Software Testing Plan
Version 1
4/1/2022
Rehab Remote

Project Sponsor
Dr. Zachary F. Lerner
BiOMOTUM, Inc.

Faculty Mentor
Felicity H. Escarzaga

Team Members
Kylie Cook
Brandon Roberts
Robert Bednarek
Katarina Marsteller

Table of Contents

Introduction

In the United States, 1 in 345 children has cerebral palsy, a set of neuromuscular disorders effective around birth. Cerebral palsy can reduce the mobility of those that have it. Around 58.9% of children with cerebral palsy can walk independently, 7.8% walk with assistance, and 33.3% have limited or no walking ability [1].

The current treatments for cerebral palsy consist of medication, therapy, and surgery. Biomotum stands within the therapy category by developing a battery-powered ankle exoskeleton that increases walking speed by 32%, stride length by 21%, and improves efficiency by 29% [2]. The company has partnered with Rehab Remote to create a web application for users such as clinics, hospitals, etc. to have the ability to access data about their patients.
Rehab Remote is currently in the software testing phase of the application creation. Software testing is necessary to ensure that the software works as expected, as well as every individual component and components that are used together for an expected outcome. In order to efficiently put the web application through software testing, the team uses two different types of testing: unit and integration. Unit testing tests individual modules in the application, while integration testing tests those modules together.

For the unit testing, the team plans to separate front end testing from back end for better testing organization. The front end testing includes simple code creation to ensure that the correct information is being retrieved from databases, button functionalities respond correctly, graphical interfaces are correct and consistent, and there is not too much loading time between button functionality and data retrieval. Back end testing includes testing inputs for each function stored in the back end of the application and comparing the outputs the function provides to outputs that the team expected.

For integration testing, the team plans to do both human testing for the physical front end section of the Wix site and Python testing for the functions that are primarily back end and deal with file organization to be stored in the Biomotum Google Drive. The testing plan summarized above was created due to using Python scripts within AWS cloud servers, JavaScript functions within Wix, and front end CSS capabilities with Wix's available elements. Each testing plan thoroughly tests each of these pieces. The main portion of testing is in the back end due to the front end HTML and CSS being controlled and monitored by Wix. The portion that makes all databases and websites work together correctly remains in the back end and is most necessary for the website to retrieve the correct data for users.

Moving forward, the next sections explain these testing plans in depth, with more insight on what each test is about, along with different functions tested for unit testing, the modules put together for integration testing, and how we plan to test the website with end-users.

Unit Testing

Software unit testing is the first stage of the software design process life cycle. Unit testing involves taking apart each of the major components of the project as a whole and breaking them down into their individual components to be tested with certain conditions. For example we can compare our web portal to a car. When testing our "car" we will split our car up into parts such as, wheels, engine, suspension, chassis, and so on. We then take each of those parts and ensure they meet the conditions they were designed to handle such as the doors properly closing, the engine delivering power and more. The general intent is to ensure that all parts of our codebase gets tested individually to ensure that they are all functioning properly. The goal is to ensure that no major outstanding defects are delivered when presenting our project to our client. We will be splitting our unit tests into two groups including front end testing and back end testing.

Front End

The front end of our project is the face of our work, this is what the client and users will see and interact with at all stages of the interaction. It is vital that we ensure the front end is thoroughly tested to ensure that the user does not encounter any major errors with the interface. Our front end is Primarily composed of Wix tools and Javascript that is in charge of interacting with the buttons to achieve certain functionality. Our testing will consist of testing the functionality of the Wix widgets, the buttons, the responsiveness, and the functionality of our javascript code, that is in charge of receiving the data from the buttons. We will start with the beginning of our front end development functionality by initially testing our homepage.

In order to test the functionality of the front end portion of our site we need to ensure that:

1. The front end correctly references and collects data from our database
2. That when the submit buttons is pressed, the proper database requests are sent
3. The front end button functionality remains reactive and consistent
4. The graphical displays stay consistent error free
5. How fast can we load the information we are looking for?

1. In order to test that the front end is correctly references and collects data from our database we will create new locations that attempt to reference the data when a button is triggered, IE: if we click a button that says "retrieve data", our text box will then be filled with the data that is requested, whether that be the username, the device id or anything else. This interaction will be tested in multiple locations to ensure functionality.

2 . Now in order to test that the proper database requests are being sent and correct information is then processed, we will create a default request template. This request template will be used to

send a request to the database, and observe the value that is then sent back to the default request template. If the data is expected, then we know that the proper requests are being sent.

3. This unit test will ensure that the user experiences no frustration or difficulty when interacting with the buttons on our website. We will need to ensure that the buttons remain functional and also that the buttons functions do not deviate from the guidelines. Our testing method for this will consist of creating new locations that react when the specific button is clicked. When the button is clicked the reactions should remain consistent and will be proven to do so by having statements that react accordingly when the interaction happens.

4. The process for going about testing our graphical displays and ensuring that they remain consistent and error free is the most intensive part of our unit testing. In order to properly test our display we will create a new module that exists with the sole purpose of receiving data and then creating a display. We will ensure that the display remains consistent and error free by inputting many different data templates, then ensuring that the display remains consistent with the data that is being sent and also ensuring the display formatting does not deviate.

5. Finally, in order to test how fast we can retrieve the information that we are looking for we will have to set up a simple module for timing. This module will be rather simple, the module will encompass our previous code that retrieves, parses and then outputs the data. First we start the timer and then after the retrieving, parsing and output is done we will retrieve the time, ensuring it remains consistent and fast. We will also conduct this test with multiple data sets to ensure reliability.

Back End

The back end of this project is where all of the major connections exist so we need to ensure that the back end functions are working properly so that correct data is displayed to the user. This is where all important calculations and functions are stored that allow for the front end to display accurate statistics and graphs. The back end will be tested by running and testing each individual function from the four back end scripts we have in our project as well as testing the databases to ensure they are updating correctly.

More specifically, in order to test the our back end portion of our site we need to ensure that:

1. All individual functions within the “google_sheets.jsw” script are working properly
2. All individual functions within the “googledrive.js” script are working properly
3. All individual functions within the “other.jsw” script are working properly
4. All individual functions within the “s3.jsw” script are working properly
5. The “Members” Content Manager database is updating and storing user data correctly

Table 1: List of Back End Functions

Script	Function	Input	Output
google_sheets.jsw	getData	Name of sheet	Spreadsheet values
googledrive.js	uploadFileByUrl	folderId, url, name, description	Google drive file data
googledrive.js	getFileById	fileId, fields	File's metadata
googledrive.js	listFiles	query, fields	Metadata of list of files
googledrive.js	deleteFileById	fileId	Status of deletion
googledrive.js	createPermissions	fileId, permissionInfo	Data from permission
other.jsw	getId	email	Id associated to email
s3.jsw	getExoList	N/A	List of exoskeletons
s3.jsw	getAccessibleData	List of exoskeletons	Dictionary for each exoskeleton
s3.jsw	downloadFile	File key string	Url of the downloaded file

As shown in Table 1, we will test the getData function from the "google_sheets.jsw" back end script by inputting an active google sheet. From there we will make sure that the values from that spreadsheet are outputted correctly or a null is returned if the google sheet is inactive.

In the "googledrive.js" script we will test multiple individual functions. The first one being uploadFileByUrl where a folderId, url, name, and description will be provided as inputs to check that the url provided is going to the correct folder specified in the google drive. The getByFileId function will be tested by inputting a fileId and metadata fields to ensure the correct metadata associated with that file specified is returned. The listFiles function will take in a query and fields as input where we will run it multiple times to ensure that the function is querying the correct metadata as specified from the list of files in the google drive. The deleteFileById function will simply take a fileId as the parameter and permanently delete the file, bypassing the trash bin so we will ensure that files to be deleted are only deleted when the user has organizer permissions from the google drive. The last function from the "googledrive.js" script is createPermissions function where a fileId and permissionInfo will be used as input to create permission for that file where we will ensure it gives the user file/folder access based off the permission information specified.

Within the “other.jsw” back end script, we have one main function called getId that retrieves the unique user id from the inputted email address. This function simply looks up the email address provided as input from the “Private Members” database and returns the corresponding unique id associated with that user so we will test this function to ensure the resulting id returned matches the same id in the database or false is returned if the email does not exist.

The last back end script is “s3.jsw” and the first function is getExoList which doesn’t have an input and simply returns a list of all available exoskeletons. We will run this function and compare the resulting list to the database to ensure that all of the exoskeletons from the database have been outputted. The getAccessibleData function takes the getExoList function a step further by taking the list of exoskeletons as input and organizing them into a dictionary so we will simply check that all of the available exoskeletons are in the dictionary and ensure the format of the dictionary is correct. The format to check for is starting from the exoskeleton, then the user of that exoskeleton, then the dates it was active and the specific trials for each date. The last function that we will be unit testing is the downloadFile function which takes a file key string as input and returns the url of the downloadable file. We will test to make sure the url is valid and able to download the file based on a valid key string or not do anything if the key string is invalid.

Integration Testing

Integration testing requires testing modules that work together at any point of application use. It is necessary to make sure that the application modules do not have any errors while working together, and that each module interacts/reacts in the appropriate way when affected by another module. While the team determined which modules should be tested and how, we based it on the characteristics above. Modules in the project that work with other modules were chosen and will be tested by triggering every interaction possible between the modules and watching for errors and/or the correct outcomes.

Some modules created during the time given for this project are not testable for the application, such as the Python script that organizes and transfers the Biomotum data files to the Biomotum Google Drive for employees to access and better understand. Instead, the script contains code that creates change and error logs to make sure all files are placed in the correct area and that all errors are tracked so they can be fixed. Most of the functions used within the script work together and can be tested using Python's library unittest. While unittest has a name that seems to be unit test specific, the library can be used for integration testing practices as well. For every function that uses a different function within the code, a specific input to the first function used is set as well as the expected outcome. Then, the set input is used as the first function's parameter, and the returned data is compared to the expected outcome. If they are the same, the integration test is successful, if not, it fails. This testing will be done for the Python script that connects to Google Drive, as well as the Python script that creates Biomotum exoskeleton and patient data summaries stored in the Biomotum Google Drive.

As for functions and modules implemented within the Wix website itself through the JavaScript language, a similar process as explained above will be used to go through integration testing. Although, instead of testing through code (like the Python unittest library), the tests will be through human interaction of the website.

To start off the testing, the Google Sheet involving the global summary of all exoskeletons and users will be manually changed to insure that changes are made on the website, since the Google Sheet is connected to the site and should automatically be updated when there are any changes made. Then, the sign up function is tested to make sure that the new user's company name shows up in the admin's dashboard, where the admin can assign and unassign exoskeletons per user. Once the user is included in the dashboard, the user's patient page is tested by first making sure that the user has no access to any exoskeletons. Then, exoskeletons will be added to the user's patient page and checked by going to the page and going through each exo, user, and data sheet provided. The removal of access to exoskeletons will be tested by removing a few, and then all exoskeletons previously granted access to the user, and the patient profile of

the user will be examined to make sure that there are no exoskeletons or users to access once again.

Lastly, the connection to each exoskeleton, user, and csv summary commences. This is in the same realm as the testing mentioned above, where access to certain exoskeletons exist. Although, this testing is to make sure that the information within each summary Google Sheet for the exoskeletons and users are correct. To do so, about 10 exoskeletons and users will be randomly chosen to compare and contrast the summary graph created within the Wix site to the summary sheets exported to the Biomotum Google Drive. The downloadable CSV (Comma Separated Values) file will also be compared to the file sitting within the Biomotum AWS account. The discussed tests can be understood through the table below:

Table 2: Integration Testing Modules

Modules	**Input**	**Output**
Google Drive AWS Integration Script	AWS Connection Information Google Drive Connection Information	Exo Folders User Folders Date Folders CSV Files
Google Drive AWS Summaries Script	AWS Connection Information Google Drive Connection Information	Google Sheet Summaries for: Exo Folders User Folders Date Folders
Google Sheet Global Summary	Different Data: Sum of Steps Sum of Steps Per Month	Wix Font Page Global Data Changes
Sign Up and Admin Dashboard Connection	Name Company Name Password	Company Name in Admin Dashboard Ability to Assign Exoskeletons to Company
Patient Page and Admin Dashboard Connection	Given Exoskeleton Access Removed Exoskeleton Access	User Ability to See Given Exoskeleton Access, Users, and Files
Google Sheet AWS Summaries and Patient Page	Given Exo, User, and CSV Files	Graphs for Each Exo, User, and File
AWS Account and Wix Download	AWS URL to CSV Chosen	A Download of the CSV

Usability

Usability testing is very important when it comes to making the client happy with the software being delivered. It focuses on the interactions that end users will have with the software system and the goal is to ensure that users are able to effectively use all of the functionality provided by the software. This is where we get feedback from the users to make any necessary user interface changes that will contribute to the ease of use of the software.

The web portal that we have developed tracks data from exoskeleton prototypes that are still being tested by Biomotum employees. Therefore, the usability testing will be conducted among multiple Biomotum employees to get their opinions on the design and usability of the web portal. This is due to the exoskeleton being in the developmental phase and has not gone out to clinics. The primary use cases that will be examined will be the end user and administrators. The end users will be Biomotum employees who will be given the role of a clinic working with the exoskeleton. The administrative controls provided through Wix will be given to a Biomotum administrator who has permissions to make changes to the accessible exoskeletons. In addition to the mentioned use cases, we will be developing a survey for both cases to help us understand exactly what the experience was.

The surveys that will be given to the participating employees will be split into two categories for both of our cases. The first is the clinic survey which will focus on the account creation, viewing of data, and the downloading of the CSV files from the portal. The second survey will be for administrators which will be the individuals working in Biomotum that approve accounts and edit the exoskeletons available to particular “clinics”. These surveys will cover design choices for the website, aesthetic and overall look of the portal, the ease of finding said features, the understanding of the features available for both parties, and overall functionality of the features.

We chose surveys as our medium of testing because it allows for mass feedback on what was bad for both use cases. The regime will have two groups of students for both use cases with both unable to interact with each other. They will be designated as administrators and clinics. The clinics will be given minimal instructions on how to create a new account under the company name of “Company Trial_Clinic_*clinic number*” and navigate to download a particular CSV file with a naming scheme of “Trial_Clinic_*clinic number*”- the clinic number will be provided upon the start of the trial. Likewise the administration will be given minimal instructions to add the device with the desired CSV for the clinic user and then take a screenshot of the device in the accessible list of the “Trial_Clinic_*admin number*” - the admin number will be provided upon the start of the trial and match the clinic number. After both tasks are completed, the users will be given the survey for the clinic and the administrator accordingly.

The survey results will provide us with lots of detailed feedback regarding each section of the use cases.

It will include tasks to complete with little instruction such as:

1. Sign up for an account with the company name: CompanyTrial_Clinic_*clinic number*
2. Once approved sign in to your new account
3. Select an exoskeleton, user, and trial

Some of the follow up questions that will be asked to receive detailed feedback will be in the form of 1-10 scale and short response answers:

1. On a scale of 1-10, how easy did you find creating an account was?
2. How did you feel about the layout and design of the sign in page?
3. On a scale of 1-10 how easy was it to navigate to the exoskeleton selection dropdown?
4. Do you have any suggestions for improvements to the site?

A very similar process will be followed for the administration controls so that we have feedback for both sides of the web portal. Once we gather all of these surveys, they will be carefully reviewed one by one and a list of improvements that should be focused on will be compliled. After that, we will follow up with the client to display the revised web portal and ensure we did not miss any crucial interface improvements. This usability testing will help to ensure maximum client satisfaction with the entire web portal.

Conclusion

Throughout this software testing plan, we have detailed the ideas and roadmap for testing and monitoring the performance of our software tests and provide in-depth examples of the tests that are to be performed. Together as a team, we broke down the functionality of our application developed to help our client, Dr. Zachary Lerner and ensure that the teaching unit of code performs as expected while also remaining reliable and sturdy. Every single one of our tests exists to uncover potential unexpected behavior and errors that other types of tests can not easily detect, this will again ensure that all ground is covered and verify the validity of our code.

Unit tests ensure that each piece of code is working as expected. This helps us find errors in individual components by being able to break down our code and test it part by part. We break down our code and then test it for a variety of conditions, whether that be correct output or input, or correct calculations occurring inside the code itself. Furthermore, Integration testing then ensures that components are working together as a whole and communicating correctly with one another, this is done by creating checkpoints along the way, and ensuring that those checkpoints are meeting the conditions that we require. Now onwards to our usability testing, which is key to making sure that our end users can provide feedback on various aspects of our application that need to be improved through their eyes.

Using the results of our testing (Unit testing, usability testing, and integration testing), we will ensure that our product is up to par. Given the introduction of unforeseen problems, we will then make changes to remedy these issues and improve the software quality and user experience. With our rigorous testing, we will be sure that our software functions work cohesively and more importantly our web application is working correctly and easy to use. This will allow future clinics to understand their information in an immediate and proper manner.

References

[1] Centers for Disease Control and Prevention. (2020, December 31). Data and statistics for Cerebral Palsy. Centers for Disease Control and Prevention. Retrieved November 5, 2021, from https://www.cdc.gov/ncbddd/cp/data.html.

[2] Science & Outcomes. Biomotum. (n.d.). Retrieved November 5, 2021, from https://www.biomotum.com/science.