# Team HomeAide

---

# Final Report

## Version 1.0

## April 27, 2021

---

**Project Sponsors:** Kelly Roberts, PH. D and Jill Pleasant, MA

**Team Faculty Mentor:** Fabio Santos

**Team members:** Seth Borkovec, Ethan Donnelly, Courtney Richmond, Noah Baxter

# Table of Contents


**Introduction** **2**

**Process Overview** **4**

**Requirements** **5**

Functional Requirements 5

Non-Functional Requirements 7

Environmental Requirements 9

**Architecture and Implementation** **10**

Architecture Overview 10

Implementation 13

Mobile Application 13

Website 14

Database 15

Apache2 17

Azure Cloud Server and Virtual Machine 17

As-Planned Versus As-Built 18

**Testing** **21**

**Project Timeline** **25**

**Future Work** **29**

**Conclusion** **32**

**Glossary** **33**

**Appendix A: Development Environment and Toolchain** **34**

Microsoft Windows 35

Ubuntu Linux 39

# I. Introduction

In the United States alone, the number of individuals who are age 60 and over is more than 65,000,000 people and of those - nearly 23,000,000 have at least one identified disability they are living with. Taking this into consideration, there are thousands of Assistive Technology (AT) products that can enable persons with disabilities and those experiencing aging-related limitations to be more productive and self-sufficient in daily activities. These items range from simple to complex, inexpensive to costly and commercially available to customized. Devices can be used at home, work, school, or in the community to reduce barriers, enhance participation and increase or regain independence.

Assistive technology devices help people to **compensate** for lost function. For example, a flashing light doorbell alert can be used by a deaf person to let them know that someone is at their door, or assistive technology can be used to **enhance** and support a declining ability – installing grab bars in the bathroom so that a person with poor balance can use the toilet without falling. High tech wheelchairs with sophisticated controls allow people with paralysis to be independently mobile and even drive vehicles. Small speech generating tablet devices produce speech for people unable to talk as a result of brain injuries, strokes, cerebral palsy and autism.

Market-wise the Assistive Technology market is expected to reach $26 billion by 2024 which is almost doubled in size compared to $14 billion in 2015. There is a great need for these devices in people's homes as well as a consumer need for learning what's available and useful to the person who needs it.

Our sponsor is Northern Arizona University's Institute for Human Development (IHD). IHD is one of NAU's oldest Institutes and focuses its research, teaching, training/technical assistance initiatives, community service and dissemination efforts on issues that affect persons with disabilities across the age span. As a federally designated University Center for Excellence in Developmental Disabilities, emphasis is placed on advancing positive attitudes, universal access and full inclusion in all aspects of life for people with intellectual and developmental disabilities.

Leading the IHD team of more than 45 staff members is IHD Executive Director Kelly Roberts Ph.D., and Jill Pleasant, MA, OTR/L Associate Director. Dr. Roberts is a researcher, tenured professor and has many years' experience with assistive technology. Ms. Pleasant has an extensive occupational therapy background specializing in assistive technology. IHD is also the home to the Phoenix based Arizona Technology Access Program (AzTAP), Arizona's designated Assistive Technology Act Program. Their staff members are also contributing AT content area expertise to this project centering on matching product knowledge to consumer needs.

A quick Google search for “assistive technology” will yield pages of results, but deciding on a specific product that matches a particular person’s needs related to their type of disability or limitation, what they want to accomplish, where they will use it and their budget is complicated. Individuals with disabilities, older adults and family members are often unfamiliar with the resources available to obtain assistive technology information and support.

With this information, it has led our team to work with our sponsors to create a solution that involves us building a cross-platform mobile application which will help these individuals find the Assistive Technology that will best suit their needs. This mobile app pairs individuals with devices tailored to their specific difficulties and is able to offer them tips and resources so that they can be best prepared to live their life to the fullest extent.

Following this idea, some of the key user level requirements that we had in this application are a method for determining the best fit assistive technology, accessibility options, and the ability to provide advice to the user. With this in mind we also had to consider the functional requirements that describe what our system can do which include having the user be able to provide their general limitations, allowing them to identify their trouble areas in the house/room, having the user create a user profile, and having the app allow the user to save their favorite products they have viewed. Then our performance requirements which will help our system achieve those functional requirements will be to have the app sync to the database regularly, have the app accommodate color blindness and screen readers, and to have the app be able to perform a query or display a timeout in less than 10 seconds. One more thing to consider is the environmental constraints that we may face which mainly includes being HIPAA compliant for user privacy.

With all of this in mind the rest of this document will serve as a guide as to how this software will be implemented and the specifics to the architecture of the application. There are many things to consider when designing software and this will provide the reader a better understanding as to our plan going forward.

# II. Process Overview

Over the course of the project our team mainly used the agile development process with some influence from the waterfall process. Overall our process for design was to make small incremental changes that would then be presented to the client. This allowed them to provide us with useful feedback that we could use to target specific parts of the project that the client wanted to be updated or changed. We continued this process throughout the project showing our client our progress in each of our biweekly meetings.

We divided the project up into smaller modules that broke up our main tasks into pieces that were exclusively for the website, exclusively for the mobile application, and ones that overlapped and covered both. By doing this we were able to better target specific parts of the website, or mobile application that needed to be developed next in order to provide our client with a final product.

To help with the development of this project our team used several different apps and supporting tools. The team used GitHub for our version control as well as our issue tracking later on in the semester. Trello was used as our Kanban board where we kept an updated list of all assignments, deliverables and coding tasks that were coming up. We used Microsoft Azure for our cloud server used to host the website, Django for the backend of the website, SQLite3 for the database, and Flutter for the mobile application. The team also used draw.io for making UML diagrams, and erdplus.com for database modeling. And finally the team used Discord as our main means of communication.

The team consisted of 4 team members. Seth Borkovec was our team leader, client communicator, and served as the lead on the website portion of the project. Ethan Donnelly was our team recorder, release manager, and served as the lead for the mobile application portion of the project. Courtney Richmond was the team editor, product research manager, and also maintained the team's website. Noah Baxter was the team architect. Overall the team roles were only loosely followed and many team members would help out where it was needed.

The team held weekly meetings on Fridays that lasted for about two hours in which the team would discuss the upcoming agenda and tasks would be assigned to each team member. The team also implemented a protocol for rescheduling a meeting. The team agreed to have a backup meeting time set aside on Sunday should the team be notified in advance that the previously agreed upon Friday time would not work for one of more of the team members.

# III. Requirements

The acquisition process of our requirements was very straightforward. We were able to gather what needed to be done for the project by having weekly meetings with our clients throughout the fall semester. Research into the project itself and the technologies we could use to develop this project also helped us further refine these requirements into what is and isn't feasible with the technology. Lastly, the development of documents mostly through the fall semester also helped us to refine these requirements.

In this section, we will outline all of the project requirements for the project. We will begin by reviewing the functional requirements which detail what the system can do. Next, we will review the non-functional requirements, or the performance requirements, that specify how the functional requirements are conducted. Lastly, we will go over the environmental requirements which include constraints from external sources or laws.

## A. Functional Requirements

### 1. Mobile App

a. The user needs to be able to provide their general limitations.

   A limitation refers to any physical difficulty that may hinder a person's ability to perform a specific task. The mobile application must be able to gather this information from a user.

b. The user needs to be able to identify their trouble areas in their home.

   By being able to navigate through a virtual home, a user will be able to identify a specific room in their home that gives them the most trouble.

c. The user needs to be able to change rooms in a virtual house.

d. The app needs to remember user limitations.

   After gathering this type of information from the user, the application must be able to remember this for future reference, along with any identified trouble areas in their home.

e. The user needs to be able to create a user profile.

   A user profile is very standard in mobile application development. This profile is what will be able to link the user to their limitations so that they can be stored for future reference.

f. The user needs to be able to log in to the app.

g. The user will be able to recommend new AT devices.

   In the event that a list of AT devices are being recommended to a user and the user doesn't see a specific device, they are able to send a recommendation to the clients.

h. The app will have an option to give the user a fresh start and create a new profile.

i. The app needs to give recommended AT based on the difficulty of a task in a particular area of the home.

   The app must take a user's physical limitations and identify trouble areas in their home and generate a list of ideal AT devices suitable for their needs.

j. The app needs to allow the user to save their favorite products they have viewed.

k. The app needs to refer the user to local services contact information.

   This could not only include local disability resources, but also resources that help connect users with AT devices. This allows users another way of identifying which AT devices are best for them.

l. The app will display a disclaimer that the recommendations are only suggestions and that the benefit of recommendations is not guaranteed.

   It is critical that the mobile app does inform users that any recommendations made to the user are merely suggestions and user satisfaction with these devices are not guaranteed.

m. The app asks users if they would like to share their AT recommendations/profile with their local resources via email or printer.

### 2. Website

a. The clients need to be able to manage the assistive technology inventory in the database.

   Database management includes a variety of tasks such as: adding, removing, searching, or modifying the AT devices in the database. This also includes the ability to add on additional categories to the AT devices that will provide more information about them.

b. The client needs to be able to send a Qualtrics survey to the users.

   In essence, the client will be able to send a link to a Qualtrics survey to users, in which viewing that link will navigate them to a survey being hosted on the Qualtrics framework.

c. The client needs to be able to view the results of a Qualtrics survey.

   If users submit a survey, the clients will need to be able to review these results in Qualtrics.

d. The client needs to be able to create administrative accounts on the website used to manage the database.

e. The client needs to be able to remove administrative accounts on the website used to manage the database.

## B. Non-Functional Requirements

### 1. Mobile App

a. App needs to sync to the database regularly

   To ensure that a user is getting the best information, the mobile app needs to regularly sync with the database.

b. The app needs to accommodate color blindness.

   Users will be able to select from a variety of color themes that provide significant color contrast.

c. The app needs to accommodate screen readers and text-to-speech

   The mobile app will be able to integrate the text-to-speech functionality that exists on the users device (if they are using a device that supports text-to-speech).

d. The app needs to accommodate voice input

   Similar to the requirement above, the app will be able to integrate with the operating system on the users device to allow for voice input.

e. The app will autosave the session in case the user leaves the app so that they can pick up where they left off

f. The app needs to be able to perform a query or display a timeout in less than 10 seconds.

## 2. Website

a. The administrative website should require user authentication

   In order to protect access to the website, administrative users will use a password to log in.

## 3. Database

a. The database needs to be secure from unauthorized access

   Access to the database will require that a user is authenticated using Django's built in authentication system.

## C. Environmental Requirements

### 1. HIPAA Compliance

Given that the nature of this project can associate a user with a physical limitation that can be documented in health forms, the nature of this project must be HIPAA compliant in that private medical information about a user is kept protected. Due to this, the application will not be asking for nor recording private user information such as their name or location. The application will not be taking in any medical diagnoses, but merely offers a framework in which users can select from a predefined list of limitations that they are experiencing. The application will also be taking in a username. This user name can be whatever the user wants it to be and will be associated with that user's active user profile. To ensure that this information is protected and is HIPAA compliant, the username will be encrypted before it is sent to the database for permanent storage.

# IV. Architecture and Implementation

In this section, we describe the architecture of the system to give you an understanding of what components are involved, how they interact, and their purpose. We'll start by talking about the overview, then we'll go into the details for each component, and finally we'll finish up with differences from the project as-planned. Please refer to our Software Design document for even greater details about the architecture. In the section following this one, we will talk about the testing activities.

## A. Architecture Overview

Our system involves two main components: a mobile application and a web server. The main role of the mobile application is to provide AT recommendations to end users. To do this, the mobile app relies on the web server to manage an inventory of AT. The web server consists of a SQLite3 database, a Django website, and an Apache2 HTTP server on an Ubuntu operating system hosted on a virtual machine in the Microsoft Azure Cloud. The end users only interact with the mobile app. Our clients act as administrators for the inventory using the website. Please refer to the diagram below (Figure 1: Architectural overview) for a visual of the system.

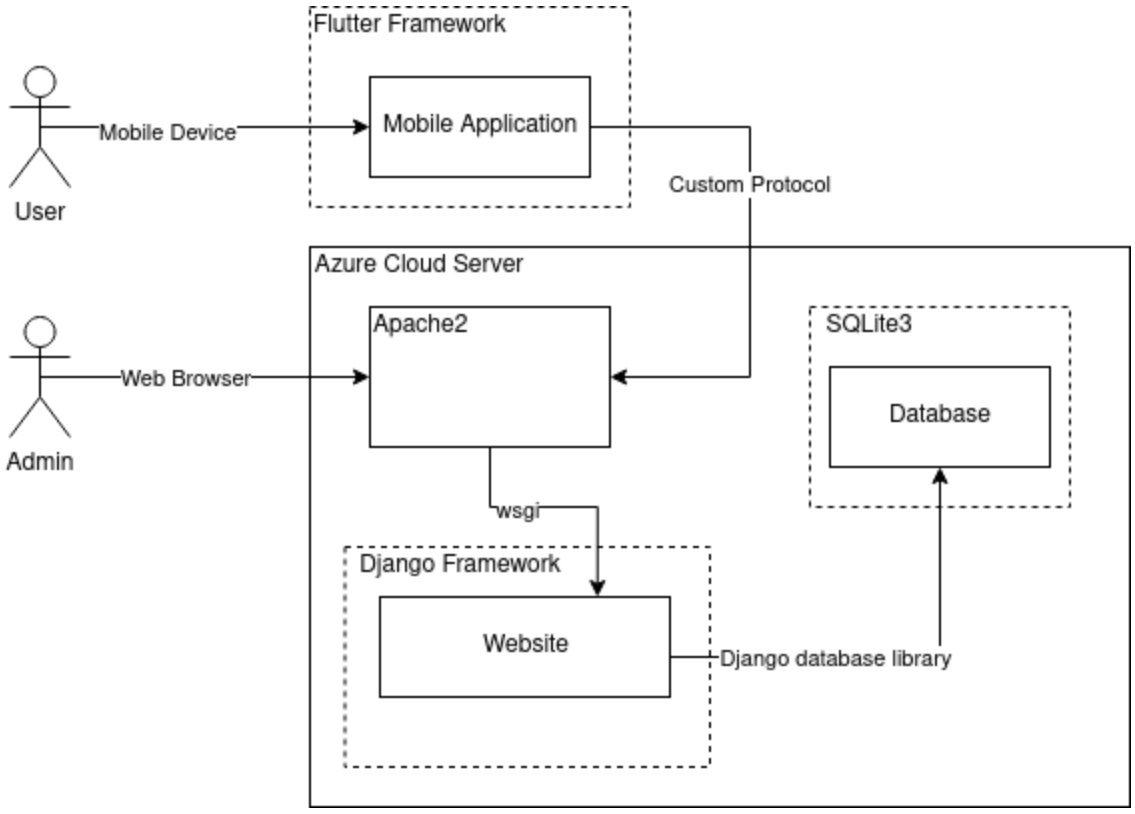


Figure 1: Architectural overview.

The main role of the mobile application is to provide AT device recommendations to the end user. The end user only interacts with the app and not with the website. The app allows the user to create a profile to save information, but also has app settings that the user can change to their preferences. The user can navigate through a virtual representation of a home with different rooms and receive recommendations for AT devices that will help them in those areas. To do this, the app needs to get information from the database, but it doesn't connect to the database directly. Instead, it uses a custom API we designed so that the request is forwarded to the Django framework.

The website provides a front-end for the administrators to manage the inventory of AT devices in the database. Only the administrators interact with the website, and they can add, remove, and edit AT devices in the inventory; as well as sending messages or surveys to the end users via the database. The website is built in the Django framework which has built-in libraries to interact with the database.

The database only interacts with the website via Django. The database inventories the AT devices, messages, surveys, user profiles, and administrator profiles. No SQL statements will be used, and instead everything is managed using the website.

The Cloud server provides a home for both the website and the database. We are using Ubuntu as the operating system for the server. The software used to serve the website is Apache2 which serves the Django-based website using the module WSGI. Apache2 does not handle any requests itself and instead forwards all requests to the Django-based website.

The communication between the mobile application and the website uses a custom protocol we developed for secure communication which relies on a unique ID and key pair. Initially, the app has no ID or key. When the app is first run, it sends a predefined default ID and key pair to the server to initiate a handshake. The server then assigns the unique ID and key pair and returns these to the app. The app sends this new pair back to the server for verification that they are correct and saves it to the app's permanent storage after the server acknowledges it. Every communication between the app and server then requires the unique ID and key pair for authentication.

The overall architecture is a server-client relationship where there is one server and many clients. The custom protocol we developed between the app and server is based on the TCP 3-way handshake protocol for computer networks.

To provide a walk-through of the process, let's use an example. Suppose an end user on the mobile application has selected a room in the house to get recommendations for. The app will compile the user's declared physical limitations as set in their user profile on the app and send that along with the location in the house, and the handshake ID and key pair, to the website. The website first authenticates the request by verifying that the ID and key pair are correct. If they happen to be incorrect, the website will respond with

a JSON object that has a "status" of "error." Otherwise, the website will use the provided limitations (if any) along with the location in the house to create a list of AT that match. This list has two tiers: one tier of AT that match only on location in the house, and another tier that matches on both location and on limitations. The website compiles this list as a JSON object and returns it as a JSON response to the mobile app. The mobile app parses the JSON object with the two-tiered list of AT, and presents them to the end user as recommendations.

Next, we'll go into more details about these components.

## B. Implementation

Here we provide a detailed design description for each module mentioned above. We start with the mobile app, the website (which involves Azure, Apache2, and Django), and then the database.

### 1. Mobile Application

The mobile application is designed in the Flutter framework to allow for cross-platform builds for both iOS and Android.

#### a. Responsibilities:

(1) Providing an interface to the end user

Provides a visual interface as a mobile application for the end user. The user can interact using touch and/or with accessibility technologies.

(2) Giving AT recommendations

Gives recommended AT devices to the end user to help with a problem area or difficulty based on the room or object and the user's specified general difficulties.

(3) Navigating a virtual house with rooms

The user can touch/click on rooms in a virtual representation of a house to view options for AT devices and get recommendations for that room.

(4) Allows user to suggest new AT devices

If the user has used an AT device that may be helpful for others but doesn't appear to be in the inventory, the user can suggest to the admins that it be added.

(5) Creating a user profile

The user creates a user profile that stores their general difficulties, app preferences, favorite AT devices, and past recommendations. The user can also choose to delete their profile and create a new one.

(6) Provides information for local resources

Information about local organizations who can assist users with professional advice about AT is stored in the website. The app needs to provide this information to the end user.

(7) Interacts with the website when getting AT recommendations

Using input provided by the user when navigating rooms and their general difficulties, the app will query the database using the admin website. The results will be displayed to the user as recommendations.

## 2. Website

The website is developed through Django that provides front-end access to the database for administrators, and serves database information to the mobile app.

### a) Responsibilities

(1) Provides administrative access to the database

Acts as a front-end website that will allow administrators to access information stored in the database.

(2) Allows administrators to add, remove, or edit information stored in the database

The website provides administrators the ability to add, remove, or edit information about the AT devices that are stored in the database. It also provides administrators with the ability to add additional information to AT devices in the method of adding or removing categories attached to them.

(3) Allows administrators to send messages and surveys to mobile app users

Surveys can be created in sites that use a URL for the survey such as Qualtrics. Messages and these survey URLs can be sent to the users of the mobile phone. The Website will keep a history of sent messages and surveys to view the results of the surveys.

(4) Defines the layout of the house

The house is organized by rooms and each room is organized by room areas/objects. The admin can manage the organization of these in the website by defining new entities and changing their associations. This is directly reflected in the mobile app and how end users navigate the house.

(5) Defines physical limitations that app users may have

To provide better recommendations, the admins can define limitations or physical difficulties that the app user may be experiencing. When defined here, they will be available to the mobile app user to select from.

(6) Allows the admins to view app user suggestions

The mobile app users can create suggestions for AT devices to be included in recommendations that get sent back to the website. These suggestions are stored in the database and allow the admins to review them.

(7) Provides a way to customize the wording of features in the app

The mobile app has a terms of service, financial assistance information, and an auto-generated email for the user to share their recommendations. The wording for each of these may need to be changed without changing the source code.

(8) Administrator management

The website gives administrators the ability to create, delete, or disable administrative accounts.

## 3. Database

The database is created in SQLite3 and is accessible to the mobile app only through the website.

a) Responsibilities

(1) AT Inventory

Stores information about each AT device in the inventory. An AT device can be associated with a room, room area/object, and/or with a limitation. These are provided in the app as recommendations.

(2) Advice and Tips

Similar to AT Inventory but these are not for devices. These can be associated with a room or room area/object only. These are provided in the app alongside the AT recommendations.

(3) Rooms

Allows admins to manage the custom organization of a home for all mobile app users. This affects how app users navigate the house to get their recommendations.

(4) Room Areas/Objects

These provide a way for administrators to divide a room into smaller areas or objects inside the room. It helps the app user to get more specific recommendations. A room area/object can be associated with multiple rooms such as how there can be a sink in the kitchen and a sink in the bathroom.

(5) Limitations

These are the general physical difficulties that the app users may be experiencing. An app users selects from these in their profile on the mobile app to get better recommendations.

(6) Messages and Surveys

These are created in the website and sent to all mobile app users. The difference between a message and a survey is that a message does not include a URL link. If a message or survey is deleted on the website, it is no longer visible to the mobile app users either. However, mobile app users can hide messages and surveys independently.

(7) App User Suggestions

These are suggestions for AT devices to be included in the AT inventory. Only app users can create a suggestion.

(8) State Resources

Information about organizations and other local resources for each state are stored here. These are viewed by the app users when they want to find their local resources.

(9) Other Configuration

This stores other miscellaneous information such as the wording for the app terms of service, grant information, the wording of the email and web page for sharing recommendations, extra custom categories for the AT, and defining the relative prices for AT.

## 4. Apache2

This is the HTTP server that directly responds to HTTP requests from both the mobile app and administrators who are using the website. This uses a module named WSGI to serve the web files that are created in Django.

### a) Responsibilities

(1) Serve HTTP requests

Forwards HTTP requests from HTTP clients to the Django website, and forwards HTTP responses from the Django website to the HTTP clients.

(2) Log files

Maintains log files for HTTP accesses and errors.

## 5. Azure Cloud Server and Virtual Machine

Our virtual machine is hosted in Microsoft's Azure Cloud Server. The virtual machine uses Ubuntu Linux as the operating system on which Apache2, the Django website, and the SQLite3 database are located.

## C. As-Planned Versus As-Built

When we started planning this project, we were unaware of our capabilities, and we planned an ambitious timeline. Some features were not included in the as-built version because we did not have enough time to implement them or were too complicated to be solved within the project time frame. On the other hand, some features were built but were not originally planned. In this case, these features were easy enough to be included and important enough to our clients to replace another feature in our development cycle. We'll start by discussing the features that were originally planned but not built, and then we'll finish with features that were built but not part of the original plans.

### 1. Features Planned But Not Built

a) Mobile app user can change the color theme in the app

The issue with this feature is that the app is given the theme when the app first loads, and this theme cannot be changed dynamically very easily. There were third-party packages to help with this but we did not have enough time in the development to understand and implement them.

b) Mobile app user can change their avatar

As this was a low-priority feature, we did not implement it because we did not have enough time. It was considered to be low priority because it does not have any influence on the functionality of the app.

c) Mobile app user can rate their AT recommendations

As with the other features left out, we ran out of time in the development cycle to implement this. It was also considered less important than the other features that were planned.

### 2. Features Built But Not Planned

a) The mobile app gives advice and tips along with AT recommendations

Being able to help end users by providing tips and advice around the home was important to our clients. It was also an easy feature to implement because the functionality is mostly the same as the regular AT inventory. The one thing it did change was that the recommendations in the app became a screen where the user could switch between the AT devices and the advice and tips.

b) The extra categories for the AT devices can be shown in the app

   Having another category for an AT device isn't very useful unless it can be shared with the mobile app user. To add this feature, we had to change how the recommendations were compiled in the website and how the mobile app parsed the recommendations JSON. The JSON with the recommendations also includes another JSON for these dynamically-created custom categories as well.

c) Mobile app users can hide messages and surveys

   As an app user, it is important to be able to mark messages and surveys as viewed, and also to dismiss them. This required some interaction to be built for the messages in the app. If the app user taps on a message, it is marked as read. The app user can swipe the message off the screen to permanently hide it.

d) Mobile app users are shown a "terms of service" agreement which can be customized by the admins

   We did have plans to include some kind of user agreement for the mobile app, but it was decided later that our clients would need to customize this and update the wording without changing the source code. This was built into the database and the mobile app gets the terms of service from the website when it loads the app.

e) Mobile app users are shown extra information in the resources tab

   Simply providing a list of local state organizations that can assist the users with professional advice about AT is supplemented by some additional information that our clients can customize. This allows our clients to include anything from how to get help with financing AT purchases to general advice.

f) Admins can change how relative prices are categorized

   The relative prices of AT was decided later on to become a style often used for restaurants where more "$" characters represent more expensive restaurants. Our clients wanted full control over how many levels exist as well as wording to be associated with each level.

g) Admins can customize the wording for the email created when sharing recommendations in the app

   The wording in the email when sharing AT recommendations with someone else needs to be customized without changing the source code. Our clients wanted to be able to have this ability, so we included it in the website. The app gets this information when it loads.

Now that we've covered the architecture of the system, along with some detail about the implementation, the next section will discuss testing of the system and its components. Again, for more details about the architecture of this project, please refer to our Software Design document.

# V. Testing

The team created an extensive in-depth testing plan that covered unit testing, integration testing, as well as usability testing. For our unit testing we created driver functions written in the same language. These driver functions test the boundaries for inputs, and validate outputs. We used a "black box" approach where we knew the inputs and expected outputs, but we are not concerned with the inner workings of the functions. Our tests will also check for proper handling of errors by purposefully giving inputs outside of the expected range when appropriate. For the website on the Django framework, these driver functions will be written in Python. For the mobile application which is in the Flutter framework, these driver functions will be written in Dart. we then split up the functions in our code into modules covering categories of responsibilities that each set of functions had. We then broke this list down farther into functions for the Web application, functions for the mobile application, and global functions. For each module we provided a description of what this group of functions does, the boundary values, as well as some example values, and finally outlined what the expected behavior was. If you would like a more detailed look into exactly what each of our modules consisted of please refer to our more in depth Software Testing document.

To determine what to test for integration testing, we needed to consider what major components interact with other components. In our project, these include the database, the website, and the mobile application. The database and the mobile application only interact with the website and not directly with each other. Since the interactions between the website and the mobile application occur over a network, we needed to test for situations involving network connectivity and security. The interaction between the website and the database needed to be tested for data correctness and security. Security can be tested by trying to modify the database or sending an HTTP request without correct authentication. Data correctness in the database is tested by making queries in the website and looking at the records to make sure they are correct. Connectivity over the network can be tested by simulating network problems.

## A. Website and Database Integration

The website and the database communicate using Django and its built-in database libraries. The website has full control over the data manipulation and organization, therefore it is important to test that the website is modifying the database correctly.

To test the integration of these two components, we can utilize the website functions by sending test data. These functions should manipulate the database in an expected manner and we can verify correctness using the SQLite command-line tools to view the data. We can also input data directly into the database using these same tools and verify that the data is being extracted correctly by the website by looking at the data returned from the website functions. Correctness can be directly observed if the data input into the database or returned from the website functions match what is expected.

Since the website is the only interface to the database, there can be no other point of access. We will also know about any errors or problems with the database integration with the website because the Django server will not start if it detects a problem in the logic of the functions. There are rare cases in which functions could provide bad data or faulty logic when interacting with the database in which case we will implement try-catch clauses to prevent a crash.

## B. Mobile Application and Website Integration

As mentioned above, the website is the only point of interaction with the database, so the mobile application needs to interact with the website. This interaction occurs over a network which provides for some challenges with connectivity. Since the device the mobile application is running on may not have network connectivity guaranteed, the integration between these two needs to account for disconnects and timeouts in addition to the normal data validation.

To test the behavior of each the mobile application and the website in conditions of poor or no network connectivity, we can simulate the conditions by forcibly disabling responses from one or the other. In either case, there should not be places in the app or in the website where it hangs while waiting for the response that will never come. Instead, after a timeout, it should gracefully back down to what it was doing with a message to the user on the app explaining that there are problems with network connectivity and to try again later.

To test the normal behavior when network connectivity is not a factor, we use the functions in both the mobile application and in the website that deal with these interactions. We can use debugging logs to verify that the interactions were successful and that the data is correct.

And finally we created a plan for conducting usability testing with both our clients and end users to get a better understanding of how both the mobile application and web server would be used. In order to test our user interface with the mobile app users we will give a set of tasks to our group of test users and see how easily they are able to perform each of the taks. The test users will not be given any guidance on how to complete these tasks and we will take note of which tasks take the most time as well as if there are some tasks that could not be completed by some of the testers. We will also look for tasks in which the users struggled to find the solution and what sections of the application they went through while trying to achieve the desired results. The tasks the users will be asked  to complete are as follows:

- Create a new account
- Set a general limitation for your profile
- Save most relevant AT recommendation from a room of your choosing
- Share your AT device recommendations using e-mail
- Suggest a New AT device to be added
- Find your local services contact information

When looking into the results of the tests we conduct we will be able to gain insights into important areas where the app could use improvements in order to make it more user friendly as well as what parts are working well. By testing across different age groups we will be able to determine if our user interface becomes harder to navigate within different age groups. We will also be able to what tasks were the hardest for users to complete indicating areas that could be improved. And finally by understanding what mistakes users made when trying to complete the above list of tasks we will be able to get a better understanding of where users naturally assume different places in the app should be giving us insight into exactly how we could modify the user interface have a layout that places things where the user naturally goes to look for them

For the website the plan we layed out was to start off by measuring the amount of time it takes for the admins to add new rooms, room objects, and limitations to the pool of available options to select from. All of these attributes are used in the AT devices table and need to be entered before the database can be populated with AT devices. We will observe how long it takes for each of the website admins to make several entries into each of these categories and keep track of not only how long the average entry takes but also how long it took for the admins to develop a repeatable flow when it comes to entering data into each of the above categories. If too much time is spent attempting to accomplish these tasks, it will indicate that changes need to be made in order to either make the process quicker, or to make it easier to learn.

Next using the data that the admins entered we will have them add several new AT devices to the table and time how long the average entry takes. While the average time is important we will also keep track of how long it takes for the admins to find the correct option for the rooms, room objects, and limitations field. This is important because as more entries are placed into each field it is possible that the amount of time it takes to find the desired option will become significantly greater. To properly test this once we have the initial time it took we will have the admins once again add several new AT devices to the table, this time with significantly increased options to choose from in the aforementioned categories. Once we have both times for the initial and follow up tests we will be able to compare them to determine if the amount of time it takes to input new AT devices is significantly impacted by the amount of data in those categories. Again, if too much time is spent completing these tasks, that will indicate that the manner in which the data is displayed for selection needs to be reconsidered to make it have less of an impact on overall time.

Finally as new website admins will likely be added it is also important that the website is easy enough to understand that the current admins are able to teach new ones how to do all of the above actions. For this test we will have the current website admins explain how to use the website to someone who up to this point has not interacted with it at all. We will be interested in seeing how much time it takes for this new person to understand all the important tasks that need to be done in order to maintain the website as well as if the current admins are able to effectively explain how to handle everything. The purpose of this test is to determine if the overall user interface of the website is intuitive enough that it can not only be effectively explained quickly but also that there are no issues or hang ups that arise should someone new be added onto the administration team.

# VI. Project Timeline

In this section, we will go over the timeline over both semesters. The first semester was mainly about requirements gathering, while the second semester was about the software design and development.

## A. Fall 2020 Semester

There were three main milestones in the plan for the Fall Semester. These included: the Requirements Specification document, the first Design Review presentation, and a tech demo. Each of these three milestones has two phases: a draft or plan, and then the final version.

- Requirements Specification Document

  This document provides the details for the project, including justifications and a catalog of features to be implemented. It also serves as a contract between our team and the client to outline the expectations from our team to declare a satisfactory finished product. This document was created with the help of our clients and includes their input.

- Design Review 1

  This presentation preludes the final version of the Requirements Specification Document mentioned above. It presents much of the same information in less detail, but as a video with presenters. This video is available on our team website (https://ceias.nau.edu/capstone/projects/CS/2021/HomeAide-F20/documents.html#design-review-1).

- Technological Demonstration

  In order to prove the integration of our chosen technologies that were introduced in the Technological Feasibility Document, we created a demonstration which was performed for both our team mentor and our clients. The demonstration was important because we could not continue with development until we had proven that our chosen technologies work together.

## B. Spring 2021 Semester

Development on the mobile application, website, and database began on January 24th, 2021 and completed on April 27th, 2021. The March deadline was the estimated date in which a working version of the application should be up and running, and each of the modules listed below will be in working order. The remainder of March and April was spent on UI, design, and adding finishing touches to the application.

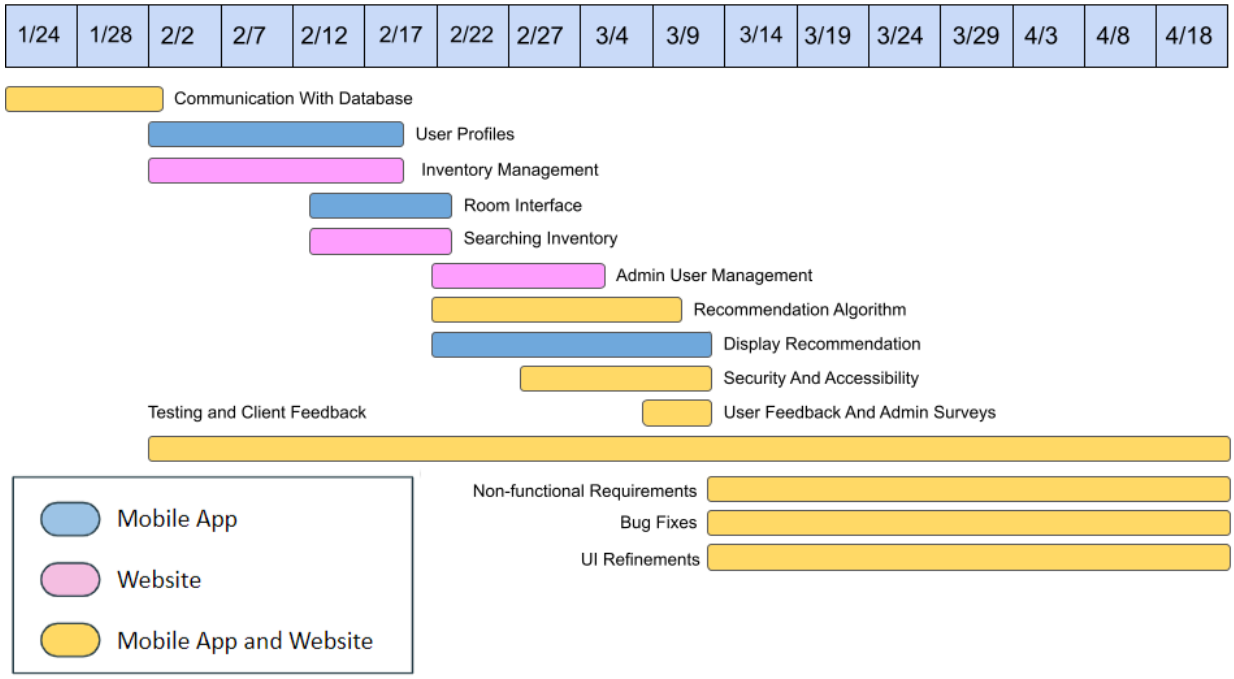


Figure 2: Gantt chart for Spring 2021 semester

### Module Breakdown:

- Communication with database

  The initial focus at the beginning of development was making sure the website and the mobile application can access the database and be able to pull data from it.

- User profiles

  This step focused on allowing users to be able to make profiles. These profiles include their username, passwords, and a list of any limitations they may be experiencing. This data also needs to be able to be stored in the database.

- Inventory management

  Focus here was spent allowing administrators to manage all of the AT devices stored in the database. They are able to add, remove, or edit any devices contained in the database.

- Room interface

  This is the core functionality of the application that allows a user to navigate around a virtual home and select rooms. From here, they can be given any advice or recommendations if the user may be having difficulties in that room.

- Searching inventory

  Additionally functionality is added to the inventory in which the administrators are able to search for information contained within the database. Searching parameters can look by name of the device, the type, or even what limitations they could assist with.

- Recommendation algorithm

  This is the algorithm that considers user limitations and room difficulties to provide any AT devices or general advice to assist the user. It is crucial that the algorithm can access the database and pull information about the AT devices out of the database and display to the user.

- Display recommendation

  This is more of a sub-module of the above in which advice or any recommended AT devices is displayed. This part needed to be adaptive in how it is displayed depending on how much information the user wants to see at a time.

- Admin user management

  Provides the administrators with the ability to be able to manage other administrators. They can determine the level of access each administrator will have to the database.

- Security and accessibility

  Encryption functionality and even administrative activity will be developed to help secure the database.

- User feedback and admin surveys

  The application has integration with Qualtrics which acts as a third party application that provides surveys to users about their experience. These responses are sent back to the clients where they can evaluate the data from these surveys.

- Non-functional Requirements

  This module is reserved mostly for post alpha prototype development. In this case, the non-functional requirements for this project are implemented. This includes database encryption or accessibility options

- Bug Fixes

  As with all software projects, the mobile application and its webpage component contain some bugs and glitches. This can include the webpage displaying “Add” when you’re really editing an item, or the mobile application not displaying that a checkbox has been selected.

- UI Refinements

  In order to get a working alpha prototype, UI was not much of a priority and the initial prototype was developed to just contain wrappers for the working components. Given that the requirements for the project had been met by the alpha prototype, additional focus was placed on UI and making the applications easy and simple to use.

The work was divided among the team members so that Ethan led the development of the mobile application, while Seth led the development of the website. Courtney and Noah helped with the website, and Seth helped Ethan with the backend of the mobile application. We are currently at the end of development, and have delivered the final product to our clients. In the next section, we'll talk about proposals for future development.

# VII. Future Work

Although we have been able to achieve a lot with this project, there is still more room for features that could be implemented in the future, and even room for improvement on the existing framework. Some of the items listed below are a mix of features that were considered at one point but were considered unnecessary or too much for this year's capstone, features that were discussed mid-development but were scrapped for the same reasons, or features that were intended but were discovered later on to not be possible.

- Users can share AT from their favorites list

  A requirement for this project discussed giving a user the ability to share their user profile/recommendations with their local resources. It might also be helpful that a user should be able to share any AT devices that they have favorited with their local resources to narrow down which devices truly are best suited for their needs.

  This was a feature that came up mid development, but we merely did not have the time to implement.

- User can define their own house layout

  The current state of the mobile application gives a house layout that is defined by the administrators via the website. To make things more personal, a user might want to create a virtual home in the mobile application that is similar to the actual structure of their home.

  This is also a feature that came up mid development, but we did not have the time to implement.

- Web application alongside the mobile application

  Not everybody has access to a smartphone and is not able to use the mobile application. More people have access to a working computer than they do a smartphone. A web application would have near identical implementation as the mobile application and would achieve the same goals.

  This was something that was considered at the very beginning of development, but was recognized early on that not only would there not be enough time to develop a working web and mobile application, but that it wasn't necessary at this stage.

- App users can rate recommendations given

  At the moment, all the application does it display recommended AT devices to users. But it was considered actually rather late in development that it could further benefit the recommendation algorithm if a user is able to give a rating on AT devices recommended to them. This rating would not only include rating how appropriate it was for the application to recommend a particular device, but to also display user satisfaction if they had that device in their possession.

  This was discussed rather late in development and there wasn't enough time to implement it.

- Web forum for users

  A lot of websites and mobile applications on the market right now have a web forum attached to them. These are usually just boards in which users can communicate with other users about the application and what certain things mean.

  This was also something that came up rather late in development. If this were to be implemented, it would best be attached to the web application if that gets developed in the future.

- Virtual assistant to help with using the mobile app

  As seen in a lot of other applications, the idea of having a mini AI that would act as a virtual assistant to a user to help them use the mobile application.

  Again, this was discussed late in development but also was considered unnecessary for this stage in the capstone.

- Move website into the NAU domain

  As it stands, the website that hosts the database is currently being hosted on Microsoft Azure. There was some work done to try to get the website and database hosted by NAU instead, but it didn't work out with NAU ITS. By having the website hosted by NAU, it would not only give NAU some insight into the project, but also help our clients with any future funding or grants.

  Again, this is something that we worked with our clients to try to accomplish during development. But, it just didn't work out and we were not successful.

- Password recovery in the mobile app

  Almost any and all applications contain some form of password recovery in the event that it is forgotten. However, password recovery is something that we as a team just overlooked and didn't think of until development was beginning to finish up. This task would still be a tad difficult to accomplish as the mobile application does not take in user email (due to HIPAA compliance) which is often required for password recovery.

- When creating an admin account on the website, send an email to that user

  Unlike the mobile application, the web component hosting the database does store the email of an administrator. It would be best for the security of an administrator that once their email is signed up with the web component, they receive an email asking them to confirm that it was them that chose to associate their email with the web component.

  This was something considered during mid development, but was not considered to be critical for this stage of the capstone.

# VIII. Conclusion

Aging is inevitable, but the goal is to age well by having access to resources, services, and support including a continually expanding number of assistive technology (AT) products designed to promote safe independence. The problem is that although there are thousands of AT devices on the market, many older adults and their family members do not have the knowledge or resources to connect with these devices or even realize they exist. This is where our solution comes in; we aim to create an easy to use mobile application that will guide individuals towards information and AT targeted to their specific needs to help maintain or achieve more independence especially for routine tasks that they do at home on a daily basis.

This document was created in order to introduce and outline the problem at hand and to provide insight towards the software designs that will implement this solution. This document also helped us explain specifically what steps are being taken to ensure our solution is being met through our implementation plan. We showcased the big picture that we are after with our implementation overview and the technologies that will be used to reach these goals. Our architectural overview outlined the specific components of our software so that we can see how each feature is implemented. Most importantly our module and interface descriptions broke down each component on a deeper level so that we left no stone unturned when it comes to understanding every aspect of our system.  After everything we've considered and discussed in this document, we feel confident about the state of our project and know we are on track to delivering a promising product to our client.

# IX. Glossary

**Administrator:** The administrators can be thought of as those who will be using the front-end web component to help manage the database. In this context, the clients for this project and administrators can be thought of as the same entity.

**Assistive Technology (AT):** a type of technology that can assist people in a variety of ways. Some assistive technologies are aimed at those with limitations and others are not aimed towards any group of people at all. However, many assistive technologies are designed with disabilities in mind.

**Clients:** Kelly Robts, Ph.D and Jill Pleasant, MA are the clients for this project. This means that they have requested that this product be developed by Team HomeAide and will take over the product maintenance and further development once the initial development period has finished.

**Database:** A software that specifically is used for storing similar information in an organized manner. A database can be thought of as an inventory of raw information.

**Front-end:** This refers to the web component that will be used to host the database. In this context, front-end refers to the part of the website that the client will have direct interaction with (such as clicking on buttons or typing text into text boxes). Backed refers to something that the client will not have direct access to, such as the database, and will only be able to update the database through front-end actions.

**HIPAA**: Health Insurance Portability and Accountability Act of 1996. This is a regulation that protects the privacy and security of certain health information.

**Limitation:** A type of impairment that someone may have that impedes on their ability to complete certain tasks.

**User:** The user refers to those who will be using the mobile application component of this project.

**Virtual Machine:** A virtual machine is software that emulates a computer system. They are based on modern computer architectures and simulate the same functionalities as a physical computer.

# X. Appendix A: Development Environment and Toolchain

**Hardware**:
For the hardware that our team used we had different environments where two of us developed on Microsoft Windows while the other two developed on Ubuntu Linux. Overall, all of the team members had at least an i5 processor or its equivalent in their computer, and at least 8GB of RAM. With these specs in mind none of us had issues with running the Android emulators on our computers, which was the most hardware-demanding part of the production cycle.

**ToolChain**:
There was an array of software tools that were used in the development cycle of our app and many of those we felt were crucial in the production of our project.

Visual Studio Code (VSCode):
When it came to our development environment, all of us used VSCode to write and organize our code. VSCode is the source-code editor that is optimized for building and debugging web and cloud applications which makes it a perfect fit for us. It was especially helpful because VSCode comes with a lot of plugins that can be useful for developing a mobile app and it has very nice debug features. It was also chosen for its integration with the Android emulators so that we could easily test changes in the app.

With VSCode we had to install two plugins that were necessary for development. One of which was the Dart plugin to provide support for the Dart language and debugger. The other plugin that was necessary was the Flutter plugin which helps with debugging and building the app.

Android Studio:
Android Studio is an integrated development environment that is designed specifically for Android development. Android Studio is required for the Android SDK and Android emulator management.

Github:
Github is a provider of project hosting for software development and version control that utilizes git. For this project we needed to use GitHub to manage the source code so that each team member could work on separate features at the same time, and to track issues.

Apache2:
Apache2 is an open-source HTTP server for modern operating systems for both Windows and Linux/Unix. This is needed to serve the Django website to HTTP clients.

The plugin mod-wsgi is required for Apache2 to work with Django.

Django:
Django is a popular free open-source web framework based on Python. It has a lot of built-in features that make it easy to build a website with a lot of functionality very quickly. It also includes security features to protect access, and built-in support for the SQLite database.

SQLite:
SQLite is the most commonly used database for mobile applications. Since this is a popular database for mobile applications we felt it would provide the best use for us and this is needed for the project because a database is required to store all user information with regards to the mobile application. Django includes SQLite by default.

Microsoft Azure:
Microsoft Azure is a cloud computing service that is used for testing, building, and managing applications and services through data centers. Azure was needed for this project as a means for hosting the database and web framework, and having the project on a cloud server is helpful as it provides a means for secure and reliable access to the services with the project.

**Setup**:
The setup of the technologies is slightly different for Windows and Linux, so we will go over each separately starting with Windows.

## Microsoft Windows

### A. Flutter

1. Install Visual Studio code (https://code.visualstudio.com/)
2. Install Android Studio (https://developer.android.com/studio/)
3. Get the Flutter SDK, download the Flutter SDK located here: https://storage.googleapis.com/flutter_infra/releases/stable/windows/flutter_windows_2.0.5-stable.zip
4. Extract the zip file and place in the contained flutter in the desired location for the Flutter SDK such as `C:\src\flutter`

5. Update your path

    a. From the Start search bar, enter ‘env’ and select Edit environment variables for your account

    b. Under User variables check if there is an entry called Path:

    If the entry exists, append the full path to flutter\bin using ";" as a separator from existing values.

    If the entry doesn’t exist, create a new user variable named Path with the full path to flutter\bin as its value.

6. Run flutter doctor

    a. From a console window that has the Flutter directory in the path, run the following command to see if there are any platform dependencies you need to complete the setup:

    ```
    C:\src\flutter>flutter doctor
    ```

    b. Based on the results of this check the output carefully for some other software that may need to be installed or other tasks that may need to be performed.

    c. Once any issues are resolved and everything is installed, run flutter doctor command again to verify that everything is set up correctly.

7. Set up the Android Emulator

    a. Enable VM acceleration on your machine.

    b. Launch Android Studio, click the AVD Manager icon, and select Create Virtual Device.

    c. Choose a device definition and select Next.

    d. Select one or more system images for the Android versions you want to emulate, and select Next. An x86 or x86_64 image is recommended.

    e. Under Emulated Performance, select Hardware - GLES 2.0 to enable hardware acceleration.

    f. Verify the AVD configuration is correct, and select Finish.

8. Install the Flutter and Dart Plugins.
    a. Start VSCode.
    b. Invoke View > Command Palette.
    c. Type “install”, and select Extensions: Install Extensions.
    d. Type “flutter” in the extensions search field, select Flutter in the list, and click Install. This also installs the required Dart plugin.
9. Validate setup with flutter doctor again.

## B. SQLite

1. Download SQLite tools
    a. First, go to the https://www.sqlite.org website.
    b. Second, open the download page https://www.sqlite.org/download.html
    c. Select an appropriate version to download for your operating system and download.
2. Run SQLite tools
    a. First, create a new folder e.g., `C:\sqlite`.
    b. Second, extract the content of the file that you downloaded in the previous section to the `C:\sqlite` folder. You should see three programs in the `C:\sqlite` folder
        - sqldiff.exe
        - sqlite3.exe
        - sqlite3_analyzer.exe
    c. Open the command line window and navigate to the `C:\sqlite` folder

d. Type `sqlite3` and press `enter`, you should see the following output:

```
C:\sqlite>sqlite3
SQLite version 3.29.0 2019-07-10 17:32:03
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

## C. Django

1. Install Python
2. Open Powershell
3. Verify Python Installation
    a. Type `python -V` on the prompt to verify that Python has been successfully installed. You should see the Python version installed, being printed like below:

        Python 3.7.4

4. Upgrade pip
    a. Run the command:

    ```
    python -m pip install --upgrade pip
    ```

5. Create a Project Directory
    a. To create the directory run the command:

    ```
    > mkdir django_project
    ```

    b. Change into the django_project directory by running the command:

    ```
    > cd django_project
    ```

6. Create Virtual Environment
    a. To create a virtual environment run the command:

```
> python -m venv venv
```

7. Activate Virtual Environment
    a. Run the following command to activate the virtualenv:

```
> venv\Scripts\activate
```

8. Install Django
    a. Run the following command:

```
(venv)>pip install django
```

9. Start a new Project
    a. Run the following command:

```
(venv)> django-admin startproject testsite
```

    b. Navigate to the project directory by using the following command

```
(venv)> cd testsite
```

10. Run the Server
    a. Run the server by typing

```
(venv)> python manage.py runserver
```

# Ubuntu Linux

## A. Android Studio

1. Download and install Android Studio (https://developer.android.com/studio/)
2. Install at least one Android emulator
    a. Launch Android Studio
    b. In the AVD Manager, create a virtual device
        i. Choose a device definition
        ii. Select a system image for the device

## B. Flutter

1. Install using snap (install snap if you don't have it installed)

```
$ sudo snap install flutter --classic
```

2. Run Flutter doctor

```
$ flutter doctor
```

## C. SQLite3

1. Install from apt-get

```
$ sudo apt-get install sqlite3
```

## D. Django

1. Install pip

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

2. Set up a Python virtual environment

```
$ python3 -m pip install venv
$ python3 -m venv /<path to virtual environment>
$ source <path to virtual environment>/bin/activate
```

3. Install Django

```
(venv)$ python3 -m pip install Django
```

4. Install the required plugins for this project

```
(venv)$ python3 -m pip install django-crontab
(venv)$ python3 -m pip install django-dbbackup
```

**Production Cycle:**

For the production cycle the first thing to be done is download the latest version of the source code for the application that is located in the GitHub. From here we will extract the zip file to a folder promptly named for the version of the code that is being worked on. Following this we will open up Visual studio code and open the folder which contains all the code to bring it into the workspace so we can begin to code. Now that the code is loaded into the workspace we will run the code on the emulator to make sure that everything is running smoothly and there are no issues getting the app to run. From here if we would like to create a new version of the app we will look at what we want to implement, for example creating an alert dialog to confirm the user would like to logout. We would then look at how to implement this feature and then once implemented we will go ahead and run the app on the emulator to ensure that what we implemented is functional and didn't break anything else in the code. Following this, when we

feel that this feature has been properly implemented we will make sure the files we made edits to are saved. Then we will go into github and make a pull request for the changes that have been made and title these changes accordingly. Then another team member will take a look at the changes that were made and run it on their machine to ensure that everything is working and from there we will merge that pull request with the main branch and we have a new version of the application located in the GitHub repo.