Advanced Software Design Coursework

Design and implementation of a budget spending app

Romain Charles Kuhne: w1972584
Al Masum: w2012132
Mobeen Raja: w1803881

University of Westminster

Department of Computer Science and Engineering

Module leader: Dr Simon Courtenage
Module code: 7SENG003W

Assignment due date: 21/03/2024

Table of contents

Introduction:

This report focuses on the design, implementation and testing of the creation of a budget application used to track finances of a single user. The roles and responsibilities that have been assigned are as follows, Mobeen Raja as the project manager and tester, Romain Charles as the designer and then finally Al Masum as the developer.

This report will go into detail about the structuring of tasks and meetings required to implement this project. In the first section of this report, the UML design of this application will be discussed, with a detail of each key entity. Then, the code's implementation with each entity will be presented. Finally, in order to respect the software life-cycle, a testing section will be discussed to ensure quality control to deliver a fully functional finished product

Designing the UML diagram:
The UML diagram of the budget tracking app is composed of 4 classes: Wallet, Budget, Category and Transaction. Initially, a Note class referring to a transaction's description used to be present, however, in order to simplify the diagram, the group unanimously decided to remove it and instead include a transaction description represented by a string as a parameter of the transaction entity. The overall architecture of this application is based on multiple aggregations relationships in which the wallet class contains a list of global monthly budgets and categories, the latter storing transactions entered by the user in specific groups according to their dates of entry and their nature.

Wallet entity:
The main entity of the budget tracking application is the wallet class, responsible for financial operations such as instantiating a transaction, allocating them to a category or for creating a monthly budget and assigning it to a specific category. In addition to these functionalities, this entity also possesses features enabling the user to display each and every transaction that occurred at a particular period from each category and to track the total amount spent in each group. Moreover, this class also enables the user to track whether he has exceeded his monthly budget or not and would indicate the remaining or the exceeding amount. In order to achieve these functionalities and to ensure the storage of the required data, two separate lists of categories and total monthly budgets were added to this class.

Monthly budget entity:
The monthly budget entity's primary aim is to enable the user to track the total sum of money he would be willing to spend each month and to track whether his spendings remains within the monthly boundary set by this financial constraint. In addition to this constraint, the app would also allow the user to allocate a budget from the total monthly amount to a particular category, deducting the sum he is allowed to assign. Regarding its attributes, this class has a sum to allocate which is constant, a static variable labelled as category_budget to indicate whether the entire portion of the monthly amount was assigned to some categories and a date represented by a month and a year to separate this budget from others created. Finally, with the aim to prevent a user from exceeding the amount he's allowed to allocate, an interface called isBudgetAvailable was implemented.

Category entity:
Concerning the category entity, this class specifies the blueprint to enable the efficient storage and retrieval of monthly transactions entered by the user. To maintain the integrity of data and to ensure retrieval of the correct category from the list of categories in the wallet, a date represented by a month and a year, and a label were implemented as attributes. Moreover, to

indicate whether the category concerns a repository for expenses or incomes, an attribute named isExpense of type boolean was added to ensure separations of operations as groups labelled as expenses would hold a budget as opposed to incomes. Finally, a list of transactions was added to store all monthly journal entries recorded by the user.

Transactions entity:
The transaction entity delimits the required data stored within each journal entry respectively an amount, a date, a unique identifier, a description and a variable called monthly repeat to indicate whether a transaction is repeating every month.

Limitations of the design:
One of the main limitations with the architecture of the Budget tracking app is directly related to the chain of containment structure mentioned above between the wallet class, the budget and the transaction. The problem lies within the highly dependent links between core entities rendering the system tightly coupled. This disables the system to be flexible regarding any changes of implementation made to an entity.

Implementation:

In the development of the application, the initial step involved creating a foundational structure by establishing all necessary classes and interfaces, while also defining their interrelationships. A helper class was introduced to encapsulate common functionalities, such as the automatic generation of transaction numbers. Originally, several functions within the Wallet class were designed to accept multiple parameters directly. This approach was later refined to enhance the system's architectural integrity: objects are now created based on user inputs within the Program class and subsequently passed to the Wallet class methods. This adaptation ensures that user interactions do not compromise the clean, initial UML design.

Abstraction was employed to maintain the encapsulation of class properties, keeping them private and accessible only through getter and setter methods. This promotes the principle of data hiding and encapsulation. Furthermore, the application leverages inheritance and polymorphism through the implementation of interfaces such as IBudgetAvailable and ITransactionDate, and by extending the Wallet class within the Program class. Such use of object-oriented principles not only adds to the robustness and scalability of the application but also ensures its components are reusable and adaptable to change.

The repeated utilization of certain functionalities across different parts of the application exemplifies the dynamic nature of the system, showcasing polymorphism. This approach not only enhances code reusability but also contributes to the overall flexibility and maintainability of the application.

The application initiates with preset categories and demo data to illustrate its capabilities from the start, offering an immediate understanding of its functionalities. Error handling and user feedback are integral to the class's operations, ensuring that users are informed of the outcomes of their actions, whether successful or requiring attention. This level of interaction enhances the user experience, making financial management more accessible and less daunting. We have used red colourful text for error and warning messages.

Wallet class exemplifies a well-structured approach to managing complex financial data, which is the core class for this project. Its methods are designed with clear responsibilities, adhering to single-responsibility principles and encapsulation. This results in a maintainable

and extensible codebase that can accommodate future enhancements or modifications with minimal impact on existing functionalities.

The classes also demonstrates strategic use of LINQ for data manipulation, offering concise and readable code for filtering, aggregating, and modifying financial data. This not only optimizes performance but also simplifies the implementation of complex queries.

Please refer to the appendix for the full listing of the code.

Testing introduction

This section of the report will address the testing of the program developed, the formatting of the test, the testing itself, the iterations of the tests, retesting and then a summary.

The testing involved creating a test plan that had different sections needing data, the first being "Test Number", this was created to easily identify a test for referencing purposes and displaying the screenshots of the tests taken. Secondly, "Test Case Type" was the type of test occurring, the program is console based and has a simple interface and so the main types of tests were functionality but there were two test types, the other being usability based and so the thought process was that the distinction warranted a clarity tag, also not knowing how many of these tests of each type would be occurring also meant that this had to be created as precaution.

The Next tag was "Test Description", which simply described the test's purpose and reasoning behind it, followed by "Test Steps" which was explaining the steps taken in testing and giving a step by step description of the test steps, then there is the "Test Data" which contained all the data that the test was using, this was important because this could be easily referenced for future tests and analysis purposes, this section really helped the program developer see what was the true input of data and it helped with problem solving failed tests.

One of the final tags for the test plan was "Expected Results", this would indicate what the test result should look like and what it would need to do; to be considered a pass. Following this was "Actual Result", this is the result of the test and it explains how the test went, so if it went as expected it will describe so and if not and the test failed, it'll let the reader know how the test failed and why it failed, this is important because efforts were made to make sure to describe at what step the test failed, as a test can have more than a single step, there had to be a thoroughly explanation of what went right and what went wrong if it did at all. Lastly is the last tag which was "Status", describing the result of the test as simple as passing or failing.

Lastly, the structure of the test plan included, the test plan table with the above tags and information for each test filled, the screenshots of the tests, this was so for the purpose of having visual clarity at which point if failure does occur, visual clarification would be a guidance that could pinpoint where and perhaps why. Then the final part when needed was a short section for notes for anything that was missed and applicable to put inside the test plan table due to limitation of tags only being suited for test information.

Testing

Test version 1:

Testing will now be discussed, the order in which it was done and the iterations of the tests, as there were more than one. For testing the thought was starting with the core basics and so the very first thing that was tested was the creation of a transaction as this is a budget application, this involved creating labels, setting amounts for the transactions and then the

date in which they occurred, this first test went as planned, then to test if the viewing of the transaction would work, so this would also verify the first test which was the creation of the transaction and so that's what transpired and the program let you view the transaction.

Next test was if the creation of a category was working and it was working in some sense and some tests showed failure in category options, such as checking to see if you could rename a category, which you couldn't, some other tests involved testing to see if the program had preset categories and it didn't, these issues were problematic as an budget app should not be static and should allow freedom of changing labels and amounts which leads to the next test failure in which trying to change the amount a transaction was also did not prevail correctly.

Some of the next tests that were performed were to do with testing to see if a transaction could be removed, that test passed. Testing if the changing of a transaction's category was working as intended and it was, then lastly testing basic usability such as the ability to exit the program and if you could browse through the menus with the ability to return back to the menu, these tests succeeded. The summary of the first set of tests was to hold out on testing other features because the failing tests were mandatory to fix and testing the other features would've only been more time consuming and possibly up for change since the program had to be fixed.

The tests that failed were given reasons on why they failed and tests that passed were given reasons why they passed but there's a gap in this in terms of quality of life changes and or small errors unrelated to the tests that occur and so adding notes to communicate thoughts regarding what could be improved and errors that were seen.

The notes for this test plan that included fixing up when needing exact dates such as day, month and year are not always needed, for example using setting a monthly budget only needs the month and year, another note added entails the entire budget menu not functioning correctly or as intended so that was noted and showed for the code developer to see and amend.

Another note was to do with the transaction ID codes that were assigned to transactions and how they were always set to "TR00000000" and so when trying to interact with a created transaction, having more than one created meant that they would all share the same code and trying to correctly pick them each time was impossible, so this was noted and then also the ability to create a transaction when having no budget assigned to a category was possible and it shouldn't have been as transactions are always tied to a category; meaning their expense value is equal or less to the category's budget and never higher.

Test version 1 retesting:

The retesting involved testing all the tests that failed from the first test plan version 1, so it meant the tests were limited to old testing and no new unique tests were performed, which meant testing of the new features had to wait until this was done, so one of the first tests was to do with inserting an transaction that exceeded the budget that was given to it was being allowed and it shouldn't and so this was retested and passed with flying colours, followed up by testing to see if the application had preset categories and it finally did, it contained good generic categories found in budget apps which exceeded the expected results this time round.

Another test that failed and needed resting this time was the ability to change a category's label, of course this time round the test was a success and then finally there was some confusion on testing to the modification of a transaction, the program needed there to be a monthly budget set, a budget for the category and then finally the ability to create a transaction, this test had to be performed three times before reaching the pass status it needed..

The notes for this section of retesting involved confusion with the test that had to be done three times and what concluded from it and the changes made to the test's iteration each time it was performed.

Test version 2:

The second round of testing was shorted than the first round mainly because a lot of the core issues were tackled but some still remained, so the first thing tested this time round was the ability to display all ongoing transactions and so this started with assigning a monthly budget, a category budget which the transaction was going to be assigned to and then finally creating the transaction and using the "Display all ongoing transactions" feature to see the transaction occurring, then using similar creation test step, testing the viewing of a category's only transaction list and this feature worked as intended too.

Thirdly was testing the ability to change a category's budget, this was important and tied to a previous test to do with the existence of preset categories, so it was important and that the ability to create a category and assign it a value worked as intended but the preset categories by default were set to "0" and so if you wanted to use a preset category you obvious had to update it and so this test was designed around changing a preset's category and then viewing the change having been made, the test was a success.

A test that failed this time round was a test to check if the program was letting you create a category that shared a label with a pre-existing category, which shouldn't have been allowed and it was and so this needed to be fixed and was given the fail tag.

The last two tests involved checking to see a month's budget and so it involved creating a budget for the month and then checking it's existence through the viewing budget for the month feature, the test following this was the removal of a month's budget and this worked as intended.

Test plan version 2 only contained a single failure and all other tests were successful and so tests from the previous version had helped in the development of the code and gave keen insight on how to handle issues.

Test version 3:

This was the last test version and so to start it off testing the feature of recurrence and to see if it was working as expected, so the ability to set a transaction to recurring should exist and it should describe a transaction as such in the viewing of transactions, and it did exactly this so this was a big success. Following the first test was the testing of income as an transaction option and seeing again its viewing and correct description, also a success.

The next test was the tracking of the budget, and by this, it means seeing your expense being compared to a budget and how much remaining balanced you have, and if you have any income it would be displayed in this feature, so the testing process involved creating a category budget, a monthly budget, an expense transaction and an income transaction and the result was amazing as it did all these things as needed.

The last test was a usability test entailing visual clarity confirmation and this was tested by inputting string text into a field that should only take numeric values, the test concluded with the program not only giving an error message but also displaying that message in a red colour which was very nice and helped with usability of the program.

Testing Summary

Overall the tests were concluded true to the test plan layout, it was created thinking adding important fields would help, what was learnt was that it really did help and it helped the development of the code know what directions to take to fix it the issue as the tests tracked the test steps, gave the exact data used, summarised the test failure all whilst giving image evidence in the form of screenshots.

Another lesson that was learnt was the quality tests that contained many quality tags meant that it would consume a lot of time to take the tests with precision but the effort was worth it, as the tests continued the failure rate of tests went down significantly. The first test plan helped starting off the tests and showed real issues mentioned previously that the program had, as more tests were conducted it was realised by the last testing phase that two ignored major aspects which were the recurrence feature and the income feature and through testing this lack of feature was exposed.

So to conclude the tests carried out had major accuracy which contributed to the development of the code and helped with design and code changes amongst each testing iteration, and we now understand the importance of testing a code, as the issues that were present were beyond what we thought would even appear.

Mobeen's contribution:

This section is written by Mobeen, I was the project manager and the tester for the budget project, this meant I had to organise meetings and discuss appropriate timings with all members to meet with, and also the tracking of meetings in a detailed format, I was also in charge of setting the agenda for the followings meetings held.

For the testing aspect I solo tested the application and gave feedback fast to the team to help make changes to the design and development where applicable (mentioned in testing), the tests helped realised flaws in the project and it helped pave the way for development and fixing of errors, on top of basically pointing out application breaking errors, I helped the team discuss and realised what we lacked, such as an income, we did so much work and forgot to add income as a form of transaction.

Masum's contribution:

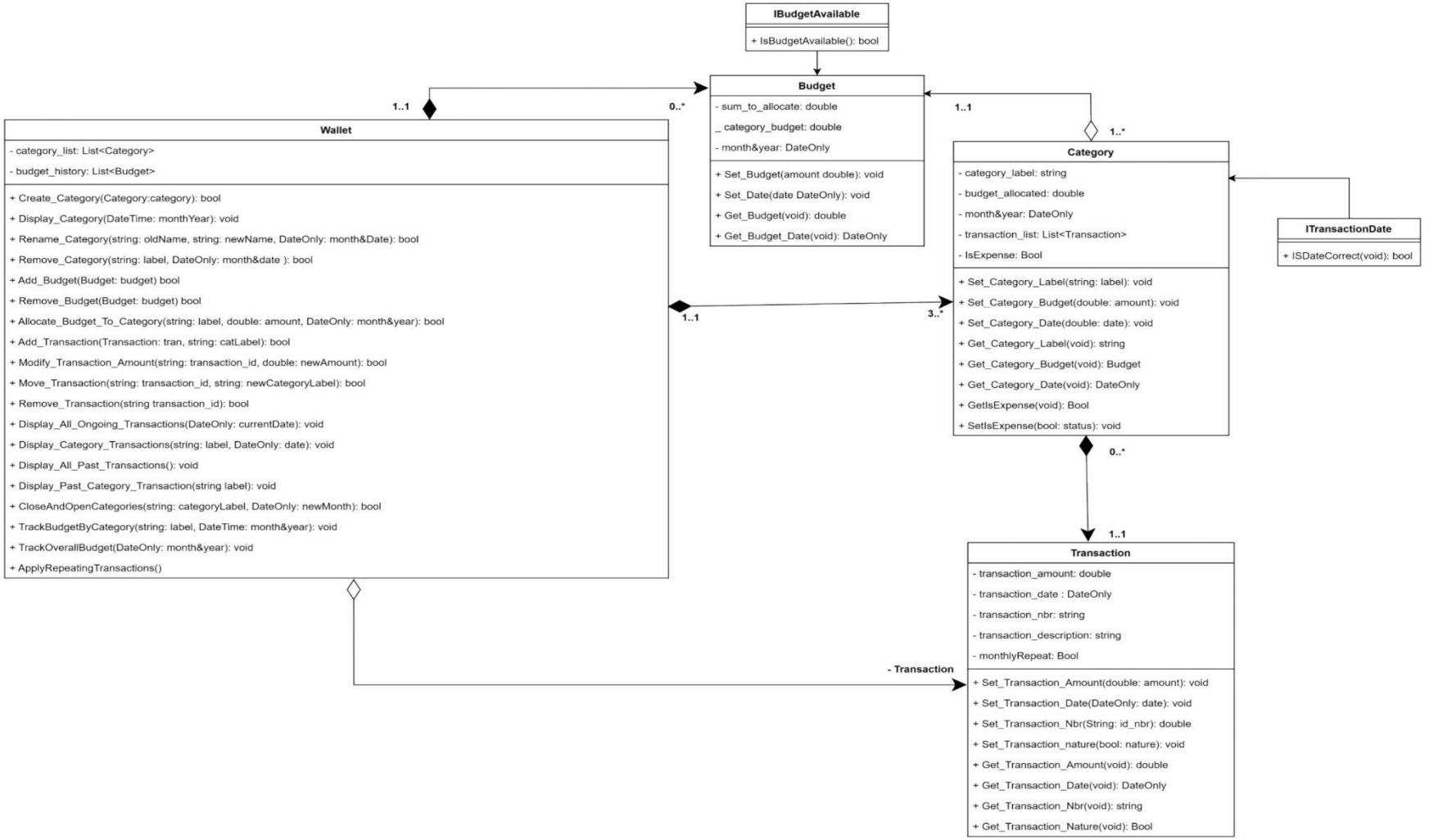
I did the full development of this application as a developer. While developing, I suggested some better designs to our designer, and we had a discussion, and then we modified our system. Even after every test done by the tester, I worked on the bugs and solved every issue. I also helped my team with version control in Git Hub.

Charles' contribution:

As a system designer, my task involved reflecting about the early required functionalities of our personal finance tracking application and to build a first prototype of the project to understand the most essential features, entities and data used to implement a functional finished product. In order to ensure delivering a functional UML design, I collaborated and discussed with my fellow team members on elements that could potentially be either removed or improved for our project. My task is also required to continuously redefine the design throughout our development process.

APPENDIX

UML Design:


IBudgetAvailable
+ IsBudgetAvailable(): bool
Budget
- sum_to_allocate: double
_ category_budget: double
- month&year: DateOnly
+ Set_Budget(amount double): void
+ Set_Date(date DateOnly): void
+ Get_Budget(void): double
+ Get_Budget_Date(void): DateOnly
1..1
0..*
1..1
1..*
Wallet
- category_list: List<Category>
- budget_history: List<Budget>
+ Create_Category(Category:category): bool
+ Display_Category(DateTime: monthYear): void
+ Rename_Category(string: oldName, string: newName, DateOnly: month&Date): bool
+ Remove_Category(string: label, DateOnly: month&date): bool
+ Add_Budget(Budget: budget) bool
+ Remove_Budget(Budget: budget) bool
+ Allocate_Budget_To_Category(string: label, double: amount, DateOnly: month&year): bool
+ Add_Transaction(Transaction: tran, string: catLabel): bool
+ Modify_Transaction_Amount(string: transaction_id, double: newAmount): bool
+ Move_Transaction(string: transaction_id, string: newCategoryLabel): bool
+ Remove_Transaction(string transaction_id): bool
+ Display_All_Ongoing_Transactions(DateOnly: currentDate): void
+ Display_Category_Transactions(string: label, DateOnly: date): void
+ Display_All_Past_Transactions(): void
+ Display_Past_Category_Transaction(string label): void
+ CloseAndOpenCategories(string: categoryLabel, DateOnly: newMonth): bool
+ TrackBudgetByCategory(string: label, DateTime: month&year): void
+ TrackOverallBudget(DateOnly: month&year): void
+ ApplyRepeatingTransactions()
1..1
3..*
Category
- category_label: string
- budget_allocated: double
- month&year: DateOnly
- transaction_list: List<Transaction>
- IsExpense: Bool
+ Set_Category_Label(string: label): void
+ Set_Category_Budget(double: amount): void
+ Set_Category_Date(double: date): void
+ Get_Category_Label(void): string
+ Get_Category_Budget(void): Budget
+ Get_Category_Date(void): DateOnly
+ GetIsExpense(void): Bool
+ SetIsExpense(bool: status): void
ITransactionDate
+ ISDateCorrect(void): bool
0..*
1..1
- Transaction
Transaction
- transaction_amount: double
- transaction_date : DateOnly
- transaction_nbr: string
- transaction_description: string
- monthlyRepeat: Bool
+ Set_Transaction_Amount(double: amount): void
+ Set_Transaction_Date(DateOnly: date): void
+ Set_Transaction_Nbr(String: id_nbr): double
+ Set_Transaction_nature(bool: nature): void
+ Get_Transaction_Amount(void): double
+ Get_Transaction_Date(void): DateOnly
+ Get_Transaction_Nbr(void): string
+ Get_Transaction_Nature(void): Bool

TEST PLAN VERSION 1

Test Number	Test Case Type	Description	Test Steps	Test Data	Expected Results	Actual Results	Status
1	Functionality	Program correctly allows users to add transaction.	Selecting “Add Transaction” from transaction Menu and then inserting it into “Internet” category. Then navigating to Menu and clicking on “View Transactions”.	Transaction for category label: “Utilities”. Amount: “45”. Transaction date (DD/MM/YYYY): “11/03/2024”.	The program should allow the correct input fields to be inputted and then correctly display the information inputted into “Utilities” category from the viewing transaction feature.	The program created the transaction flawlessly and displayed it back to us with the correct information.	Pass
2	Functionality	Verifying viewing of transaction feature, so inserting a transaction and then checking to see if it updates on the view transaction feature.	Inserting a transaction from “Transaction Menu”, traversing back to Menu, and clicking on the view transaction feature h	Transaction for category label: “Utilities”. Amount: “65”. Transaction date (DD/MM/YYYY): ”08/03/2024”. Transaction Description: “Phone bill”.	The program should let you view the transaction you just created and any other transaction that was there already.	The newly created transaction was made with ease and viewing it also confirmed its creation, so the viewing feature is working as expected.	Pass
3	Functionality	This test involves moving a transaction from one category to another category	Navigating to “Transaction”, using the move transaction feature, and moving the phone bill related	Transaction Description: “Phone bill”.	The program should let you move the transaction from one category to another without issue and then viewing it	Creating the transaction went as excepted and moving it from one category to another went smoothly	Pass

		and then clicking on view transaction to see what category it is in, using data already in the program of a transaction.	transaction from “Utilities” category to “Luxury” category.	New assigned category: “Luxury”.	shows the new category set to “Luxury”.	resulting in a successful test.	
4	Functionality	Testing the modification of transaction feature, so changing the amount the transaction used from x to x, firstly verifying the phone bill transaction is in the program and then attempting to change it.	Navigating to “Transaction”, clicking on “Display All Ongoing Transaction”. Then using the update feature to change amount. Finally, then clicking on “Display All Ongoing Transaction”.	Transaction used: “Phone Bill”. Transaction amount: “65”. Transaction ID: “TR00000000”. New transaction amount: “50”.	Amount used in created transaction should change from “65” to “50” and clicking on the view transaction feature list should verify this by displaying that change.	The creation of the transaction “Phone Bill” went smoothly but unexpected error arose trying to update the transaction amount, which resulted in the program saying, “Make sure the Transaction ID is correct”, even though it was correct.	Fail
5	Functionality	Removal of transaction working as intended, so adding a new transaction, assigning it to “Utilities” categories and then verifying it was made by viewing it in the view transaction feature.	Navigating to transaction menu and creating the new transaction. Then navigating to “Display All Ongoing Transactions”. Then navigating back to Menu and removing that transaction through “Delete” transaction.	Transaction category assigned to: “Utilities”. Transaction amount: “80”. Transaction Description: “Water bill”. Transaction date	The program should allow the deletion of transaction, so verifying that the created transaction was added correctly to “Utilities” by adding it and then viewing it in the view transaction feature and then removing it and checking the view transaction feature again to see its removal.	This test went as expected, through the creation of the transaction and the removal of it, successful test.	Pass

		Once verified, move to the removal of it and then the viewing of all transaction to see its removal.	Then navigating to “Display All Ongoing Transactions”.	(DD/MM/YYYY): ”05/03/2024”. Transaction ID: “TR00000000”.			
6	Functionality	This test will involve checking to see if the budget feature works as intended, and an input of income higher than the budget will result in an error message. Firstly, we’d need to create a category and a new transaction to test this.	Navigating to “Category”, creating a new category, then navigating to “Transaction” and creating a transaction that belongs to the newly created category, assigning a transaction amount exceeding the budget for the category.	Category label: “Essentials”. Category budget: “350”. Category date (MM/YYYY): “03/2024” Transaction label: “iPhone 15”. Transaction amount: “1100” Transaction date (DD/MM/YYYY): “10/03/2024”	Progressing through the creation of the category should pose no issues, once creating the transaction, there should be an error when attempting to exceed the limit of the budget of assigned category, so some form of recognition from the program stating what has occurred and or an error message.	Creating the category and assigning it a budget poses no problems but when it comes to creating the transaction, the program does not care for the budget set by the category it is being assigned to, and so the exceeding budget transaction is allowed to occur and without any error message too, which should not be allowed.	Fail
7	Functionality	Checking to see if the preset categories exist and the feature isn’t empty.	Navigating to “Categories”.	Category key to view all categories: “4”.	After navigating to categories feature and using the view function, the program should display a preset of categories	The program does not contain any list of preset categories from viewing all categories for the current month when starting the	Fail

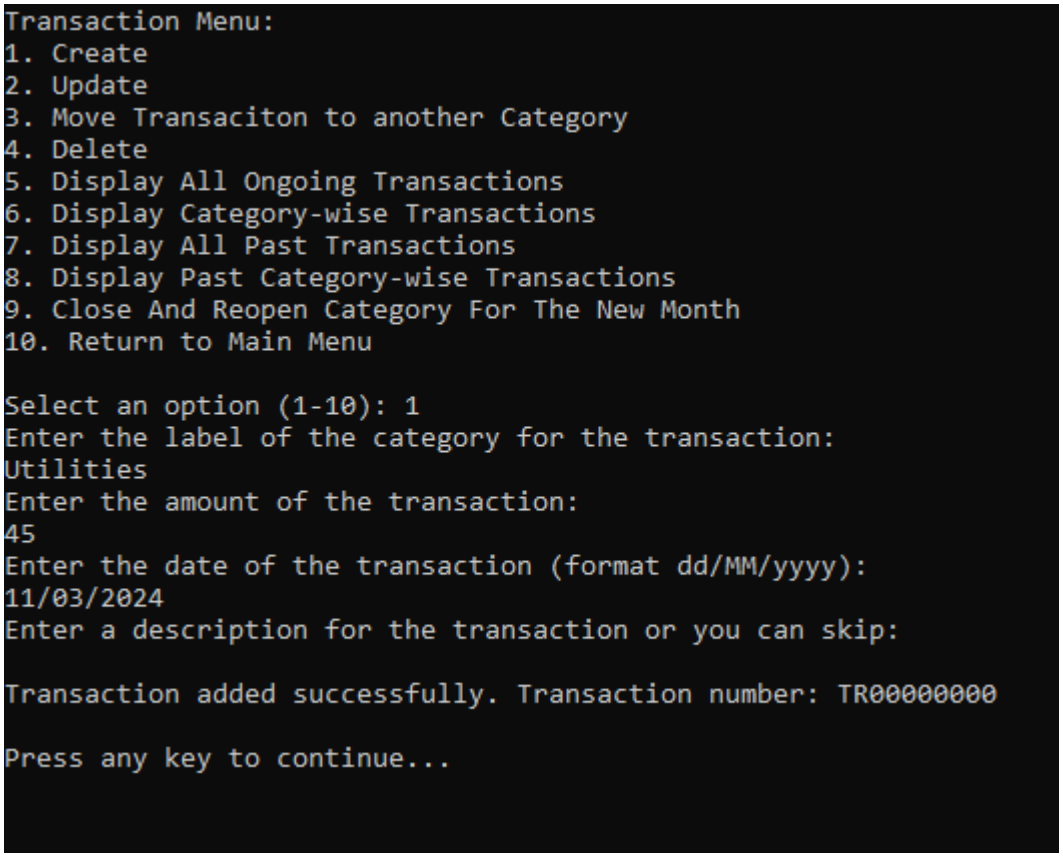
					and the list should not be empty	program from scratch, this was unexpected and is a failed test.	
8	Functionality	Addition of a category and then clicking view categories to see its existence.	Navigating to "Categories", inserting category. Clicking on view categories and viewing category created.	Category label: "Internet" Budget allocated: "150". Date allotted (DD/MM/YYYY): "03/2024".	After navigating to "Categories" feature, the program should display a preset of categories and the list should not be empty	The creation of a new category and then the viewing of went exactly as expected, working without any issues.	Pass
9	Functionality	The modification to a categories' name, seeing if it is working as intended, so creating a new category then changing its name, then viewing list categories to verify the change.	Navigating to "Categories", adding a new category, then clicking view all categories. Then start renaming the categories and again clicking view all categories.	Category label: "Mobile". Budget allocated: "60". Date allotted (MM/YYYY): "01/2024". Category label renamed to: "Sim".	The program should let you add a new category, then after seeing it added, renaming it from "Mobile" to "Sim" should pose no issue then verifying the change through viewing all categories and seeing the change being made faithfully.	Creating the category "Mobile" went as expected functioning correctly, but the issue came when attempting to change category to "Sim" as the program did not recognise the change wanting to be made, unexpected and a failure.	Fail
10	Functionality	The removal of a category test, so adding a new category, verifying it was created through viewing it and then removing it and viewing the category list to see its removal.	Navigating to "Categories", adding a new category, then clicking view all categories. Then navigate back to categories and remove the category	Category label: "Internet". Budget allocated: "150". Date allotted (MM/YYYY): "03/2024".	The addition of a new category should be created with no hassle and then the program should allow you to view that category created, once created the removal of that category should remove it and then viewing that month's category in which it was	Creation of a new category went successfully and the viewing of it, to verify it was made and then the removing of it and checking if it was removed also went as expected.	Pass

			created then click on view categories.		created should display its absence.		
11	Usability	Program Menu allows smooth navigation, allowing users to click on a feature and allowing them to return to the Menu.	Navigating to Menu and clicking on “Transaction” and then Returning to Menu. Then navigating to “Category” and then Returning to Menu. Then navigating to “Budget” and then Returning to Menu.	“Transaction” traversal key: “1”. “Transaction” return to menu key: “10”. “Category” traversal key: “2”. “Category” return to menu key: “5”. “Budget” traversal key: “3”. “Budget” return to menu key: “”.	Navigation is smooth and the result should indicate that the traversal back to the menu occurs without any trouble.	Traversal of the program was smooth and all working correctly and so the it was as expected.	Pass
12	Usability	Simply checking if the program closes correctly from the main Menu.	Navigating to the Menu and then inserting the key to exit the program	Exit key from Menu: “4”.	The program should just close and nothing else occurs after entering the exit key	The program loaded correctly and then simply exiting it worked and it closed the program instantly.	Pass

Test Screenshots

Test Number 1:

Creating a transaction:



Viewing that transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
11/03/2024
Displaying all transactions for 11/03/2024:
Category Name: Utilities
Transaction info: TR00000000,Date:11/03/2024 00:00:00, Amount:45,Note:

Press any key to continue...
```

Test Number 2:

Creating a transaction for “Utilities”:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Utilities
Enter the amount of the transaction:
65
Enter the date of the transaction (format dd/MM/yyyy):
08/03/2024
Enter a description for the transaction or you can skip:
Phone bill
Transaction added successfully. Transaction number: TR00000000

Press any key to continue...
```

Viewing that transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
08/03/2024
Displaying all transactions for 08/03/2024:
Category Name: Utilities
Transaction info: TR00000000,Date:08/03/2024 00:00:00, Amount:65,Note:Phone bill

Press any key to continue...
```

Test Number 3:

Current data on phone bill transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
08/03/2024
Displaying all transactions for 08/03/2024:
Category Name: Utilities
Transaction info: TR00000000,Date:08/03/2024 00:00:00, Amount:65,Note:Phone bill
Category Name: Luxury

Press any key to continue...
```

Changing the category for the transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 3
Enter the transaction ID you wish to move:
TR00000000
Enter the label of the new category for this transaction:
Luxury
Transaction successfully moved to the new category.

Press any key to continue...
```

Viewing the transactions to see the change from “Utilities” to “Luxury”:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
08/03/2024
Displaying all transactions for 08/03/2024:
Category Name: Utilities
Category Name: Luxury
Transaction info: TR00000000,Date:08/03/2024 00:00:00, Amount:65,Note:Phone bill

Press any key to continue...
```

Test Number 4:

Transaction descriptive view:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
08/03/2024
Displaying all transactions for 08/03/2024:
Category Name: Utilities
Category Name: Luxury
Transaction info: TR00000000,Date:08/03/2024 00:00:00, Amount:65,Note:Phone bill

Press any key to continue...
```

Changing the amount from “65” to “50”:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 2
Enter the transaction ID:
TR00000000
Enter the new amount for the transaction:
50
Failed to update transaction amount. Make sure the transaction ID is correct.

Press any key to continue...
```

Test Number 5:

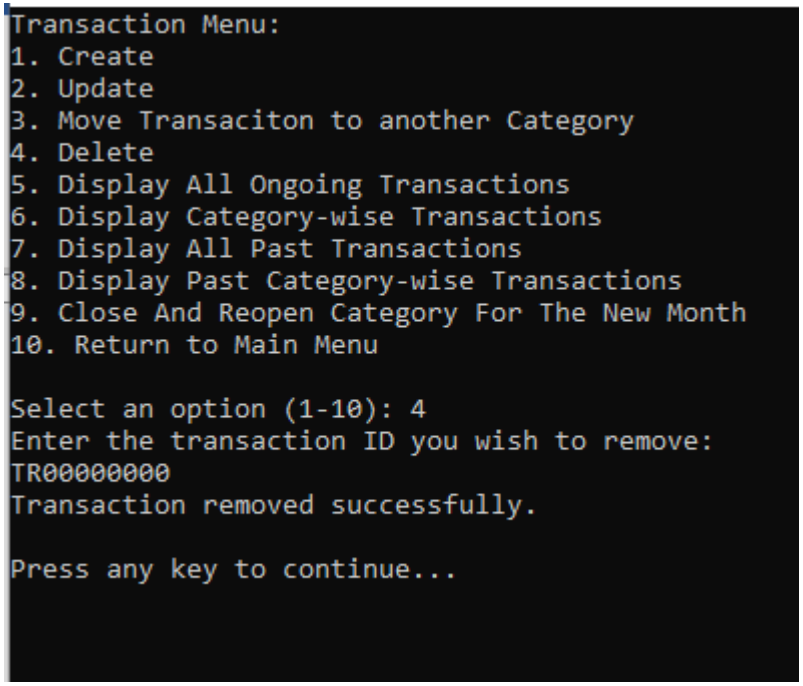
Transaction creation:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Utilities
Enter the amount of the transaction:
80
Enter the date of the transaction (format dd/MM/yyyy):
05/03/2024
Enter a description for the transaction or you can skip:
Water Bill
Transaction added successfully. Transaction number: TR00000000

Press any key to continue...
```

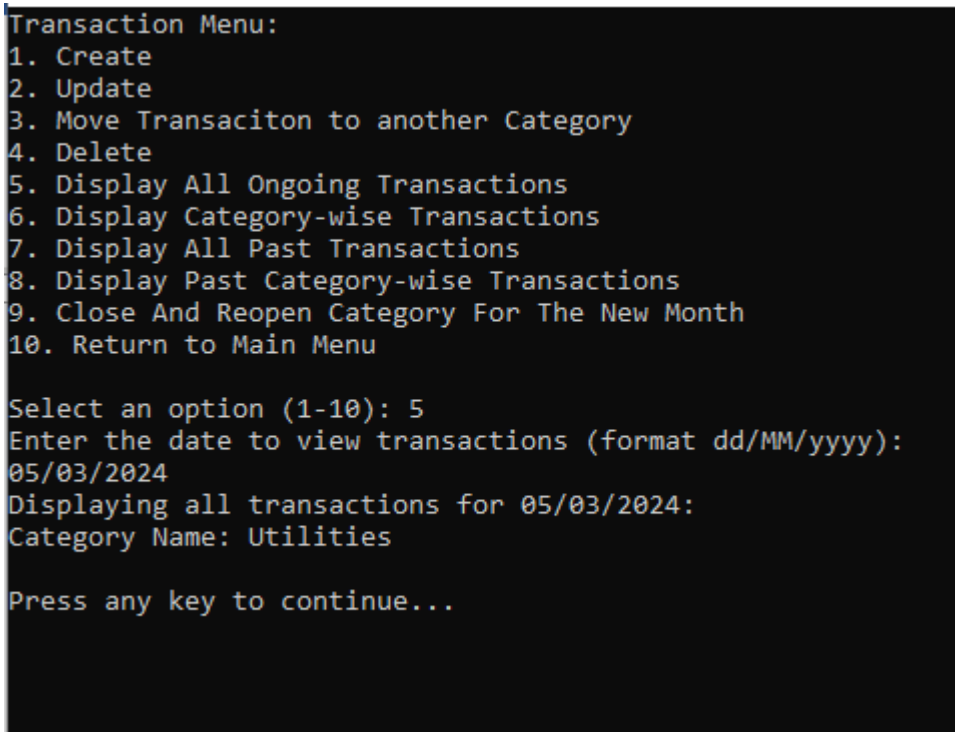
Viewing of transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
05/03/2024
Displaying all transactions for 05/03/2024:
Category Name: Utilities
Transaction info: TR00000000,Date:05/03/2024 00:00:00, Amount:80,Note:Water Bill

Press any key to continue...
```

Removal of transaction:

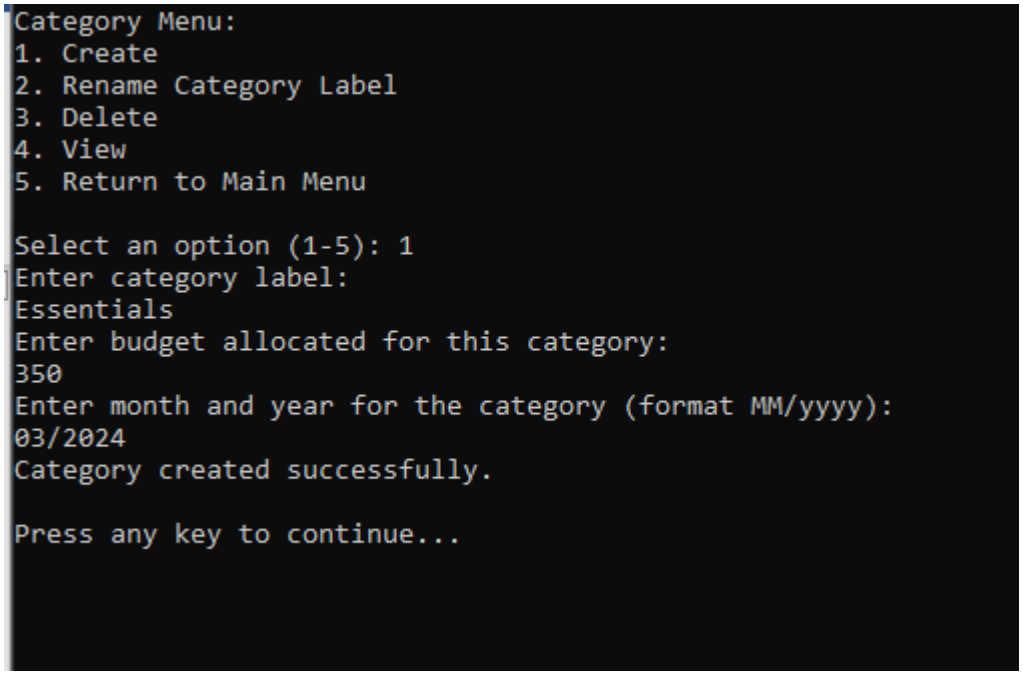


Viewing of all ongoing transactions:



Test Number 6:

Creating the category:



Creating the transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Essentials
Enter the amount of the transaction:
1100
Enter the date of the transaction (format dd/MM/yyyy):
10/03/2024
Enter a description for the transaction or you can skip:
iPhone 15
Transaction added successfully. Transaction number: TR00000000

Press any key to continue...
```

Viewing transaction from “Display All Ongoing Transactions”:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
10/03/2024
Displaying all transactions for 10/03/2024:
Category Name: Essentials
Transaction info: TR00000000,Date:10/03/2024 00:00:00, Amount:1100,Note:iPhone 15

Press any key to continue...
```

Test Number 7:

Entering the current date into the view categories feature:

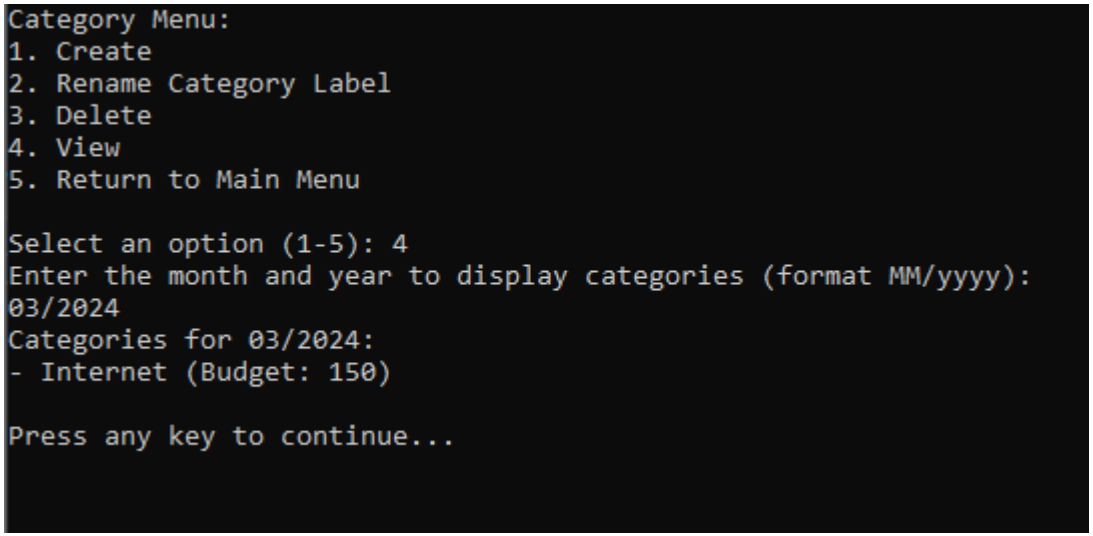
```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to display categories (format MM/yyyy):
03/2024
No categories found for the specified month and year.

Press any key to continue...
```

Test Number 8:

Creation of category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Internet
Enter budget allocated for this category:
150
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Category being displayed in the view feature:



Test Number 9:

Creation of category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Mobile
Enter budget allocated for this category:
60
Enter month and year for the category (format MM/yyyy):
01/2024
Category created successfully.

Press any key to continue...
```

Viewing of newly created category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to display categories (format MM/yyyy):
01/2024
Categories for 01/2024:
- Mobile (Budget: 60)

Press any key to continue...
```

Attempting to rename the category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 2
Current month's categories:
No categories found for the specified month and year.
Enter the label of the category you wish to rename:
Mobile
Enter the new label for the category:
Sim
Failed to rename category. Make sure it exists and you entered the correct date.

Press any key to continue...
```

Test Number 10:

Creation of category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Internet
Enter budget allocated for this category:
150
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Category being displayed in the view feature:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Internet (Budget: 150)

Press any key to continue...
```

Removal of category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 3
Enter the label of the category you want to remove:
Internet
Enter the month and year of the category to remove (format MM/yyyy):
03/2024
Category successfully removed.

Press any key to continue...
```

Verifying its removal through viewing categories:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to display categories (format MM/yyyy):
03/2024
No categories found for the specified month and year.

Press any key to continue...
```

Test Number 11:

Traversing to “Transaction”:

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Exit

Select an option (1-4): 1
```

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10):
```

Traversing back to menu:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 10
```

Traversing to “Category”:

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Exit

Select an option (1-4): 2
```

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5):
```

Traversing back to menu:

```
Category Menu:
1. Create
2. Rename Category Label
3. Delete
4. View
5. Return to Main Menu

Select an option (1-5): 5
```

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Exit

Select an option (1-4):
```

Traversing to “Budget”:

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Exit

Select an option (1-4): 3
```

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5):
```

Traversing back to menu:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 5
```

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Exit

Select an option (1-4):
```

Test Number 12:

Entering the exit button key on the menu:

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Exit

Select an option (1-4): 4
```

Notes:

- Create a list of preset categories
- Transacation menu number 6 is buggy and or just not working
- Attempt to not require exact dates when wanting to view transactions
- Changing of category shouldnt require a long 10 digit code

- The TR code is always the same, set to “TR00000000” does not change even with new transactions
- Why can I create transaction without having set a budget to a category through the budget option, unless the budget is truly assigned from the creation of the category itself
- Option “4” from the budget menu doesn’t seem to function correctly

TEST PLAN VERSION 1 RETESTING

Previous failed test retesting:

Test Number	Test Case Type	Description	Test Steps	Test Data	Expected Results	Actual Results	Status
4	Functionality	Testing the modification of transaction feature, so changing the amount the transaction used from x to x, firstly verifying the phone bill transaction is in the program and then attempting to change it, this will fall under a new category created called “Luxury”.	Navigating to “Category”, creating a new category. Then navigating to “Transaction”, clicking on “Display All Ongoing Transaction”. Then using the update feature to change amount. Finally, then clicking on “Display All Ongoing Transaction”.	Category label: “Luxury”. Category budget: “200”. Category date (MM/YYYY): “03/2024” Transaction used: “Phone Bill”. Transaction amount: “65”. New transaction amount: “50”. Transaction date	The creation of the category should proceed smoothly with no errors and then the amount used in created transaction should change from “65” to “50” and clicking on the view transaction feature list should verify this by displaying that change.	The result was correctly created category but incorrect transaction creation, the program did not recognise “Luxury” as either being created in the same month or being created at all and so the transaction was not even allowed to be created so the final testing of the modification couldn’t fully be tested as failure occurred in the creation process.	Fail

				(DD/MM/YYYY): "07/03/2024"			
6	Functionality	This test will involve checking to see if the budget feature works as intended, and an input of income higher than the budget will result in an error message. Firstly, we'd need to create a category and a new transaction to test this.	Navigating to "Category", creating a new category, then navigating to "Transaction" and creating a transaction that belongs to the newly created category, assigning a transaction amount exceeding the budget for the category.	Category label: "Essentials". Category budget: "350". Category date (MM/YYYY): "03/2024" Transaction label: "iPhone 15". Transaction amount: "1100" Transaction date (DD/MM/YYYY): "10/03/2024"	Progressing through the creation of the category should pose no issues, once creating the transaction, there should be an error when attempting to exceed the limit of the budget of assigned category, so some form of recognition from the program stating what has occurred and or an error message.	The feature works exactly as intended as the creation of the category went smoothly, with the budget that we wanted being assigned correctly and then the transaction creation of trying to exceed that budget with its expense amount did result in an error message and so this problem has been solved.	Pass
7	Functionality	Checking to see if the preset categories exist and the feature isn't empty.	Navigating to "Categories".	Category key to view all categories: "5".	After navigating to categories feature and using the view function, the program should display a preset of categories and the list should not be empty.	The program now correctly displays a list of preset categories when running the program from scratch, working as expected now.	Pass
9	Functionality	The modification to a categories' name,	Navigating to "Categories", adding	Category label: "Mobile".	The program should let you add a new category, then after	The result was not as expected, the program does	Fail

		seeing if it is working as intended, so creating a new category then changing its name, then viewing list categories to verify the change.	a new category, then clicking view all categories. Then start renaming the categories and again clicking view all categories.	Budget allocated: “60”. Date allotted (MM/YYYY): “01/2024”. Category label renamed to: “Sim”.	seeing it added, renaming it from “Mobile” to “Sim” should pose no issue then verifying the change through viewing all categories and seeing the change being made faithfully.	not let you change a category if its date is not set to the current month’s date, when amending the date to the current date it works as expected but the test data used is of (MM/YYYY) “01/2024” and not “03/2024” which caused conflict	

Test Notes:

Test case 4 was altered slightly to include creation of a category as in the original testing, it was done in succession to another test that had created a category and as now this was being tested solely, the creation aspect was needed and so that’s a small change made.

Test Screenshots

Test number 4:

Creating the category to be used in this test called “Luxury”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Luxury
Enter budget allocated for this category:
200
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Creation of transaction with budget set to “65”:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Luxury
Enter the amount of the transaction:
65
Enter the date of the transaction (format dd/MM/yyyy):
07/03/2024
Enter a description for the transaction or you can skip:
Phone Bill
No budget found for the current month. Transaction cannot be added.

Press any key to continue...
```

Re-verifying that the category created has the correct budget and date:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 5
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)
- Luxury (Budget: 200)

Press any key to continue...
```

Test number 6:

Creation of category “Essential”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Essentials
Enter budget allocated for this category:
350
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Creation of transaction “iPhone 15” with exceeding expense of category being assigned to:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Essentials
Enter the amount of the transaction:
1100
Enter the date of the transaction (format dd/MM/yyyy):
10/03/2024
Enter a description for the transaction or you can skip:
iPhone 15
No budget found for the current month. Transaction cannot be added.

Press any key to continue...
```

Test number 7:

Checking to see if there are a preset of categories:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 5
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)

Press any key to continue...
```

Test number 9:

Creation of category “Mobile”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Mobile
Enter budget allocated for this category:
60
Enter month and year for the category (format MM/yyyy):
01/2024
Category created successfully.

Press any key to continue...
```

Renaming of newly created category “Mobile” to “Sim”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 2
Current month's categories:
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)
Enter the label of the category you wish to rename:
Mobile
Enter the new label for the category:
Sim
Failed to rename category. Make sure it exists and you entered the correct date.

Press any key to continue...
```

Creating the new category using the current month's date:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Mobile
Enter budget allocated for this category:
60
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Renaming that category from “Mobile” to “Sim”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 2
Current month's categories:
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)
- Mobile (Budget: 60)
Enter the label of the category you wish to rename:
Mobile
Enter the new label for the category:
Sim
Category renamed successfully.

Press any key to continue...
```

Viewing all categories to see the change being made:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 5
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)
- Sim (Budget: 60)

Press any key to continue...
```

Amended test retesting:

Test Number	Test Case Type	Description	Test Steps	Test Data	Expected Results	Actual Results	Status
4	Functionality	Testing the modification of transaction feature, so changing the amount the transaction used from x to x, firstly verifying the phone bill transaction is in the program and	Navigating to “Category”, creating a new category. Then navigating to “Budget” and assigning a budget for the month “Luxury” was created for.	Category label: “Luxury”. Category budget: “200”. Category date (MM/YYYY): “03/2024”.	The creation of the category should proceed smoothly with no errors, the allocation of a monthly budget for “Luxury” month should proceed smoothly too and then the amount used in created transaction should change from “65” to “50” and clicking on the view transaction feature	The result occurred as expected the creation of a category and then a budget creation for that month went as intended, so did the creation of the transaction and finally the modification of the transaction.	Pass

		then attempting to change it, this will fall under a new category created called “Luxury”, as well assigning a budget for the category’s month.	Then navigating to “Transaction”, clicking on “Display All Ongoing Transaction”. Then using the update feature to change amount. Finally, then clicking on “Display All Ongoing Transaction”.	Budget assigned for the month: “1000”. Budget date: (MM/YYYY): “03/2024”. Transaction used: “Phone Bill”. Transaction amount: “65”. New transaction amount: “50”. Transaction date (DD/MM/YYYY): “07/03/2024”	list should verify this by displaying that change.		

Test Notes:

Test case 4 was altered yet again, due to a misunderstanding I had, the test was now performed using different test data, so assigning a budget to a category and then making sure the category created as a monthly budget, so category creation, then budget creation and then the true testing of the modification of transaction amount.

Test Screenshots

Test number 4:

Creating a category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Luxury
Enter budget allocated for this category (optional):
200
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Assigning a budget for the month:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
1000
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Creating transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Luxury
Enter the amount of the transaction:
65
Enter the date of the transaction (format dd/MM/yyyy):
07/03/2024
Enter a description for the transaction or you can skip:
Phone Bill
For Luxury in 03/2024, you have spent 65 out of your budget of 200.
Transaction added successfully. Transaction number: TRe877bced

Press any key to continue...
```

Changing transaction amount:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 2
Enter the transaction ID:
TRe877bced
Enter the new amount for the transaction:
50
For Luxury in 03/2024, you have spent 50 out of your budget of 200.
Transaction amount updated successfully.

Press any key to continue...
```

TEST PLAN VERSION 2

Test Number	Test Case Type	Description	Test Steps	Test Data	Expected Results	Actual Results	Status
1	Functionality	Moving a transaction to another category, so creating two new categories,	Navigate to “Categories”, creating two new categories.	Category label: “Essentials”. Category budget: “400”.	The creation of the categories should occur without fail and then the assigning of a budget to their retrospective month, creation of transaction and the	The program allowed for correct creation of categories and budget, then for the transaction, its creation proceeded as	Pass

		both in the same month, and assigning a budget that exceeds their total combined and then creation of a transaction and assigning it to one of newly created categories and then the attempt to move it to the other category created.	Then navigating to transaction and creating a new transaction. Moving the transaction to another category. Viewing all ongoing transactions.	Category date (MM/YYYY): "03/2024" Category label: "Luxury". Category budget: "200". Category date (MM/YYYY): "03/2024" Budget amount: "1000" Transaction used: "Netflix Subscription". Transaction amount: "25". Transaction date (DD/MM/YYYY): "15/03/2024"	changing of its category it belongs to should also pose no issues.	intended and changing its category worked as intended too, successful test.	
2	Functionality	Display all ongoing transactions, initially, so creating transactions,	Navigate to "Budget" and assign a budget, then navigate to "Category" and set a	Budget for the month: "500"	The program should let you create a budget for a month, let you change a category's budget and then create more	This actual result was a success, creation of budgets and assignment of category's budget worked,	Pass

		assigning a category budget and a monthly budget, and checking the feature that displays all ongoing transactions.	budget for a category and then create two transactions and then view them using display all ongoing transactions feature.	Budget date (DD/MM/YYYY): "03/2024" Category used: "Food". Category budget: "200" Transaction used: "Water Bottles". Transaction amount: "8". Transaction date (DD/MM/YYYY): "01/03/2024" Transaction used: "Restaurant". Transaction amount: "25". Transaction date (DD/MM/YYYY): "06/03/2024"	than a single transaction. Then finally seeing the transactions from the displaying of all ongoing transactions.	so did the creation of the transactions and then the viewing of them too from the view all ongoing transactions feature.	
3	Functionality	Displaying all transaction a certain category	Navigating to "Budget", assigning a budget, then	Budget for the month: "500"	The program should let you create a budget and assign a budget to a preset category	The test went exactly as planned, allowing correct creation and modifications	Pass

		has and so creating a budget for a month and changing a preset category's budget and then finally creating a transaction for the category we want to view transactions for and then using the "Category-wise Transactions" feature to test it.	navigating to "Category" and allocating "Food" a budget. Navigating back to categories menu and clicking on view "Category-wise transaction".	Budget date (DD/MM/YYYY): "03/2024" Category used: "Food". Category budget: "200" Transaction used: "Water Bottles". Transaction amount: "8". Transaction date (DD/MM/YYYY): "01/03/2024"	and then allow the creation of a transaction. Then the main point of showing all transactions a certain category has, and the category we used in the test data should contain the transaction we assigned to it and this should be display in the "Category-wise Transaction" viewing feature.	of budget, category and transaction and then finally the viewing of a category's transaction in which we created a transaction for, and it displayed exactly that.	
4	Functionality	Allocation of budget to a category, so the point of this test is to make sure that the updating of any category's budget can be updated, this is a needed test as the preset categories are set to "0" budget and so this test will ensure at a base level the program	Navigate to "Categories" and change the amount of preset category "Food" to "200". Then navigate to the view feature to see the change implemented.	Category budget amount: "200".	The test should allow for changing of a preset category's budget and then allow correct budget allocation for the current month, then viewing the categories list to see the change being made.	The test went as expected, allowing the changing of a preset category's budget.	Pass

		can run without needing to create a category.					
5	Functionality	Creation of a category that already exists should result in an error message, so same labelled names shouldn't be allowed, so this test will involve creating a category that already exists (preset category).	Navigate to "Category" and create a category. Then use the view categories feature.	Category label: "Food". Category date (DD/MM/YY): "03/2024".	The test should come up with some form of error message saying, "duplicate value being used" or something like indicate name change must occur.	The test failed and did not go as expected, the program allows for creation of category that share the same label.	Fail
6	Functionality	viewing of budget feature, so checking the budget for a month, starting with assigning a budget for a month and then verifying it from the viewing feature.	Navigate to "Budget" and assign a budget for a date. Then use the viewing feature to see budget for a month.	Budget amount: "350". Budget date (MM/YYYY): "01/2022".	This test should allow you to set a budget for a certain date and then check to see what a certain date's budget is, so confirming the budget initially set does work as intended.	The test went exactly as expected with the budget feature working accordingly to the test.	Pass
7	Functionality	Removal of a budget for a month, so assigning a budget for the current month, checking it was created and then removing that budget and checking it was removed, the point	Navigate to "Budget" and assign a budget for a date. Then use the viewing feature to see budget for a month. Then using the remove feature to now remove it.	Budget amount: "350". Budget date (MM/YYYY): "01/2022".	The program should allow easy and correct creation of a budget and the viewing of it and then also easy removal of a month's budget and then the viewing of that month's budget to see its removal having occurred.	The test went as planned, was creation and removal of the budget working and seeing verifying the creation and removal using the viewing feature also working as expected.	Pass

		of this test is to verify the budget feature works as needed, as it is a core aspect of the program.	Then use the viewing feature to see its removal.				

Test Notes:

When wanting to view transactions etc. I think it is a good idea to only ask for month and year, instead of specific dates as you are going to display the entire month and not information on a specific day.

Test Screenshots

Test number 1:

Creation of category “Essentials”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Essentials
Enter budget allocated for this category (optional):
400
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Creation of category “Luxury”:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Luxury
Enter budget allocated for this category (optional):
200
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Creation of budget:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
1000
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Creation of transaction, assigning to “Essentials” category:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Essentials
Enter the amount of the transaction:
25
Enter the date of the transaction (format dd/MM/yyyy):
15/03/2024
Enter a description for the transaction or you can skip:
Netflix Subscription
For Essentials in 03/2024, you have spent 25 out of your budget of 400.
Transaction added successfully. Transaction number: TRaaebe2fe

Press any key to continue...
```

Changing of category for created transaction from “Essentials” category to “Luxury” category:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 3
Enter the transaction ID you wish to move:
TRaaebe2fe
Enter the label of the new category for this transaction:
Luxury
Transaction successfully moved to the new category.

Press any key to continue...
```

Viewing all on going transactions to see the change:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
15/03/2024
Displaying all transactions for 15/03/2024:
Category Name: Food
Category Name: Transportation
Category Name: Entertainment
Category Name: Bills
Category Name: Essentials
Category Name: Luxury
---Transaction info: TRaaebe2fe,Date:15/03/2024, Amount:25,Note:Netflix Subscription

Press any key to continue...
```

Test number 2:

Assigning a budget for the month to be used:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
500
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Assigning a budget to the category to be used:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 3
Enter the category label:
Food
Enter the month and year for the category (format MM/yyyy):
03/2024
Enter the new budget allocated for this category:
200
Budget updated successfully for category 'Food' for the period 03/2024.

Press any key to continue...
```

Creating transactions:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Food
Enter the amount of the transaction:
8
Enter the date of the transaction (format dd/MM/yyyy):
01/03/2024
Enter a description for the transaction or you can skip:
Water Bottles
For Food in 03/2024, you have spent 8 out of your budget of 200.
Transaction added successfully. Transaction number: TRd5f6226d

Press any key to continue...
```

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Food
Enter the amount of the transaction:
25
Enter the date of the transaction (format dd/MM/yyyy):
06/03/2024
Enter a description for the transaction or you can skip:
Restaurant
For Food in 03/2024, you have spent 33 out of your budget of 200.
Transaction added successfully. Transaction number: TRa4dc354a

Press any key to continue...
```

Displaying all on going:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 5
Enter the date to view transactions (format dd/MM/yyyy):
01/03/2024
Displaying all transactions for 01/03/2024:
Category Name: Food
---Transaction info: TRd5f6226d,Date:01/03/2024, Amount:8,Note:Water Bottles
---Transaction info: TRa4dc354a,Date:06/03/2024, Amount:25,Note:Restaurant
Category Name: Transportation
Category Name: Entertainment
Category Name: Bills

Press any key to continue...
```

Test number 3:

Creating a budget for the date of “03/2024”:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
500
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

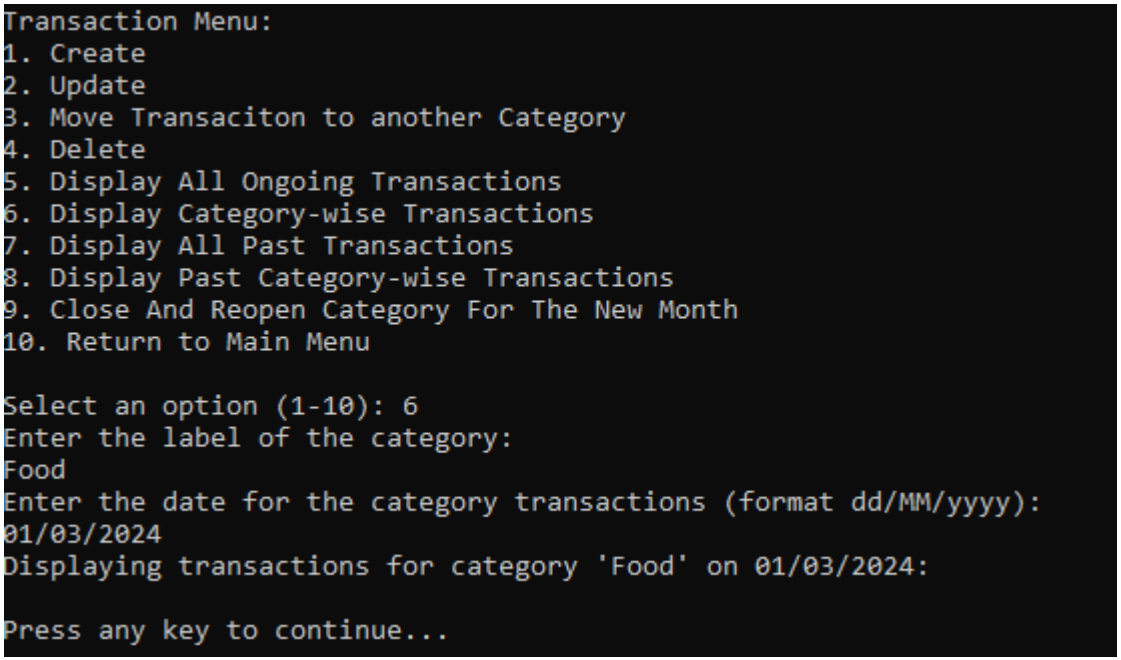
Changing the preset category “Food”’s budget:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 3
Enter the category label:
Food
Enter the month and year for the category (format MM/yyyy):
03/2024
Enter the new budget allocated for this category:
200
Budget updated successfully for category 'Food' for the period 03/2024.

Press any key to continue...
```

Creating transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10. Return to Main Menu

Select an option (1-10): 1
Enter the label of the category for the transaction:
Food
Enter the amount of the transaction:
8
Enter the date of the transaction (format dd/MM/yyyy):
01/03/2024
Enter a description for the transaction or you can skip:
Water Bottles
For Food in 03/2024, you have spent 8 out of your budget of 200.
Transaction added successfully. Transaction number: TR65f5a52e

Press any key to continue...
```

Viewing “Category-wise Transactions”:



Test number 4:

Viewing preset categories:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 5
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)

Press any key to continue...
```

Changing the amount for a category's budget. Assigning "Food" a budget of "200":

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 3
Enter the category label:
Food
Enter the month and year for the category (format MM/yyyy):
03/2024
Enter the new budget allocated for this category:
200
Budget updated successfully for category 'Food' for the period 03/2024.

Press any key to continue...
```

Viewing that change in categories list:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 5
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Food (Budget: 200)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)

Press any key to continue...
```

Test number 5:

Creating a category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Food
Enter budget allocated for this category (optional):
5000
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Viewing of categories list:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 5
Enter the month and year to display categories (format MM/yyyy):
03/2024
Categories for 03/2024:
- Food (Budget: 0)
- Transportation (Budget: 0)
- Entertainment (Budget: 0)
- Bills (Budget: 0)
- Food (Budget: 5000)

Press any key to continue...
```

Test number 6:

Creating a budget:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
350
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Viewing that budget created:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to view the budget (format MM/yyyy):
03/2024
Budget for 03/2024:
- Amount allocated: 350
- Remaining budget: 350

Press any key to continue...
```

Test number 7:

Creating a budget:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
350
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Viewing that budget created:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to view the budget (format MM/yyyy):
03/2024
Budget for 03/2024:
- Amount allocated: 350
- Remaining budget: 350

Press any key to continue...
```

Removal of that budget:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 2
Enter month and year of the budget to remove (format MM/yyyy):
03/2024
Budget removed successfully.

Press any key to continue...
```

Viewing budget for the initial budget assigned date to see if it exists or not:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 4
Enter the month and year to view the budget (format MM/yyyy):
03/2024
No budget found for the specified month and year.

Press any key to continue...
```

Notes

TEST PLAN VERSION 3

Test Number	Test Case Type	Description	Test Steps	Test Data	Expected Results	Actual Results	Status
1	Functionality	The purpose of this test is to create a transaction and check if it can be made to be set to recurring, so creating a budget for the month, assigning a budget to one of the preset categories and then finally creating the transactions and attempting to make it into a recurring transaction.	Navigating to "Budget" setting a budget, then navigating to "Category" and setting a budget for a preset category chosen. Then navigating to "Transaction" and creating a transaction and then turn this newly created transaction into a recurring transaction. Finally, navigate to display transaction feature and view newly created transaction.	Monthly budget: "1000". Category label: "Food". Category budget: "240". Category date (MM/YYYY): "03/2024" Transaction used: "Groceries". Transaction amount: "150". Transaction date (DD/MM/YYYY): "09/03/2024"	The program should allow the creation of a budget for the month and the allocation of it for a preset category and then allow creation of a transaction and the option to set it to recurring, and then finally viewing this change by displaying transactions and seeing the recurring tag be in the transaction description.	The program executed as intended, allowing easy allocation of a budget for a month and category, creation of a transaction, assigning it to a recurring tag and then finally the viewing of this from the display transaction feature.	Pass
2	Functionality	The reasoning for this test is to check if the program allows you to insert an income as an option, as it is one of the core requirements of a budget application, so the test	Navigate to "Category" and insert a label. Select option as not an expense, insert a date.	Income category label: "Bonus". Income date (MM/YYYY): "03/2024".	The program should let you create an income category without asking for a budget since an income category doesn't need a budget because it is an income, and then it should let you	The result was exactly as intended with the creation of the category income and the program not asking for a budget for an income category and then also the creation of	Pass

		will involves creating an income and then viewing it to see if it shows in the viewing feature.		Transaction label: “Storage Sale”. Transaction income value: “350”. Transaction date (DD/MM/YYYY): “01/03/2024”.	create a transaction income and then also let you view that change.	income transaction and the viewing of it went exactly as expected.	
3	Functionality	Testing to see if the budget tracker works correctly, so creation of data that fulfils this purpose, as I want to test if the program shows your expense compared to your budget at the bare minimum, so a user can track how much they have been spending, so creation of transaction and income and seeing it both get displayed.	Navigate to “Budget” and assign a budget for the month and then navigate to “Category” and change the budget for a preset category. Then navigate to “Transaction”, create a transaction and an income and an expense transaction. Finally navigate to the menu and use the budget tracker viewing option.	Budget for the month: “1500”. Budget date (MM/YYYY): “03/2024”. Category used: “Transportation”. Category budget:”45”. Transaction Income value: “350”. Transaction expense value: “7.65”.	The creation of the preset category and budget should be allowed without any issues and so should the creation for an income transaction and an expense transaction and the viewing of it should be seen in the tracking feature, it should also show how much of budget is left from deducting a month’s budget from the expense.	The result went exactly as planned, the creation of the budget for the month, preset category budget, expense and income transaction also went smoothly, then finally the main point of viewing these transactions went smoothly with displaying them correctly and the remaining budget also being displayed correctly.	Pass

				transaction expense label:			
4	Functionality	This test is the testing of an old test from the last version which involved testing if you could create a duplicate category, which shouldn't be allowed, and it was allowing and so this time around the same test will occur.	Navigate to "category" and create a category named after any preset category.	Category label used: "Food".	The test should involve some sort of indication that the creation of this category didn't come to fruition as duplicate category labels shouldn't be allowed.	The test went as planned this time round and the creation of a duplicate category was not accepted, and the program prompted an error message after attempting to do this.	Pass
5	Usability	This test is centred around seeing if the program is readable and has correct colour thematic in place, an example being testing if an error occurs does that message display in red or not, so this test will involve inserting a string into a section that requires numerical value and seeing how easy it is to tell what has gone wrong.	Navigate to "Transaction", create transaction, and use string in numerical input that asks for transaction amount.	Transaction label being used: "Food". Transaction amount: "fifty pounds".	The program should not let you insert string into numerical input field, and it should have a clear visual indication that something has gone wrong.	Exceeded expectations, as visual clarity was met with the error message being prompted in red when attempting to insert string values where numerical values should be instead.	Pass

Test number 1:

Creating budget for the month:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
1000
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Assigning a budget for a preset category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 3
Enter the category label:
Food
Enter the month and year for the category (format MM/yyyy):
03/2024
Enter the new budget allocated for this category:
240
Budget updated successfully for category 'Food' for the period 03/2024.

Press any key to continue...
```

Creating transaction and setting it to monthly:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10.Manage Recurring Transactions
11. Return to Main Menu

Select an option (1-11): 1
Enter the label of the category for the transaction:
Food
Enter the amount of the transaction:
150
Enter the date of the transaction (format dd/MM/yyyy):
09/03/2024
Enter a description for the transaction (optional, press Enter to skip):
Groceries
Is this transaction recurring monthly? (yes/no):
yes
For Food in 03/2024, you have spent 150 out of your budget of 240.
Transaction added successfully. Transaction number: TR2bbc88d3

Press any key to continue...
```

Viewing the transactions list:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10.Manage Recurring Transactions
11. Return to Main Menu

Select an option (1-11): 5
Enter the date to view transactions (format dd/MM/yyyy):
09/03/2024
Displaying all transactions for 09/03/2024:
Category Name: Food
---Transaction info: TR2bbc88d3,Date:09/03/2024, Amount:150,Note:Groceries, Monthly Recurring: Yes
Category Name: Transportation
Category Name: Salaries
Category Name: Bills

Press any key to continue...
```

Test number 2:

Creating income category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Bonus
Is this category for an expense? (yes/no):
no
Enter month and year for the category (format MM/yyyy):
03/2024
Category created successfully.

Press any key to continue...
```

Creating transactional income:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10.Manage Recurring Transactions
11. Return to Main Menu

Select an option (1-11): 1
Enter the label of the category for the transaction:
Bonus
Enter the amount of the transaction:
350
Enter the date of the transaction (format dd/MM/yyyy):
01/03/2024
Enter a description for the transaction (optional, press Enter to skip):
Storage Sale
Is this transaction recurring monthly? (yes/no):
no
Transaction added successfully. Transaction number: TRa64cb050

Press any key to continue...
```

Viewing the transaction:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10.Manage Recurring Transactions
11. Return to Main Menu

Select an option (1-11): 5
Enter the date to view transactions (format dd/MM/yyyy):
01/03/2024
Displaying all transactions for 01/03/2024:
Category Name: Food
Category Name: Transportation
Category Name: Salaries
Category Name: Bills
Category Name: Bonus
---Transaction info: TRa64cb050,Date:01/03/2024, Amount:350,Note:Storage Sale, Monthly Recurring: No

Press any key to continue...
```

Test number 3:

Budget creation:

```
Budget Menu:
1. Create
2. Delete
3. Allocate Budget To Category
4. Check Budget
5. Return to Main Menu

Select an option (1-5): 1
Enter the total sum to allocate for the budget:
1500
Enter month and year for the budget (format MM/yyyy):
03/2024
Budget added successfully.

Press any key to continue...
```

Adjusting preset category:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 3
Enter the category label:
Transportation
Enter the month and year for the category (format MM/yyyy):
03/2024
Enter the new budget allocated for this category:
45
Budget updated successfully for category 'Transportation' for the period 03/2024.

Press any key to continue...
```

Creating transaction for preset category:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10.Manage Recurring Transactions
11. Return to Main Menu

Select an option (1-11): 1
Enter the label of the category for the transaction:
Transportation
Enter the amount of the transaction:
7.65
Enter the date of the transaction (format dd/MM/yyyy):
03/03/2024
Enter a description for the transaction (optional, press Enter to skip):
Train Expense
Is this transaction recurring monthly? (yes/no):
no
For Transportation in 03/2024, you have spent 7.65 out of your budget of 45.
Transaction added successfully. Transaction number: TRa0508123

Press any key to continue...
```

Viewing budget and expense feature usage:

```
Main Menu:
1. Transaction
2. Category
3. Budget
4. Track Budget, Expense and Income
5. Exit
Note: You need to add monthly budget before doing any transaciton for that month

Select an option (1-5): 4
Enter the month and year to track overall budget and expenses (format MM/yyyy):
03/2024
__________ Budget and Expenses for 03/2024 __________
Category: Food, Budget: 0, Expenses: 0
Category: Transportation, Budget: 45, Expenses: 7.65
Category: Bills, Budget: 0, Expenses: 0
Total Budget: 1500, Total Expenses: 7.65
Remaining Budget: 1492.35
__________ Income for 03/2024 __________
Category: Salaries, Income: 0
Category: Bonus, Income: 350
Total Income: 350

Press any key to continue...
```

Test number 4:

Creating category with a label that is already being used:

```
Category Menu:
1. Create
2. Rename Category Label
3. Update Category Budget
4. Delete
5. View
6. Return to Main Menu

Select an option (1-5): 1
Enter category label:
Food
Is this category for an expense? (yes/no):
no
Enter month and year for the category (format MM/yyyy):
03/2024
Failed to create category.

Press any key to continue...
```

Test number 5:

Insertion of string value instead of numerical value:

```
Transaction Menu:
1. Create
2. Update
3. Move Transaciton to another Category
4. Delete
5. Display All Ongoing Transactions
6. Display Category-wise Transactions
7. Display All Past Transactions
8. Display Past Category-wise Transactions
9. Close And Reopen Category For The New Month
10.Manage Recurring Transactions
11. Return to Main Menu

Select an option (1-11): 1
Enter the label of the category for the transaction:
Food
Enter the amount of the transaction:
fifty pounds
Invalid amount. Please enter a valid number.

Press any key to continue...
```

MEETING MINUTES

MEETING 1:

Date: 4th of March 2024

Location: Westminster Library (Physical Meeting)

Time: 18:00 - 18:30

Attendees:

- Romain Charles Kuhne: w1972584
- Al Masum: w2012132
- Mobeen Raja: w1803881

Absentees:

- N/A

Agenda:

- UML design discussion and viewing.
- Time management.
- Testing.
- Code development.

Topic of Discussion:

- UML design alterations were made after Charles had realised changes had to be made before development could begin and so after discussing changes as a group, there was a verbal agreement on the changes and Charles was tasked with updating the UML design.
- Discussion of the frequency of meetings has we realised our group was made very late and so we decided on a date for regular meetings.
- Talks of the ideal amounts of tests that would possibly be carried out, the number that was floating around was around 15 by Mobeen but as he mentioned there is no sure way to know as the application development process is dynamic and so the testing and fixing or errors was a recurring process and so we all agreed on focusing on creating a test layout first.
- Code development was the focus of this meeting as we wanted to finalise the design so the development could begin and so we gave a soft deadline for the code development for the following weekend.

Actions Taken:

- Test layout creation (Mobeen Raja).
- UML design changes (Romain Charles Kuhne).
- Code development (Al Masum).

Project Manager's Notes:

- Soft deadline for code development is set for 10/03/2024.
- Respectful session with all involved being professional.

- Time set out for regular meetings has been set to Mondays 18:00 onwards.

Next week's agenda:

- Progress check of code.
- Discussion of testing.
- Analysis of time management.

MEETING 2:

Date: 11th of March 2024

Location: Westminster Library (Physical Meeting)

Time: 18:00 - 18:30

Attendees:

- Romain Charles Kuhne: w1972584
- Mobeen Raja: w1803881

Absentees:

- Al Masum: w2012132

Apologies Received:

- Al Masum: w2012132

Agenda:

- Code development.
- UML changes if needed.
- Testing.
- Time management.

Topic of Discussion:

- UML final design alterations were discussed which involved the Removal of wallet repository from the design.
- Further discussion into the design weighing changes against efficacy for development.
- Addition of two new interfaces called “IBudgetAvailable” and “ITransactionDate” as a safeguarding concept.
- Design naming convention changes for easier readability in terms of for the development to occur as there were minor confusions but all were fixed during the meeting.
- Discussion of the testing that should occur and the quality of tests needed, with clarity meaning screenshots of the tests that should occur to help with failed tests should they occur.

- Last discussion was of the time we needed to use as deadline is closing in and we need to make sure everyone does their part fast and efficiently.

Actions Taken:

- Testing of the project (Mobeen Raja).
- UML finalised design changes (Romain Charles Kuhne).
- Code development progression (Al Masum).

Progress made from last week's Agenda:

- Code development being worked on still (Al Masum).
- Testing structure has been made (Mobeen Raja).
- UML design finalised changes made (Romain Charles Kuhne).

Project Manager's Notes:

- The discussion was limited due to an absentee (apology was received), so the topics discussed leaned towards the design and testing as they were the roles of the attendees.
- The project is close to finishing, the development is very close to being ready for testing.
- Test structure has been made and is waiting for code development to finish to begin testing.

Next week's agenda:

- Testing analysis.
- Final code implementation analysis.

- Discussion of report.
- Discussion of presentation.

MEETING 3:

Date: 18th of March 2024

Location: Westminster Library (Physical Meeting)

Time: 18:00 - 18:45

Attendees:

- Romain Charles Kuhne: w1972584
- Mobeen Raja: w1803881
- Al Masum: w2012132

Absentees:

- N/A

Agenda:

- Last second changes for UML design.
- Code implementation discussion.
- Testing discussion.
- Report.
- Presentation.

Topic of Discussion:

- Discussion of the tight coupling present in the design.
- Discussion of the meeting minutes and the structure of it, asking for feedback and completion of it entirely.
- Consensus reached regarding presentation, its formatting and the workload assigned to all members.
- Report discussion, Charles having created a google document with an invite link for the other members to join and have editing roles, so tracking of live progress can occur.
- Masum talking of finishing last second changes to the code to possibly adding the concept of recurrence.
- Income was brought up by Mobeen and discussed with the other members, as the transaction was implemented but we noticed that income was not implemented and so we asked Charles to make a last second change to the design so we could implement an income feature, time is limited so we had to make the change fast and as Masum to work on it as soon as he could.

- Further code implementations discussions regarding budget comparison, as the ability to check a category's spending was implemented but no feature to track the month's budget, so this feature was discussed and the possibility of displaying this.
-

Actions Taken:

- Presentation slides to work on (Raja, Charles and Masum).
- Reporting document to be made (Raja, Charles and Masum).
- UML finalised design changes (Romain Charles Kuhne).
- Code development progression (Al Masum).
- Wrapping up Testing (Mobeen Raja).
- Online meeting set for 20th of March 7:30pm (By Mobeen Raja).

Progress made from last week's Agenda:

- Code development has been almost fully completed, with minor changes needing to be made (Al Masum).
- Testing has been completed, possible small testing to occur for the small code development left. (Mobeen Raja).
- UML design completed with minor change needed to be made for income and recurrence possible additions (Romain Charles Kuhne).

Project Manager's Notes:

- The discussions were the most productive to date and fast solutions was promptly presented by each member.
- Contribution to each role has been invaluable by all members up to this date and respect has been shown in abundance which has allowed free productive talk.
- A last-minute online meeting was arranged just in case something needs to be discussed or something has occurred, a form of finalised confirmation before submission by all members.
- Time constraints has weighed on the additions of numerous features such as a core feature that needs to be implemented; income and further features discussed such as displaying a month's budget expenditures.

Next Meeting's agenda:

- Report confirmation.
- Discussions of presentation.

CODE LISTING

Budget Tracking application

ITransactionDate Interface

```
internal interface ITransactionDate
{
   DateTime GetTransactionDate();
}
```

IBudgetAvailable Interface

```
public interface IBudgetAvailable
{
   bool IsBudgetAvailable(double amount);
}
```

Helper Class

```
static class Helper
{
   public static string GenerateTransactionNo()
   {
      Guid guid = Guid.NewGuid();
      string guidString = guid.ToString("N");

      string transactionNo = guidString.Substring(0, 8);

      transactionNo = "TR" + transactionNo;
```

```
        return transactionNo;
    }
}
```

Budget Class

```
public class Budget : IBudgetAvailable
{
    private double sumToAllocate;
    public  double remainingBudget;
    private DateTime monthAndYear;

    public Budget(double sumToAllocate, DateTime monthAndYear)
    {
        this.sumToAllocate = sumToAllocate;
        this.remainingBudget = sumToAllocate;
        this.monthAndYear = monthAndYear;
    }

    public void SetBudget(double amount)
    {
        sumToAllocate = amount;
    }

    public void SetremainingBudget(double amount)
    {
        remainingBudget = amount;
    }

    public void SetDate(DateTime date)
    {
        monthAndYear = date;
    }

    public double GetBudget()
    {
        return sumToAllocate;
    }
```

```
    public double GetremainingBudget()
    {
        return remainingBudget;
    }

    public DateTime GetDate()
    {
        return monthAndYear;
    }

    public bool IsBudgetAvailable(double amount)
    {
        throw new NotImplementedException();
    }


}
```

Category Class

```
public class Category: ITransactionDate
{
    private string categoryLabel;
    private double budgetAllocated;
    private DateTime monthAndYear;
    private bool isExpense;
    public List<Transaction> transactionList;

    public Category(string label, double budget, DateTime date, bool isExpense)
    {
        categoryLabel = label;
        budgetAllocated = budget;
        monthAndYear = date;
        transactionList = new List<Transaction>();
        this.isExpense = isExpense;
    }
```

```
public Category()
{
}

public void SetCategoryLabel(string label)
{
   categoryLabel = label;
}


public void SetCategoryBudget(double budget)
{
   budgetAllocated = budget;
}

public void SetCategoryDate(DateTime date)
{
   monthAndYear = date;
}

public string GetCategoryLabel()
{
   return categoryLabel;
}

public double GetCategoryBudget()
{
   return budgetAllocated;
}

public DateTime GetCategoryDate()
{
   return monthAndYear;
}

public DateTime GetTransactionDate()
{
   throw new NotImplementedException();
```

```
    }

    public void SetIsExpense(bool isExp)
    {
        isExpense = isExp;

    }

    public bool GetIsExpense()
    {
        return isExpense;
    }


}
```

Transaction Class

```
public class Transaction
{
    private double transactionAmount;
    private DateTime transactionDate;
    private string transactionNbr;
    private string transactionDescription;
    private bool isMonthlyRecurring;

    public Transaction(string trnNo, double amount, DateTime date, string note,bool isRecuring=false)
    {
        transactionAmount = amount;
        transactionDate = date;
        transactionNbr = trnNo;
        transactionDescription = note;
        isMonthlyRecurring = isRecuring;
    }
    public void SetTransactionAmount(double amount)
    {
        transactionAmount = amount;
```

```
}

public void SetTransactionDate(DateTime date)
{
    transactionDate = date;
}

public void SetTransactionNbr(string idNbr)
{
    transactionNbr = idNbr;
}

public void SetTransactionDescription(string description)
{
    transactionDescription = description;
}

public double GetTransactionAmount()
{
    return transactionAmount;
}

public DateTime GetTransactionDate()
{
    return transactionDate;
}

public string GetTransactionNbr()
{
    return transactionNbr;
}

public string GetTransactionDescription()
{
    return transactionDescription;
}

public bool GetIsMonthlyRecurring()
```

```
    {
        return isMonthlyRecurring;
    }

    public void SetIsMonthlyRecurring(bool isRec)
    {
        isMonthlyRecurring = isRec;
    }
}
```

Wallet Class

```
public class Wallet

{

    private List<Category> categoryList;

    private List<Budget> budgetHistory;


    public Wallet()

    {

        categoryList = new List<Category>();

        budgetHistory = new List<Budget>();

    }


    public bool CreateCategory(Category category)

    {
```

```
    if (category != null)
    {
        Category preCat = categoryList.Where(e => e.GetCategoryLabel().ToLower() == category.GetCategoryLabel().ToLower() &&
e.GetCategoryDate().Month==category.GetCategoryDate().Month && e.GetCategoryDate().Year ==
category.GetCategoryDate().Year).FirstOrDefault();
        if (preCat == null)
        {
            categoryList.Add(category);
            return true;
        }

    }
    return false;
  }

  public void DisplayCategories(DateTime monthYear)
  {
    var filteredCategories = categoryList.Where(c => c.GetCategoryDate().Month == monthYear.Month && c.GetCategoryDate().Year ==
monthYear.Year);
```

```
    if (!filteredCategories.Any())
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("No categories found for the specified month and year.");
        Console.ResetColor();
        return;
    }

    Console.WriteLine($"Categories for {monthYear.ToString("MM/yyyy")}:");
    foreach (var category in filteredCategories)
    {
        string type = "";
        if (category.GetIsExpense() == true)
        {
            type = "Expense";
        }
        else
        {
            type = "Income";
        }
```

```
            Console.WriteLine($"- {category.GetCategoryLabel()} (Budget: {category.GetCategoryBudget()} and Type: {type})");
        }
    }

    public bool RenameCategory(string oldLabel, string newLabel, DateTime monthYear)
    {
        Category categoryToRename = categoryList.FirstOrDefault(c => c.GetCategoryLabel().ToLower() == oldLabel.ToLower() && c.GetCategoryDate()
== monthYear);
        if (categoryToRename != null)
        {
            categoryToRename.SetCategoryLabel(newLabel);
            return true;
        }
        return false;
    }

    public bool UpdateCategoryBudget(string label, DateTime monthYear, double newBudget)
    {
        var category = categoryList.FirstOrDefault(c => c.GetCategoryLabel().ToLower() == label.ToLower() && c.GetCategoryDate() == monthYear);
        if (category != null)
```

```
    {
        category.SetCategoryBudget(newBudget);
        return true;
    }
    return false;
  }


  public bool RemoveCategory(string label, DateTime monthYear)
  {
    Category remCate = categoryList.Where(s => s.GetCategoryLabel().ToLower() == label.ToLower() && s.GetCategoryDate() ==
monthYear).FirstOrDefault();
    if (remCate != null)
    {
        categoryList.Remove(remCate);
        return true;
    }
    else
    {
        return false;
```

```
    }

}

//new added
//public void GetAllCategory()
//{
//   List<string> catList=categoryList.Select(s=>s.GetCategoryLabel()).Distinct().ToList();
//   foreach (var item in catList)
//   {
//       Console.WriteLine("Categories are :");
//       Console.WriteLine($"{item}");
//   }
//}



public bool AddBudget(Budget budget)
{
```

```
    if(budget != null)
    {
     budgetHistory.Add(budget);
    }
    return true;
  }

  public bool RemoveBudget(Budget budget)
  {
    if (budget != null)
    {
      budgetHistory.Remove(budget);
      return true;
    }
    else
    {
      return false;
    }
  }
```

```
  public bool AllocateBudgetToCategory(string categoryLabel, double amount, DateTime monthYear)
  {
    Category modifiedCategory = categoryList.FirstOrDefault(c => c.GetCategoryLabel().ToLower() == categoryLabel.ToLower() &&
c.GetCategoryDate() == monthYear);
    if (modifiedCategory != null)
    {
      modifiedCategory.SetCategoryBudget(amount);
      return true;
    }
    return false;
  }

  public Budget GetBudgetByDate(DateTime date)
  {
    return budgetHistory.FirstOrDefault(b => b.GetDate() == date);
  }

  //changed parameters.....
```

```
public bool AddTransaction(Transaction tran, string catLabel)
{
    DateTime tranDate = tran.GetTransactionDate().Date;

    Category cat = categoryList.FirstOrDefault(w => w.GetCategoryLabel().ToLower() == catLabel.ToLower()
            && w.GetCategoryDate().Month == tranDate.Month
            && w.GetCategoryDate().Year == tranDate.Year);

    if (cat != null)
    {
        Budget budget = budgetHistory.FirstOrDefault(b => b.GetDate().Month == tranDate.Month && b.GetDate().Year == tranDate.Year);

        if (cat.GetIsExpense() == false)
        {
            cat.transactionList.Add(tran);
            return true;
        }
        else
        {
            if (budget != null && cat.GetIsExpense() == true)
```

```
{
    //Check if adding the transaction would exceed the remaining budget
    double newAmountAfterTransaction = budget.GetremainingBudget() - tran.GetTransactionAmount();

    if (newAmountAfterTransaction >= 0)
    {
        cat.transactionList.Add(tran);

        budget.SetremainingBudget(newAmountAfterTransaction);
        TrackBudgetByCategory(cat.GetCategoryLabel(), tran.GetTransactionDate());

        return true;
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Adding this transaction would exceed the remaining budget for this month.");
        Console.ResetColor();
        return false;
    }
```

```
        }
        else
        {
            //No budget found for the current month
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("No budget found for the current month. Transaction cannot be added.");
            Console.ResetColor();
            return false;
        }
    }

}
else
{
    //Category not found
    Console.ForegroundColor = ConsoleColor.DarkRed;
    Console.WriteLine("Category not found. Transaction cannot be added.");
    Console.ResetColor();
    return false;
}
```

```
  }



  public bool ModifyTransactionAmount(string transactionId, double newAmount)
  {
    Transaction transaction = categoryList.SelectMany(cat => cat.transactionList)
      .FirstOrDefault(tran => tran.GetTransactionNbr().ToLower() == transactionId.ToLower());

    if (transaction != null)
    {
      // Calculating the difference between the new and current transaction amount
      double amountDifference = newAmount - transaction.GetTransactionAmount();

      DateTime transactionDate = transaction.GetTransactionDate();
      Budget currentBudget = budgetHistory.FirstOrDefault(b => b.GetDate().Month == transactionDate.Month && b.GetDate().Year ==
transactionDate.Year);

      if (currentBudget != null)
      {
        // Checking if applying the difference would exceed the remaining budget.....
```

```
        double newRemainingBudget = currentBudget.GetremainingBudget() - amountDifference;

        if (newRemainingBudget >= 0)
        {
            transaction.SetTransactionAmount(newAmount);

            currentBudget.SetremainingBudget(newRemainingBudget);
            Category cat = categoryList.FirstOrDefault(s => s.transactionList.Any(t => t.GetTransactionNbr() == transactionId));
            if (cat != null)
            {
                TrackBudgetByCategory(cat.GetCategoryLabel(), transactionDate);
            }

            return true;
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine("Modifying this transaction would exceed the remaining budget for this month.");
            Console.ResetColor();
```

```
                return false;
            }
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine("No budget found for the month of the transaction. Cannot modify the transaction amount.");
            Console.ResetColor();
            return false;
        }
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Transaction not found.");
        Console.ResetColor();
        return false;
    }
}
```

```
  public bool MoveTransaction(string transactionId, string newCategoryLabel)
  {
    var originalCategory = categoryList.FirstOrDefault(cat => cat.transactionList.Any(tran => tran.GetTransactionNbr().ToLower() ==
transactionId.ToLower()));
    if (originalCategory == null)
    {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Original category not found.");
      Console.ResetColor();
      return false;
    }

    var transaction = originalCategory.transactionList.FirstOrDefault(tran => tran.GetTransactionNbr().ToLower() == transactionId.ToLower());
    if (transaction == null)
    {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Transaction not found.");
      Console.ResetColor();
      return false;
    }
```

```
var newCategory = categoryList.FirstOrDefault(cat => cat.GetCategoryLabel().ToLower() == newCategoryLabel.ToLower());
if (newCategory == null)
{  Console.ForegroundColor= ConsoleColor.DarkRed;
   Console.WriteLine("New category not found.");
   Console.ResetColor();
   return false;
}

if (originalCategory.GetIsExpense() == newCategory.GetIsExpense())
{
   originalCategory.transactionList.Remove(transaction);
   newCategory.transactionList.Add(transaction);
   return true;
}
else
{
   Console.ForegroundColor = ConsoleColor.DarkRed;
   Console.WriteLine("Category Types are not same. Cannot move transaction between Income and Expencse type");
   Console.ResetColor();
```

```
    }

    return false;
  }

  // removed parameter string label
  public bool RemoveTransaction(string transactionId)
  {
    Transaction transaction = categoryList.Select(w => w.transactionList.Where(s => s.GetTransactionNbr() ==
transactionId).FirstOrDefault()).FirstOrDefault();
    foreach (var category in categoryList)
    {
     category.transactionList.Remove(transaction);
    }

    return true;
  }

  public void DisplayAllOngoingTransactions(DateTime currentDate)
  {
```

```
    List<Category> currentCategoryList = categoryList.Where(s => s.GetCategoryDate().Month == currentDate.Month &&
s.GetCategoryDate().Year==currentDate.Year).ToList();
    foreach (var category in currentCategoryList)
    {

      Console.WriteLine($"Category Name: {category.GetCategoryLabel()}");
      foreach (var transaction in category.transactionList)
      {
        string recuring;
        if (transaction.GetIsMonthlyRecurring() == true)
        {
          recuring = "Yes";
        }
        else
        {
          recuring = "No";
        }

        Console.WriteLine($"---Transaction info: {transaction.GetTransactionNbr()},Date:{transaction.GetTransactionDate().ToString("dd/MM/yyyy")},
Amount:{transaction.GetTransactionAmount()},Note:{transaction.GetTransactionDescription()}, Monthly Recurring: {recuring}");
      }
```

```
    }
  }

  public void DisplayCategoryTransactions(string label, DateTime date)
  {
    Category category = categoryList.Where(s => s.GetCategoryLabel().ToLower() == label.ToLower() & s.GetCategoryDate().Month == date.Month &&
s.GetCategoryDate().Year==date.Year).FirstOrDefault();


    Console.WriteLine($"Category Name: {category.GetCategoryLabel()}, Month: {date.Month}-{date.Year}");
    foreach (var transaction in category.transactionList)
    {
      string recuring;
      if (transaction.GetIsMonthlyRecurring() == true)
      {
        recuring = "Yes";
      }
      else
      {
        recuring = "No";
```

```
        }

        Console.WriteLine($"---Transaction info: {transaction.GetTransactionNbr()},Date:{transaction.GetTransactionDate().ToString("dd/MM/yyyy")},
Amount:{transaction.GetTransactionAmount()},Note:{transaction.GetTransactionDescription()}, Monthly Recurring: {recuring}");
      }
    }

    public void DisplayAllPastTransactions()
    {
      foreach (var category in categoryList)
      {
        Console.WriteLine($"Category Name: {category.GetCategoryLabel()}");
        Console.WriteLine($"  Month: {category.GetCategoryDate().Month}/{category.GetCategoryDate().Year}");
        foreach (var transaction in category.transactionList)
        {
          string recuring;
          if (transaction.GetIsMonthlyRecurring() == true)
          {
            recuring = "Yes";
          }
```

```
            else
            {
               recuring = "No";
            }

            Console.WriteLine($"    Transaction info: {transaction.GetTransactionNbr()},Date:{transaction.GetTransactionDate().ToString("dd/MM/yyyy")},
Amount:{transaction.GetTransactionAmount()},Note:{transaction.GetTransactionDescription()}, Monthly Recurring: {recuring}");
        }
      }
    }

    public void DisplayPastCategoryTransaction(string label)
    {
      List<Category> currentCategoryList = categoryList.Where(s => s.GetCategoryLabel().ToLower() == label.ToLower()).ToList();

      foreach (var category in currentCategoryList)
      {
        Console.WriteLine($"Category Name: {category.GetCategoryLabel()}");
        foreach (var transaction in category.transactionList)
        {
```

```
            string recuring;
            if (transaction.GetIsMonthlyRecurring() == true)
            {
               recuring = "Yes";
            }
            else
            {
               recuring = "No";
            }

            Console.WriteLine($"---Transaction info: {transaction.GetTransactionNbr()},Date:{transaction.GetTransactionDate().ToString("dd/MM/yyyy")},
Amount:{transaction.GetTransactionAmount()},Note:{transaction.GetTransactionDescription()}, Monthly Recurring: {recuring}");
        }
      }
   }

   //deleted previous month parameter and added category label
   public bool CloseAndOpenCategories(string catLabel,DateTime newMonth)
   {
```

```
    Category newCategory = new Category();
    newCategory = categoryList.Where(s => s.GetCategoryLabel() == catLabel).OrderByDescending(s=>s.GetCategoryDate()).FirstOrDefault();
    if (newCategory != null)
    {
      newCategory.SetCategoryDate(newMonth);
      if (!categoryList.Contains(newCategory))
      {
        categoryList.Add(newCategory);
      }

      return true;
    }
    return false;
  }

  //name cahnged from TrackBudget
  public void TrackBudgetByCategory(string catLabel, DateTime monthYear)
  {
    Category category = categoryList.Where(e => e.GetCategoryLabel().ToLower() == catLabel.ToLower() & e.GetCategoryDate().Month ==
monthYear.Month & e.GetCategoryDate().Year == monthYear.Year).FirstOrDefault();
```

```
    if (category == null)
    {
        Console.WriteLine($"No category found for {catLabel} in {monthYear.ToString("MM/yyyy")}.");
        return;
    }

    double totalSpent = category.transactionList.Where(tr => tr.GetTransactionAmount() > 0)
                                .Sum(tr => tr.GetTransactionAmount());

    // Compare the total spent against the budget
    if (category.GetCategoryBudget()>0 & totalSpent > category.GetCategoryBudget() )
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine($"<Warning>: You have exceeded your budget for {category.GetCategoryLabel()} in {monthYear.ToString("MM/yyyy")}.
Budget: {category.GetCategoryBudget()}, Spent: {totalSpent}");
        Console.ResetColor();
    }
    else
    {
```

```
        Console.ForegroundColor = ConsoleColor.Blue;
        Console.WriteLine($"For {category.GetCategoryLabel()} in {monthYear.ToString("MM/yyyy")}, you have spent {totalSpent} out of your budget of {category.GetCategoryBudget()}.");
        Console.ResetColor();
    }
}

public void TrackOverallBudget(DateTime monthYear)
{
    double totalBudget = budgetHistory.Where(s=>s.GetDate().Month==monthYear.Month && s.GetDate().Year==monthYear.Year).Select(b=>b.GetBudget()).FirstOrDefault();
    double totalExpenses = 0;
    double totalIncome = 0;

    var filteredCategories = categoryList
                .Where(c => c.GetCategoryDate().Year == monthYear.Year &&
                    c.GetCategoryDate().Month == monthYear.Month)
                .ToList();
    Console.WriteLine($"__________ Budget and Expenses for {monthYear:MM/yyyy} __________");
```

```
foreach (var category in filteredCategories.Where(s=>s.GetIsExpense()==true))
{
   double categoryBudget = category.GetCategoryBudget();
   double categoryExpenses = category.transactionList
                    .Sum(t => t.GetTransactionAmount());
   totalExpenses += categoryExpenses;
   Console.ForegroundColor = ConsoleColor.Red;
   Console.WriteLine($"Category: {category.GetCategoryLabel()}, Budget: {categoryBudget}, Expenses: {categoryExpenses}");
   Console.ResetColor();
}

Console.WriteLine($"Total Budget: {totalBudget}, Total Expenses: {totalExpenses}");

double remainingBudget = totalBudget - totalExpenses;
Console.WriteLine($"Remaining Budget: {remainingBudget}");

Console.WriteLine($"__________ Income for {monthYear:MM/yyyy} __________");
foreach (var category in filteredCategories.Where(s => s.GetIsExpense() == false))
{
   double categoryIncome = category.transactionList
```

```
                    .Sum(t => t.GetTransactionAmount());
        totalIncome += categoryIncome;
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine($"Category: {category.GetCategoryLabel()}, Income: {totalIncome}");
    }
    Console.WriteLine($"Total Income: {totalIncome}");
    Console.ResetColor();

}

public bool ApplyRepeatingTransactions()
{
    DateTime fromMonthYear = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1).AddMonths(-1);
    DateTime toMonthYear = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);

    Budget budget = budgetHistory.FirstOrDefault(b => b.GetDate().Month == toMonthYear.Month && b.GetDate().Year == toMonthYear.Year);

    if (budget == null)
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
```

```
        Console.WriteLine("You need to set Monthly budget first for any transaction");
        Console.ResetColor();
        return false;
    }

    foreach (var category in categoryList.Where(c => c.GetCategoryDate().Month == fromMonthYear.Month && c.GetCategoryDate().Year == fromMonthYear.Year))
    {
        var recurringTransactions = category.transactionList.Where(t => t.GetIsMonthlyRecurring()).ToList();

        // Duplicate each recurring transaction for the new month.
        foreach (var transaction in recurringTransactions)
        {
            Transaction newTransaction = new Transaction(
                Helper.GenerateTransactionNo(),
                transaction.GetTransactionAmount(),
                new DateTime(toMonthYear.Year, toMonthYear.Month, Math.Min(DateTime.DaysInMonth(toMonthYear.Year, toMonthYear.Month), transaction.GetTransactionDate().Day)),
                transaction.GetTransactionDescription(),
                true
            );
```

```
        var newMonthCategory = categoryList.FirstOrDefault(c =>
            c.GetCategoryLabel().Equals(category.GetCategoryLabel(), StringComparison.OrdinalIgnoreCase) &&
            c.GetCategoryDate().Month == toMonthYear.Month &&
            c.GetCategoryDate().Year == toMonthYear.Year);

        if (newMonthCategory == null)
        {
            newMonthCategory = new Category(
                category.GetCategoryLabel(),
                0,
                new DateTime(toMonthYear.Year, toMonthYear.Month, 1),
                category.GetIsExpense()
            );
            categoryList.Add(newMonthCategory);
        }

        newMonthCategory.transactionList.Add(newTransaction);
    }
}
```

```
        return true;
    }
}
```

Program Class

```
class Program
{
    private static Wallet wallet = new Wallet();
    static void Main(string[] args)
    {
        InitializePresetCategories();
        InitializeDemoData();

        while (true)
        {
            Console.Clear();
            Console.WriteLine("Main Menu:");
            Console.WriteLine("1. Transaction");
            Console.WriteLine("2. Category");
```

```
        Console.WriteLine("3. Budget");
        Console.WriteLine("4. Track Budget, Expense and Income");
        Console.WriteLine("5. Exit");
        Console.WriteLine("Note: You need to add monthly budget before doing any transaciton for that month");
        Console.Write("\nSelect an option (1-5): ");

        if (!int.TryParse(Console.ReadLine(), out int mainChoice) || mainChoice < 1 || mainChoice > 5)
        {
            Console.WriteLine("Invalid choice, please try again.");
            ContinuePrompt();
            continue;
        }

        if (mainChoice == 5) break; // Exit the program

        switch (mainChoice)
        {
            case 1:
                TransactionSubMenu();
                break;
```

```
            case 2:
                CategorySubMenu();
                break;
            case 3:
                BudgetSubMenu();
                break;
            case 4:
                TrackOverallBudgetFromInput();
                break;
        }
    }
}

static void TransactionSubMenu()
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine($"Transaction Menu:");
        Console.WriteLine("1. Create ");
```

```
        Console.WriteLine("2. Update");
        Console.WriteLine("3. Move Transaciton to another Category");
        Console.WriteLine("4. Delete");
        Console.WriteLine("5. Display All Ongoing Transactions");
        Console.WriteLine("6. Display Category-wise Transactions");
        Console.WriteLine("7. Display All Past Transactions");
        Console.WriteLine("8. Display Past Category-wise Transactions");
        Console.WriteLine("9. Close And Reopen Category For The New Month");
        Console.WriteLine("10.Manage Recurring Transactions");

        Console.WriteLine("11. Return to Main Menu");
        Console.Write("\nSelect an option (1-11): ");

        if (!int.TryParse(Console.ReadLine(), out int subChoice) || subChoice < 1 || subChoice > 11)
        {
            Console.WriteLine("Invalid choice, please try again.");
            ContinuePrompt();
            continue;
        }
```

```
if (subChoice >= 11) break; // Return to Main Menu
else if (subChoice == 1)
{
   //Console.WriteLine("Transaction create function called");
   AddTransactionFromInput();
}
else if (subChoice == 2)
{
  //Console.WriteLine("Transaction Update function called");
   ModifyTransactionAmountFromInput();
}
else if (subChoice == 3)
{
   //Console.WriteLine("Transaction move function called");
   MoveTransactionFromInput();
}
else if (subChoice == 4)
{
   //Console.WriteLine("Transaction Delete function called");
   RemoveTransactionFromInput();
```

```
        }
        else if (subChoice == 5)
        {
            //Console.WriteLine("Transaction Delete function called");
            DisplayAllOngoingTransactionsFromInput();
        }
        else if (subChoice == 6)
        {
            //Console.WriteLine("Transaction Delete function called");
            DisplayCategoryTransactionsFromInput();
        }
        else if (subChoice == 7)
        {
            //Console.WriteLine("Transaction Delete function called");
            DisplayAllPastTransactionsFromInput();
        }
        else if (subChoice == 8)
        {
            //Console.WriteLine("Transaction Delete function called");
            DisplayPastCategoryTransactionsFromInput();
```

```
        }
        else if (subChoice == 9)
        {
            //Console.WriteLine("Transaction Delete function called");
            CloseAndOpenCategoriesFromInput();
        }
        else if (subChoice == 10)
        {
            //Console.WriteLine("Transaction Delete function called");
            ManageRecurringTransactions();
        }
        ContinuePrompt();
    }
}
static void CategorySubMenu()
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine($"Category Menu:");
```

```
        Console.WriteLine("1. Create");
        Console.WriteLine("2. Rename Category Label");
        Console.WriteLine("3. Update Category Budget");
        Console.WriteLine("4. Delete");
        Console.WriteLine("5. View");
        Console.WriteLine("6. Return to Main Menu");
        Console.Write("\nSelect an option (1-5): ");

        if (!int.TryParse(Console.ReadLine(), out int subChoice) || subChoice < 1 || subChoice > 6)
        {
            Console.WriteLine("Invalid choice, please try again.");
            ContinuePrompt();
            continue;
        }

        if (subChoice >= 6) break;
        else if (subChoice == 1)
        {
            //Console.WriteLine("Category create function called");
            CreateCategoryFromInput();
```

```
        }
        else if (subChoice == 2)
        {
            //Console.WriteLine("Category Rename function called");
            RenameCategoryFromInput();
        }
        else if (subChoice == 3)
        {
            //Console.WriteLine("Category Delete function called");
            UpdateCategoryBudgetFromInput();
        }
        else if (subChoice == 4)
        {
            //Console.WriteLine("Category Delete function called");
            RemoveCategoryFromInput();
        }
        else if (subChoice == 5)
        {
            //Console.WriteLine("Category View function called");
            DisplayCategoriesFromInput();
```

```
        }
        ContinuePrompt();
    }
}
static void BudgetSubMenu()
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine("Budget Menu:");
        Console.WriteLine("1. Create");
        Console.WriteLine("2. Delete");
        Console.WriteLine("3. Allocate Budget To Category");
        Console.WriteLine("4. Check Budget");
        Console.WriteLine("5. Return to Main Menu");
        Console.Write("\nSelect an option (1-5): ");

        if (!int.TryParse(Console.ReadLine(), out int subChoice) || subChoice < 1 || subChoice > 5)
        {
            Console.WriteLine("Invalid choice, please try again.");
```

```
        ContinuePrompt();
        continue;
    }

    if (subChoice >= 5) break; // Return to Main Menu
    else if (subChoice == 1)
    {
        //Console.WriteLine("Budget create function called");
        CreateBudgetFromInput();
    }
    else if (subChoice == 2)
    {
        //Console.WriteLine("Budget Delete function called");
        RemoveBudgetFromInput();

    }
    else if (subChoice == 3)
    {
        //Console.WriteLine("Allocate Budget To Category function called");
        AllocateBudgetToCategoryFromInput();
```

```
        }
        else if (subChoice == 4)
        {
            //Console.WriteLine("Budget View function called");
            ViewBudgetByDate();
        }

        ContinuePrompt();
    }
}

static void ContinuePrompt()
{
    Console.WriteLine("\nPress any key to continue...");
    Console.ReadKey();
}


static void InitializePresetCategories()
{
```

```
    var currentDate = DateTime.Now;
    var monthAndYear = new DateTime(currentDate.Year, currentDate.Month, 1);

    // Preset categories for current month....
    var presetCategories = new List<Category>
{
    new Category("Food", 0, monthAndYear,true),
    new Category("Transportation", 0, monthAndYear,true),
    new Category("Salaries", 0, monthAndYear,false),
    new Category("Bills", 0, monthAndYear,true)
};

    foreach (var category in presetCategories)
    {
        wallet.CreateCategory(category);
    }
}


static void InitializeDemoData()
```

```
{
    // Setup demo Budget
    Budget januaryBudget = new Budget(2000, new DateTime(2024, 1, 1));
    wallet.AddBudget(januaryBudget);
    Budget februaryBudget = new Budget(2500, new DateTime(2024, 2, 1));
    wallet.AddBudget(februaryBudget);

    // Setup demo Categories
    Category foodCategoryJan = new Category("Food", 500, new DateTime(2024, 1, 1), true);
    Category salaryCategoryJan = new Category("Salary", 0, new DateTime(2024, 1, 1), false);
    Category foodCategoryFeb = new Category("Food", 600, new DateTime(2024,2,1), true);
    Category salaryCategoryFeb = new Category("Salary", 0, new DateTime(2024, 2, 1), false);
    Category BillsCategoryFeb = new Category("Bills", 0, new DateTime(2024, 2, 13), true);
    wallet.CreateCategory(foodCategoryJan);
    wallet.CreateCategory(salaryCategoryJan);
    wallet.CreateCategory(foodCategoryFeb);
    wallet.CreateCategory(salaryCategoryFeb);
    wallet.CreateCategory(BillsCategoryFeb);

    // Setup demo Transactions
```

```
    Transaction foodTransaction = new Transaction("TR1234", 250, new DateTime(2024, 1, 5), "Grocery shopping", false);
    Transaction salaryTransaction = new Transaction("TR5678", 3000, new DateTime(2024, 1, 1), "Monthly Salary", false);
    Transaction foodTransactionFebruary = new Transaction("TR2345", 300, new DateTime(2024, 2, 7), "Grocery shopping", false);
    Transaction salaryTransactionFebruary = new Transaction("TR6789", 3200, new DateTime(2024, 2, 1), "Monthly Salary", false);
    //Transaction BillsTransactionFeb = new Transaction("TR8910", 150, new DateTime(2024, 2, 14), "Weekend Dinner", true);
    Transaction BillsTransactionFeb = new Transaction("TR8910", 150, new DateTime(2024, 2, 14), "Weekend Dinner", true);

    wallet.AddTransaction(foodTransaction, foodCategoryJan.GetCategoryLabel());
    wallet.AddTransaction(salaryTransaction, salaryCategoryJan.GetCategoryLabel());
    wallet.AddTransaction(foodTransactionFebruary, foodCategoryFeb.GetCategoryLabel());
    wallet.AddTransaction(salaryTransactionFebruary, salaryCategoryFeb.GetCategoryLabel());
    wallet.AddTransaction(BillsTransactionFeb, BillsCategoryFeb.GetCategoryLabel());

}

#region Category
static void CreateCategoryFromInput()
{
    Console.WriteLine("Enter category label:");
    string label = Console.ReadLine();
```

```
Console.WriteLine("Is this category for an expense? (yes/no):");
string isExpenseInput = Console.ReadLine().Trim().ToLower();
bool isExpense = isExpenseInput == "yes" || isExpenseInput == "y";

double budgetAllocated = 0;
if (isExpense)
{
   Console.WriteLine("Enter budget allocated for this category (optional, press Enter to skip):");
   string budgetInput = Console.ReadLine();
   if (!string.IsNullOrWhiteSpace(budgetInput) && !double.TryParse(budgetInput, out budgetAllocated))
   {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Invalid budget format. Setting budget to default (0).");
      Console.ResetColor();
   }
}

Console.WriteLine("Enter month and year for the category (format MM/yyyy):");
string monthYearInput = Console.ReadLine();
```

```
if (DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime monthYear))
{
    Category newCategory = new Category(label, budgetAllocated, monthYear, isExpense);
    if (wallet.CreateCategory(newCategory))
    {
        Console.WriteLine("Category created successfully.");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to create category.");
        Console.ResetColor();
    }
}
else
{
    Console.ForegroundColor = ConsoleColor.DarkRed;
    Console.WriteLine("Invalid date format. Please try again.");
    Console.ResetColor();
```

```
    }
  }

  static void RenameCategoryFromInput()
  {

    Console.WriteLine("Enter the Month and Year for the category you wish to rename (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();
    DateTime monthYear;

    if (!DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out monthYear))
    {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Invalid date format. Please use MM/yyyy format.");
      Console.ResetColor();
      return;
    }

    Console.WriteLine($"Current categories for {monthYear.ToString("MM/yyyy")}:");
    wallet.DisplayCategories(monthYear);
```

```
    // Asking for the old label
    Console.WriteLine("Enter the label of the category you wish to rename:");
    string oldLabel = Console.ReadLine();

    // Asking for the new label
    Console.WriteLine("Enter the new label for the category:");
    string newLabel = Console.ReadLine();

    if (wallet.RenameCategory(oldLabel, newLabel, monthYear))
    {
        Console.WriteLine("Category renamed successfully.");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to rename category. Make sure it exists and you entered the correct date.");
        Console.ResetColor();
    }
}
```

```
static void UpdateCategoryBudgetFromInput()
{
    Console.WriteLine("Enter the category label:");
    string label = Console.ReadLine();

    Console.WriteLine("Enter the month and year for the category (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();
    if (!DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime monthYear))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format.");
        Console.ResetColor();
        return;
    }

    Console.WriteLine("Enter the new budget allocated for this category:");
    if (!double.TryParse(Console.ReadLine(), out double newBudget))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid budget format. Please enter a valid number.");
```

```
        Console.ResetColor();
        return;
    }

    if (wallet.UpdateCategoryBudget(label, monthYear, newBudget))
    {
        Console.WriteLine($"Budget updated successfully for category '{label}' for the period {monthYearInput}.");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to update budget. Category not found.");
        Console.ResetColor();
    }
}

public static void RemoveCategoryFromInput()
{
    Console.WriteLine("Enter the label of the category you want to remove:");
    string label = Console.ReadLine();
```

```
Console.WriteLine("Enter the month and year of the category to remove (format MM/yyyy):");
string monthYearInput = Console.ReadLine();

if (!DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime monthYear))
{
   Console.ForegroundColor = ConsoleColor.DarkRed;
   Console.WriteLine("Invalid date format. Please use MM/yyyy format.");
   Console.ResetColor();
   return;
}

// Assuming 'wallet' is your static Wallet instance accessible from here
bool isRemoved = wallet.RemoveCategory(label, monthYear);
if (isRemoved)
{
   Console.WriteLine("Category successfully removed.");
}
else
{
```

```
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Category not found or could not be removed.");
        Console.ResetColor();
    }
}
public static void DisplayCategoriesFromInput()
{
    Console.WriteLine("Enter the month and year to display categories (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();

    if (!DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime monthYear))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please use MM/yyyy format.");
        Console.ResetColor();
        return;
    }
    wallet.DisplayCategories(monthYear);
}
```

```
#endregion

#region Budget
static void CreateBudgetFromInput()
{
    Console.WriteLine("Enter the total sum to allocate for the budget:");
    string sumInput = Console.ReadLine();
    if (!double.TryParse(sumInput, out double sumToAllocate))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid sum format. Please enter a valid number.");
        Console.ResetColor();
        return;
    }

    Console.WriteLine("Enter month and year for the budget (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();
    if (DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime monthYear))
    {
        Budget newBudget = new Budget(sumToAllocate, monthYear);
```

```
        if (wallet.AddBudget(newBudget))
        {
            Console.WriteLine("Budget added successfully.");
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine("Failed to add budget.");
            Console.ResetColor();
        }
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please follow the MM/yyyy format.");
        Console.ResetColor();
    }
}
static void RemoveBudgetFromInput()
{
```

```
Console.WriteLine("Enter month and year of the budget to remove (format MM/yyyy):");
string monthYearInput = Console.ReadLine();
if (DateTime.TryParseExact(monthYearInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime monthYear))
{
    Budget budgetToRemove = wallet.GetBudgetByDate(monthYear);
    if (budgetToRemove != null)
    {
        if (wallet.RemoveBudget(budgetToRemove))
        {
            Console.WriteLine("Budget removed successfully.");
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine("Failed to remove budget.");
            Console.ResetColor();
        }
    }
    else
    {
```

```
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine($"No budget found for {monthYear.ToString("MM/yyyy")}");
            Console.ForegroundColor = ConsoleColor.DarkRed;
        }
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please follow the MM/yyyy format.");
        Console.ResetColor();
    }
}
static void AllocateBudgetToCategoryFromInput()
{
    Console.WriteLine("Enter the label of the category to allocate budget to:");
    string categoryLabel = Console.ReadLine();

    Console.WriteLine("Enter the amount to allocate:");
    if (!double.TryParse(Console.ReadLine(), out double amount))
    {
```

```
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid amount format. Please enter a valid number.");
        Console.ResetColor();
        return;
    }

    Console.WriteLine("Enter the month and year for the category (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();
    if (!DateTime.TryParseExact(monthYearInput, "MM/yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime monthYear))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please follow the MM/yyyy format.");
        Console.ResetColor();
        return;
    }

    if (wallet.AllocateBudgetToCategory(categoryLabel, amount, monthYear))
    {
        Console.WriteLine($"Budget of {amount} allocated to category '{categoryLabel}' for {monthYear.ToString("MM/yyyy")}.");
    }
```

```
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to allocate budget. Make sure the category exists and the date is correct.");
        Console.ResetColor();
    }
}
static void ViewBudgetByDate()
{
    Console.WriteLine("Enter the month and year to view the budget (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();
    DateTime monthYear;
    if (!DateTime.TryParseExact(monthYearInput, "MM/yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out monthYear))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please follow the MM/yyyy format.");
        Console.ResetColor();
        return;
    }
```

```
    Budget budget = wallet.GetBudgetByDate(monthYear);
    if (budget != null)
    {
        Console.WriteLine($"Budget for {monthYear.ToString("MM/yyyy")}:");
        Console.WriteLine($"- Amount allocated: {budget.GetBudget()}");
        Console.WriteLine($"- Remaining budget: {budget.GetremainingBudget()}");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("No budget found for the specified month and year.");
        Console.ResetColor();
    }
}
#endregion

#region Transaction
static void AddTransactionFromInput()
{
    Console.WriteLine("Enter the label of the category for the transaction:");
```

```
    string catLabel = Console.ReadLine();


    Console.WriteLine("Enter the amount of the transaction:");
    if (!double.TryParse(Console.ReadLine(), out double amount))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid amount. Please enter a valid number.");
        Console.ResetColor();
        return;
    }


    Console.WriteLine("Enter the date of the transaction (format dd/MM/yyyy):");
    if (!DateTime.TryParseExact(Console.ReadLine(), "dd/MM/yyyy", System.Globalization.CultureInfo.InvariantCulture,
System.Globalization.DateTimeStyles.None, out DateTime date))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please follow the dd/MM/yyyy format.");
        Console.ResetColor();
        return;
    }
```

```
    Console.WriteLine("Enter a description for the transaction (optional, press Enter to skip):");
    string description = Console.ReadLine();

    Console.WriteLine("Is this transaction recurring monthly? (yes/no):");
    string recurringInput = Console.ReadLine().Trim().ToLower();
    bool isRecurring = recurringInput.Equals("yes", StringComparison.OrdinalIgnoreCase) || recurringInput.Equals("y",
StringComparison.OrdinalIgnoreCase);

    // Generate a transaction number
    string transactionNo = Helper.GenerateTransactionNo();

    Transaction newTransaction = new Transaction(transactionNo, amount, date, description, isRecurring);

    if (wallet.AddTransaction(newTransaction, catLabel))
    {
       Console.WriteLine($"Transaction added successfully. Transaction number: {transactionNo}");
    }
    else
    {
```

```
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to add transaction. Make sure the category exists and the date is within the category's budget period.");
        Console.ResetColor();
    }
}

static void ModifyTransactionAmountFromInput()
{
    Console.WriteLine("Enter the transaction ID:");
    string transactionId = Console.ReadLine();

    Console.WriteLine("Enter the new amount for the transaction:");
    if (!double.TryParse(Console.ReadLine(), out double newAmount))
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid amount format. Please enter a valid number.");
        Console.ResetColor();
        return;
    }
```

```
    if (wallet.ModifyTransactionAmount(transactionId, newAmount))
    {
        Console.WriteLine("Transaction amount updated successfully.");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to update transaction amount. Make sure the transaction ID is correct.");
        Console.ResetColor();
    }
}
static void MoveTransactionFromInput()
{
    Console.WriteLine("Enter the transaction ID you wish to move:");
    string transactionId = Console.ReadLine();

    Console.WriteLine("Enter the label of the new category for this transaction:");
    string newCategoryLabel = Console.ReadLine();

    bool result = wallet.MoveTransaction(transactionId, newCategoryLabel);
```

```
    if (result)
    {
        Console.WriteLine("Transaction successfully moved to the new category.");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to move the transaction. Please check the inputs and try again.");
        Console.ResetColor();
    }
}
static void RemoveTransactionFromInput()
{
    Console.WriteLine("Enter the transaction ID you wish to remove:");
    string transactionId = Console.ReadLine();

    if (wallet.RemoveTransaction(transactionId))
    {
        Console.WriteLine("Transaction removed successfully.");
```

```
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to remove transaction. Please ensure the transaction ID is correct.");
        Console.ResetColor();
    }
}
static void DisplayAllOngoingTransactionsFromInput()
{
    Console.WriteLine("Enter the date to view transactions (format dd/MM/yyyy):");
    string dateInput = Console.ReadLine();

    if (!DateTime.TryParseExact(dateInput, "dd/MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime currentDate))
    {
        Console.WriteLine("Invalid date format. Please enter the date in dd/MM/yyyy format.");
        return;
    }

    Console.WriteLine($"Displaying all transactions for {currentDate.ToString("dd/MM/yyyy")}:");
```

```
    wallet.DisplayAllOngoingTransactions(currentDate);
  }
  static void DisplayCategoryTransactionsFromInput()
  {
    Console.WriteLine("Enter the label of the category:");
    string label = Console.ReadLine();

    Console.WriteLine("Enter the date for the category transactions (format dd/MM/yyyy):");
    string dateInput = Console.ReadLine();

    if (!DateTime.TryParseExact(dateInput, "dd/MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime date))
    {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Invalid date format. Please enter the date in dd/MM/yyyy format.");
      Console.ResetColor();
      return;
    }

    Console.WriteLine($"Displaying transactions for category '{label}' on {date.ToString("dd/MM/yyyy")}:");
    wallet.DisplayCategoryTransactions(label, date);
```

```
    }
    static void DisplayAllPastTransactionsFromInput()
    {
        Console.WriteLine("Displaying all past transactions:");
        wallet.DisplayAllPastTransactions();
    }
    static void DisplayPastCategoryTransactionsFromInput()
    {
        Console.WriteLine("Enter the label of the category to display past transactions:");
        string label = Console.ReadLine();

        Console.WriteLine($"Displaying past transactions for category: {label}");
        wallet.DisplayPastCategoryTransaction(label);
    }
    static void CloseAndOpenCategoriesFromInput()
    {
        Console.WriteLine("Enter the label of the category to close and reopen for the new month:");
        string catLabel = Console.ReadLine();

        Console.WriteLine("Enter the new month and year for the category (format MM/yyyy):");
```

```
    string newMonthInput = Console.ReadLine();

    if (DateTime.TryParseExact(newMonthInput, "MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime newMonth))
    {
      if (wallet.CloseAndOpenCategories(catLabel, newMonth))
      {
        Console.WriteLine("Category closed for the current month and opened for the new month successfully.");
      }
      else
      {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Failed to close and open category. Make sure the category exists.");
        Console.ResetColor();
      }
    }
    else
    {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Invalid date format.");
      Console.ResetColor();
```

```
    }
  }

  static void ManageRecurringTransactions()
  {
    Console.WriteLine("Do you want to carry over recurring transactions from the last month?");
    Console.WriteLine("1. Yes");
    Console.WriteLine("2. No");
    Console.Write("\nSelect an option (1-2): ");

    if (!int.TryParse(Console.ReadLine(), out int choice) || choice < 1 || choice > 2)
    {
      Console.ForegroundColor = ConsoleColor.DarkRed;
      Console.WriteLine("Invalid choice, please try again.");
      Console.ResetColor();
      return;
    }

    if (choice == 1)
    {
```

```
        bool success = wallet.ApplyRepeatingTransactions();

        if (success)
        {
            Console.WriteLine("Recurring transactions carried over successfully.");
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine("Failed to carry over recurring transactions.");
            Console.ResetColor();
        }
    }
    else
    {
        Console.WriteLine("No recurring transactions carried over.");
    }
}

#endregion
```

```
static void TrackOverallBudgetFromInput()
{
    Console.WriteLine("Enter the month and year to track overall budget and expenses (format MM/yyyy):");
    string monthYearInput = Console.ReadLine();

    if (DateTime.TryParseExact(monthYearInput, "MM/yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime monthYear))
    {
        wallet.TrackOverallBudget(monthYear);
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid date format. Please follow the MM/yyyy format.");
        Console.ResetColor();
    }

    ContinuePrompt();

}
```

}