Backpack Buddy

Authors: Joon Cha, Aaron Li, Janet Li: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Backpack Buddy is a backpack-based smart inventory system meant for on-the-go students. Based on which tagged items are packed inside a bag or backpack, it utilizes Bluetooth Low Energy tags (similar to Tile) on items for active asset-tracking in a backpack. Unlike existing technologies, which only focus on tracking of individual items, Backpack Buddy's web application allows users to manage collections of items in relation to their schedules.**

***Index Terms*—Asset-tracking, Bluetooth Low Energy, Image recognition, Raspberry Pi Zero, RSSI, Smart inventory, Web application**

1 INTRODUCTION

One of the most common struggles students face is trying to keep track of their items between all the classes, meetings, sports practices, or other events they may have. Students can lose things like their wallets, keys, water bottles, and even skateboards (as can be seen in the Facebook group "CMU Lost and Found"). It's clear that there needs to be a better way for students to keep track of their items with their busy schedules.

Existing solutions involve social groups like "CMU Lost and Found" – however, this relies on the kindness of strangers, the luck that your missing items are found, and the fact that students are only notified retrospectively after their item has been lost – and the Tile application, where tags are attached to each item and are tracked individually. However, this often entails a large burden on the user and isn't manageable for large collections of items. Our project, Backpack Buddy, offers a better asset-tracking solution by focusing on *where* these items are stored to be moved around: the backpacks and bags of students.

Backpack Buddy is a smart inventory system that involves three components: (1) BLE tags on items to-be-tracked (the same technology that Tile uses in their tags), (2) A backpack-based scanner which keeps track of which tagged items are inside the backpack, and (3) A mobile-accessible web application which allows users to manage their items in relation to their schedules. This application will also have an image recognition component to streamline the item registration process.

For each of these components, we have specific design requirements. The BLE tags should be easy to attach and unobtrusive, no larger than 40 x 40 x 10mm. The scanner should have a less than 1 second delay for tag detection per tag and be able to accurately detect 10 items within a 0.5-meter range (± 10cm) with 100% accuracy and last 18 hours without recharge while fully operating. For the phone application, the interface delay between when an item is added/removed to the backpack should be less than 3 seconds per item, and the item recognition built into the application should have at least 60% accuracy (i.e. out of the three names suggestion, the correct name should be one of the options at least 60% of the time).

2 DESIGN REQUIREMENTS

We've planned tests for each of the requirements that our system must meet. For the tag communication and detection, we require that the scanner should have less than a 1 second delay for tag detection and be able to accurately detect 10 items within a 0.5-meter range (± 10cm) with 100% accuracy and last 18 hours without recharge while fully operating. We will test the latency for tag detection by placing a tagged item into the backpack and noting when the scanner is able to receive its signal. We'll plan to tag 10 items within a 0.5-meter range (± 10cm). More strictly, we'll place these 10 items within the backpack itself and test whether the item list from the scanner displays all 10 ID's of the tags correctly. This is a significant requirement, as it lies at the core of our project's goal and interference from BLE tags may prove to be a significant problem (as further discussed in the "Risk Management" section). To test the battery life of the backpack device, we plan on changing the item list, either by adding or removing a tagged item from the backpack, every 30 minutes to avoid low power consumption and checking whether the system is still functional after 18 hours.

The item recognition component will suggest three names for each item based on the three highest ranked options from the model. Our 60% accuracy requirement entails that one out of the three suggested names is the correct one. We'll test this requirement thoroughly with a curated dataset of 21 common items a student would carry and see whether the model is able to correctly identify each image within its top 3 highest suggestions.

To test the interface delay requirement (less than 3 seconds) from an item change in our system, we'll be calculating both the delay between when an item is added/removed and the detection by the scanner as well as the delay between when the scanner reports the new item list and when our interface will update. The total interface delay will be the sum of these delays.

We will also need our interface application to manage scheduling up to 25 weekly events. We chose this requirement of 25 weekly events since that's about the average number of events a student might attend per week. Events will be stored in a database on the AWS server. These events can either be manually created by the user or imported from Google Calendar, depending on what the user

decides. Each event requires a subset of tagged items to be assigned to that event. This can either be done manually or through schedule learning.

Schedule learning is a feature that collects data about which tagged items are inside the backpack at what times. Using the user's schedule of events, it automatically handles the item assignment process (i.e. assigns items to events). To avoid prolonged testing (e.g. it's unfeasible to collect 2 weeks of data each time we'd like to test this feature), we'll be creating synthetic data for two parts: a fake collection of data for which tagged items are placed inside the backpack at specific times and days of the week, and the "correct" schedule of a user that will be used to determine how accurate our schedule learning is.

Once our project has met these requirements, we've explicitly planned in our schedule to make time for usability testing. We'll be testing whether our entire system is usable from a recruited group of users and follow a think-aloud protocol as well as a retrospective survey on the usability of our system.

3 ARCHITECTURE OVERVIEW

There are three main modules to our project: the BLE tags and the scanner, the web application to manage items and events, and the image recognition for item registration.

The BLE tags will transmit their UUID information via Bluetooth to the scanner, which will be an RPi Zero attached to a backpack or bag. The RPi Zero will use a distance-pruning algorithm (which we will write ourselves) to determine which UUID's are coming from tagged items within our 0.5-meter (± 10cm) range, simulating whether or not an item is actually inside the backpack. Once the list of items within the backpack is determined, this list will be communicated to our web application using Bluetooth.

Our Web Application will receive the list of UUID's from the RPi Zero and use it to create the checklist interface where users are able to track which items are currently inside their backpack. This list of UUID's also helps feed information about which items are missing for a specified event. As shown by our entity-relationship diagram (see Figure 2), every user has multiple events and multiple tags, and each event contains a subset of all tagged items. This means that multiple or no items are associated to each event that the user has, as shown by the intersection table "EventItems". These events can either be created manually or imported as empty events from the user's Google Calendar and automatically populated with items from the schedule learning feature.

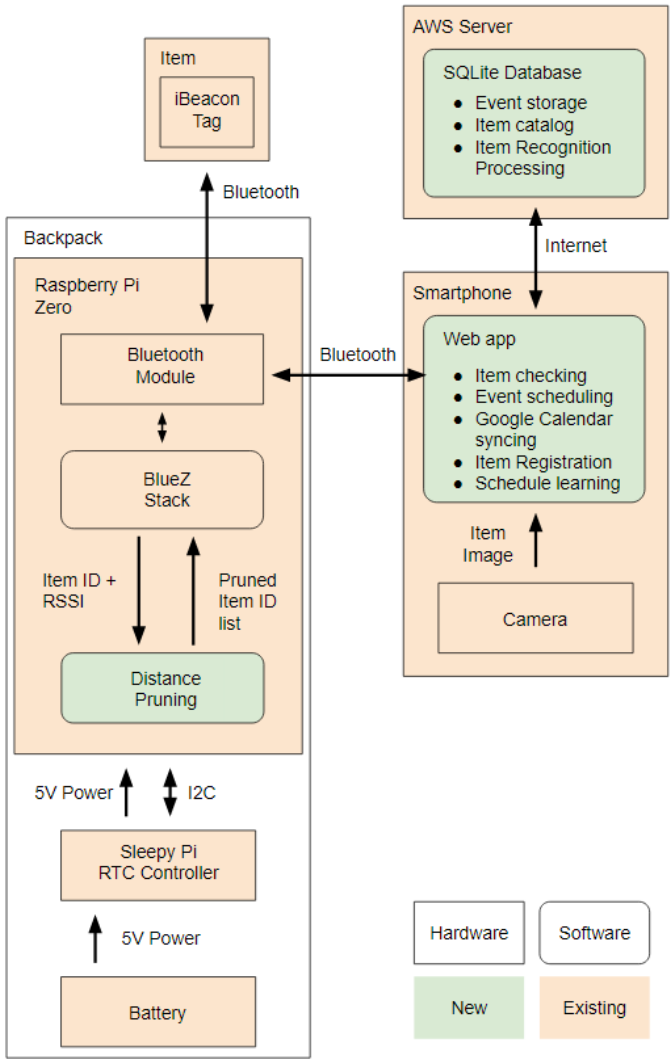

Figure 1: Overall System Block Diagram

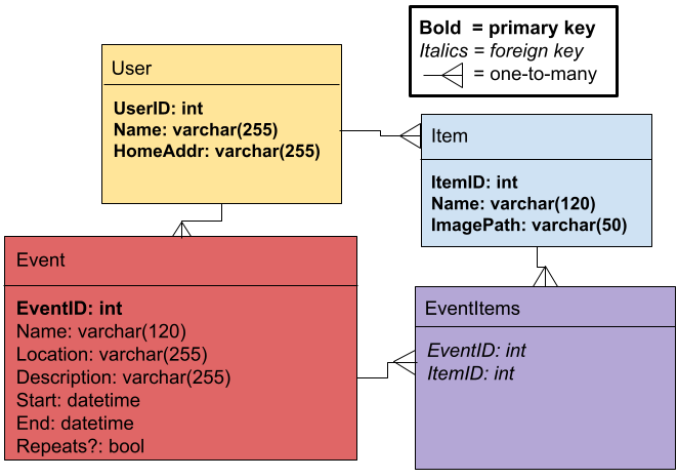

Figure 2: Entity-relationship diagram

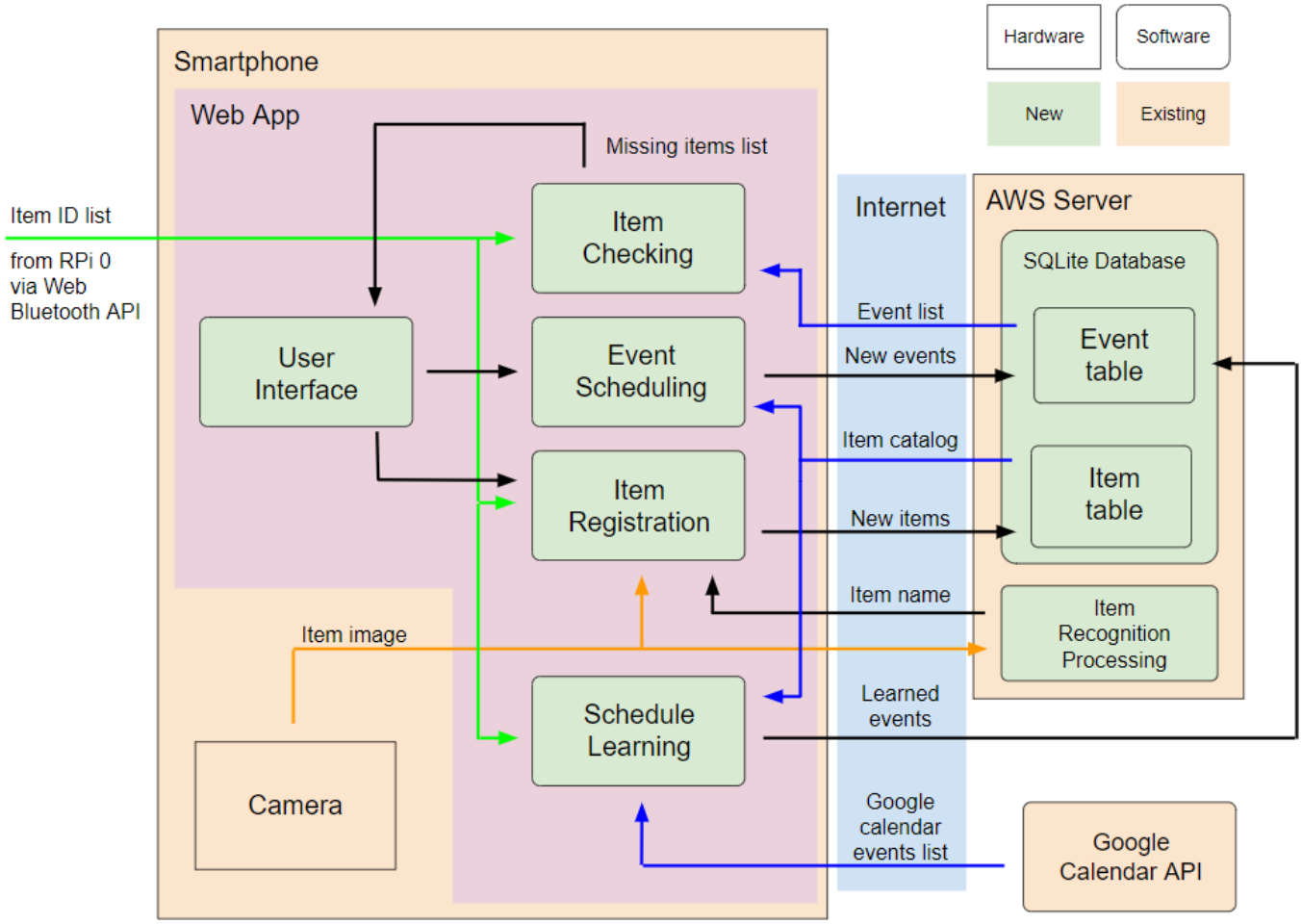

Figure 3: Software Block Diagram

To learn the user's schedule, the schedule learning feature collects data about which tagged items are inside the backpack at what times. By correlating these times with events from the user's Google Calendar, the schedule learning system can determine which items are most often located in the backpack during each event. After obtaining two week's worth of data, the schedule learning system automatically assigns items to the events based on appearance frequency. The system continues to refine the item lists as the user attends more events. This automatic assignment of items saves the user the burden of having to assign items to events manually.

Our image recognition will be built into the web application during the onboarding process when users are first registering their tagged items. Using a phone camera, the user will take a picture of the item with a physical tag. Figure 4 describes the image recognition and classification process using Python. A Convolution Neural Network (CNN) model, which is trained with scraped images from a search engine, will identify each image. These scraped images are augmented by image processing techniques to increase the size of the training data. Through the CNN model, the classification of the input image is given. The three highest ranked classifications will become the top three suggestions for the item name.

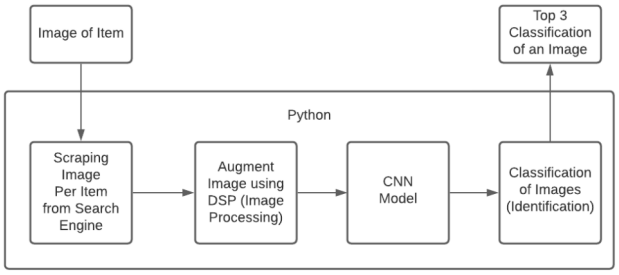

Figure 4: Overall Image Recognition Block Diagram

4 DESIGN TRADE STUDIES

4.1 Controller Comparison

We chose to use a Raspberry Pi Zero W for Backpack Buddy rather than other alternatives, such as an Arduino or an FPGA. This was because writing a Bluetooth communication protocol program for either the Arduino or FPGA was infeasible for our project. On the other hand, the Raspberry Pi Zero W runs on the Linux-based Raspbian operating system. We can use the built-in BlueZ Bluetooth protocol stack for Linux to handle Bluetooth communications, including scanning for tags and connecting to the user's smartphone. We chose the Raspberry Pi Zero W over other Raspberry Pi models for its small form factor, low power consumption, and built-in WiFi/Bluetooth antenna.

4.2 BLE vs. RFID

Once our team decided to build a smart-inventory system, we had to compare existing asset-tracking technologies. Two of the most important considerations we made were budget constraints and integration with other components of our system, such as the web application. As shown in Figure 5, BLE best fits our needs in terms of budget and integration, particularly for smartphone compatibility [8]. Although RFID tags are quite cheap, their reader costs far overextend beyond our allotted budget (1 would cost anywhere from $800-$1600, which much overextends our allotted budget). Individual BLE tags are more expensive than RFID tags, and also have a limited battery life (2-5 years), however the tags can be read with any common Bluetooth scanner, and BLE technology is already very commonly used in many asset-tracking commodities such as Tile. With its simple integration, smartphone compatibility, and budget-friendly reader cost, BLE is the correct asset-tracking technology for our project.

Features	Radio Frequency Identification (RFID)	Bluetooth Low Energy (BLE)
Response	Identification	Identification + Positioning + Sensor
Frequency Range	3 - 5 m per reader	5 – 50 m per reader
Reader Range	1 - 5 ft.	20 ft.
Reader Cost	$1500 - $2000	$10 - $70
Tag Cost	$0.10	$20
Enabled Sensor	No	Yes
Memory Storage	No	Yes
Integration	Difficult	Easy
Smartphone Compatibility	No	Yes
Battery Life	Really Long	2 - 5 Years

Figure 5: BLE vs. RFID

4.3 BLE Distance Analysis

To determine which items are located within our backpack, we are using a distance-based gating algorithm. The approximate distance of each item to the Raspberry Pi is calculated from the received signal strength indicator value associated with the item's Bluetooth signal. To do the conversion from power level in decibels of milliwatts (dBm) to meters, we are using a formula published by David Young in a blog post about contact tracing for COVID-19:[11]

$$d = 10^{\frac{p-s}{10n}} \quad (1)$$

In this equation, d represents the distance in meters, p is the measured power at 1 meter distance in dBm, s is the received signal strength in dBm, and n is a calibration constant which represents how quickly the signal decays in air. The constant p can be experimentally determined by averaging the signal strengths of multiple beacons at 1 meter distance from the Bluetooth sensor. The constant n can

then be determined by measuring the signal strength at varying distances from 0.5m to 2m in 10cm increments, and fitting the curve using least-squares.

Bluetooth distance estimates are not accurate enough to provide centimeter-level precision, and the precision worsens as the distance between the beacon and Bluetooth sensor increases. For our purposes, we do not need precise distances, and the distance between the beacon and Bluetooth sensor (the Raspberry Pi Zero) is short (less than 1 meter) for the beacons whose distance we care about. Therefore, Bluetooth distance estimation is suitable for our application and eliminates the need to use triangulation or other localization techniques to determine which items are in the backpack.

4.4 Web App vs. Native Phone App

In both our proposal presentation and design review presentation, we planned on developing a phone app for Android phones in Kotlin using Android Studio. We chose to develop specifically for Android devices for ease of testing, as all three team members use Android phones. We chose Kotlin over Java for its coroutines, code safety, and other features that would be advantageous to our project.

However, we quickly ran into issues with Android Studio. Our two main concerns were the clunkiness of collaboration on Android Studio as well as the numerous errors with Gradle, Android Studio's builder.

Since our main focus is building our MVP as quickly in preparation for the interim demo, we decided that the best course of action would be switching to developing our interface application as a web application. Not only does this streamline collaboration between our team members, it also offers more accessibility from more platforms instead of Android phones.

4.5 Integrated Device

Our system incorporates a web app loaded on the user's phone to display the items present inside of the backpack. Another potential approach was to have an integrated LCD panel in the backpack which would display the items present, rather than a web app. The panel would have a set of arrow and control buttons next to it to enable navigation and configuration of the system. However, this integrated system would have a number of disadvantages versus a smartphone-based app.

First, the usability of our system would be reduced with an integrated system. With a smartphone app, the user can easily pull their phone out and quickly view what items are in their backpack. Additionally, users are more accommodated to navigating and using smartphone apps, versus a novel integrated system where the user would have to learn how to use the arrow keys to navigate the options on the LCD panel. Although touchscreen LCD panels are available, they are more expensive than non-touchscreen LCD panels, and often have worse touch detection than common smartphones. Since most users already own a touchscreen in the form of their smartphone, it was more sensible for our system to integrate into the smartphone rather than have its own dedicated touchscreen LCD panel. Moreover, it would be difficult to design an integrated system that would allow the user to interact with the system while wearing the backpack. Such a design would require having the LCD panel and controls on a separate arm that would extend from behind the user (where the backpack is) to the front. Having a smartphone-based app means the user can simply look at their smartphone while wearing the backpack and still have view and control of the system.

Second, the smartphone app enables more features than an integrated system. The smartphone app allows the user to schedule events and register new items which are tasks that would be difficult to do with an integrated system. Additionally, most smartphones have built-in cameras, so a smartphone app allows the user to take images of new items for the registration process. To implement this functionality on an integrated system, we would have to include a camera in the backpack, which would likely be lower quality than a smartphone camera and increase cost. Smartphones usually have cellular data, meaning that the app will have constant access to the cloud server where the events and items are stored. Although an integrated system could have WiFi access, the integrated system's internet access would be cut as soon as the user left their home WiFi network, rendering the system unable to communicate with the cloud server while the user is travelling.

Given these factors, we decided to use a smartphone-based web app as the user interface for our system, rather than having a system fully integrated into the backpack.

4.6 Image Recognition Model

For the image recognition algorithm, a suitable machine learning model needs to be trained. Among various machine learning models, including Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, we chose to use a Convolutional Neural Network (CNN) to accomplish our image classification.

CNN's are the best choice for image classification for two main reasons. First, whereas other machine learning models first require a separate step of feature engineering before the model can be trained, CNN's automatically extract features from the image datasets and therefore do not require feature engineering. Using the learned features, CNN's use multiple layers extract the feature weights through downsampling and convolution.[9] Finally, the last prediction layer uses the feature weights to output the classification result of the image.

The second reason the CNN model is superior to other models is that CNN's are scalable for large datasets. In order to produce an accurate classifier, the model needs to be trained on a large number of labeled student items. Normally, the large number of images leads to more parameters in a neural network, which makes the training of the model exponentially more computationally heavy. However, the multiple convolution filters present in the convolution lay-

ers of a CNN effectively reduce the dimensions of each the image, reducing computational load. Even though the image dimensions are reduced, the images still retain the information from the original image, resulting in efficient but accurate image classification.[9]

4.7 Image Recognition Datasets

To train the CNN model for the image classifier, the dataset of images needs to be generated. In machine learning, we need to increase the size of the training data to achieve higher accuracy. One option of choosing an image dataset is to use a pre-existing dataset such as ImageNet. ImageNet is used for various CNN research papers and is an appropriate dataset for the image classification task. However, since we only need to identify 21 common items that students would carry in their backpacks, a dataset containing a variety of images will decrease the classification accuracy for the item recognition. Therefore, we decided to scrape the images of student items from the search engine, DuckDuckGo, using an Image Dataset Tool (IDT) image scraper library.

There are several reasons that the IDT library is superior to other software libraries. First, the library is designed around creating datasets specifically for training machine learning models such as the CNN model for image classification. IDT supports separation of training data and test data, which prevents the model from training on test images and potentially skewing the test results.[5] Second, the library supports standardization of the images, as images from search engine results are of different sizes and aspect ratios. Using the IDT library, images can also be downscaled and compressed for more efficient storage on our AWS server while we are training the CNN model. Finally, the library allows for multiple searches with different keywords. For example, when searching images for a reusable water bottle, a user will search it by typing different keywords such as “water bottle,” “reusable water bottle,” or “plastic bottle” for the same results. The IDT library enables the user to put multiple keywords so that the user can collect various images to better train the machine learning model.

5 SYSTEM DESCRIPTION

The hardware portion of our system consists of three parts: 10 iBeacon tags attached to individual items, a backpack-mounted Raspberry Pi Zero W with a rechargeable battery and power controller, and the user’s smartphone. The control flow for our system starts with the iBeacons. Each iBeacon tag broadcasts its UUID over Bluetooth every 1000ms. Then, the Raspberry Pi scans for tag UUID’s and determines which tags are close enough to be considered inside the backpack based on the received signal strength of each broadcast. The Raspberry Pi then transmits the list of UUID’s of the tags inside the backpack to the user’s smartphone via a Bluetooth connection. Finally, The user’s smartphone handles events, item information, and notifying the user when they are missing items.

5.1 iBeacon Tags

The iBeacons used by our system are the H1 Beacons from Moko Technology Ltd.[4] These beacons are based on the NORDIC nRF52 chipset and have adjustable transmission power and broadcast intervals. For our project, the H1 Beacons will be set to transmit at the lowest power level (-12 dBm) and longest interval (1000ms) available. This provides the maximum possible battery life for the beacons, which is approximately 15 months as specified by the datasheet. The dimensions of the H1 Beacons are 32.5 x 32.55 x 7mm with a weight of 7.8g. This is small enough and light enough to be securely attached to most user items. The beacons also feature a keyring hole, which allows the beacon to be attached via a keyring to an item if the user so desires. The beacons also have flat backs, and can be attached via double-sided adhesive tape to items for which a keyring is not suitable.

5.2 Raspberry Pi Power Management

To reduce power consumption and conserve battery life, our system uses a Sleepy Pi 2 from Spell Foundry to shutdown and wake the Raspberry Pi at predetermined times.[3] The Sleepy Pi 2 has an onboard Real-Time Clock (RTC) which enables it to track the current time even while the Raspberry Pi is powered off. The Sleepy Pi 2 also features a user button, which can be used to manually wake the Raspberry Pi if needed. For our system, the smartphone app actively tracks gaps between events, and it transmits a shutdown signal and wakeup time via Bluetooth to the Raspberry Pi for sufficiently long gaps. The Raspberry Pi sends the wakeup time to the Sleepy Pi 2 via I2C before shutting down, and the Sleepy Pi 2 wakes the Raspberry Pi once the wakeup time arrives. Should the user need to use the system while the Raspberry Pi is asleep, the user can press the user button on the Sleepy Pi 2 to manually wake the Raspberry Pi before its wakeup time. The Raspberry Pi and Sleepy Pi 2 combined system will be powered by an Aukey 10000mAh portable USB LiPo power bank. The user will be responsible for plugging in the power bank to charge every night so that the power bank will have full charge the next day.

5.3 Raspberry Pi Software

The Raspberry Pi Zero in our system is running the Raspbian Lite operating system. This OS only has a command-line interface instead of a graphical desktop, to reduce the computational load on the Raspberry Pi. At startup, the Raspberry Pi runs a Python script which continuously scans for Bluetooth beacons. The script utilizes the bluepy package created by Ian Harvey to control the Linux BlueZ Bluetooth protocol stack from Python.[7] The script is based off of a similar project done by Github user

nullhart, who created a proximity-based light switch using Bluetooth.[1] The distance to the beacons is determined by converting the received signal strength indicator (RSSI) value into a distance using Equation 1 found in section 4.3. The constant n in the equation is set beforehand via calibration tests done with power level measurements of multiple tags at a set one meter distance. After excluding all beacons whose distance is determined to be greater than 0.5m, the script then sends a list of beacon UUID's to the user's smartphone over Bluetooth. To facilitate this connection, the Raspberry Pi continuously broadcasts itself over Bluetooth until a pairing request is made. The Raspberry Pi accepts the pairing request, and subsequently sends the item UUID list to the paired device every 1000ms while it is awake.

5.4 Smartphone Web App

Our web application will be built using the Django framework. We'll be using the Web Bluetooth API [10], which allows us to communicate with other Bluetooth devices via JavaScript, to receive the list of UUID's of items currently inside the backpack from the RPi Zero. This API retains the full Bluetooth functionality of a native phone app, including requesting and connecting to nearby BLE devices, reading/writing Bluetooth characteristics & descriptors, and receiving GATT notifications when devices get disconnected. The web application will use this list of UUID's to create the checklist interface, where users can track which items are currently inside the backpack.

The web application will also manage events and display them on a calendar-like interface. By design of our system, users will have multiple events in their schedule, where each event contains a subset of items from the entire list of tagged items (i.e. each event can have multiple or no items associated with it). Users have the option to manually create new events through the web app or to import empty events from their Google Calendar (using the Google Calendar API). In addition to managing events, the web application provides a separate interface for users to register their items. The user is asked to take an image of each new item. This image is then sent to the image recognition algorithm, which suggests a name for the item to the user. The user can choose to either accept the suggested name or manually override the name. Once the items and events are created, the user can choose to either assign items to events manually, or have the schedule learning feature automatically assign the items to events. All events, items, and assignments of items to events are then stored on the AWS server database for the web application to retrieve later.

5.5 Schedule Learning

The schedule learning feature monitors which tagged items are present inside the backpack at specific times and days of the week. After two weeks of monitoring, the feature will produce an initial automatic assignment of items to events. These assignments will continue to be refined after this two week period as more data is collected. The automatic assignment of events reduces user burden by eliminating the initial registration and manual assignment of items to events. Users can also customize here when they'd like to begin receiving notifications for each event in their schedule (e.g. "Please notify me if I'm missing any items for 'Soccer Practice' starting from 1 hour before the event").

5.6 Image Recognition

Our image recognition will be primarily developed using Python. We chose Python because it is the most commonly used language in machine learning development and can be easily integrated with the web application, also written in Python. Within Python, the CNN model will be built using the Tensorflow library. The overall CNN model will be trained on an AWS server.

The CNN model is trained using a dataset of images from the search engine DuckDuckGo scraped with an Image Dataset Tool (IDT) library. Through the IDT library, images of the 21 identified student items are collected. 250 images per item are collected for the training data and 150 different images per item are collected for the test data. The images used for the test dataset are scraped and saved in different directories for testing purposes. Then the training data images are augmented with image processing techniques such as image distortions and mirroring to create up to 1000 images per item (from 250) using Python and MATLAB. This image augmentation is performed to create multiple variants of a single image. This also accounts for the user taking distorted or slanted photos of items during registration. The training image is then resized to 256x256 pixels and the CNN model classifies the image into one of 21 classes of items. The model will output the top 3 highest matching classifications and communicate these to the web application. The user can select the correct classification result or manually input the information if the classification results are inaccurate.

6 PROJECT MANAGEMENT

6.1 Schedule

Our schedule (see Figures 7 and 8 in Appendix A) has remained largely the same since the beginning of the proposal presentation. However, we have now included "schedule learning" as a major subsystem of our project on the schedule with relevant subtasks. This change was made following the proposal presentation from a suggestion from Professor Kim, our advisor. Schedule learning involves learning, over a period of time, what items the user brings at what times. However, this component will not remain a focus of our project until after the interim demo in Week 11. Additionally, we have 2 weeks of slack time built into the beginning and end of our schedule.

From this week (Week 7) of the design report onwards,

we'll be largely focusing on completing our MVP before the interim demo in Week 11. We plan to have finished the most pivotal component of our project, the communication between the tag scanner and the phone app, between Weeks 9 and 10 so that integration between each of our respective modules (phone app, item recognition, hardware) can happen. Once we finish integrating our parts of the project, we'll follow with implementing the schedule learning featuer as well as rigorous testing and debugging of our system. At the very end, we'll be spending considerable time on our final presentation and report.

6.2 Team Member Responsibilities

We've divided up each team member's primary responsibilities based on the separate modules of our project as well as the ECE area strengths of each member. Aaron will be responsible for the tags and Bluetooth scanner, Janet will focus on developing the phone application, and Joon will implement the item recognition component. All three members will work on the schedule learning feature. For each team member's secondary tasks, Janet and Aaron will work together on the communication between the Bluetooth scanner and the phone application, and Janet and Joon will work together on the integration of the item recognition component into the phone application.

6.3 Budget

Item	Source	Quantity	Who	Total
Raspberry Pi System				
Raspberry Pi Zero WH	Adafruit	1	Aaron	$19.88
SanDisk 16GB MicroSD card	Amazon	1	Aaron	$6.16
Mini HDMI to HDMI cable	Amazon	1	Aaron	$6.32
Sleepy Pi 2	PiShop.us	1	Aaron	$58.90
Power				
Anker Portable Charger 10000mAh	Amazon	1	Aaron	$19.07
One-Port USB Wall Charger	Amazon	1	Aaron	$9.00
USB C charging cable	Amazon	1	Aaron	$4.12
USB Micro B charging cable	Amazon	1	Aaron	$5.56
Bluetooth				
Bluetooth Low Energy Beacon	Alibaba	20	Aaron	$192.00
Double sided tape	Amazon	1	Aaron	$7.41
Assorted Split Key Rings	Amazon	10	Aaron	$15.58
Tile tracker tag	Amazon	1	Aaron	$26.49
Feasycom iBeacon tag	Amazon	1	Aaron	$16.95
AWS				
AWS credits	AWS	3	All	$0.00
Total				**$344.00**
Budget				**$600.00**
Amount Leftover				**$256.00**

Figure 6: Bill of Materials

The bill of materials for our project can be found in Figure 6. The items are separated by which component of the system they are for. The green and red items are part of the backpack system, with green being supporting components for the Raspberry Pi Zero and red being power-related items. The blue items are the Bluetooth-related items for tracking and mounting to individual items. Finally, the yellow item is the AWS credits which we are using for our AWS server database. We mostly chose parts supplied by Amazon, which reduced shipping costs as Amazon Prime has free shipping. We also elected to purchase a Tile Bluetooth tracker tag to compare the performance of our system with that of a similar existing product. We only purchased one copy of each of the hardware items, since Aaron was the only team member responsible for the hardware of our project. For software, we requested AWS credits for all three team members as we are all contributing to the software portion of the project.

6.4 Risk Management

We knew that our team had to be vigilant when it came to communication and risk management plans, especially as the three of us are located in three different time zones (US Eastern Time, US Central Time, and Korean Standard Time). Because our system is implemented in three distinct modules, we also knew that many issues could appear during integration, and this is why we have set aside time specifically dedicated to integration tasks.

One of the largest risk areas of concern in our project is the ability to accurately determine which items are in the backpack. With a gating algorithm based on a single distance to a single Bluetooth sensor (the Raspberry Pi Zero), the shape of the included volume is approximately a sphere centered on the sensor. We believe that, by placing the Raspberry Pi at the center of the backpack, the spherical shape will be close enough to the backpack's shape to determine which items are in the backpack. However, if the single-distance-based gating algorithm is unable to exclude items that are near the backpack but not inside the backpack, our risk mitigation plan is to utilize a trilateration approach using a total of 4 Bluetooth sensors spaced around the backpack to obtain a 3D location for the item. The trilateration approach and algorithm is discussed in Pu, Pu, and Lee's "Indoor Location Tracking using Received Signal Strength Indicator" paper.[6] We will use the trilateration approach as opposed to the triangulation approach, as it is easier for our system to determine the distances between each of the 4 sensors than to determine the angles between them.

Another risk element was the issue of using Android Studio to build our phone application. After discussing amongst ourselves as well as with our TA, we decided to switch to building a mobile-accessible web application after the design review presentation for multiple reasons. First, the group collaboration for the development process would be much more streamlined compared to development in Android Studio, as Visual Studio Code (which we would user for web application development) has better Git integration than Android Studio. Second, a web application offers greater accessibility to different platforms such as iOS than an app designed only for Android. Additionally, 2 out of 3 of our team members already have prior experience with web

application development, as opposed to Kotlin which none of us have experience with. We also did extensive research and found the Web Bluetooth API, which will allow us to communicate with Bluetooth devices over JavaScript, so our system would still maintain the same Bluetooth functionality. Finally, should we find that we need a component of native functionality which is missing from this web application, we can use the WebView[2] functionality to build any necessary native features in Android Studio port them deliver to our web application.

7 RELATED WORK

There are currently several other existing asset-tracking products which help users keep track of their items. One popular existing technology in this space is Tile, a small Bluetooth-based tracking tags that allows the user to locate the missing items so long as they are attached to a tag. Backpack Buddy and Tile share some common features. For instance, both require physical tags to be attached to the items to track the items, and both use Bluetooth Low Energy (BLE) technology to communicate with the tags. Both products also utilize the smartphone app to interface with the user. However, while Backpack Buddy and Tile share some similarities, Backpack Buddy differs from Tile in two major ways. First, Backpack Buddy allows students to manage collections of items in relation to their schedule, whereas Tile only tracks and locates individual items. Second, Backpack Buddy has reduced user burden with regards to item registration. Backpack Buddy uses computer vision to automatically determine item names, saving the user effort when naming items. Backpack Buddy also learns the user's schedule, such as what items a student might carry in the backpack when and where, saving the user from needing to manually assign items to events.

One CMU ECE Capstone project from the Spring 2020 semester called Sous-Chef also shares similarities to Backpack Buddy. Sous-Chef is a smart pantry storage unit that keeps a list of groceries currently inside a pantry. Sous-Chef is similar to Backpack Buddy in that it registers a food item to a database using computer vision and image processing of barcodes on food products. Sous-Chef also provides a streamlined user experience by displaying the list of food items on a web application. However, Sous-Chef has a different use case than Backpack Buddy, as it is intended for tracking of food items in a pantry, whereas Backpack Buddy is designed for tracking schoolwork- or extracurricular-activity-related items in a backpack.

8 SUMMARY

With their busy schedules and plethora of items to track, students often lose or forget important items for their events. A smart inventory system is therefore extremely valuable for managing tracking items, saving students from the burden of having to remember items and the anguish from forgetting them. Backpack Buddy not only informs students what items are inside their backpack, but also suggests what items to bring according to the students' schedules and notifies them of their missing items before each event. We intend improve the lives and mental well-being of students with our easy to use inventory system.

So far, we have learned a lot about teamwork and communication through working on Backpack Buddy. We know each of our teammate's strengths and weaknesses, and have divided our tasks and planned our schedule accordingly. Because we have been following our planned schedule, we are confident that Backpack Buddy will be completed on schedule with full functionality.

References

[1] *Bluetooth Proximity Light*. https://github.com/nullhart/bluetooth-proximity. Mar. 2016.

[2] *Building web apps in WebView*. https://developer.android.com/guide/webapps/webview. Mar. 2021.

[3] *Getting the Sleepy Pi to shutdown the Raspberry Pi*. https://spellfoundry.com/docs/getting-the-sleepy-pi-to-shutdown-the-raspberry-pi/. May 2020.

[4] *H1 Beacon Datasheet*. http://doc.mokotechnology.com/index.php?s=/page/28. Dec. 2020.

[5] *IDT - Image Dataset Tool*. https://pypi.org/project/idt/.

[6] Chuan Chin Pu, Chuan-Hsian Pu, and Hoon-Jae Lee. "Indoor Location Tracking Using Received Signal Strength Indicator". In: *Emerging Communications for Wireless Sensor Networks* (Feb. 2011).

[7] *Python interface to Bluetooth LE on Linux*. https://github.com/IanHarvey/bluepy. Dec. 2020.

[8] *RFID vs BLE: How Are They Different in Terms of Asset Tracking?* https://www.assetinfinity.com/blog/rfid-vs-ble-how-are-they-different-in-terms-of-asset-tracking. Mar. 2021.

[9] Farhana Sultana, Abu Sufian, and Paramartha Dutta. "Advancements in Image Classifications using Convolutional Neural Network". In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)* (May 2019).

[10] *Web Bluetooth API Documentation*. https://webbluetoothcg.github.io/web-bluetooth/#introduction. July 2020.

[11] David G. Young. *How Far Can You Go?* http://www.davidgyoungtech.com/2020/05/15/how-far-can-you-go. May 2020.

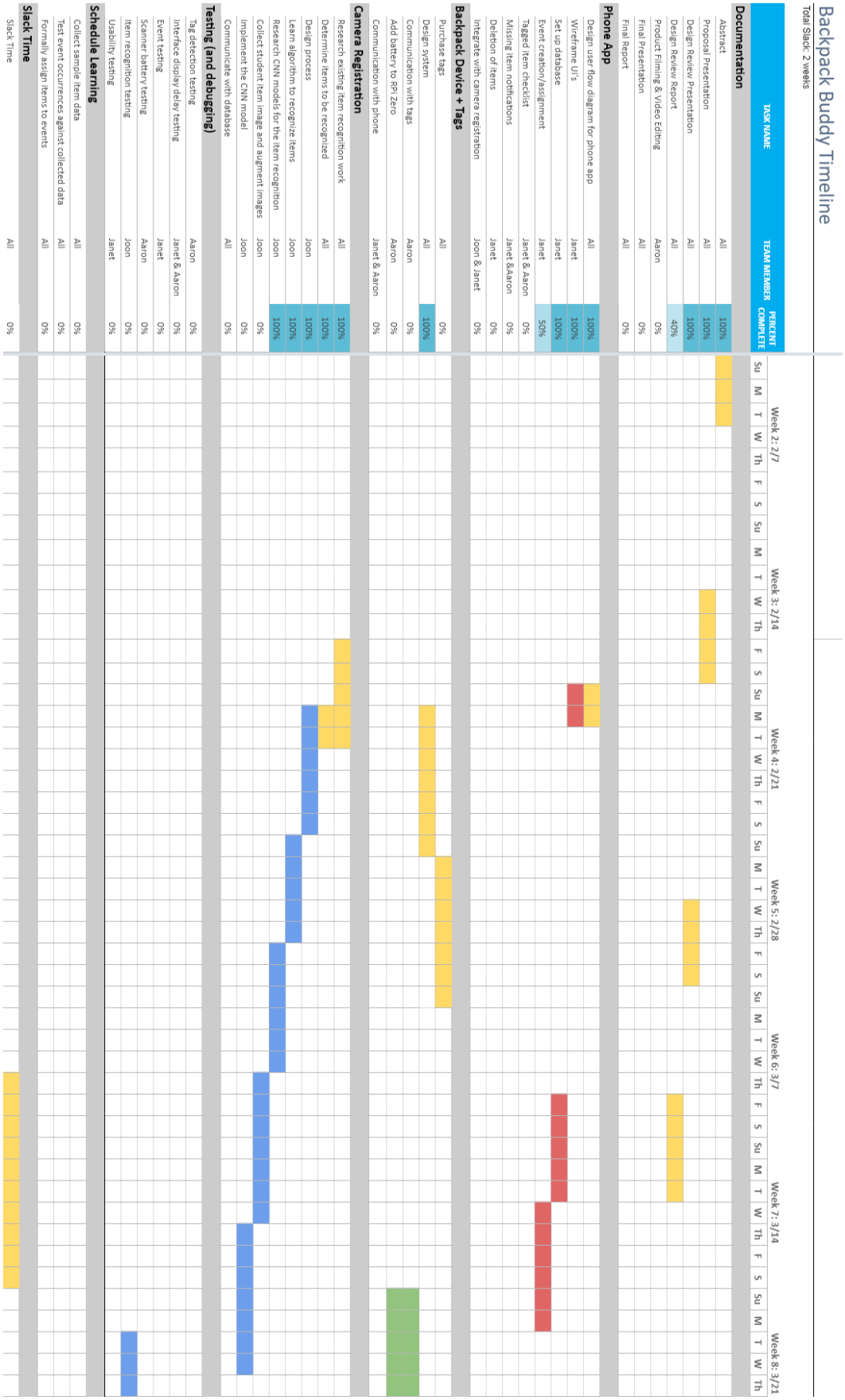

Figure 7: Gantt Chart First Half

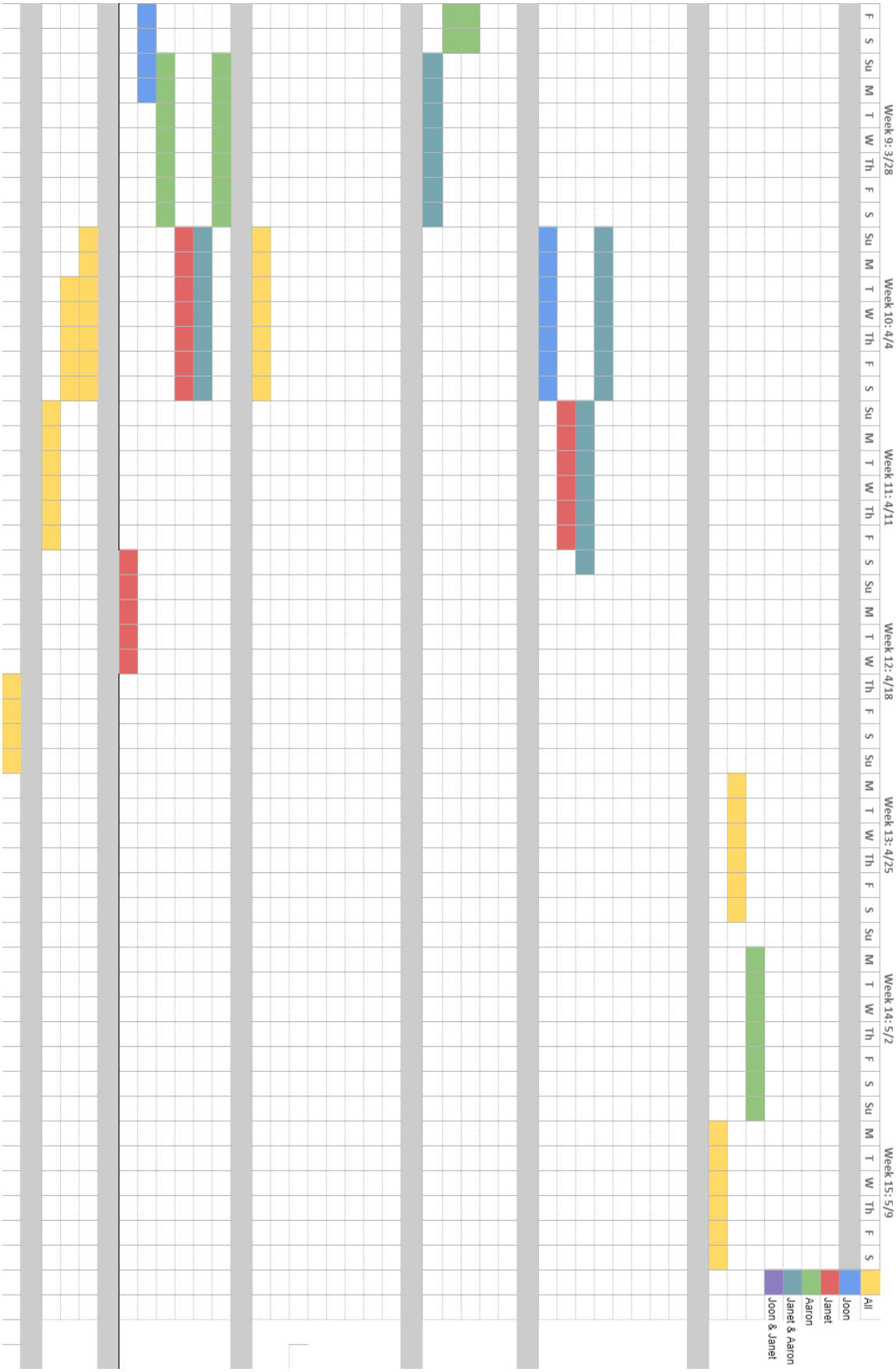

Figure 8: Gantt Chart Second Half