MOD003263

**ALIFA MUHAMMAD SHIDDE**

**2064018**

**SOFTWARE ENGINEERING**

Table of Contents

[INTRODUCTION 1](#_Toc153173989)

[THE TEAM 2](#_Toc153173990)

# INTRODUCTION

This report aims to offer a concise yet comprehensive understanding as an individual and as a team of the key aspects of the Software Engineering module. It offers a brief examination of the life cycle involved in the development of an application.

In the various sections of the report, critical components such as project analysis, software design, database implementation, and testing will be touched upon. These different parts contribute to a better understanding of the software development process.

While this report is not exhaustive, it serves to provide a foundational understanding of Software Engineering. The sections contained in this report have been organized to facilitate ease of navigation and identification for the area of interest.

Thank you for taking the time to consider this report. I trust that it will reflect a valuable understanding of Software Engineering.

## THE TEAM

The team consists of five students charged with the responsibility of reviewing the case study of the project and developing a solution. They are: Akayovwe Okugbe (2226221), George Olefir (2215336), Alifa Shidde Muhammad (2064018), Chris Lampart (2207397) and Abdullah Asif. To meet the overall project requirements as a team, different parts were assigned to each team member based on the user stories discussed in the following section. Each team member was given the responsibility to create designs specific to their part to ensure everyone got involved in all aspects of the development. The development documentation section the report discusses the role assigned to the author and the implementations.

# PROBLEM DEFINITION

***PROJECT OVERVIEW***

After carefully analysing the case study for this project as a team, we identified the following challenges:

* **Lack of Visibility and Relevant Information:** Assessing potential solutions for a client quickly is very difficult due to information being scattered currently, resulting in lost revenue.
* **Timely and Relevant Solution Ideas:** The consultancy’s ability to provide insightful and timely solution ideas is hindered due to the manual work required to gather and organize information, leading to lost opportunities.
* **Siloed Vendor Application Knowledge:** Information about vendor applications is cumbersome to identify, analyse, and compare different solution opportunities because they are stored in various places using various methods, resulting in lost time.

The Financial Services Technology Solutions Repository project aims to develop a centralized data repository and user-friendly interface to help financial services consultancies identify and select appropriate technology solutions for their clients. The repository will store information about various vendors and their products, making it easier to access, manage, and analyse solution options based on specific client needs. The goal is to streamline the process of identifying and selecting technology solutions for clients.

# PROJECT AIM

The main objective of the project is to develop a user-friendly and centralized data repository that stores information about different vendors and their products in the financial service industry. This repository will make it easier for the consultancy to access, manage, and analyse information, ultimately leading to better decision-making when selecting technology solutions for clients. In summary, the objectives are:

1. Create a centralized data repository for vendor and product information.
2. Develop a user-friendly interface for authorized users to enter, maintain and retrieve data.
3. Enable searching and filtering capabilities to find relevant vendor and product information.
4. Link to vendor website for quick access.
5. Provide the ability to open and view attached PDF documents associated with vendors and products.

# PROJECT SCOPE

The project scope includes the following components:

**Database Development -** Design and implement a relational database to store data about vendors and products. Develop database schemas to accommodate vendor details, product information, and document metadata.

**User Interface Development -** Create a web-based user interface using ASP.NET and C#. Design data entry forms for adding and updating vendor and product information. Implement search and filter functionality for data retrieval.

**Document Handling -** Implement functionality to attach and open PDF documents related to vendors and products.

**Security-** Implement user authentication and authorization mechanisms to secure the application.

**Testing and quality assurance -** Conduct thorough testing of the application, including unit testing, integration testing, and user acceptance testing. Identify and address bugs and issues to ensure a stable and reliable system.

**Maintenance -** Plan for ongoing maintenance and updates to keep the system current.

# SYSTEM REQUIREMENTS

1. Development Team: Developers with knowledge of C#, ASP.NET, and database design.
2. Database Management System.
3. Development Platform.
4. Design and Visual representation software.

# RESOURCES USED

The following are resources we have used as a team during the design, implementation, and overall production of this project:

1. Microsoft Visual Studio 2022, Version 17.7.6.
2. Microsoft SQL Server.
3. C# Programming language.
4. ASP.NET framework.
5. Microsoft Word, Version 2310.
6. Microsoft Edge, Version 119.0.2151.72.
7. Draw.io, Version 22.1.3.
8. Lucid Chart, Version 3.5.14.
9. Google Chrome, Version 119.0.6045.160.
10. Google Docs.
11. **GitHub, version 3.3.4.**

# **USER STORIES**

As a financial consultant, I want the application to provide a user-friendly and intuitive interface so that I can use and navigate the system efficiently.

As a financial consultant, I want to be able to log into the application securely so that I can access the system and carry out my tasks.

As a financial consultant, I want to be able to search for vendors and their products based on specific criteria (For example, name of vendor, category of product) so that I can easily and quickly find solutions that are relevant for my client.

As a financial consultant, I want to view all related information of a vendor and their products including any attached documents and contact details so that I can access how suitable they are for my clients.

As a financial consultant, I want to click on a link to a vendor's website directly from the application so that I can access additional information about their offerings.

As a financial consultant, I want the ability to add new vendors and their products to the system and update existing ones as they change over time so I can keep the database up to date with the most recent and accurate information.

As a financial consultant, I want to be able to attach PDF documents (Example, Products manual) to the records of vendors and products so that I can access relevant documentation in the application.

As an administrator, I want to be able to manage user accounts (create, update, delete) so that I can control who has access to the application.

As an administrator, I want the ability to manage roles and permissions for users so that I can control what actions different users can perform within the system.

# 1. WIREFRAMES

***Adding Vendor:***

The wireframe below represents a form dedicated to adding a vendor to Citi soft management system. The form provides a navigation option to the Vendor display page. The main section of the form includes the company name, website, number of employees, and established date. A button for adding is provided.

The wireframe focuses on a clear layout to demonstrate the process of adding a vendor.

A screenshot of a computer application

Description automatically generated

***Figure 9: add vendor form.***

***Vendor Display***

Figure 10 displays all the vendors saved into the system. There is also a scroll bar either to the right or left to show more information regarding a vendor. From this page, the user can navigate to the adding a new vendor to the system by clicking the “**add** **vendor**” button. The “**>>Back**” button is for navigating to the homepage. The user can also click on a particular vendor to update or delete. Furthermore, there is the ‘**link software to vendor’** drop down button, when it is clicked, the mini form appears where the consultant can input the company ID and the software ID and then click the ‘**link’** button to link the software with the vendor.

A screenshot of a computer

Description automatically generated

***Figure 10: display vendor page.***

***Vendor Update and Delete***

Figure 11 is for updating and deleting a vendor from the system. The form displays the initial information of the vendor for viewing, the user can delete by just clicking on the “**Delete**” button. Or update by clearing the available information and inserting a new one for updating. The “**>>Back**” button is for navigating to the vendor display page.

A screenshot of a computer screen

Description automatically generated

***Figure 11: update vendor form.***

***Adding a software***

Figure 12, where new software is added to the system. The form also includes attaching files. The user clicks the “**Browse**” button and attaches a PDF file in the label “**file path here**”. The **“>>Back**” button is for navigating to the previous page.

A screenshot of a software

Description automatically generated

***Figure 12: add software form.***

***Update/delete software***

Figure 13 is the software update form. It displays the vendor information for viewing, the user can delete the vendor by clicking the “**Delete**” button. The initial vendor information can also be cleared for inserting new data for updating.

A screenshot of a software

Description automatically generated

***Figure 13: update software form.***

***Software Display***

Figure 14 displays all the available software products available in the database. Assigned users can click on a particular vendor deleting or updating. The “Add software” button is for navigating the Add software form, while the “>>Back” button is for navigating to the homepage.

A screenshot of a computer

Description automatically generated

***Figure 14: Display software form.***

# DATABASE DESIGN

***15. EXTENDED ENTITY RELATIONSHIP DIAGRAM.***

A diagram of a data flow

Description automatically generated

***FIGURE 15: EXTENDED ENTITY RELATIONSHIP DIAGRAM***

***Description***

The diagram represents the data model for the proposed database design. The diagram entities include Company, Software, User, location, and business specifications, etc. The relationship between the entities is to understand the interaction among financial consultants' application systems. Each attribute in the above entities contains a data type which is used for storing each attribute in a database.

The user entity presents the user's information that is to be stored in the database. It includes attributes such as first name, last name, email, password, etc, it has the user ID as its primary key.

The **usercompany** is a linking entity between the **company** table and **user** table. It holds two foreign keys referencing **company** table and **user** table.

The **company** entity contains attributes such as the company name, company website, contact details etc. It also contains the reference number as the primary key.

The **software** entity encapsulates the software products offered by vendors. Attributes include software ID as the primary key, software name, type, and documents to be attached.

The **companysoftware** is a linking entity between the **company** table and **software** table. It carries the **ref\_no** and **softwareID** as foreign keys.

The **companylocation** is a linking entity between the company table and the location table. It holds the **Ref\_no** and **LocationID** as foreign keys, referencing the two tables.

The **location** entity stores the location of the company. It has **location** **ID** as primary. It also contains the address and the location by city and country.

The company specifications are held in the **businessspecifications** entity. It holds the **specification ID** as a primary key.

**companyBusinessSpecs** is also a linking entity between the **company** and **business specifications** table.

***Relationship:***

There is an established connection in the diagram between each entity. the relationship between the company and software signifies that a company may or may not contain a software product. However, each software at least belongs to a company.

the company at least holds one business specification, however, the business speciation may or may not be associated with a company.

The company has at least one location, and a location can or does not belong to a company.

There is an aggregate relationship between the user and the company. The user entity serves as the “**whole**” entity representing users’ interactions with the system. The user can be associated with one or more companies.

In conclusion, the EER diagram is well structured as it offers a database schema for managing companies and software. Each attribute is identified by a unique primary key ID, proper references are also made and are identified by a foreign key. The relationship ensures the efficiency of the database in the capture of valid and meaningful relationships between entities.

# DATABASE SCHEMA

|  |  |  |
| --- | --- | --- |
|  | ***USER*** |  |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| user\_id | int (10) | Primary key. Saves a unique identifier for each user. |
| firstname | varchar (50) | Stores the customer first name |
| lastname | varchar (50) | Stores the customer’s last name |
| email | varchar (100) | Stores the customer email address |
| password | varchar (200) | Stores the customer password in the hashed version |
| address | varchar (250) | Stores the street address of the customer |
| postcode | varchar (7) | Stores the postcode of the user |
| country | varchar (100) | Stores the given country of the user |
| mobile\_number | varchar (15) | Stores the customer mobile number |
| password | varchar (255) | Saves the customer password in the hashed version |
| role | varchar (50) | Stores the role associated with the given user |

|  |  |  |
| --- | --- | --- |
|  | ***COMPANY*** |  |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| Ref\_no | int (10) | Primary key. Saves a unique identifier for each company. |
| Company Name | varchar (100) | Stores the vendor(company) name |
| Company Website | varchar (100) | Stores the company website |
| Company Established | Int (4) | Stores the established date of the company |
| Contact No | varchar (12) | Stores the contact no of the company |
| No of Employees | varchar (20) | Stores the number of employees of the company |
| Internal Professional Service | varchar (3) | Stores the internal professional service of the company |
| Last Demo Date | varchar (50) | Stores the lates demo date |
| Last Review Date | varchar (50) | Stores the latest review date |
| Cloud | varchar (3) | Saves the cloud of the company |
| Additional Information | text | Stores the additional information for the company. |

|  |  |  |
| --- | --- | --- |
|  | ***UserCompany*** |  |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| user\_id | int (5) | Foreign key in the linking table. A unique identifier that references the **USER** entity |
| ref\_no | int (10) | Foreign Key that references the **COMPANY** entity |

|  |  |  |
| --- | --- | --- |
|  | ***SOFTWARE*** |  |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| Software ID | int (10) | Primary key. Saves a unique identifier for each software. |
| Software name | varchar (100) | Stores the software name |
| Software type | varchar (100) | Stores the type of software |
| description | text | Stores the description associated with software |
| Document to attach | varchar (3) | Stores the PDF attached files. |

|  |  |  |
| --- | --- | --- |
|  | ***COMPANY*** | ***SOFTWARE*** |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| Ref\_no | int (10) | Foreign key, referencing the **Ref\_No** in the company table. |
| Software ID | int (10) | Foreign key. referencing the **software ID** in the software table |

|  |  |  |
| --- | --- | --- |
|  | ***COMPANY*** | ***BUSINESS SPECS*** |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| Ref\_No | int (10) | Foreign key, referencing the **Ref\_No** in the company table |
| Specification ID | int (5) | Foreign key, referencing the **specification ID** in the business specification table |

|  |  |  |
| --- | --- | --- |
|  | ***BUSINESS*** | ***SPECIFICATION*** |
| ***ATTRIBUTES*** | ***TYPE*** | ***DESCRIPTION*** |
| Specification ID | int (5) | Primary key. Saves a unique identifier for each business specification. |
| Financial service types | varchar (100) | Stores the types of financial services |
| modules | varchar (100) | Stores the modules |
| Business area | varchar (100) | Stores the business area of the company |

# SOFTWARE DESIGN

## 16 USE CASE DIAGRAM

A diagram of a diagram

Description automatically generated

***FIGURE 16: USE CASE DIAGRAM***

## DESCRIPTION

|  |  |
| --- | --- |
| **USE CASE NAME** | **ADD NEW VENDOR** |
| **ACTOR** | **CONSULTANT** |
| FLOW OF EVENTS | 1. The user fills in the Company Name. 2. The user enters the company’s website. 3. The user fills in the number of employees. 4. The user fills in the established date. 5. The user clicks on the “Add button”. |
| ALTERNATIVE FLOWS | The user clicks on the “previous button” to go back to the display page and clicks on the “previous button” again to go back to the home page. |
| ENTRY CONDITIONS | The user is on the add vendor page. |
| EXIT CONDITIONS | The user clicks on the “>>>” button to go to the display page if he decides not to continue adding vendors.  Display vendor page automatically shows after adding a vendor. |

|  |  |
| --- | --- |
| **USE CASE NAME** | **ADD NEW PRODUCT** |
| **ACTOR** | **CONSULTANT** |
| FLOW OF EVENTS | 1. User fills in the Software Name. 2. The user fills in the Software Description. 3. The user clicks on the “Add button”. |
| ALTERNATIVE FLOWS | The user clicks on the “previous button” to go back to the display page and clicks on the “previous button” again to go back to the home page. |
| ENTRY CONDITIONS | The user is on the Add Software page. |
| EXIT CONDITION | The user clicks on the “>>>” button to go to the display page if he decides not to continue adding software.  Display software page automatically shows after adding software. |

|  |  |
| --- | --- |
| **USE CASE NAME** | **UPDATE VENDOR INFORMATION** |
| **ACTOR** | **CONSULTANT** |
| FLOW OF EVENTS | 1. The user clicks on the vendor. 2. The available information of the vendor is shown. 3. The user clicks on the vendor. 4. The user changes the information and clicks the update button. |
| ALTERNATIVE FLOWS | The user clicks on the “previous button” to go back to the display page and clicks on the “previous button” again to go back to the home page. |
| ENTRY CONDITIONS | The user is on the update vendor page. |
| EXIT CONDITIONS | The user clicks on the “>>>” button to go to the display page if he decides not to continue his update.  Display software/vendor page automatically shows after the update. |

|  |  |
| --- | --- |
| **USE CASE NAME** | **VIEW ATTACHED FILES** |
| **ACTOR** | **CONSULTANT** |
| FLOW OF EVENTS | 1. The user clicks on a cell in the software display grid view. 2. The update form is made visible with the software information for viewing. 3. The user clicks on the file path label. 4. The PDF file is opened. |
| ALTERNATIVE FLOWS | The user exists the file to go back to the update form. |
| ENTRY CONDITIONS | The user is on the update vendor page. |
| EXIT CONDITIONS | The user clicks on the cancel icon to go back to the update form. |

# CLASS DIAGRAM

The diagram below presents the class diagram. It is created while writing the user stories before its implementation on Visual Studio. It contains 7 backend classes.

The classes include:

Forms 1: this is where all the button clicks are located. In our program, we have used a single Windows form, each team member used panels to design his assigned role.

DBConnection: this class stores methods used in connecting to our project's database.

SqlQueries: this class store all the queries used in interacting with the database.

SOFTWARE: This class contains two static methods that facilitate the update and deletion of software in the database. The two methods are **changeSoftware** () and **eraseSoftware** ().

VENDOR: this class contains two static methods that facilitate the update and deletion of a vendor in the database. The two methods are **changeVendor** () and **eraseVendor** (`).

Admin: this class contains the methods that facilitate the tasks related to the admin.

password Hasher: this class encrypts the passwords before inserting them into the database.

The diagram offers a clear overview of the system’s structure, enhancing understanding and guidance during the development process.

A diagram of a software company

Description automatically generated

***FIGURE 17: CLASS DIAGRAM***

# SEQUENCE DIAGRAM

***Adding Vendor:***

the diagram presents the actions taken while adding a new vendor to the system:

USER STORY: As a financial consultant, I want the ability to add new vendors to the system so I can keep the database up to date with the most recent information.

The user inserts the company’s(vendor) details such as the company name, the company’s website, the number of employees, and the date the company was established.

The form validates the details.

If the details are valid, the system calls the query method that triggers the system to insert the company’s details in the database, and a confirmation message appears.

If the validation fails, the system displays a message to the user informing him about the validation issue.

A diagram of a company

Description automatically generated

***FIGURE 20: ADD VENDOR SEQUENCE***

***Updating Vendor***

the diagram shows the process taken when updating software information:

user story: As a financial consultant, I want to be able to update vendor information as it changes over time so that I can always maintain accurate details.

The user clicks on a **vendor** on the vendor display page.

The system opens the update page with vendor details being displayed (retrieved and displayed).

The user clears(modifies) and inserts his new update. The system then validated the modified vendor.

If the validation is successful, update will be inserted to the database.

If validation fails, the system throws exception.

A diagram of a process

Description automatically generated  
 ***FIGURE 21: UPDATE VENDOR SEQUENCE***

***DELETE VENDOR***

The user chooses a vendor in the vendor display page and clicks on the vendor.

The vendor data is retrieved, and the update page opens.

The vendor details are displayed on the update page.

The user attempts to delete vendor.

If vendor is deleted successfully, the system displays a confirmation message to confirm the deletion. However, if vendor fails to delete, the system throws an exception.

A diagram of a software development

Description automatically generated  
 ***FIGURE 22: DELETE VENDOR SEQUENCE***

***Adding Software***

the diagram below presents the actions taken while adding new software to the system:

User story: As a financial consultant, I want the ability to add new products to the system so I can keep the database up to date with the most recent information.

The user inserts the software information such as software name and description and attaches PDF files.

The form validates input fields.

If validation is successful, the system calls the method that triggers adding software to the database, and a confirmation message is displayed.

If validation fails, the system informs the user about the validation error.

A diagram of software

Description automatically generated

***FIGURE 23: ADD SOFTWARE SEQUENCE***

***Updating software***

the diagram shows the process taken when updating software information.

As a financial consultant, I want to be able to update product information as it changes over time so that I can always maintain accurate details.

The user chooses a software from the software display page and clicks on it.

Then software data is retrieved from the database and the update page is made visible.

The retrieved data is displayed in the form for the user to modify. Then the modified details are validated. If validation is successful, the system updates the data in the database. However, if validation fails, the system throws exception. A diagram of software development

Description automatically generated  
 ***FIGURE 24: UPDATE SOFTWARE SEQUENCE***

***DELETE SOFTWARE***

The user chooses a software in the software display page and clicks it.

The data of the vendor is retrieved, and the update page opens.

The vendor data is displayed on the update page.

The user attempts to delete the vendor.

If the software is deleted successfully, a friendly confirmation message appears to confirm the deletion. However, if the vendor fails to delete, the system throws an exception.

A diagram of software development

Description automatically generated

***FIGURE 25: DELETE SOFTWARE SEQUENCE***

# PERSONAL IMPLEMENTATIONS

In this section. Only the core implementation screenshots are provided. Minor button clicks and textbox event handlers are not presented in this report. However, they can be found in the implementation project in the solution file.

The interface below shows the consultant dashboard. It displays the name of the consultant with his role. There are also two buttons, vendor’s and software. Vendor is for navigating to the vendor related tasks, while software is for navigating to the software related area.

A screenshot of a computer

Description automatically generated

***CONSULTANT DASHBOARD***

FIGURE 1. User story: As a financial consultant, I want the ability to add new vendors to the system so I can keep the database up to date with the most recent information.

Figure 26 showcases the process of adding a new vendor to the system in the application. It involves adding company name, website, number of employees, and established date. The number of employees accepts only integers and ‘-’ sign, while the established date takes the date format of **(YYYY-MM-DD**). After the input of vendor details, the user clicks the “**Add**” button to add the vendor as shown in the interface. The **“>>Back”** button at the top left is for navigating to the “**vendorDisplay**” panel.

A screenshot of a computer

Description automatically generated

***FIGURE 26: ADDING A VENDOR***

Figure 27 displays vendors available in the database. The “**AddVendor**” button at the top right of the interface is for navigating the page of adding a new vendor, While the “**linkSoftwareToVendor**” button is to link a software to a vendor(owner).

A screenshot of a computer

Description automatically generated

***FIGURE 27: DISPLAYING A VENDOR***

***FORMS CLASS***

Figure 28 shows the process of adding a vendor to database. It retrieves input details from the textboxes, validates their completeness and invokes the “**AddVendorToDatabase**” method from the “**DBConnection**” class. However, prior to invoking the database operation, vendor details such as company name, website, established date and number of employees must be provided. After successful execution, textbox controls are reset and the “**vendorDisplay**” panel is made visible, making the interface intuitive. In case any of the input fields are empty, a user-friendly message appears, informing the user to complete all required fields.

A screen shot of a computer program

Description automatically generated

***FIGURE 28: IMPLEMENTING ADDING A VENDOR***

Figure 29 paints the “**vendorDisplay**” panel. “**getVendor**” is a dataset populated by loading the vendor data using the “**LoadUsers**” method from the “**dbconnection**”, which is an instance from the “**DBConnection**” class. It interacts with the database through the SQL queries by “**SqlQueries.displayVendor()**”. The first table data in the dataset will be displayed in the “**displayVendorInGridView**” panel as its data source. The whole code is to refresh and display the latest up-to-date vendor details.

A screen shot of a computer program

Description automatically generated

***FIGURE 29: LOADING VENDOR DATA TO GRID VIEW***

***SqlQueries CLASS***

Figure 30 show the query of adding a vendor to the database in the SqlQueries Class.



***FIGURE 30: QUERY FOR ADDING A VENDOR TO THE DATABASE***.

***DBConnection CLASS***

Figure 31 demonstrates the logic for inserting vendor information into database. Parametrized queries are utilised to prevent SQL injection vulnerabilities. A secure connection to a database is established, opening the database for interaction and utilizing “**SqlCommand**” object to execute SQL query. Company name, website, established date and number of employees parameters are added to the SQL command if respective values are provided. Upon executing the SQL command, a success message is shown to the user through “**MessageBox**”.

A screen shot of a computer program

Description automatically generated

***FIGURE 31: LOGIC BEHIND ADDING A VENDOR IN DB connection CLASS***

user story: As a financial consultant, I want to be able to update vendor information as it changes over time so that I can always maintain accurate details.

Figure 32 shows the process of updating a vendor. It involves modifying the current vendor information and clicking the “**update**” button. Please note that the reference number cannot be modified.

A screenshot of a computer

Description automatically generated

***FIGURE 32: UPDATING A VENDOR***

Figure 33 displays the updated vendor. The grid views handle cell clicks, and that is how we get into the update page.

A screenshot of a computer

Description automatically generated

***FIGURE 33: DISPLAY UPDATED VENDOR***

***FORMS CLASS***

Figure 34 demonstrates the implementation of updating a vendor in database. It begins by validating each textbox, to ensure fields are not left empty, displays error message if validation fails. The process involves invoking the “**changeVendor**” method in the “VENDOR” class, which connects with a database. Upon successful update, a prompt confirmation message appears, then clearing the textboxes, and a possible navigation to the vendor display page. Proper exception handling is used to handle possible errors during the update process.

A screen shot of a computer

Description automatically generated

***FIGURE 34: IMPLEMENTING UPDATING A VENDOR***

Figure 35 shows the interaction with “**displayVendorInGridView**” panel. Upon a valid cell click, the “**updateRefNoTextBox**” is configured as read-only. Vendor details, such as reference number, company name, website, established date, and number of employees are dynamically populated within the textboxes. The event handler allows viewing, interaction with vendor data and updating/deleting vendor information. The “**updateVendorPanel**” is set to be visible to allow the user to modify and update or delete. The visual selection of the entire row gives the user a clear indication that the row is selected and the read-only reference number ensures data integrity.

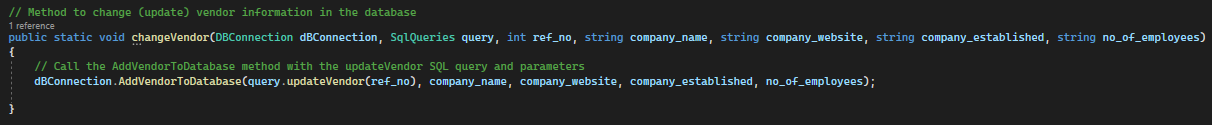
A computer screen shot of a program

Description automatically generated

***FIGURE 35: CELL CLICK EVENT IN GRID VIEW***

***VENDOR CLASS***

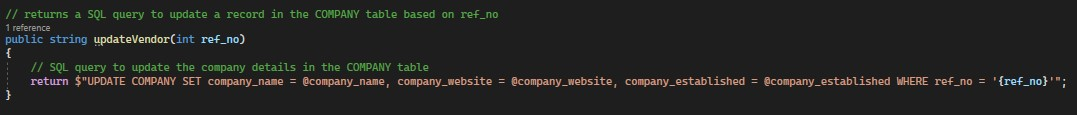
Figure 36 shows the static method “**changeVendor**” within the “**VENDOR**” class. It facilitates the update of vendor information in the database by invoking the “**AddVendorToDatabase**” given in the “**DBConnection**” instance. It passes the SQL query for update of vendors obtained from the “**SqlQueries**” instance along with parameters for update. Moreover, the code snippet promotes modularity by implementing the logic in a separate method making it more readable and maintainable.



***FIGURE 36: UPDATING A VENDOR***

***SqlQueries CLASS***

Figure 37 shows the query for updating vendor in database.



***FIGURE 37: QUERY FOR UPDATING VENDOR IN DATABASE***

user story: As a financial consultant, I want to be able to DELETE vendor information as SOME VENDORS HAVE THEIR CONTRACTS FINISH so that I can always maintain accurate details.

Figure 38 shows the process of deleting a vendor. There is no need to clear the text boxes before deleting, we can just go straight to clicking the “**Delete**” button.

A screenshot of a computer

Description automatically generated

***FIGURE 39: DELETING A SOFTWARE***

Figure 40 displays the grid view state after deleting the vendor in figure 3.1.

A computer screen with a white screen

Description automatically generated

***FIGURE 40: DISPLAY GRID VIEW AFTER DELETE***

***FORMS CLASS***

The event handler in Figure 41 triggers the deletion of the vendor from the database by calling the “**eraseVendor**” method from “**VENDOR**” class. it takes the parameters of the two instances and the text boxes. “**AddVendorToDatabase**” method in the “DBConnection” instance facilitates the deletion using the **“sqlQueries.deleteVendor()”**. The text boxes are cleared after successfully deleting a vendor to maintain a clean interface.

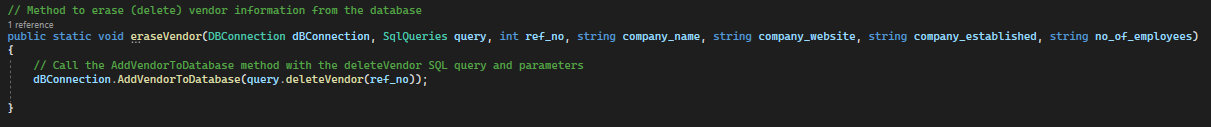
A computer screen with text on it

Description automatically generated

***FIGURE 41: DELETE IMPLEMENTATION***

***VENDOR CLASS***

Figure 42 of the “**VENDOR**” class demonstrates the logic of deleting a vendor from database. “**DBConnection**” instance is used to delete the SQL query.



***FIGURE 42: DELETING LOGIC CODE***

***SqlQueries CLASS***

Figure 43 shows the query for deleting vendors in the database.

A computer screen with text

Description automatically generated

***FIGURE 44: QUERY FOR DELETING VENDOR***

User story: As a financial consultant, I want the ability to add new products to the system so I can keep the database up to date with the most recent information.

Figure 45 demonstrates adding new software to the system in the application. It includes inputting software name, and description and attaching a PDF file. The description textbox accepts only characters, commas, and the use of the backspace key. The label path is attached as well.

A screenshot of a computer

Description automatically generated

***FIGURE 45: ADDING SOFTWARE***

Figure 46 displays the software’s in the database. The “**AddSoftware**” button is for navigating to the page of adding a new software. Cell clicks are enabled to navigate to the update page for updating or deleting software.A screenshot of a computer

Description automatically generated

***FIGURE 46: DISPLAY ADDED SOFTWARE***

***FORMS CLASS***

Figure 47 demonstrates the actions triggered when the “**btnAddSoftware**” button is clicked. It undertakes the reading of the PDF file into a byte array, and extracts the software name, description, and file path from their corresponding textboxes, it then validates the completeness of the entered software details. After successful validation, the “**addSoftwareToDatabase**” method from the “**DBConnection**” class is invoked to execute the insertion of the software details into the database.

A screen shot of a computer program

Description automatically generated

***FIGURE 47: ADDING SOFTWARE TO DATABSE IMPLEMENTATION***

Figure 48 shows the similar event of figure 11 of painting the “**vendorDisplay**” panel. However, in this event handler, it paints the “**displaySofwarePanel**”. “**getSoftware**” is the dataset populated by loading software details using the “**LoadUsers**” method from the “**dbconnection**” instance.

A screen shot of a computer program

Description automatically generated

***FIGURE 48: LOADING SOFTWARE DATA TO GRID VIEW***

***DBConnection CLASS***

The “**AddSoftwareToDatabase**” method within the “**DBConnection**” class triggers the interaction with the database. It establishes a secure connection to the database through an “**SqlConnection**” instance, opens and close the database connection. It creates an “**SqlCommand**” object to execute the SQL query. Parameters are properly set in the SQL command to guard against possible SQL injection vulnerabilities. After successfully executing the SQL command, a user-friendly success message is displayed using ‘**MessageBox’** class.

A computer screen shot of many colorful lines

Description automatically generated

***FIGURE 49: ADDING SOFTWARE TO DATABASE CONNECTION LOGIC***

***SqlQueries CLASS***

Figure 50 shows the query for adding a software to database.



***FIGURE 50: QUERY FOR ADDING SOFTWARE TO DATABASE***

User story: As a financial consultant, I want to be able to update product information as it changes over time so that I can always maintain accurate details.

Figure 51 shows the process of updating software. All the textboxes and the PDF file can be modified except the software ID.

A screenshot of a computer software

Description automatically generated

***FIGURE 51: UPDATING A SOFTWARE***

Figure 52 displays the updated software in figure 51.

A screen shot of a computer

Description automatically generated

***FIGURE 52: DISPLAY UPDATED SOFTWARE***

***FORMS CLASS***

Figure 53 begins by validating textboxes, checking for empty fields, it displays an error message when needed. The software ID is set to read-only to prevent unnecessary changes. It the updates software details by invoking the “**changeSoftware**” method in the “**SOFTWARE**” class. Upon successful update, the software display panel is set visible. Exception handling is used for possible errors during the update process, specifically prompting a message for PDF attaching.

A screen shot of a computer code

Description automatically generated

***FIGURE 53: UPDATE SOFTWARE IMPLEMENTATION***

Figure 54 demonstrates the user interaction with the “**dataGridView2**” Panel. When a cell Is clicked, it sets the “**softIDTextBox**” as read-only, selects the entire row visually, and populate the textbox controls with the clicked software details. The software update panel is adjusted to be visible for updating or deleting software information. Additionally, it enables the label path of the attached document to be clicked to open the attached PDF document (opening attached PDF is shown in figure 6).

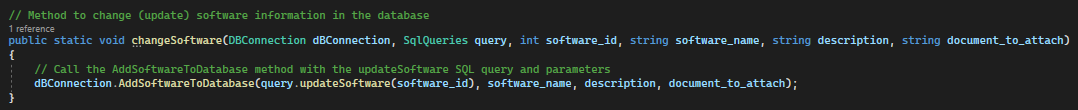
A screen shot of a computer program

Description automatically generated

***FIGURE 54: SOFTWARE DISPLAY PANEL CELL CLICK ENABLED.***

***SOFTWARE CLASS***

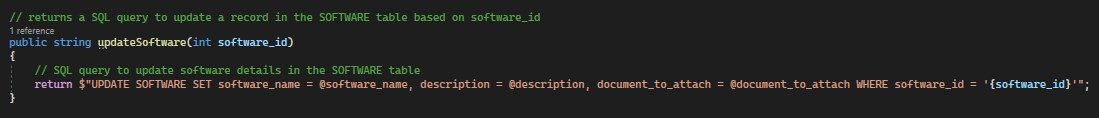
The figure below is a static method within the “SOFTWARE” class. The author designed it to update software details in the database. The method calls the “AddSoftwareToDatabase” method from the “DBConnection” instance passing SQL query for updating the software details. The logic of the way the author designed the code promotes code organization and maintainability in database related contexts.



***FIGURE 55: UPDATE SOFTWARE CODE***

***SqlQueries CLASS***

Figure 56 shows the query for updating a software in database.



***FIGURE 56: QUERY FOR UPDATING SOFTWARE IN DATABASE***

User story: As a financial consultant, I want to be able to DELETE product information as SOME PRODUCTS WILL NO LONGER BE USED BY VENDORS so that I can always maintain accurate details.

Figure 57 shows the process of deleting a software. Similar to figure 3, there is no need for clearing the text boxes before deleting.

A screenshot of a software

Description automatically generated

***FIGURE 57: DELETING A SOFTWARE***

Figure 58 displays the grid view state after deleting software in figure 57.

A computer screen shot of a grey screen

Description automatically generated

***FIGURE 58: DISPLAY GRIDVIEW AFTER DELETING SOFTWARE***

***FORMS CLASS***

Figure 59 is triggered when the “**btnDeleteSoftwareFromDatabase**” button is clicked. It initiates the deletion of a software from the database by calling the “**eraseSoftware**” method in the “**SOFTWARE**” class. The method is a static function, it takes parameters such as the instances “**dbconnection**” and “**sqlQueries**” along with the textboxes and the file path label. Th deleting of the software is done through the “**AddSoftwareToDatabase**” method of the “**DBConnection**” instance with the use of the “**deleteSoftware**” SQL query in the “**SqlQueries**” class. After successfully deleting the software, a success message appears within a message Box, and the textboxes and the document path are cleared indicating to the user that it has been deleted.

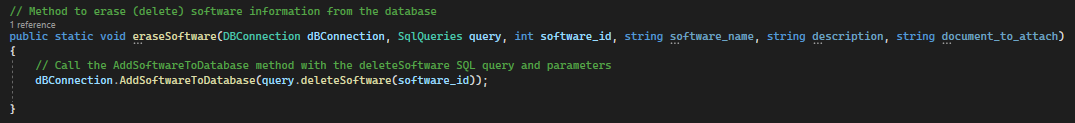
A computer screen shot of a black screen

Description automatically generated

***FIGURE 59: DELETING IMPLEMENTATION***

***SOFTWARE CLASS***

Figure 60 defined in the “**SOFTWARE**” class showcases the logic of deleting the software from database. It uses the “**DBConnection**” instance to execute the SQL query given “**query.deleteSoftware(software\_id)”**.



***FIGURE 61: DELETING CODE***

***SqlQueries CLASS***

Figure 62 shows the query for deleting software from database.

A screen shot of a computer program

Description automatically generated

***FIGURE 62: QUERY FOR DELETING SOFTWARE IN DATABASE***

User story: As a financial consultant, I want to be able to link the software to the vendor (owner) so that I can establish the connection between the software and the owner.

Figure 63 showcases the process of linking a software to vendor. A small panel is set visible after the “**linkSoftwareToVendor**” button is clicked. The user inserts the company’s reference number and the software ID of the software. The software is linked successfully after clicking the “**link**” button. The linked software will only be visible when the user acknowledges the displayed success message.

A screenshot of a computer

Description automatically generated

***FIGURE 63: LINK SOFTWARE TO VENDOR***

Figure 64 shows the vendor and the linked software in a grid view. The small linking panel is hidden after clicking on the “**linkSoftwareToVendor”.**

**A screenshot of a computer

Description automatically generated** ***FIGURE 64: DISPLAYING LINKED SOFTWARE TO VENDOR***

***FORMS CLASS***

Figure 65 is a click handler that toggles the visibility of the “**vendorToSoftwareLinkPanel**” panel. It displays a small panel for linking a software to a vendor.

A screen shot of a computer code

Description automatically generated

***FIGURE 65: TOGGLING OF THE LINKING PANEL***

The event handler in the figure 66 is triggered when the “**InitiateVenSoftLinkButton**” is clicked. It retrieves the reference number of the company(vendor) and the software ID of the software. It then calls the “**LinkVendorToSoftware**” method in the “**DBConnection**” class passing the “**SqlQueries.LINK\_QUERY**” query, vendor(company) reference number, and software ID.

A screen shot of a computer program

Description automatically generated

***FIGURE 66: LINK SOFTWARE TO VENDOR IMPLEMENTATION***

***DBConnection CLASS***

Figure 67 establishes a connection to the database and executes the SQL query of linking vendor to software using the given IDs, it then displays a success message through **MessageBox**. Parameterised queries and proper resource management are used to enhance security, providing a structured approach to database operations.

A computer screen shot of a program

Description automatically generated

***FIGURE 67: LOGIC OF ESTABLISHING CONNECTION TO DATABASE IN DBConnection CLASS***

***SqlQueries***

Figure 68 shows the query for linking a software to vendor.



***FIGURE 68: QUERY FOR LINKING SOFTWARE TO VENDOR.***

User story: As a financial consultant, I want to be able to attach PDF documents (For example, product manuals) to the records of products so that I can access relevant documentation in the application.

Figure 69 demonstrates the process of attaching a pdf file to the system. The user clicks the ‘browse’ button, then the file explorer is opened for the user to choose a PDF file to attach. Only pdf files can be attached.

A screenshot of a computer

Description automatically generated

***FIGURE 69: FILE DIALOG***

The figure below shows the attached PDF file to the system.

A screenshot of a software

Description automatically generated

***FIGURE 70: ATTACHED FILE WITH PATH***

The figure below shows the **opened** attached **PDF** file by clicking on the label path. The opened PDF file cannot be edited or modified; it is only for viewing. The author found out that for us to be able to edit or modify the **PDF** file, we must install a new **NuGet** package or even use an external source. As a result, the author proceeded with only viewing the **PDF** file and not editing or modifying it. However, the author has implemented editing of a text file by inserting the text file data into a textbox to modify, edit and save. However, since it is not part of the application requirement, the author removed it.

A screenshot of a computer

Description automatically generated

**FIGURE 71: OPENING PDF DOCUMENT**

***FORMS CLASS***

Figure 72 is designed by the author to open the **PDF** file. Its path is stored in the “**filePath**” label control. “**Process.Start(documentPath**)” is used to open the file using the default windows.

A computer screen shot of code

Description automatically generated

***FIGURE 72: OPPENING ATTACHED A FILE.***

***SOFTWARE CLASS***

Figure 73 initialises an instance “**OpenFileDialog**” and configure it to filter only PDF files. The method checks for if the user clicks the “**OK**” button in the **openFileDialog**. If file is selected, the file path will be obtained using **openFileDialog.FileName**. if file is selected, the method returns the selected file, however, if not file is selected at the end, the method returns **null**.

A computer screen shot of a program code

Description automatically generated

***FIGURE 73: ATTACHING A FILE METHOD FROM SOFTWARE CLASS***

***SOFTWARE CLASS***

Figure 74 takes string inputs to validate the input as **DATE** in the format “**YYYY-MM-DD**”.

A computer screen with text on it

Description automatically generated

***FIGURE 74: VALIDATE USING THE GIVEN DATE FORMAT***

***Dbconnection CLASS***

Figure 75 shows the method of fetching data from the database based on the provided SQL query and then returning the result as a dataset. Proper resource management is used by using the “**using**” statement for **SqlDataAdapter** and **SqlConnection**. Finally, any exception is rethrown.

A computer screen shot of a computer code

Description automatically generated ***FIGURE 75: FETCH DATA***

***FORMS CLASS***

Figures 76 and 77 limit the length of characters in the textbox. It invokes the ‘**textboxlength’** method created by a member of the group.

A computer screen shot of text

Description automatically generated

***FIGURE 76: COMPANY NAME TEXTBOX IN UPDATE VENDOR FORM***

A screen shot of a computer program

Description automatically generated

***FIGURE 77: COMPANY NAME TEXTBOX IN ADD VENDOR FORM***

The figure below enforces the maximum length of 20 characters for the entered text. It ensures the text does not exceed the limit.

**A computer screen shot of a program code

Description automatically generated**

***FIGURE 78: SOFTWARE NAME IN UPDATE AND ADD SOFTWARE FORMS***

Figure 79 makes the ‘**softwaredisplaypanel’** to be visible.

A computer screen with white text and colorful text

Description automatically generated

***FIGURE 79: NAVIGATE TO SOFTWARE DISPLAY PANEL***

Figure 80 and 81 performs DATE validation by checking if the entered text is a valid **DATE** or not. The error provider displays an error message if validation fails.

**A computer screen shot of a program code

Description automatically generated**

***FIGURE 80: TEXTBOX IN UPDATE VENDOR FORM***

**A computer screen with colorful text

Description automatically generated*FIGURE 81: TEXTBOX IN ADD VENDOR FORM***

Figure 82 is triggered when the click ‘**btnAddSoft’** is clicked. It makes the ‘**addSoftwarePanel’** to be visible.

A screen shot of a computer code

Description automatically generated

***FIGURE 82: NAVIGATE TO ADD SOFTWARE FORM***

# PERSONAL TESTING

**ADD VENDOR FORM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test no.** | **description** | **Expected result** | **Actual result** | **Pass/ fail** |
| 1 | Click “Add” button with empty fields in add vendor form. | Please enter a value in Company Name, Website, Established Date, Number of employees. | Please enter a value in Company Name, Website, Established Date, Number of employees. | PASS |
| 2 | Click “Add” button with only adding Company Name. | Please enter a value in website, established date, number of employees | Please enter a value in website, established date, number of employees | PASS |
| 3 | Click “Add” button with only adding website. | Please enter a value in company name, established date, number of employees | Please enter a value in company name, established date, number of employees | PASS |
| 4 | Click “Add” button with only adding number of employees. | Please enter a value in company name, website, established date | Please enter a value in company name, website, established date | PASS |
| 5 | Click “Add” button with only adding established date. | Please enter a value in company name website, number of employees | Please enter a value in company name website, number of employees | PASS |
| 6 | Click “Add” button with only adding company name and website. | Please enter a value in established date, number of employees | Please enter a value in established date, number of employees | PASS |
| 7 | Click “Add” button with only adding company name and number of employees. | Please enter a value in website, established date | Please enter a value in website, established date | PASS |
| 8 | Click “Add” button with only adding company name and established date. | Please enter a value in website, number of employees | Please enter a value in website, number of employees | PASS |
| 9 | Click “Add” button with only adding website and number of employees. | Please enter a value in company name, established date | Please enter a value in company name, established date | PASS |
| 10 | Click “Add” button with only adding website and established date. | Please enter a value in company name, number of employees | Please enter a value in company name, number of employees | PASS |
| 11 | Click “Add” button with only adding number of employees and established date. | Please enter a value in company name, website | Please enter a value in company name, website | PASS |
| 12 | Add all Company in formation and click” Add” button. | Company Added Successfully | Company Added Successfully | PASS |
| 13 | Add all company information except company name | Please enter a value in company name | Please enter a value in company name | PASS |
| 14 | Add all company information except website | Please enter a value in website | Please enter a value in website | PASS |
| 15 | Add all company information except number of employees | Please enter a value in number of employees. | Please enter a value in number of employees. | PASS |
| 16 | Add all company information except established date | Please enter a value in established date | Please enter a value in established date | PASS |
| 17 | Add character in number of employees | Textbox stays empty (nothing displays) | Textbox stays empty (nothing displays) | PASS |
| 18 | Add anything other than date format in established date | Please enter a valid date (YYYY-MM-DD) | Please enter a valid date (YYYY-MM-DD) | PASS |
| 19 | Try exceeding the characters limit the company name text box | No more characters can be added | No more characters can be added | PASS |
| 20 | Try exceeding the characters limit the website text box | No more characters can be added | No more characters can be added | PASS |

**UPDATING VENDOR FORM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TEST NO.** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** | **PASS/ FAIL** |
| 1 | Clear only the company name and click “update” button. | Please enter a value in company name. | Please enter a value in company name. | PASS |
| 2 | Clear only the website and click “update” button. | Please enter a value in website. | Please enter a value in website. | PASS |
| 3 | Clear only the number of employees and click “update” button. | Please enter a value in number of employees. | Please enter a value in number of employees. | PASS |
| 4 | Clear only the established date and click “update” button. | Please enter a valid date (YYYY-MM-DD). | Please enter a valid date (YYYY-MM-DD). | PASS |
| 5 | Try clearing the refence number. | Can’t be cleared. | Can’t be cleared. | PASS |
| 7 | update all Company in formation and click” Add” button. | Update is Successful. | Update is Successful. | PASS |
| 8 | Add anything other than date format in established date. | Please enter a valid date (YYYY-MM-DD). | Please enter a valid date (YYYY-MM-DD). | PASS |
| 9 | Add character in number of employees. | Textbox stays empty (nothing displays). | Textbox stays empty (nothing displays). | PASS |
| 10 | Click “Update” button with only adding established date. | Please enter a value in company name, website, number of employees. | Please enter a valid value in company name, website, number of employees | PASS |
| 11 | Click “Update” button with only adding number of employees and established date. | Please enter a value in company name, website. | Please enter a value in company name, website. | PASS |
| 12 | Click “Update” button with only adding website and established date. | Please enter a value in company name, number of employees. | Please enter a value in company name, number of employees. | PASS |
| 13 | Click “Update” button with only adding website and number of employees. | Please enter a value in company name, established date. | Please enter a value in company name, established date. | PASS |
| 14 | Click “Update” button with only adding company name and established date. | Please enter a value in website, number of employees. | Please enter a value in website, number of employees. | PASS |
| 15 | Click “Update” button with only adding established date. | Please enter a value in company name, website, number of employees. | Please enter a value in company name, website, number of employees. | PASS |
| 16 | Try inserting or clearing the reference number. | Reference number stays the same, it is read only. | Reference number stays the same, it is read only. | PASS |

**ADDING A SOFTWARE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TEST NO.** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** | **PASS/ FAIL** |
| 1 | Click “Add” button with clear textboxes. | Please enter a value in software name, description. | Please enter a value in software name, description. | PASS |
| 2 | Click “Add” button with only adding the software name. | Please enter a value in description. | Please enter a value in description. | PASS |
| 3 | Click “add” button adding software name and description. | Please attach PDF file. | Please attach PDF file. | PASS |
| 4 | Click “add” button with only attaching a file. | Please enter a value in software name, description. | Please enter a value in software name, description. | PASS |
| 5 | Click “add” button with only attaching file and adding description. | Please enter a value in software name. | Please enter a value in software name. | PASS |
| 6 | Click “add” button with attaching file and adding software name. | Please enter a value in description. | Please enter a value in description. | PASS |
| 7 | Try attaching any file other than a PDF. | Can’t attach the file. | Can’t attach the file. | PASS |
| 8 | Try exceeding the characters limit in the software name text box | No more characters can be added | No more characters can be added | PASS |
| 9 | Try exceeding the character limit for adding software description | No more characters can be added | No more characters can be added | PASS |

**UPDATING A SOFTWARE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TEST NO.** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** | **PASS/ FAIL** |
| 1 | Click “Update” button with clear textboxes. | Please enter a value in software name, description. | Please enter a value in software name, description. | PASS |
| 2 | Click “Update” button with only adding the software name. | Please enter a value in description. | Please enter a value in description. | PASS |
| 3 | Click “Update” button adding software name and description. | Please attach PDF file. | Please attach PDF file. | PASS |
| 4 | Click “Update” button with only attaching a file. | Please enter a value in software name, description. | Please enter a value in software name, description. | PASS |
| 5 | Click “Update” button with only attaching file and adding description. | Please enter a value in software name. | Please enter a value in software name. | PASS |
| 6 | Click “Update” button with attaching file and adding software name. | Please enter a value in description. | Please enter a value in description. | PASS |
| 7 | Try attaching any file other than a PDF. | Can’t attach the file. | Can’t attach the file. | PASS |
| 8 | Try clearing or inserting a value in Software ID | Can’t insert or clear, stays the same, because it is read only. | Can’t insert or clear, stays the same, because it is read only. | PASS |

# PLAN FOR THE SOFTWARE MAINTENANCE

The Application meets the requirements mentioned in the case study. However, bugs are not something that can be avoided in applications. manual testing is done to identify bugs in the program. As a result, all the identified bugs are fixed, however, there might be hidden bugs that are not visible to the user for now. Improvements can be made to make sure the application can be used by users more smoothly.

1. The application does not support the use of multiple languages. So, it cannot be used internationally, implementing multilingual support can be a good idea to allow users to switch between different languages for the user interface and content.
2. The PDF in this application can be opened only for viewing, it does not allow users to modify or edit. Therefore, it must be implemented.
3. There is a need to break the forms class into consultant, login and signup classes. However, in this application, a single Windows form is used to implement both.
4. Having strong backup plans and disaster recovery systems in place to guard against data loss and guarantee prompt recovery in the event of an unplanned event or system failure is an important feature for implementation.
5. The forgotten password in the Log in page has not been implemented. Implementing this functionality would result in better user experience.
6. Implementing Automated testing will also help to improve early identification of bugs in the application.
7. Regular enhancements of the user interface and trending features are implementations that will increase the overall user-friendliness of the application.

# CONCLUSION

Throughout the project lifecycle, we have been through a lot of challenges especially trying to understand how to use Windows forms. We made critical design decisions and made a unified approach based on evolving needs. The phase of the design is a cornerstone for the foundation of our system, which addresses both current needs and is poised for future adaptations and enhancements.

The approach made during the design process ensured that the final application aligned with the expectations. the inclusion of attaching files and opening them for access to more details about vendors and their software showcases our commitment to providing clients with an easy way of delivering their information.

During the transition from the design phase to the development and testing stages, we carry forward valuable insights. Our design methodology made the development more flexible and responsive, making sure that adjustments can be made in response to emerging requirements.

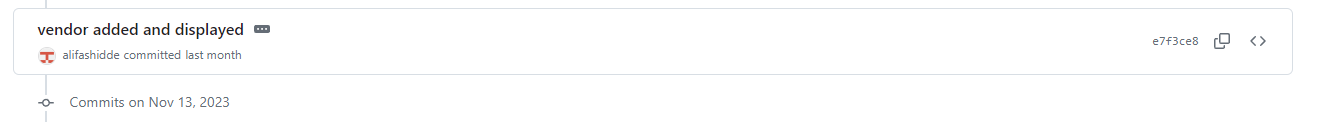
The software design shows the collective expertise, cooperation and commitment of our team, there is a high confidence in us that our solid design will serve as a flexible platform for high-quality and impactful software solutions.

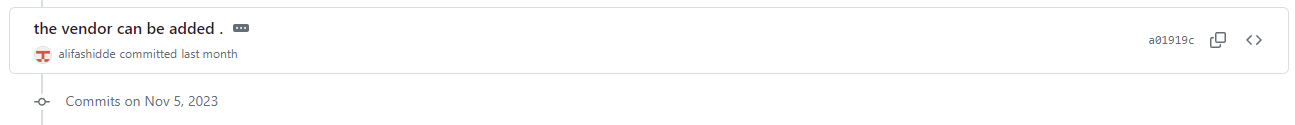
I extend gratitude to all the team members who worked together with me into making the project into a promising reality. We look forward to the deployment of our designed software.

***GITHUB PERSONAL COMMITS***

A white paper with black lines

Description automatically generated





A white rectangular object with a black border

Description automatically generated with medium confidence

# REFERENCES

How do I determine the minimum system requirements for my software deployment by Judyh on the 15th of May 2012?

<https://forums.ni.com/t5/LabWindows-CVI/How-do-I-determine-minimum-system-requirements-for-my-software/td-p/1992341#:~:text=If%20you%20want%20to%20get,the%20minimum%20requirements%20should%20be>. On September 30th.

How to connect and use local database in visual studio 2019 by fox learn, 4 years ago. Available on: <https://www.youtube.com/watch?v=mgtfxtjKoaA> (accessed on: 1st November, 2023)

How to implement custom paging in DataGridView by fox learn, 7 years ago, available on: <https://www.youtube.com/watch?v=L1wpQ_fKjVw> (accessed on: 3rd November, 2023)

Insert, update, delete, and search data from database in C#.net by Amit Andipara, 8 years ago. Available on: <https://www.youtube.com/watch?v=TIAOr2S6-SY> (accessed on: 11th November, 2023)

SQL tutorial by w3 schools, available on: <https://www.w3schools.com/sql/> (accessed on: 14 november, 2023)

How to make a UML sequence diagram by Lucid software, 5 years ago. Available on: <https://www.youtube.com/watch?v=pCK6prSq8aw> (accessed on: 6th October, 2023)

UML class diagrams by lucid software, 4 months ago, available on: <https://www.youtube.com/watch?v=6XrL5jXmTwM> (accessed on: 4th October 2023)

UML use case diagrams by lucid software, 2 months ago available on: <https://www.youtube.com/watch?v=4emxjxonNRI> (accessed on: 26th October, 2023)

Extended entity relationship model by ARU database design and implementation module, (accessed on 4th November, 2023).

Coding with C# by ChatGPT on 1st november.