# Upcycle Build

JIB 4342

Charles Wyner, Arthur Riechert, Edward Jin, Bereket Yoseph

Client: Eric Murdock

GitHub Repository: https://github.com/cwyner/JIB-4342-RamblinRecyclers.git

# Table of Contents

# List of Figures

# Terminology

**API (Application Programming Interface)**: A contract between two applications that defines how they interact with each other.

**Back-end**: The behind-the-scenes part of the application that the user cannot interact with.

**SQL**: A programming language for managing and manipulating databases.

**Cloud Firestore**: A NoSQL, document-oriented database service offered by Firebase.

**Expo Go**: A sandbox environment that allows developers to quickly build and experiment with features for mobile applications.

**Firebase**: A cloud service provided by Google that enables data storage into a centralized database, as well as user authentication.

**Front-end**: The part of the application that the user sees and interacts with

**Inventory Receiving Clerk**: A volunteer at a reuse center who oversees recording the new donations for the reuse center.

**JSON (JavaScript Object Notation)**: A commonly used data format that uses human-readable text to store information.

**React Native**: A JavaScript UI framework for cross platform mobile app development.

**Reuse Center**: A volunteer-led, non-profit organization that takes donations of construction materials with the purpose of refurbishing and selling them for proper use.

**TypeScript**: A strongly typed object-oriented programming language that is a syntactic superset of JavaScript.

# Introduction

## Background

Our team, the Ramblin' Recyclers, has built Upcycle Build. Upcycle Build is an IOS and Android app that aims to alleviate the increasing problem of construction and demolition waste by developing an easy-to-use platform whereby community organizations upcycle materials for reuse. This app, meant for employees and volunteers at reuse centers, simplifies material intake, upcycling activities, and the management of tasks. Among many features, the Activity Calendar and real-time inventory tracking will go a long way in reducing waste and fostering sustainability within an organization. Upcycle Build responds to the dire need for effective waste management, integrating accessible technology with thoughtful design that fosters community-driven environmental impact.

## Document Summary

The section of *System Architecture* describes how the components of the system interact both statically and dynamically and gives insight into the reasoning behind the architecture decisions. Static architecture defines the structure of the system, focusing on the organization of the Presentation Layer, Application Logic Layer, and Database Layer, along with the relationships and flow among these layers. The dynamic architecture underlines the runtime behavior of those components, showcasing the processing and live updates of different tasks related to material intake and task assignments for seamless operation and usability.

The *Component Detailed Design* provides a look into our system architecture's specific components. It too consists of a static diagram and a dynamic diagram. The static diagram shows the app's low-level components and their static relationships, and the dynamic diagram details the runtime interactions of the static components for a specific scenario, namely editing an existing donation in inventory.

The *Data Design* section describes how we manage data in our Firestore database. The diagram shows the large, umbrella stores of data such as User Data and Donation data, as well as the smaller fields included within the aforementioned.

The *UI Design* section shows each of the screens that a user will interact with when using our app. Each screenshot provides specific descriptions of what the screen does, how or why a user may be presented with this screen, and the design heuristics we followed when creating it.

# System Architecture

## Static System Architecture

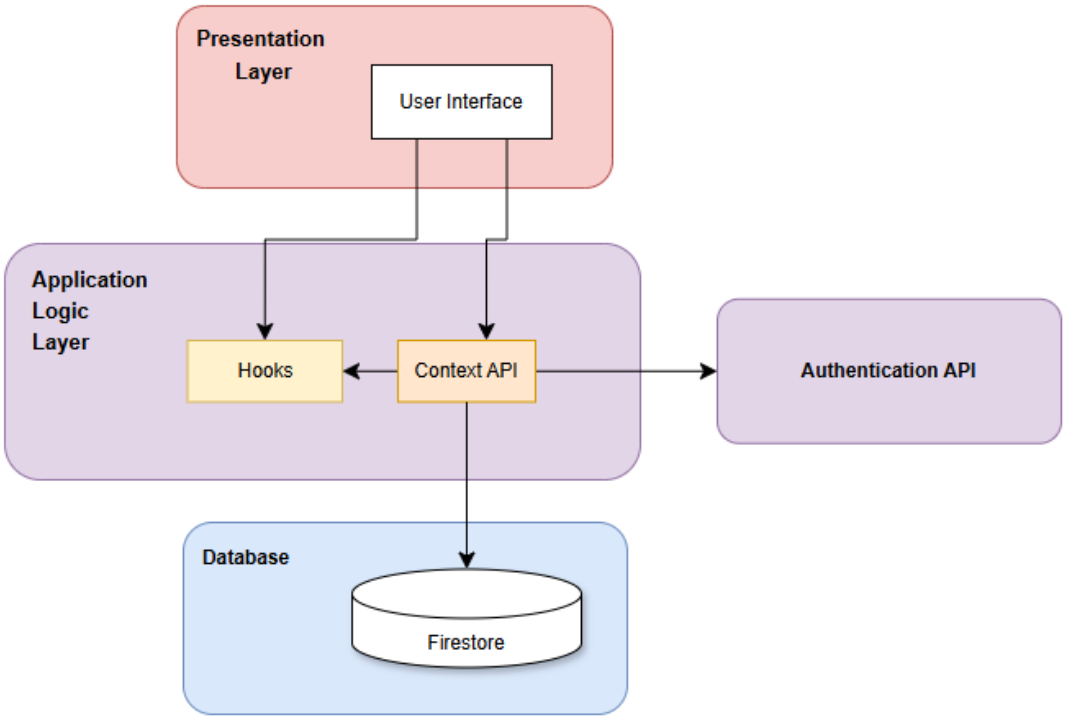


*Figure 1: N-Tier Architecture*

### Description:

Static System Architecture: The Upcycle Build application is designed to be scalable, modular, and efficient. This Presentation Layer provides a seamless and alike user experience on any device, be it mobile or a desktop, using Flutter, for tracking tasks and inventories or logging upcycled materials. Application Logic Layer: The Track Construction Material module manages the workflow in this layer of the system, while integration with React Native supports flexible task scheduling and inventory management. Material and task information, along with user data, is kept within the Database Layer, powered by Firebase, providing real-time scalable storage that can also work in offline modes during connectivity-constrained scenarios. It allows for seamless integration via external APIs and accommodates future scaling without sacrificing performance or usability.

## Dynamic System Architecture

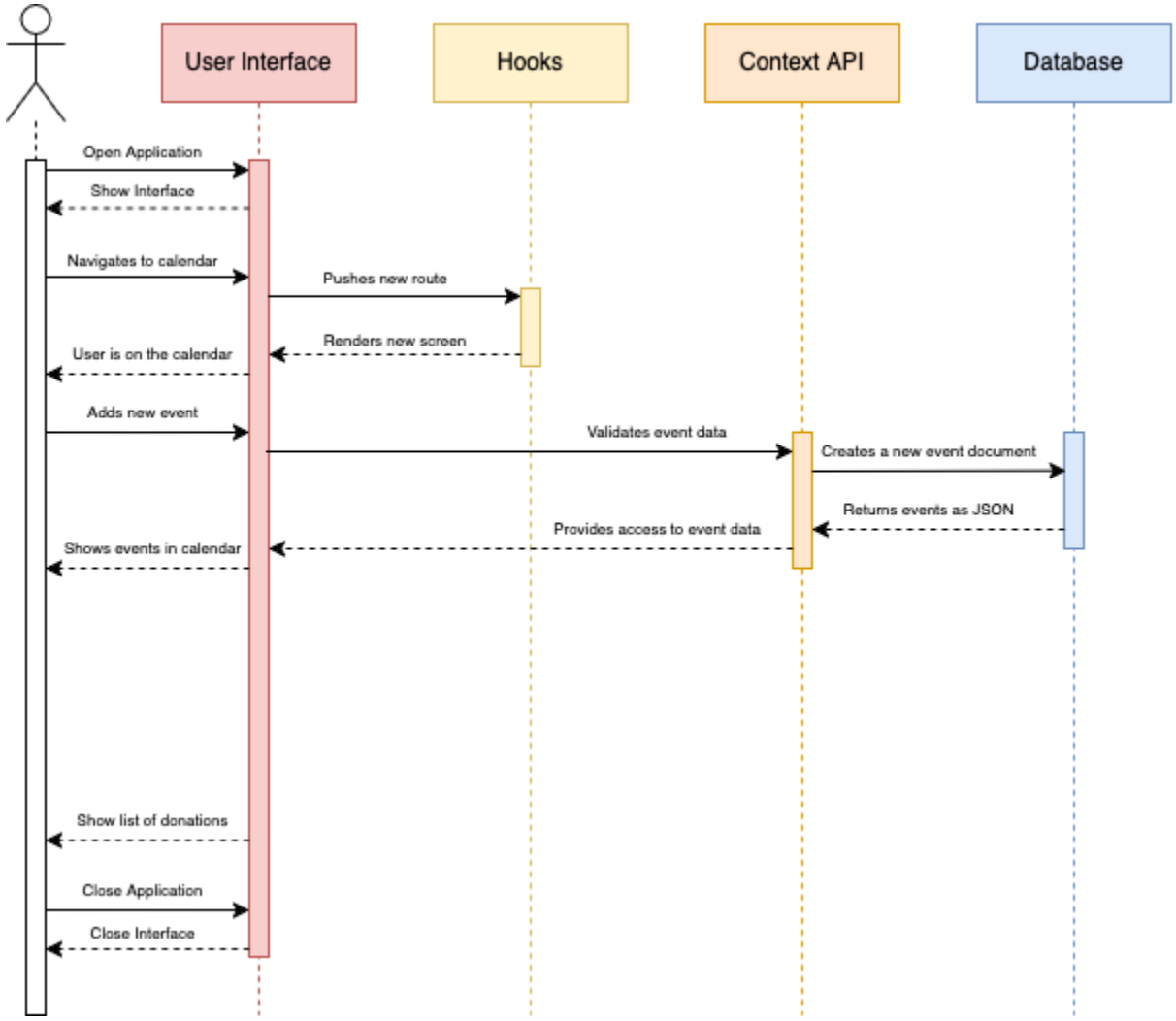


*Figure 2: Dynamic System Architecture*

### Description:

The Dynamic System Architecture diagram shows the interaction among the various elements of our system, and how they work together to respond to actions made by the user. The above diagram highlights the expected flow for a user when they want to add a new event to their calendar.

A user inputs information about their event such as the date, the time, and the event's description. Then, the User Interface will interact with the Context API to validate the data, and the Context API will finally query the database to add a new document representing the event. The events from the database will be provided to the Context API in a JSON-

parse able format. Furthermore, the Context API will expose this data to the frontend, so it can be rendered on the user's calendar.

We chose this feature because it is generalizable. For instance, the standard flow for modeling the creation of new donations follows a similar pattern. In general, interacting with the database for any feature will use a flow like event creation. For future developers, following this pattern will make it easier to scale our application and add new features.

# Component Detailed Design

## Introduction

The Component Detailed Design section provides a comprehensive breakdown of the key components of our application and how they interact with the front-end and database, as well as each other. The three main UI components of our app are the Calendar, Receiving, and Upcycling tabs. Each of these tabs is represented by a code-based component named accordingly (e.g. CalendarView represents the Calendar). Then, each of these code-based components has some interaction with the user and the database, as detailed in the diagrams below.

## Static Component Diagram

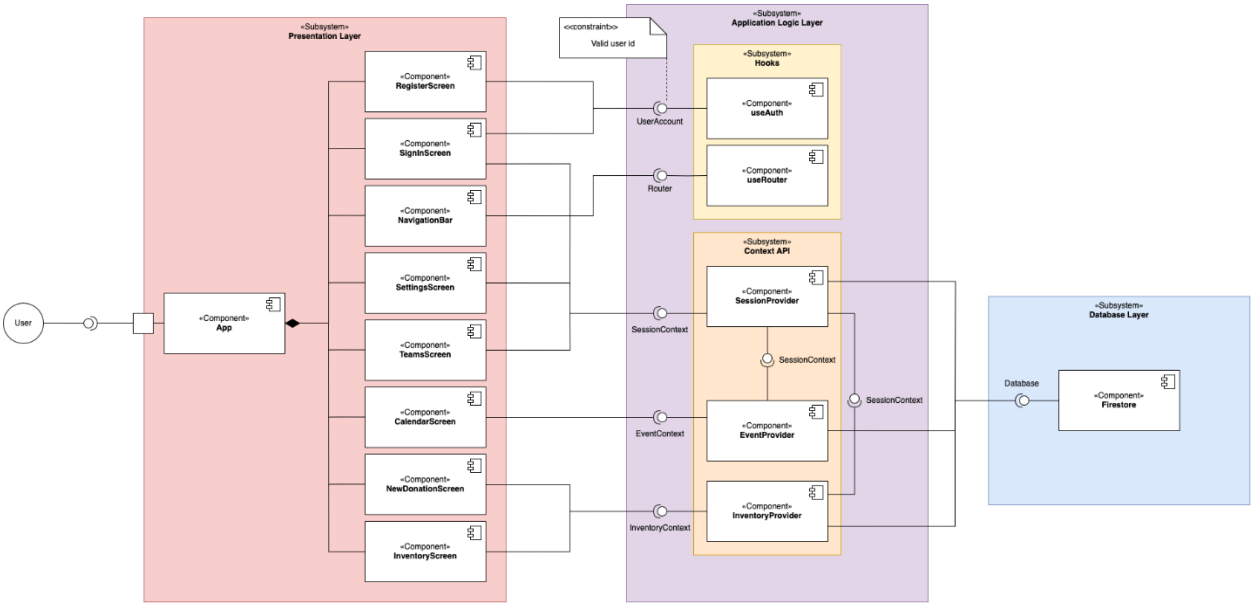


*Figure 3: Static Component Diagram*

### **Description**:

Our static system architecture consists of three primary layers: the Presentation Layer, the Application Logic Layer, and the Database Layer. Within the Presentation Layer, users interact with multiple components including the RegisterScreen, SignInScreen, NavigationBar, SettingsScreen, TeamsScreen, CalendarScreen, NewDonationScreen, and InventoryScreen. These components are all accessed through the core App component.

When a user signs in or registers, their credentials are passed through the useAuth and useRouter components within the Hooks subsystem of the Application Logic Layer. If the

user is validated, the session context is established via the SessionProvider. This context is shared across the app and used by EventProvider and InventoryProvider to manage event- and inventory-related data, respectively.

These providers interact with the Database Layer through Firestore, where data such as user accounts, scheduled events, and inventory records are stored and retrieved. For instance, when a user views the CalendarScreen or InventoryScreen, data is fetched through the respective context providers from Firestore. Similarly, when a new donation is added via the NewDonationScreen, the data flows through the InventoryProvider and is saved into the database. The entire system is modular, context-aware, and supports scalable state management across screens.

## Dynamic Component Diagram

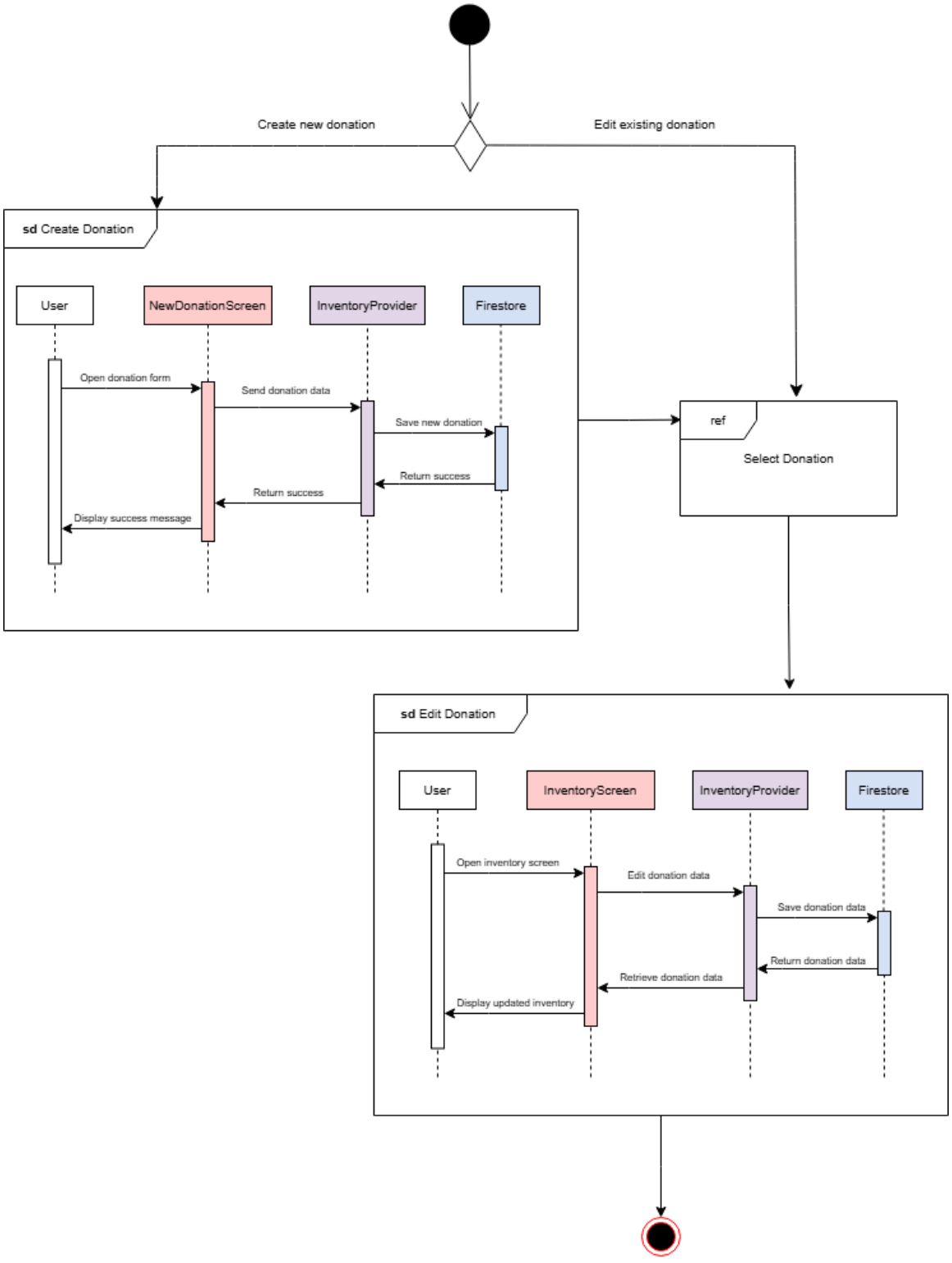


*Figure 4: Dynamic Component Diagram*

## **Description**:

The above is an interaction overview diagram depicting the scenario where a user may either want to create a new donation or edit an existing one. Should they choose to create a new one, they can open the donation form, where they will find the NewDonationScreen. Upon inputting all the necessary information, the data will be passed to the InventoryProvider and then saved in the Firestore. The Firestore will tell the InventoryProvider the new data is saved, which will get relayed to the NewDonationScreen, and a message detailing the success of the new entry will be displayed to the user.

Now, consider the scenario where a user wants to simply edit an existing donation. They may open the InventoryScreen, where they can edit data, passing it down the pipeline until it's saved in the Firestore. The Firestore will return the newly updated entry back up the pipeline, and the user will see the updated entry on their InventoryScreen.

Notice how there is an arrow from the Create Donation box to the Select Donation box. This is here to show that a user can, of course, choose to edit their new donation if they have just made one. However, should there exist items in inventory already, a user also has the ability to bypass the creation of a donation and go straight to the screen where they can view and edit what they already have.

# Data Design

## Introduction

We are using Cloud Firestore, a document-oriented NoSQL database, to store our data. In Cloud Firestore, the unit of storage is called a document. Each document contains fields that map to values, meaning each document is essentially a set of key-value pairs. The Firestore documents are their own thing, but in general they look like and can be treated like JSON files.

The documents in our Firestore live inside of collections. Collections are simply containers for the documents so that they look nice and organized if we ever need to visit them in the Firestore Console. One example of a collection in our Firestore is the “donations” collection, in which all the documents created upon receiving a donation reside.

## Data Design Diagram


*Figure 5: Data Design Diagram*

**Description**:

Each of the boxes in the diagram represents a collection. Once again, a collection is simply a grouping of documents, and the documents hold our data. There are 5 collections in our Cloud Firestore: Users, Organizations, Teams, Events, and Donations. Each of the lines denote many-to-one connections between collections, with the branched part of the line denoting the "many". We will explain what this means with an example. As you can see, there is a connection between Organizations and Teams, where the branches point towards the box for Teams. This simply means that there can be *many* Teams within *one* Organization. Furthermore, you shall notice that the line connecting the two is attached by the field "orgId", since this is how our application connects the two logically. The rest of the connections between each of the collections are denoted in the same manner. There is no particular way to read this diagram, although beginning from the left at Users and continuing right to Donations may provide the reader with the best sense of how our data flows between components of the application.

## File Use

As mentioned in the introduction, each created record is stored as a document in Cloud Firestore. These documents, however, are not actual files. They are simply sets of key-value pairs that can be easily referenced to store or retrieve data.

The only files that our app uses are various .png files (such as an Upcycle Build logo) that are simply received by one line of backend code and output as images on the UI for users to enjoy.

While the app doesn't really use many files, it does hold the ability to create them. A donation receipt .pdf file is generated each time a user clicks on the download button for the corresponding entry in the materials section. Upon generating this receipt, the user has many choices of what to do with it, one of which is storing it locally (such as to the Files app on an iOS device).

## Data Exchange

We do not exchange any data between any devices. All information is input manually into the user's device.

## Security Concerns

There are no current security concerns as our app and database do not deal with any sensitive information. Registered users' passwords are protected by a hashing scheme.

The client has, however, mentioned a few stretch goals to be expanded upon by future developers, outside the scope of this project. It is worth mentioning these goals and how they may bring security concerns, in case someone who is looking to continue the project is reviewing this document. The first of these goals is a map view integrated into the app, so that users can see nearby reuse centers. This of course would require the user's current location. A responsible way of handling this sensitive information would be to dish it out to Google's services. If one were to implement this map view by using Google Maps services, this concern should be taken care of due to Google's current robust solution of using such data.

The second of these goals is an image recognition scanner within the app that can identify the donated material and estimate its weight and value without requiring the receiving clerk to manually type it in. There is nothing inherently dangerous about something like this, as the scans will all be of inanimate construction materials. However, there may exist a case where someone's face is caught in the background of an image, or another similar scenario like this. A future developer would have to consider privacy concerns for something like this, either by introducing a privacy policy or making sure the image is not saved anywhere once it has been scanned.

# UI Design

## Introduction

This section details the User Interface of our mobile application. We will provide a walkthrough of the app, showing the important screens that users will encounter depending on the task they are trying to accomplish.

The user opens the application to the Receiving screen, in which they are presented with a donation form that they are required to fill out to log a donation (Figure 6). The donation form contains all the important information that a reuse center would need to keep track of when accepting donations including donor information, date and time received, item description and quantity. A user can also add multiple items to a single donation upon clicking “Add item”.

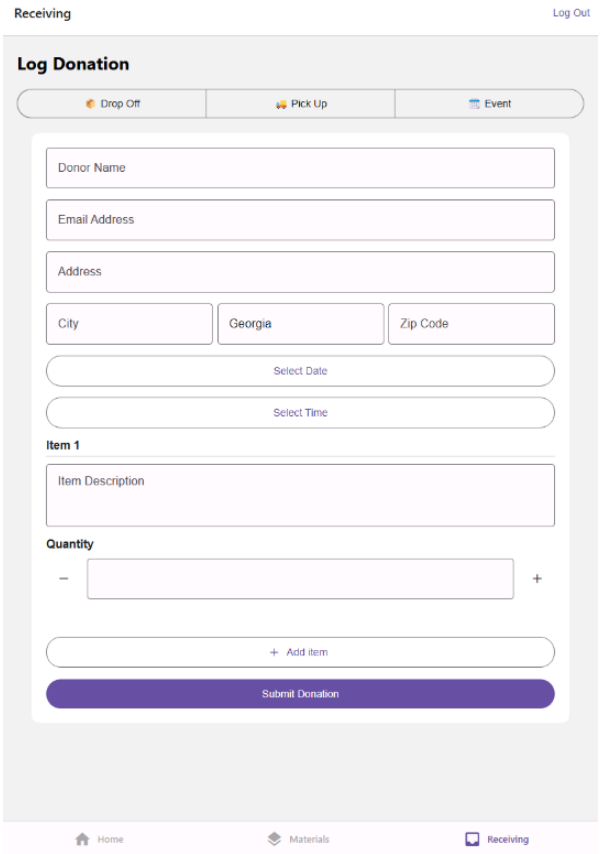


*Figure 6: Donation Form on Receiving screen*

This screen encapsulates the matching between the system and real world heuristic. It is set up much like a paper version of the form would be laid out, so users should not be

confused when trying to find certain fields. Additionally, it allows for flexibility and efficiency of use, which is made especially apparent by the ability to add an item and increment/decrement quantity. This allows the user to either add multiple items at once, or submit them all on separate forms, whichever is easier for them. Finally, it provides an aesthetic and minimalistic design, including only the necessary fields with no clutter. The emojis at the top make for an aesthetically pleasing look.

Upon successful entry of a donation, a reuse center employee may want to view the updated list of inventory. Upon clicking the “Materials” tab at the bottom of the page, they will be sent to a screen that lists the entire inventory for that reuse center (Figure 7). The top of the page allows users to filter records by method of reception and the donation’s status. Each record shows the information logged from the donation form on the previous screen such as the donor’s name and description of the item. Additionally, users can tap entries to edit them by adding comments, updating refurbishing status, and removing items that have been deemed as unsalvageable (Figure 8).

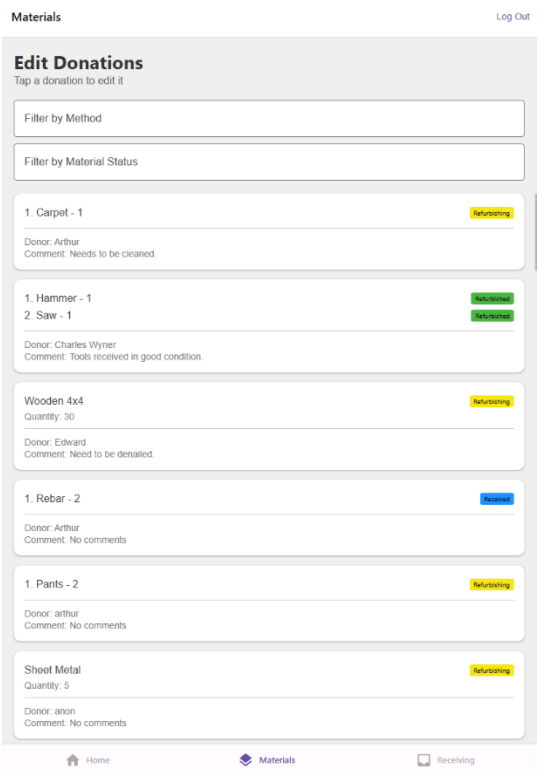


*Figure 7: Materials screen listing inventory records*

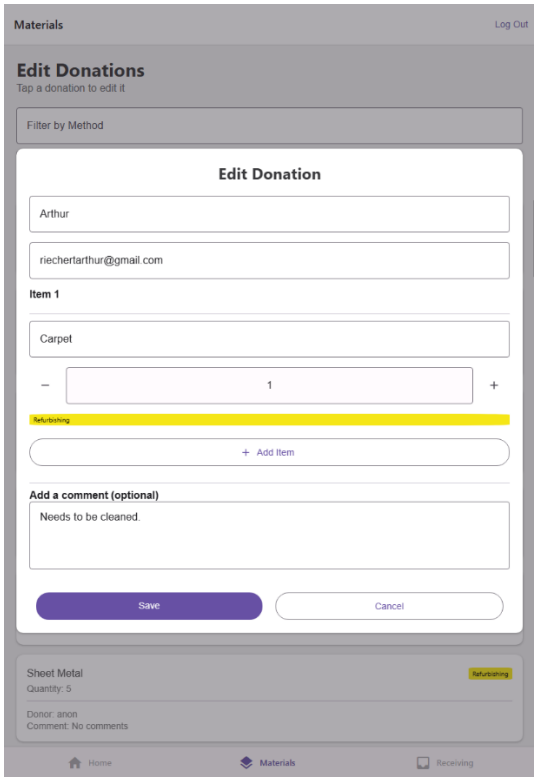


*Figure 8: Edit Donation screen*

These screens follow the consistency and standards heuristic, because the screen that shows up when an item is selected looks just like the donation form seen in Figure 6. There is also clear user control and freedom; if an item is accidentally clicked in the inventory screen, there is a cancel button at the bottom, and no changes will be saved. Finally, the inventory screen shows clear visibility of system status. If an item is edited, it will immediately show the updated status in the inventory screen. The colored icons also make it very clear what an item's refurbishing status is at any given time.

The last of the three main screens is the Calendar screen. This screen provides reuse center employees with a way to see all their activities for the day, including incoming drop-offs and upcoming pick-ups that they have to make (Figure 9). Users can also add events by clicking the "+" in the bottom right as well as edit the details of existing events by tapping on them (Figure 10).

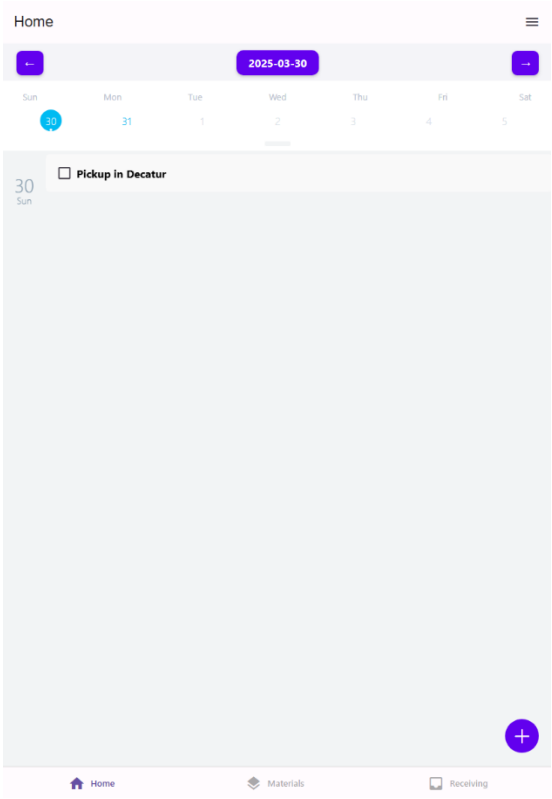


*Figure 9: Calendar screen*

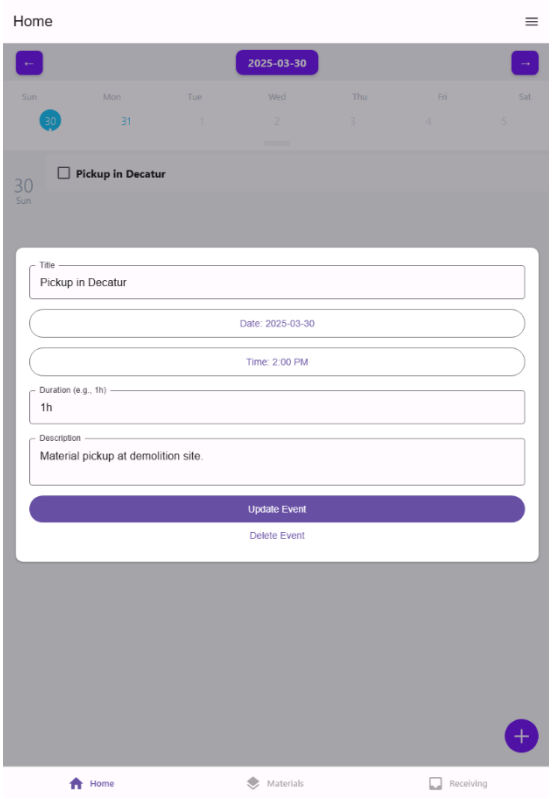


*Figure 10: Edit event*

Once again, these screens follow the "consistency and standards" heuristic. While it is a different form than the last, it still retains much of the same structure and it should be very clear to a user what they are meant to do. The hamburger menu in the top right is also a standard among many mobile applications, so the user will most likely identify that this is where they can find the settings page. The heuristic of recognition rather than recall is also

present here. The arrows at the top make switching between weeks an intuitive process, and the plus button in the bottom right makes it easy to add new events. Finally, this screen also practices user control and freedom; if an event is accidentally created, the user can easily delete it with the button at the bottom.

# Appendix

## Team Member Contributions and Contact Information

The Upcycle Build team consists of four dedicated members, each contributing unique expertise to develop a functional, user-friendly solution aimed at streamlining inventory, upcycling, and task management for community reuse centers.

| Name | Role | Contributions | Emails |
|---|---|---|---|
| Arthur Riechert | Developer and Documentation | Frontend<br>Backend<br>General technology wizard<br>Documentation | ariechert3@gatech.edu |
| Bereket Yoseph | Developer and Scribe | Frontend<br>Backend<br>Scribe of all things spoken<br>Documentation | byoseph3@gatech.edu |
| Charles Wyner | Team Leader | Frontend<br>Backend<br>Client correspondence captain<br>Documentation | cwyner3@gatech.edu |
| Edward Jin | Developer and Documenter | Frontend<br>Backend<br>Resident diagram maker and wordsmith<br>Documentation | ejin32@gatech.edu |