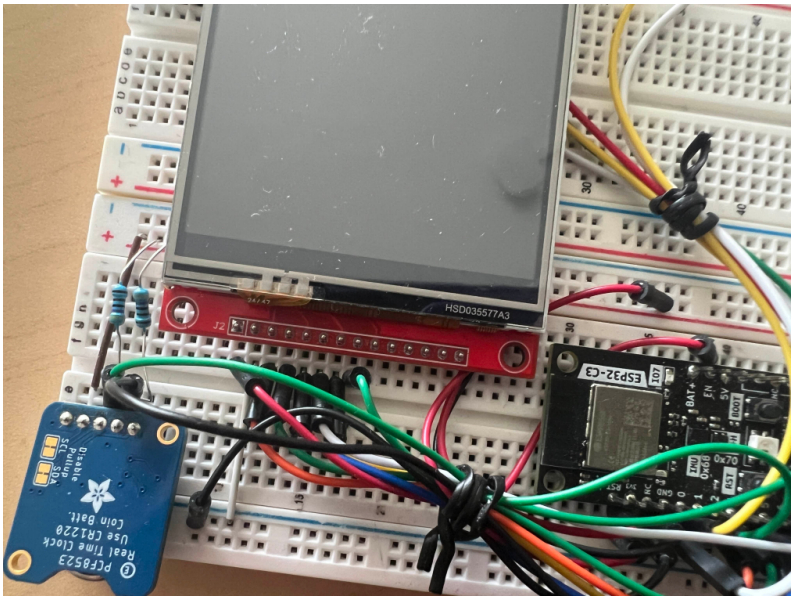


Figure 23: Prototype RTC and screen connections

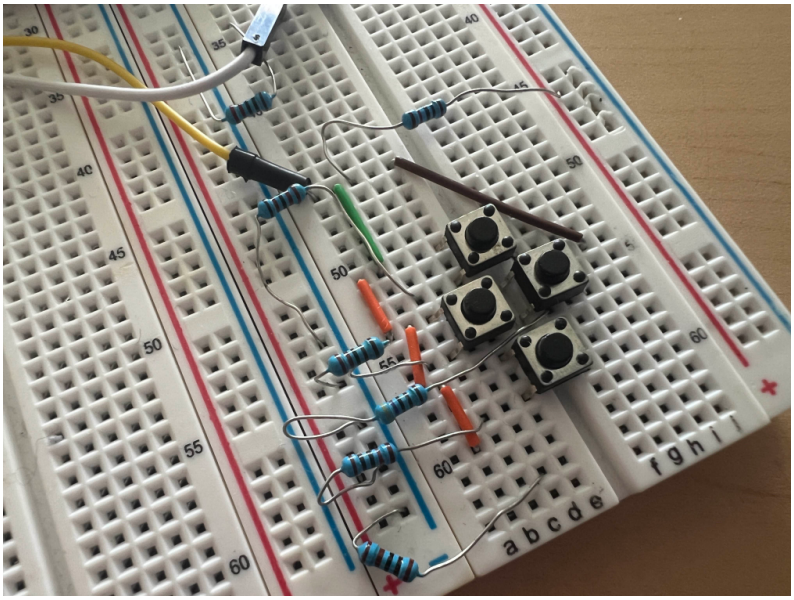


Figure 24: Prototype tactile buttons closeup

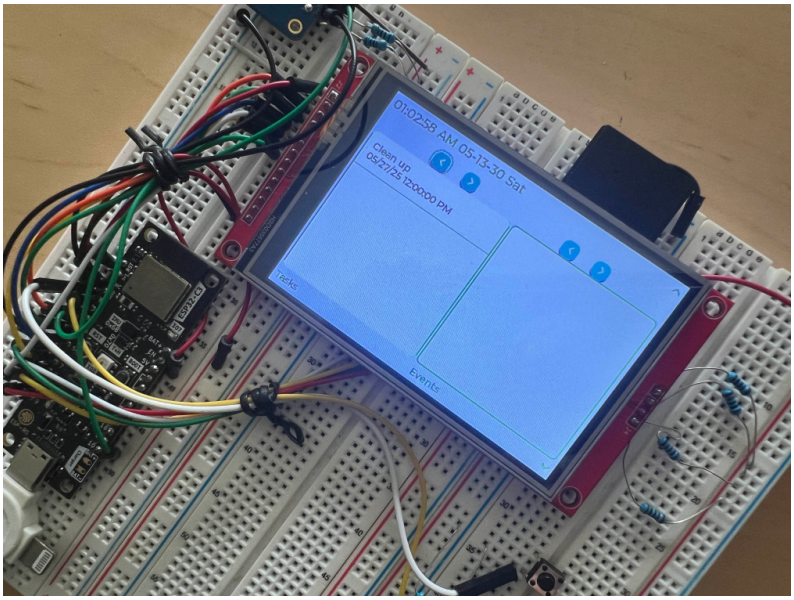


Figure 25: Prototype home screen closeup

modified their entries on the client, then receives a copy of all data within the server's database for the client's database to compare to.

The GUI is the most called system during runtime, which outputs to the device's monitor and inputs from the button interface on each cycle. Monitor communication is established through Serial Peripheral Interface (SPI) protocol, and rendered to using the Light and Versatile Graphics Library (LGVL). This graphics library provided a lightweight and fast C based GUI for display on the screen. Inputs were handled using the esp-iot button component, implementing our buttons for directional navigation, item selection, and on-demand synchronization.

The database component serves as the persistent storage and retrieval system for all user-generated data, including tasks, events, habits, and habit entries. Implemented using SQLite3, it offers a lightweight yet robust solution well-suited for the restricted resources on the ESP32-C3, balancing performance with minimal resource overhead. To conserve memory, the system is designed to keep only the entries relevant to the current GUI screen in active memory. All long-term data is stored within flash memory through the database, allowing for efficient retrieval while maintaining a low runtime footprint. On screen changes, the GUI calls the database to retrieve the relevant entries for display. To improve modularity and reduce processing overhead on the interface side, these entries are returned in a pre-sorted state, ready for direct use by the frontend logic. Furthermore, when a task is marked as complete or deleted, the database is responsible for updating the underlying records accordingly, ensuring consistency between the visual representation and the stored data.

The prototype makes use of the coreMQTT and mbedTLS libraries to establish a secure connection to the MQTT broker. This connection is only established when the user presses the synchronization button, at which case the device will begin the multiple step process of synchronizing the user's data between the client and server. To prevent data races, the user's input is blocked

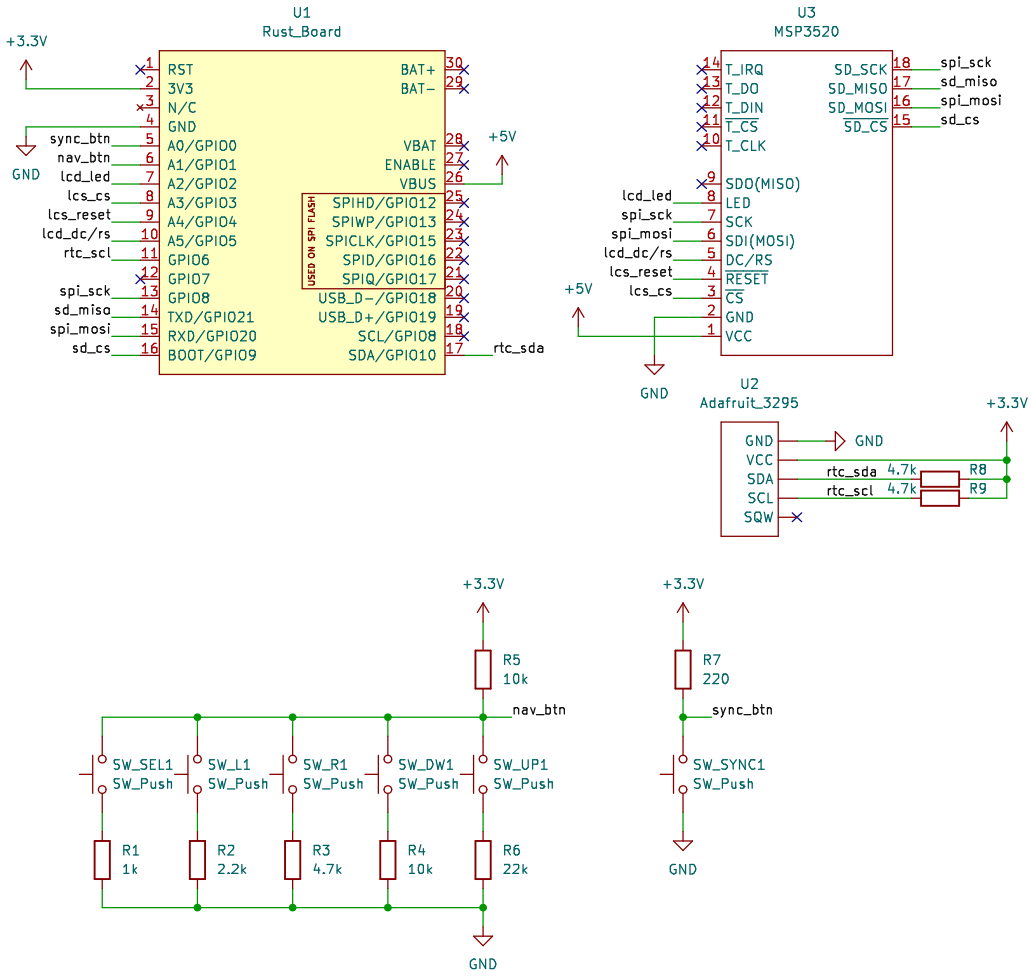


Figure 26: Schematic of the prototype

and a loading popup will display on the monitor until all procedures are complete.

While the GUI and database components interact through random access memory, the cloud and database components interact through flash memory. Since the ESP32-C3 had limited remaining RAM while the MQTT broker was connected, we had to rely on our generous amount of flash memory at our disposal. We used an intermediary system that retained the JSON messages and wrote the plaintext to a separate temporary file, as this would avoid the burden of having to deserialize the message and place it into the relational database while our resources were at their most stressed state. After accepting all messages from the broker, the client could then disconnect, freeing up enough resources for our database to then effectively parse all data left within the temporary file. Sending all server information to the client is the safest and simplest method to ensure the client and server display have identical data sets.

When alterations are made to an entry on the database a similar procedure is performed. When a user performs one of the several possible alterations to an entry or creates a new entry in the habit system, a new corresponding plaintext file is produced holding the relevant information about the change. Correlating each entry with its own file was intentionally done to avoid creating multiple tickets in case the user made several alterations to an entry's state before a cloud synchronization was performed, in which case the alteration file would be continuously overwritten with the most recent change, effectively avoiding any redundant information from being parsed to the server. During cloud synchronization, each of these response files would be serialized into JSON and sent to the server, and upon acknowledgement the file would then be removed from flash memory. This would make sure that the client information is sent at least once. The server will compare the response with its own database, thus sending multiple alteration requests on the same entry will have no effect, and the client will be ensured that the server will have been given the same modification that the user made on the client.

This system of housing all data on the disk between cloud synchronization is highly robust, as all unfinished work will remain within non-volatile memory between power cycles. Furthermore, as our device must be capable of total offline use, all unsent alterations to the database will be preserved until broker connection could once again be secured, regardless of power cycling or later adjustments to the status of an entry by the user.

The device's Wi-Fi configuration is done through a HTTP server running on the device bound to its Wi-Fi access point. The access point allows the user to connect to the device and access a landing page with a form to submit an SSID and password. The device changes between the Wi-Fi connection modes — station only mode (sta), softAP only mode, and softAP + station mode (apsta) — based on its current state. When the device boots, it checks the non-volatile storage (NVS) for an existing connection configuration. Based on the results, it starts the ap or sta mode. Figure 27 shows the steps taken after the device boots. Because the esp_wifi component is based on event callbacks, the provisioning process is able to run without blocking. The HTTP server can be configured to either use a captive server, which redirects all HTTP requests to the login page, or with HTTPS. Unfortunately these features are mutually exclusive, though a newer version of the esp-idf has introduced a feature that is able to do both at the same time with a different technology.

The device is able to communicate with the server using coreMQTT and mbedTLS to establish a connection to the MQTT broker. The main process must initialize the MQTT state, then follow these steps to communicate with the server:

1. Connect to the endpoint and clean any broken sessions that are present.
2. Subscribe to the device's topic and wait for a confirmation from the broker.
3. Publish the desired message to the topic. (Outlined in section 4.5)

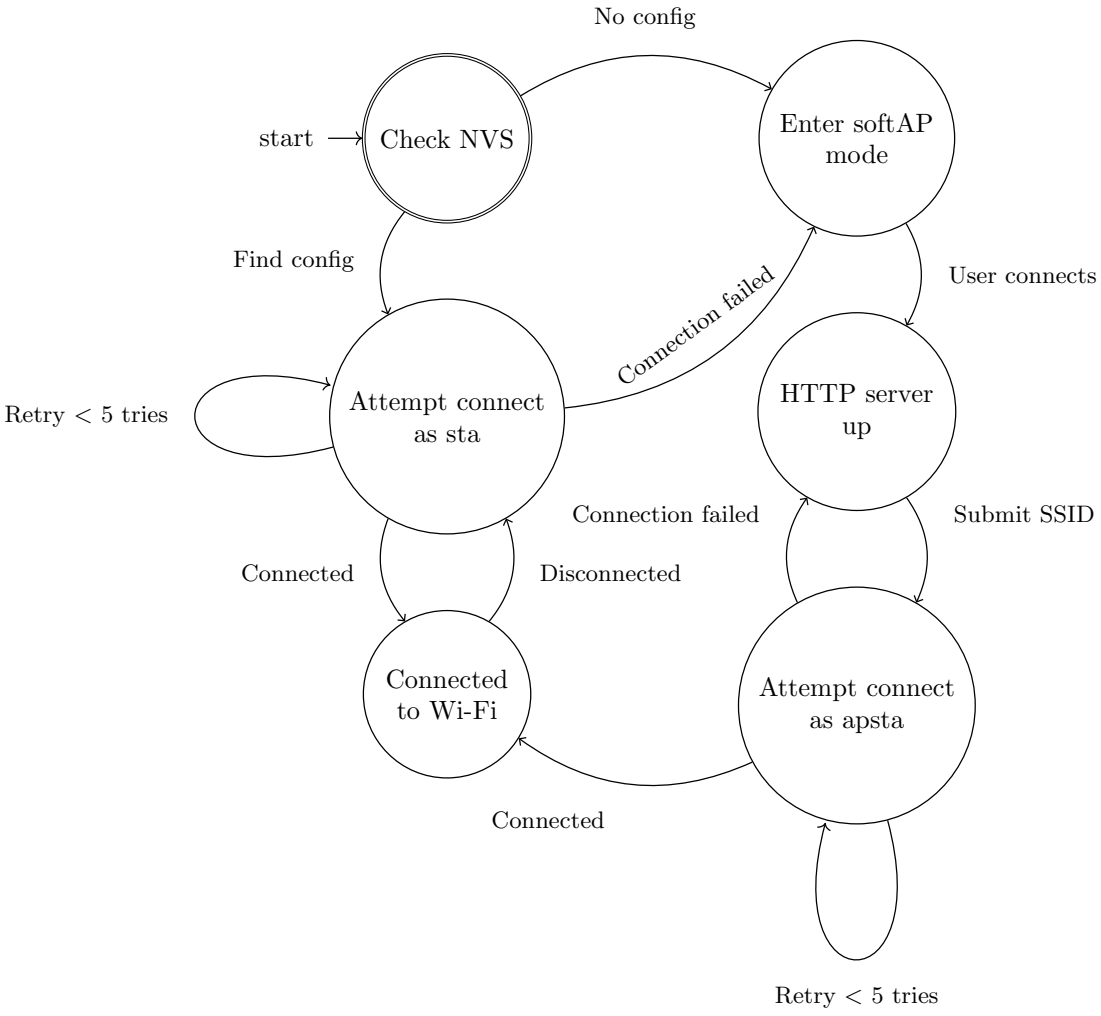


Figure 27: W-Fi provisioning flow chart

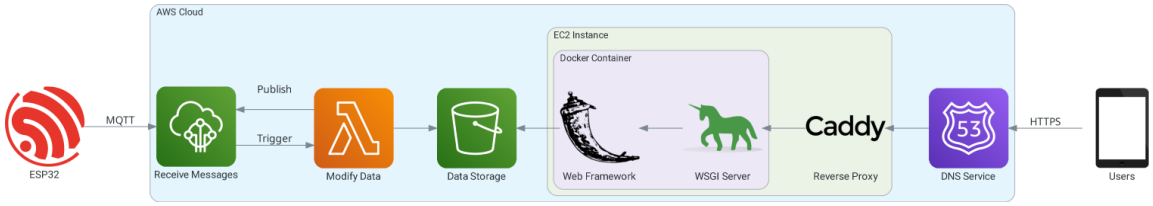


Figure 28: Connection between the prototype device and server

4. Enter the process loop for a specified length of time.
5. Unsubscribe from the topic and wait for an acknowledgement.
6. Disconnect from the endpoint and close the connection.

coreMQTT is created to be used with an event callback structure, so any incoming messages must be handled in the callback as well. In order to access the published payloads from outside of the library's functions, a callback function type may be given to the initialization function. This callback function is defined in the app_main.c file, and is triggered when any message is published on the device's topic. The payload is then saved on disk to be processed later.

5.1.3 Server

Figure 28 shows the structure of the cloud server during the prototyping stage. The server consists of two parts: an MQTT endpoint that processes incoming and outbound messages, and a web server that hosts the web app and processes any changes to the user data. Both components exchange data with the S3 data storage, where the user and device data exists.

The device connects to the AWS IoT MQTT endpoint and subscribes to the topic in the format of `iotdevice/<DEVICE ID>/datas3`. Messages posted to the topic are routed based on the following rules: `SELECT id, type FROM 'iotdevice/+/datas3' WHERE action='refresh' AND (type='task' OR type='habit' OR type='event')` for requesting a refresh of the device's backup, and `SELECT * FROM 'iotdevice/+/datas3' WHERE action = 'update'` for requesting making changes to entries. Each rule triggers a lambda function that retrieves the device's backup from the S3 device backups bucket and processes it as necessary. The refresh retrieval function publishes each entry as a single message. The update function reads the contents of the incoming message and makes changes to the entry of the same ID, then writes the new data back to S3.

The web server is hosted on a AWS EC2 instance deploying Caddy, gunicorn, and flask. Caddy is a web server that automatically generates and renews TLS certificates. The Caddy server is configured with a Caddyfile to act as an HTTPS reverse proxy for our flask server running over HTTP. Caddy communicates with the client through HTTPS, and forwards requests to the docker container that contains the gunicorn and flask server. The docker container is configured with a Dockerfile and credentials stored in a `.env` file. Flask is used in conjunction with boto3 — the Python AWS SDK — to serve the web app and to fetch user and device data.

5.2 Manufactured Design Improvements

5.2.1 Hardware

Presented below are suggestions for selecting hardware components that overcome limits and design issues found in the prototype. Suggestions are based on products that exist at the time of writing.

- Microcontroller:
  - Baseline: ESP32-C3 - 32-bit RISC-V core, 400 KB SRAM, 22 GPIOs
  - ESP32-S3 series - Xtensa LX7, 512 KB SRAM, 45 GPIOs
  - STM32H7 series - ARM Cortex-M7 & -M4, 564 KB to 1.4 MB RAM, 46 to 168 GPIOs
- Development boards:
  - Raspberry Pi Zero 2W - ARM Cortex-A53, 512MB of SDRAM
  - Teensy 4.1 - ARM Cortex-M7, 1024K RAM

The development kit used in the prototype ran into several limits: the number of available GPIO pins, and the memory availability. At compile time, roughly 188kB of static memory was in use, with 133kB remaining for dynamic memory. When in use, the device had very little remaining free RAM which would be occupied when performing synchronization. The prototype ran into memory allocation issues when running the basic program loop and in order to get all of the components working simultaneously, wifi and graphics buffers were reduced heavily, resulting in a much slower UI and synchronization process. A microcontroller with more RAM is necessary to operate the device without making compromises to the program in order to save on memory usage. The prototype also did not make use of multithreading, running entiirely on one thread which meant much less performance and the user could not interact with the device during synchronization. The manufactured design will utilize multithreading, making for a much smoother user experience. SoCs like the STM32H7 with integrated graphics IP may be beneficial for better screen rendering performance.

When synchronizing, the device would recieve the entirety of the database, even if the content wasn't changed. An HTTP server that responds to "changes since X" is a more efficient approach. The prototype did not have enough GPIO pins to make use of touch controls, thus it uses push-buttons for navigation. More GPIO pins are necessary for enabling touch controls, along with implementing more features such as sound and vibration output. When synchronizing, the prototype would connect, recieve all of its data, then disconnect from the server, which made the process much slower. The manufactured design will utilize a persistent internet connection. The prototype used SoftAP, but for the manufactured design bluetooth Wifi provisioning using a phone app makes for a better user experience.

- A higher resolution screen with more colors and capacitive touch input, as long as the communication protocol is supported by the controller.
- The real-time clock did not cause any difficulties in the design. The same module may be used.

5.3 Testing

5.3.1 Test Plan

Some tests require programs to automate usage of the productivity device. This testing suite should use pytest, which can compare expected output values directly to values from the microcontroller. The ESP-IDF environment includes the unit testing framework Unity which allows for easy creation, execution, and repeatability of unit tests.

Some tests can be performed on the prototype, but additional testing on the manufactured product will have to be performed as well. The following are tests that will be performed on both the prototype and the manufactured product:

- Boot time
- Data cloud synchronization
- Many Users to One Device
- Device Sync after Offline Use
- Persistence of data after power-off
- User data restoration on new device/Many devices to one user
- Secure connection
- Modifying entries to and from the device.
- Sync time
- Wifi Connection
- Cloud synchronization updates: basic
- Cloud synchronization updates: dual updates
- Cloud synchronization updates: undoing edits

The following are tests should be performed on samples of the manufactured product to ensure its construction quality and additional features beyond the prototype.

- Battery Life
- Battery Life Cycle
- Charge Speed
- Drop Testing
- Electromagnetic interference and electromagentic compatability
- Factory Reset verification
- LCD functionality Verification
- Haptics and Sound functionality Verification
- Port Durability
- Software Update verification
- Tensile Strength testing
- Thermal Cycling: Cold and Dry Heat
- Thermal Shock: Air to Air
- UV Exposure Testing
- Qualitative analysis should also be performed to determine if the device can positively affect productivity where a user would be provided the device and have it placed on their workspace for approximately two to three hours, afterwards they would be given a questionnaire that would ask users to rate on a scale of -5 to 5 to determine how much the device improved their workflow.

The following describe a series of testing programs and their expected behaviors necessary for certain tests. The host computer should be able to connect to the productivity device using a JTAG debug interface on the PCB.

test_sound.py - plays a sound from the speakers of the device and activates the vibration motors to verify these components work as expected.

test_screen.py - Screen displays a white background with different red targets that can be tapped to verify that the screen and touch functionality are working as expected.

test_batterylife.py - disables automatic power saving mode on low battery and runs a series of functions to emulate daily use of the device unsupervised. It will also begin a timer from the start of the test to when the device is unresponsive.

test_battery_cycling.py - cycles the device through 1000 battery charge and discharge cycles while monitoring the battery life. It also runs a series of functions to emulate daily use of the device unsupervised. Test is complete after 1000 battery charge cycles.

test_SoftwareUpdate.py - connects to the server to deliver a test version of the UI to be updated to the device to demonstrate the software update functionality.

test_clean.py - reads the device SD card and verifies that it is empty

function_test.py - performs a series of functions to emulate daily use of the device unsupervised. It should connect to the server and add various tasks and events, verify that they're updated on the device, perform focus mode on different tasks, and update habits.

5.3.2 Tests

5.3.3 Boot time

Introduction: One of the key design goals of this device was for the boot time to be under 3 seconds. This test verifies this requirement.

Scope: device functionality
Apparatus: productivity device, timer
Independent variables: device powering on
Dependent variables: power on time
Procedure:

1. Begin a timer right as the device is powered on. End the timer as soon as the screen turns on and awaits interaction.

Expectation: Within 3 seconds, the device should be booted.

5.3.4 Data cloud synchronization

Introduction: Cloud synchronization and wifi connection are one of the key components of productivity device features and must be tested to ensure full functionality.

Scope: flash storage, file operations, cloud operations

Apparatus: productivity device, laptop or mobile device for web app interaction

Independent variables: cloud data storage, flash storage

Dependent variables: File contents (database), power on behavior (planned)

Procedure:

1. Power on the device and verify it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials.
2. Create a test user on the web app.
3. Log into the user via a laptop or mobile device.
4. Add the device to the user using the corresponding device ID.
5. Add a task, event, and habit to be written to the device. Change the time setting from 12 hour to 24 hour display.
6. Check the device to verify that the new schedule data has been added and the settings have been changed.

Expectation: The device should be updated with the new schedule and setting information after syncing.

5.3.5 Many Users to One Device

Introduction: If, for example, two parents wanted to monitor their child's device, they would both need to access the data on one device. This test verifies the capability of editing and sending new tasks to the same device from two separate accounts.

Scope: flash storage, file operations, cloud operations
Apparatus: productivity device, laptop or mobile device for web app interaction
Independent variables: cloud data storage, flash storage
Dependent variables: File contents (database), power on behavior (planned)
Procedure:

1. Power on the device and verify it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials. If the test already has existing user information on it and is set up, skip to step 7.
2. Create a test user on the web app.
3. Log into the user via a laptop or mobile device.
4. Add the device to the user using the corresponding device ID.
5. Add a task, event, and habit to be written to the device. Change the time setting from 12 hour to 24 hour display.
6. Check the device to verify that the new schedule data has been added and the settings have been changed.
7. Create a second test user on the web app.
8. Log into the second user via a laptop or mobile device.
9. Add the device to that user using the corresponding device ID.
10. The device dashboard should show the earlier tasks that were created from the first user account.
11. Add another set of tasks, events, and habits to be written to the device. Change the time setting from 24 hour to 12 hour display.
12. Check the device to verify that the new schedule data has been added and the settings have been changed.

Expectation: Both users should be able to view and edit the same set of schedule information that is synchronized to the device.

5.3.6 Device Sync after Offline Use

Introduction: One of the key features of the productivity device is its offline usage and that the data from the device will be synchronized to the cloud after reconnection to the internet.

Scope: flash storage, file operations
Apparatus: productivity device, host machine for flashing and running tests
Independent variables: The flash device and model
Dependent variables: File contents (database), file size (bytes)
Procedure:

1. Power on the device and verify it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials. If the device already has existing a schedule and is registered under a user, skip to step 7.
2. Create a test user on the web app.
3. Log into the user via a laptop or mobile device.
4. Add the device to the user using the corresponding device ID.
5. Add a task, event, and habit to be written to the device. Change the time setting from 12 hour to 24 hour display.
6. Check the device to verify that the new schedule data has been added and the settings have been changed.
7. Disconnect the device from the internet in the settings menu. Return to the main menu and ensure the wifi symbol is not displayed.
8. Navigate around the device and check that the earlier data is still visible on the device.
9. Interact with the device by setting one of the tasks to complete, deleting an event, and updating a habit. Wait 10 minutes.
10. Reconnect the device to the internet by navigating back to the settings and turning the wifi back on.
11. Refresh the web app after 10 seconds. The modified tasks, events, and habits should now be updated on the web app.

Expectation: The modified tasks, events, and habits should be updated on the web app after reconnection to the device within 10 seconds.

5.3.7 Persistence of data after power-off

Introduction: One of the key features of the productivity device is its offline usage, meaning that data must persist after being turned off and without internet access.

Scope: flash storage, file operations

Apparatus: productivity device, host machine for flashing and running tests, laptop or mobile device for web app interaction

Independent variables: Device data and file contents

Dependent variables: File contents (database), file size (bytes), power off behaviour (planned, power-loss)

Procedure:

1. Power on the device and verify it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials. If the device already has existing a schedule and is registered under a user, skip to step 7.
2. Create a test user on the web app if it does not already exist.
3. Log into the user via a laptop or mobile device.
4. Add the device to the user using the corresponding device ID.
5. Add a task, event, and habit to be written to the device. Change the time setting from 12 hour to 24 hour display.
6. Check the device to verify that the new schedule data has been added and the settings have been changed.
7. Disconnect the device from the internet in the settings menu. Return to the main menu and ensure the wifi symbol is not displayed.
8. Navigate around the device and check that the earlier data is still visible on the device.
9. Remove power to the device, wait 10 seconds before restoring power.
10. Navigate around the device and verify that the earlier data has not changed.

Expectation: All file contents stay the same before and after powering off.

5.3.8 Many devices to one user

Introduction: If a user intends on controlling multiple devices, they must be able to add and edit a second device on their account.

Scope: flash storage, device operations

Apparatus: 2x productivity devices, one with existing user data and one clean device, laptop or mobile device for web app interaction

Independent variables: Device data and file contents

Dependent variables: Existing user data on web app

Procedure:

1. Power on the clean device and verify it is completely clean of all user information. Check it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials.
2. Log into an existing test user via a laptop or mobile device.
3. Add the device to the user using the new device ID. Confirm that the ID of the new device shows up under user devices.
4. Add a task, event, and habit to be written to the new device. Change the time setting from 12 hour to 24 hour display.
5. Navigate the new device to verify that the user's schedule data has been added and the settings have been changed.

Expectation: New device displays new schedule contents to the original device, all using the same user account.

5.3.9 Secure connection test

Scope: cloud operations, security

Apparatus: productivity device, host machine for flashing and running tests, cloud server access
Independent variables: device, firmware
Dependent variables: security protocol, device certificate
Procedure:

1. Verify the device is connected to the internet.
2. Attempt connecting to the cloud endpoint using a url with `http://`
3. Ensure that the connection forces `https://`
4. Test adding a task.

Expectation: The connection must be made with ssl.

5.3.10 Sync time

Introduction: The productivity device periodically synchronizes its data with the server. One of the key design goals of this device was for the sync time to be under 10 seconds.

Scope: flash storage, file operations, cloud operations
Apparatus: productivity device, laptop or mobile device for web app interaction, timer
Independent variables: cloud data storage, flash storage
Dependent variables: File contents (database), power on behavior (planned)
Procedure:

1. Power on the device and verify it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials.
2. Create a test user on the web app.
3. Log into the user via a laptop or mobile device.
4. Add the device to the user using the corresponding device ID.
5. Add a task, event, and habit to be written to the device.
6. Begin a timer when the device starts to sync and stop it once the device is updated with the new data.

Expectation: Within 10 seconds, the device should be updated with the new schedule and setting information.

5.3.11 Wifi Connection

Introduction: Since the device does not have a keyboard for inputing wifi information, the user must connect to the device access point using a laptop or mobile device to enter in this information. The following test checks the functionality of this feature.

Scope: wifi connection, access point
Apparatus: productivity device, laptop or mobile device for web app interaction
Independent variables: device wifi access
Dependent variables: Wifi information
Procedure:

1. Power on the device.
2. Check if it is connected to the internet by looking for the wifi symbol in the top right. If the wifi symbol is displayed, either turn off the wifi router, or enter the device settings and select factory reset. Allow the device a few minutes to reset to factory settings.
3. If the wifi symbol is not displayed, open a laptop or mobile device and look for an access point available for connection called "Productivity Partner". Connect to the access point and enter the password listed on the bottom of the device.
4. A captive portal should automatically open on your device. Enter in the wifi information in the text boxes.
5. Verify that the captive portal displays "OK".
6. Check if the productivity device is connected to the internet by looking for the wifi symbol in the top right. The device's time should also change to be updated with the current time.

Expectation: Device is connected to the internet after connection to the device access point and wifi information is entered.

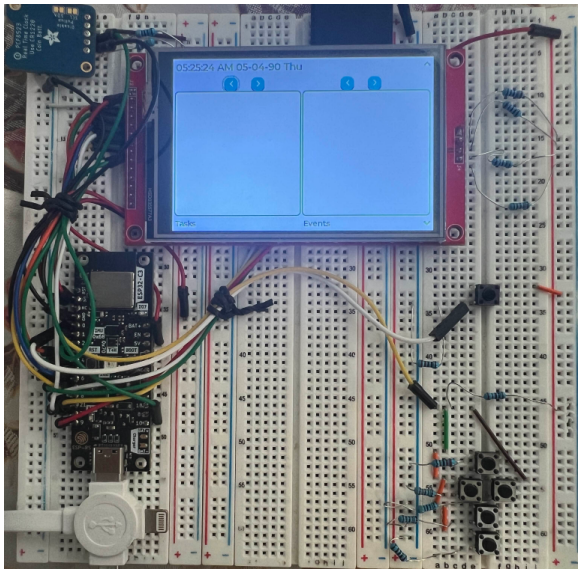


Figure 29: Clean device with no schedule info

5.3.12 Test Results

5.3.13 Boot time test report

Test Purpose: This is intended to verify that the device is capable of booting within 3 seconds as is described in our initial design objectives.
Test Procedure: Exact testing procedures can be found at 5.3.3. Results are visually observed.
Test Results:

- Timer was begun as the device was plugged in and stopped when the device screen was fully refreshed
- Timer was stopped at 1s 38ms

Final Analysis: The device passes, since it booted within the alloted 3 seconds.

5.3.14 Wifi Connection test report

Test Purpose: This is intended to verify that the device is capable of connecting to the internet.
Test Procedure: Exact testing procedures can be found at 5.3.11. Results are visually observed.
Test Results:

- Device was booted with no information to connect to the internet as shown in fig. 29
- Connected to prototype device via SoftAP as shown in the fig. 30 and entered in wifi credentials
- After pressing the synch button, the device displayed a wifi symbol at the top right.

Final Analysis: The device passes. It is able to connect to wifi as expected.

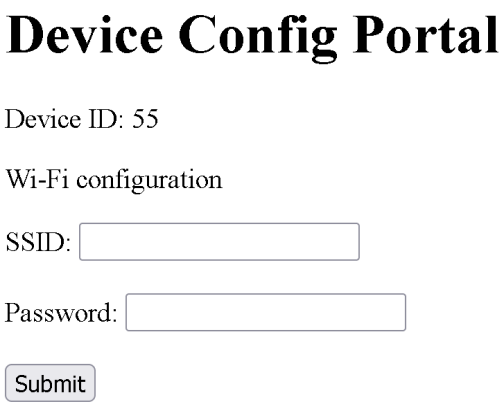


Figure 30: Captive Portal

5.3.15 Secure connection test report

Test Purpose: This is intended to verify that the device connection to the server is secure and encrypted.
Test Procedure: Exact testing procedures can be found at 5.3.9. Results are visually observed.
Test Results:

- Device had existing wifi credentials on it from 5.3.14 so test is begun from step 2
- Using a laptop, a connection to the MQTT broker endpoint using 'http' was made
- The connection forced a https connection, ensuring the connection was secure

Final Analysis: The endpoint connection passes. The connection always forces https.

5.3.16 Data cloud synchronization test report

Test Purpose: This is intended to verify that the device is capable of connecting to the internet and synchronizing with server information.
Test Procedure: Exact testing procedures can be found at 5.3.4. Results are visually observed.
Test Results:

- Device had existing wifi credentials on it from 5.3.14 so test is begun from step 2
- Logged into scheduleapp.click using an existing test account as shown in the fig. 31
- test user has no devices listed as shown in fig. 32 so the test device was added as shown in fig. 33
- Dashboard was empty, so added multiple test tasks, habits, and events as shown in fig. 34

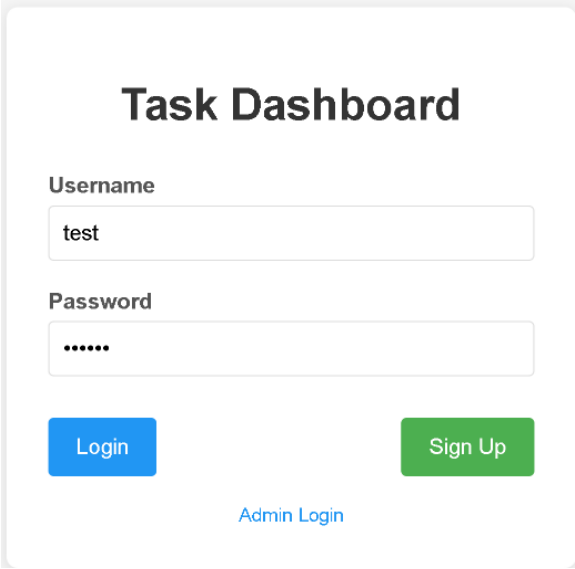


Figure 31: Logging in with test user credentials

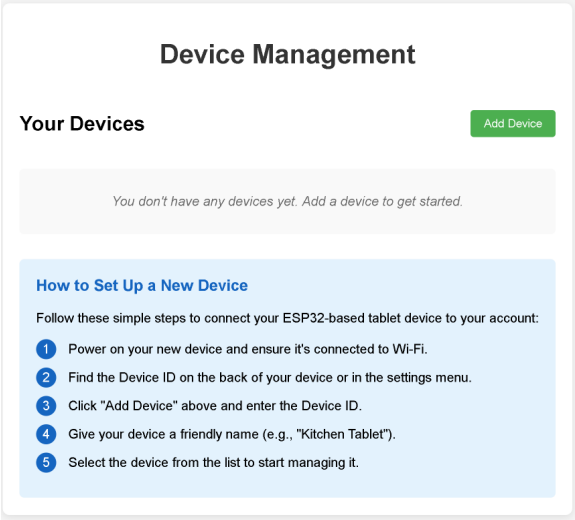


Figure 32: No devices under the test user

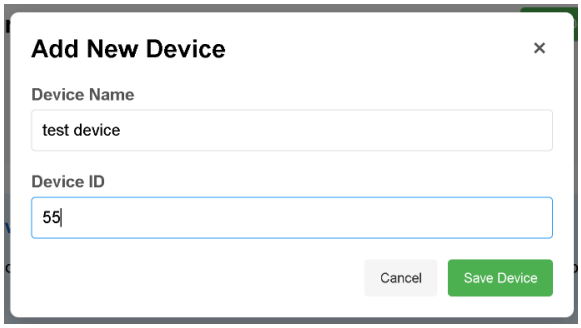


Figure 33: Adding device under test user

Figure 34: Adding a test schedule

- After pressing the synch button on the prototype, the prototype updated and displayed the new schedule information

Final Analysis: The device passes. It is able to connect to wifi and display user information from the server.

5.3.17 Many Users to One Device test report

Test Purpose: This is intended to verify that the device is capable of many users to one device.
Test Procedure: Exact testing procedures can be found at 5.3.5. Results are visually observed.
Test Results:

- Device had existing user data on it from 5.3.16 so test is begun from step 7
- Created a second user and logged in as shown in fig. 35
- added the same device ID
- upon entering the device's dashboard, all of the schedule information from the initial account are visible and editable as shown in fig. 36
- In the second account, added new tasks and events as shown in fig. 37
- After pressing the sync button on the device, the device updated with the new tasks added from the second user

Final Analysis: The device passes. The device is able to display schedule information for both users and the web app also shares the information between the two users.

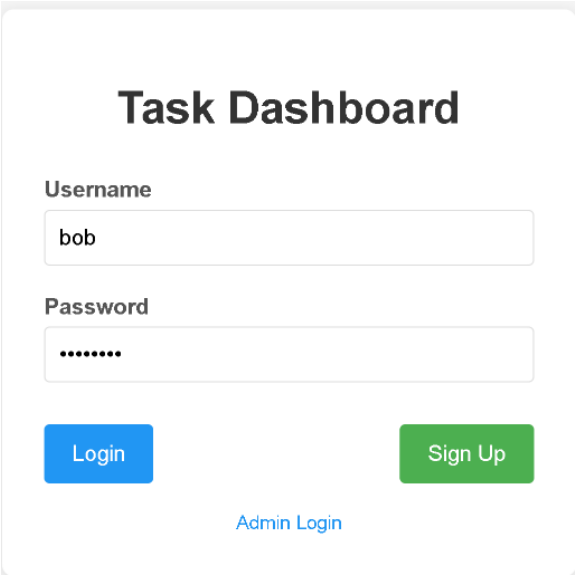


Figure 35: Logging into second user

Figure 36: Same content displayed to second user

Figure 37: New information displayed

5.3.18 Device Sync after Offline Use test report

Test Purpose: This is intended to verify that the device is capable of operation when not connected to the internet and that the server will update with device changes after reconnecting to the internet.
Test Procedure: Exact testing procedures can be found at 5.3.6. Results are visually observed.
Test Results:

- Device had existing user data on it from 5.3.16 so test is begun from step 7
- Device was disconnected from the internet by powering off the hotspot it was connected to
- After pressing the synch button, the wifi symbol in the top right disappeared
- Test task was focused for 9 minutes and 5 seconds before setting the task as complete
- After pressing the synch button on the prototype, the web app updated and displayed the task as complete.

Final Analysis: The device passes. It is able to operate disconnected from the internet and any changes are updated to the server after synchronization.

5.3.19 Persistence of data after power-off test report

Test Purpose: This is intended to verify that the device is capable of retaining user data after shutdown without internet connection.
Test Procedure: Exact testing procedures can be found at 5.3.7. Results are visually observed.
Test Results:

- Device had existing user data on it from 5.3.16 so test is begun from step 7
- Device was disconnected from the internet by powering off the hotspot it was connected to
- After pressing the synch button, the wifi symbol in the top right disappeared
- Device was then unplugged for 10 seconds before plugging the power cable back in
- After the device powered on, device was confirmed to not be connected to the internet by visually checking for the wifi symbol
- The device displayed the same user data as before it was powered off

Final Analysis: The device passes. It is able to operate retain user data after losing wifi connection and being powered off and on.

5.3.20 Many devices to one user test report

Test Purpose: This is intended to verify that multiple devices can be added to one account and that both devices update accordingly.
Test Procedure: Exact testing procedures can be found at 5.3.8. Results are visually observed.
Test Results: - New device was booted with no information to connect to the internet - Connected to prototype device via SoftAP - After pressing the synch button, the device displayed a wifi symbol at the top right. - Logged into scheduleapp.click using an existing test account - Added clean device to user - Dashboard was empty, so added multiple test tasks, habits, and events - After pressing the synch button on the new productivity device, the prototype updated and displayed the new schedule information.
Final Analysis: The device passes. It is able to connect to wifi and display information from a user with an existing device.

5.3.21 Cloud sync: basic edits

Test Purpose: To test edits to a task from the device.
Test Method: Add an incomplete task on the web app tasks pane. Press the sync button on the device to get the task from the cloud. Mark the task as completed on the device. Press the sync button on the device. Verify that the task is complete on the web app.
Test Results: The web app shows a completed task.
Final Analysis: The web server is able to handle requests for modifying a task coming from the device.

5.3.22 Cloud sync: simultaneous edits

Test Purpose: To test concurrent edits to a task from the device and web app.
Test Method: Add a task on the web app tasks pane. Press the sync button on the device to get the task from the cloud. Turn off the Wi-Fi access point that the device is connected to temporarily. Mark the task as deleted on the web app, and completed on the device. Turn on the Wi-Fi access point, and reconnect to it from the device (use provisioning if needed). Press the sync button on the device.
Test Results: The device has deleted the task, and the task does not show up on the web app.
Final Analysis: The device is able to handle when the completion status of a task is different locally and on the cloud.

5.3.23 Cloud sync: undoing edits

Test Purpose: To test edits to a task from the device.
Test Method: Add an incomplete task on the web app tasks pane. Press the sync button on the device to get the task from the cloud. Mark the task as completed on the device. Press the sync button on the device. Verify that the task is complete on the web app. Mark the task as incomplete on the device. Press the sync button on the device. Verify that the task is incomplete on the web app.
Test Results: The web app shows that the corresponding task in not complete.
Final Analysis: The web server is able to handle requests for incompleting a completed task.

5.3.24 Sync time test report

Test Purpose: This is intended to verify that the device is capable of synchronizing within 10 seconds as is described in our design objectives.
Test Procedure: Exact testing procedures can be found at 5.3.10. Results are visually observed.
Test Results:

- Device had existing user data on it from 5.3.16 so test is begun from step 6
- After pressing the synch button on the prototype, timer was begun
- Timer was stopped once the loading screen was full cleared from the screen, taking 28s and 58ms.

Final Analysis: The device failed. It was not able to complete syncing within the alloted 10 seconds.

5.4 Feasibility of Design

After running through multiple tests using the prototype device, the manufactured design was demonstrated to be a practical product. Wifi and server connection as well as basic device functionality were shown to work. With device processor and memory improvements, additional features will be added to further expand the productivity device's capabilities. Referencing our initial A.1design objectives, we revised the specifications of the manufactured design as shown in the table below.

Design Objective	Units	Target/Range
Price	USD	$150
Devices per user	Units	10
Users per device	Units	10
Cloud sync time	Seconds	100m
Simultaneous entries	Entries	16000
Database Size	Bytes	32M
Standby Battery life	Hours	168
Wakeup time	Seconds	100m

Assuming entries are at most 2k in size, (32 * (1024 * 1024)) / (2048) = 16384. Rounding down to 16000 leaves 768k for other storage uses.

Appendices

A Problem Formulation

A.1 Design Objectives

1. Presence: the user wants to be reminded of it, but not too much such that they are distracted by it.
2. Portability/ease of set up: the device will be portable, providing an alternative way to view your schedule without relying on your phone. The device although offline when on the go, should still display data from when it was last connected to the internet
3. Configurable: customizable for the user's habits and workflow. With integration from existing digital calendar and productivity tools such as Google Calendar.
4. Accessible: Schedule and settings are visible and configurable on the go, without the need for the device. Also allowing for the data backup if the device is broken.

Design Objective	Units	Target/Range
Price	USD	< $100
Device per user	Units	Unlimited
User per device	Units	Unlimited
Cloud sync time	Seconds	< 10
Simultaneous tasks	Tasks	Up to 100
Database Size	Bytes	Up to 60 GB, non-volatile
Battery life	Hours	˜200
Wakeup time	Seconds	< 3

A.2 Conceptualizations

User Input	Output	Connectivity	Form Factor	Power
Physical Buttons	Speaker	WiFi	Watch/Pocket	Battery Powered
Touchscreen	LCD/EINK	SD Card	Tablet/Puck	Lithium Rechargeable
Camera	Haptic Feedback	Bluetooth	Desk/Chair	Solar
Microphone	Web Interface	USB	House	Crank Powered
Web Interface	Lights	Remote Controlled	Drone	AC Adapter

A.3 Brainstorming

A.3.1 Methodology

We came up with the following existing methods of time management:

- Planner
- Accountability

- Calendar
- Personal assistant
- To-do list
- Post-it notes/index cards
  - Voice memo
- Gantt chart
- Journaling
- Notification
  - Noise
  - Pop-up/visual
  - Haptic feedback
- Routines
  - Habit building
- Vlog
- Reduce distractions
  - White noise
  - Music
  - Pomodoro/time block
- Gamifying tasks
  - Streak
  - Level up
  - Fun buttons/crank
    - Fidget toy

A.3.2 Time Management Hardware (Ideas)

We came up with the following design ideas that map to at least one of the methods above:

- PDA device
- Focus chair
  - Aromatherapy
- Tablet-sized digital planner
- Virtual reality
- Software-paired device
  - Upload data to device
  - Reduce distractions
  - Tracks phone usage metrics
- Face-tracking sprayer
- A taser/baseball bat
- Game device

- Focus desk
- Security camera/smart house
- Speaker assistant (e.g. Alexa)
- Wearable technology
- Augmented reality
- Phone case w/sprayer

A.4 Concept Selection

A.4.1 Device Criteria

Our solution included the following criteria tackling the questions:

- **Budget:** How expensive would the device be to produce?
- **Presence:** How often will our user acknowledge the device during their work period?
- **Ease of Use:** How convenient is it to interface with the device?
- **Barrier of Entry:** How much will the user be forced to learn in order to properly use the device?
- **Sustainability:** Is the device created from sustainable materials?
- **Privacy:** Is the user's data being safely managed? Is the data collected sensitive?

These topics were distributed into a weighting system such that our designs could be properly scored, which can be seen in the table below:

Criteria	Weight
Budget	20
Presence	20
Ease of Use	20
Barrier of Entry	20
Sustainability	10
Privacy	10

A.4.2 Brainstorming

After a brainstorming session, 8 general concepts were considered:

- A PDA style, portable device
- Tablet planner
- Face-tracking water sprayer
- Game Device
- Focus Desk
- Security camera system
- Speaker Assistant
- Wearable Device

A PDA device

A portable interface device designed to be handheld, providing tasks as needed in a self-contained network.

Criteria	Score	Weighted Score	Note
Budget	8	160	Portable devices are often more expensive
Presence	3	60	While on the user the device would be rather small
Ease of Use	6	120	
Barrier of Entry	4	80	
Sustainability	3	30	Portables require batteries which are bad for the environment.
Privacy	10	100	Only relying on data user provides

Total: 550

A Tablet Planner

Criteria	Score	Weighted Score	Note
Budget	7	140	
Presence	6	120	
Ease of Use	6	120	
Barrier of Entry	7	140	
Sustainability	4	40	
Privacy	7	70	

Total: 630

Face-tracking Water Sprayer

A device that sprays water at the user when they appear distracted during focus sessions. This system would use facial tracking for determining user focus and for aiming the sprayer.

Criteria	Score	Weighted Score	Note
Budget	7	140	
Presence	7	140	
Ease of Use	3	60	
Barrier of Entry	3	60	
Sustainability	4	40	
Privacy	4	40	

Total: 480

Game Device

A device designed to be located on a desktop designed to gamify tasks. The device would have unorthodox controls to allow for more fun interactions, as well as a robust speaker and lighting system for maximum presence and feedback.

Criteria	Score	Weighted Score	Note
Budget	9	180	Using simple LEDs and LCDs may be a lot cheaper
Presence	7	140	
Ease of Use	4	60	
Barrier of Entry	4	60	
Sustainability	5	40	
Privacy	10	40	

Total: 630

Focus Desk

A smart desk with an in-built notification system and lighting system to improve focus.

Criteria	Score	Weighted Score	Note
Budget	1	20	
Presence	4	80	
Ease of Use	5	100	
Barrier of Entry	3	60	
Sustainability	3	30	
Privacy	5	50	

Total: 340

Security Camera System

An array of cameras to set up around the house that monitors the user with a cloud-powered system

Criteria	Score	Weighted Score	Note
Budget	2	40	Multiple units costs higher
Presence	8	160	Always watching
Ease of Use	4	80	Set up and replace batteries consistently
Barrier of Entry	3	60	Need to set cameras and replace batteries
Sustainability	3	30	Cameras each haver batteries, which are hard to recycle
Privacy	1	10	Sends highly sensitive data to cloud constantly

Total: 380

Speaker Assistant

An assistant that communicates when tasks should be done.

Criteria	Score	Weighted Score	Note
Budget	4	80	Can get expensive with multiple units
Presence	4	80	
Ease of Use	8	160	Easy to use, just talk to it
Barrier of Entry	5	100	Some set up required
Sustainability	4	40	
Privacy	3	30	Always listening

Total: 490

Wearable Device

A watch-like device with a gantt chart-like interface.

Criteria	Score	Weighted Score	Note
Budget	5	10	Small, less materials, but hard to fit
Presence	4	80	Always together with user
Ease of Use	6	120	
Barrier of Entry	4	80	
Sustainability	3	30	Requires lightweight batteries
Privacy	7	70	Data given by user, yet wireless communications may leak with the companion app.

Total: 480

A.4.3 Conclusion

Both the **tablet planner** and **game style device** appear to be the most ideal for the considerations given. A combination of the two was chosen in the end: a sizeable desktop device with a large and present screen as was suggested from the tablet planner, yet with the addition of several interfacing buttons.

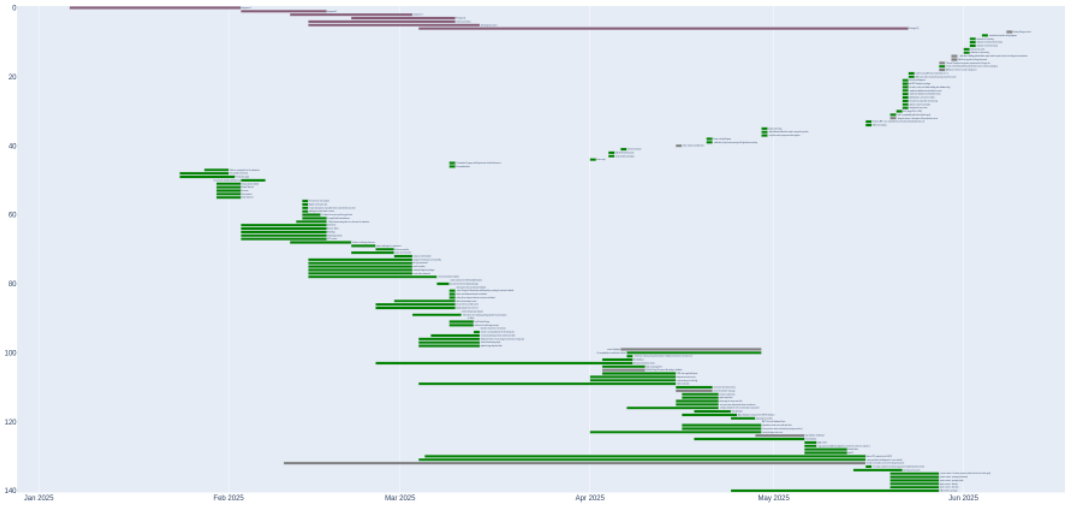


Figure 38: The gannt chart at the end of the prototype development

B Planning

B.1 Gantt chart

The overall plan for the project was managed with a gantt chart. Issue metadata was used to generate the chart when there were updates to the plan.

B.2 Division of Labor

Isabella, Lennan, and Mason worked on writing software to run on the hardware. Akanksha, Lennan, and Sulaiman worked on the cloud computing software bring up.

Major tasks involved:

- Hardware
  - Creating the database manager component
  - Creating the user interface component
  - Creating the Wi-Fi provisioning component
  - Creating the MQTT communication component
  - Connecting components to communicate with each other in a main loop
- Cloud
  - Configuring the virtual server
    - Configuring the server and api reverse proxy
    - Configuring the DNS forwarding service
  - Creating the web application back-end

- Creating the web app front-end and its UI
- Configuring the bridge between the device and the data-storage
    - Define the packet structure, as well as its storage and transmission

B.3 Collaboration

The prototype development and documentation was organized on the UCSC Gitlab instance. Tasks were assigned by creating an issue and ranked on the issue board by its current status. Open issues were for tasks that were being considered as necessary. Stretch goals started here, and went to closed if they were deemed unnecessary / out of time. The issue gets tagged *pending* if it was required for making progress. Once there was a team member actively working on that issue, they would self-assign and tag it as *in-progress*. Once resolved, it would be moved to the *closed* status. If it needed more work, it would be moved back to the *pending* status. A minimal amount of work was done on the Google Workspace platform.

C Manufactured Device Tests

C.0.1 Battery Life Testing

Introduction: While the device is intended to be generally stationary, the productivity device also includes a battery for portable use. The following test ensures the device's battery life lasts the expected duration of time.

Scope: Battery, Power

Apparatus: productivity device, host machine for flashing and running tests, charging cable, 1A power source

Independent variables: Device battery

Dependent variables: power on behaviour, device interaction

Procedure:

1. Power on the device and ensure it is at full charge as shown by the battery indicator in the top right of the UI. If it is not fully charged, charge the device to full.
2. Disconnect the device from the power cable. Check the settings to ensure the device is not set to power saving mode. Connect the productivity device to the host machine.
3. Run the test_batterylife.py pytest tool. This test program will disable automatic power saving mode on low battery and will run a series of functions to emulate daily use of the device. It will also begin a timer from the start of the test to when the device is unresponsive.
4. Leave the device on and undisturbed. The screen should remain on for the entire duration of the test.
5. Check on the test setup every three hours to monitor its battery life. Test ends when the device runs out of charge and the test_batterylife.py tool returns a final time on the device battery life duration.

Expectation: The device should not run out of charge for at least four hours.

C.0.2 Battery Life Cycle

Introduction: While the device is intended to be generally stationary, the productivity device also includes a battery for portable use. Lithium ion batteries experience capacity degredation as they are charged and discharged throughout their lifespan. Battery degradation from 100% down to 80% can be seen anywhere between 500 to 2000 charging cycles. To ensure that the device doesn't experience severe battery life fall off, battery cycling testing is performed to ensure charge management circuitry functionality and battery quality.

Scope: Battery, Power

Apparatus: productivity device, host machine for flashing and running tests, charging cable, 1A power source

Independent variables: Device battery

Dependent variables: power on behaviour, device interaction

Procedure:

1. Power on the device and ensure it is at full charge as shown by the battery indicator in the top right of the UI. If it is not fully charged, charge the device to full.
2. Connect the productivity device to the host machine.
3. Run the test_battery_cycling.py pytest tool. This test program will cycle the device through 1000 battery charge and discharge cycles while monitoring the battery life. Place the test setup in a room temperature area to be undisturbed for multiple weeks. Check on the test setup once a day to ensure the test has not halted and the test is proceeding as expected. Test is complete after 1000 battery charge cycles.

Expectation: The device battery life degradation should not exceed 20% after 1000 battery charge cycles.

C.0.3 Charge Speed

Introduction: While the device is intended to be generally stationary, the productivity device also includes a battery for portable use. The following test ensures the device is capable of charging to full in the expected duration of time.

Scope: Battery, Charging, Power
Apparatus: productivity device, charging cable, 1A power source
Independent variables: Device battery
Dependent variables: charging behaviour (planned)
Procedure:

1. Ensure the device battery is at low charge. When pressing the power button, the device should not display anything.
2. Plug the device into a charging cable and a 1A power source. Within five to ten minutes, press and hold the power button to turn the device on.
3. Keep the device plugged in and do not press the power button to wake it from sleep mode until at least 3 hours have passed. Check device battery level at 3 hours from test start.
4. Keep device plugged in until fully charged.

Expectation: Within four to five hours, the device should be fully charged.

C.0.4 Chemical Exposure

Introduction: Users may use household cleaning products on the surface of the device to sterilize or clean it of dust. These chemicals may weaken or damage the device housing. This test determines what chemicals the users must avoid use on the device. Material samples must be at least 3 inches by 3 inches.

Scope: chemical material testing

Apparatus: 6x productivity device plastic housing material samples, 6x LCD screen protector samples

Independent variables: chemicals

Dependent variables: plastic housing integrity

Procedure:

Chemicals:

- 99% Isopropyl Alchohol
- Acetone
- Clorox
- 409 cleaner
- Windex
- Bleach

1. Non porous disposable gloves, safety glasses, and mask of at least N99 rating or higher are required for this test to reduce fume inhalation and protect eyes from any chemical splashing. Prepare multiple cotton swabs on a sterile surface, there will have to be at least one swab per chemical tested.
2. Take pictures of all samples before applying any chemicals to the surface.
3. Begin with saturating a cotton swab in one of the chemicals and apply a thin layer to the surface of a housing sample and a screen protector sample.
4. Set aside the samples in a room temperature area undisturbed where it is not exposed to significant sunlight.
5. Discard the cotton swab. Discard the gloves if they were exposed to the chemical and apply fresh gloves to prevent cross contamination.
6. Repeat for all chemicals and all samples.
7. Allow the samples to remain undisturbed for 72 hours from application before checking on them.
8. Inspect and take pictures of all samples.

Expectation: Samples may show some deformation or weakness as a result of bleach or acetone, but should not show any damage from the remaining chemicals. Housing material samples must undergo a tensile strength test and have their tensile strength results directly compared to a typical sample to verify their structural integrity.

C.0.5 Drop testing

Introduction: Everyday lightweight devices face wear and tear. One of the common sources of damage being dropping the device. In order to replicate all the ways in which the productivity device can be damaged, drop testing is performed to ensure the device remains functional after damage.

Scope: device structural integrity

Apparatus: productivity device, host machine for functionality tests, measuring tape, adjustable height desk

Independent variables: device integrity, full device functionality

Dependent variables: drop height, device drop orientation

Procedure:

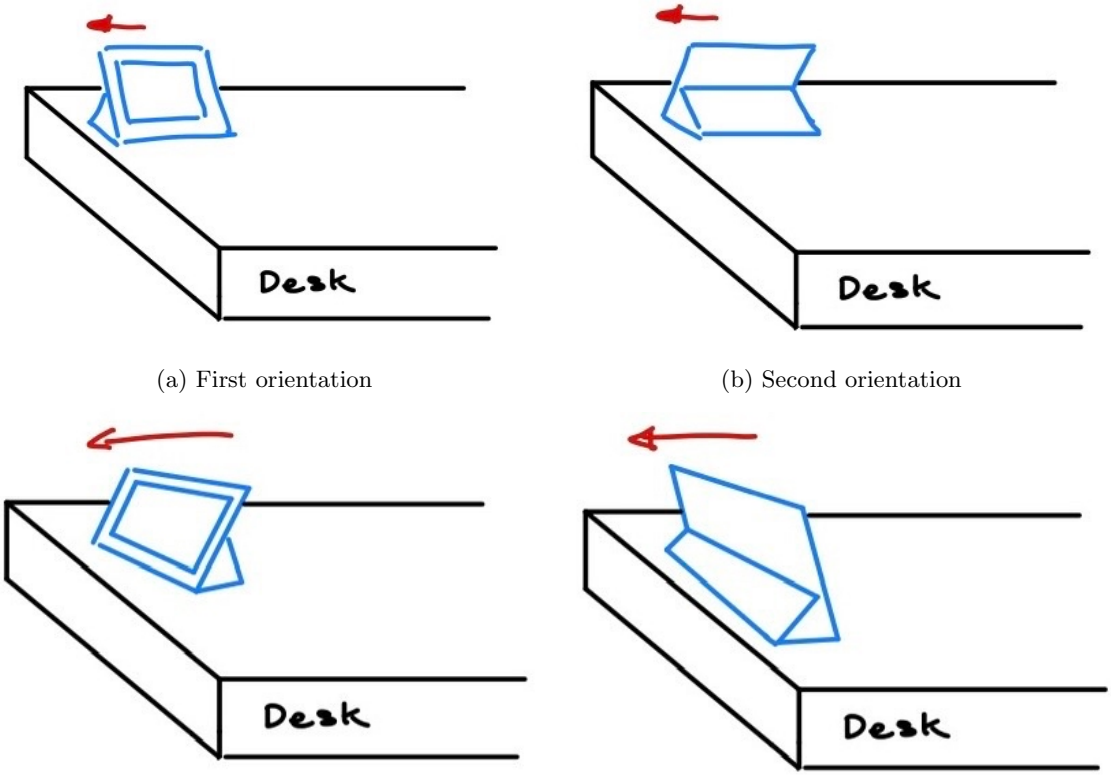
1. Verify productivity device is unplugged. Clear a space to perform the drop test and ensure all test participants are wearing close toed shoes. Flooring should be hard such as concrete, linoleum, or hardwood, avoid carpeted spaces for this test.

2. Using the measuring tape, measure a height from the ground up to 100 centimeters as is compliant with IEC drop testing standards. Adjust the adjustable height desk such that the top edge is at 100 centimeters.

3. The device will be dropped four times off the edge of the desk, each with specific orientations, intended to emulate the possible ways in which the device could fall or be dropped. The device should be slid towards the edge of the desk in the orientations shown in Fig. 39. Pictures should be taken of the device at all angles to document the progressive damage and a functionality test must be performed after each drop.

4. The device will be dropped once more, this time by having the tester hold the device facing themselves 100 centimeters above the ground and letting go of the device as shown in Fig. 40. Once again, images should be taken of the device and a functionality test must be performed after the drop.

Expectation: No cracking on the LCD screen and device remains fully functional after all five drops. Minor defects on the plastic housing are expected, but damage should be minimal without large cracks in the plastic.



(a) First orientation

(b) Second orientation

(c) Third orientation

(d) Fourth orientation

Figure 39: Different drop orientations

C.0.6 Factory Reset verification

Introduction: When issues arise in a device or if a user wants a fresh start, the productivity device must be able to be reset to a clean slate.

Scope: flash storage, device operations

Apparatus: productivity device, host machine for flashing and running tests, laptop or mobile device for web app interaction

Independent variables: Device data and file contents

Dependent variables: Existing user data on web app

Procedure:

1. Power on the device and check it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials. If the device is already connected to a user and has schedule information, skip to step 7.
2. Create a test user on the web app if it does not already exist.
3. Log into the user via a laptop or mobile device.

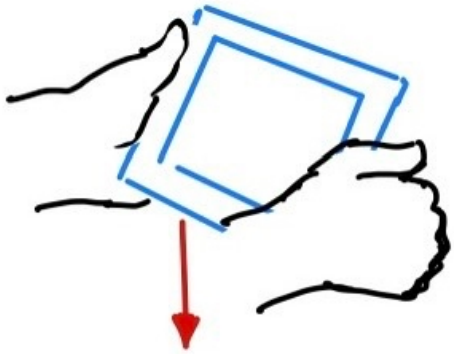


Figure 40: Final drop orientation

4. Add the device to the user using the corresponding device ID.
5. Add a task, event, and habit to be written to the device. Change the time setting from 12 hour to 24 hour display.
6. Navigate the device to verify that the user's schedule data has been added and the settings have been changed.
7. Navigate to the device settings and select factory reset. The device will take a few moments to erase existing user data.
8. After the device reboots, navigate around the device and verify that the wifi symbol is not displayed and no existing events, tasks, or habits remain on the device.
9. On the web app, return to the devices tab and check that the device ID of the newly wiped productivity device is no longer listed under devices registered under the user.
10. Connect the device to the host machine and run test_clean.py. This program will verify that the device is free of all existing user data and settings.

Expectation: No existing schedule data, wifi information, or user settings. Web app updates to show the device is no longer registered under the user.

C.0.7 LCD functionality Verification

Introduction: The main form of interaction the user has to the productivity device is the LCD touchscreen. This aspect must remain functional to allow for the user to view and interact with their schedule. The following test ensures the screen behaves as expected.

Scope: LCD Screen
Apparatus: productivity device, host machine for flashing and running tests
Independent variables: LCD visuals
Dependent variables: power on behaviour (planned)
Procedure:

1. Turn on the device and connect it to the host machine. Visually confirm that the LCD boots within 10 seconds of powering on.
2. Run the test_screen.py pytest tool. This test program runs through a gamut of visuals and will calibrate touch input.

Expectation: LCD is visually confirmed to be capable of displaying a full range of RGB colors with no dead pixels or cracked screen. Touch input is reactive and calibrated such that touches are accurate to the screen display.

C.0.8 Haptics and Sound functionality Verification

Introduction: The productivity device is intended to play small noises and haptics when the user touches and interacts with the device. The following test ensures the speakers and vibration motors behave as expected.

Scope: Speakers, Vibration motor
Apparatus: productivity device, host machine for flashing and running tests
Independent variables: Sounds and haptics
Dependent variables: power on behaviour (planned)
Procedure:

1. Turn on the device and connect it to the host machine.

2. Run the test_sound.py pytest tool. This test program will play all of the expected sounds and activate the vibration motor at varying intensities.

Expectation: Device sound is confirmed to work via auditory testing and vibrations are determined functional by touch inspection.

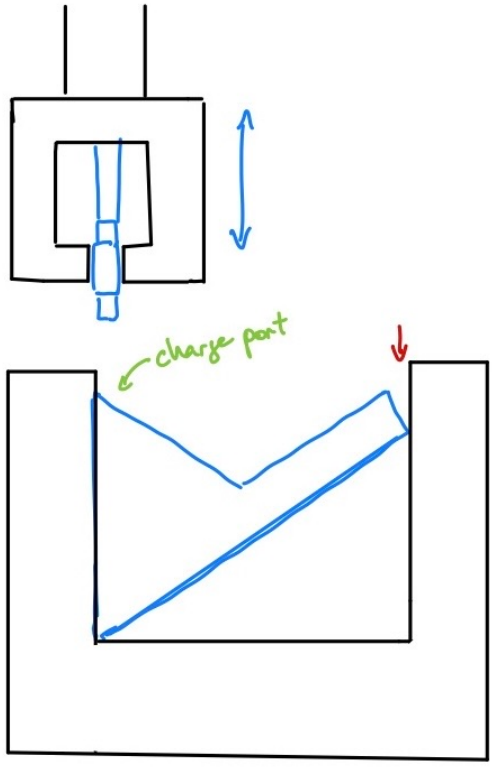


Figure 41: Test setup

C.0.9 Port Durability

Introduction: The USBC port is the main way the device gets recharged and is subject to wear as the user plugs and unplugs the device. In order to verify the longevity of the device, the charging port of the device is subject to 1000 plug cycles.

Scope: Charging port, strength test

Apparatus: productivity device, plug durability testing apparatus, USBC cable.

Independent variables: Pressure

Dependent variables: Charge port durability

Procedure: 1. Take pictures of the charge port before beginning the test. 2. Set up the port durability testing machine based off of fig. 41. The portability device is an uneven shape which makes it difficult to clamp on to. Ensure that the clamp is not deforming the shape of the device. The red arrow indicates an area in which some manner of wadding or firm sponge could be placed to better conform to the shape of the device and provide stability during testing. Make sure the device setup is clamped to the port durability testing device to prevent shifting. Ensure that the end of the USBC cable is properly aligned such that as the arm descends, it will plug into the productivity device.

3. Program the device to plug and unplug 1000 cycles.
4. Run the first 10 tests supervised, afterwards check on the device every 30 minutes to ensure the test is progressing smoothly.

Expectation: The productivity device's USBC port is expected to withstand a minimum of 1000 plug cycles.

C.0.10 Software Update test

Introduction: In order to provide security improvements and new features, the device needs to be able to recieve software updates over the internet.

Scope: device software, device operations

Apparatus: productivity device, host machine for flashing and running tests, laptop or mobile device for web app interaction

Independent variables: Device recieved contents

Dependent variables: Device software version

Procedure:

1. Power on the device and check it is connected to the internet by checking the wifi symbol in the top right. If the wifi symbol is not displayed, connect to the device's access point and provide it wifi credentials. If the device is already connected to a user and has schedule information, skip to step 7.
2. Create a test user on the web app if it does not already exist.
3. Log into the user via a laptop or mobile device.
4. Add the device to the user using the corresponding device ID.
5. Using the host machine for flashing and running tests, run test_softwareUpdate.py with the corresponding device ID in the following syntax: `python test_softwareUpdate.py [device_id]`. This will inform the server to prepare a software update for this particular device.
6. Back on the web app, enter the device dashboard and navigate to the device settings. Select "Check for Updates". The web app will send a signal the device to check for software updates. The device should display a window that asks the user to confirm the update with a button on the touch screen to press. Select the button to confirm the update.
7. The device will undergo a system update which will take some time. The device may restart multiple times. Once complete, the UI will be a different color to indicate the software update was sucessful.

Expectation: The device will show a different colored UI to demonstrate the software update has been completed succesfully.

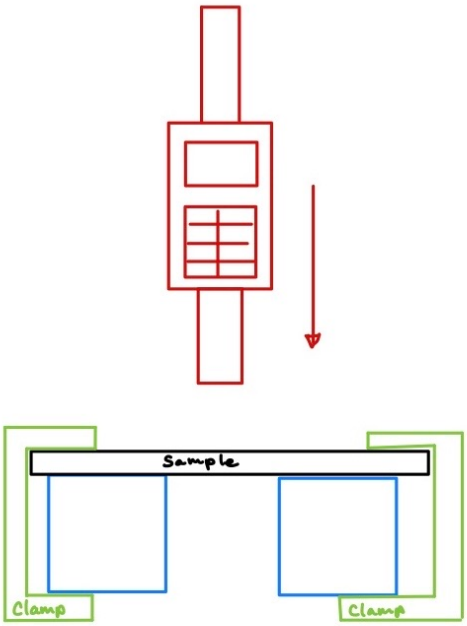


Figure 42: Test setup

C.0.11 Tensile Strength testing

Introduction: In order to ensure the integrity of the plastic housing of the productivity device, the material must under go compressive pressure testing to verify its strength.

Scope: Housing material, strength test

Apparatus: 6x productivity device plastic housing material samples, tensile testing apparatus, acrylic blast shield

Independent variables: Pressure applied

Dependent variables: Pressure before materials crack

Procedure: 1. Take pictures of all samples before beginning the test. 2. Safety glasses are required for this test. The following test was designed with a Mark-10 motorized tensile test stand in mind but can be adapted to other motorized tensile testers. Set up the tensile test machine based off of fig. 42. Ensure that the sample is firmly clamped to the base of the tensile testing machine. This set up is intended to apply pressure on the sample until it deforms past usability or cracks.

3. Put up the acrylic blast shield and ensure all persons are located behind it to reduce the possbility of injury from flying plastic pieces.
4. Ensure the device screen shows the maximum pressure from the start of the test. Begin lowering the pushing end of the tensile machine into the plastic piece, slowly applying more and more pressure until it cracks or deforms past the point of usability.
5. When the test is complete, raise the tensile testing arm, and remove the acrylic blast shield when safe to do so. Inspect and take pictures of all samples.

Expectation: Record the value of the maximum pressure the plastic housing endured. If testing chemically exposed materials, compare the values with non exposed samples.

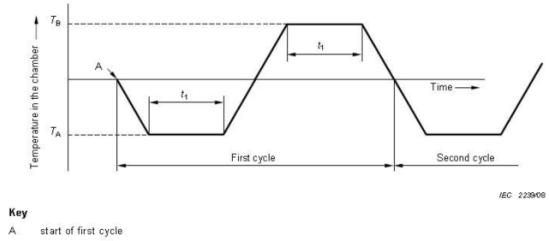


Figure 3 – Nb test cycle

Figure 43: IEC 60068-2-14 Thermal Cycling

C.0.12 Thermal Cycling: Cold and Dry Heat

Introduction: Thermal Cycling exposes samples to extreme temperature changes, which replicate accelerated aging of the device and ensures that the sample can operate at its rated maximum and minimum temperatures. Test procedures follow IEC 60068-2-14 thermal cycling procedures intended for electronic components.

Scope: Operating temperatures, full functionality
Apparatus: productivity device, host machine for functionality tests, thermal testing chamber
Independent variables: full device functionality
Dependent variables: chamber temperature and humidity
Procedure:

1. Remove the device's lithium ion cells and set them aside. Place the device inside of the thermal chamber.
2. Feed a cable connecting the device through the port of the thermal chamber. Connect the end of the cable to the monitoring host machine. Check that the productivity device is on, connected to the internet and is loaded with an existing test account with tasks, events, and habits. Host machine should be set up to monitor device functionality for the entire duration of the test.
3. Close the test chamber.
4. Enter temperature profile into test chamber settings in accordance to IEC 60068-2-14 with a temperature maximum of 85C and a minimum of -40C in accordance to typical maximum and minimum operating temperatures of FR4 PCBs[5] and microcontrollers[6]. Up and down ramp should be set to 5C/min. Dwell at min and max temperatures for 3hrs each. Perform a total of two cycles, then set the chamber return to 25C with a ramp of 5C/min and dwell for 2 hrs to allow the device to return to room temperature. Ensure humidity is set as low as possible.

Expectation: Device should remain operational through then entire duration of the test and outer casing should be free of cracks and remain structurally intact.

[5]https://fr4material.com/index.php/understanding-fr4-sheet-temperature-specifications/
[6]https://copperhilltech.com/content/The%20Operating%20Temperature%20For%20A%20Raspberry%20Pi%20%E2%80%93%20Technol

C.0.13 Thermal Shock: Air to Air

Introduction: Plastics and PCBs when exposed to sudden changes in temperature can experience cracking and solder joints can fail. Thermal shock testing is performed accordance to IEC 60068-2-14. The temperature maxima of 85C and a minima of -40C in accordance to typical maximum and minimum operating temperatures of FR4 PCBs[7] and microcontrollers[8].

Scope: Operating temperatures, full functionality

Apparatus: productivity device, 2x host machine for functionality tests, 2x thermal testing chambers or a thermal shock testing chamber

Independent variables: full device functionality

Dependent variables: chamber temperature and humidity

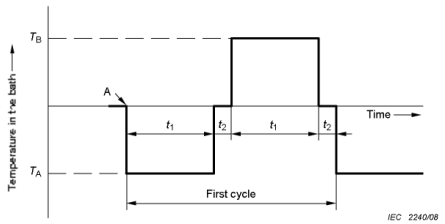
Procedure for 2x thermal testing chambers:

1. Remove the device's lithium ion cells and set them aside. Place the device atop a flat surface such as wood or a metal tray.
2. Feed a cable for the host machine through the port of the thermal chamber. Connect the end of the cable to the monitoring host machine.
3. Before connecting the productivity device, close and cool the test chamber to the temperature minimum of -40C. Down ramp should be set to 5C/min.
4. Once thermal chamber is stable at the minimum temperature, open the test chamber and place the device and the tray inside of the thermal chamber.
5. Wearing gloves, connect the cable from the host machine into the productivity device. Check that the productivity device is on, connected to the internet and is loaded with an existing test account with tasks, events, and habits. Host machine should be set up to monitor device functionality for the entire duration of the test. Device should be dwell at this temperature for 3hrs.
6. Set up a second host machine in the second thermal chamber in the same way as the first thermal chamber. Close and heat the chamber to 85C with an up ramp of 5C/min.
7. Open both test chambers and unplug the device while wearing protective gloves or using tongs. Transfer the device to the second hot chamber as quickly as possible. Plug the productivity device into the second monitoring host device, ensure the device is recieving power and communicating to the host device. Close the test chamber. Device should dwell at this temperature for 3hrs.
8. Repeat the transfer process from the cold chamber to the hot chamber. Perform a total of two cycles, then set the chamber to return to 25C with a ramp of 5C/min and dwell for 2 hrs to allow the device to return to room temperature.

Procedure for a thermal shock testing chamber: 1. Remove the device's lithium ion cells and set them aside. Feed a cable for the monitoring host machine through the port of the thermal chamber. 2. Set one chamber to 85C and the other to -40C. 3. Close the test chamber. 4. Once the chambers are at stable temperature. While wearing protective gloves, place the productivity device inside of one of the chambers of the thermal chamber. Connect the cable to it and check that the productivity device is on, connected to the internet and is loaded with an existing test account with tasks, events, and habits. Host machine should be set up to monitor device functionality for the entire duration of the test. Device should be dwell at this temperature for 3hrs. 5. After 3 hrs,

[7] https://fr4material.com/index.php/understanding-fr4-sheet-temperature-specifications/

[8] https://copperhilltech.com/content/The%20Operating%20Temperature%20For%20A%20Raspberry%20Pi%20%E2%80%93%20Technol



Key

A start of first cycle

Figure 4 – Nc test cycle

Figure 44: IEC 60068-2-14 Thermal Shock

drop the device to the lower chamber. Device should dwell at this temperature for 3 hrs. 6. Raise the device to the upper chamber to repeat the thermal shock cycle one more time. After the cycles are complete, set the chamber to return to 25C and dwell for 2 hrs to allow the device to return to room temperature.

Expectation: Device should remain functional through then entire duration of the test and outer casing should be free of cracks and remain structurally intact.

C.0.14 Electromagnetic interference testing

Introduction: Electronic devices require rigorous testing to guarantee electromagnetic compatibility and reliable performance throughout their operational lifespan. Electromagnetic interference represents a significant regulatory and performance challenge for portable tablet devices, particularly those integrating multiple wireless functions and high-frequency digital circuits. The primary testing standards utilized for EMI/EMC evaluation include FCC Part 15 for North American markets, which establishes emission limits and certification requirements for both unintentional and intentional radiators. This regulatory framework is complemented by CISPR 32/EN 55032 for emission testing, CISPR 35/EN 55035 for immunity requirements, and IEC 61000-4 series standards for specific immunity phenomena.

Apparatus: productivity device, semi-anechoic chamber, EMI test receiver, calibrated antennas (biconical, log-periodic, horn), line impedance stabilization network (LISN), RF signal generator, power amplifier, coupling/decoupling networks (CDN), ESD generator, near-field probes, host machine for testing

Independent variables: test frequency ranges (150 kHz-30 MHz, 30 MHz-1 GHz, 1-6 GHz), field strength levels (1 V/m, 3 V/m, 10 V/m), EUT operational modes (standby, active use, wireless transmission), antenna orientations (horizontal, vertical polarization), measurement distances (3m, 10m)

Dependent variables: emission amplitude (dBuV, dBuV/m), immunity performance criteria (A, B, C), functional operation status (pass/fail), wireless communication quality (%), touchscreen sensitivity (%), error rates during immunity testing (%), field uniformity measurements (dB)

Procedure:

1. Verify the device is fully charged and operational by powering on and checking all basic functions.

2. **FCC Part 15 Baseline Configuration:**

   - Record initial device configuration including wireless module states
   - Document operational modes: standby, typical use, maximum transmission
   - Test touch sensitivity and wireless connectivity baseline performance
   - Establish cable configuration per ANSI C63.4 requirements

3. **CISPR 32 Conducted Emissions Testing (150 kHz - 30 MHz):**

   - Connect device AC adapter through 50uH/50ohm LISN on ground plane
   - Position device on non-metallic table 80 cm above ground plane
   - Configure EMI receiver with 9 kHz resolution bandwidth and quasi-peak detector
   - Exercise all device functions during measurement including wireless transmission
   - Monitor both line and neutral conductors with Class B limits (631 µV quasi-peak and 200 µV average for frequencies above 500 kHz)

4. **ANSI C63.4 Radiated Emissions Testing (30 MHz - 1 GHz):**

   - Place device in semi-anechoic chamber 3 meters from receiving antenna
   - Scan antenna height 1-4 meters while rotating device through 360 degrees
   - Use biconical antenna (30-300 MHz) and log-periodic antenna (200 MHz-1 GHz)

- Measure both horizontal and vertical polarizations with quasi-peak detector
- Apply Class B limits (100-200 uV/m depending on frequency)

5. **Extended Radiated Emissions Testing (1-6 GHz):**

   - Continue testing using horn antennas for frequencies above 1 GHz
   - Switch to average detector with 1 MHz resolution bandwidth
   - Focus on wireless module fundamental and harmonic frequencies
   - Document peak emissions during maximum data transmission

6. **IEC 61000-4-3 Radiated Immunity Testing (80 MHz - 6 GHz):**

   - Generate 3 V/m uniform field using calibrated transmit antennas
   - Apply 80% amplitude modulation at 1 kHz with 1% frequency stepping
   - Monitor device functions continuously during 3-second dwell per frequency
   - Test both horizontal and vertical field polarizations
   - Assess performance per Criterion A (normal operation) requirements

7. **IEC 61000-4-6 Conducted Immunity Testing (150 kHz - 80 MHz):**

   - Inject 3V EMF using appropriate coupling/decoupling networks
   - Apply 150ohm common-mode impedance for unshielded cables
   - Use 80% amplitude modulation with systematic frequency stepping
   - Monitor charging, data transfer, and wireless functions

8. **IEC 61000-4-2 ESD Immunity Testing:**

   - Apply ±4 kV contact discharge to conductive surfaces
   - Apply ±8 kV air discharge to insulated surfaces and near openings
   - Test 10 discharges per point with 1-second intervals
   - Evaluate touchscreen functionality and wireless connectivity recovery

9. **Wireless Coexistence Testing:**

   - Activate all wireless modules simultaneously (Wi-Fi, Bluetooth, cellular)
   - Measure emissions during maximum data throughput scenarios
   - Verify no interference between internal wireless subsystems
   - Test performance across all supported wireless channels

10. **Battery Operation Assessment:**

    - Repeat critical emission measurements on battery power only
    - Verify compliance maintained without AC adapter connection
    - Document any emission differences between AC and battery operation

11. **Multi-Configuration Testing:**

    - Test device in portrait and landscape orientations
    - Connect various peripherals (USB devices, audio cables, stylus)

- Evaluate EMC performance with different screen brightness levels
- Test with various applications and processing loads active

Expectation:

The device should demonstrate full compliance with FCC Part 15 Class B emission limits showing conducted emissions below 631 µV quasi-peak and 200 µV average (150 kHz-30 MHz) and radiated emissions below 100-200 uV/m at 3 meters (30 MHz-1 GHz), maintain Performance Criterion A during radiated immunity testing at 3 V/m field strength, show no functional degradation during ±4 kV ESD contact discharge testing, exhibit wireless coexistence with no interference between internal modules during simultaneous operation, and demonstrate consistent EMC performance across all operational configurations including battery operation, multiple orientations, and various peripheral connections.

C.0.15 UV Exposure Testing

Introduction: Electronic devices require rigorous testing to guarantee longevity and reliable performance throughout their operational lifespan. Ultraviolet radiation exposure represents a significant environmental hazard for portable tablet devices, particularly those used in outdoor applications or near windows where solar radiation can cause material degradation, display performance deterioration, and housing discoloration. The primary testing standard utilized for UV exposure evaluation is IEC 60068-2-5, which specifies comprehensive methods for testing electronic equipment under simulated solar radiation and accelerated weathering conditions. This standard is complemented by ASTM G154 for fluorescent UV lamp testing, ASTM G155 for xenon arc exposure simulation, and ISO 4892 series for international harmonization.

Scope: display performance, touch sensitivity, housing materials, optical properties

Apparatus: productivity device, UV chamber with fluorescent UV lamps (UVA-340 and UVB-313), xenon arc lamp chamber, UV radiometer, spectrophotometer, controlled environment chamber, host machine for testing

Independent variables: UV exposure type (UVA, UVB, full spectrum solar simulation), exposure duration (15 kWh/m2, 50 kWh/m2, 100 kWh/m2), irradiance level (0.89 W/m2/nm at 340nm), temperature during exposure (60 degrees C +/- 5 degrees C, 65 degrees C +/- 3 degrees C), humidity level (50% RH, 65% RH)

Dependent variables: Display brightness levels (cd/m2), color coordinates (CIE Lab*), luminance uniformity (%), touch sensitivity response time (ms), surface gloss retention (%), housing color change (Delta E), material degradation (visual inspection), optical transmittance (%)

Procedure:

1. Verify the device is fully charged and operational by powering on and checking all basic functions.

2. **IEC 60068-2-5 Baseline Testing:**

   - Record initial display brightness at maximum setting using calibrated photometer
   - Measure color accuracy using standard test patterns and spectrophotometer
   - Document surface gloss levels using 60 degree gloss meter
   - Test touch sensitivity across all screen areas and record response times

3. **ASTM G154 Fluorescent UV Accelerated Aging Testing (specific wavelength):**

   - Place device in UV chamber with UVA-340 lamps
   - Set irradiance to 0.89 W/m2/nm at 340nm wavelength
   - Maintain specimen temperature at 60 degrees C +/- 5 degrees C (standard specimen temperature)
   - Expose for cycles totaling 15 kWh/m2 UV dose
   - Monitor UV intensity continuously with calibrated radiometer

4. **ASTM G155 Xenon Arc Testing (full spectrum wavelength):**

   - Transfer device to xenon arc chamber for full spectrum solar simulation
   - Set irradiance to 0.55 W/m2 at 340nm with daylight filters
   - Maintain black panel temperature at 65 degrees C +/- 3 degrees C (black panel temperature)

- Include moisture cycles: 102 minutes dry / 18 minutes water spray
- Continue exposure to reach total of 50 kWh/m2

5. **Post-Exposure Evaluation:**

   - Allow device to stabilize at room temperature for 4 hours
   - Repeat all baseline measurements
   - Calculate retention percentages for all optical properties
   - Perform visual inspection for surface defects, crazing, or chalking
   - Test all device functions including wifi connectivity and data persistence

6. **Additional Assessments per Standards:**

   - Measure surface hardness change using pencil hardness test
   - Evaluate adhesion of coatings using tape test method
   - Document any delamination or blistering of screen protectors

Expectation:

After 65 kWh/m2 total UV exposure across multiple test methods, the device should maintain at least 85% of original display brightness, show color change (Delta E) less than 3.0, retain full touch sensitivity with response times within 10% of baseline, and exhibit no visible surface degradation or functional failures. Housing materials should show minimal discoloration and maintain structural integrity per IEC 60068-2-5 requirements.

D Review

D.1 Akanksha

Considering our hardware limitations, we were able to successfully demonstrate most of the features that we set out to implement. We were able to get most of the core functionality working, but it would be nice to have a device which could handle more features. Something we did well in this project was delegating tasks effectively. We all worked on different aspects of the device, and were able to work in parallel on the backend and frontend. Something that could have helped our project go smoother is more extensive planning, probably starting from the first quarter. Because we didn't go into the second quarter with a solid plan and idea of who would be working on what, it took us a few weeks to find our footing. If we started with a better plan, we might have been able to discover the memory issues earlier and make some hardware changes. Additionally, there were a few times where we had to make changes to the web app, resulting in having to refactor the Python server code. Sometimes this was trivial, but in other cases it took a while to figure out how to modify the code such that the web app to server and server to database communication would still work correctly. If we had fleshed out the details of our design a bit more, we might have been able to avoid this and have time to implement more features.

D.2 Isabella

Working on this project was a long and difficult process. As far as what went well, I'm incredibly grateful to my talented team members; in many ways it was my peers who I learned the most from as we worked hard together, bounced ideas, helped each other out, and troubleshooted issues together. I'm immensely proud of our resulting prototype, manufactured design, design document, and presentations. Although the limitations of the hardware proved incredibly difficult to work with, I'm fairly satsified with the way the device turned out in the end, all things considered. Our choice of library, ESP-IDF saved us a lot of memory. The Arduino library is easier to develop for but tends to eat up more memory. As far as what could be improved, for sure I wish I could have spent more time on improving the device functionality rather than fighting the memory limitations of the device. During development, it was hard to determine exactly how much memory we would need for this sort of device and by the time we realized just how limited we were, we didn't have enough time to migrate over to a new system. We did go back and forth about possibly purchasing a ESP32-S3, but the base devkit we were looking at wasn't a significant improvement in memory, and we decided to focus on what we had on hand. I think it was this hardware choice that made finishing the prototype so slow unfortunately. If we had the chance to add in some more features, such as time blocking or touch functionality, I think the prototype would have felt much closer to the manufactured product.

D.3 Lennan

The git environment worked well for managing tasks, as well as collaboration on the documentation. The firmware components had less collaboration, as they were separated by function, which needed less collaboration until bringing everything together. Each could be developed and tested individually, but that ended up causing problems with memory availability when all put together. I've never been in a situation when the program hit a memory limit, so I hadn't thought of keeping track of the memory usage of each component. I think the AWS side of the development was fine.

Maybe I could have configured the account access control earlier, so that everyone had access from the start.

D.4 Mason

The development of our prototype demonstrated strong capabilities despite the limited resources available on the ESP32-C3. We were able to implement the most critical aspect of our design: managing user entries relevant to daily planning. This core functionality was successfully realized, validating the feasibility of our design concept. However, in hindsight, I wish we had been able to integrate more of our planned features, particularly the time blocking system, which our hardware ultimately could not support. We spent a substantial amount of time refactoring instead of focusing on providing a functional and fluid system one would want on a device designed to motivate and remind. One major lesson learned was the importance of benchmarking early to ensure we are choosing a suitable platform for our prototype, instead of one that we had found to be the most available. Had we thoroughly evaluated the performance limitations of the ESP32-C3 beforehand, we might have opted for a more powerful platform like the Raspberry Pi series of processors during the prototyping phase. This would have allowed us to fully test and iterate on our feature-rich design, and only then select a device more appropriate for our final use case. Instead, we found ourselves constrained by the hardware, forced to cut back on functionality to fit within its limitations.

D.5 Sulaiman

The development process of both our device and prototype was deeply insightful. We experienced the entire process from identifying a need, designing a solution, and actualizing a prototype. Our team worked incredibly well together in all the processes of the project. We spent the appropriate amount of time brainstorming, planning, and coordinating. We delegated the different sections of the prototype for streamlined completion. Although we had specialized roles, there was a significant amount of overlap in our work between the front and backends. Our planning could have used a little refinement, as we may have potentially avoided the memory bottleneck with a bit more brainstorming early in our prototype. We completed parts of the prototype in parallel, and this was instrumental in completing the prototype by the deadline. We were also able to recognize the adjustments that we'd need to make for the device, as it was fairly apparent and straightforward, but the prototype itself is fairly detailed relative to the device. The only real adjustments outside of refining our research would be meeting more often, as this would have made us more resilient throughout the design process. I primarily focused on the front end, and I know there were many refinements, such as making habits to read and write from the web side. Though the task specialization was beneficial to the process, I know working on the backend would've provided insight for a more robust implementation that would've more closely resembled our device.