Organizational Tracking Management and Budget System

CRITERION A

DEFINING THE PROBLEM:

NPO X is an organization that revolves on commitment to improving the quality of life of every individual, including persons with disabilities by allowing them to reach their utmost potential. Many unfortunate individuals face discrimination when faced with society, where some have to leave their lives and start a new one because of it. NPO X provides these individuals with a second opportunity at life, providing jobs that are funded by donors, and donations that fund employment for PWDs. However, the organization has faced issues regarding the organization of resources, and outlining personnels due to their lack of management in validating tasks or fundings. Ms Y, a representative of the organization exclaims how inefficient the state of their current system is — budgets are tracked through memory, their donors and donations through spreadsheets, and logbooks for funds and sales. She also discusses how employees of the organization are always in arguments with each other for their lack of communication on tasks done, and what needs to be done. Additionally, the issues of information being decentralized and handled by different individuals limit the organization's ability to accomplish tasks, while being transparent with one another. As such, Ms Y and X Organization are requesting a newer system that provides flow of information that allows communication between one employee to another at different times at work.

RATIONALE FOR SOLUTION:

X Organization derives their activities and projects from the information they collect, such as personal records of donors and clients, or records of funds and expenses. Their request for a database system in a centralized area provides ease of accessibility and presentability. The program is a tracking management system that contains information of the organization's donors, donations, inventory, sales, as well as expenses, which are the vital information they mainly track for daily activities. Because information is sensitive, the program should utilize log-ins and audit logs to keep track of changes. This allows the organization to maintain transparency, especially when accessing, which allows a solution on miscommunication and mishandling of funds.

Python is the programming language chosen, as it implements key features of security with quick efficiency and massive scalability in a short amount of time, so that the time in which delivery is expected will be met, alongside a safe and quality software for X Organization. It also allows installation of other packages such as cryptography and psycopg2, which allows encryption and decryption of sensitive and confidential information important to the Organization, while being able to communicate with the database that is PostgreSQL. For graphical user interface, Custom TKinter (CTk) will be used to communicate the backend with the frontend, so that the visualization of information can be easily processed.

SUCCESS CRITERIA:

1. A notes page facilitates communication between clients
2. Calculations within the calculator are saved while the program is on
3. The program allows navigation based on specific tasks
4. Logging in is facilitated to provide information security for the client
5. The client can view the organization's information through tables
6. The program must provide clients with the option to delete data of no relevance
7. The program must allow the client to add information depending on the task at hand
8. The program ensures edits can be made by the client to edit necessary information
9. The client can filter information through search function for easy access of information
10. All client's items can be referenced with a uniquely generated ID
11. The client can archive information for better finance management
12. All client data must be linked to the client's branches
13. The program saves the client's progress when multitasking
14. The program utilizes error handling for managing complex tasks

CRITERION B

HOME/NOTES PAGE

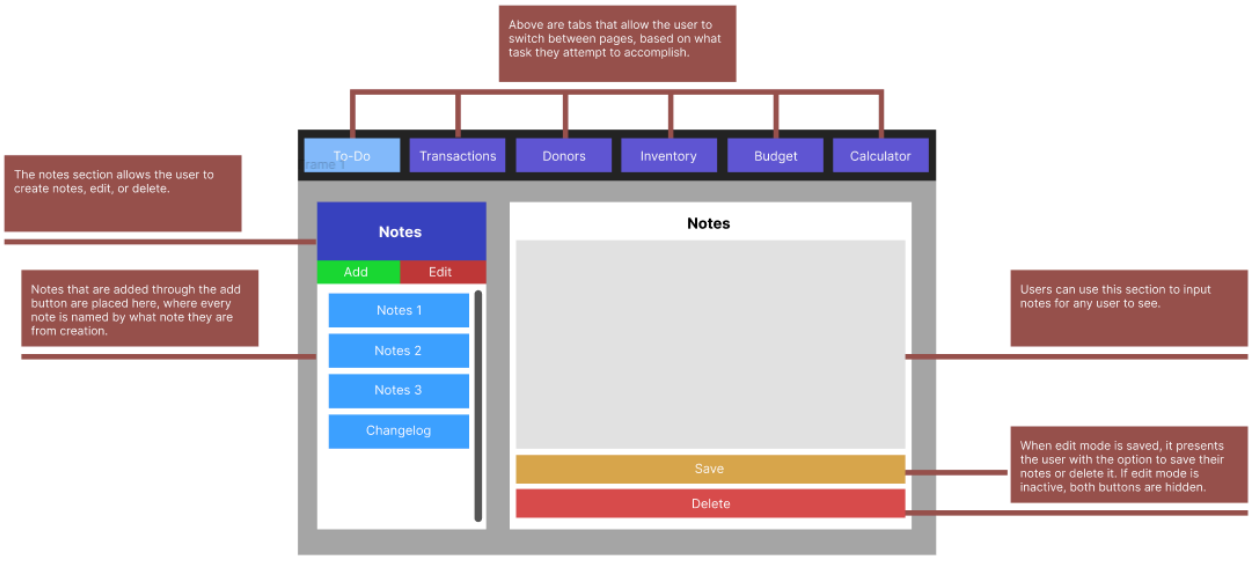


Figure 1. Home Page User Interface

LOGIN-CREDENTIALS

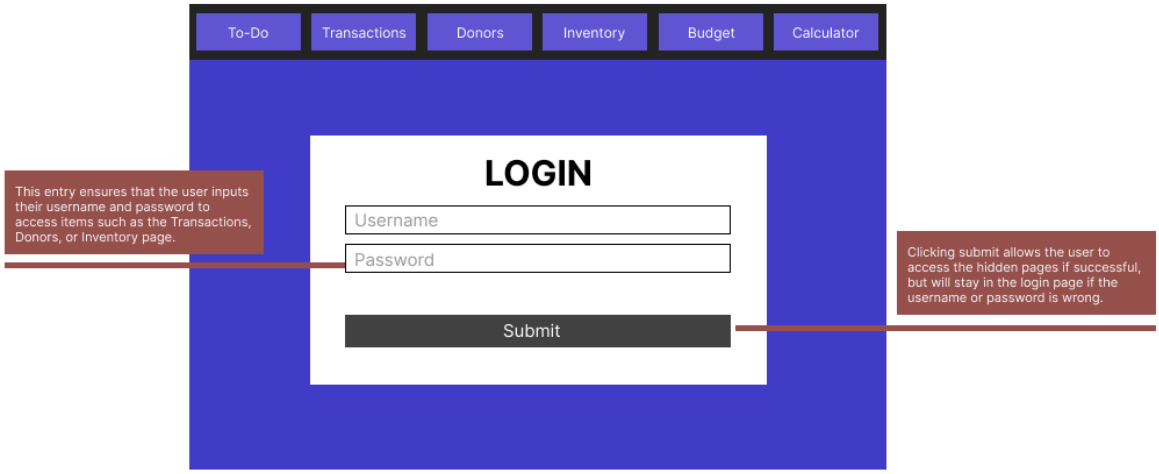


Figure 2. Log-in User Interface

CLIENT/DONATOR *REPLACE SALES WITH BUDGET/FINANCE, and ASSETS WITH INVENTORY/ASSETS

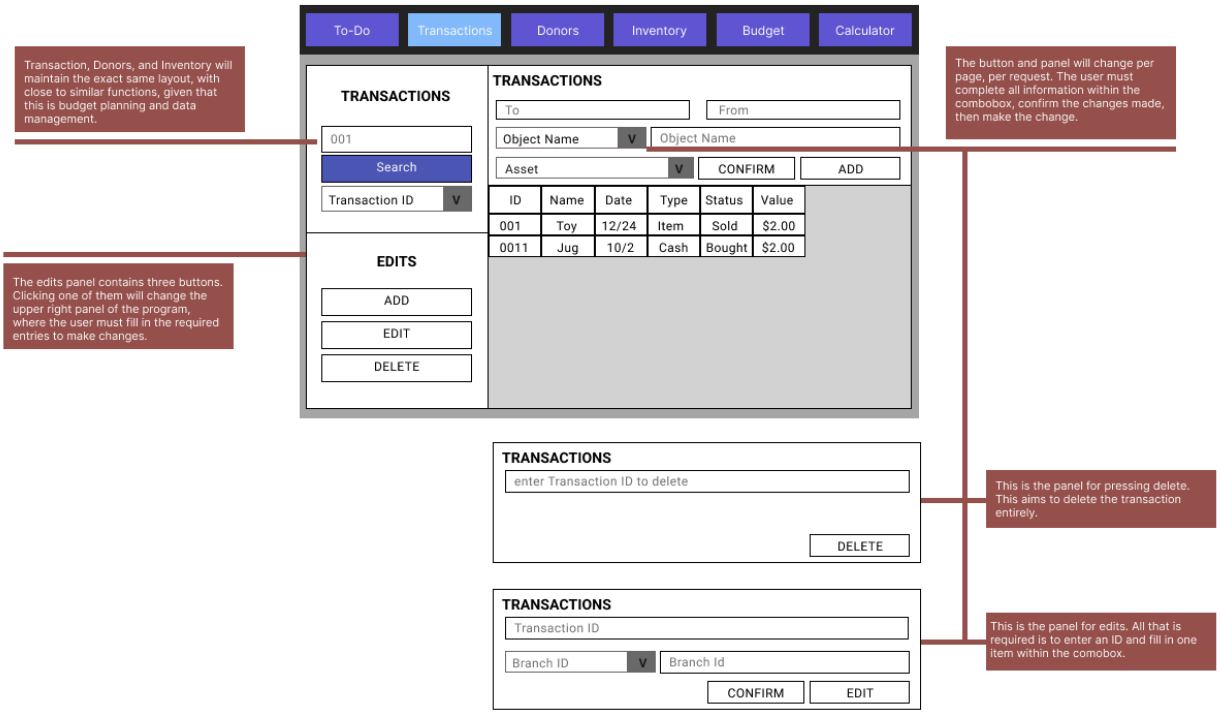


Figure 3. Main Page User Interface

BUDGET PAGE

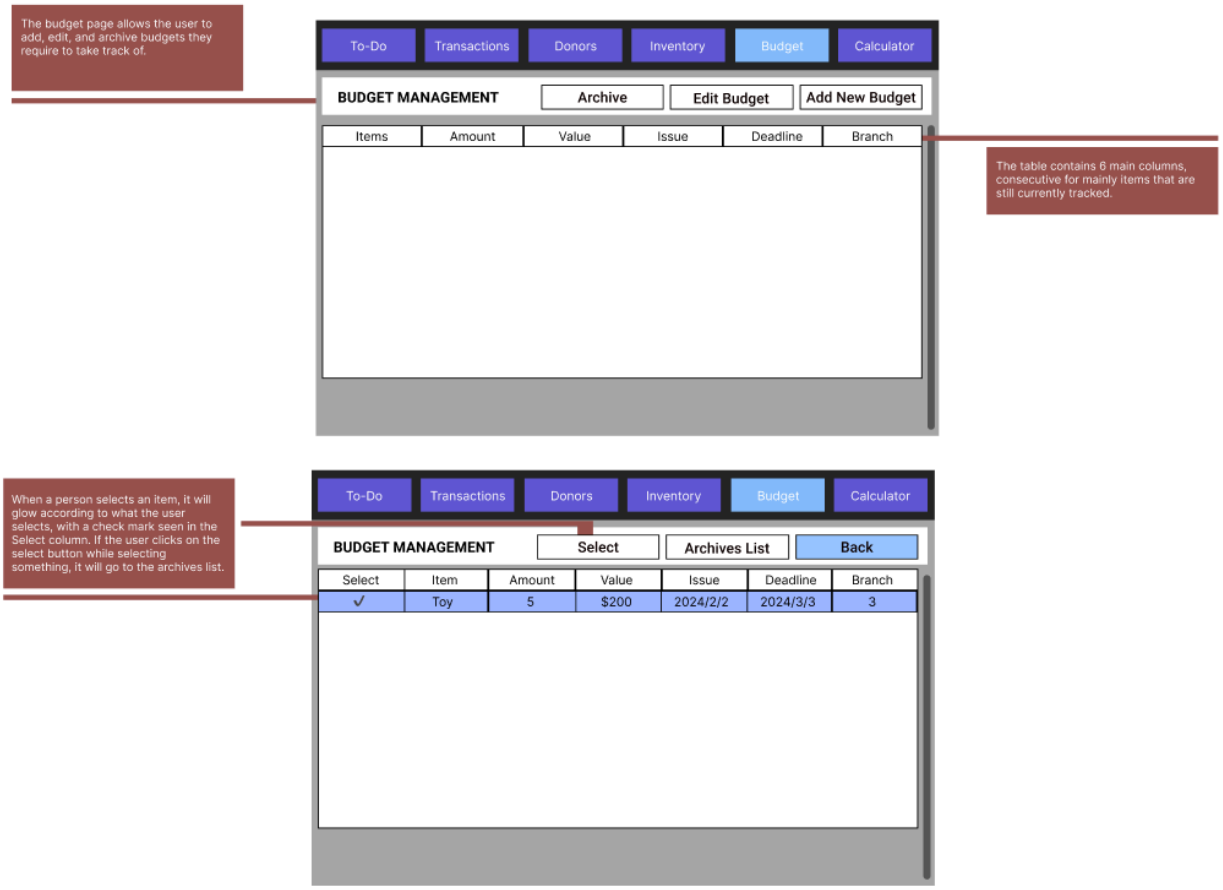


Figure 3. Budget Page Table User Interface

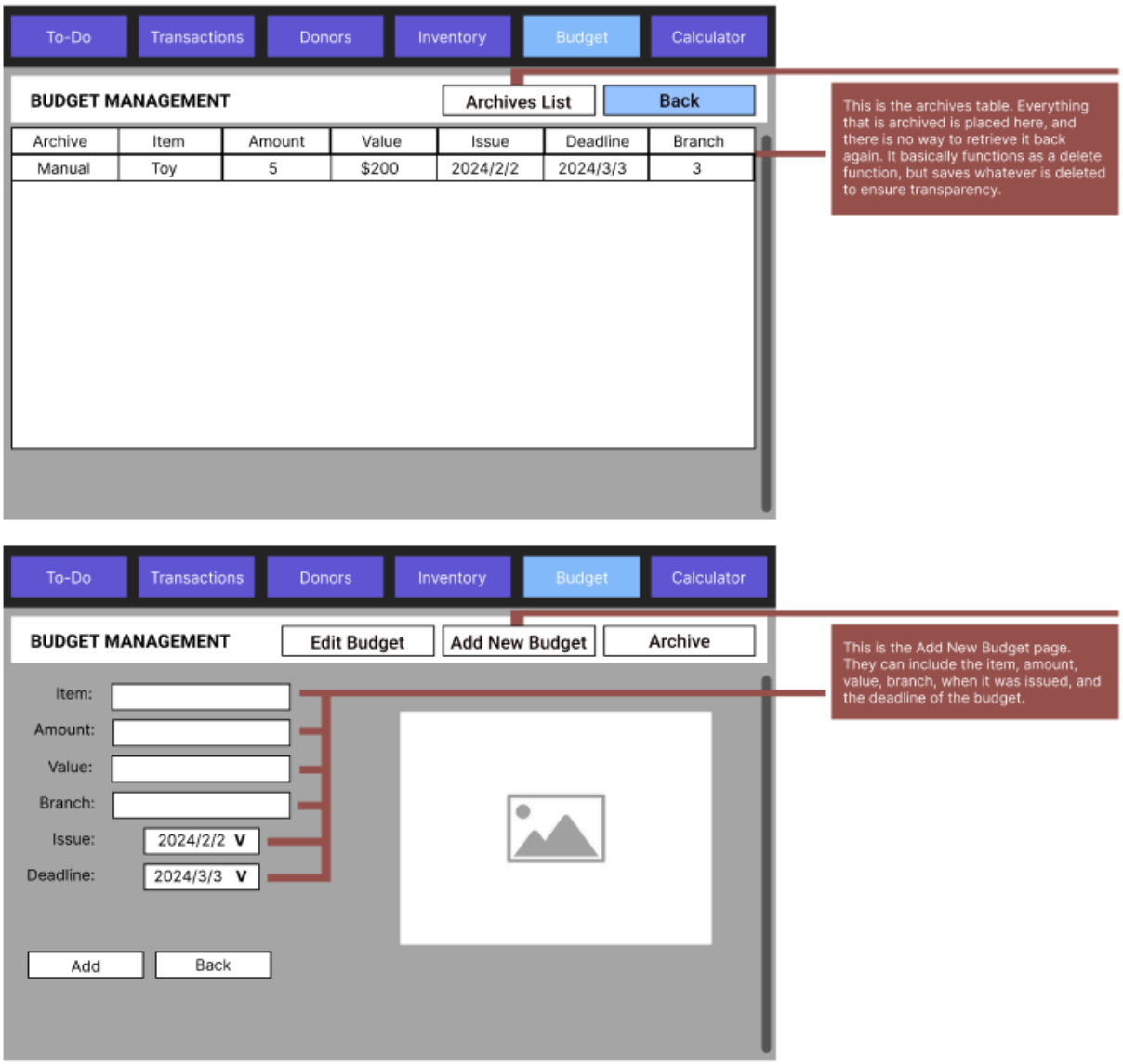


Figure 4. Budget Page Add User Interface and Archive Table

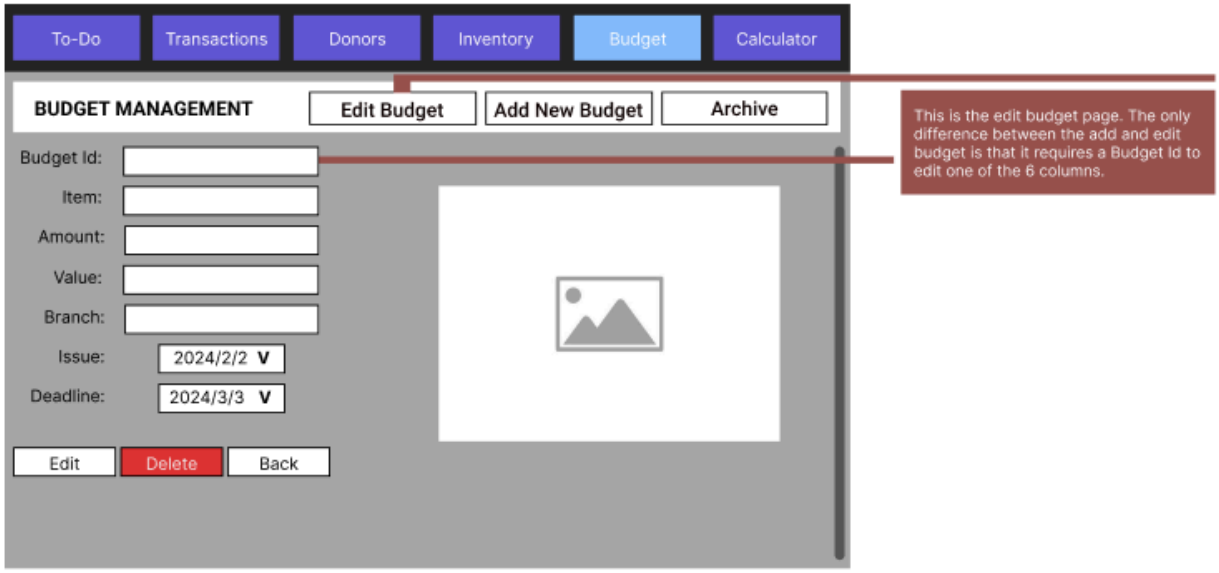


Figure 5. Budget Page Edit User Interface

CALCULATOR

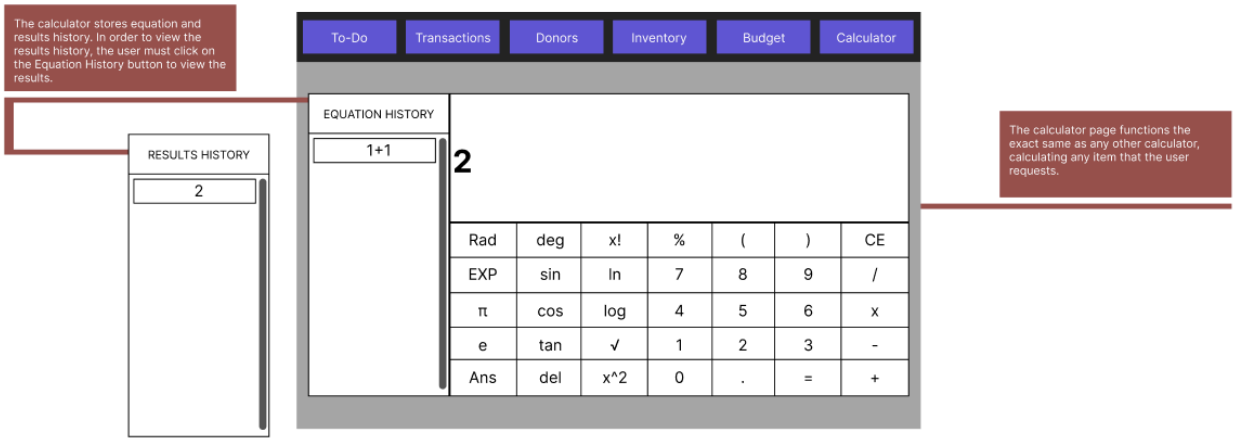


Figure 6. Calculator User Interface

FLOWCHARTS

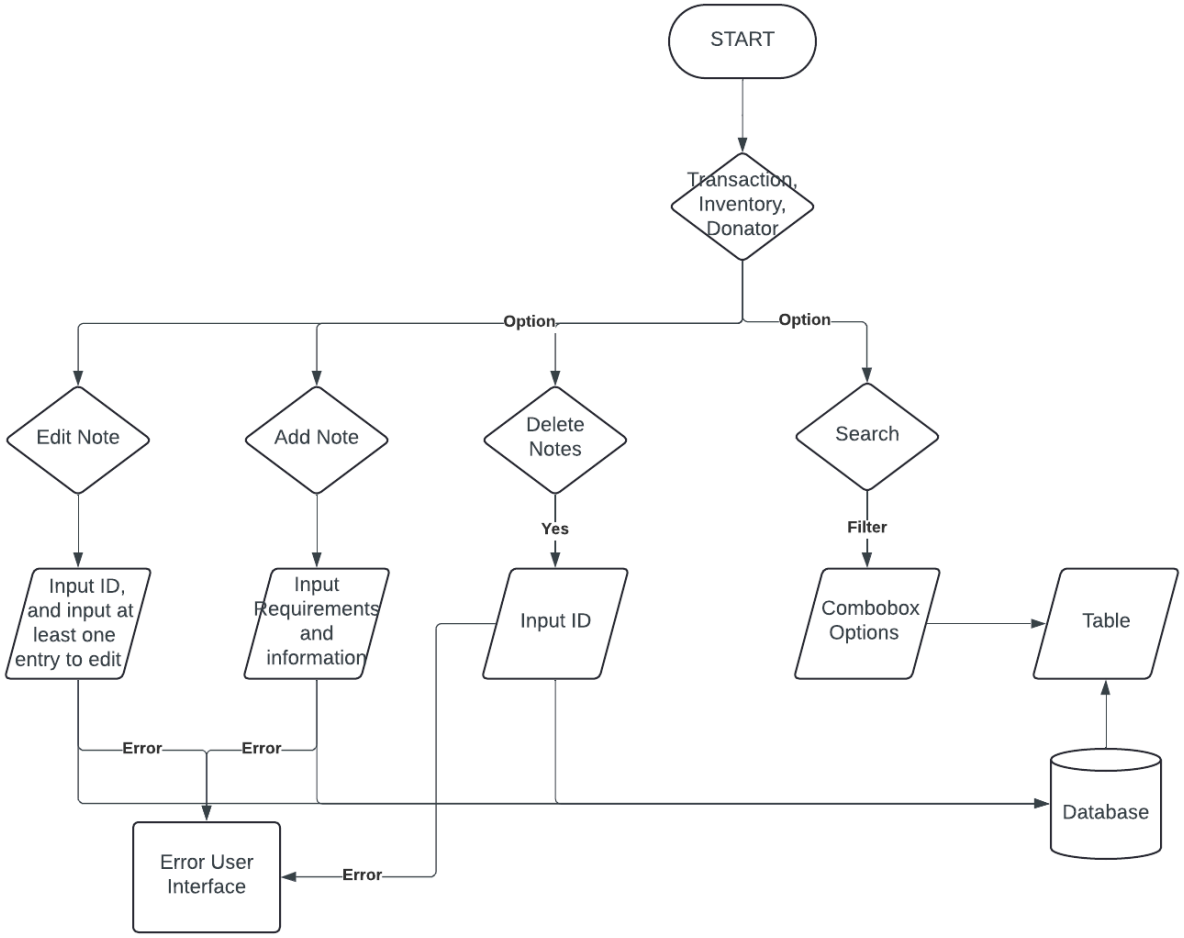


Figure 7.Main Pages Flowchart

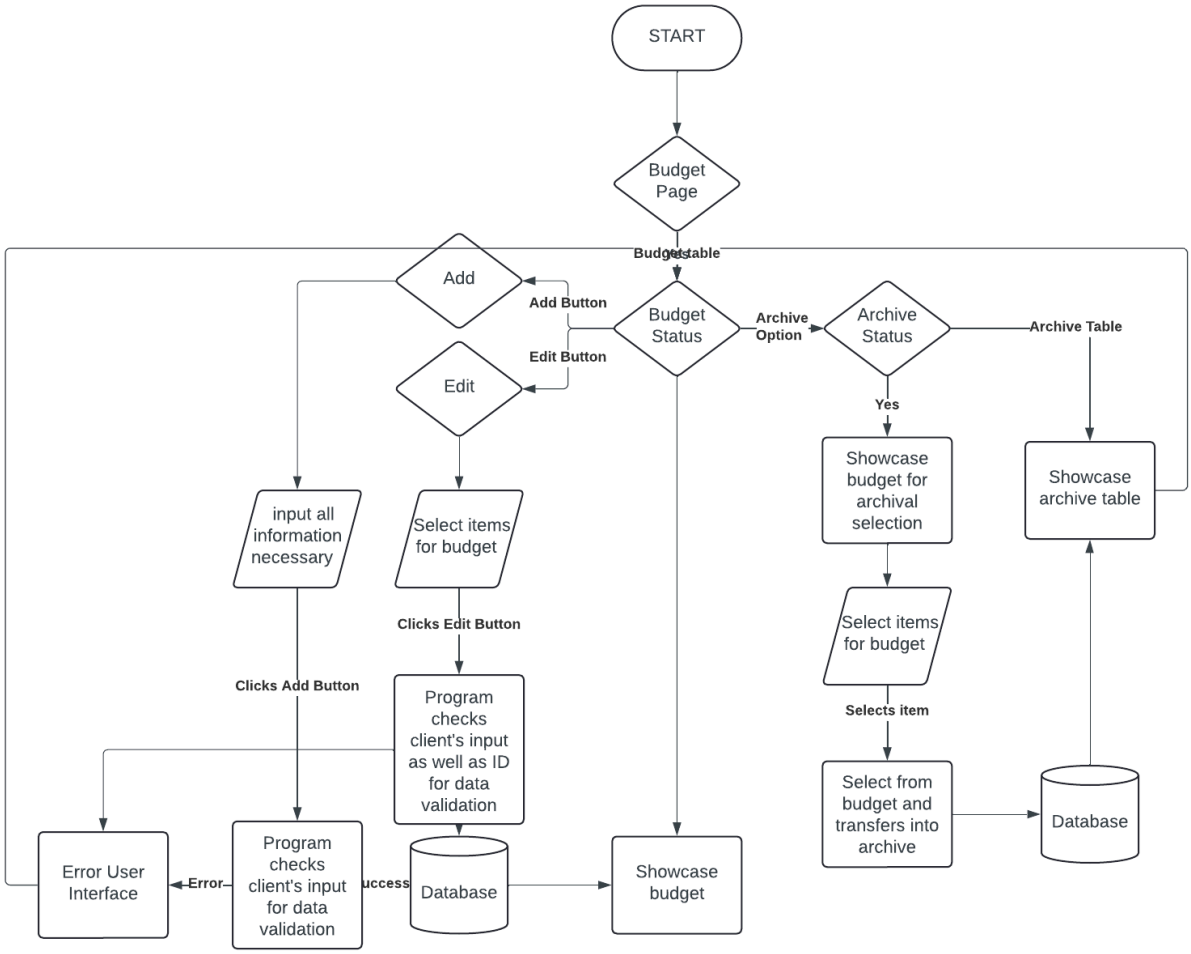

START
Budget Page
Budget Table
Add
Add Button
Budget Status
Archive Option
Archive Status
Archive Table
Edit Button
Edit
Yes
Showcase budget for archival selection
Showcase archive table
input all information necessary
Select items for budget
Clicks Edit Button
Select items for budget
Clicks Add Button
Program checks client's input as well as ID for data validation
Selects item
Select from budget and transfers into archive
Database
Error User Interface
Error
Program checks client's input for data validation
Database
Showcase budget

Figure 8. Budget Page Flowchart

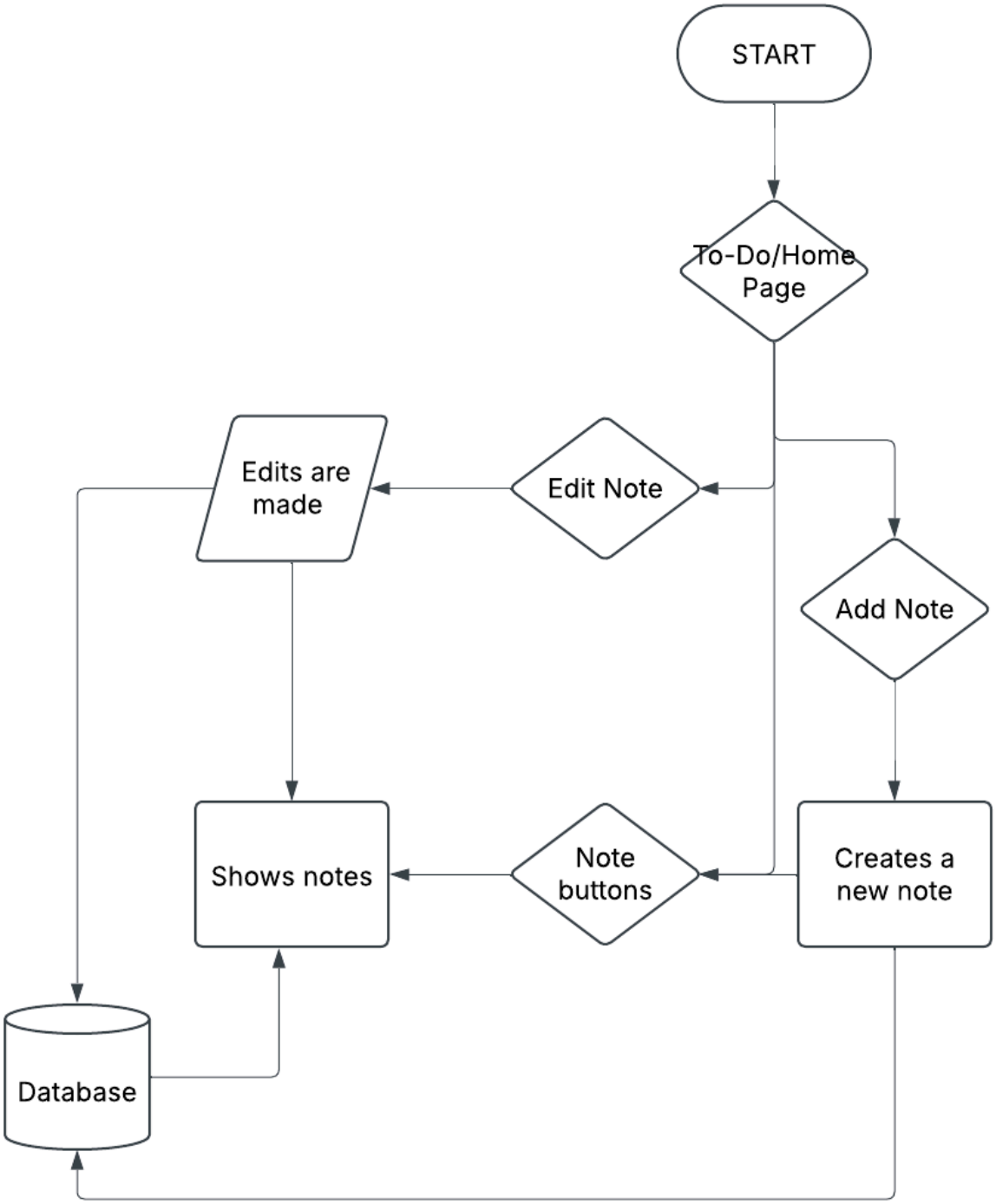
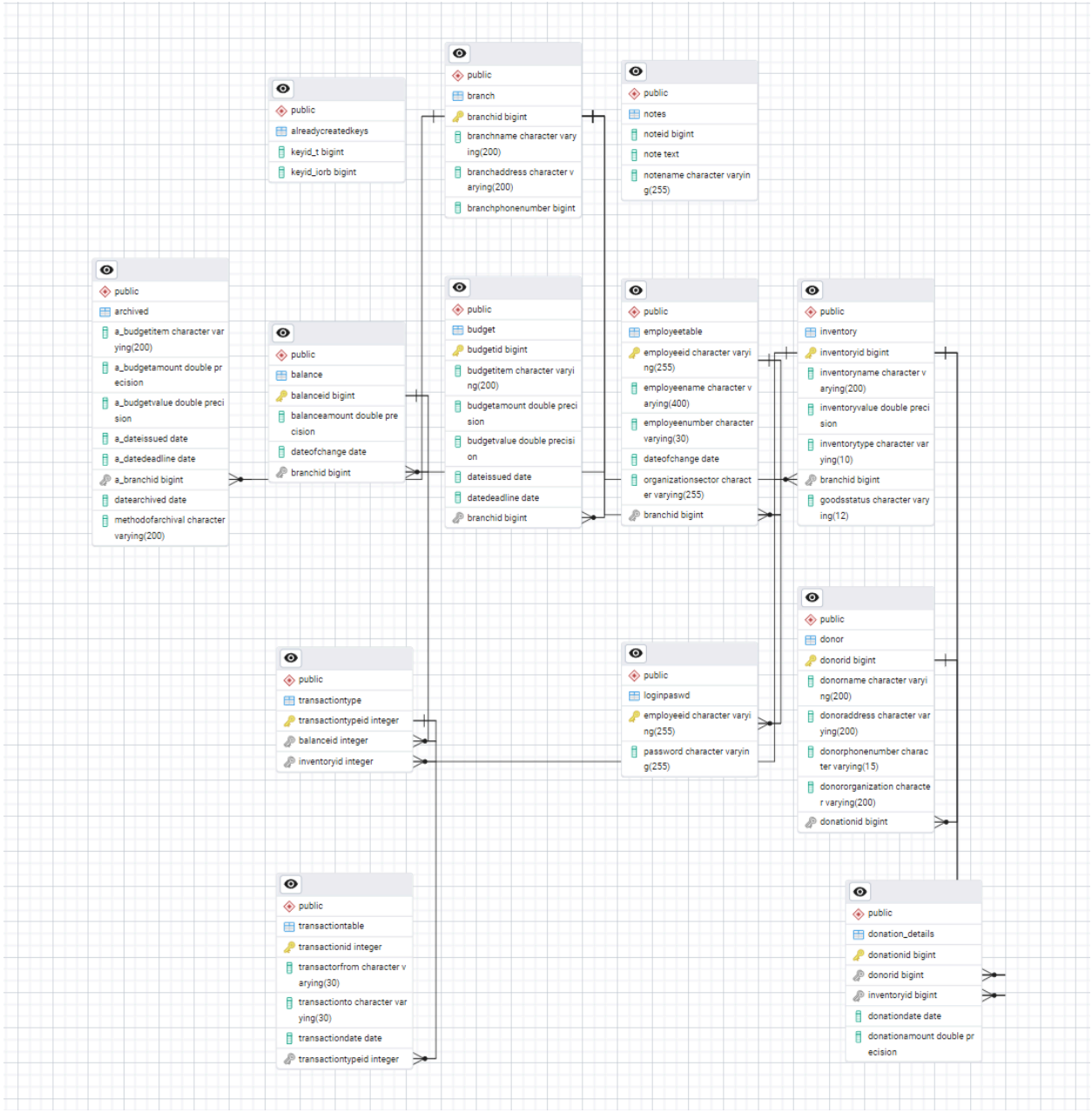


Figure 9. Notes/Home Page Flowchart

ASSETS, SALES, AND FUNDS ERD

Entity Relationship Diagram



LOGIN AND PASSWORD

Data Dictionary

Field Name	Data Type	Data Format	Field Size	Description	Example
Table Name: LOGINPASSWD					
EmployeeID (FK)	varchar	string	255	Employee's Identifier	ADMIN1
Password	varchar	string	50	Employee's personal password to log in to the program	Password123

Field Name	Data Type	Data Format	Field Size	Description	Example
Table Name: ALREADYCREATEDKEYS					
keyid_t (FK)	bigint	integer	N/A	Checks if an ID has already been created. If an ID has already been created in the transactions page, it will be found here.	19232, 12832
keyid_iorb	bigint	integer	N/A	Checks if an ID has already been created. If an ID for item or balance has already been	19232, 12832

				created in the inventory page, it will be found here.	

Field Name	Data Type	Data Format	Field Size	Description	Example
Table Name: EmployeeTable					
EmployeeID (PK)	varchar	string	255	Employee's Identifier	ADMIN1
EmployeeNumber	varchar	string	50	Employee's personal password to log in to the program	Password123
DateofChange	Date	YYYY/MM/DD	n/a	Date when employee's information was last updated	2023/10/4
OrganizationSector	varchar	string	255	Which organization sector the employee is in	'Health care', 'Precision manufacturing', 'Engineering', 'Finance/accounting', 'Information technology'
BranchID (FK)	bigint	integer	8 bytes	Identity for individual branch	1

Table Name: Branch					
branchID (PK)	bigint	integer	8 bytes	Identity for individual Branches	1
branchName	varchar	string	200	Name of specific branch	Branch Mystic Falls Eldoria
branchAddress	varchar	string	200	Address of branch	221B Castle Lane Mystic Falls Eldoria, 153467
branchPhoneNumber	bigint	integer	5-15	Branch contact number	0911 111 1111

Table Name: Donation_Details					
DonationId (PK)	Bigint	integer	8 bytes	Identity of donation	12345
DonatorId (FK)	bigint	string	200	Identity of donator	12345
InventoryId (FK)	bigint	string	200	Identity of the item within the inventory of the branch	12345
DonationDate	Date	integer	15	Donation Date	2012/12/20

Table Name: archived

a_BudgetItem	Varchar	String	200	The item archived	Toy
a_BudgetAmount	Float	Decimal	4 bytes	How many items did the item budget have originally	5
a_BudgetValue	Float	Decimal	4 bytes	The value of the budgeted item that's currently archived	$600
a_DateIssued	Date	YYYY/MM/DD	4 bytes	When was the budget issued	2020/12/13
a_DateDeadline	Date	YYYY/MM/DD	4 bytes	The set deadline for archival for the budget	2021/12/20
a_BranchId	Bigint	Integer	8 bytes	The branch of where the item originated from	1
DateArchived	Date	YYYY/MM/DD	4 bytes	The date of when it was archived	2021/12/20
MethodOfArchival	Varchar	String	200	The method in which the item was archived	Manually/Automatically

Table Name: Budgets					
BudgetId (PK)	bigint	Integer	8 bytes	Identifier for the item being budgeted to not confuse the user if there are two	12345

				items with the same name	
BudgetItem	varchar	String	200	The name of the item being budget	Toy
BudgetAmount	float	Decimal	4 bytes	How many items did the item budget have originally	5
BudgetValue	float	Decimal	4 bytes	The value of the budgeted item	$600
DateIssued	date	YYYY-MM-DD	4 bytes	When was the budget issued	2020/12/13
DateDeadline	date	YYYY-MM-DD	4 bytes	The set deadline for archival for the budget	2021/12/20
BranchId	bigint	Integer	8 bytes	The branch of where the item originated from	1

Table Name: Inventory					
InventoryId (PK)	bigint	Integer	8 bytes	Inventory identifier	12345
InventoryName	varchar	String	200	The name of the item within the inventory	Toy
InventoryValue	float	Decimal	4 bytes	The value of the item	$600
InventoryType	varchar	String	10	The type of	Asset

				inventory, whether asset, liability, or none	
BranchId(FK)	bigint	Integer	8 bytes	The branch the inventory item is stored in	1
GoodsStatus	varchar	String	7	Whether the item is bought, sold, or donated.	Sold

Table Name: transactionTable					
transactionId (PK)	bigint	Integer	8 bytes	Identification of the transaction	12345
transactorFrom	varchar	String	200	Who initiated the transaction	Stark
transactionTo	varchar	String	200	Who the transaction was for	Jarvis
transactionDate	date	YYYY-MM-DD	4 bytes	Date of when transaction was done	2012/12/20
transactionType	varchar	String	5	Cash/Item	Cash
transactionTypeId (FK)	bigint	Integer	8 bytes	The transactiontypeid simplifies inner/left join and connects transaction table to balance and inventory tables	12345

Table Name: transactionType					
transactionTypeId (PK)	bigint	Integer	8 bytes	The identifier that connects the balance table and inventory table to the transaction table	12345
balanceID	bigint	Integer	8 bytes	Identification of items within the balance table	12345
InventoryId	bigint	Integer	8 bytes	Identification of items within the inventory table	12345

Table Name: Balance					
BalanceID (PK)	bigint	Integer	8 bytes	Identification items within the balance table	1
BalanceAmount	float	Decimal	4 bytes	How much the balance is	$300
DateOfChange	date	YYYY-MM-DD	4 bytes	The date of exchange of balance	2023/12/21
BranchId	bigint	Integer	8 bytes	Where the balance is stored, or where it is supposed to be in	1

Table Name: Donator					
DonatorID (PK)	bigint	Integer	8 bytes	Identification of donator	12345
DonatorName	varchar	String	200	Donator's name	Elliot Alderson
DonatorAddress	varchar	String	200	Donator's address	221B Castle Lane Mystic Falls Eldoria, 153467
DonatorPhoneNumber	varchar	String	15	Donator's phone number	12345678
DonatorOrganization	varchar	String	200	Which organization the donator is in	Allsafe
DonationID	bigint	Integer	8 bytes	The donation that the donator sent	12345

Table Name: note					
noteid (PK)	bigint	Integer	8 bytes	What note it is	1
note	varchar	Text	Variable	What is in the note/message	Hello everyone! I hope you are doing well today.
notename	varchar	Text	255	The name of the note (MUST BE NOTE ID)	1

TEST PLAN

The user undergoes multiple testing phases to ensure that the solution provided is refined, ensuring satisfaction of the client in meeting requirements, through assessment that involves client interaction. The client will interact with specific functionalities through black-box testing, provided that feedback is produced to refine the program along the way.

Test No.	Success Criteria	Test Actions	Method of Testing and Expected Test Results
1	A notes page facilitates communication between clients at different points of time to ensure information is passed despite times of access	Ensure that notes page/ homepage's functionality regarding adding,editing, and deleting notes showcase proper query handling with saving notes and smooth transition of user interface in displaying the notes	If the user wishes to utilize the notes functionality within the notes page, the user is provided with options to create a note by clicking on the add button. It then transitions into an empty note on the side, which can only be edited by pressing the edit button, taking them to the edit status. If the user chooses to delete, it can also be accessed in the edit status.
2	Calculations within the calculator are saved while the program is on	Ensure that the calculator must save the calculations of the user within the program itself as long as the program is on.	After inputting their own calculation or equation into the calculator which provides a correct answer, it must then save the results of that equation and the equation itself into a box that provides the user with that specific data. However, if the user closes the program, then all items within will reset to ensure security and privacy of the user's actions.

3	The program allows navigation based on specific tasks	Ensure that the tabs function functionally provides the page the user requests.	The program must open with the notes /home page open. If the user attempts to click on a button above that showcases tab buttons, it leads the user into the specific page that is requested by the user themselves.
4	Logging in is facilitated to provide information security for the client	Ensures that the log-in page functionality locks the user if no access	If the user has logged in, user has access to every part of the program, however if not, then they are locked out of the program until login access is provided.
5	The client can view the organization's information through tables	Ensures that the budget page showcases the information presented in a table	When the budget page is opened, the user has access to all information, provided they click the proper buttons. If the user clicks the archive button, it must show the archive however if it showcases the budget page mainly.
6	The program must provide clients with the option to delete data of no relevance	Ensure that the main pages (inventory, donator, transaction) allow deletion of elements	If the user deletes an element in a program, it must ensure that all records are deleted alongside it. If the element is linked to another table however, and it is dependent on that table such as how a transaction item is reliant on a donator item, then the program will produce an error.
7	The program must allow the client to add information depending on the task at hand	Ensure that the main pages (inventory, donator, transaction) allow the addition of new elements when fulfilling all requirements within the combobox).	If the user wishes to add a new element, then the program will add all information if all combobox sections are inputted accordingly by the user. However, if the program has either empty items within a combobox section or if the item is not in the expected format, it will produce an error.
8	The program ensures edits can be made by the client to edit necessary	Ensures that editing an item is done by inserting the item ID	An item can be identified by the client which can be referenced in their attempts to edit or delete an item. If successful, then the item with an ID is deleted or

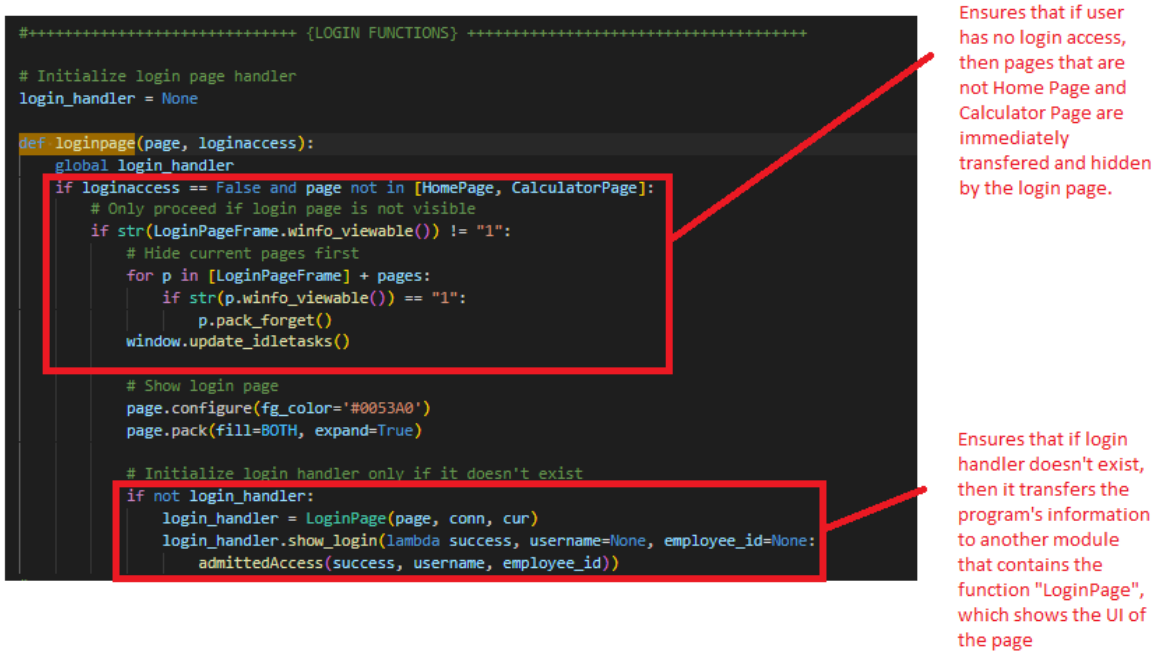
	information		edited, however if not, then it would produce an error saying that an ID does not exist or is not created.
9	The client can filter information through search function for easy access of information	Ensure that the search function provides the data that the user requests, with multiple options provided that further filter the search function.	The user is given multiple options to search, where depending on what combobox option they are given, then it will provide the user with their request. However, if the item being searched does not exist, then it will provide the user with an error stating that it does not exist.
10	All client's items can be referenced with a uniquely generated ID	Client's items are automatically generated to call out an item	The client identification generator ensures that the id created can be used to call out items when utilizing a specific add edit or delete function.
11	The client can archive information for better finance management	Ensure that the user can test if archive function allows the selection and the archival of items from the current budget page.	If the client archives the information, then they must first be able to select which item they want to archive within the budget page, then once selected and entered, will then move directly from the budget page onto the archival page.
12	All client data must be linked to the client's branches	ll Ensure that ALL items contain the branch in which is affected by the item.	In order to allocate resources and budgets, each item is connected to each other by branch, ensuring that there is a common theme of relation within them. The expectation is that if the user adds a value or an element, then it is connected by a branch. However, if the user forgets to include a branch, the query will not go through.
13	The program saves the client's progress when multitasking	Ensure that if a user is working on one thing, then moves onto a different page, and then goes back, it saves the progress of the user.	If a user is working on something currently, however, needs to access another part of the program, then the program must save the progress of the user.

14	The program utilizes error handling for managing complex tasks	Ensure that all errors are handled by the error handle module, popping up a user interface whenever there is an error in the program.	If a user produces an error, all errors are handled by an error handler module. However, if the user meets a successful criteria, then the same user interface will be used to output a successful interface, rather than an error interface.

CRITERION C

1. BUDGET PROGRAM

A. LOGIN



```
#++++++++++++++++++++++++++++++ {LOGIN FUNCTIONS} ++++++++++++++++++++++++++++++++++++

# Initialize login page handler
login_handler = None

def loginpage(page, loginaccess):
    global login_handler
    if loginaccess == False and page not in [HomePage, CalculatorPage]:
        # Only proceed if login page is not visible
        if str(LoginPageFrame.winfo_viewable()) != "1":
            # Hide current pages first
            for p in [LoginPageFrame] + pages:
                if str(p.winfo_viewable()) == "1":
                    p.pack_forget()
            window.update_idletasks()

            # Show login page
            page.configure(fg_color='#0053A0')
            page.pack(fill=BOTH, expand=True)

            # Initialize login handler only if it doesn't exist
            if not login_handler:
                login_handler = LoginPage(page, conn, cur)
                login_handler.show_login(lambda success, username=None, employee_id=None:
                    admittedAccess(success, username, employee_id))
```

Ensures that if user has no login access, then pages that are not Home Page and Calculator Page are immediately transfered and hidden by the login page.

Ensures that if login handler doesn't exist, then it transfers the program's information to another module that contains the function "LoginPage", which shows the UI of the page

The loginpage function ensures that the LoginPage module can communicate from the main-tester.py file, where as long as *loginaccess* is false while the user wants to access pages that are unauthorized, then the *loginpage* function will send necessary information for the module to show the actual login page, ensuring that users may see the login page before access.

```
    # Check if trying to access restricted page without login
    if loginaccess == False and page not in [HomePage, CalculatorPage]:
        # Check if login page exists and is not visible
        login_exists = login_handler is not None
        login_visible = str(LoginPageFrame.winfo_viewable()) == "1"

        if not login_visible:
            if current_page:
                current_page.pack_forget()
                window.update_idletasks()
            # If login handler exists, just show the frame without creating new handler
            if login_exists:
                LoginPageFrame.configure(fg_color='#0053A0')
                LoginPageFrame.pack(fill=BOTH, expand=True)
                login_handler.show_login(lambda success: admittedAccess(success))
            else:
                loginpage(LoginPageFrame, loginaccess)
        return  # Exit early to prevent showing other pages
```

The *loginaccess* variable is a boolean that allows the user to maintain access for a session as long as the program is open and not closed. As long as the user has not inputted the proper credentials, then the attempt to access the program is locked by the login page, ensuring security.

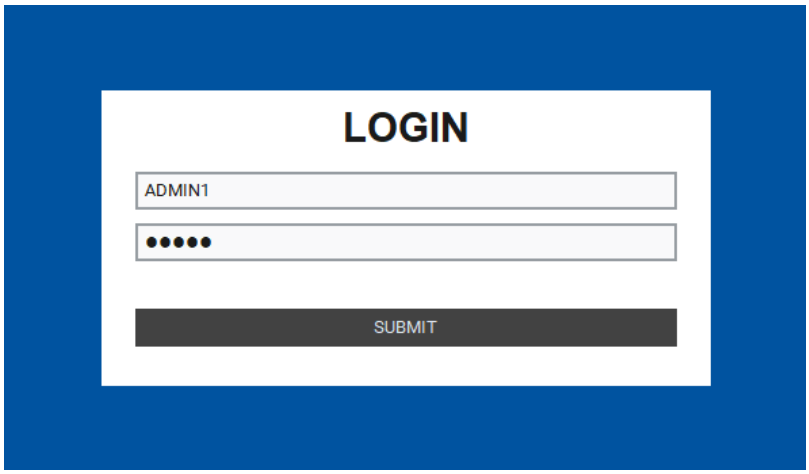


Figure 1. Login Page

```
def admittedAccess(success, username=None, employee_id=None):
    global loginaccess, HomePagePost
    if success:
        loginaccess = True

        # Set user info and log successful login
        set_current_user(username or "Unknown", employee_id or "UNKNOWN")
        activity_logger.log_login(
            employee_id,
            username,
            True,
            "User logged in successfully"
        )
```

The function *AdmittedAccess* ensures security for the actual users of the database, where as long as the user logs in with proper credentials, then a ui may pop that showcases that they may have access to the sensitive data.

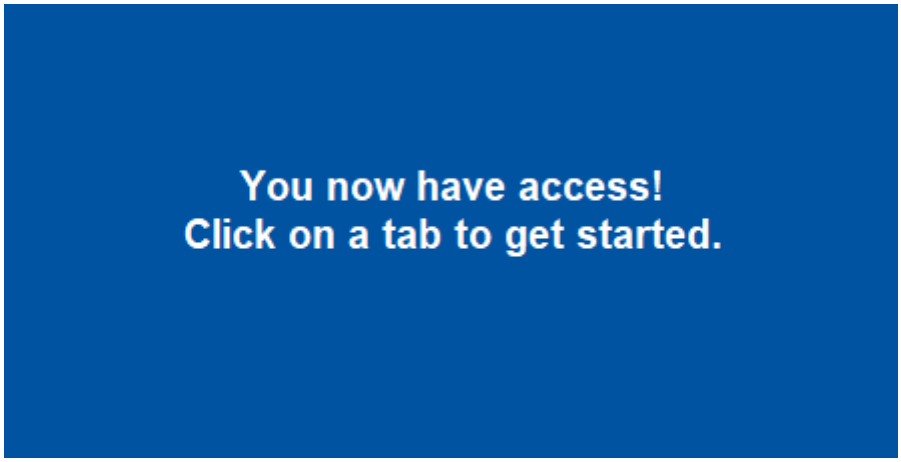


Figure 2. Login Success

B. Error Handler

Snippet of code from InventoryPage:

```
execute_safe_query(cur, "ROLLBACK")
show_error_window("Inventory not found or must be sold before deletion")
```

Snippets from errorHandler.py:

Showcases the UI with customized error message

```
def show_error_window(message):
    try:
        error_window = CTkToplevel()
        error_window.title("Error")
        error_window.geometry("400x150")
        error_window.resizable(False, False)

        # Center the window
        error_window.withdraw()  # Hide window initially
        error_window.after(0, lambda: center_error_window(error_window))

        # Error message
        CTkLabel(error_window, text="Error occurred:",
                 font=("Helvetica", 12, "bold")).pack(pady=10)
        CTkLabel(error_window, text=message,
                 wraplength=350).pack(pady=10)

        # OK button
        CTkButton(error_window, text="OK",
                  command=error_window.destroy,
                  width=100).pack(pady=10)

        # Make window modal
        error_window.transient()
        error_window.grab_set()
        error_window.focus_set()

    except Exception as e:
        print(f"Failed to show error window: {str(e)}")
        print(f"Original error message: {message}")
```

Provides safe query execution through try-catch function, where very specific queries get passed through with or without parameters. If function succeeds, then the program will continue to run however if not, then it will showcase an error, therefore handling the error and informing the user.

```
def execute_safe_query(cursor, query, params=None):
    try:
        if params:
            cursor.execute(query, params)
        else:
            cursor.execute(query)
        return True, None
    except psycopg2.Error as e:
        return False, f"Database Error: {e}"
    except Exception as e:
        return False, f"Error: {e}"
```

The errorHandler file ensures that SQL query errors pop up when attempting to manage the database, ensuring safe transactions as the user manages items through the program, ensuring the prevention of issues through knowing what went wrong and how to fix it.

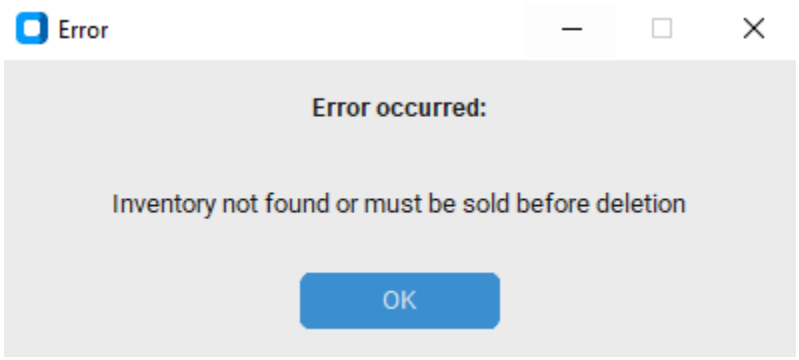


Figure 3: Error message example

C. DATA STRUCTURES

Arrays

Page Frame Storing

Arrays are used for storing all accessible pages, as seen in this array:

```
pages = [HomePage, TransactionsPage, DonatorPage, InventoryPage, BudgetPage, CalculatorPage]
```

The pages within the array are listed as their own widgets with individual UI:

```
HomePage = CTkFrame(window, corner_radius=0)
setattr(HomePage, 'post_flag', 0)
TransactionsPage = CTkFrame(window, corner_radius=0)
DonatorPage = CTkFrame(window, corner_radius=0)  # Changed from ClientPage to DonatorPage
InventoryPage = CTkFrame(window, corner_radius=0)
BudgetPage = CTkFrame(window, corner_radius=0)
CalculatorPage = CTkFrame(window, corner_radius=0)
LoginPageFrame = CTkFrame(window, corner_radius=0, bg_color='#0053A0')
```

Arrays are used to store important data that can and will be used later on, which ensures that specific scenarios such as the traversal of the user from different pages are stored in arrays, allowing ease of access and flexibility per request of the user.

Stacks

INITIALIZATION:

```
EquationStack = []
ResultsStack = []
```

A stack is a linear data structure that stores items through Last In First Out (LIFO) manner. Python's built-in data structure list can be used similar to a stack.

PUSH/APPEND:

```
print("Result:", finalresult)
ResultsStack.append(finalresult)
OutputCalculations.configure(state="normal")
OutputCalculations.delete(0, END)
OutputCalculations.insert(0, str(finalresult))
OutputCalculations.configure(state="readonly")
```

```
expression = OutputCalculations.get()
EquationStack.append(expression)
print("Expression:", expression)
```

Items that wish to be stored in a stack are "pushed" onto it. The calculator that uses the stack function stores both equation and results in their respective stacks to ensure that the user may view the history of their inputs (expressions) and the outputs (results given by the calculator).

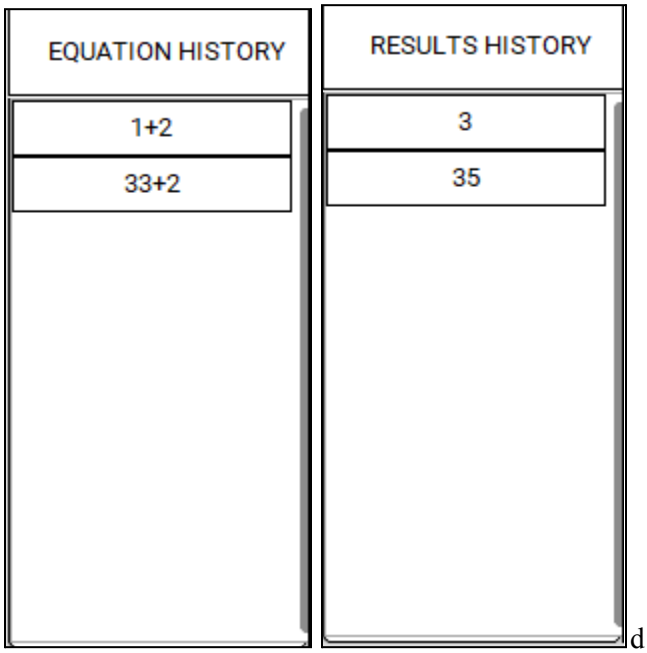
POP:

```
for I in EquationStack:
    EquationPop = str(EquationStack.pop())
    print(EquationPop)
    NewEquationArray.append(EquationPop)
```

```
for I in ResultsStack:
    ResultsPop = str(ResultsStack.pop())
    print(ResultsPop)
    NewResultsArray.append(ResultsPop)
```

Items from the top of the stack are then popped first, following LIFO, which then gets stored into the respective linear arrays to be used for the equation/results history.

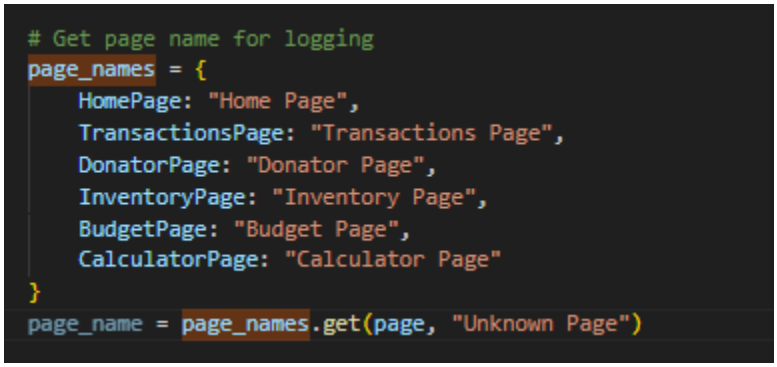
Stacks were used to ensure that the elements stored in the array were of the same data type. This can be seen all throughout the code, where simple objects such as *ResultsStack.pop()* or *finalresult* were converted into string. While arrays in python can contain different types of data, it may cause issues if certain functions of the code require the use of only one data type, making the use of stacks effective for proper functionality for the end user.



d

Figure 4: Equation and Results history example

Dictionaries (Hash Maps)

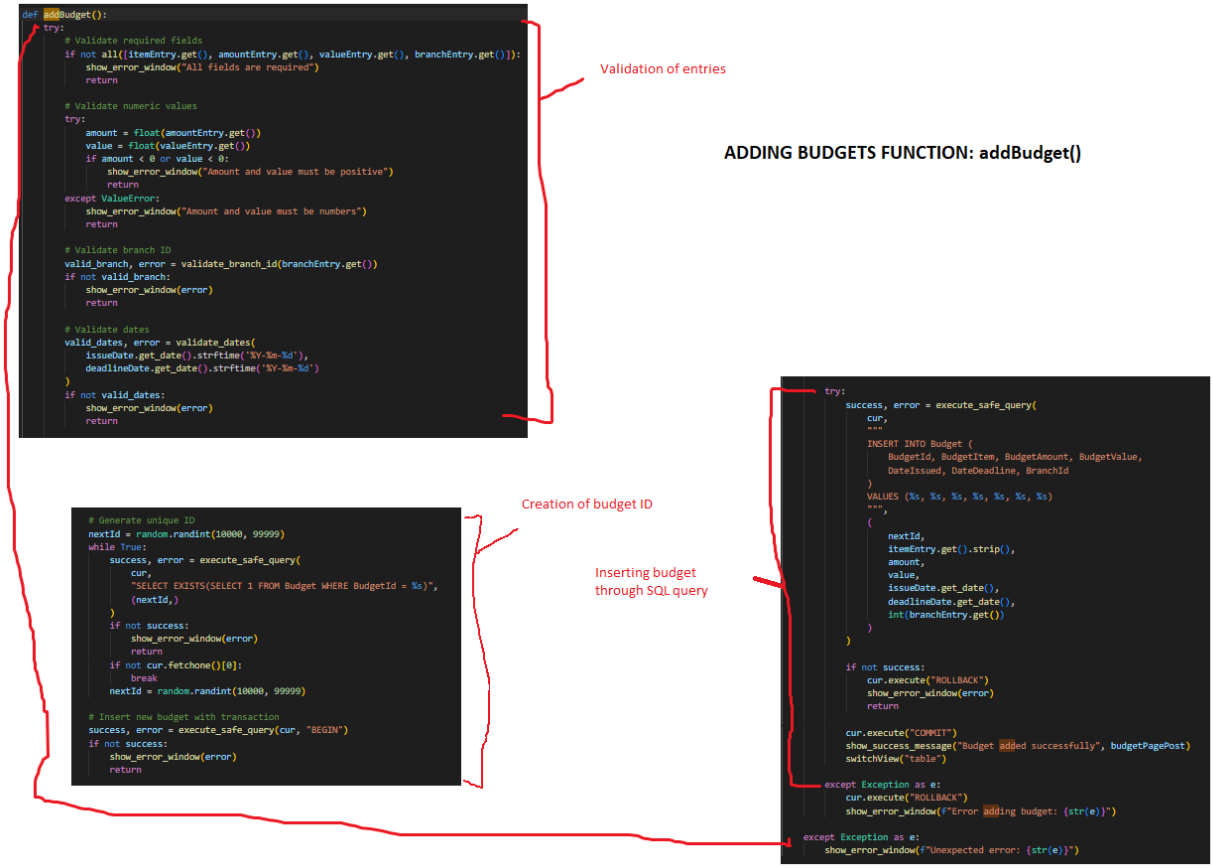


```
# Get page name for logging
page_names = {
    HomePage: "Home Page",
    TransactionsPage: "Transactions Page",
    DonatorPage: "Donator Page",
    InventoryPage: "Inventory Page",
    BudgetPage: "Budget Page",
    CalculatorPage: "Calculator Page"
}
page_name = page_names.get(page, "Unknown Page")
```

By creating the entry seen above, it allows the program to map certain objects to their own display names. In consideration of the code itself, the use of hash maps ensures that items are logged when switching pages, allowing the admin to keep track of what is being accessed.

D. BUDGET (ADD, EDIT, ARCHIVE)

Add



Adding budgets contain three main sectors: validation, ID creation, and querying. Validation ensures that every item that is inserted (such as the amount of the budget or the value) is correctly identified and follows proper conventions of the program, where items within if statements and try except statements are identified as either successful or not. If they are, then it will move onto the next step, which is ID creation. This section attempts to create an ID for the budget item, which is fully automated. Once the budget ID is created, the querying section ensures that all the validated items are saved in the table correctly, through the use of INSERT INTO. This allows the user to add their requested budgets into the program, saving their budgets for future use.

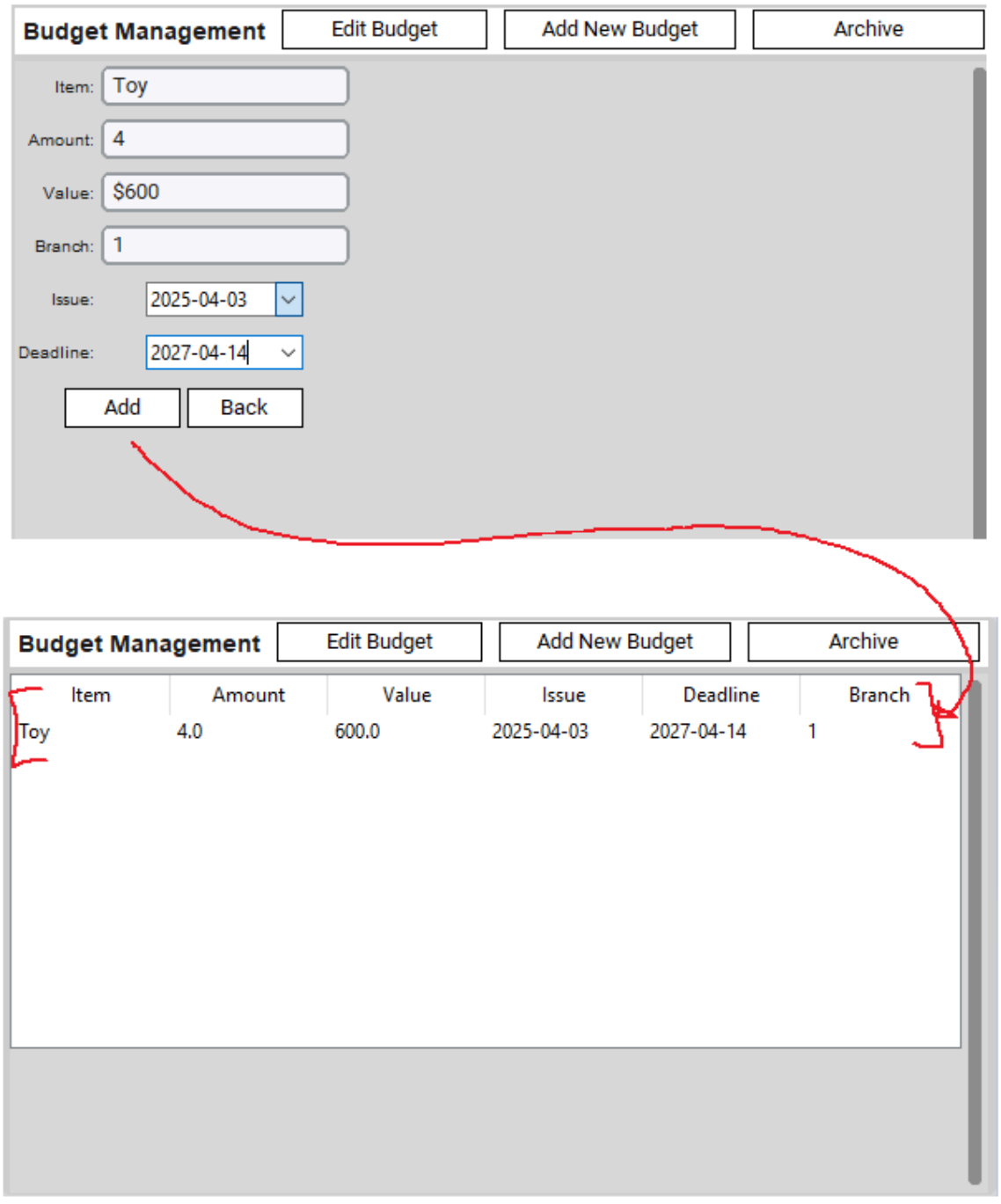


Figure 5. Adding budget items

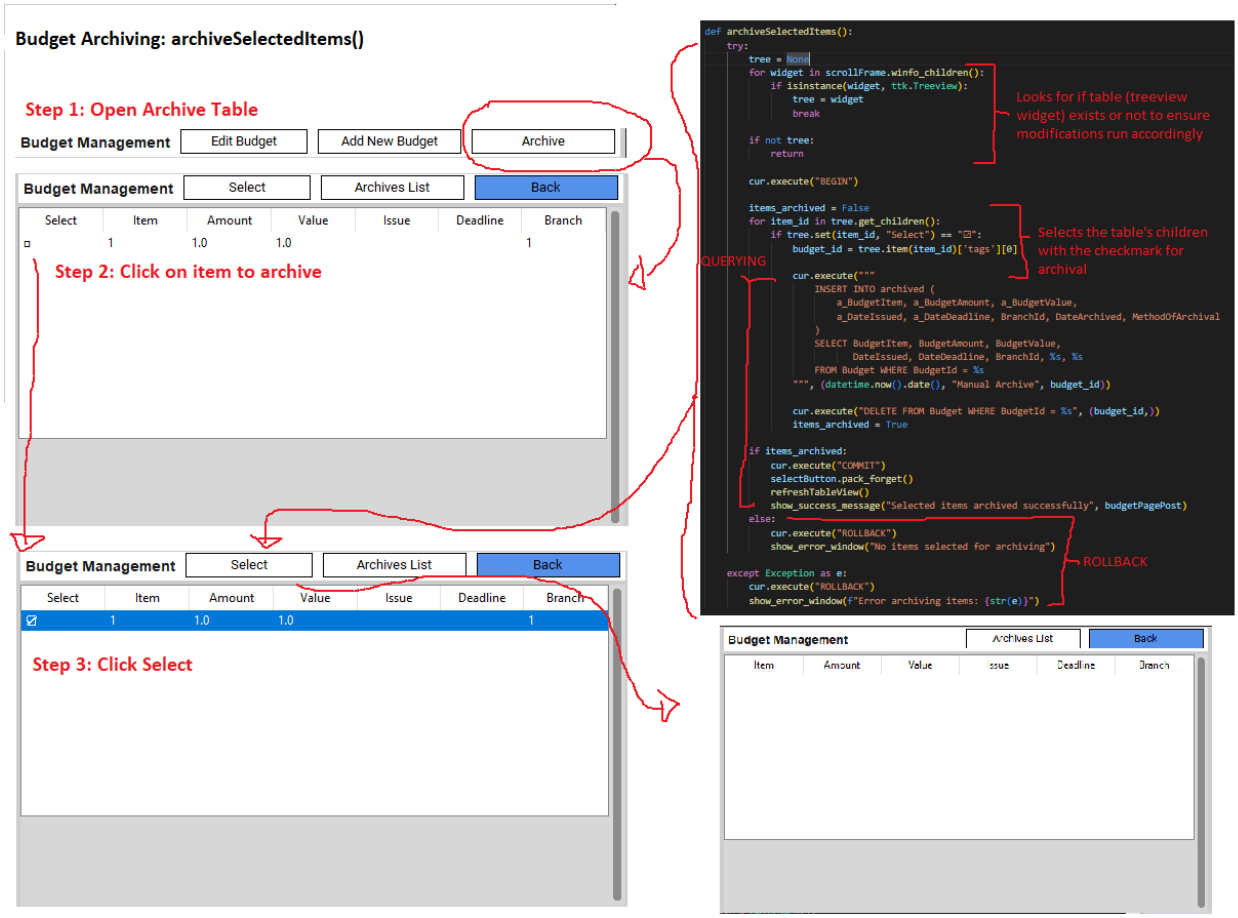
Edit

```
def editBudget():
    try:
        if not all([budgetIDEntry.get(), itemEntry.get(), amountEntry.get(), valueEntry.get(), branchEntry.get()]):
            show_error_window("All fields are required")
            return

        success, error = execute_safe_query(
            cur,
            """
            UPDATE Budget
            SET BudgetItem = %s, BudgetAmount = %s, BudgetValue = %s,
                DateIssued = %s, DateDeadline = %s, BranchId = %s
            WHERE BudgetId = %s
            """,
            (
                itemEntry.get(),
                float(amountEntry.get()),
                float(valueEntry.get()),
                issueDate.get_date(),
                deadlineDate.get_date(),
                int(branchEntry.get()),
                int(budgetIDEntry.get())
            )
        )

        if not success:
            show_error_window(error)
            return

        conn.commit()
        show_success_message("Budget edited successfully", budgetPagePost)
        switchView("table")

    except ValueError:
        show_error_window("Please enter valid numbers for Amount, Value, and Branch ID")
    except Exception as e:
        show_error_window(f"Unexpected error: {str(e)}")
        conn.rollback()
```

Edit follows a different but similar structure in the sense where it still utilizes try and except methods for error handling, but in replace of validation, attempts to query first as seen in the UPDATE Budget part of the code, and if an issue arises, will rollback in order to still allow the user to continue to edit, without

having to be blocked by postgres providing the information that the transaction will not continue until the original query has gone through. Editing allows the user ease of access in managing their current budgets.

Archive

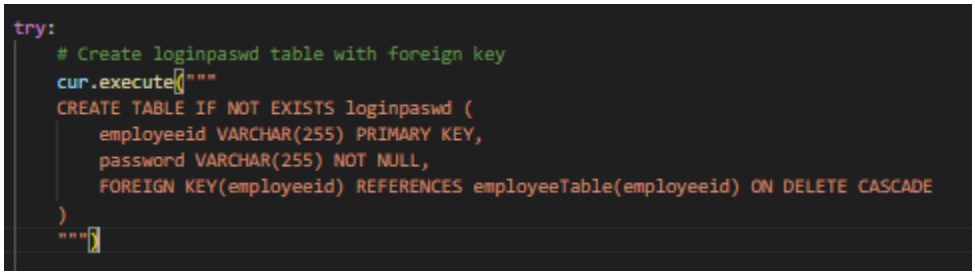


The code provided mainly focuses on step 3 of archiving, which is clicking the Select button. The code first attempts to see if the archive table exists. However if it does not, then it will return. The will also attempt to traverse through utilizing a for loop also to check the items the user has selected. Once done, this will then move onto archiving all items through the utilization of INSERT INTO into the archived table, while deleting the item in the budget table. This allows the user to archive selected items that are not of use, ensuring transparency that the budget is handled accordingly.

2. STANDARD QUERY LANGUAGE - SQL FUNCTIONS

CREATE, READ, UPDATE, DELETE (CRUD)

CREATE - Fault Tolerance



```
try:
    # Create loginpaswd table with foreign key
    cur.execute("""
    CREATE TABLE IF NOT EXISTS loginpaswd (
        employeeid VARCHAR(255) PRIMARY KEY,
        password VARCHAR(255) NOT NULL,
        FOREIGN KEY(employeeid) REFERENCES employeeTable(employeeid) ON DELETE CASCADE
    )
    """)
```

This code in the loginpage.py file attempts to create a *loginpaswd* table, which is the table named for users that can login into a program, containing the username and password of the user. The use of this code ensures the ability of fault tolerance to fix itself given that if in the case that the database gets wiped, the program still runs accordingly, as well as security from locking users from accessing the program.

READ - Left Join

```
try:
    search_value = SearchEntry.get().strip()
    params = {'search_value': search_value}

    base_query = """
    SELECT
        i.InventoryId,
        i.InventoryName,
        i.InventoryValue,
        i.InventoryType,
        i.BranchId,
        i.GoodsStatus,
        g.BranchName
    FROM Inventory i
    LEFT JOIN goodwillbranch g ON i.BranchId = g.BranchId
    WHERE 1=1
    """

    search_conditions = {
        "Inventory ID": "i.InventoryId = %(search_value)s",
        "Inventory Name": "LOWER(i.InventoryName) LIKE LOWER(%(like_value)s)",
        "Inventory Type": "LOWER(i.InventoryType) LIKE LOWER(%(like_value)s)",
        "Branch ID": "i.BranchId = %(search_value)s"
    }
```

By utilizing SELECT FROM in conjunction with LEFT JOIN, this allows the user to view specific items of the database from any page that includes the search function (transaction, inventory, or donator's page) while remaining true to the first table. This allows further accessibility for the user to access information they require, without having to traverse through pages of information at a time to find the correct item.

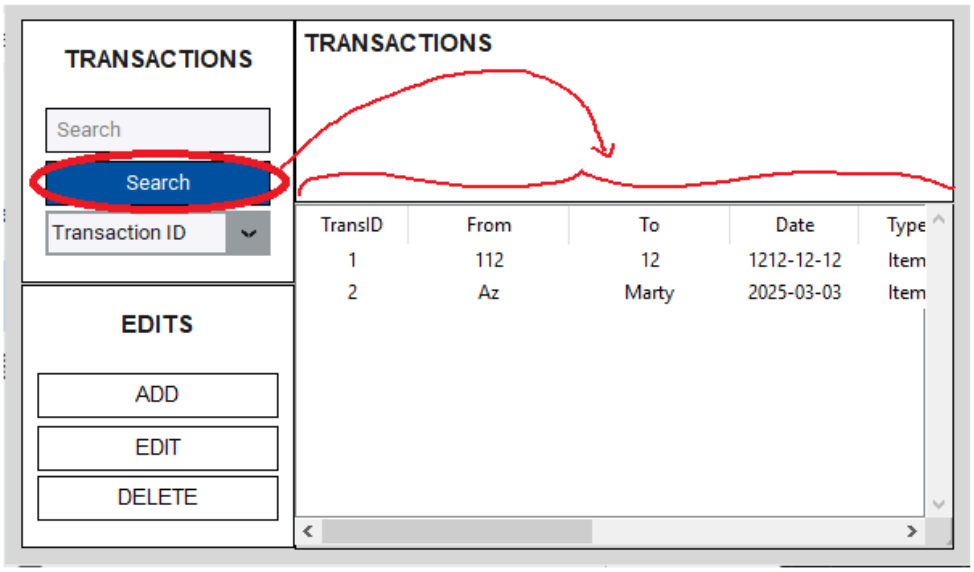


Figure 6. Search information using LEFTJOIN

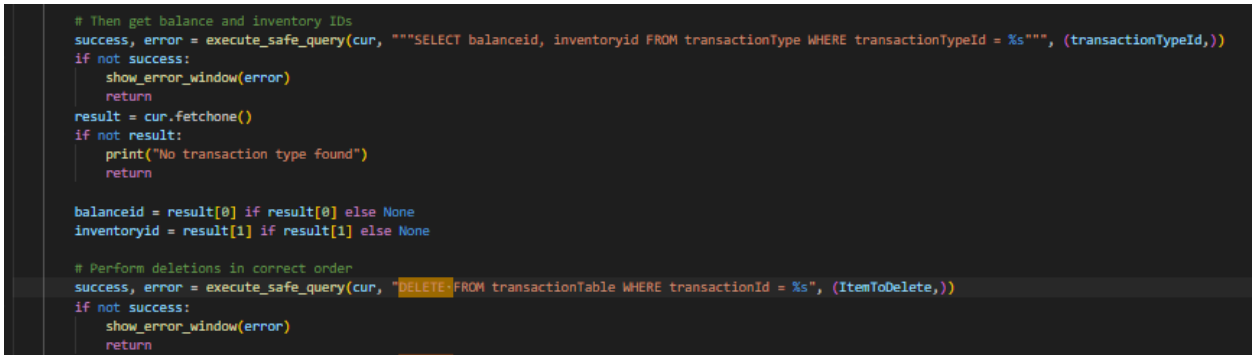
UPDATE - Nested IF statements for Error Handling

```
if BranchIDFlag:
    # Verify branch exists
    success, error = execute_safe_query(cur, """
        SELECT EXISTS(
            SELECT 1 FROM goodwillbranch
            WHERE BranchId = %s
        )
    """, (BranchIDHolder,))
    if not success:
        show_error_window(error)
        return
    if cur.fetchone()[0]:
        success, error = execute_safe_query(cur, """
            UPDATE Inventory
            SET BranchId = %s
            WHERE InventoryId = %s
        """, (BranchIDHolder, inventory_id))
        if not success:
            show_error_window(error)
            return
        updates_made = True
    else:
        execute_safe_query(cur, "ROLLBACK")
        show_error_window("Invalid Branch ID")
        return
```

The update functionality which can be mostly seen in pages that include the edit function is used to change or update items within a table. For example, within the code that can be found in the inventory page, because what the code attempts to do is change the branch of where an item is from, it first has to

search if the code actually exists through the use of Nested IF statements. By doing so, this minimizes the risk of receiving a bad output, therefore allowing the user to minimize the defects and errors (which is still managed by error handling) while updating their request.

DELETE - Try Catch Exceptions with Error Handling



```
# Then get balance and inventory IDs
success, error = execute_safe_query(cur, """SELECT balanceid, inventoryid FROM transactionType WHERE transactionTypeId = %s""", (transactionTypeId,))
if not success:
    show_error_window(error)
    return
result = cur.fetchone()
if not result:
    print("No transaction type found")
    return

balanceid = result[0] if result[0] else None
inventoryid = result[1] if result[1] else None

# Perform deletions in correct order
success, error = execute_safe_query(cur, "DELETE FROM transactionTable WHERE transactionId = %s", (ItemToDelete,))
if not success:
    show_error_window(error)
    return
```

This code attempts the same procedure as the updating query, where it attempts to search by first selecting the item to delete with SELECT and FROM to point out which table it is in. If it is successful, then the item being asked to delete will delete as seen with the four DELETE FROM queries.With the try exception technique, it allows the user to delete the requested information without having to come across errors they may not understand, and with the help of error handling technique, ensures that they understand the error they may encounter.

CRITERION E

Client Feedback:
Overall useful but can do with further functionalities

Success Criterion	Progress Accomplishment	Summary/Feedback
A to-do page facilitates communication between clients at different points of time to ensure information is passed despite times of access	Met	To-do list is present in the home page, allowing the client to add edit or delete notes.
Logging in is facilitated to provide information security for the client	Met	When navigating to a sensitive page, it is locked by a login page.
The program allows navigation based on specific tasks	Yes	The tabs allow page navigation depending on what the client aims to do.
The client can filter information through search function for easy access of information	Met	A search function was implemented to filter and search through the data.
The client can view the organization's information through tables	Met	Information of the organization can be viewed easily in tables that can be pulled by clicking the search button.
The program must delete provide clients with the option to delete data of no relevance	Met	In the main information pages, the client can delete data by referencing the ID and inputting it in.
The program must allow the client to add information depending on the task at hand	Met	The client can add items provided that it follows the requirements to add each information.
The program ensures edits can be made by the client to edit necessary information	Met	The user can edit information by referencing the ID of that item, then provide at least one change to that item.
The program saves the client's	Met	The program saves the progress

progress when multitasking		whenever the user changes pages, allowing the user to multitask without worrying about the page resetting.
All client's items can be referenced with a uniquely generated ID	Met	All items added by the user generate an ID, and cannot be changed unless the item is deleted.
The client can archive information for better finance management	Met	For the budget page, the information that is manually or automatically archived
All client data must be linked to the client's branches	Met	Each item other than the notes pages require the client to input a branch to ensure organization of all items.
The program utilizes error handling for managing complex tasks	Met	For error handling, each complex task such as database manipulation through queries use error handling to ensure that if an issue occurs, the program knows how to handle the issue.
Calculations within the calculator are saved while the program is on	Met	Reduces client's stress in typing lengthy formulas within the calculator.

Recommendations

1. Ensure that more filtering options are available for the client to search
2. Deadline and reminders would be most appropriate for the client to ensure everything is done on time
3. Implement admin and employee access, similar to an admin panel that may host the user interface for changelogs in the program, permissions to access the program, employee/user information editing, etc. for further security.