

# Design Report

## Coursework 03:

### Enhanced Personal Finance Tracker (GUI Implementation with Tkinter and OOP)

**To** : Head Lecturer, SD1, IIT

**From** : P. A. Ashane Navith Perera

**IIT ID** : 20232667

**UoW ID** : w2082259

**Date** : 28/04/2024

**Subject** : Software Development 1

**Batch** : Jan 23/24

**Purpose** : The purpose of this report is to provide a comprehensive overview of the design aspects pertaining to the graphical user interface (GUI) elements of the enhanced financial tracker application. This report outlines the design, rationale, structure, and functionality of the GUI components developed using Tkinter and ttkbootstrap. It aims to elucidate the design choices made, highlighting the design principles that were considered when making these decisions. Moreover, the integration of OOP principles to enhance modularity, usability, and maintainability of the application is also covered in this report.

Additionally, the report highlights the implementation of features such as data loading, search functionality, and sorting capabilities, ensuring a user-friendly and robust interface for effective financial management.

<b>Executive Summary.....</b>	<b>3</b>
<b>GUI Design.....</b>	<b>4</b>
Wire Frame.....	4
Hierarchy Of Elements and Widgets.....	5
Visual Design Elements.....	6
Color Palette Visualisation.....	8
Bauhaus Art.....	9
The BIG Font.....	10
Icons.....	11
<b>Designing Classes for the GUI.....</b>	<b>13</b>
Class: Custom Main Window.....	14
Class: Custom Frame With Border.....	15
Class: Custom Frame.....	15
Class: Pop Up Window.....	16
Class: Custom Treeview.....	17
Class: Custom Menu Button.....	19
Class: Custom Entry Widget.....	20
Class: Custom Back Button.....	20
Class: Custom Search Button.....	21
<b>Testing the Program.....</b>	<b>24</b>
Test Summary.....	24
Testing the GUI elements.....	24
Testing the loading of transactions from a JSON file.....	25
Testing the search function.....	27
Testing the Sorting function.....	31

# Executive Summary.

The program features a sleek and user-friendly GUI design inspired by Bauhaus art principles, emphasizing simplicity and ease of use. Careful planning ensured a logical layout and consistent visual elements throughout.

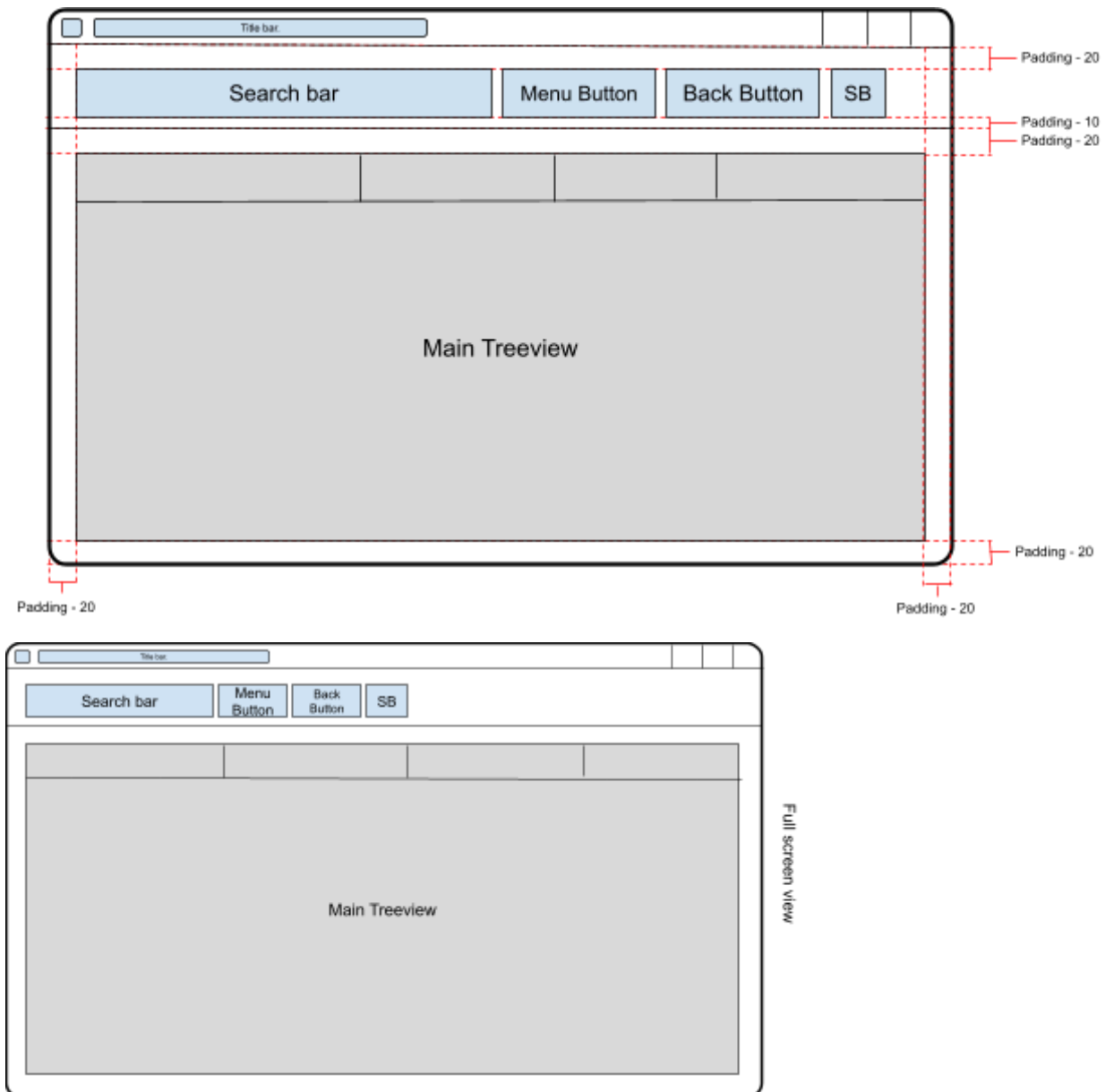
Icons and fonts were thoughtfully chosen to enhance usability and visual appeal. Custom classes were meticulously designed for modularity and scalability.

Thorough testing confirmed the program's reliability, with all 12 tests passing successfully. Overall, the program offers a polished and intuitive user experience, meeting modern design standards effectively.

# GUI Design.

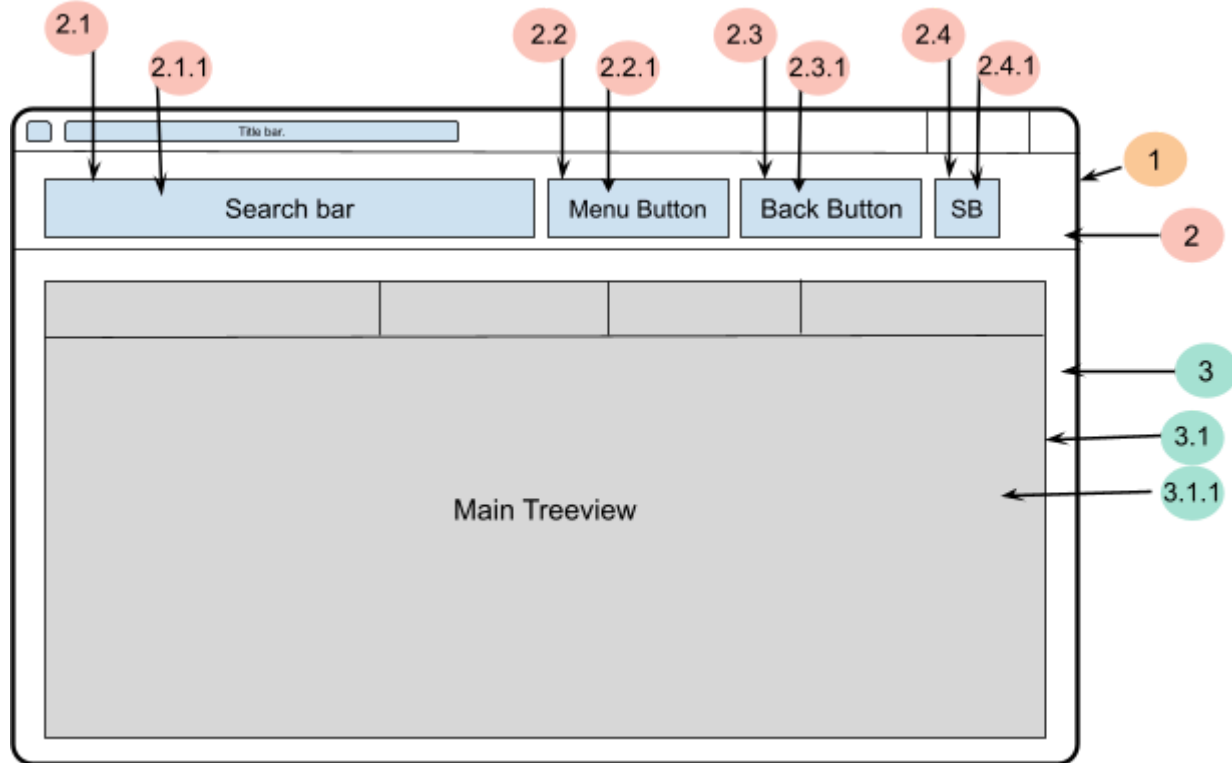
## Wire Frame.

The wire frame seen in image 1.0 will act as the base building block for the design of the GUI for this enhanced finance tracker. It allows the author to get a fundamental understanding of where each element will be housed and its size relative to all other elements.



## Hierarchy Of Elements and Widgets.

Utilizing the wire frame for the program the hierarchy and grid for each element/widget will now be created.



### 1. Main Window

### 2. Frame One (Pack, no padding, fill = both ways and expand)

- 2.1. Entry Frame with Border (Grid, col = 0, row = 0)
  - 2.1.1. Entry Widget (Pack, no padding fill = both ways and expand)
- 2.2. Menu Button Frame With Border (Grid, col = 1, row = 0)
  - 2.2.1. Menu Button (Pack, no padding fill = both ways and expand)
- 2.3. Back Button Frame with Border (Grid, col = 2, row = 0)
  - 2.3.1. Back Button (Pack, no padding fill = both ways and expand)
- 2.4. Search Button Frame with Border (Grid, col = 3, row = 0)
  - 2.4.1. Search Button (Pack, no padding fill = both ways and expand)

### 3. Frame Two (Pack, no padding, fill = both ways and expand)

- 3.1. Main Treeview Frame with Border (Pack, Padding = 20, Fill = both and expand)
  - 3.1.1. Main Treeview (Pack, no padding fill = both ways and expand)

# Visual Design Elements.

The GUI design draws inspiration from a series of pastel color palettes depicted in Image 1.1. The selection of these color palettes was driven by the visual appeal they held for the author/developer of this program and GUI.

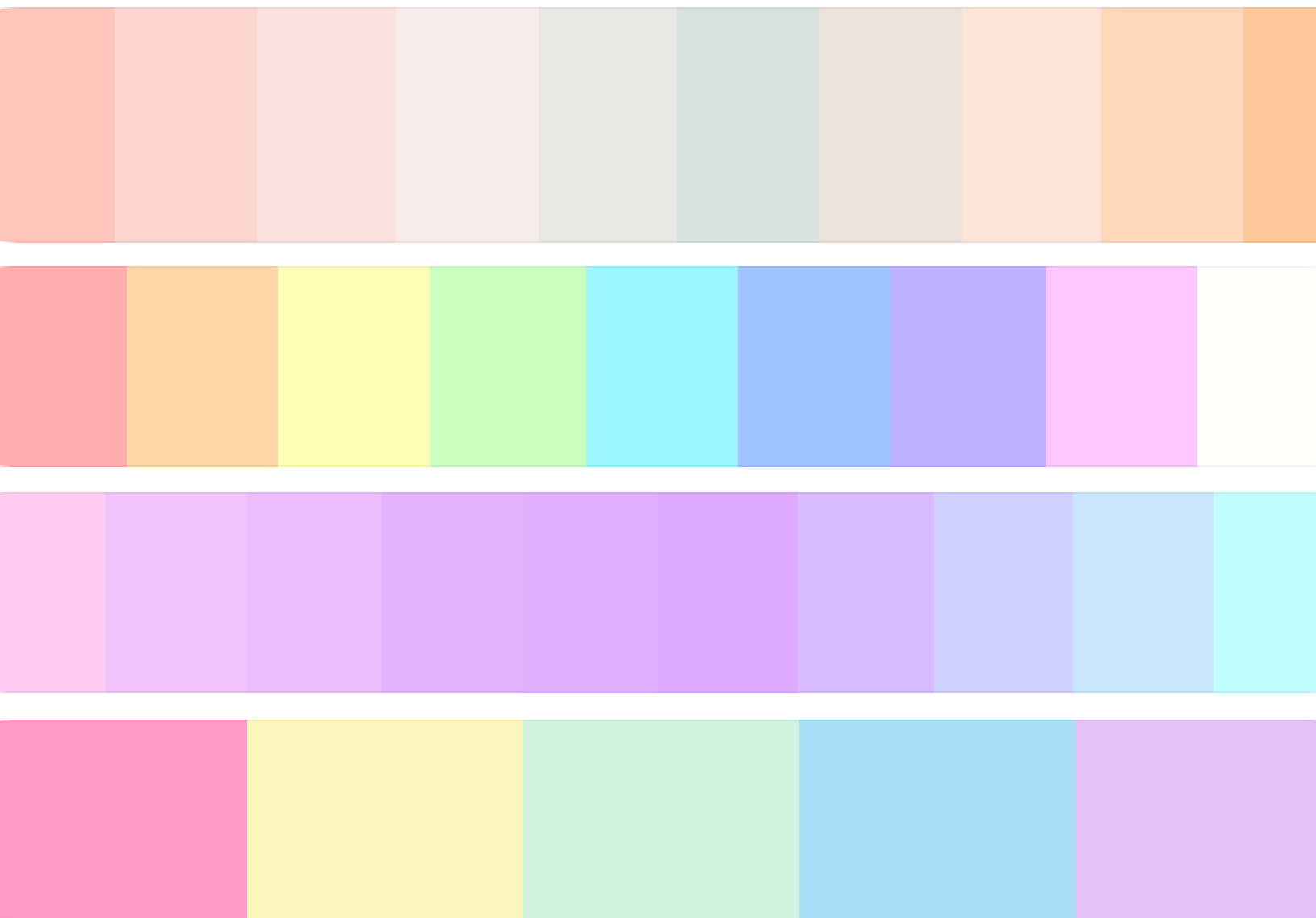


Image 1.1 Key Inspiration Elements.

Upon closer examination, it was observed that while all four color palettes were visually appealing, the latter three palettes, characterized by pastel blues, purples, and pinks, could potentially carry political connotations, particularly associated with the LGBTQ movement. To avoid any unintended implications or associations, the author made a deliberate decision to exclude these three palettes. This choice was not intended as a form of discrimination against any group, but rather as a precaution to ensure the project remains neutral and free from unintentional political undertones. Consequently, the chosen color palette for this project is the first one depicted in Image 1.2.

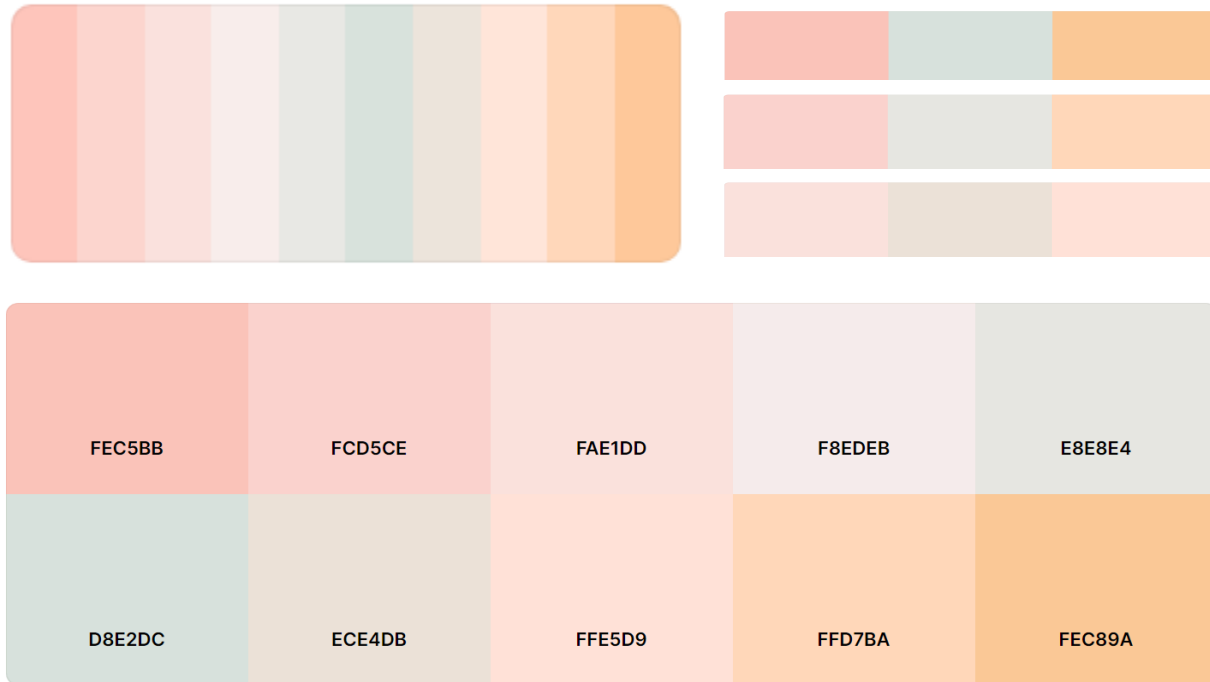


Image 1.2

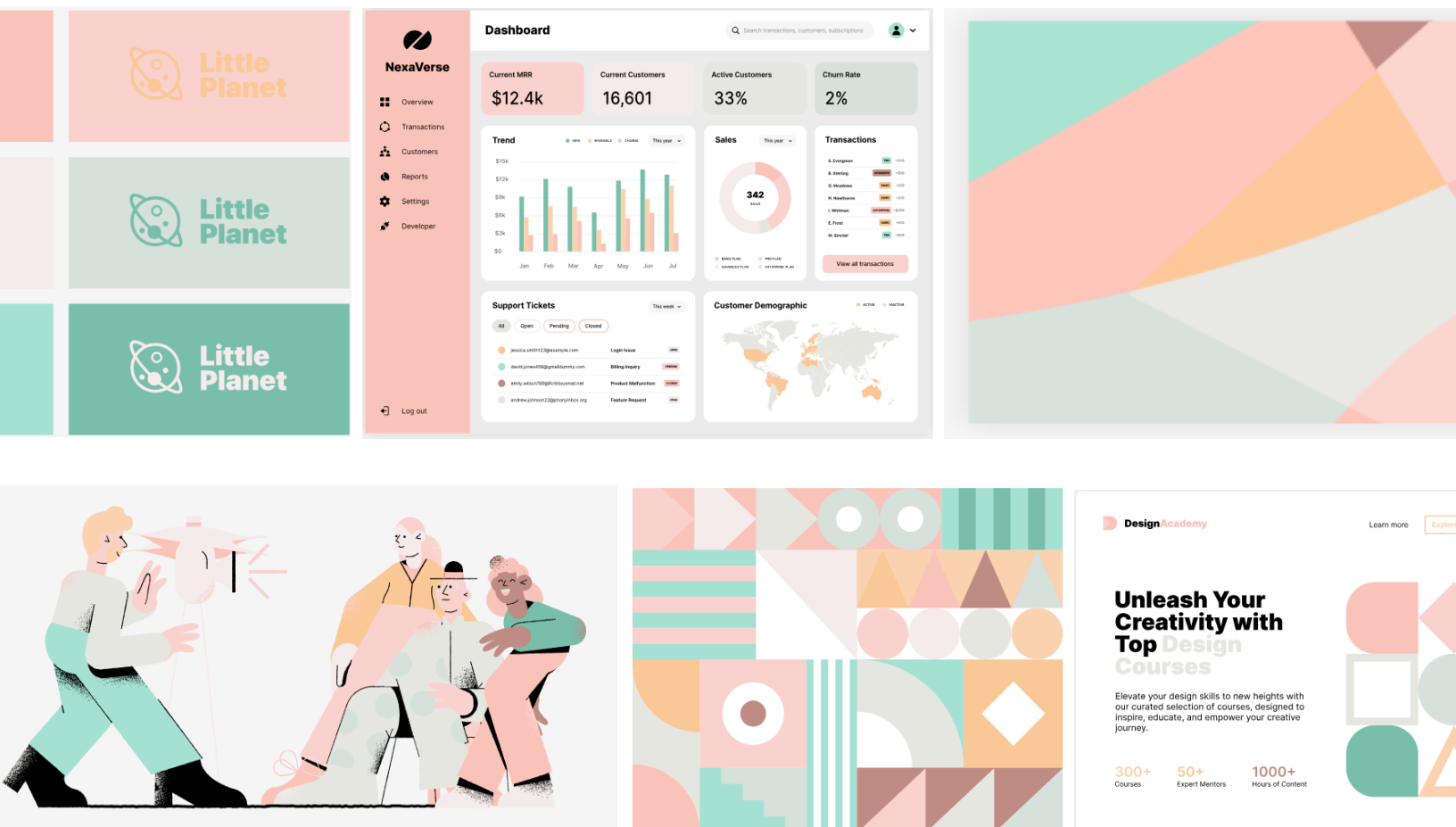


This selected color palette offers a cohesive aesthetic with the three main colors: Melon, Platinum, and Peach. To enhance visual interest and maintain clarity within the interface, the author will also incorporate Cambridge Blue, Tiffany Blue and Rosy Brown to create contrast between key elements such as; buttons, scrollbars, etc.

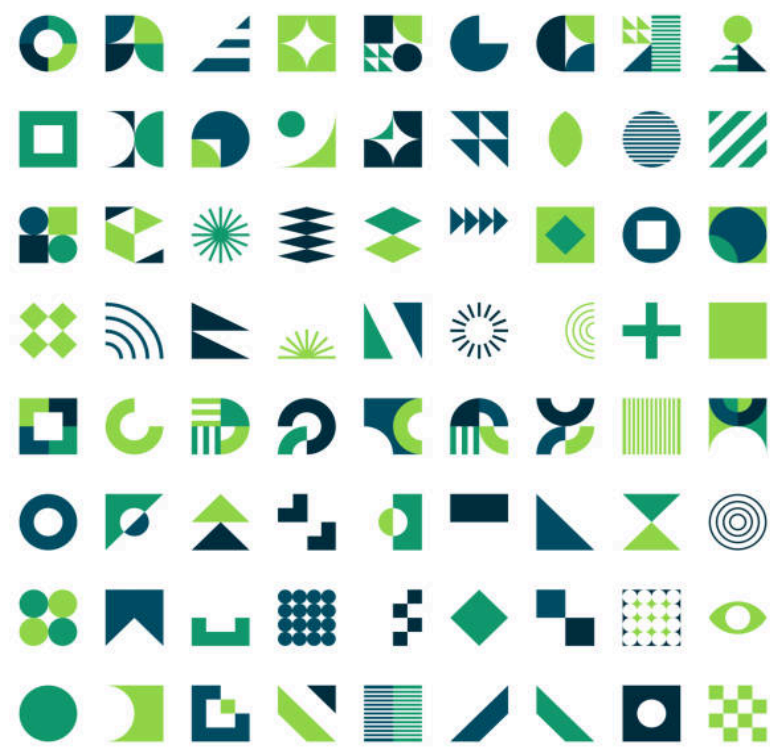
Taking all of that into account, the finalized color palette will contain the ten colors below.



## Color Palette Visualisation.







## Bauhaus Art.

One of the key visual elements that the author used as inspiration for designing this GUI was an early 20th century art style that grew in popularity in Germany named the Bauhaus Movement or aesthetic. This art style incorporates principles of simplicity and minimalism through the use of geometric shapes and patterns.



# The BIG Font.

In order to Incorporate the Bauhaus aesthetic, the GUI of this program features the iconic **Berlin Sans FB Demi** font which takes direct inspiration from the **Bauhaus 93** font. With its bold geometric shapes and clean lines, it epitomizes the avant-garde spirit of the Bauhaus movement. By integrating this font, the design attempts to capture the Bauhaus principles of modernism, simplicity, and clarity.

# Bauhaus 93!

**This is some really small text.**

**This is bigger but it's still small.**

**This seems decently sized.**

**This is pretty big now!**

**This is even bigger!**

**1 2 3 4 5 6 7 8 9 0**

**A B C D E F G H I**

**! @ # \$ % ^ & \* ( )**

**J K L M N O P Q**

**- + = / \ [ ] { } : ; . ' ?**

**R S T U V W X Y Z**

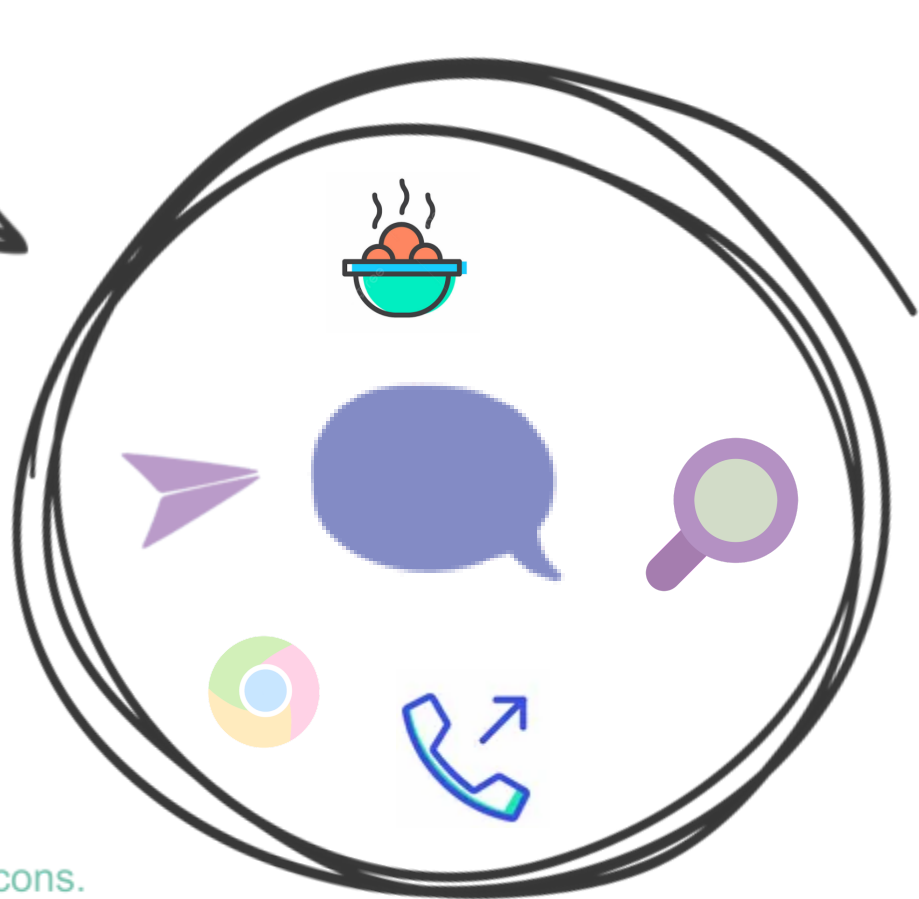


**a b c d e f g h i j k l m n o p q r s t u v w x y z**

# Icons

Being a relatively small program, it will only use three icon classes;

Main icons,  
Search icons and,  
Warning or Error icons.



When selecting icons, careful consideration is given to their size, typically ranging from 24x24 pixels to 128X128 pixels. At these small sizes, it's crucial that the icons effectively communicate their intended meaning. Therefore, the chosen icons should retain their key identity through clear and concise details.



28x28 Pixel  
icon



48x48 Pixel  
icon



128x128 Pixel icon

With this in mind, the author has selected the following icons for the three icon classes.



### Main Window Icon.

The main window icon will feature a simple depiction of money, serving as a clear indication to users that the program is finance-related. This icon has been specifically chosen for its effectiveness in conveying its message, even at a small size of 5x5 pixels.

### Main Search Icon.

The search icon will feature a straightforward magnifying glass symbol, selected for its association with search functionality and compatibility with the chosen color palette. Given its larger size of 48x48 pixels compared to other icons in the GUI, it can include more intricate details without sacrificing clarity.



### Warning Window Icon.



The warning window icon, designed as a 5x5 pixel image, conveys its message with utmost simplicity. It consists of a basic warning sign, identifiable by its bright orange color and transparent exclamation mark, effectively communicating its fundamental purpose.

Wireframe For Program.

# Designing Classes for the GUI

As required by the project brief, the development of this program and GUI will take an object oriented programming (OOP) approach, as such each type of element in this GUI will be made using a custom class created by the author.

Moreover, the classes for this GUI will inherit from object classes from the Tkinter (as tk) and ttkbootstrap (as ttkb) modules as super classes. They are;

- Window class from ttkbootstrap.
- Window class from tkinter.
- Frame class from ttkbootstrap.
- Treeview class from tkinter.
- Menu button from ttkbootstrap.
- Button from ttkbootstrap.
- Entry from ttkbootstrap.

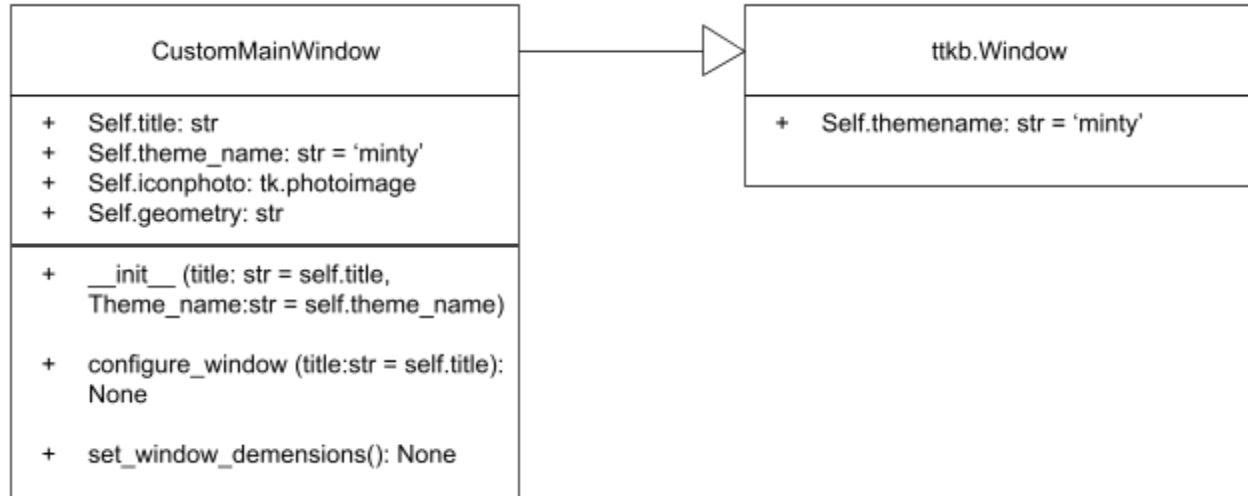
The design for each class will consist of a UML class diagram and Pseudo code for any class methods that require additional understanding of the logic behind them.

To begin with designing the classes for this GUI the author has identified the following classes as the classes that are needed to bring the GUI to life. These classes are identified by closely studying the grid and the hierarchy of elements/widgets of the GUI design (see previous headings)

1. **Class: Custom Main Window** - a class for creating custom main windows objects.
2. **Class: Custom Frame with Border** - a class for creating a frame object that acts as a border.
3. **Class: Custom Frame** - a class for creating a custom frame object.
4. **Class: Pop Up Window** - a class for creating a popup window.
  - i. Dependencies: Custom frame class
5. **Class: Custom Treeview** - a class to create a custom treeview object.
6. **Class: Custom Menu Button** - a class to create a custom menu button.
7. **Class: Custom Entry** - a class that creates a custom entry widget
8. **Class: Custom Back Button** - creates a custom back button
9. **Class: Custom Search Button** - a class that creates a custom search button.

## Class: Custom Main Window.

**UML implementation class diagram:**



**This class inherits from the Window superclass from ttkbootstrap**, the initialisation method of the superclass is called at the initialization of the CustomMainWindow class. This class takes two arguments/parameters: title: a string which will be the title of the window, and theme\_name: a string which will be the theme of the window which is set to 'minty' as default.

### The `__init__()` method.

Calls the superclass initialization method.

Calls the `set_window_dimensions()` method.

Calls the `configure_window()` method.

### The `configure_window` method

Configures the window object by setting the following attributes:

Self.title: str = title (passes as argument for `__init__()` method)

Self.geometry: str = '1000x563'

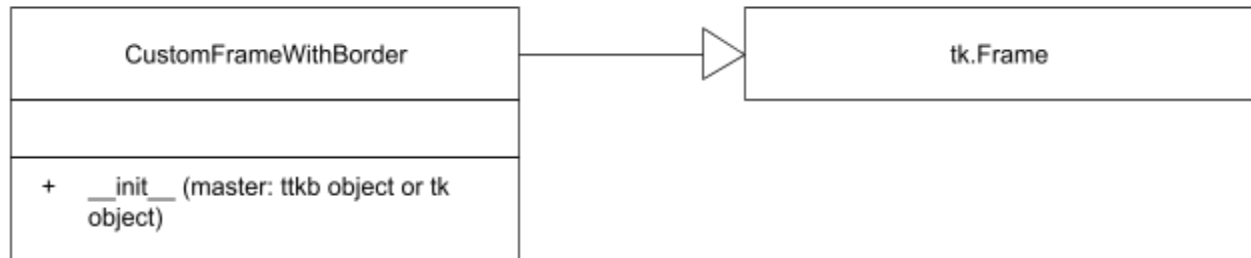
Self.iconphoto: tk.PhotoImage

### The `set_window_dimensions()` method.

This method sets the min dimensions for the window object.

## Class: Custom Frame With Border.

**UML implementation class diagram:**



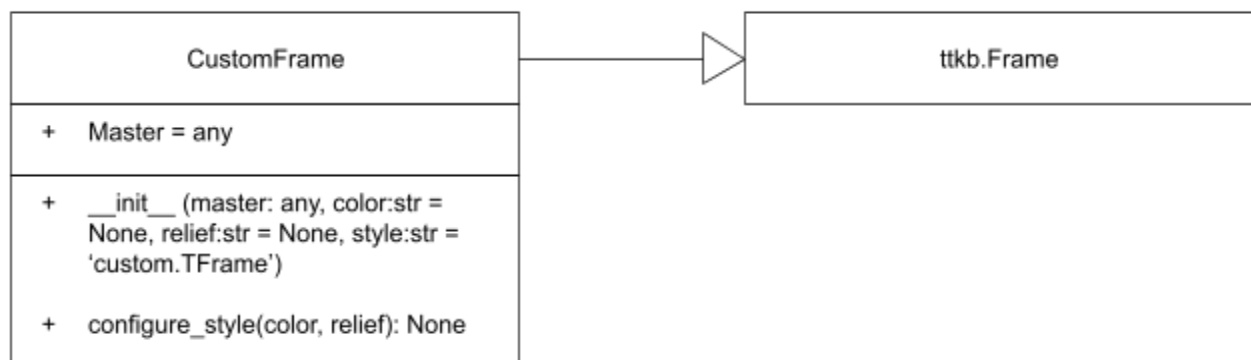
**\_\_init\_\_ method.**

Calls the initialization method of the superclass and passes the following parameters as arguments;

- master = master (passed as arg for init method of the subclass).
- highlightbackground:str = 'black'
- highlightthickness:int = 2
- background:str = any hex value as a str.

## Class: Custom Frame

**UML implementation class diagram:**



**\_\_init\_\_ method.**

Calls the **configure\_style()** method. (this styles the custom.TFrame style)

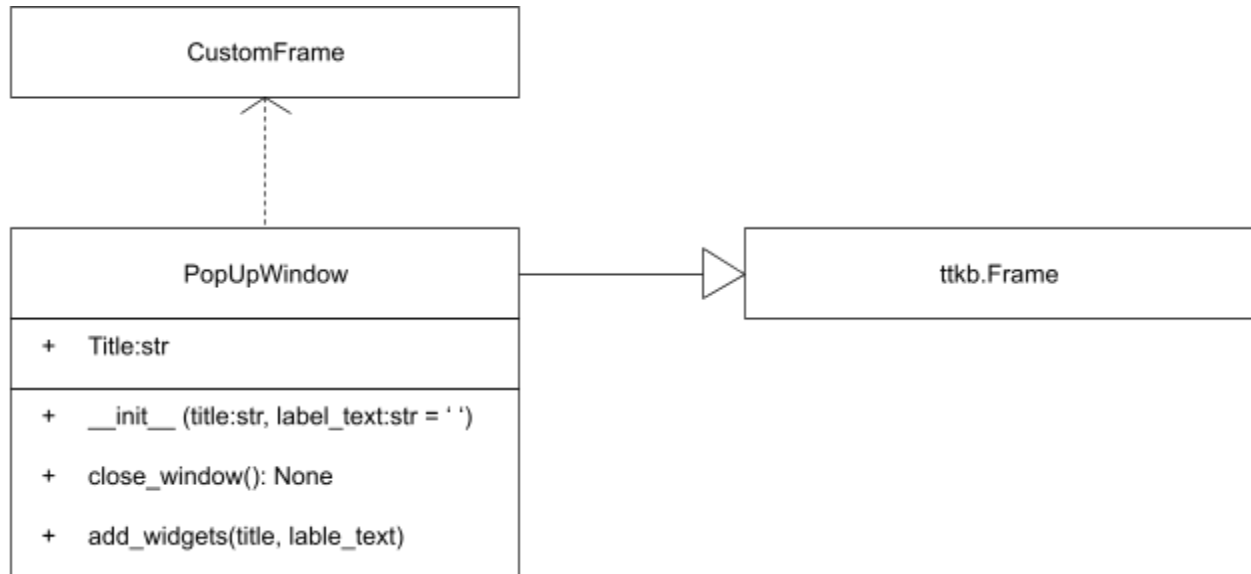
Calls the superclass initialization method and passes the following as args;

master = master

style=style

## Class: Pop Up Window.

**UML implementation class diagram:**



### Add\_widgets method

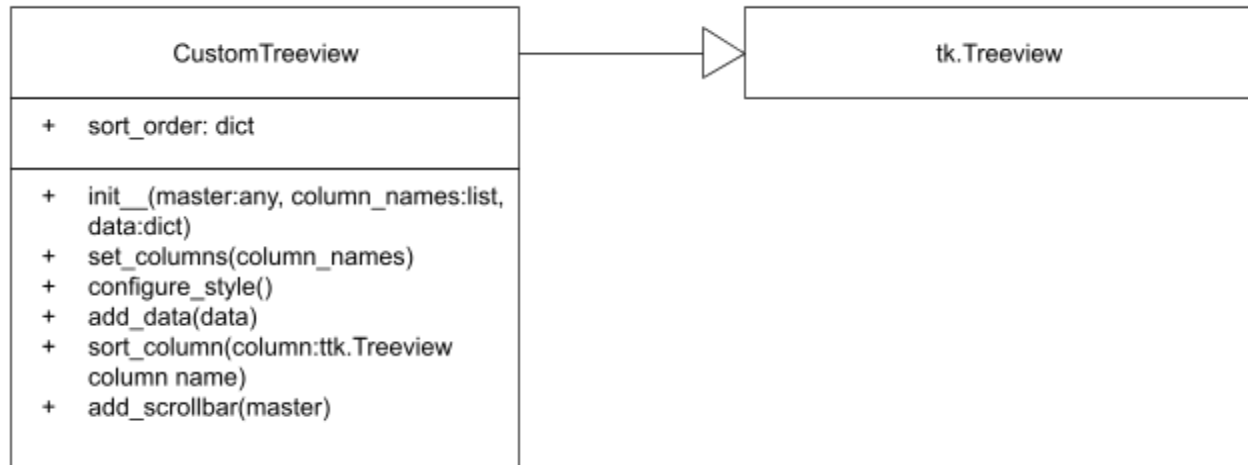
Adds the following widgets to the popup window;

- Frame one: CustomFrame object
- Label 01: ttk.Label object
- Label 02: ttk.Label object
- Button: a ttk.Button object with the text 'Okay' and the command close\_window()



## Class: Custom Treeview.

**UML implementation class diagram:**



Pseudo code and logic for the methods of this class are explained below.

### **\_\_init\_\_ method.**

#### **Pseudo code**

Call the init method of the superclass (args: master = master, show = headings).

Initialize an empty dictionary as an attribute to store the sort order data of each column as `sort_order`.

Pass the `column_names:list` passes to the init method as columns to self.

Call the `set_columns` method (args: `column_names`)

Create a for loop to set headings → for each column in `column_names` list:

Set the heading of that column with the text = column and command = `sort_column()` function with args as column.

Create a new key in `sort_order` as column and set its value as `True`

Call the `configure_style()` method

Call the `add_data()` method

Call the `add_scrollbar()` method

**set\_columns method.****Args:** column\_names: list**Pseudo Code:**

Create a for loop → for each column in the list column\_names:

Set the column for self with name= column, width as 100, centered.

**add\_data() method.****Args:** data: dict**Pseudo Code:**

In an try/except block try:

Initialize count as 0

Create a for loop → for each description, transaction data in the dict data:

Crate variable transaction\_type:str = transaction data [with key 'type']

Create var transactions:list = transaction data [with key 'transactions']

A for loop → for transaction in list transactions:

Add 1 to count.

If count is even:

Insert the translation to treeview with the tag 'evenrow'

Else:

Insert the transaction to treeview with the tag 'oddrow'

If an exception happens

Print it.

**Add\_scrollbar method****Args:** master**Pseudo Code:**

Crate scrollbar using tkinter

Pack it to the aster and align to right side.

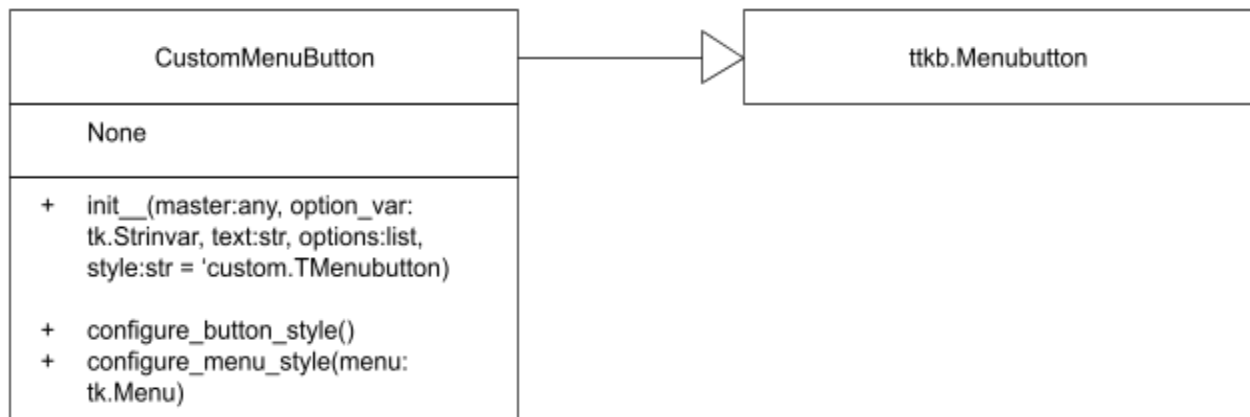
**sort\_column method****Args:** column: any**Pseudo code:**

Initialize a list containing tuples, each tuple should contain the position of the row(child) of the treeview(self) and the row itself (child)

Initialize a variable named reverse\_order:bool and set it to the attribute sort\_order[with key as 'column']

Then sort the newly created list of tuples with reverse order as not reverse\_order

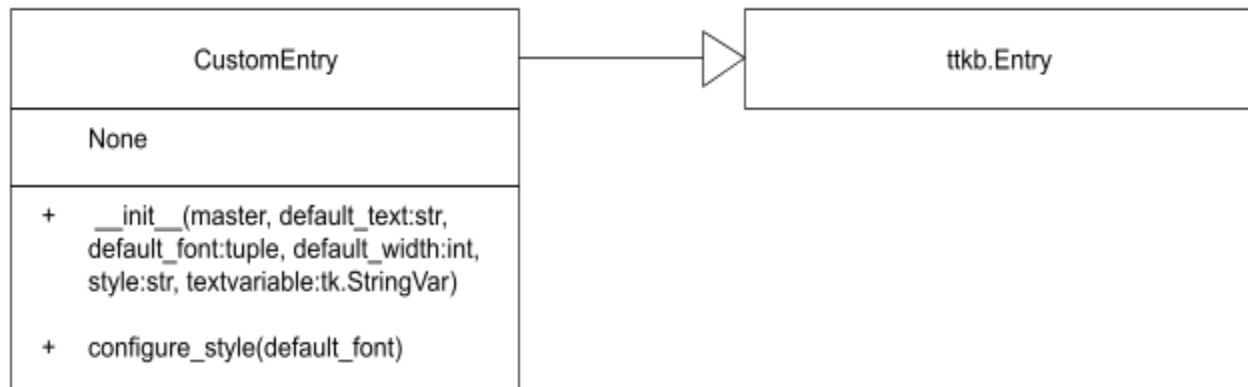
## Class: Custom Menu Button.

**UML implementation class diagram:**

The methods of this class are self explanatory and as such no pseudo code is required.

## Class: Custom Entry Widget.

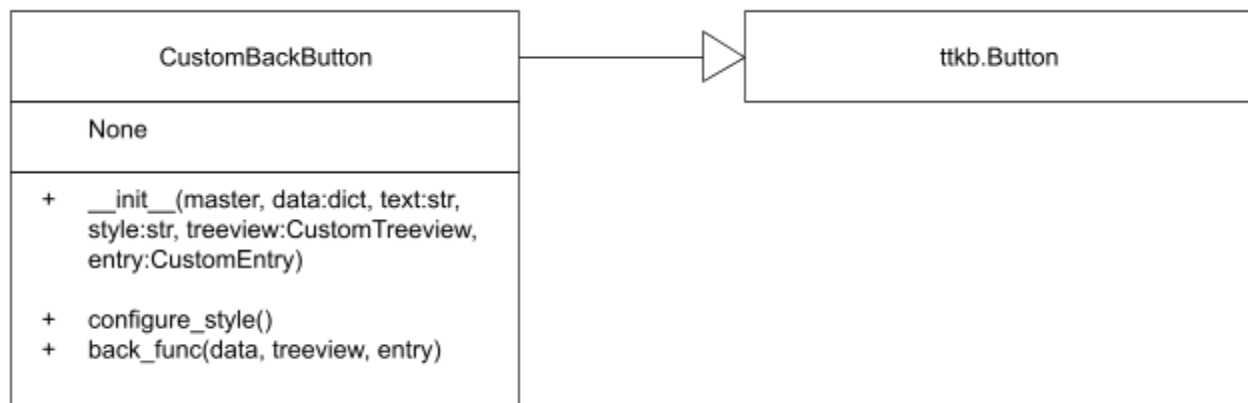
**UML implementation class diagram:**



The methods of this class are self explanatory and as such no pseudo code is required.

## Class: Custom Back Button.

**UML implementation class diagram:**



**back\_func pseudo code:**

Get the children of the treeview and delete it.

Call the `.add_data` method on the treeview and pass the dict data as an arg.

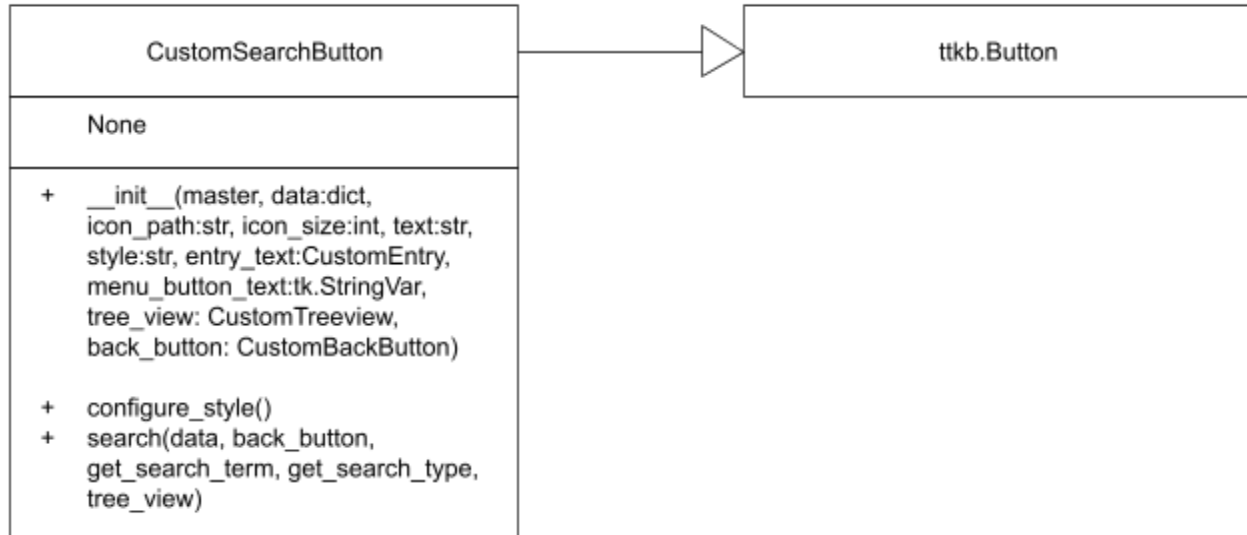
Delete the text on the entry

Insert the string 'search for a transaction...' into the entry

Set the configuration of self to disabled.

## Class: Custom Search Button

**UML implementation class diagram:**



### Search Function.

#### Pseudo Code:

if entry\_text is None or menu\_button\_text is None or tree\_view is None or if data dict is empty:

    print('Error: Missing required variables')

    Return from the function.

Create a tuple named 'search\_type\_tuple' containing valid search types as strings.

Get the text from the menu\_button\_text and assign to a variable named search\_type.

Get the text from the entry\_text and also assign to a variable 'search\_term'

if search\_term is None or search\_term is equal to 'search for a transaction...' or if search\_type is not in search\_type\_tuple:

    Create a Pop up window with the right titles and labels

    Return from the function.

else:

    Configure the state of the Back\_button to 'normal'

    Initialize a new empty dict to store searched data

if search\_type is 'description':

    Create a for loop for every 'entry' in data:

        if entry(make lower case) is search\_term:

            New\_data[key is 'entry'] = data[key is 'entry']

elif amount:

    In a try block:

        Create a for loop for every entry, entry\_data in data (get items):

            Convert search\_term into float,

            If search term is equal to the amount in data, add that transaction to new\_data

    Except if ValueError:

        Create a pop up window.

        Return from function

elif type:

    Create a for loop for every entry, entry\_data in data (get items):

        if entry\_data[ key 'type'] is equal to search\_term:

            Add the entry to new\_data.

elif date:

    Try to :

        Convert search term string to datetime.date object

    except if ValueError:

        Set back\_button configuration state to 'disabled'

        Create pop up window

        Return from function.

    Create a for loop for every entry, entry\_data in data (get items):

        Add entry to new\_data but make the key transactions' reference an empty list

        for transaction in entry\_data['transactions']:

            if transaction date is search\_date:

                Append transaction to the empty list made earlier

```
if new_data is empty {}:
    Create pop up widow with error message
    Return from function
else:
    Delete the everything in the treeview
    Call the .add_data method on the treeview.
```

# Testing the Program.

## Test Summary

**Total Tests: 12**

**Total Tests Passed: 12**

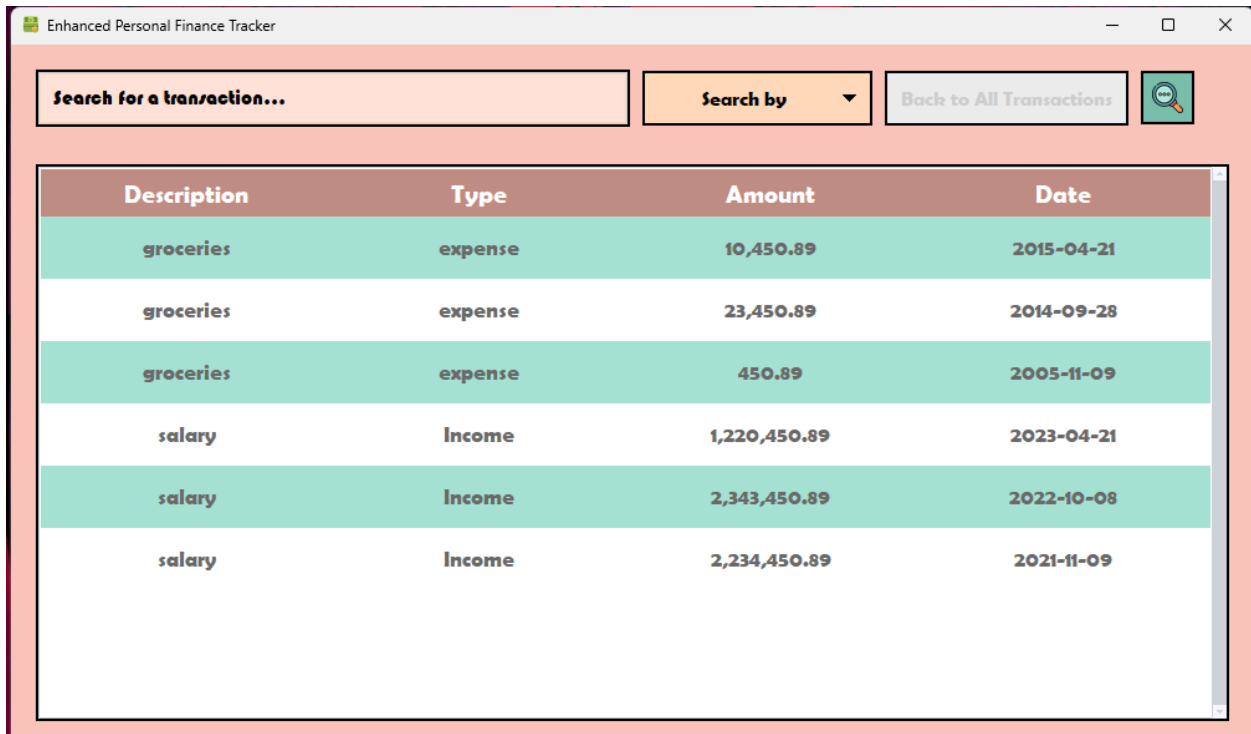
**Total Tests Failed: 0**

## Testing the GUI elements.

The test will involve running the main program code, and observing the various elements.

**Expected outcome:** Display all the required elements in the correct position in both windowed and full screen mode.

**Results:** Passed



Description	Type	Amount	Date
groceries	expense	10,450.89	2015-04-21
groceries	expense	23,450.89	2014-09-28
groceries	expense	450.89	2005-11-09
salary	Income	1,220,450.89	2023-04-21
salary	Income	2,343,450.89	2022-10-08
salary	Income	2,234,450.89	2021-11-09



Enhanced Personal Finance Tracker

Search for a transaction... search by Back to All Transactions

Description	Type	Amount	Date
groceries	expense	10,450.89	2015-04-21
groceries	expense	23,450.89	2014-09-28
groceries	expense	450.89	2005-11-09
salary	Income	1,220,450.89	2023-04-21
salary	Income	2,343,450.89	2022-10-08
salary	Income	2,234,450.89	2021-11-09

## Testing the loading of transactions from a JSON file.

This test will involve loading transactions from three different json files, each containing different formats of data that is seen below:

File 1:

```
{
  "Groceries":
  {
    "type": "expense", "transactions":
    [
      {"amount": 10450.89, "date": "2015-04-21"},
      {"amount": 23450.89, "date": "2014-09-28"},
      {"amount": 450.89, "date": "2005-11-09"}
    ]
  },
  "Salary":
  {
    "type": "Income", "transactions":
    [
      {"amount": 1220450.89, "date": "2023-04-21"},

```

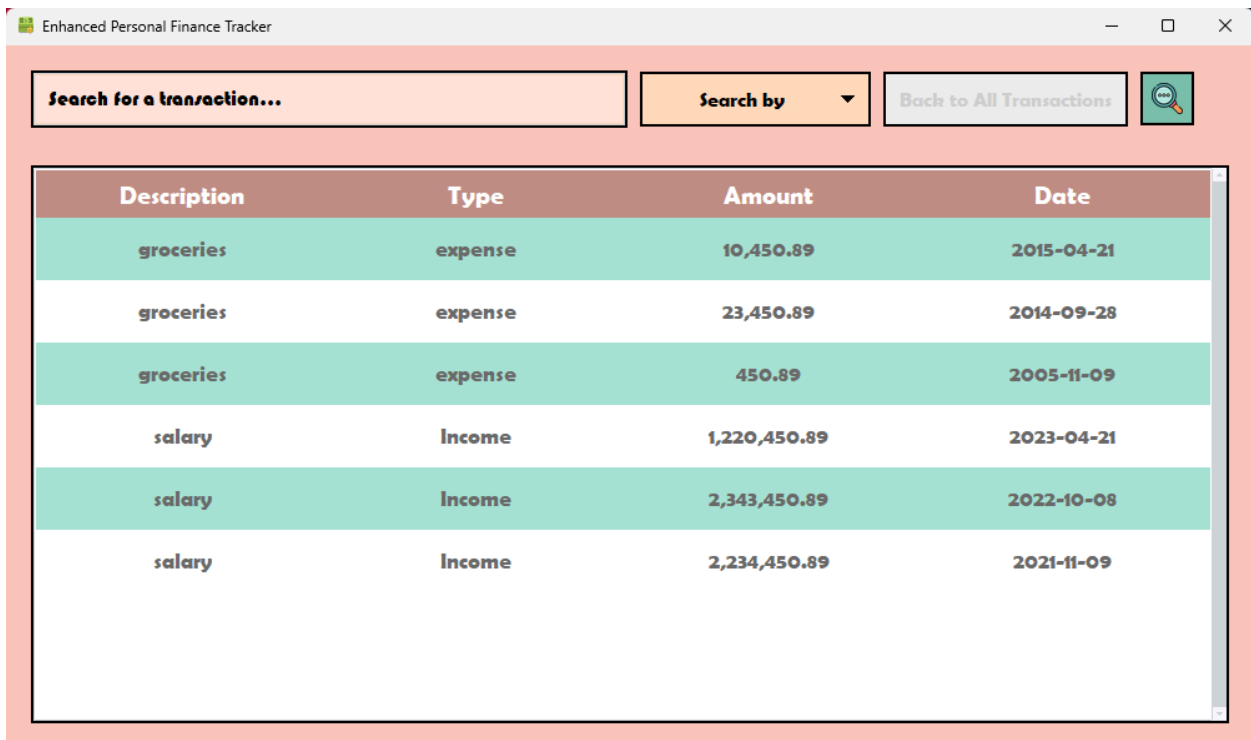
```
{
  "amount": 2343450.89, "date": "2022-10-08"},
  {"amount": 2234450.89, "date": "2021-11-09"}
]
```

File 2:

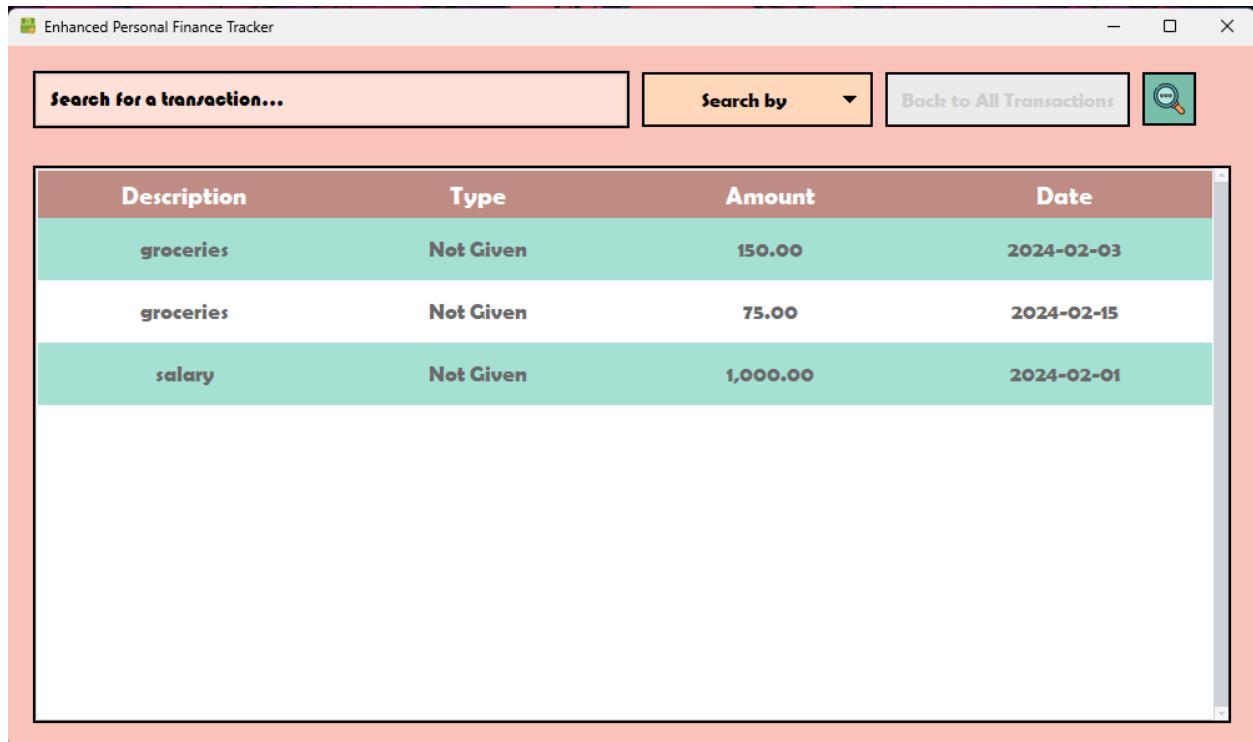
```
{
  "Groceries": [
    {"amount": 150, "date": "2024-02-03"},
    {"amount": 75, "date": "2024-02-15"}
  ],
  "Salary": [
    {"amount": 1000, "date": "2024-02-01"}
  ]
}
```

**Expected outcome:** Display file 1 and 2 correctly

**Results:** Passed



Description	Type	Amount	Date
groceries	expense	10,450.89	2015-04-21
groceries	expense	23,450.89	2014-09-28
groceries	expense	450.89	2005-11-09
salary	Income	1,220,450.89	2023-04-21
salary	Income	2,343,450.89	2022-10-08
salary	Income	2,234,450.89	2021-11-09



The screenshot shows the 'Enhanced Personal Finance Tracker' application. At the top, there is a search bar with the placeholder text 'Search for a transaction...', a 'Search by' dropdown menu, and a 'Back to All Transactions' button. Below the search bar is a table with four columns: 'Description', 'Type', 'Amount', and 'Date'. The table contains three rows of transactions:

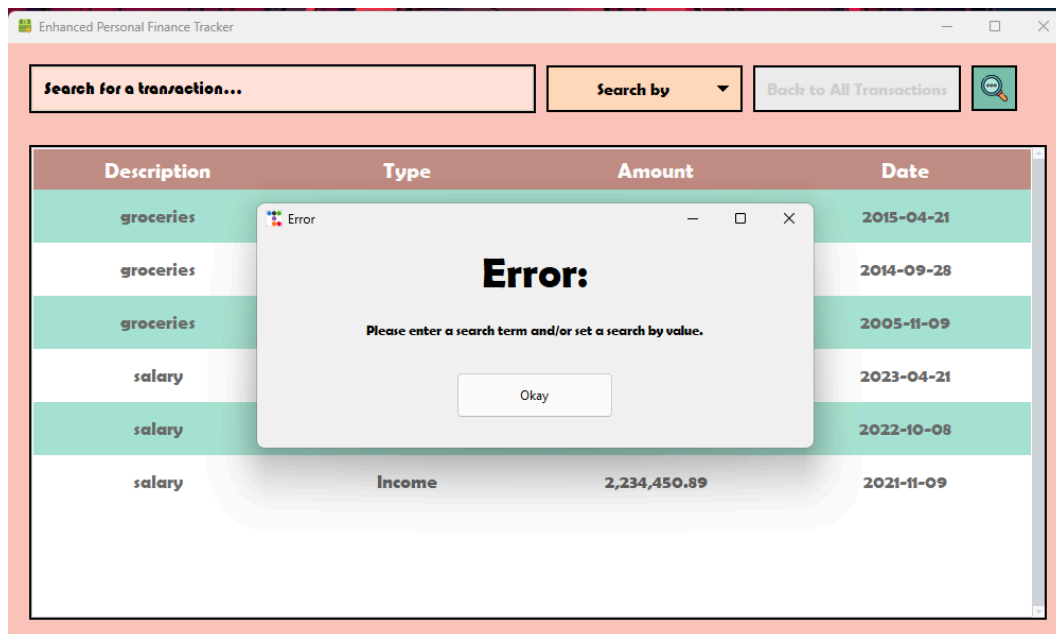
Description	Type	Amount	Date
groceries	Not Given	150.00	2024-02-03
groceries	Not Given	75.00	2024-02-15
salary	Not Given	1,000.00	2024-02-01

## Testing the search function.

**Test 01: search without entering anything**

**Expected outcome:** error pop up window

**Results:** Passed



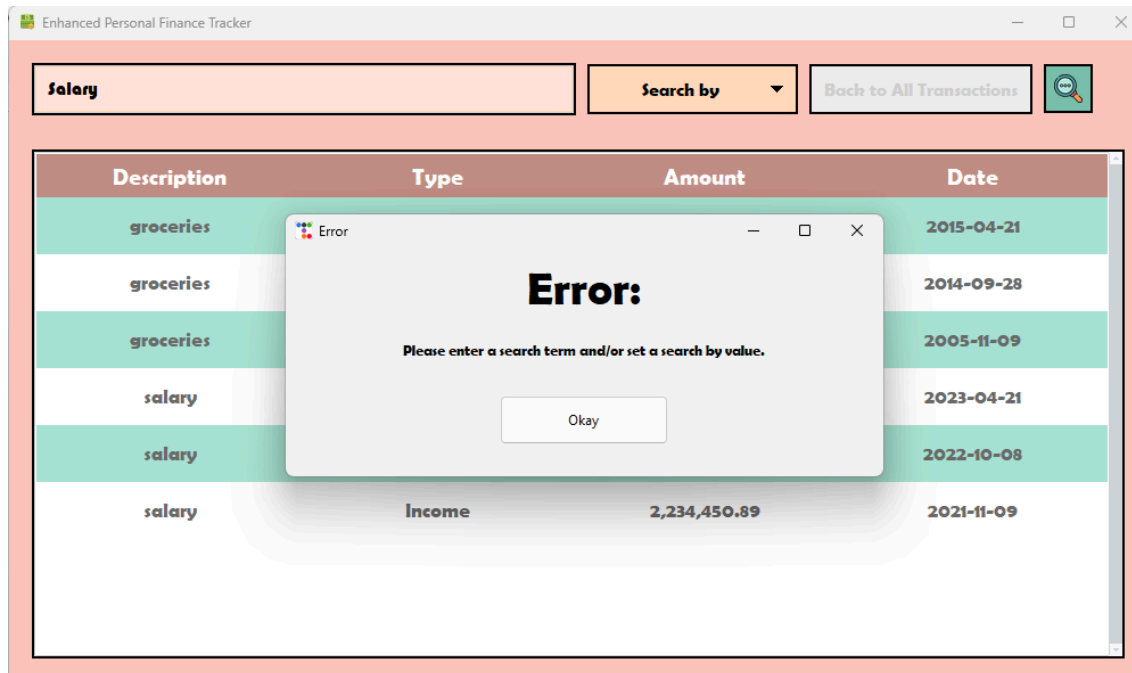
The screenshot shows the 'Enhanced Personal Finance Tracker' application with an error message displayed. The error message is a dialog box with the title 'Error' and the text 'Please enter a search term and/or set a search by value.' Below the text is an 'Okay' button. The background shows the same search interface and table as the previous screenshot, but the table is partially obscured by the error dialog.

Description	Type	Amount	Date
groceries			2015-04-21
groceries			2014-09-28
groceries			2005-11-09
salary			2023-04-21
salary			2022-10-08
salary	Income	2,234,450.89	2021-11-09

**Test 02: enter something but do not select a search by type and hit search button**

**expected result:** error pop up window

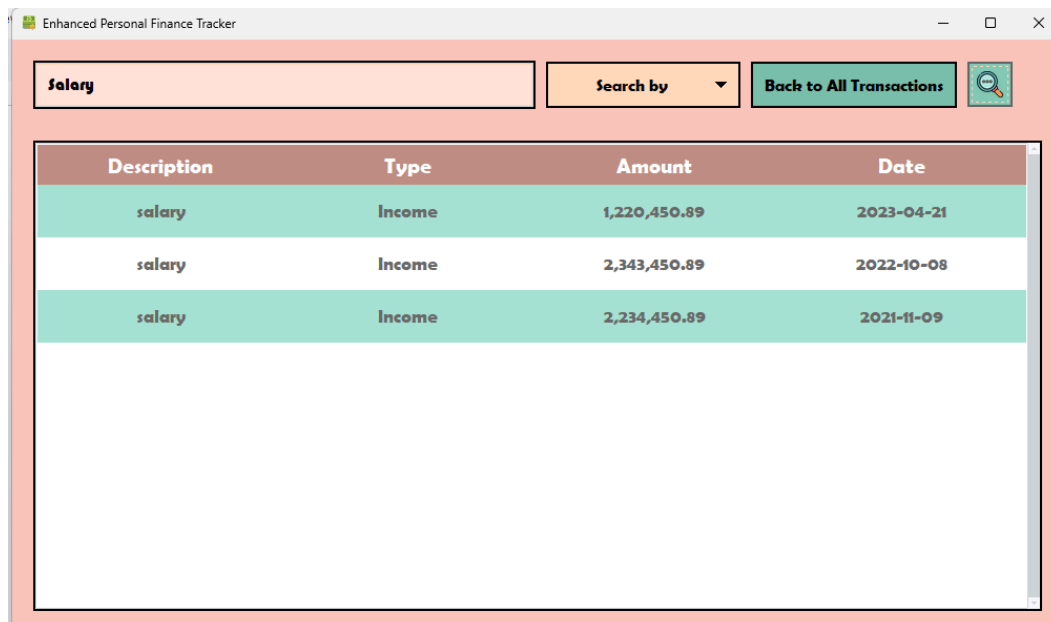
**Results:** Passed

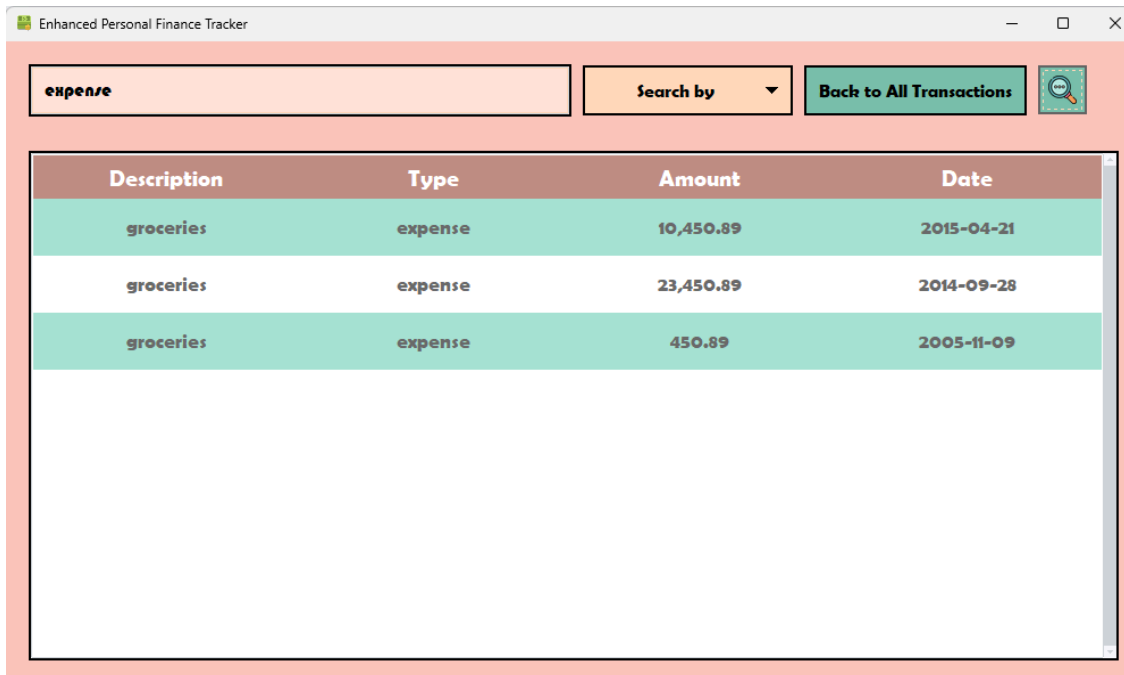


**Test 03: search something using description and enter 'Salary'**

**Expected Outcome:** correctly display the relevant results

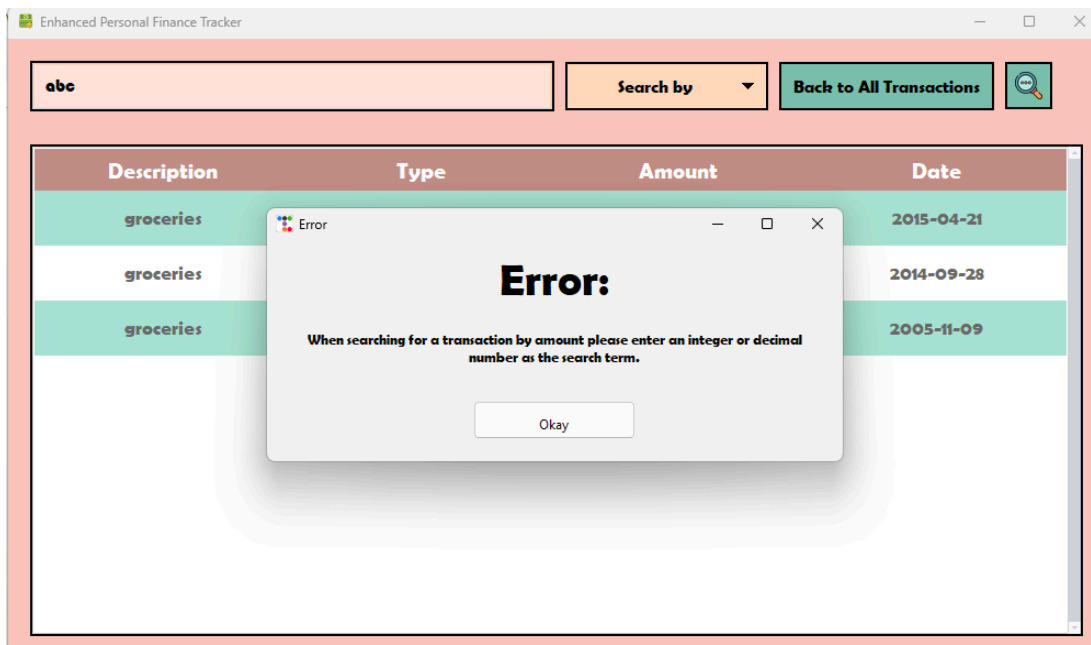
**Result:** Passed



**Test 04: search by type, enter 'expense'****Expected Outcome:** correctly display the relevant results**Result: Passed**

The screenshot shows the 'Enhanced Personal Finance Tracker' application window. At the top, there is a search bar containing the text 'expense', a 'Search by' dropdown menu, and a 'Back to All Transactions' button. Below the search bar is a table with the following data:

Description	Type	Amount	Date
groceries	expense	10,450.89	2015-04-21
groceries	expense	23,450.89	2014-09-28
groceries	expense	450.89	2005-11-09

**Test 05: Search by Amount enter 'abc'****Expected result:** error pop up window**Result passed:**

The screenshot shows the 'Enhanced Personal Finance Tracker' application window. The search bar contains the text 'abc'. An error message dialog box is displayed in the center of the screen, with the following text:

**Error:**

When searching for a transaction by amount please enter an integer or decimal number as the search term.

Okay

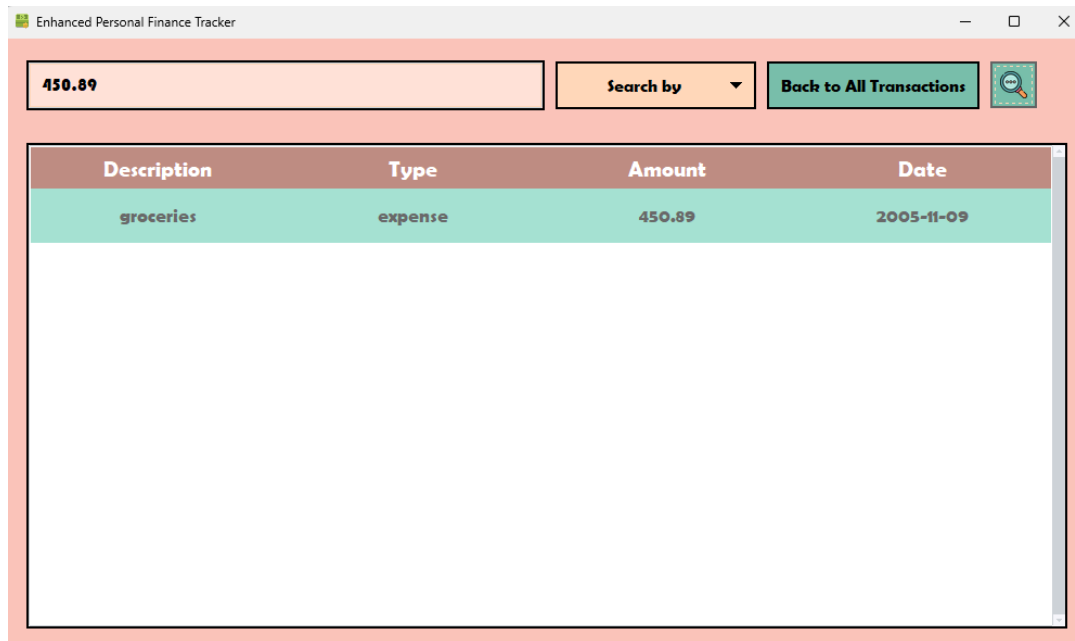
The background table is partially visible, showing the same data as in the previous screenshot:

Description	Type	Amount	Date
groceries			2015-04-21
groceries			2014-09-28
groceries			2005-11-09

**Test 06: search by amount, and enter a valid amount.**

**Expected result:** correctly display the relevant results

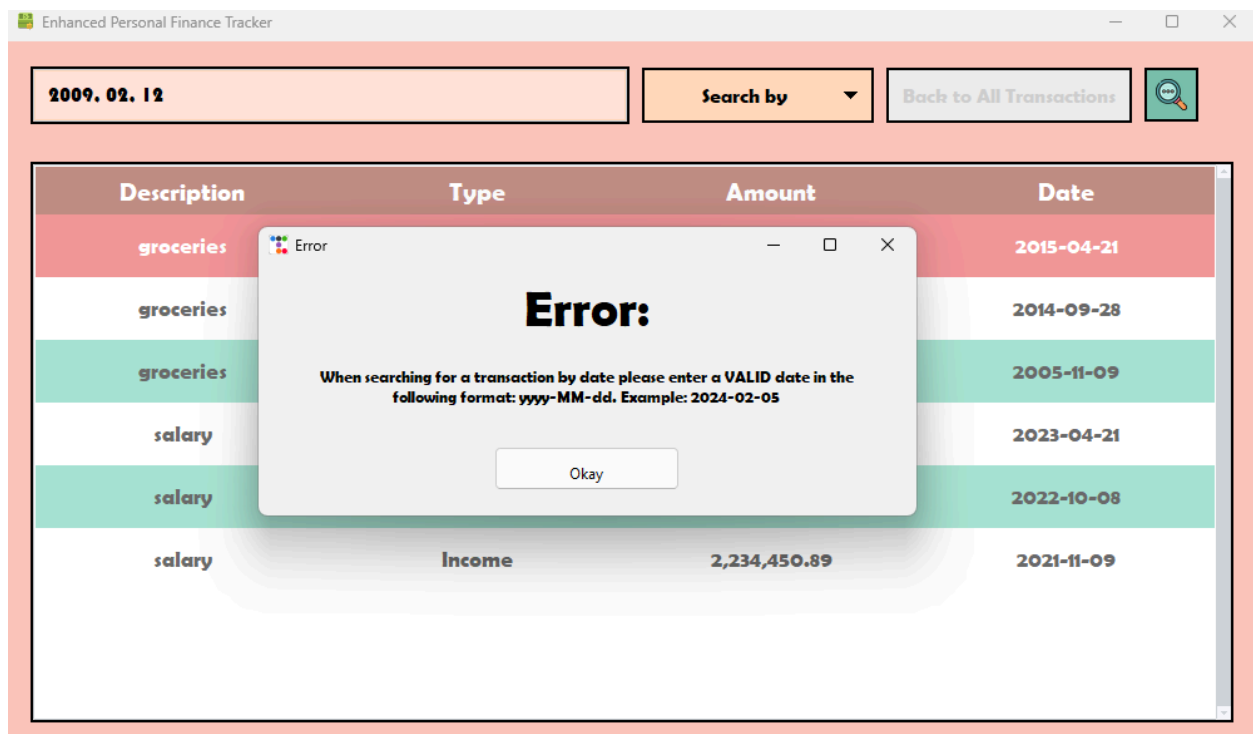
**Result passed:**



**Test 07: search by date and enter an invalid date:**

**Expected result:** error pop up window

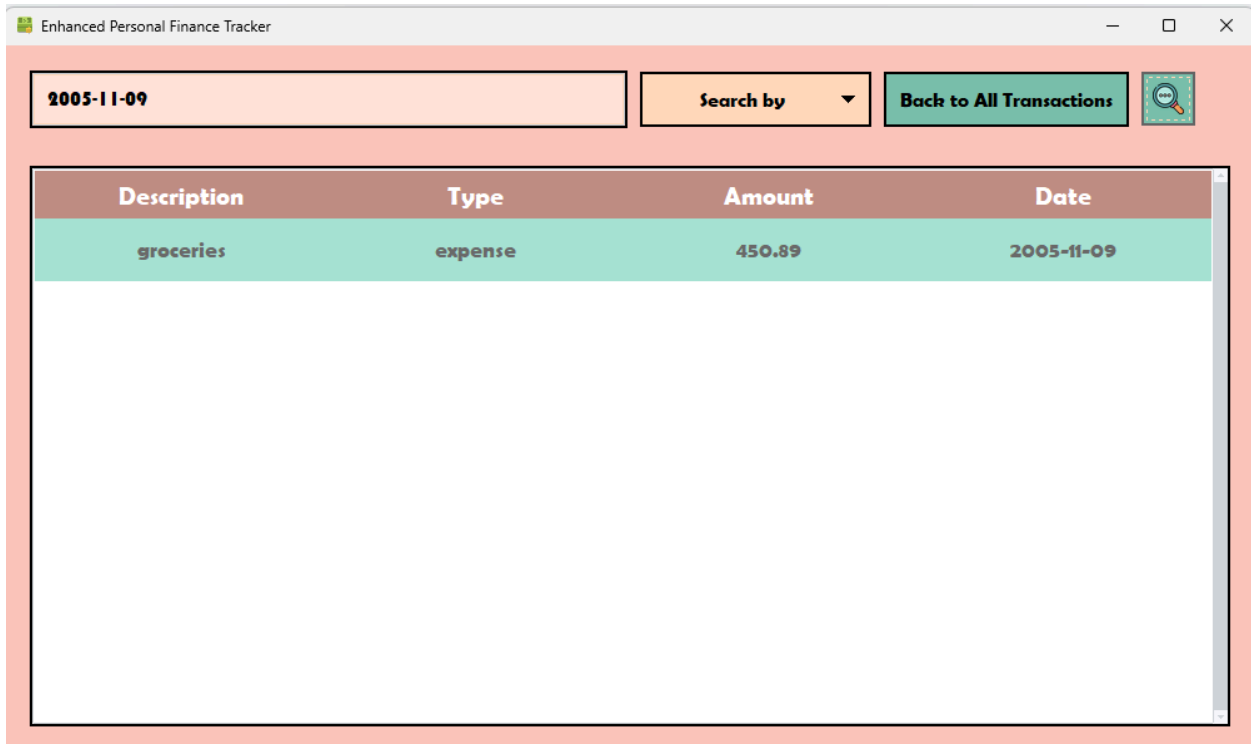
**Result passed:**



**Test 08: Search by date, enter a valid date.**

**Expected result:** correctly display the relevant results

**Result passed:**



The screenshot shows a web application titled "Enhanced Personal Finance Tracker". At the top, there is a search bar containing the date "2005-11-09", a "Search by" dropdown menu, and a "Back to All Transactions" button with a magnifying glass icon. Below the search bar is a table with the following data:

Description	Type	Amount	Date
groceries	expense	450.89	2005-11-09

## Testing the Sorting function.

The test will involve the event of clicking on a column of the treeview, for this test the Description column will be used.

**Expected result:** Correctly sort the data

**Result:** Passed

Next page.

Enhanced Personal Finance Tracker

Search for a transaction... Search by Back to All Transactions

Description	Type	Amount	Date
groceries	expense	10,450.89	2015-04-21
groceries	expense	23,450.89	2014-09-28
groceries	expense	450.89	2005-11-09
salary	Income	1,220,450.89	2023-04-21
salary	Income	2,343,450.89	2022-10-08
salary	Income	2,234,450.89	2021-11-09

Enhanced Personal Finance Tracker

Search for a transaction... Search by Back to All Transactions

Description	Type	Amount	Date
salary	Income	2,234,450.89	2021-11-09
salary	Income	2,343,450.89	2022-10-08
salary	Income	1,220,450.89	2023-04-21
groceries	expense	450.89	2005-11-09
groceries	expense	23,450.89	2014-09-28
groceries	expense	10,450.89	2015-04-21