Team HomeAide

Software Testing Plan

March 26, 2021

Version 1.0

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

Within the United States, there are roughly 23,000,000 individuals aged 60 or older that are living with at least one limitation that impairs their ability to live life comfortably or independently. At the moment, there are thousands of Assistive Technologies (AT) that are available on the market that would assist these individuals with their limitations and improve their quality of life. The problem is that many of those in need of AT devices are unaware of what is available to them. The goal of this project is to be a means to help provide information about these devices that will assist those living with limitations or impairments.

The solution to this is to build a mobile application that will provide users with information about AT devices that may be most helpful to their situations. The user will be able to input information about what they may be having difficulty with and/or what areas of their house are troublesome for them. The mobile application will be supported by a database that will hold all information about AT devices and how these devices may interact with a user's home or limitations. Administrators will be able to access this database via a front-end web application.

Given that the current project is being developed with the idea that it will eventually be put onto the market for use, proper software testing is required. Software testing is a critical component of software development, especially when working with multiple different modules that all have to interact with each other. There are various approaches to software testing, and it generally is up to the developer how extensive they want to be. Software testing could range from testing each and every small function or method to see if they generate the desired results, to testing that multiple methods and functions can successfully work together to generate the desired results. It also allows the developer to detect any vulnerabilities or issues that may exist in their product, even if their product returns the desired results.

Testing for this project will be observing if multiple methods/functions can operate together to produce the expected results. How the testing is being conducted is also further broken down into what type of application is being tested. The mobile application has testing that is specific only to the mobile application, the web application/database has testing specific only to itself, and both will contain testing on whether or not mobile application and web application/database can communicate with each other. When testing each module, it is even further broken down into what that module is capable of doing. For example, the mobile application has the functionality of being able to recommend AT devices to a user based upon any identified limitations or trouble spots in their home. Testing will be conducted so that the app can consider these limitations, use those limitations to run a search in the database, retrieve devices from the database that are relevant to these limitations, and then present them to a user in a readable fashion.

Our team believes that observing how multiple methods/functions operate together to be a better form of testing rather than just testing individual functions by themselves. While developing these individual methods, extensive testing was already conducted to make sure they work at all. At this point, there is already confirmation that the methods do work on an individual level. But, their functionality really means nothing if they are not used in their appropriate context. For example, the database contains a simple and advanced searching function that will search through a database and return AT devices contained within the database based on some keywords. This searching ability does indeed work. But, as mentioned in the example above, the recommendation algorithm also relies on this search function to be able to do what it needs to do. In this context, testing will conclude that the searching method can help assist the recommendation method in completing what it needs to be done. In order to observe how these methods work together, the project was broken down into unit testing.

UNIT TESTING

Unit testing involves testing individual modules within each major component such as the user management unit of the website, or the reading and writing to and from storage in the mobile app. This will also help in determining whether or not the product is performing optimally. The units for this project are broken down in the two main sections: the website and the mobile application. From there, we group functions that are similar in scope within both of those modules, and further down into similar functionality. The website will be concerned with administrator user management and inventory management. The mobile application will be divided into classes which correspond to the database tables, and the global functions that connect the UI to these classes and to the website.

In general, our unit tests will be created ourselves using driver functions written in the same language. These driver functions will test the boundaries for inputs, and validate outputs. We are using a "black box" approach where we know 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.

Website

The website module deals with administrative user management and database management for the inventory. These functions depend heavily on Django's built-in libraries, so a lot of error prevention is already in place. The Django server will also fail to start if it detects errors in logic which helps a lot in error prevention for when the server is running. These unit tests will be written in Python, the same language that is used by the Django framework.

Web Application: Users

Functionality for this aspect of the website is specific to how users, or administrators in this case, would be given access to the website.

Unit

1. Functions that give administrators access to website

1. Functions that give administrators access to the website

Description:

These functions provide administrators access to the website, which in turn gives them access to the database. Administrators will not only be able to create or remove the accounts of other administrators, but they will also be able to monitor activity or even disable accounts. In order to test these functions, all that really must be done is attempt to give or remove an administrators ability to access the website. There are two ways to monitor whether or not these functions are successful: (1) have an administrator attempt to log into the website or (2) view the section of the database that stores administrative information and determine if the function was successful.

Boundary Values:

These functions do require that the user input valid data when prompted to. For example, creating a new administrator account requires viable data in a desired format. An administrative account requires at least a username and a password. Although there are no restrictions to the type of password, the username must be in the format of an email. Without a valid email input, a new administrative account will not be created.

Example Values:

function signin(String request)

This function is what would allow an administrator to sign into the website. It will take in a request (which contains the information the administrator provided) and check it against the database to see if this information belongs to an administrator saved by the database. If this has been validated, the administrator is redirected to a page that gives them full access to the website. Otherwise, they remain on the same screen and receive a warning that the information provided is incorrect.

Expected Behavior:

If the administrator enters in valid and viable data where appropriate, the administrator account will be able to gain access to the website. Failure to enter the proper data may result in the administrator not being given access to the website or the ability to update information related to their account.

Web Application: Main Functionality

Functionality here is all about how the administrator is able to interact with the website. Other functions aren't really related to the administrators and will run on their own in the background as they are responsible for authenticating requests sent from the mobile app.

Units

1. Functions that allow administrators to manipulate information in the database
2. Functions that allow administrators to retrieve information in the database
3. Functions that communicate with the mobile application
4. Functions that authenticate mobile requests
5. Function that gives AT recommendations

1. Functions that allow administrators to manipulate information in the database
Description: These functions give administrators front-end access to the database itself. Not only will the administrator be able to view the data, they will also be able to manipulate data in the database. These include being able to add, remove, or edit information about AT devices, Rooms, Room Objects, Limitations, Surveys, and State Resources.
Boundary Values: A significant portion of changes that can be made into the database require text input. The exception from this will be in terms of deleting entries which is performed via buttons. Some inputs are required and the website includes form validation to make sure that they have been provided, which means that the empty string is not allowed.
Example Values: function limitations(HttpRequest request) This method allows the user to add, edit, or remove a new limitation to be stored into the database. The request object must be a valid HTTP request, and it is impossible to call this function without an HTTP request because it is directed from a browser request. If the request method is GET, the function returns a response for a page used to display all limitations in the database. If the request method is POST and includes an action of the following: 'new', 'save', 'edit', or 'delete', the function will take the corresponding action on the database and return a listing of the limitations after the action has completed.
Expected Behavior: As long as the user provides all of the mandatory information, these changes will be

reflected in the current state of the database and these changes will be made immediately. Once these changes have been finalized, they will also be reflected in the mobile application as well.

2. Functions that allow administrators to retrieve information in the database

Description:

While making attempts to manipulate information contained within the database, information about entries will immediately be put on display for the user. However, this will contain every single entry of its category. The more entries that exist in a category, the more difficult it is to read through all available options. To make this easier, there is a searching mechanic that allows administrators to look up specific information within the database.

Boundary Values:

Much like manipulating information, administrators will also need to provide information via text input. Here, there isn't a mandatory type of input. However, if the administrator were to provide no information, the search will provide no results. If the administrator were to provide some information about what they are looking for, the database will return results if the database finds similarities between entries and what the administrator inputted. If no similarities are found, the database will then return no results.

Example Values:

function search(List keyword)

The example above will take in a list of keywords that the administrator has entered from their query. The keyword parameter must be a List object, but can be empty. In the case that it is empty, it returns all entries from the database. Let us assume that *keyword* is the following list: ['kitchen', 'echo']. The function will search the database for entries that include 'kitchen' and 'echo' in their fields.

Expected Behavior:

Upon inputting information into the proper text boxes, the database will be able to search through itself to find what the administrator may be looking for. Depending on how detailed the administrator was, the database may return with a few results or precisely one. That is, if the database happens to contain what the administrator is looking for. Regardless, the database will redirect to a results page containing a list of all entries that the database found. Sometimes, this may mean that the database found nothing, so the results page will be empty.

3. Functions that communicate with the mobile application

Description:

The mobile application allows users to send feedback to the administrators, and the website allows administrators to send messages to mobile users. These functions provide the functionality for sending or receiving data between the website and mobile app. The mode of communication is through JSON objects sent through HTTP requests and responses.

Boundary Values:

These functions rely on JSON objects with the correct data provided for that function, so at a minimum, a JSON object is required. Each function expects specific values to be included in the JSON that are specific to that function.

Example Values:

Function submit_suggestion(HttpRequest request)

This function allows the mobile app to send a JSON with information about a suggested AT device. Let us assume that the request object includes the following JSON: [{'handshake_id': '1', 'handshake_key': '350f6899a854738a714db3e1486cedfc', 'at_name': 'Acme Pro'}]. This function will authenticate the request using the handshake id and key, and will add the suggestion to the database if it passes authentication.

Expected Behavior:

If the JSON is not provided, or if the values expected in the JSON are missing, an HttpResponse object is returned with an error message. If the correct values are present in the JSON, the functions perform their actions and return an HttpResponse object with the requested data.

4. Functions that authenticate mobile requests

Description:

Before allowing access to the database, it needs to make sure that the agent requesting access has permission to access the database. This is done through a handshake protocol in which the database will authenticate that the request is coming from the mobile application. If this request is authenticated, it will be able to retrieve information from the database.

Boundary Values:

The handshake protocol that is conducted upon request needs an id and a key. These two values are going to be linked to a user profile. If the request is not a valid request (meaning it is not coming from a genuine version of the mobile application), access to the database will be denied regardless of what the id or key contains. But if the request is considered authentic, the database will then use the id and key to sync with a user profile that is stored in the database.

Example Values:

function authenticate_app(int id, String key)

This instance will check to see if a user profile already exists for the application based upon the given id and key. Let us assume an id value of 1 and a key of '350f6899a854738a714db3e1486cedfc'. This function will check that this pair matches a stored pair in the database and returns True if it matches, otherwise it returns False.

Expected Behavior:

If a genuine version of the mobile application is attempting to access the database, the application will be given access. It will use user information to retrieve information about their profile that is contained within the database and allows the users to make live updates to their profile. If no user profile exists within the database, it will create one for them.

5. Function that gives AT recommendations

Description:

To provide AT recommendations to the user of the mobile app, the mobile app sends information about the user's location in a house as well as any of their specified limitations. This function in the website handles the logic to provide appropriate recommendations back to the mobile app.

Boundary Values:

Similar to the functions that communicate with the mobile app, this function requires an HttpRequest as an input. This request needs to use the POST method and include a JSON object with the app authentication values, user limitations, and the area of the house to look in. The authentication values must include an int for the ID and a String for the key. The limitations are a comma-separated list as a String of the primary key ID for the corresponding limitations in the database. The limitations can be an empty String. The area of the house is given by an int for the room's primary key ID in the database or the room object's primary key ID in the database. The room is required but the room object is not.

Example Values:

function get_devices(HttpRequest request)

Let us assume we have a request object from the mobile app using the POST method. The request object contains a JSON with includes the following values
id: 1
key: '350f6899a854738a714db3e1486cedfc'
limitations: '3,5,15'
room: 2
room_object: 3

Expected Behavior:

If any of the expected values are missing or the authentication fails using the id/key pair, an HttpResponse is given with an error message. If the values provided by the request have erroneous data such as a room ID that doesn't exist, or characters provided instead of an integer, the function will return an HttpResponse with an error message. Otherwise, if all the JSON information is correct, the function will return a JSON object that contains information about the AT in the recommendation.

Mobile Application

The mobile application involves functionality involving the app UI and interacting with the website. Many of these functions rely on third-party libraries for accessing the device storage and for connecting to the website, and we will be assuming that these libraries have proper testing done by their authors. We split this module into two subsections: classes and global functions. The classes are used to create objects that generally correspond to the tables in the database. This allows the app to access the data in an object-oriented manner. The global functions deal with communication with the website, reading and writing to and from the device storage, and authentication. The driver functions for these unit tests will be written in Dart, the same language which is used by the Flutter framework.

Classes

The functions covered in this section are class functions. These classes correspond to the database tables in the website to provide usable objects for the application. They mostly include functionality to change data of the object or to manipulate how the data is organized for exporting for actions such as creating a JSON.

Units

1. Constructor functions
2. Functions that convert the class fields to JSON
3. Class sorting functions
4. Debug functions

1. Constructor functions
Description: These functions create the class objects. Since these classes correspond to database tables, they have fields that are required and field types that relate to the database fields. By creating an object of a class, the data from the database becomes usable for the app.
Boundary Values: The required fields must be provided, however optional fields can be left out of the constructor call. Parameters that require an int can be any integer value. If an int parameter is an incorrect value for that object, it will be validated and handled by a function that tries to retrieve it from the database.

Parameters that require a String can be of any String value including the empty String.

Parameters that require an instance of a specific class can take an instance of that class or a Null object.

Example Values:

class Room(int id, String name, int order, String icon, RoomObject[] roomObjects)
To create an object of type Room, all of these parameters are required. Examples of correctly calling this constructor include the following:

new Room(1, 'Kitchen', 3, 'kitchen.jpg', null)
new Room(-3, '', -100, '', null)

Expected Behavior:

Dart does not allow calling a constructor without all the required fields, and parameters of the wrong type. If one is missing or of the wrong type, compiling the code will fail. If the parameters are correct, an instance of the class will be returned.

2. Functions that convert the class fields to JSON

Description:

These functions parse the class fields into a JSON object which can be easily stored in app storage or sent to the website.

Boundary Values:

These functions take no inputs.

Example Values:

class AT().toJson()
Returns a JSON object with the fields of the instance of the AT class.

Expected Behavior:

The JSON object returned has correctly formatted values representing the fields of the class.

3. Class sorting functions

Description:

Some classes have a List as a field. It is sometimes necessary to sort the list such as when to determine the order of the Rooms to display in the app. These functions sort the List and save the sorted list back to the objects fields.

Boundary Values:

These functions do not take any input. The condition to sort on is predetermined.

Example Values:

class Room().sortRoomObjects()

A RoomObject has an integer field for ordering. This function will sort the list of RoomObjects for that Room.

Expected Behavior:

There is no return value. After calling these functions, the list of objects will be sorted and saved to the parent object's field.

4. Debug functions

Description:

These functions are not required for the production of the app, but are used for debugging issues that may occur. They print out information to the debug console that will be helpful in troubleshooting.

Boundary Values:

These functions take no inputs.

Example Values:

class Room().printRoomObjects()

This function will iterate through the RoomObjects contained within this class and provide a readable String output that can be printed in the debug console.

Expected Behavior:

These functions return a String that shows information about that class object to be displayed in the debug console.

Global Functions

The functions in this section connect the components in the app with the UI and phone storage. They also communicate with the website.

Units

1. Functions to communicate with the website
2. Functions that manage the phone storage
3. Authentication functions

1. Functions to communicate with the website
Description: These functions serve to send and receive data to and from the website. Since these functions communicate over a network, they are prone to network problems which must be handled. In order to test these functions, we will simulate the network problems by disabling responses from the website or by making the website unavailable. The functions will need to handle a situation where a response is not received in these cases, as well as the usual case where the response may not be the expected data as defined in the boundary values. The communication over HTTP with the website relies on a third-party library called "http" which has some built-in features to protect against errors, including a timeout when waiting for an HTTP response.
Boundary Values: All of these functions require a HandshakePair object which includes an ID and key pair that the website needs to authenticate the request. Unacceptable inputs would include NULL or an object from another class other than the HandshakePair class. A few of these classes also require an int or String input that needs to be passed along to the website. In these cases, the input is always required, and they cannot be NULL or of the incorrect type. Since these values are of the correct type and passed to the website, it is the website's responsibility to determine the correctness and handle incorrect data. Therefore, there is no restriction on the range of accepted values.
Example Values: function getResourcesFromServer(HandshakePair pair, String state) In this case, let us assume we have an instance of the HandshakePair class called "handshake_pair." Then we can call this function as such: getResourcesFromServer(handshake_pair, "Arizona")

Expected Behavior:

Upon receiving data from the website that falls within the boundary values, the functions will parse the response into a data object from one of the classes defined in the previous sub-section. The now usable data object is then returned to the calling object.

If the response is not received before the library's defined timeout expires, or if the response does not include the expected data in the correct format, these functions will return a NULL object.

2. Functions that manage the phone storage

Description:

These functions read and write data to the persistent phone storage. They rely on a third-party library called "share_preferences" that handles the low-level operations and has its own error-checking. To test our functions in this unit, we will utilize our own testing suite of driver functions also written in Dart. This test suite will call each function in the unit, pass in parameters, and verify that the returned data is what is expected as defined in the expected behavior below. Validation on the inputs being within a certain range is not handled by these functions and are instead the responsibility of the calling object.

Boundary Values:

Each of these functions which save data to the phone storage requires inputs for the data that is to be saved. In the case that the expected input is an int or a String, the parameter type must match, however there is no restriction on what the actual values of that type are, as long as they are not NULL. In the case that the input is a specific class object, the instance must be of the correct type, but can also be NULL.

None of the functions which retrieve data from phone storage have inputs.

Example Values:

function saveHandsharePair(String id, String key)

We can call this function in the following ways without error:

saveHandshakePair("", "")
saveHandshakePair("0", "some secure key")
saveHandshakePair("some secure key", "0")
saveHandshakePair("-1", " ")

Expected Behavior:

The functions for saving data return a Boolean. If the function was successful, it returns True, otherwise it returns False.

The functions for getting data from storage return a Future of a specific type. If the data does not exist in storage, or if there was an error retrieving stored data, the Future returns a NULL object. Otherwise it returns the expected class's object if it is successful.

3. Authentication functions

Description:

These functions handle the authentication for when the user logs into the app, and for managing the app's unique ID and key pair for securely communicating with the website. Since these functions depend on user input which is validated using Flutter's built-in widgets, we will test them by providing inputs via Flutter's widgets and verify they are working by printing out messages to the debug console.

Boundary Values:

A username can be any String value except for the empty String, and up to 32 characters in length. Any character except for escape characters are valid.

A password can be any String value except for the empty String, and between 8 to 32 characters in length. Any character except for escape characters are valid.

The app ID must be an integer, but is saved as a String. The integer value must be greater than 0, but there is no upper bound.

The app key must be a String value exactly 32 characters in length. Any character except for escape characters can be used.

Example Values:

Username: "myemail@server.net", "app_user", "my profile"
Password: "12345678", "4s3cr3tp455w0rd"
ID: "1", "10000"
Key: "350f6899a854738a714db3e1486cedfc"

Expected Behavior:

These functions forward the behavior of the functions they depend on which return Boolean values. If they are successful, True is returned. If the inputs are invalid or a problem occurs, False is returned.

INTEGRATION TESTING

This section focuses on the interactions and exchange of data among the components discussed in the previous section. Integration testing describes the valid behavior of these interactions among the various units, and whether the data is being exchanged correctly. While the previous section only focuses on specific components as isolated from the system, integration testing makes sure that these specific components are working together without fault. This is important to validate the integrity of the entire system. If the individual components work as intended but the components do not interact correctly, the system will be full of problems.

To determine what to test, we need 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 need to test for situations involving network connectivity and security. The interaction between the website and the database needs 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.

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.

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.

Now that we have covered the lower-level components as well as their integration with each other, the last piece of the puzzle is the user interface. This is discussed in the next section with usability testing.

USABILITY TESTING

Usability testing focuses on the end user and the actual software systems so that the users can properly utilize the intended functionality of the software. The goals of the usability testing is to showcase the overall quality and ease of navigation that our system has with regards to the user interface. How usability testing works is we take an end user and have them explore the software and have them evaluate the ease of use that our software provides and inform the developers how usable and intuitive the software is and if it helps them reach their goals.

Mobile App End Users:

Having an easy to understand interface for the mobile application is important due to the fact that the age range of our end users may vary greatly. As such it is important to test to make sure that the user interface is as simple to understand as possible to ensure that the app is accessible to everyone who ends up using it. It is also important that the key roles of the app do not take a long time to understand otherwise the end user might decide to seek out other, easier to understand options in order to help them discover new AT devices.

The user interface for the mobile application needs to be easy enough to use so that the end user can perform all the necessary actions required to be given relevant AT device recommendations. This will include being able to define their general limitations, navigate through different rooms within the virtual house, and saving or sending the recommendations they receive should they choose to do so. In order to test how easy it is for users who do not have any prior experience with the mobile application to perform all of these tasks we will select potential test users from a varying set of age groups and with varying sets of difficulties in order to not only test how accessible our app is for people of different age ranges, but also to determine if more specialised difficulties results in a noticeable change in how effectively the end user is able to receive AT recommendations.

In order to test our user interface 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.

Website Admins:

For the Website Admins there are several key features that they will need to understand how to perform in order to keep the backend up to date as well as the initial data entry into the database. Our clients have expressed concern when it comes to the amount of data entry that will be required in order to provide the end user with as much information about AT devices as possible. For this reason it is important that the user interface for the website is intuitive so that the website admins are able to perform all the required data management as efficiently as possible.

In order for this to be achieved the website admins need to be able to add several categories of data including new users, rooms, room objects, and limitations as well as be able to combine this data into the main AT device table. For our testing plan for the website admins we will focus on the amount of time that it takes them to perform each of these tasks, analize what areas took the most time to complete, and take note of every time that the admins got stuck trying to navigate the website. It is also important that the current admins are able to understand how to perform each of these tasks well enough in order to be able to show potential new admins how to do them in the future.

We will 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.

CONCLUSION

There are thousands of AT devices on the market that are ready and able to help assist those who need them the most. Team HomeAide is excited to be able to produce a solid product that will help those gain access to information about these devices. This product will be a combination of a mobile application and a web application that will host the database that will provide information to the mobile application.

In this document, we laid out the plans for testing from the bottom-up. This means starting with units of similar functions at the lowest level for each module. This ensures that we have a solid foundation of reliable functions to build on. The next step was to plan testing for how these units interact with each other as a system. This is important because while a function may be solid in it's individual requirements, it needs to work together with the other functions to provide a reliable system that functions properly. The last step after verifying the system is to test usability of the system with the people who will use it. The last step is also very important because a system that works well but isn't user friendly won't be of much use.

Our methods of testing include writing our own driver functions for the low level functionality, system modification for the integration, and observation of users using the system to test usability. Having covered all aspects from the lowest level up to the users, we are confident that our project will be reliable and work as expected.