The Test Plan Document

CS 300 - Group Project (Team 1)

Bader Alshaya

Boodhi Alshaikh

Dhwanil Chokshi

Zack Salah

**Table of Contents**

***1******Introduction***

*1.1 Purpose and Scope*

*1.2 Target Audience*

*1.3 Terms and Definitions*

***2******Test Plan Description***

*2.1 Scope of Testing*

*2.2 Testing Schedule*

*2.3 Release Criteria*

***3******Unit Testing***

*3.1 Unit test cases*

*3.1.1 Case one -*  Terminal Functionality

*3.1.2 Case two - Services Assistant*

*3.1.3 Case three - Form Generation*

*3.2 Strategy*

*3.2.1 Execution Testing*

*3.2.1.1 Branch Testing*

*3.2.2 Mutation Testing*

*3.2.2.1 Value Mutation Testing*

*3.3 Code Coverage*

*3.4 Unit vs Integration Testing*

*3.5 Excluded Units*

***4******Smoke Testing***

*4.1 Terminal Functionality*

*4.1.1 - Terminal Access*

*4.1.2 - Unexpected Input*

*4.1.3 - Exit Functionality*

*4.2 Service Assistance*

*4.2.1 - Member Validation*

*4.2.2 - Disk Record Saving*

*4.2.3 - Provider Verification Form*

*4.3 Form Generation*

*4.3.1 - Forms*

*4.3.2 - Form Access*

***5******System Testing***

*5.1 Functional Testing*

*5.2 Structure Testing*

*5.3 Performance Testing*

# Introduction

The test plan document is an important plan that marks the final phases of the development process. It discuss in further details the specific testing methodologies that our development team will use as part of the ChocAn project. In this regard, it is necessary that different types of the testing methods are used accurately to ensure that the customer is receiving a well structured product. This document covers several known test including the unit, smoke, and systems tests in addition to the their unit components that defines each type. These different tests are designed so we can collect more information about the anomalies, errors, or information about the program’s functional and nonfunctional attributes. In general, this document is aimed to enhance the sustainability and efficiency of the software.

* 1. **Purpose and Scope**

Since the test plan document layouts the fundamental structure of the testing procedure which is used by the team of development during the process of implementing the ChocAn software, it follows that the such process aims more into the technical part of the software. This is to define how exactly units within the system ought to interact with each other as defined by the ChocAn’s software design with respect to monitoring and securing the different interactions during and after the implementation of the software. This is accomplished by adhering to good testing procedures that helps reducing the number of future bugs as much as possible.

Therefore, one of the main reasons this document exists is to establish the essential platform to detect bugs and errors before writing any code, since these bugs might hinder the flow of the program and have various undefined impact during the execution of the software or even after. On the other hand, testing is an important part of the verification and validation stages because it helps development team understand the functionality of the source code and how it would handle errors without breaking the whole system. A lot of the testing processes, which are further discussed on the following sections below, will use specific testing methods which in turn will lead towards enhancing the efficiency and the overall security of the ChocAn system by overloading artificial data under reasonable and related specifications.

* 1. **Target Audience**

This document is designed as an illustration of the different phases and testing processes that will take place during the development and the implementation process of the software. As mentioned previously, this document offers a feasible environment and the best methodology that aids the process of deducting all notable errors beforehand and, thus, eliminate a huge portion of trivial bugs way before the deployment of the finalized version of the ChocAn’s software. Henceforth, it should be clear that this document is tailored to the development team and more specifically the head team responsible of the testing procedure. This is due to its technical specification that can be well understood only by expert developers who have a wide experience in the field and know how to conduct such different testing procedures as needed.

* 1. **Terms and Definitions**

The following chart is meant as a short and quick reference guide for the development team to have all the new and vital terminologies along with their associated definitions listed on the top of this document.

|  |  |
| --- | --- |
| **Terms** | **Definition** |
| Unit Testing | A separate test cases applied to any functional source code of the overall software |
| Smoke Testing | A test for the main functionality of the software to verify the stability of these functions and decide a suitable related action |
| System Testing | A test for the entire system of the software to validate the different interactions between the different units of the software |
| Fault Tolerance | Property that helps the software continue functioning in the event of a failure |
| Maximum Fault Tolerance | Is the maximum possible failure handling that can be reached at any giving time without having an effect on the functionality |
| Terminal Access | Gaining access to the terminal by using ID |
| Exit Functionality | Going back to the terminal sign-in page after finishing the task |
| Member Validation | Seeing if the member exists in the system as a valid or invalid member |
| Disk Record Saving | Saving the service record into a file |
| Provider Verification Form | Form which verify the services provided in the event that information is lost |
| Forms | Member and Provider forms with information about services |
| Form Access | Being able to access individual provider forms |
| Volume Testing | Testing the software with a certain amount of data |
| Timing testing | The time functions take to finish their tasks |
| Recovery testing | Testing how well the software is able to recover from failure |
| Accepting testing | Test conducted to see if the system meets the requirement specification |
| Security Testing | Reveal bugs and flaws in the security mechanism of software that could harm the data |

# Test Plan Description

The test plan defines the procedure the development team will undergo to write a software that is more of a “bug-free” than a functioning software full of unhandled bugs. Hence, this plan controls the overall flow of the ChocAn’s software and increase the sustainability of the whole system. To go more in depth into the description of the different steps and phases this plan will cover, consider the following sections.

* 1. **Scope of Testing**

In this software project, we will be using three main testing procedure to control the software as a system and all the in-between interactions that occur during the run-time of the finalized ChocAn’s software. Namely, a unit testing will be the first test to be conducted for the software. Each separate unit or functionality of the software will be tested using the test unit approach that is explained thoroughly in Section 3 of this document. The second testing procedure is called smoke testing. This testing procedure can be considered as the “firewall” of the software. It tests the main functionality or the most valuable unit within the whole system of the software during any changes or erroneous inputs. This allows the whole system to work safely and prevent it from failure or crashes that might result in a long lasting effect on the software, the customer, and the type of product it serves. Finally, the final testing procedure is commonly known as the system testing. This testing usually become so handy at the final stages of the software creation time. It ensures a proper functioning and safer interactions between the different units of functional source code and helps to create an environment that is more prone to future changes and updates.

* 1. **Testing Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Developer(s)** | **Starting Time** | **Finishing Time** |
| **Unit Testing** | Zack and Bader | November 14, 2017 | November 21, 2017 |
| **Smoke Testing** | Dhwanil | November 22, 2017 | November 25, 2017 |
| **System Testing** | Boodhi | November 25, 2017 | November 30, 2017 |

* 1. **Release Criteria**

Fault tolerance is the property that allows the system of the software, ChocAn, to remain functional at the occurrence of one or more unit failure(s) during the runtime of the software. Accordingly, when a maximum fault tolerance gets implemented correctly within the system, the overall software will be granted a more flexible and sustained functionality that does not intercept the process of an ongoing system. On the other hand, without this property, the system is considered more “fragile” and more sensitive to minor bugs and errors that might cause massive impact to the entire system, which in some cases, irreparable.

The criteria that this system must meet before the deployment of the software reveals as follow. First, the software must meet the specified requirements of the customer as stated in the requirements document. Second, the software must adhere to the design specifications agreed upon within the design document. Third, the system must be cohesive and well defined using proper coding techniques. Moreover, the system must follow the testing policies and requirements discussed in this document to ensure a secure and sustain software that functions properly during and after the execution of the software. Finally, the tests implemented must be written in such manner that it becomes more flexible in terms of future updates and changes.

# Unit Testing

Unit testing is used to test individual components of the software. It tests for syntax errors, logic errors, assignments, or functions. Unit testing is language specific and it tests the correctness of each unit before integrating. This allows the software engineer to find bugs that other type of test wouldn't. Unit testing will be automated.

This section will introduce the detailed description of the strategies of unit testing. This involves the white box. Additionally, the section will include the chunks of the software that will be unit tested from the source code and descriptions on why these same programs will be unit tested. Similarly, it will state the reason for why some part is not unit tested. Furthermore, It will cover items like code coverage and difference between unit and integration testing.

* 1. **Unit Testing Cases**

The software will contain three terminals. These terminal are the heart of the software. Each terminal is designed specifically for a designated user. Their terminals are provider terminal, manager terminal, an operator terminal. Each terminal needs an access key in order to log in.

The provider terminal is used to provide a service to ChaoAn members. The provider must ask the member for their card in order to provide a service to the member. The provider can also access the provider directory to look for a service name and code via the terminal. After a service is provided to the member, the provider needs to fill out two forms. The first form is the disk record and the provider verification form.

The manager terminal is one of the most important parts of the software in regards to stakeholders. The purpose of the terminal is to ensure that every provider is paid. The terminal is used to view The EFT report, view individual reports. The manager terminal is also responsible for generating a summary report. The functionality of the Manager terminal is very limited.

Additionally, The Operator Terminal is essential to part of the software. Its purpose is to add, delete, or update date a particular report, members, or providers. These operators will produce a file that will contain new newly added members and resigned member as providers to send to Acme Accounting services. However, updates of to any individual are updated in ChocAn Data Center.

The units that we will be testing from the terminals will be sectioned into three cases:

* + 1. **Case One - Terminal Functionality:**
* Access:

The terminal access will require a key in order to login. We will be unit testing this functionality. Will will generate mock data with its expected return value and run it through Function.

* Input/ output validation:

Providers, managers, and operators will need to access the menu items by entering keys associated with the menu item. We use Branch testing for every menu item. Will also test entries that are not on the menu to ensure that invalid entries are handled.

* Exit functionality

After the providers, managers, operators are done with the terminal, the terminal is supposed to exit without any difficulty. A unit test can help us identify these errors with existing the terminal.

* + 1. **Case Two - Services Assistant:**
* Validity of Members

When providers provide a service for a member. They need to enter the member number in order to do so. Unit testing will be tested using the mocked data. This allows us to ensure that all mocked number that should be accepted get “Valid” and all numbers that are not in not registered in the mocked will show “Invalid”. Numbers with balance will show suspended.

* Disk record

When a provider provides a service to a member. The need to fill out a disk record. This record will be store in a file and a data structure. The Unit test will ensure that files are correctly generated and it makes sure that each disc record is stored in a data structure.

* Provider form

After the providers finish with the disk record, they need to will out another from, which is the provider form. The unit test will be testing if the enter files are not empty. Also, it checks if the date of disk record matches the provider from’s.

* + 1. **Case three - Form generation:**
* Forms

There are multiple forms that will be generated by the software. We will be testing this by making sure that with each generation, a text file is formed. The content, however, it cannot be unit tested.

* Form access

Once the reports are generated, the manager can view individual files if need be.Unit testing can test this by mocking the report and make sure that the view function works properly.

* 1. **Strategy**

White Box Testing will involve testing the inputs and outputs by mocking inputs data and comparing it against its expected outputs. This section does not test for functionality of the software or the requirements. White Box Testing only involves testing the logic and it improves the design and usability of the code. Its purpose is to check for internal security of the code, poorly structured paths in the coding process, the flow of specified input and its expected output, and conditional loops. White Box Testing tests can be scoped to a statement, a function, or an object. These are tested individually.

* + 1. **Execution Testing**

Test designers will be implementing two strategies of White Box Testing. The first testing method will be the Execution Testing.This allows us to test conditional statements, Loop validity, and Expected outputs on some given inputs. This ensures us that the logic and the syntax of the code is well written and structured. Moreover, it ensures that every path in conditional statements is tested and execute as intended. Lastly, it ensures that every input will output the corresponding output.

* + - 1. **Branch Testing**

Test designers will be applying one of the Executive Testing strategies. They will be implementing the Branch Testing strategy. This method will help to test each possible condition at least once and executed at least once. Furthermore, it ensures and tests that each branch does not have any flaws, it ensures that branches are executed properly. Branch Testing eliminates the possibility of having problems with conditional statements.

* + 1. **Mutation Testing**

Other Testing methods, that the designers will implement is the mutation testing. This will allow us to find more problematic errors that might occur in the software. When it comes to hidden imperfections, Mutation Testing is one of the most effective ways to recognize them. Mutation Testing will simplify maintenance of the software.

* + - 1. **Value Mutation Testing**

The test designers will implement one of Mutation Testing strategies. They will implement the Value Mutation Testing. This will allow us to find hidden errors by manipulating the parameter. This can be done by increasing or decreasing the parameter values, swapping the parameters values, set the then to specific values.

* 1. **Code Coverage**

The programmers will be implementing the ChocAn Data Center software using C++. Thus, we will be using Gcov as a code coverage. This will ensure us that our unit test implementation covers the needed lines. Gcov will generate a file with the covered lines and the percentage of code that is cover from the overall source code. The code coverage number will be 70 percent.

* 1. **Unit vs Integration Testing**

Unit Testing test on a component of the software at a time. The connection between the component must be mocked to fully test it. Unit testing is focused on an individual function, conditional statements, or class object. A unit test is fast and it can find errors other tests such as integration test will not. It will show where exactly their errors are. This type of testing happens first and it’s conducted by the developer

On the other hand, Integration testing is used to test multiple components. The connection between them can be mocked if needed, but it's usually not. Integration testing errors are harder to define. It's very slow compared to unit testing. Integration testing happens after unit testing and before system testing. This will be carried out by the testing team.

* 1. **Excluded Units**
* storing (Data structure)

Data structure will be difficult to test by Unit Testing

* File Generation

contents of files can not be tested.

* Updating

Data structure will be difficult to test by Unit Testing. Also, contents of files can not be tested.

* Appending to file

contents of files can not be tested.

# Smoke Testing

Smoke testing (aka Build Verification Testing) is a subset of software testing, which can allow or disallow testers from moving forward with additional testing. This test is done on the most important functions to check that they work and we can move forward with other types of tests on the software. If the important software product does not pass smoke test, then the necessary changes need to be made to the software with an updated build. With data processing software like the one being produced for ChocAn, we have to ensure that the system has working functionality so it doesn’t break on simple user errors. Below are the test cases on which we will test the functionality of ChocAn’s software.

* 1. **Test Case 1 - Terminal Functionality**

The terminal is an important aspect of this program, giving the proper accessibility to the correct individual so data and private information is not jeopardized. The provider’s terminal is important because it is used to check validity and give service to members. The manager’s terminal is important because the manager should be able to access individual provider forms. The operator’s terminal is essential to ensure that they are able to delete, add, adjust member and provider information as necessary. Therefore testing these three terminals for full functionality is essential to the operations of ChocAn.

4.1.1

Smoke Test Element #1: **Terminal Access**: When the provider, manager, or operator use the software, they their respective identification number which they have to provide to gain access to the terminal. These ID numbers ensure data protection and confidentiality which are important characteristics of a system that stores private information about individuals. Therefore to test the functionality of this component of software, we will type in the correct and incorrect identification numbers in all terminals. The process will be repeated more than once. Entering correct identification numbers should give us access to the respective terminal, guaranteeing that the provider, manager, or operator has been identified in the system as a valid staff member. Entering incorrect identification should not allow access, informing us that the number entered does not match a valid staff member. However, since errors are part of human nature, the software should not cease execution when incorrect information is entered. There should be a follow-up prompt that informs the user that incorrect information has been entered and they should try again.

4.1.2

Smoke Test Element #2: **Unexpected Input**: To label the software as “usable”, testing has to ensure that all aspects of a component are working. Therefore, when prompting the provider, manager, and operator to enter their ID for verification, we should handle cases where the input is unexpected. The user could have misunderstood the prompt and may enter information which is totally different from what the software is asking. We cannot simply assume the input will be all numbers and only handle cases for numbers. The user can misunderstand the software instruction and enter alphabets, symbols, or other alphanumeric characters which we have to take into account. If we handle only the cases dealing with numbers, the system will break if letters are entered. The test case does not simply end here. If an unverified user is using the system, they could potentially enter something which can overflow the buffer. This action could be hazardous for ChocAn since their system can be broken by entering a long string. To guarantee protection from buffer overflow, the software has to ignore characters after certain point, or force user to type within certain parameters.

4.1.3

Smoke Test Element #3: **Exit Functionality**: Once the use of terminal is finished by the user, it should swiftly exit the interface and switch back to it’s original page waiting for a provider, manager, or operator to sign in. This is important because the provider page should not, under any circumstance stay open once the service has been provided. This could be potentially risky as any person could come to the terminal and use it without proper verification. Furthermore, operator’s terminal has the ability to add, delete, and adjust information, so it is very vital that we limit access to the verified user. Program exit should always work without depending on another software component.

* 1. **Test Case 2 - Service Assistance**

An important part of ChocAn’s software is the “meat” of its purpose; providing assistance. Therefore, without the proper functionality of this component, the software is rendered useless. Requirements specification emphasize the importance of providing service to the valid members and recording the service information that was provided by that specific provider. This information will be used later when we are generating forms for users and providers.

4.2.1

Smoke Test Element #1: **Member Validation**: Once provider gains access to their terminal, they should be able to provide a service. For a service to be provided to a member, we have to make sure that the member is a valid member in the system. Therefore, member validation should have proper functionality. When a provider slides a member card, or enters a member number, the software checks the members number and displays whether that member is valid or invalid. When testing, valid member names will be entered to check if they show up as valid. Invalid member names will be entered to show that they are invalid or not recognized. This test should be passed in order to move forward. Additionally, unrecognized input and buffer overflow should not break the checking functionality, since we are not allowed to assume that only preferred input will be entered.

4.2.2

Smoke Test Element #2: **Disk Record Saving**: After the service has been offered, the provider looks up the service code using the provider’s directory. When the correct service name has been typed, the service pops up with it’s respective service code. Upon keying in the service code, the software should display the service that is associated with that code. The testing will be done to handle service look-ups using names corresponding to it’s match in the provider’s directory, and names that do not match. The ones that match should display the service code and ones that don’t match should display an error message. The provider enters information to be saved into the disk record. This information includes current date/time, date service was provided, provider number, member number, service code, and comments. This will be saved in the BST and written out to file. The testing on this functionality will be done by entering the information from user’s end and seeing that the information is saved into a centralized text file that stores all information about service provided by providers.

4.2.3

Smoke Test Element #3: **Provider Verification Form**: The provider verification form is essential to keeping track of the services that each provider has provided to members including the fee of the service. To test for verification form functionality, we will simulate production of information from user’s end by entering correct information about service and member into the verification form. The verification form should be saved into the text file under the provider’s name. This form keeps track of all services that have been provided incase there is system failure. The verification form is taking user input. Therefore, we have to ensure that the information entered is valid. When the service is looked up using the service code, the service service name and it’s fee should be shifted onto the verification form. The provider should not type in this information manually since it is sensitive and easily be manipulated if typed. Testing on this component will be done to ensure that service code, fee are saved by software, while the member name and number is typed by provider.

* 1. **Test Case 3 - Form Generation**

Generating provider and member forms to be sent out once a week is an important part

that needs to be implemented correctly. The requirement specifications state that forms are to be sent out once a week to the members and providers with respective information about services provided and used during that week. The functionality of the form is important to permit the flow of data processing software which works to benefit the ChocAn staff.

4.3.1

Smoke Testing Element #1: **Forms**: The provider and member forms should be generated with the inclusion of all details that the form asks for. This will be tested by simulating different users going to different providers for service and having their forms be generated and stored as a text file. To pass this test, the member form should have the correct member information stored in the text file and the provider form should have the correct provider information stored in their text file. When the information is being written to the disk record and verification form, it is also being stored in the BST data structure for providers and members form to ensure that the generated reports contain all information. To check the functionality of this component, we are going to run a test where 5 validated members are provided service from one provider. This should generate one form for each member and five different forms for the same provider. Additionally, we will test using 5 validated members receiving service from 5 different providers. This should generate one form for each member and one form for each of the 5 providers who provided a service to their unique member. Lastly, we will simulate a run which will have 5 providers providing a service to one user. That user should have 5 forms from 5 different providers and each of the 5 provider should have a form for the service they provided to the same user.

4.3.2

Smoke Testing Element #2: **Form Access**: Individual form access can only be done by managers from manager terminal. Therefore, testing methodologies will check that there is no form access from provider’s terminal. The manager’s terminal should be able to access provider records of a certain provider. Testing on this component will be done by accessing manager terminal and searching for provider forms under a provider’s name. This should bring up all the forms from that provider for that week and in the past. When searching for provider’s name, we will also have to enter their number. If their number matches the number in the system under providers, then we display the forms. If there is no match, there should be an error message stating that provider was not found and the software will prompt again.

# System Testing

In this section, we will test if the system’s features works as it supposed to be working. The system testing is the last testing after the systems has been developed which falls under the black-box testing. We will be doing three main types of testing which are Functional Testing, Structure Testing and Performance Testing. We will check out the main functions and the functionality of our system. We will make sure that all exposed bugs are corrected. Also, we will check memory leaks in many ways to make sure we don’t lose data in our system. Every data structure will be check when adding or deleting to make sure no memory leaks occur. We will check how the system handles large amount of data and if it would break if the system received unexpected amount of data. In the end, we will make sure that this application is created to meet the requirements of ChocAn’s and every component and subcomponent are created as they customer wanted and by doing that we will verify if our system is fixed for release.

## Functional Testing

For the functional testing, we will check the validation of the application and if the application’s functionality meets the ChocAn’s specifications. We will check each component of our system and if every component and subcomponent is behaving how it should be behaving.

* Terminals

We will check if our system operates as it should be operating by checking if every terminal can be accessed by the user’s ID number. When a manager logs in, a manager should be accessed not operator’s or providers’ terminal. A manager's terminal should have access to view the summary reports but not edit them. The summary report should show the providers to be paid for the week, the number of consultations each provider had and the provider’s total fee for the week. Also, the manager terminal should have access to any of the provider’s forms in read only mode, so it can’t be edited as well. On the other hand, the operator’s terminal must have access to all data saved in the system. The operator should be allowed to add new members or providers, edit existing members or providers and delete members who canceled their memberships or delete providers from the provider’s and the member’s forms. The provider's terminal should be able to add service to members and check the validation of the members, but not edit or delete existing data.

* Service assistance

For this section, we will check if our system would allow the providers to offer services to members.the provider should be able to access the Provider’s Directory and choose one of the services that will be displayed in alphabetical order to add it to the member’s form and the disk record. Also, the provider should also be able to save data to the disk record which will have the member’s number, the service number, the provider’s number and the date and time.after the provider finds the service a fee for the service should be displayed in the terminal. Furthermore, the provider must check if the member’s status is valid in order to offer the member a service. After provider enters the member’s number, a message must be displayed that says the status of the member which valid, invalid or suspended. The member’s status is going to be taking care of by a third party company. Moreover, a form will be prompt to the provider for verifications purposes and the provider must enter the current date and time, the date the service was provided, member’s name and number, service code and the fee to be paid.

## Structure Testing

The structure of the system will rely on external data files which the system will read from and write into them. The member’s forms are going to be written into a text a file along with the provider’s forms and the disk record and the summary reports.

* The Disk Record

The disk record should be written by the provider after confirming a service to a member and it should have the following:

* Current date and time.
* Date the service was provided.
* Provider’s number.
* Member’s number.
* Service code.
* The Summary Report

The summary report should be generated by the system for the ChocAn’s managers and should be only accessible through the manager’s terminal. The summary report should contain these following information:

* List of providers to be paid that week.
* The number of consultations each provider had.
* The provider’s total fee to be paid for that week.
* The total number of providers who provided services.
* The total number of consultations for all providers.
* The overall fee total.
* Forms Generation

The member's forms will be accessed by the operators and the providers by their terminals. The provider will be able to add members forms after offering a service to a member. These forms should be saved into an external file so it can be accessed by the operators. The operators should have the authorization to edit these forms, delete these forms or add to these forms. The member’s form should have these data:

* Member name.
* Member number.
* Member address.
* Member city.
* Member state.
* Member zip code.

And for each service provider the following should be there:

* Date of service.
* Provider name.
* Service name.

On the other hand, the provider’s form should have the following data:

* Provider name.
* Provider number.
* Provider address.
* Provider city.
* Provider state.
* Provider zip code.
* Total number of consultations.
* Total fee for the week.

And for each service the following data:

* Date of service.
* Date and time data were received by the system.
* Member name.
* Member number.
* Service code.
* Fee to be paid.

## Performance Testing

* Volume Testing

For volume testing, we are going to read from a file a lot of provider’s forms and member’s forms, so that we see how the system is going to deal with a huge number of data. We will make sure that our system can deal with high volume of data at once.

* Timing testing

For timing testing, we are going to record the time for each operation in the system like adding a provider’s form or retrieving information from the provider’s directory. We will evaluate each main function and check if we chose the write data structures for the system.

* Recovery testing

For recovery testing, we are going to check how the system reacts to illegal operations like displaying provider’s forms if there was no data or deleting member’s forms if there was nothing to delete.

* Accepting testing

The purpose of this test is to make sure the system is ready for operational use. ChocAn will be the party who will be performing the acceptance testing by running the application and make sure it meets ChocoAn’s requirements.

* Security Testing

For security testing, we are going to try violating and manipulate security requirements like accessing any of the terminals with an invalid id number and see if we would have access to the protected data.