**Quality Assurance Test Plan**

|  |  |
| --- | --- |
| **Name**  **WGU Student ID** | Kendall Freed  010213248 |

# A. Overview

## 1. Software design plan summary: My software design plan for Endothon Finance was created to address the issue in their web application wherein the first five fiscal years of a business were shown when the previous five fiscal years should have been requested and displayed in the loan application. In this plan, I addressed reviewing the logic to ensure the last five years of financial data would be returned for business applicants with five or more years of data to provide, reviewing the logic of businesses that were younger than five years, and ensuring the forecasted fields appeared for them and finally displaying the fields in a more readable way by updating the placeholder text and including a header for forecasted data.

## 2. Functional requirements objective: The objective of the functional requirements to be tested during the quality assurance process is to ensure the logic is fixed and the application collects the correct information from the loan applicants. At the end of the process, the application will return the last five years of financial data for business applicants with five or more years of data to provide. It should also return five years of financial data for business applicants who have been in business for less than five years by returning the forecasted data for years to come to a total of five years. Additionally, the loan application fields should now be more readable to ensure the applicants can understand what data they need to enter.

### 2a. Functional requirements objective metrics:

## Functional Accuracy: The application should be accurate for every single test case. 100% of test cases (retrieving and displaying the correct years) should be passed without errors. This would ensure the loan application logic correctly identifies and shows the accurate financial data fields based on the business establishment year and the current year. It would fix the ticket and allow the web application to work again as intended.

## Readability: The number of errors created when entering financial data into fields should be less than 5% for every 100 fields entered. Every time a user attempts to enter something in a field incorrectly, it will produce an error. Ideally, this metric strongly indicates how intuitive the loan application readability is.

## 3. Non-functional requirements objective: The objective of the non-functional requirements is to ensure the supporting software and supporting aspects are functional and compliant with regulations. For this software design plan, we have identified database security and internet browser compatibility as the main non-functional requirements to be tested.

### 3a. Non-functional requirements objective metrics:

### Database Security: The database should have no unnecessary users accessing the data. One primary requirement for the database to comply with SOX is that all users must be monitored for correct access restrictions. This will add security to the data and allow the database to pass internal and external SOX audits. The metric would be a failure if there are any users with access who should not be on the list.

### Browser Compatibility: The application should be compatible with over 90% of user devices and browsers. The application will be run through the software BrowserStack to test Core features (all loan application fields working correctly) and visual consistency across all browsers. This will ensure that users on almost any device can access and use the application.

# B. Scope

## 1. In-scope functional requirements1.

## Functional Accuracy: This test will determine if the logic is working correctly. To do so, we will write unit tests with various years (older than five years, younger than five years, created this year, made in the future). The correct fields will appear on the user interface for each valid year and be saved correctly. For any year after the current year, it should display an error message, not show any fields for the user to enter, and not allow the loan application to be submitted. These tests should all pass for the metric.

## Readability: This test will show the number of errors created when users enter data into financial fields. To do this, we will add to a new counter variable for each error encountered when a user incorrectly enters data into a field. A loan application counter variable will also be created within the same function and added to when each loan application is filled out. Doing this will allow us to see the percentage of errors made and give us an idea of how readable the loan application is. The test should have a rate of errors of less than five to be considered a pass.

## 2. In-scope non-functional requirements

## Database Security: This test pulls up the users in MySQL and compares them to an internal list of users that should have access to the database. If there are any discrepancies, it is an automatic failure.

## Browser Compatibility: The test will be run through the BrowserStack program. This program takes the application and tests its compatibility across over two thousand devices and browsers. This test will be run with every update of the application to ensure compatibility. The metric we have is to be compatible with more than 90% of devices.

## 3. Out-of-scope functionalities & explanation

## -Financial Data Validation: The financial data provided by the businesses are required for the loan application. However, it is up to Endothon Finance and the business applying to ensure the financial data is accurate and valid. This requires documents of old financial data for verification and should be reviewed by a human for validity. Because of the significance of the data influencing the loan, it should not be automated, or if it is, it should have a final review done by a human before the loan is granted.

## -Financial Data Forecasting: The forecasting of the financial data is influenced by many factors, including industry, marketing, current and future projects, season, etc. These are factors that each business has (or should be aware of) that influence its forecasting. It would require algorithms that are outside this project to determine a forecast.

# C. Test Strategy

## 1. Testing overview

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case Table** | | | | |
| **Test Type** | **Description of Test** | **Objective** | **Test Owner** | **Environment** |
| Unit Test | We will be testing 7 different years: current\_year (using Year.now().getValue() to set it to current year), current\_year + 5, current\_year - 10, current\_year – 4, current\_year – 3, current\_year – 2, and current\_year -1. For each test, we will have a counter for the number of fields displayed: forecasted\_field\_counter and occurred\_field\_counter. We will also have the error be tested as pass/fail if it occurs for the test of the current\_year + 5. For the other tests, the number of fields will be checked with the correct number, and a pass/fail result will be given if it is the same or different. Example for current\_year -1: if forecasted\_field\_counter == 4 && occurred\_field\_counter == 1 return true. | Functional Accuracy | QA Tester | Java |
| Unit Test | For each time an error is triggered in the code, we would also include a statement to add to a variable called errorTracker that is initialized at zero. This variable will save the amount to a text file at the end of each application. The text file can then be read with the number of lines equal to the number of applications that have been processed and the total number of errors that occurred. Dividing the number of applications with the number of errors would give us the result of the error % we are experiencing in our application process. We can compare this number with .05 for a pass if it is smaller than .05. | Readability | QA Tester | Java, Spring Boot, Notepad |
| Unit Test | Collect a list of who accesses the database (using query ‘SELECT user, host FROM mysql.user;’) and save it to a text file in a secure folder. This text file is then compared with the text file of the internal list of user access privileges approved by the network administrator. Using the Java function Files.mismatch(), which returns -1 if there is no mismatch in the content of the files, we can write a test to see if the output is -1, and this would return a pass. | Database Security | IT | MySQL, Java, Notepad |
| End-To-End Test | The web application is uploaded to the BrowserStack cloud platform, which tests the application across multiple browsers and devices with different operating systems. It provides various visuals and lists of which browsers/devices are compatible and which are not. | Browser Compatibility | QA Tester working with BrowserStack | Cloud Platform through AWS, which runs:  -Physical Android and iPhones.  -Browsers: Chrome, Firefox, Edge, Safar, Opera, Internet Explorer. |

## 2. Sequence of testing

## 1) Application Functional Accuracy: Since this is the reason for the ticket and the purpose of the application, it will be addressed first and as quickly as possible to ensure the application can run again.

## 2) Browser Compatibility: Any update, specifically something on the front end, should be run through the compatibility site to determine if any errors need to be addressed. This should be done before the first test is deployed and after it is written/completed. It will also need to be done after the third test, as that one is more front-end focused and can have more issues displaying on different browsers and devices.

## 3) Usability: The usability of the application will be updated third because fixing the ticket and getting the application working again is paramount. However, before it is deployed as an update, it must be run through the BrowserStack in step two to ensure it is compatible with multiple browsers and devices.

## 4) Database Security: Because we did not add any users to access the code or change the database, we can leave this test for last. It can be run on its regular schedule.