Criterion A: Planning

An interview was held with the client, Mr. Hari, on 19/09/2023. The interview transcript has been included in Appendix A1. Please refer to Appendix A1 for further details.

The Scenario (Based upon Appendix A1)

My client is Mr. Hari, a Regional Manager at Store XXX. He said that he manages a large volume of transactions for the many stores in his region as well as for his family. However, he has expressed his discontent with his initial and current finance management systems.

Initially, he said that he used a pen-and-paper approach to record his personal and professional transactions. However, this approach had several issues. One key issue according to him was searching through past transactions because he had to go through multiple folders manually, which took up a lot of time. Moreover, Mr. Hari mentioned an instance where he mixed up his personal and office transactions, highlighting the unsuitability of this manual approach for the management of his critical home and office finances as he could mix them. Editing or deleting transactions was another major pain point according to him, as he had to redo many folders of paperwork, which required significant effort and led to immense frustration.

One significant issue was the difficulty in managing and searching through several Excel sheets. A major problem was the inability to access and update his professional financial records in real time while at work, as the Excel file was only stored on his home MacBook. This limitation added a layer of inconvenience and inefficiency, as he couldn't manage his many important office transactions during the workday. Lastly, the lack of data backup increases the risk of losing his critical financial data, especially since he said his children could damage his Macbook, potentially leading to the corruption of his vital financial data.

Mr. Hari now desires a more streamlined, reliable solution that allows him to effectively manage his finances from both devices seamlessly, without the inefficiencies of his current systems.

Rationale for proposed solution

As my client uses two primary computer devices with different operating systems, a web application which can be easily accessed by both systems while also not affecting the devices' performance would be most suitable. The goal of the application is to provide him with the aforementioned features.

The application will be primarily developed using a combination of JavaScript and JSX, with some use of HTML5 and CSS for basic structuring and styling respectively of the user interface. Using the React JavaScript framework will enable performant cross-platform (Windows and MacOS) usage of the application in both of the client's devices, while also ensuring that his financial information is updated dynamically from the Firestore database.

Firestore's cloud storage functionality enables the client to access the database for storing and retrieving his financial data on either his Macbook or Windows systems, hence solving the problem with his current

system regarding not being able to record his professional transactions at the office. Moreover, Firestore also allows the client to store a large volume of transactions data in the database for free.

Success Criteria (Based upon Appendix A1)

1. The client must be able to switch between his personal and work Gmail accounts without losing any of his financial data. Each account's data should be isolated, ensuring that only the transactions and categories related to the signed-in account are accessible.
2. The application must ensure that sensitive financial data is only accessible when the client is logged in using one of their Gmail accounts. Attempts to access the application without logging in must redirect the user to the sign in page.
3. The client should be able to add new incomes and expenses on the Home page. The transactions must also be validated – transaction amounts cannot exceed 100,000 and transaction names cannot be longer than 20 characters long.
4. The client must be able to view five of his recently made transactions in the Home page.
5. The client must have the ability to edit or delete existing transactions. Editing transactions will carry the same validation limitations (maximum amount being 100,000 and maximum transaction name length being 20 characters).
6. In order to segregate transactions, the solution should have the functionality to create, edit, and delete transaction categories, and they should be less than or equal to 25 characters in length.
7. The client must be able to filter transactions by their type (income/expense), category, date added and the magnitude of the amount.
8. The implemented solution should display monthly transaction data in graphs (daily incomes/expenses, daily balance, and monthly categorical earnings/spendings). This data must be filtered by month and years.
9. The application must be fully accessible and functional on both the client's MacBook and Windows desktops. The interface should maintain consistency across devices, ensuring that there is no loss of functionality between platforms.
10. The product should be able to handle data validation errors and provide feedback regarding the completion of database operations with clear success/warning/error messages.
11. All changes (adding, editing, or deleting transactions and categories) must be updated in real-time across the client's devices and reflected immediately in the application.

Word count: 502

Criterion B: Record of tasks

Task number	Planned action	Planned outcome	Time estimated	Target completion date	Criterion
1	Discussion of the project with the CS teacher	Clearing of doubts regarding app development for the IA and getting approval	1 day	18/09/2023	Planning
2	Preliminary discussion with Mr. Hari (client)	Determine his problem and his current method of dealing with it	1 day	19/09/2023	Planning
3	Meeting with advisor	Determine the product details (languages and frameworks to be used etc.) and clear development-related doubts	1 day	20/09/2023	Planning
4	Solution discussion with Mr. Hari	Solution proposed to client and his approval gained	1 day	21/09/2023	Planning
5	Set project tasks and deadlines	To keep myself on track with the development process	1 days	22/09/2023	Planning
6	Learning Firebase's authentication and data storage features	Have the knowledge necessary for implementing Firebase services in the product	1 day	23/09/2023	Development
7	Designing SignIn page	Have an outline of the SignIn page and its components	1 day	24/09/2023	Design
8	Designing Home page	Have an outline of the Home page and its components	1 day	25/09/2023	Design

9	Designing Budget page	Have an outline of the Budget page and its components	1 day	26/09/2023	Design
10	Designing Reports page	Have an outline of the Reports page and its components	1 day	27/09/2023	Design
11	Designing Settings page	Have an outline of the Settings page and its components	1 day	28/09/2023	Design
12	Obtaining client approval for application design	Present the design outlines of the pages of the application to the client for his approval to commence development	1 day	29/09/2023	Design
13	Creating SignIn page	To allow the client to sign in using his Google account	1 day	30/09/2023	Development
14	Testing SignIn page	To find any errors in the sign in process	1 day	1/10/2023	Testing
15	Fixing errors relating to database connection	Troubleshooting and fixing errors with the database connection	2 days	3/10/2023	Development
16	Creating IncomeForm component	To allow the client to add a new income to the database	1 day	5/10/2023	Development
17	Creating ExpenseForm component	To allow the client to add a new expense to the database	1 day	6/10/2023	Development
18	Testing the IncomeForm and ExpenseForm components	To find if there are any errors with the components	1 day	7/10/2023	Testing

19	Creating RecentTransactionsTable component	To allow client to view his four recent transactions	1 day	8/10/2023	Development
20	Testing the RecentTransactionsTable component	Determining and fixing errors with the display of the recent transactions	1 day	9/10/2023	Testing
21	Creating Budget page	Allow client to view and filter his transactions	2 days	11/10/2023	Development
22	Creating TransactionFilter component	To allow the client to filter the transactions by the transactions type (income/expense)	1 day	12/10/2023	Development
23	Testing TransactionFilter component	Checking for errors with the filtering algorithm	1 day	13/10/2023	Testing
24	Creating DateFilter component	To allow the client to filter the transactions by time	1 day	14/10/2023	Development
25	Testing DateFilter component	Checking for any errors or problems with the filtering algorithm	1 day	15/10/2023	Testing
26	Creating AmountFilter component	To allow the client to filter the transactions by the magnitude of the transaction	1 day	16/10/2023	Development
27	Testing AmountFilter component	Checking for any errors or problems with the filtering algorithm	1 day	17/10/2023	Testing
28	Creating CategoryFilter component	To allow the client to filter the transactions by the category of the transactions	1 day	18/10/2023	Development

29	Testing CategoryFilter component	Checking for any errors or problems with the filtering algorithm	1 day	19/10/2023	Testing
30	Testing all the transactions filtering components together	To find any errors in the combined query that is sent to the database	1 day	20/10/2023	Testing
31	Fixing database query issues in the Budget page	Troubleshooting and fixing errors relating to the combined database query in the Budget page	3 days	23/10/2023	Development
32	Creating Reports page	Creating graphs to allow the client to view the daily incomes, expenses, and balances for a specific selected month and year	2 days	25/10/2023	Development
33	Creating MonthlyIncomeGraph, MonthlyExpenseGraph, and MonthlyBalanceGraph components	To allow the client to view the daily income/expense/balance for a specific month and year	3 days	28/10/2023	Development
34	Testing functionality of aforementioned graph components	Checking for any errors or problems with the combined query in the graph components	1 day	29/10/2023	Testing
35	Creating MonthlyIncomePiechart and MonthlyExpensePiechart components	To allow the client to view which categories they have the most income/expense from for every month	1 day	30/10/2023	Development

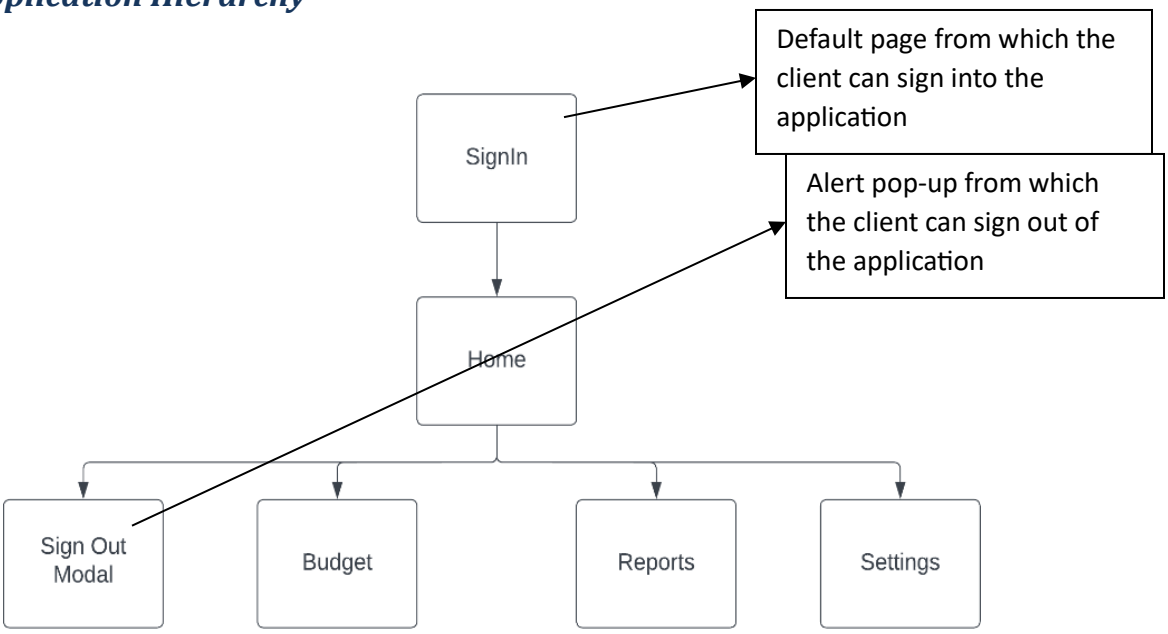
36	Testing MonthlyIncomePiechart and MonthlyExpense Piechart components	Checking for any errors or problems with the database query	1 day	31/10/2023	Testing
37	Creating Settings page	To plan out how to design and where to place the CategoryTable component	1 day	1/11/2023	Development
38	Creating CategoryTable component	To allow the client to create, view, edit or deletetransaction categories	1 day	2/11/2023	Development
39	Testing CategoryTable component	To check for any errors in the database queries	1 day	3/11/2023	Testing
40	Obtaining client approval for product	Presenting the product to the client to gain his approval for further testing	1 day	4/11/2023	Testing
41	Conducting alpha testing	Checking for errors or problems with the overall application	1 day	5/11/2023	Testing
42	Conducting user acceptance testing	For getting client's initial product feedback	1 day	6/11/2023	Testing
43	Implementing product in client's systems	For obtaining the client's feedback for the product	7 days	13/11/2023	Implementation
44	Feedback meeting with client	Listening to the client's experience with the product	1 day	14/11/2023	Implementation
45	Implementing client's suggestions in the product	Adding graph axes to the MonthlyIncomeGraph,	2 days	16/11/2023	Development

		MonthlyExpenseGraph and MonthlyBalanceGraph components as requested by the client during the feedback meeting			
46	Handover of product	Installation of final product on client's systems	1 day	17/11/2023	Implementation

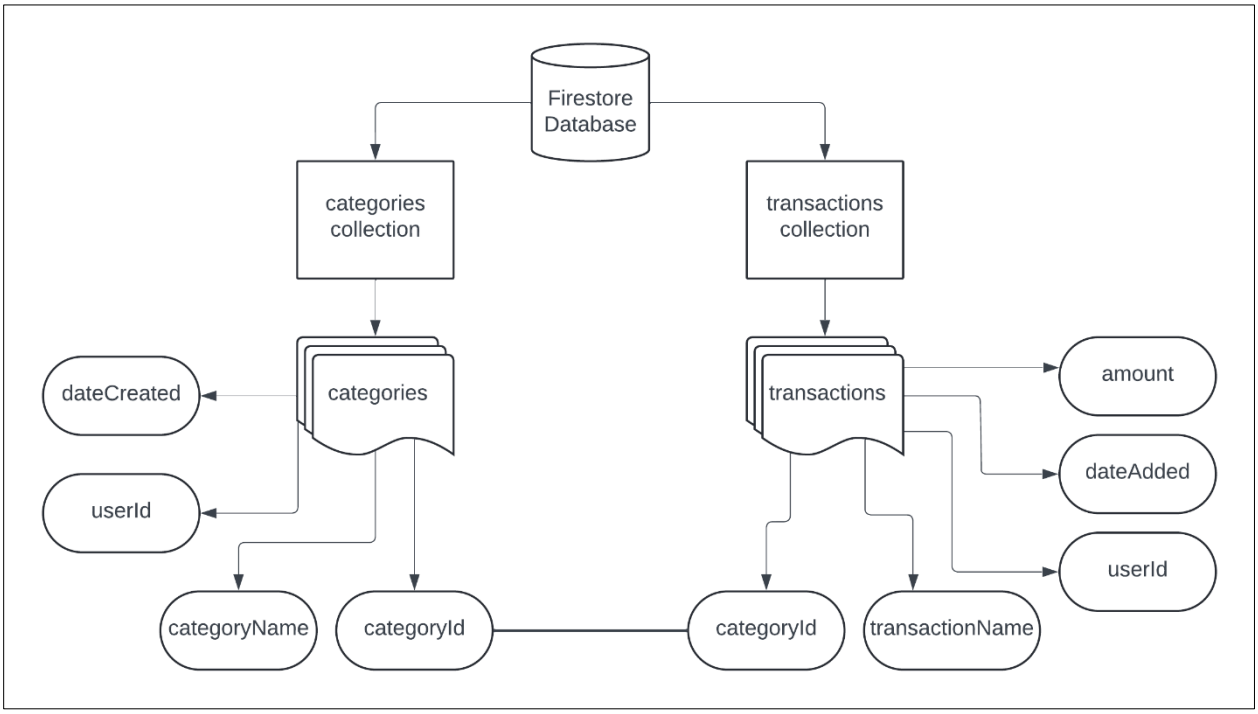
Criterion B: Design Overview

All user interface designs were made using the graphic design platform Canva *(Canva)*.

Application Hierarchy

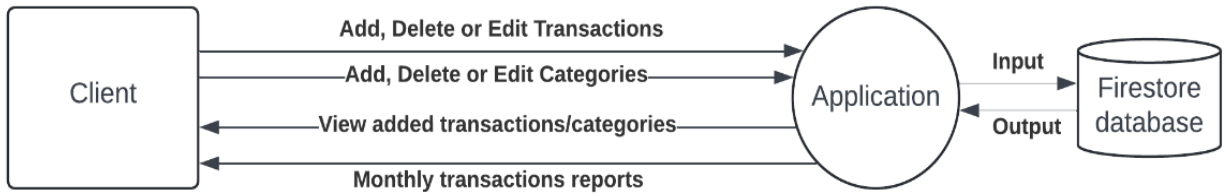


Firestore Database Model

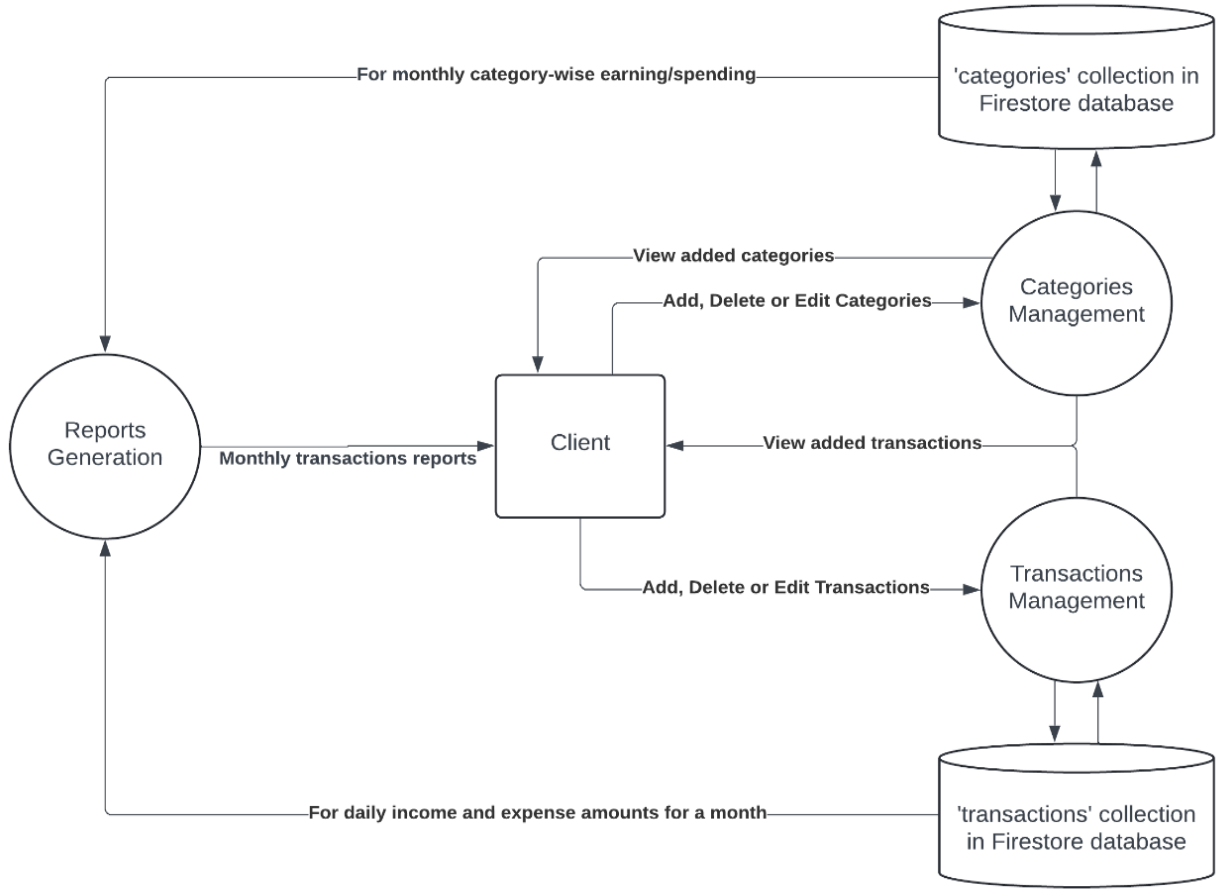


Application Data Flow Diagrams (DFDs)

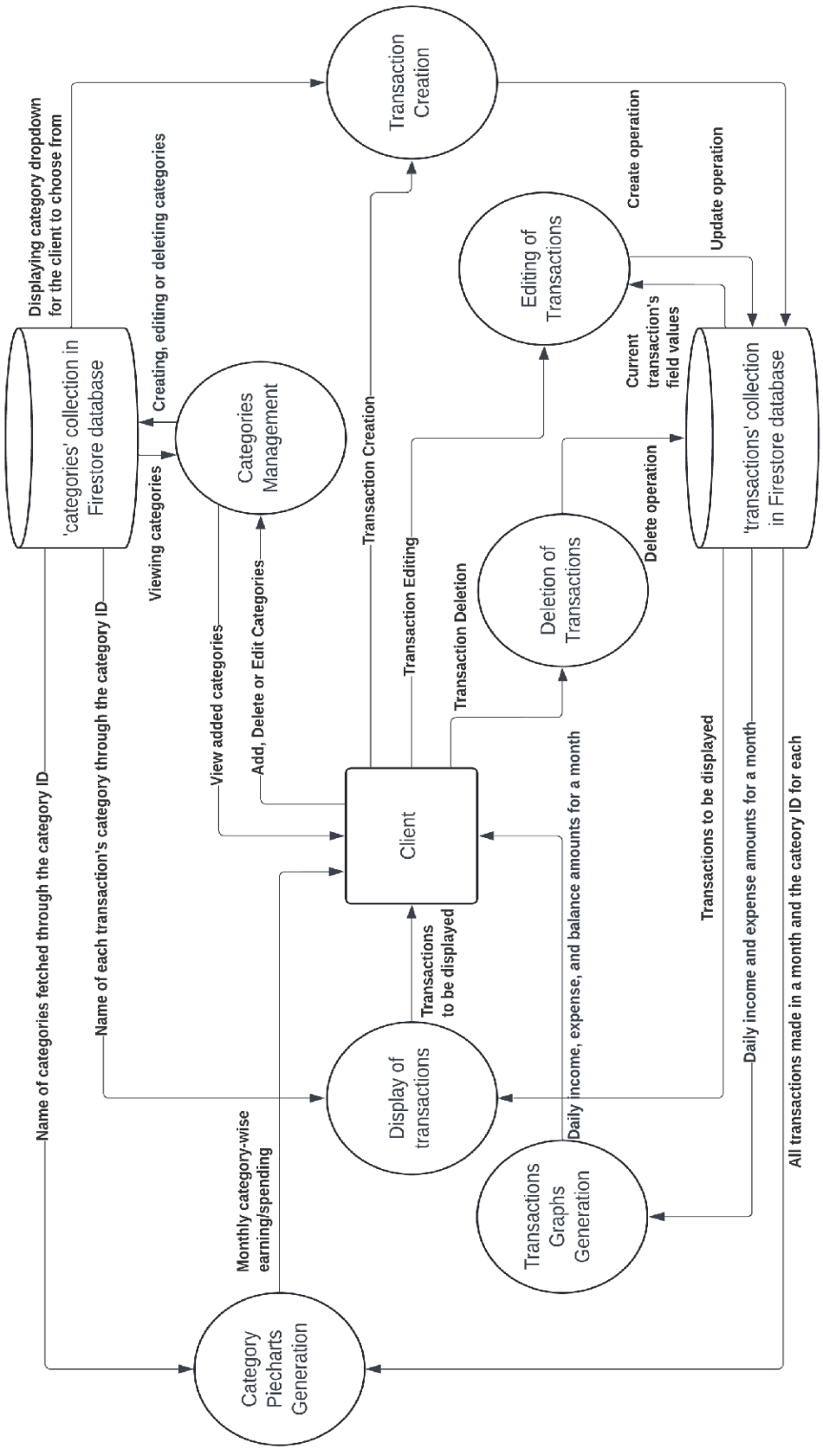
Level 0



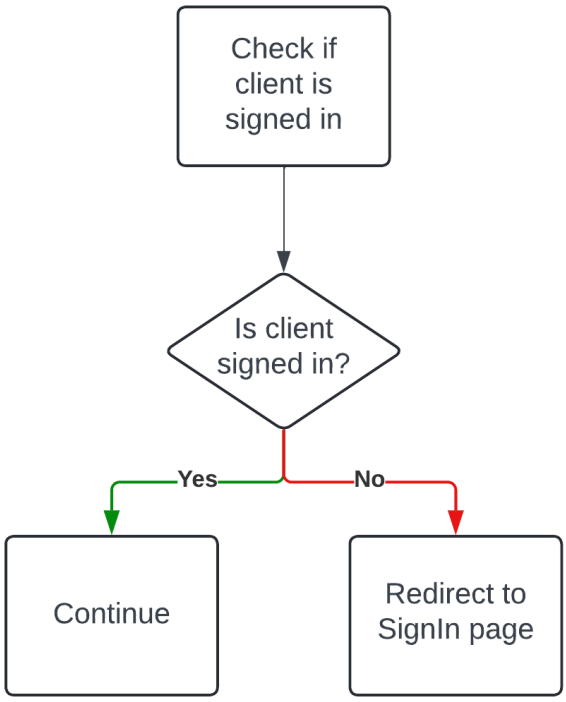
Level 1



Level 2


Transaction Creation
Displaying category dropdown for the client to choose from
Creating, editing or deleting categories
Editing of Transactions
Create operation
Update operation
'categories' collection in Firestore database
Categories Management
Current transaction's field values
'transactions' collection in Firestore database
Viewing categories
Transaction Creation
Transaction Editing
Transaction Deletion
Delete operation
Deletion of Transactions
View added categories
Add, Delete or Edit Categories
Client
Name of each transaction's category through the category ID
Name of categories fetched through the category ID
Transactions to be displayed
Transactions to be displayed
Daily income, expense, and balance amounts for a month
Display of transactions
Monthly category-wise earning/spending
Daily income and expense amounts for a month
Transactions Graphs Generation
All transactions made in a month and the cateory ID for each
Category Piecharts Generation

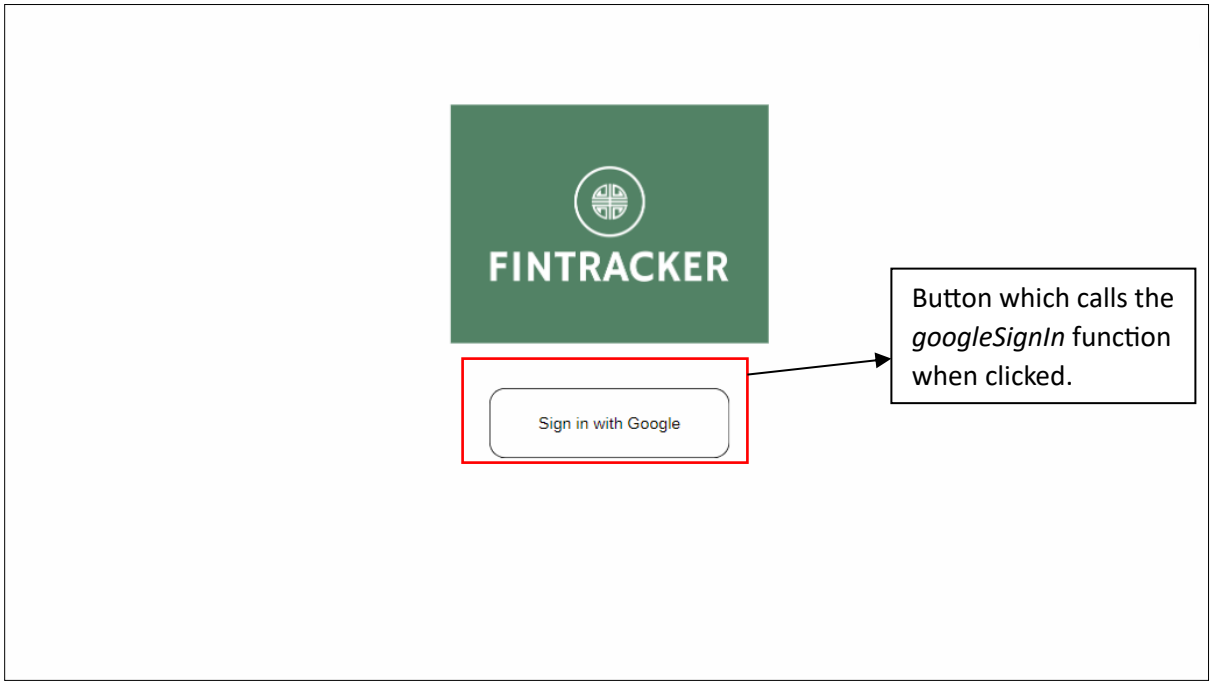
Authentication Check Flowchart and Pseudocode for All Pages

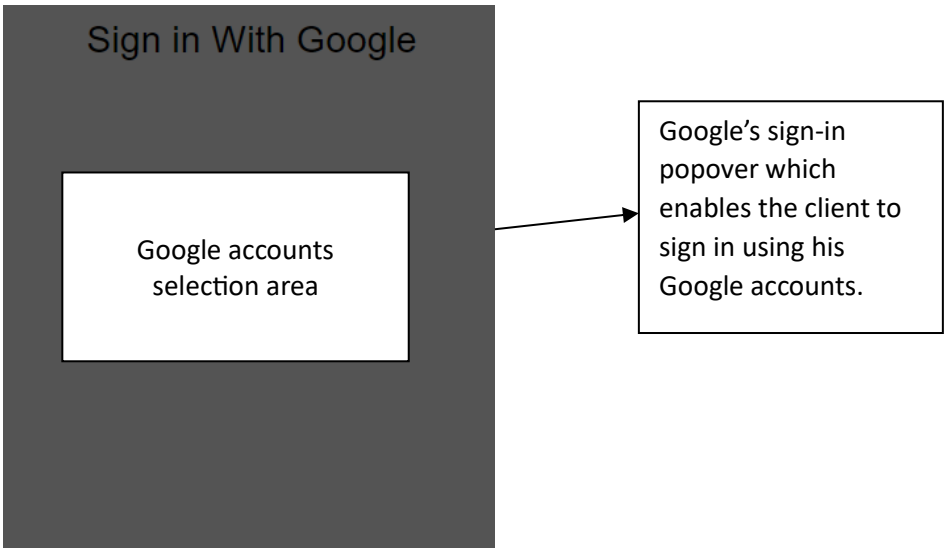



```
signedInStatus = checkSignInStatus()  // Check if the client is signed in

if signedInStatus = false then  // If true, then the page loads as normal
  redirectToSignInPage()  // Redirects client to SignIn page if he is not signed in with a Gmail account
end if
```

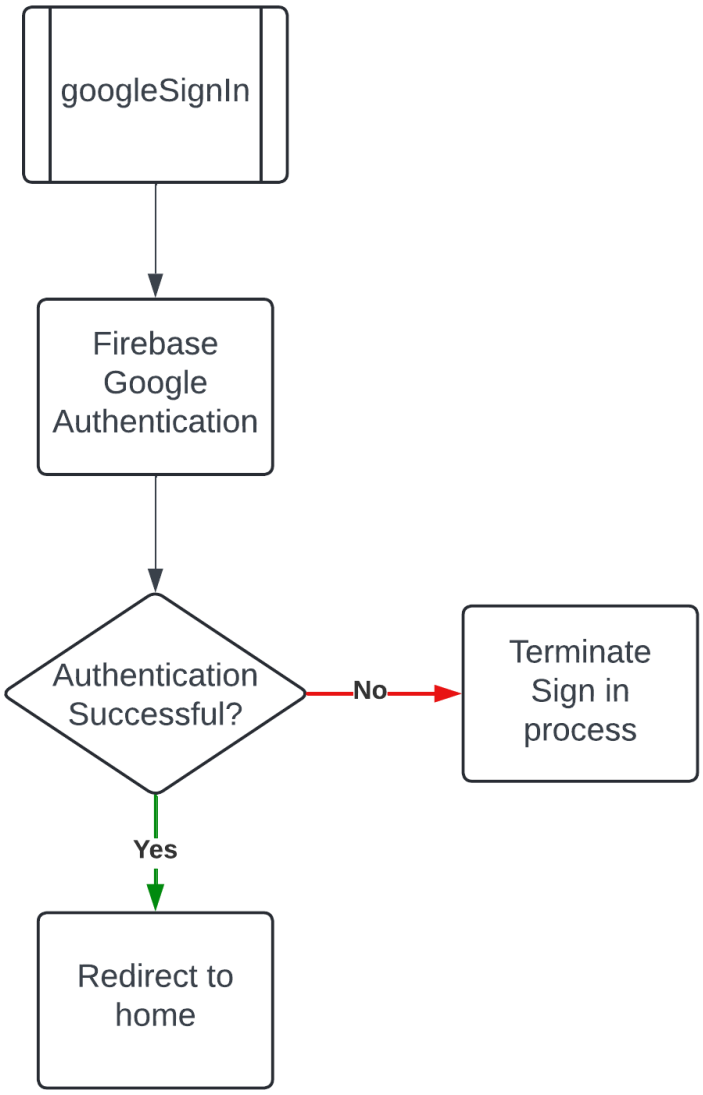
SignIn Page – Design, Structure, and Flowcharts

Page Design and Sign in with Google screen popover



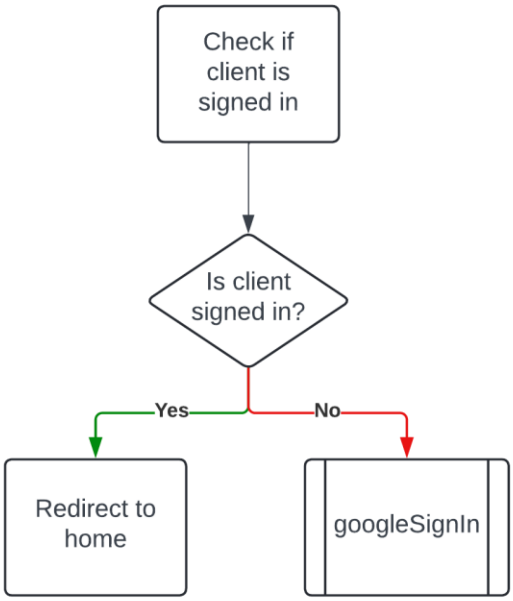


googleSignIn Flowchart and Pseudocode




```
subprogram googleSignIn (
  authenticationStatus = firebaseGoogleAuth()  // Perform Firebase Google Authentication

  if authenticationStatus = true then  // Check if authentication was successful
    redirectToHomePage()  // Redirect to the Home page if client signs in successfully
  else
    cancel()  // Terminate the sign in process if a failure occurs
    alert("Sign in unsuccessful, try again.")  // Inform client sign in process was unsuccessful.
  end if
)
```

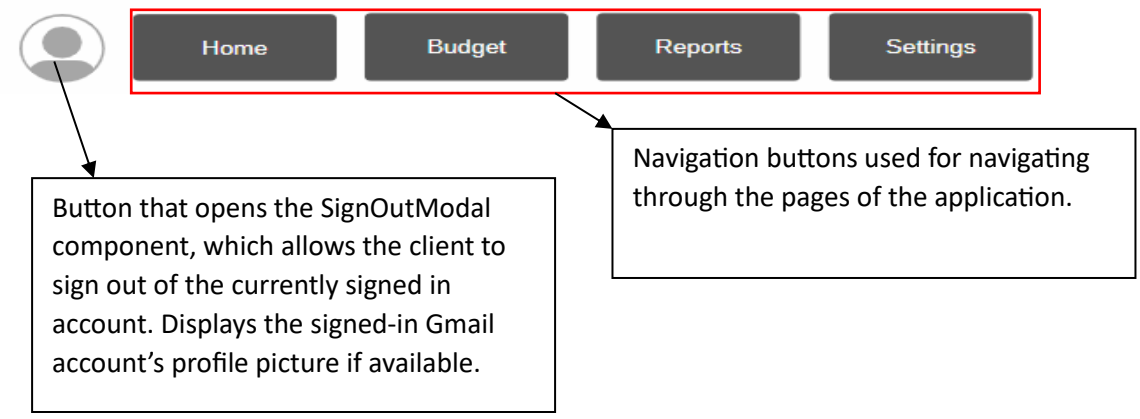
Authentication Check Flowchart and Pseudocode (for SignIn page only)



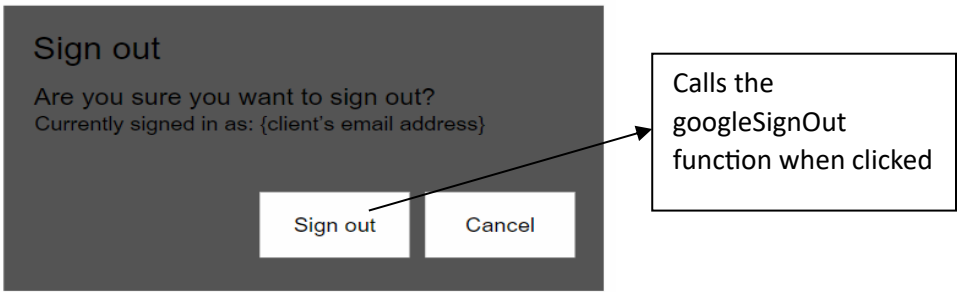

```
signedInStatus = checkSignInStatus()  // Check if the client is signed in

if signedInStatus = true then  // If true, then the client gets redirected to the Home page
  redirectToHomePage()  // Redirects client to SignIn page if he is not signed in with a Gmail account
else
  googleSignIn()
end if
```

Header Component

Design



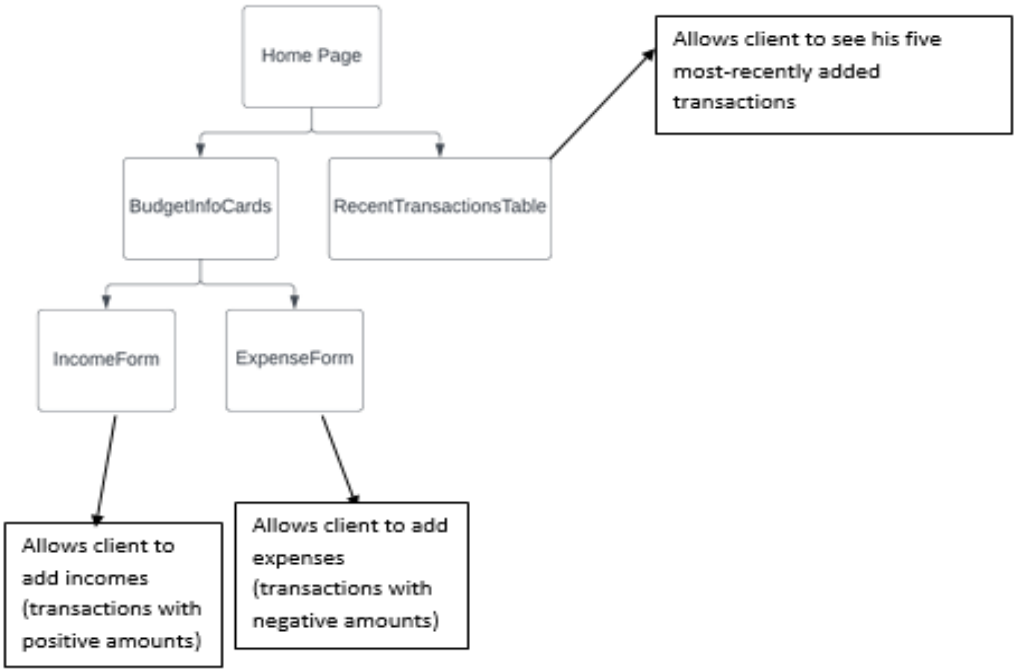
Header – Sign Out Modal



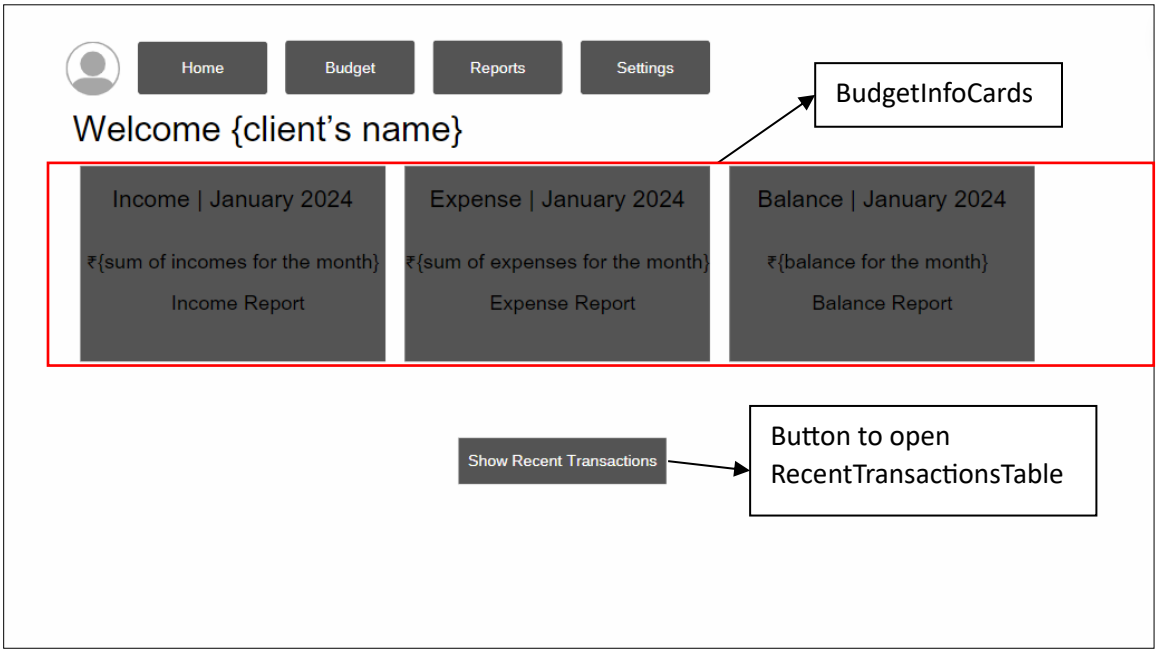

```
subprogram googleSignOut (
  signOut = firebaseGoogleSignOut()  // Sign out function from Firebase

  if signOut = true then  // Check if signOut was successful
    redirectToSignInPage()  // Redirect to the Home page if client signs in successfully
  else
    cancel()  // Terminate the sign out process if a failure occurs
    alert("Sign out unsuccessful, try again.")  // Inform client sign out process was unsuccessful.
  end if
)
```

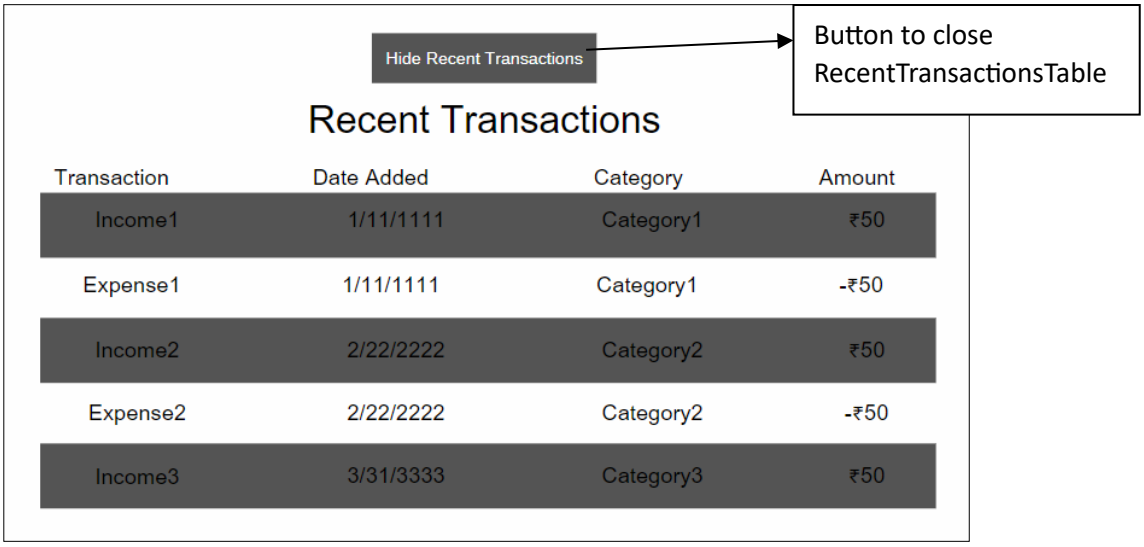
Home Page – Design, Structure, and Flowcharts

Structure

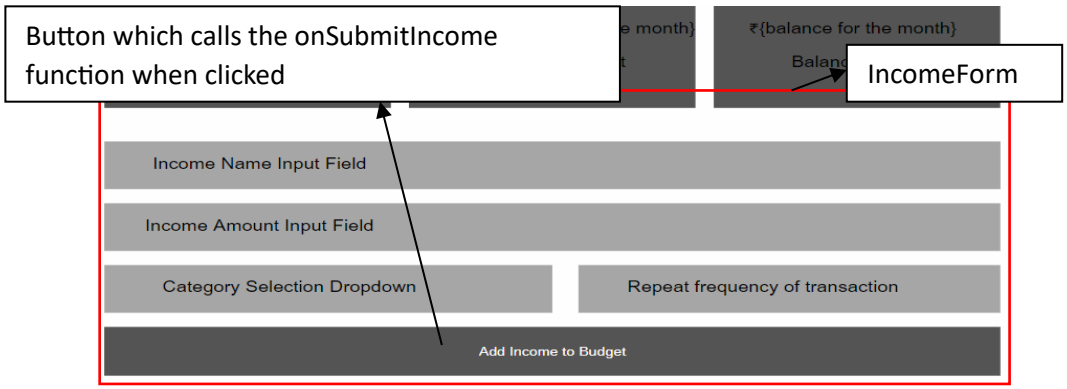


Page Design and RecentTransactionsTable when Opened

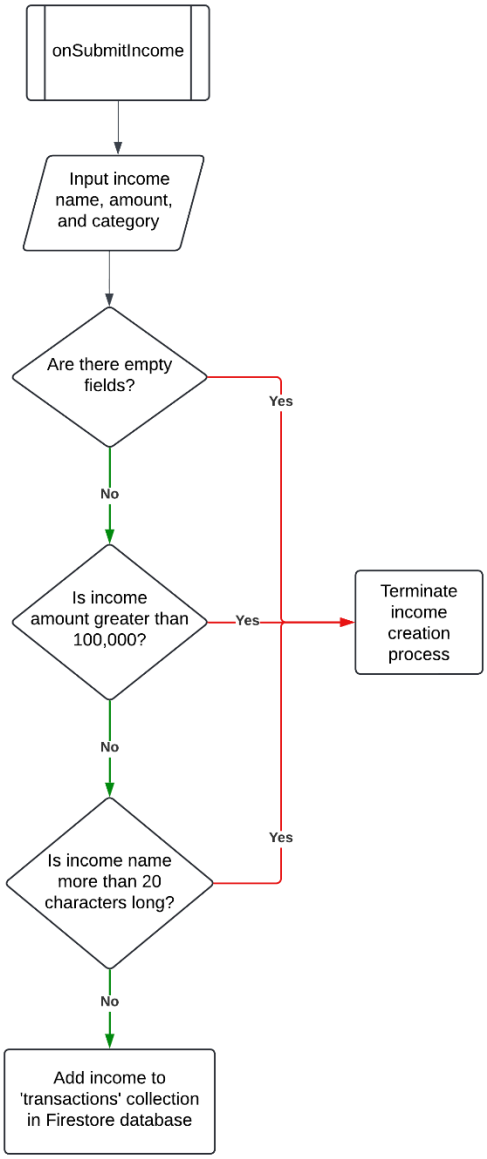




IncomeForm

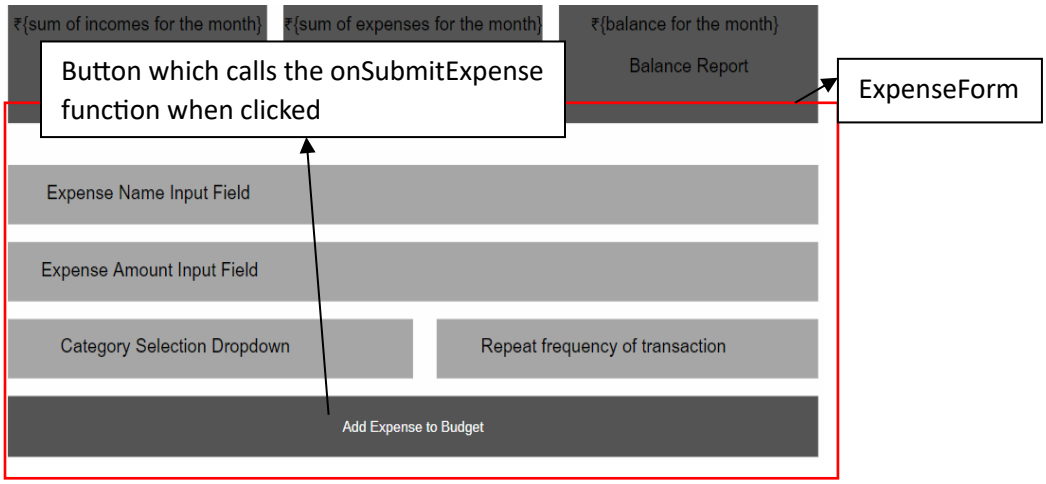


onSubmitIncome Flowchart and Pseudocode

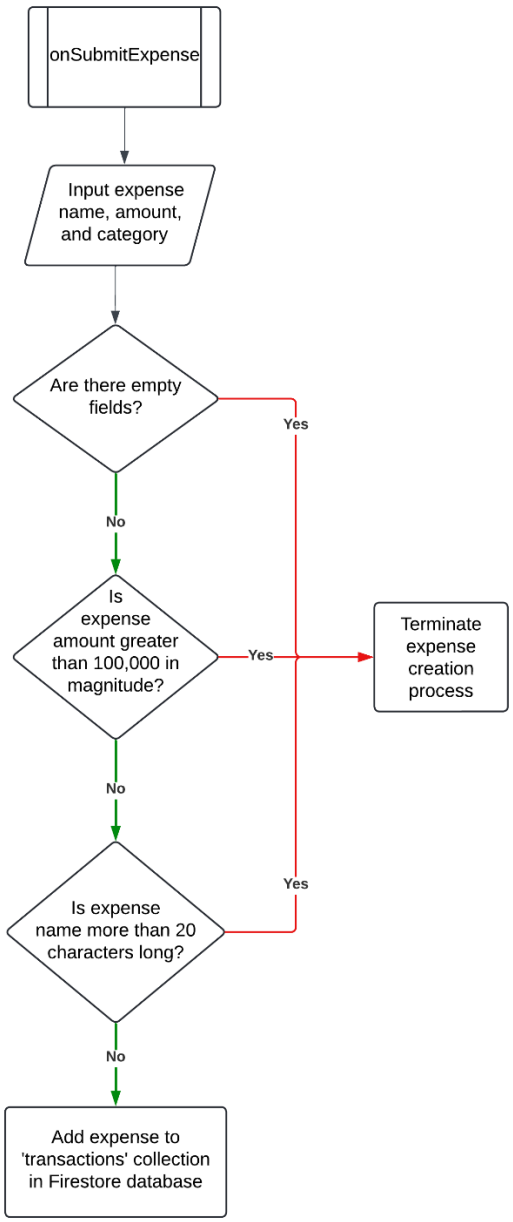


```
subprogram onSubmitIncome(
  incomeDocument = input(incomeName, incomeAmount, transactionCategory)

  if anyFieldIsEmpty(incomeDocument) = true then  // Check if any of the input fields are empty
    cancel()  // Terminate income creation process
    alert("Please fill out all the fields.")  // Inform client that all the fields need to be filled out.

  else if incomeAmount > 100,000 then  // Check if the income amount is greater than 100,000
    cancel()
    alert("Income amount should be less than or equal to 100,000")
    // Inform client that incomeAmount should  be less than or equal to 100,000.

  else if incomeName.length  > 20 then
  // Check if incomeName has more than 20 characters.
    cancel()
    alert("Income name should be less than 20 characters long")
    // Inform client that incomeName should be less than 20 characters long.

  else
    addToFirestore(incomeDocument)
    // Add income to the 'transactions' collection in the Firestore database.
  end if
)
```

ExpenseForm



onSubmitExpense Flowchart and Pseudocode

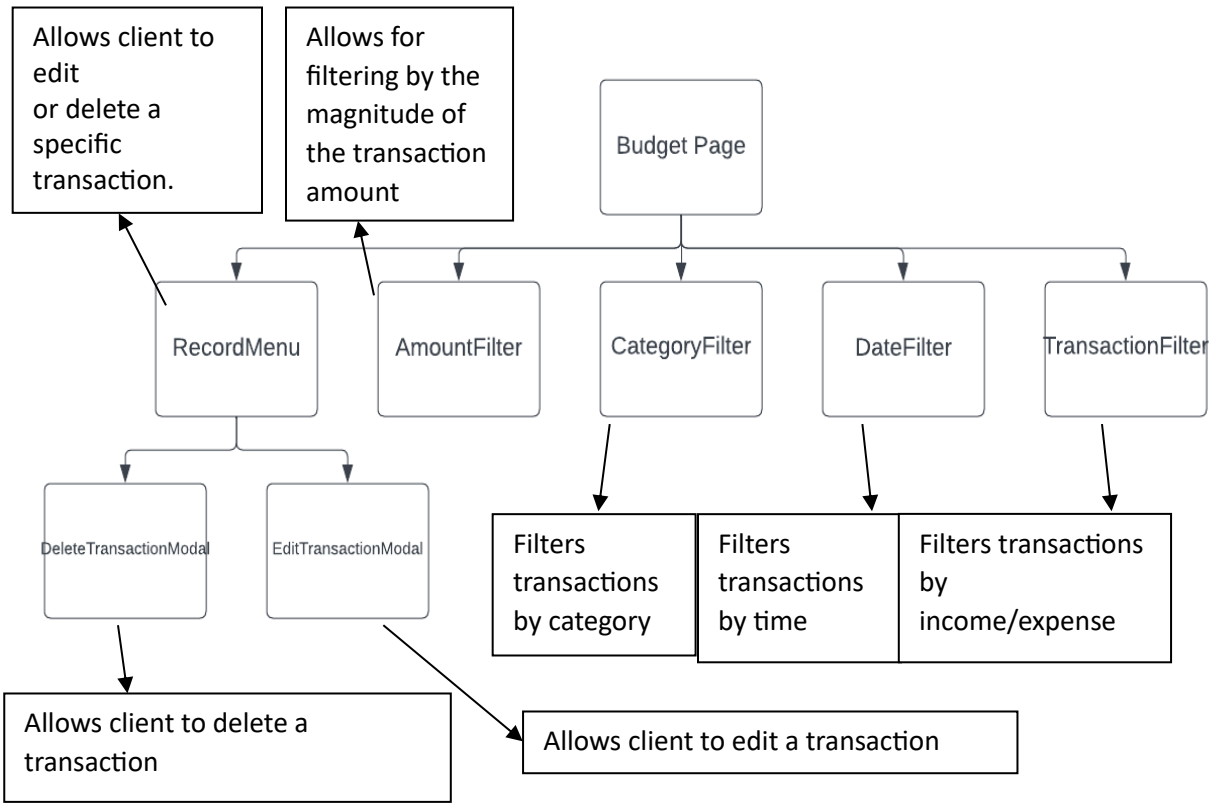



```
subprogram onSubmitExpense(
  expenseDocument = input(expenseName, expenseAmount, transactionCategory)

  if anyFieldIsEmpty(expenseDocument) = true then  // Check if any of the input fields are empty
    cancel()  // Terminate expense creation process
    alert("Please fill out all the fields.")  // Inform client that all the fields need to be filled out.
```

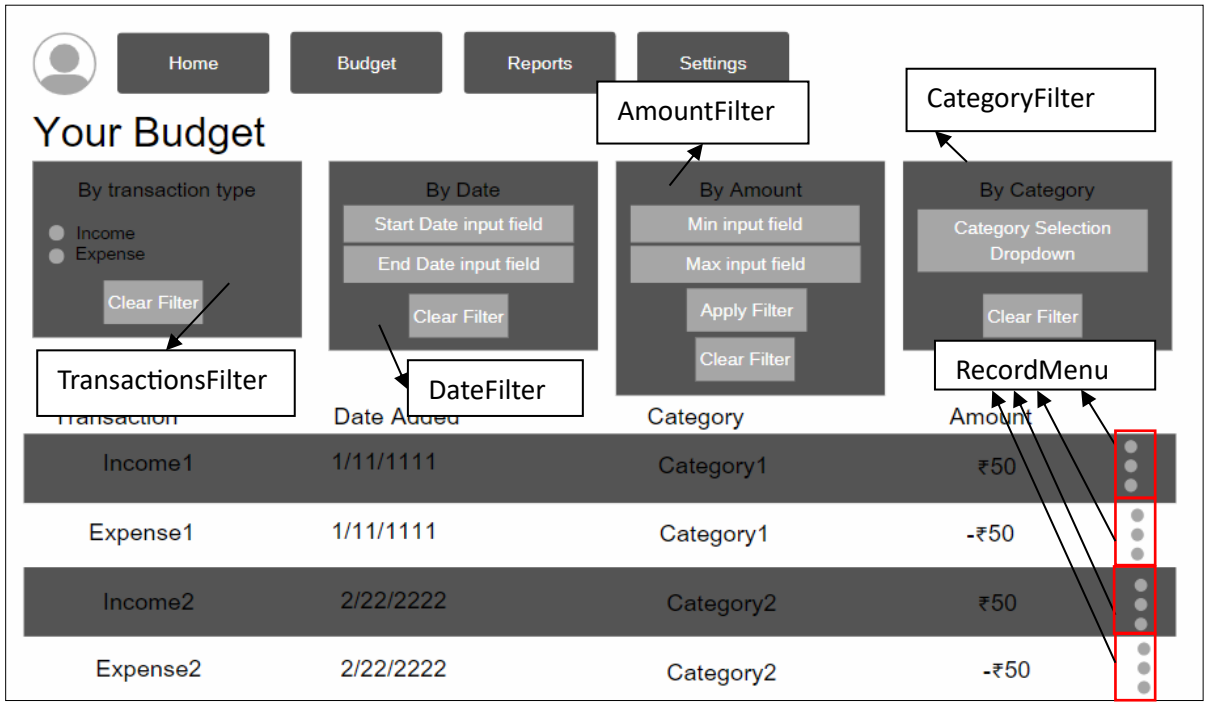
```
  else if expenseAmount > 100,000 then  // Check if the expense amount is greater than 100,000
    cancel()
    alert("Expense amount should be less than or equal to 100,000")
    // Inform client that expenseAmount should  be less than or equal to 100,000.

  else if expenseName.length  > 20 then
  // Check if expenseName has more than 20 characters.
    cancel()
    alert("Expense name should less than 20 characters long")
    // Inform client that expenseName should be less than 20characters long.

  else
    addToFirestore(expenseDocument)
    // Add expense to the 'transactions' collection in the Firestore database.
  end if
)
```

Budget Page – Design, Structure, and Flowcharts

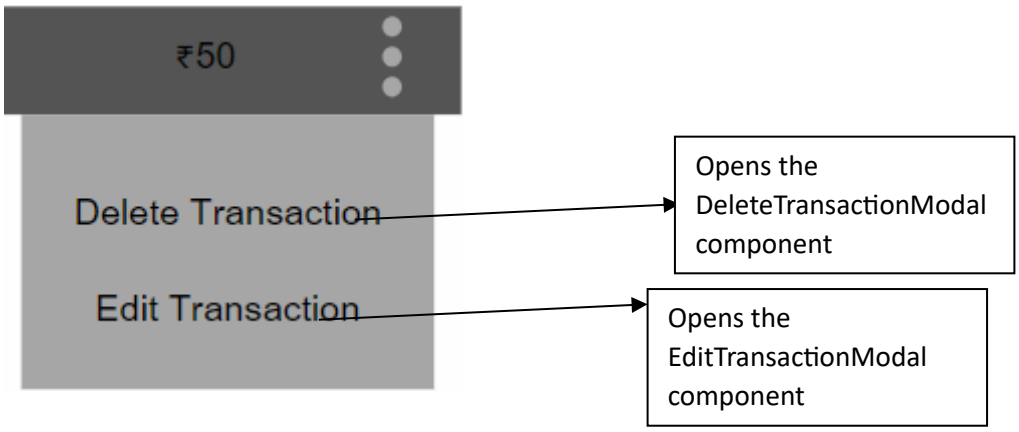
Structure



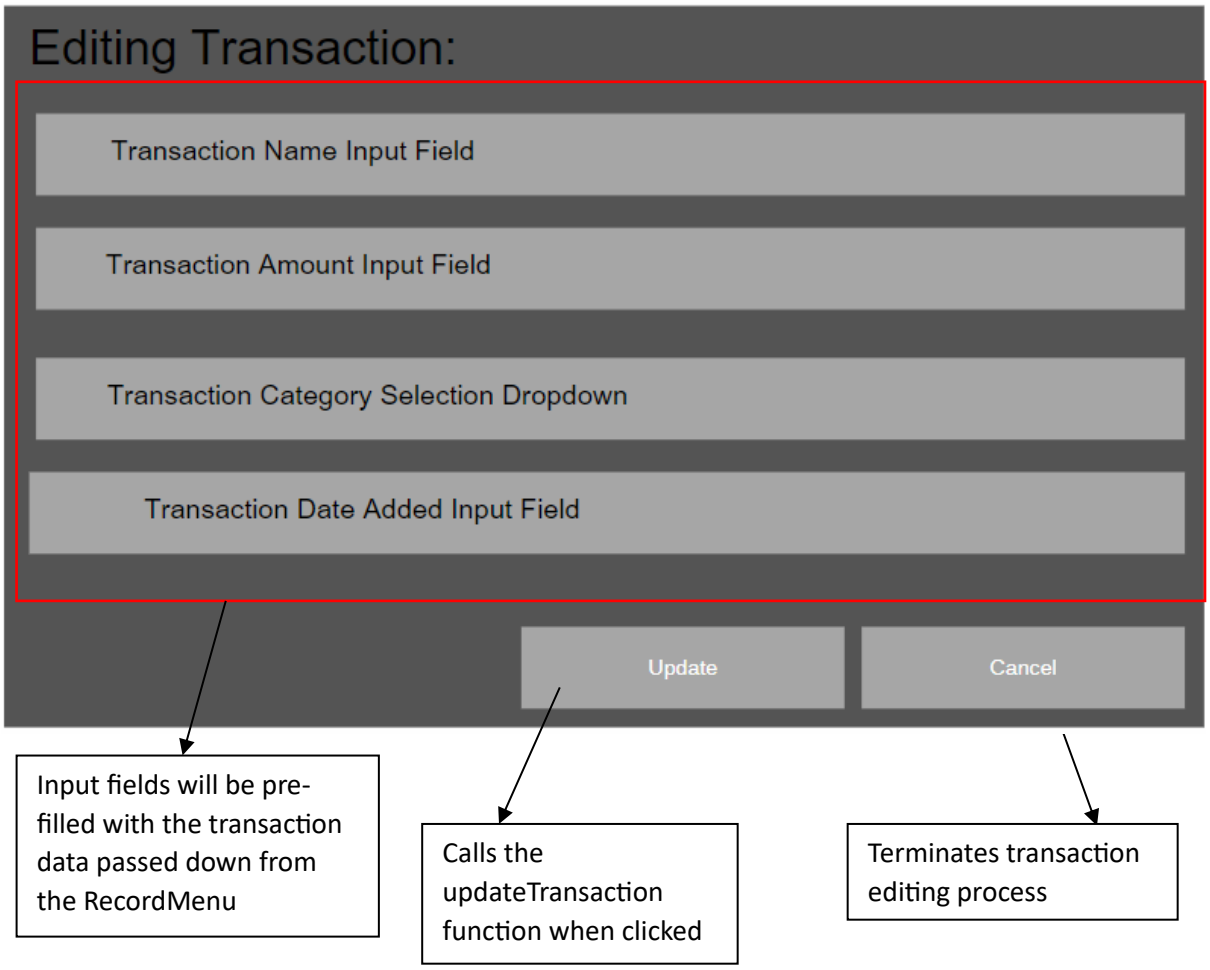
Page Design



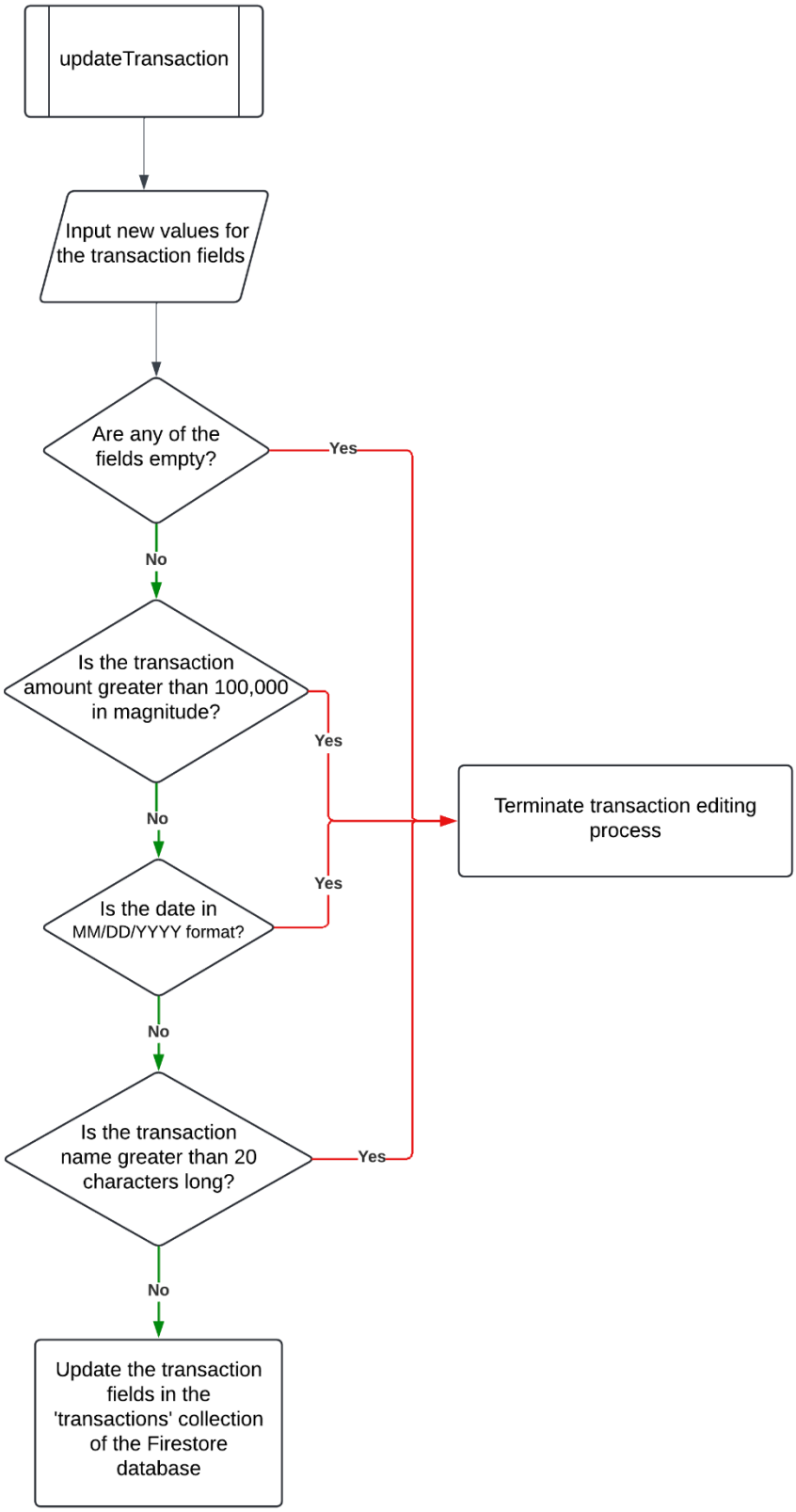
RecordMenu when Opened



EditTransactionsModal

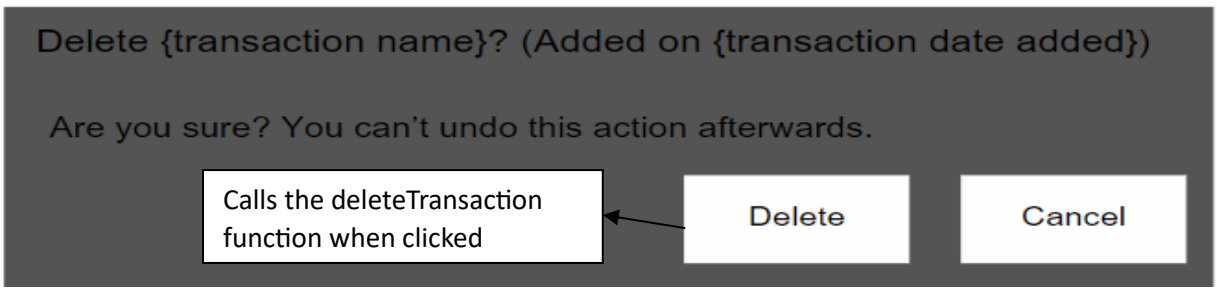


updateTransaction Flowchart and Pseudocode

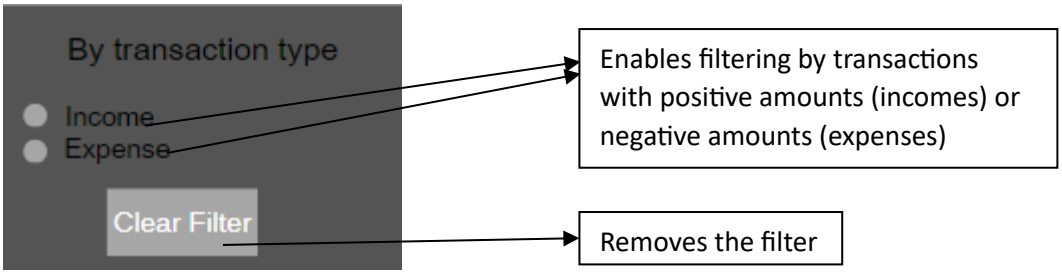


```
subprogram updateTransaction(
  transactionDocument = input(transactionName, transactionAmount, transactionCategory, transaction Date)

  if anyFieldIsEmpty(transactionDocument) = true then  // Check if any of the input fields are empty
    cancel()  // Terminate transaction update process
    alert("Please fill out all the fields.")  // Inform client that all the fields need to be filled out.

  else if transactionAmount > 100,000 then
  // Check if the transaction amount is greater than 100,000.
    cancel()
    alert("Transaction amount should be less than 100,000")
    // Inform client that transactionAmount should be less than 100,000.

  else if transactionDate ≠ "MM/DD/YYYY" then // Check if date is in MM/DD/YYYY format
    cancel()
    alert("Date must be in MM/DD/YYYY format.")
    // Inform client that transactionDate must be in MM/DD/YYYY format.

  else if transactionName.length > 20 then // Check if name is more than 20 characters long
    cancel()
    alert("Transaction name should be less than 20 characters long.")
    // Inform client that transactionName must be less than 20 characters long.

  else
    updateInFirestore(transactionDocument)
    // Update the transaction document in the 'transactions' collection in the Firestore database.
  end if
)
```

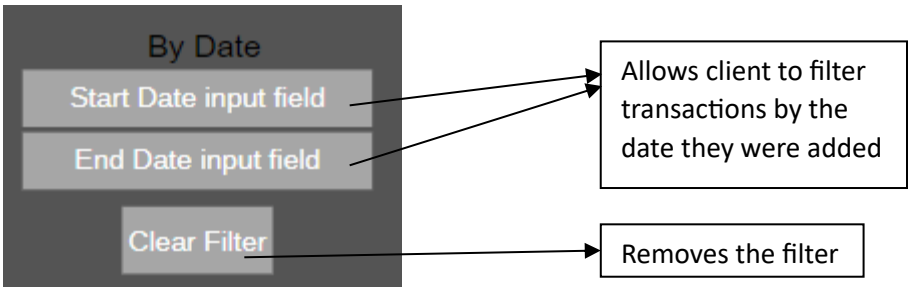
DeleteTransactionsModal



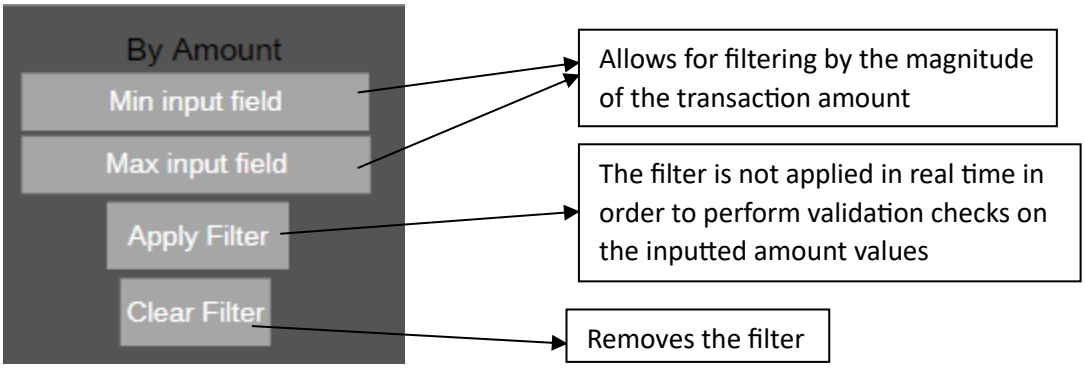
TransactionFilter



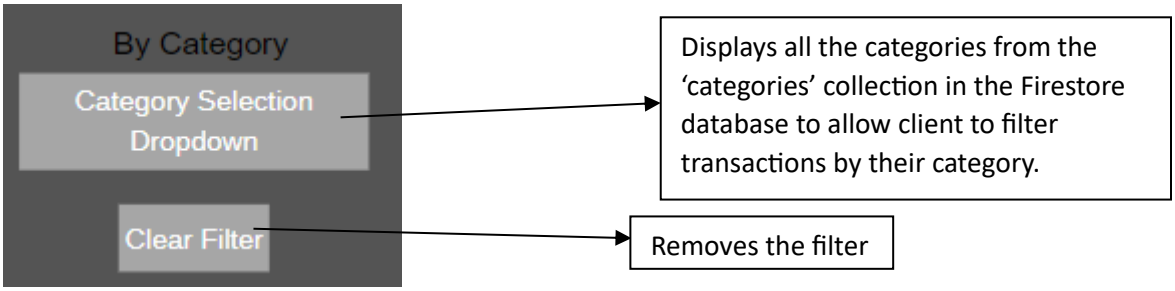
DateFilter



AmountFilter



CategoryFilter

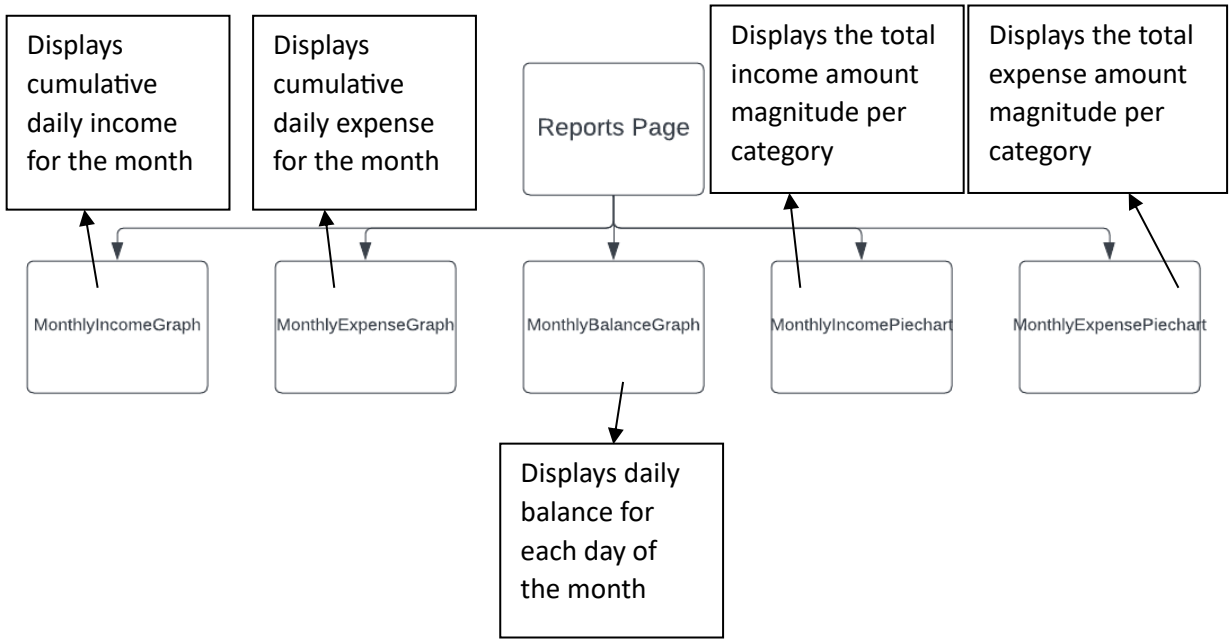


Transactions Filtering Pseudocode

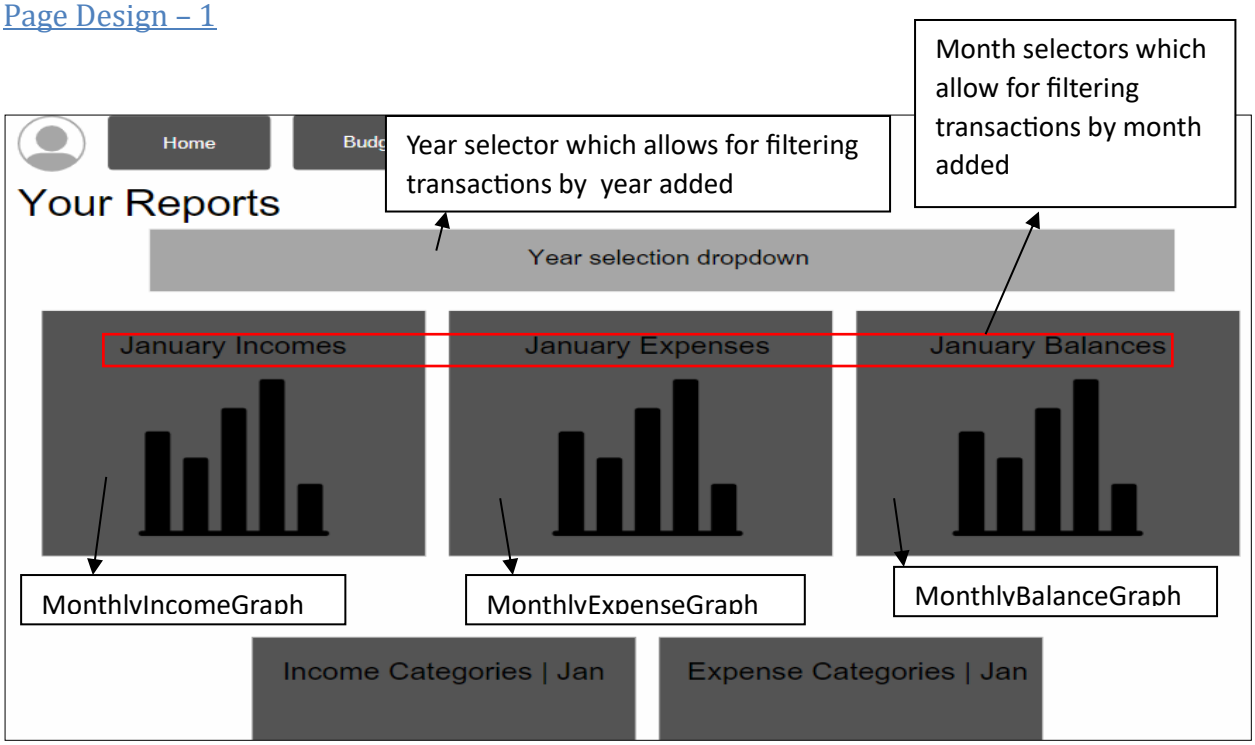
```
const query = transactionsQuery(clientAccountID, transactionType, startDate, endDate, minAmount, maxAmount, category)


firestoreDatabaseQuery(query) // Applies the client's transaction filters on the Firestore database.


subprogram transactionsQuery(clientAccountID, transactionType, startDate, endDate, minAmount, maxAmount, category) (
  TransactionFilter(transactionType)  // Apply the transaction type filter (income/expenses)
  dateFilters(startDate, endDate)  // Apply date filters
  amountFilters(minAmount, maxAmount)  // Apply amount filter
  categoryFilter(category)  // Apply category filter
)
```

Reports Page – Design, Structure, and Flowcharts

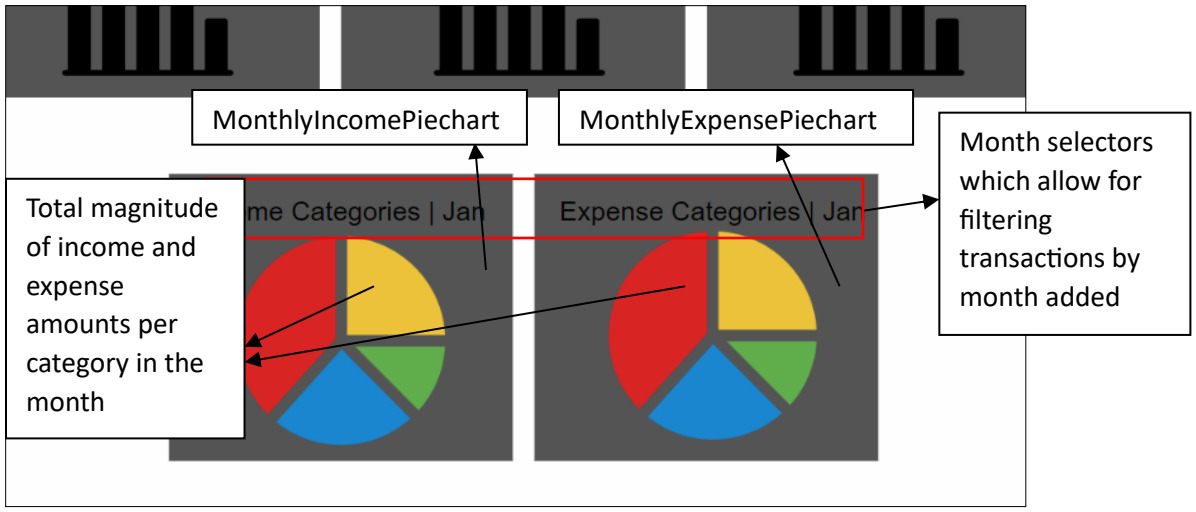
Structure



Page Design – 1

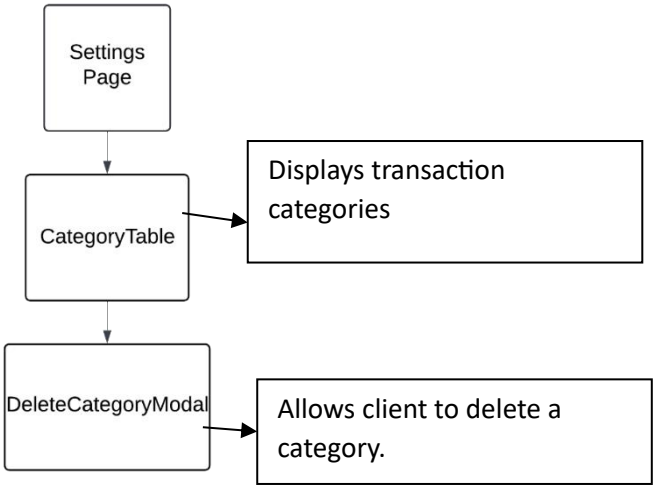


Page Design – 2

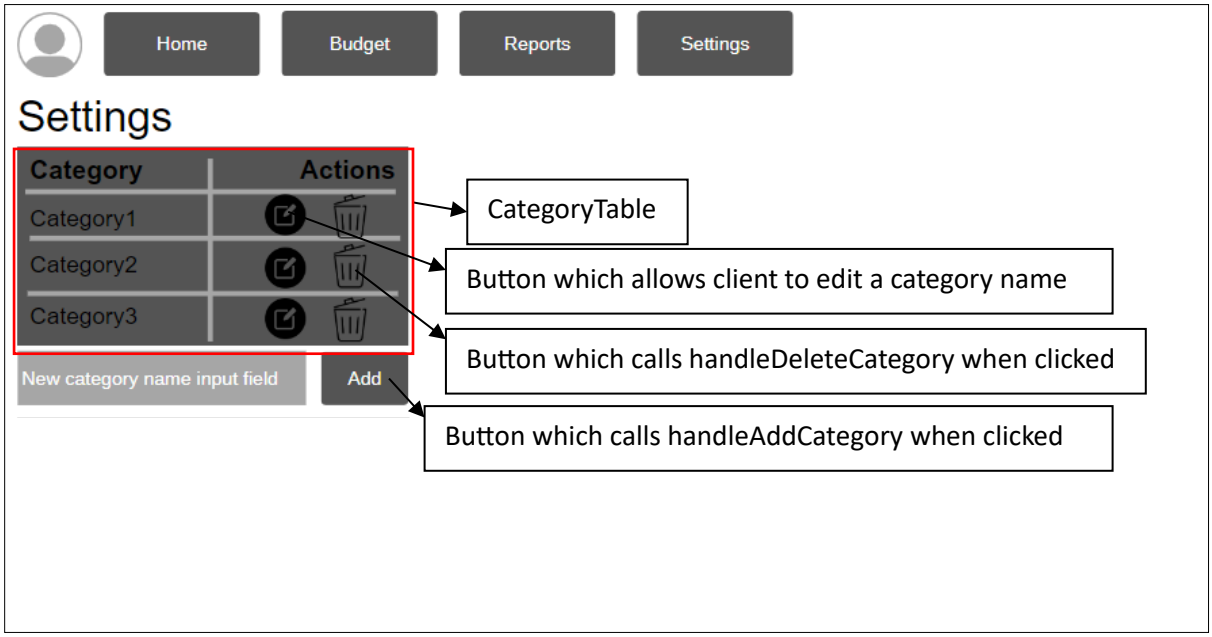


Settings Page – Design, Structure, and Flowcharts

Structure

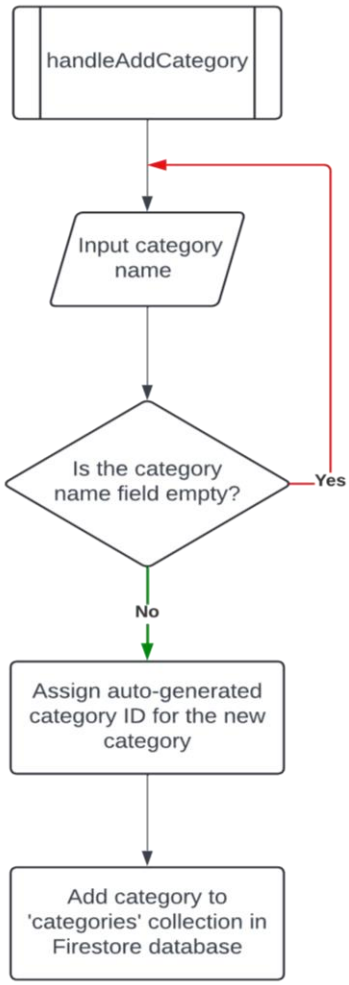


Page Design



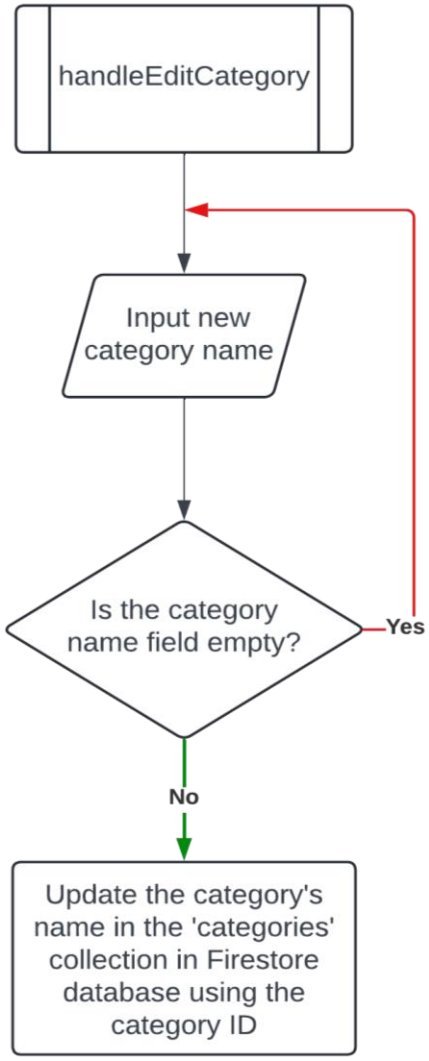
Page Design – Editing Category

handleAddCategory Flowchart and Pseudocode



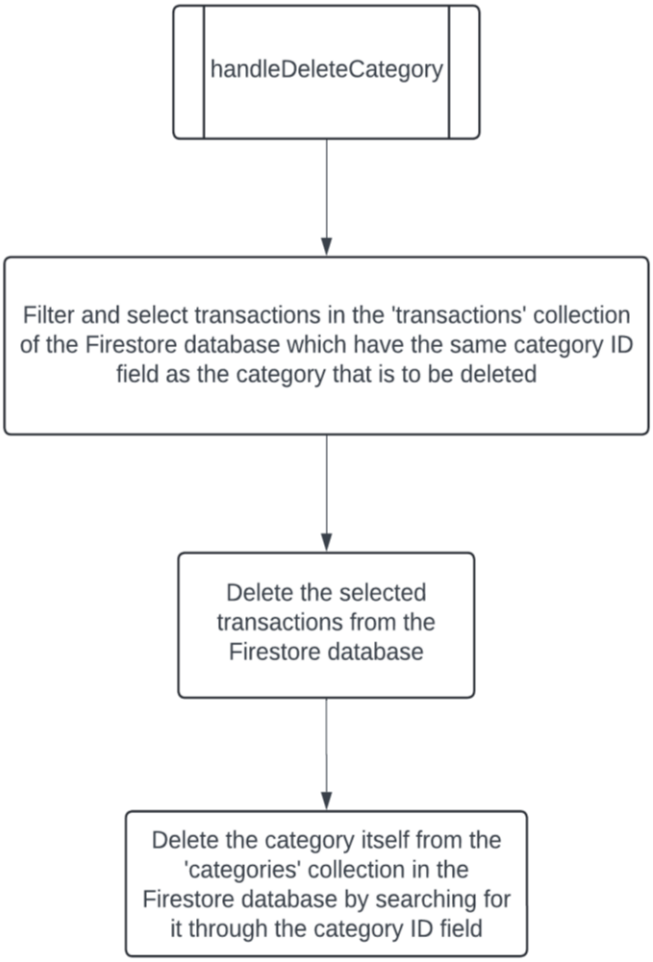

```
subprogram handleAddCategory(
  categoryName = input("Enter category name")  // Client inputs category name

  if categoryName = "" then  // Check if the category name field is empty
    cancel()  // Terminate category creation process
    alert("Category name cannot be empty.")  // Inform client that the category name is required.

  else
    categoryID = generateCategoryID()  // Assign auto-generated category ID for the new category
    addToFirestore(categoryID, categoryName)
   // Add category to 'categories' collection in Firestore database.
  end if
)
```

handleEditCategory Flowchart and Pseudocode




```
subprogram handleEditCategory(
  newCategoryName = input()  // Input new category name

  if newCategoryName = "" then  // Check if the category name field is empty
    cancel()  // Terminate category editing process
    alert("Category name cannot be empty.")  // Inform client that a category name is required.

  else
    updateCategoryInFirestore(categoryID, newCategoryName)
    // Update category name in the 'categories' collection in Firestore using the category ID.
  end if
)
```

handleDeleteCategory Flowchart and Pseudocode




```
subprogram handleDeleteCategory(

  transactionsToBeDeleted = filterTransactionsByCategory(categoryID)
  // Filter and select transactions in the 'transactions' collection of the Firestore database which have the same category ID field as the category that is to be deleted.


  deleteInFirestore(transactionsToBeDeleted)
  // Delete the selected transactions from the Firestore database.


  deleteCategoryInFirestore(categoryID)
  // Delete the category itself from the 'categories' collection in the Firestore database by searching for it through the category ID field.

)
```

Bibliography

Canva. *Canva Homepage*. n.d. 25 September 2023. <https://www.canva.com/>.

Test Plan

Success Criteria Tested	Action to be tested	Test method	Expected Result
Success Criterion 1: The client must be able to switch between his personal and work Gmail accounts without losing any of his financial data. Each account's data should be isolated, ensuring that only the transactions and categories related to the signed-in account are accessible.	Ability to sign in and out using different Gmail accounts	Signing into the application using the SignIn page using different Gmail accounts and then signing out	Clicking on the 'Sign in with Google' button opens up the Google sign-in popup. After signing in successfully, the profile picture of the signed-in Gmail account is shown in the left side of the Header component at the top of the application. Clicking on this opens the SignOutModal component which displays their current Gmail address. Clicking on the 'Sign Out' button in the aforementioned component signs the user out and redirects them to the SignIn page
	Effect of closing Gmail sign-in popup before singing in	Clicking on the "Sign in With Google" button and closing the popup that appear afterwards before signing in	Application does redirect client to the Home page. An error message is displayed to the client, which should prompt him to try again
	Effect of inputting an incorrect password and then closing the Gmail sign-in popup	Clicking on the "Sign in With Google" button and then selecting an account to sign in with. After inputting an incorrect password, the popup is then closed	Application does redirect client to the Home page. An error message is displayed to the client, which should prompt him to try again
	Testing if the transactions and transaction categories made in different Gmail accounts are visible in	Signing into the application using a new Gmail account (other than the test Gmail account which contains some transactions and	When the user is signed into a new Gmail account, they should not be able to see or access the transactions and categories that

	other Gmail accounts as well	transaction categories which are visible in the Firestore database)	were made in different Gmail accounts in any part of the application whatsoever. This should be achieved through filtering the Firestore data using the Firebase-provided userId value before showing the data to the user
Success Criterion 2: The application must ensure that sensitive financial data is only accessible when the client is logged in using one of their Gmail accounts. Attempts to access the application without logging in must redirect the user to the sign in page.	Accessing the application without signing into a Gmail account	Navigating to the different pages of the application by changing the URL after signing out	The application redirects the user to the SignIn page, thus preventing malicious actors from being able to access the financial data
	Trying to access transactions and categories created by other Gmail accounts	Signing into the application using a new Gmail account (other than the test Gmail account which contains some transactions and transaction categories which are visible in the Firestore database)	When the user is signed into a new Gmail account, they should not be able to see or access the transactions and categories that were made in different Gmail accounts in any part of the application whatsoever. This ensures that the client's kids cannot view his office finances and vice versa for his co-workers
Success Criterion 3: The client should be able to add new incomes and expenses on the Home page. The transactions must also be validated – transaction amounts cannot exceed 100,000 and transaction names cannot be longer than 20 characters long.	Addition of an income to the database with amount that falls under the 100,000 limit (normal value)	Creating a new income via the IncomeForm component and typing in 50,000 in the amount field	Able to see success/error message regarding the addition of the income. If addition was successful, the newly added income should be visible in the 'transactions' collection in the database as a Firestore document with the correct values in each field
	Addition of an expense to the database with	Creating a new expense via the ExpenseForm	Able to see success/error message

amount that falls under the 100,000 limit (normal value)	component and typing in 50,000 in the amount field	regarding the addition of the expense. If addition was successful, the newly added expense should be visible in the 'transactions' collection in the database as a Firestore document with the correct values in each field
Addition of an income to the database with amount 100,000 (extreme value)	Creating a new income via the IncomeForm component and typing in 100,000 in the amount field	Able to see success/error message regarding the addition of the income. If addition was successful, the newly added income should be visible in the 'transactions' collection in the database as a Firestore document with the correct values in each field
Addition of an expense to the database with amount 100,000 (extreme value)	Creating a new expense via the ExpenseForm component and typing in 100,000 in the amount field	Able to see success/error message regarding the addition of the expense. If addition was successful, the newly added expense should be visible in the 'transactions' collection in the database as a Firestore document with the correct values in each field
Addition of an income to the database with amount greater than 100,000 (abnormal value)	Creating a new income via the IncomeForm component and typing in 2,00,000 in the amount field	Able to see error message informing the user that the amount specified must be under 100,000
Addition of an expense to the database with amount greater than 100,000 (abnormal value)	Creating a new expense via the ExpenseForm component and typing in 2,00,000 in the amount field	Able to see error message informing the user that the amount specified must be under 100,000

	Adding a transaction name that is less than 20 characters in length (normal value)	Creating a new income and expense via the IncomeForm and ExpenseForm components and typing in a 15-character name for the transaction	Transaction name should be permitted and, if successfully added to the 'transactions' collection of the database, should be visible in the 'transactionName' field of the transaction's Firestore document
	Adding a transaction name that is 20 characters in length (extreme value)	Creating a new income and expense via the IncomeForm and ExpenseForm components and typing in a 20-character name for the transaction	Transaction name should be permitted and, if successfully added to the 'transactions' collection of the database, should be visible in the 'transactionName' field of the transaction's Firestore document
	Adding a transaction name that is longer than 20 characters in length (abnormal value)	Creating a new income and expense via the IncomeForm and ExpenseForm components and typing in a 25-character name for the transaction	Error message shown to client alerting that transaction names should be less than or equal to 20 characters long
	Effect of leaving a field's value as blank when creating a new income (abnormal value)	Creating a new income but leaving the transaction name field blank. Clicking on the "Add Income" button. Steps to be repeated for each field in the IncomeForm component	Able to see warning message informing the user that all transaction fields must have a value
	Effect of leaving a field's value as blank when creating a new expense (abnormal value)	Creating a new expense but leaving the transaction name field blank. Clicking on the "Add Expense" button. Steps to be repeated for each field in the ExpenseForm component	Able to see warning message informing the user that all transaction fields must have a value
	Effect of non-numerical characters in the	Inputting non-numerical characters in the	Non-numerical text not being added in the

	transaction amount field in the IncomeForm and ExpenseForm components (abnormal value)	transaction amount field in the IncomeForm and ExpenseForm components	transaction amount field
Success Criterion 4: The client must be able to view five of his recently made transactions in the Home page.	Effect of clicking on the 'Show Transactions' button in the Home page	Clicking on the 'Show Transactions' button at the bottom of the Home page	The RecentTransactionsTable component should come into view, the 'Show Transactions' button should now display 'Hide Transactions', and the user's view window should automatically scroll downwards to keep the RecentTransactionsTable closer to the center of the screen
	Effect of clicking on the 'Hide Transactions' button in the Home page when the RecentTransactionsTable is visible	Clicking on the 'Hide Transactions' button at the top of the RecentTransactionsTable	RecentTransactionsTable goes out of view, the 'Hide Transactions' button should display 'Show Transactions', and the user's view window should automatically scroll upwards to keep the rest of the components in the Home page at the center of the screen
	Testing the transaction limiting filter of the database query function in RecentTransactionsTable	Clicking on the 'Show Transactions' button at the bottom of the Home page when there are 4, 5, and 6 transactions in the database	The component must display all 4 transactions in the first case, then all 5 transactions in the second case, then only the 5 most recent transactions in the third case
	Effect of changing the dataAdded field in an old transaction so that it becomes the recent-most transaction and vice versa	Making 6 transactions in the database, opening the RecentTransactionsTable component, then changing the dateAdded fields of the oldest and	In the first opening of the component, the oldest transaction should not be visible while the newest transaction should be on the top. In the

		newest transactions so that they become the newest and oldest respectively then re-opening the RecentTransactionsTable	second opening, the previously oldest transaction should be on top and the previously newest transaction should not be displayed
Success Criterion 5: The client must have the ability to edit or delete existing transactions. Editing transactions will carry the same validation limitations (maximum amount being 100,000 and maximum transaction name length being 20 characters).	Deletion of transactions	Deleting a transaction through the DeleteTransactionsModal component	Able to see success/error message. If deletion was successful, the deleted transaction document should no longer be visible in the 'transactions' collection of the database. The deleted transaction must not appear in any components of the application
	Ability to edit the fields of existing transactions	Editing all the fields for a transaction with different normal data values through the EditTransactionsModal component	Able to see success/error message. If successful, the edited transaction document in the 'transactions' collection of the database should show the updated values for all the fields. The changes in the field values must be reflected in all components of the application
	Effect of leaving a field's value as blank when trying to edit a transaction (abnormal value)	Deleting the pre-filled value of one of a transaction's fields in EditTransactionsModal and clicking on the update button. Steps to be repeated for each field in the EditTransactionsModal component	Able to see warning message informing the user that all transaction fields must have a value and the transaction editing process not be executed
	Effect of inputting a date that doesn't follow the	Editing the date added field of a transaction via the EditTransactionsModel	Able to see warning message informing the user that the date added field of the

MM/DD/YYYY format (abnormal value)	to be in the DD/MM/YYYY format (where the number of the month is above 12 and/or the date is above 31) . Other transaction fields left unchanged	transaction must be in the MM/DD/YYYY format and the transaction editing process not be executed
Editing transaction amount to be less than 100,000 (normal value)	Editing an existing transaction's amount field to be 50,000 (if it is not already). Other transaction fields left unchanged	Able to see success/error message regarding the edit of the transaction amount. If the update was successful, the new transaction amount should be visible in the 'transactions' collection in the database as a Firestore document with the correct updated values in each field
Editing transaction amount to be equal to 100,000 (extreme value)	Editing an existing transaction's amount field to be 100,000 (if it is not already). Other transaction fields left unchanged	Able to see success/error message regarding the edit of the transaction amount. If the update was successful, the new transaction amount should be visible in the 'transactions' collection in the database as a Firestore document with the correct updated values in each field
Editing transaction amount to be greater than 100,000 (abnormal value)	Editing an existing transaction's amount field to be 2,00,000. Other transaction fields left unchanged	Able to see error message informing the user that the amount specified must be under 100,000 and the transaction editing process not be executed
Adding a transaction name that is less than 20 characters in length in	Editing an existing transaction's name to be 15-characters long. Other transaction fields left unchanged	The new transaction name should be permitted and, if successfully added to the 'transactions'

	EditTransactionsModal (normal value)		collection of the database, should be visible in the 'transactionName' field of the transaction's Firestore document
	Adding a transaction name that is 20 characters in length in EditTransactionsModal (extreme value)	Editing an existing transaction's name to be 20-characters long. Other transaction fields left unchanged	The new transaction name should be permitted and, if successfully added to the 'transactions' collection of the database, should be visible in the 'transactionName' field of the transaction's Firestore document
	Adding a transaction name that is longer than 20 characters in length in EditTransactionsModal (abnormal value)	Editing an existing transaction's name to be 25-characters long. Other transaction fields left unchanged	Error message shown to client alerting that transaction names should be less than or equal to 20 characters long and the transaction editing process not be executed
	Effect of non-numerical characters in the transaction amount field in the EditTransactionModal component (abnormal value)	Inputting non-numerical characters in the amount field in the EditTransactionsModal component	Non-numerical text not being displayed in the transaction amount field
	Effect of non-numerical characters in the date added field in the EditTransactionModal component (abnormal value)	Inputting non-numerical characters in the date added field in the EditTransactionsModal component	Non-numerical text not being displayed in the date added field
Success Criterion 6: In order to segregate transactions, the solution should have the functionality to create, edit, and delete transaction categories, and they	Creating a category with a name that less than 25 characters in length (normal value)	Creating a new transaction category via the CategoryTable component and typing in a 20-character name for the category	Category name should be permitted and, if successfully added to the 'categories' collection of the database, should be visible in the 'categoryName' field of

should be less than or equal to 25 characters in length.			the category's Firestore document
	Creating a category with a name that is 25 characters in length (extreme value)	Creating a new transaction category via the CategoryTable component and typing in a 25-character name for the category	Category name should be permitted and, if successfully added to the 'categories' collection of the database, should be visible in the 'categoryName' field of the category's Firestore document
	Creating a category with a name that longer than 25 characters in length (abnormal value)	Creating a new transaction category via the CategoryTable component and typing in a 30-character name for the category	An error message should be shown which alerts the client that the category name cannot be longer than 25 characters in length
	Effect of leaving the category name field blank when trying to add a category (abnormal value)	Not inputting any value in the category name input field in the CategoryTable component and clicking on the add button	Able to see warning message informing the user that the category name field must have a value and not be left blank
	Editing the name of an existing transaction category with a name that is less than 25 characters long (normal value)	Clicking on the edit button next to a category's name in the CategoryTable component and inputting a new name that is 20-characters long	Able to see success/error message. If successful, the edited category document in the 'categories' collection of the database should show the updated value for the categoryName field. The change in the field's value must be reflected in all components of the application
	Editing the name of an existing transaction category with a name that is 25 characters long (extreme value)	Clicking on the edit button next to a category's name in the CategoryTable component and inputting a new name that is 25-characters long	Able to see success/error message. If successful, the edited category document in the 'categories' collection of the database should show the updated value for the categoryName

			field. The change in the field's value must be reflected in all components of the application
	Editing the name of an existing transaction category with a name that longer than 25 characters long (abnormal value)	Clicking on the edit button next to a category's name in the CategoryTable component and inputting a new name that is 30-characters long	An error message should be shown to the client which alerts him that category names cannot be longer than 25 characters in length
	Effect of leaving the category name field blank when trying to edit a category's name (abnormal value)	Clicking on the edit button next to a category's name in the CategoryTable component and then not inputting any value in the category name input field and clicking on the tick button	Able to see warning message informing the user that the category name field must have a value and not be left blank
	Deleting transaction categories	Clicking on the trash-bin button next to a category's name in the CategoryTable component	Able to see the DeleteCategoriesModal component. Clicking on 'Delete' button in the aforementioned component should result in the user being able to see success/error message. If deletion was successful, the deleted category document should no longer be visible in the 'categories' collection of the database. The deleted category must not appear in any components of the application. Moreover, the transactions that were within the deleted category must not exist anywhere in the application or in the database as documents

			in the ‘transactions’ collection
Success Criterion 7: The client must be able to filter transactions by their type (income/expense), category, date added and the magnitude of the amount.	Ability to filter transactions by their type (income/expense)	Selecting either "Income" or "Expense" from the TransactionFilter component’s dropdown menu. No filters specified for the other filtering components	Only the transactions of the selected type (income/expense) are displayed in the Budget page
	Ability to filter transactions by category	Selecting a category from the CategoryFilter component’s dropdown menu. No filters specified for the other filtering components	Only the transactions belonging to the selected category should be displayed
	Ability to filter transactions by the date they were added	Selecting a start and end date within the current year using the DateFilter component. The end date should be on or after the start date. No filters specified for the other filtering components	Only the transactions within the selected date range should be displayed
	Ability to filter transactions by the transaction amount	Entering a minimum and maximum amount (normal values only) in the AmountFilter component and clicking on the “Apply Filter” button. Repeated by inserting a hyphen sign (-) in the input fields. No filters specified for the other filtering components	For positive values, incomes with an amount that falls within the specified range are displayed. For negative values, expenses with an amount that falls within the specified range are displayed
	Ability to remove filters and display all transactions	Clicking on the "Remove Filter" button in the CategoryFilter, DateFilter, and AmountFilter components. No filters specified for the other filtering components	All transactions should be displayed
	Ability to apply multiple filters simultaneously	Selecting valid values and options in multiple	Only the transactions that match all selected

	(type, category, date, amount)	filters: TransactionFilter, CategoryFilter, DateFilter, and AmountFilter components. No filters specified for the other filtering components	filters should be displayed
	Validation of the amount input in the AmountFilter fields using normal data	Inputting 50,000 and 70,000 in the AmountFilter fields for the minimum and maximum amount values respectively. No filters specified for the other filtering components	Transactions with an amount that falls within the specified amount range are displayed
	Validation of the amount input in the AmountFilter fields using extreme data	Inputting 0 and 100,000 in the AmountFilter fields for the minimum and maximum amount values respectively. No filters specified for the other filtering components	All transactions are shown
	Validation of the amount input in the AmountFilter fields using abnormal data	Attempting to input non-numerical characters and then 200,000 in either of the AmountFilter fields. No filters specified for the other filtering components	Non-numerical values not displayed in the input fields. Alert shown to client if either of the amount fields exceed 100,000 in magnitude. All transactions are shown
	Validation of the amount input in the AmountFilter fields when the maximum amount is lower than the minimum amount	Inputting 10,000 and 5,000 in the AmountFilter fields for the minimum and maximum amount values respectively. No filters specified for the other filtering components	Alert shown to client regarding this data validation error. All transactions are shown
	Selecting an “End Date” date that is before the “Start Date”	Selecting a “Start Date” as the current date and attempting to select yesterday’s date in the “End Date” field	This action is not permitted as the dates before the current date are “grayed out” (meaning they cannot be selected by the

			client). All transactions are shown
Success Criterion 8: The implemented solution should display monthly transaction data in graphs (daily incomes/expenses, daily balance, and monthly categorical earnings/spendings). This data must be filtered by month and year.	Effect of changing the selected month in each graph in the Reports page	Selecting various months through the dropdown located in the top of each graph component in the Reports page	Each graph must update with the new transactions to reflect the changes in the selected month
	Effect of changing the selected year in the Reports page	Selecting various years through the dropdown located at the top of the Reports page	All graphs must update with the new transactions to reflect the changes in the selected year
	Effect of hovering over each bar and section in the bar graphs and pie charts respectively	Hovering the mouse cursor over each bar and section in the bar graphs and pie charts respectively	The graphs must show a hover-box below the cursor to show more information regarding each bar/section (hovering over bars should show the magnitude of the daily income/expense or the daily balance) (hovering over sections should show the name of the category and the sum of its incomes/expenses)
Success Criterion 9: The application must be fully accessible and functional on both the client's MacBook and Windows desktops. The interface should maintain consistency across devices, ensuring that there is no loss of functionality between platforms.	Testing if and how the application appears in the client's Macbook laptop and Windows desktop	Opening the application in both the client's Macbook at home as well as his Windows desktop at the office	The application must function as expected in both operating systems. The app's components should be placed in similar positions/areas in both systems after accounting for the varying screen dimensions
	Adding, deleting and editing transactions in both of the client's computer systems	Adding, deleting and editing transactions using normal, extreme, and abnormal data in the client's Macbook and then performing the same transaction	The transaction operations must process as expected without any errors in both systems

		operations in his Windows desktop	
Success Criterion 10: The product should be able to handle data validation errors and provide feedback regarding the completion of database operations with clear success/warning/error messages.	Checking if application prevents submission when any transaction form fields are left empty	Attempting to submit the IncomeForm and ExpenseForm components with empty fields and clicking on the submit button	Client receives an error message (“Please fill out all the fields.”) and the form submission is cancelled
	Ensuring that transactions with amounts greater than 100,000 are rejected	Entering 200,000 (abnormal value) as the amount value in the IncomeForm and ExpenseForm components then clicking on the submit button	Error message "Transaction Amount should be less than or equal to 100,000" appears, and form submission is cancelled
	Verifying that the transaction name cannot exceed 20 characters	Entering transaction name 30 characters in length (abnormal value) in the IncomeForm and ExpenseForm components then clicking on the submit button	Error message "Transaction name should be less than or equal to 20 characters long" should be displayed, and the form submission should be cancelled
	Ensuring that transaction dates follow the "MM/DD/YYYY" format during transactions editing	Editing the date field for a transaction in the EditTransactionModal component and inputting a date in an incorrect format (DD/MM/YYYY) then clicking on the “Update” button	An error message alerts the user with "Date must be in MM/DD/YYYY format" and the edit does not proceed
	Ensuring that category names do not exceed 25 characters	Enter a category name 30 characters in length (abnormal value) in the CategoryTable and then clicking on the “Add” button	An error message alerts the client with "Category name cannot be longer than 25 characters," and the category should not be added
	Verifying that the user receives feedback when a transaction is successfully added to the database	Create an income or expense with normal data values and submit the form	A success message appears, confirming that the income or expense has been added to the database for the client

	Ensuring that the user is notified when a transaction is edited successfully	Edit a transaction in the EditTransactionModal using normal values and clicking on the "Update" button	A success message appears, and the updated transaction values should be reflected in the database
	Ensuring that the user receives error feedback when invalid data prevents a transaction from being updated	Edit a transaction with invalid data (e.g., name longer than 20 characters or amount greater than 100,000) in the EditTransactionModal and click the "Update" button	The client receives an error message explaining the issue relating to the specific field in which the abnormal value is present in (ex. "Transaction amount should be less than or equal to 100,000") and the transaction is not updated
Success Criterion 11: All changes (adding, editing, or deleting transactions and categories) must be updated in real-time across the client's devices and reflected immediately in the application. 11. All changes (adding, editing, or deleting transactions and categories) must be updated in real-time across the client's devices and reflected immediately in the application.	Adding a transaction is reflected in real-time across all devices	Add a transaction using the IncomeForm component on one device. Immediately check the same transaction on a different device	The newly added transaction should be visible in the transactions table on both devices without the need for a page refresh
	Editing a transaction is reflected in real-time across all devices	Edit an existing transaction using the EditTransactionModal component on one device. Immediately check the updated transaction on a different device	The edited transaction with updated values should be visible on both devices in the transactions table without requiring a page refresh
	Deleting a transaction is reflected in real-time across all devices	Delete a transaction using the DeleteTransactionsModal component on one device. Immediately check if the transaction has been removed from a different device	The deleted transaction should no longer appear in the transactions table on both devices without needing a page refresh
	Adding a category is reflected in real-time across all devices	Add a new transaction category using the CategoryTable component on one device. Immediately	The newly added category should appear in the categories dropdowns and lists across both devices without a page refresh

		check the category list on a different device	
	Editing a category is reflected in real-time across all devices	Edit an existing transaction category using the EditCategoryModal component on one device. Immediately check the category on a different device	The edited category name should be updated in the categories list on both devices without a page refresh
	Deleting a category is reflected in real-time across all devices	Delete a category using the DeleteCategoriesModal component on one device. Immediately check the category list on a different device	The deleted category should no longer appear in the categories dropdowns and lists across both devices without needing a page refresh. Any transactions associated with the deleted category should also no longer appear in any component across both devices
	Filtering transactions by category should update in real-time across devices	Apply a category filter using the CategoryFilter component on one device. Immediately check if the filter is applied consistently on a different device	The filtered list of transactions should be displayed correctly across both devices
	Filtering transactions by amount should update in real-time across devices	Apply an amount filter using the AmountFilter component on one device. Immediately check if the filtered results are reflected on a different device	The filtered list of transactions by amount should appear correctly on both devices without needing a page refresh
	Adding a large number of transactions should update in real-time across devices without delay	Add a batch of 50 transactions using the IncomeForm and ExpenseForm components on one device. Check for any delays or missing transactions on a different device	All added transactions should appear in real-time on both devices without any delay

Criterion C: Development

Techniques Used

1 Database Interactions

Given the client's requirement that the transactions be accessible in both of his primary devices (Criterion 9), local database solutions (like SQLite, localStorage) were ruled out. Therefore, the use of an online database was considered to be the most appropriate.

For this purpose, Firebase and Firestore (Firestore being a NoSQL database introduced and provided by Firebase) were used in the product as their services were used to effectively fulfil many of the criteria:

- User authentication and application security (Criterion 2) and signing in feature using Gmail accounts (Criterion 1)
- Adding, storing (Criterion 3) and managing (Criterion 5) transactions
- Storing and managing transaction categories (Criterion 6)
- Being able to access transactions on both of the client's primary devices (Criterion 9)

Firestore's file-based storage system (shown in Figure 1) facilitates complex querying and high storage scalability for free *(GeeksForGeeks)*, which enabled features such as generating graphs for the Reports page (Criterion 8).

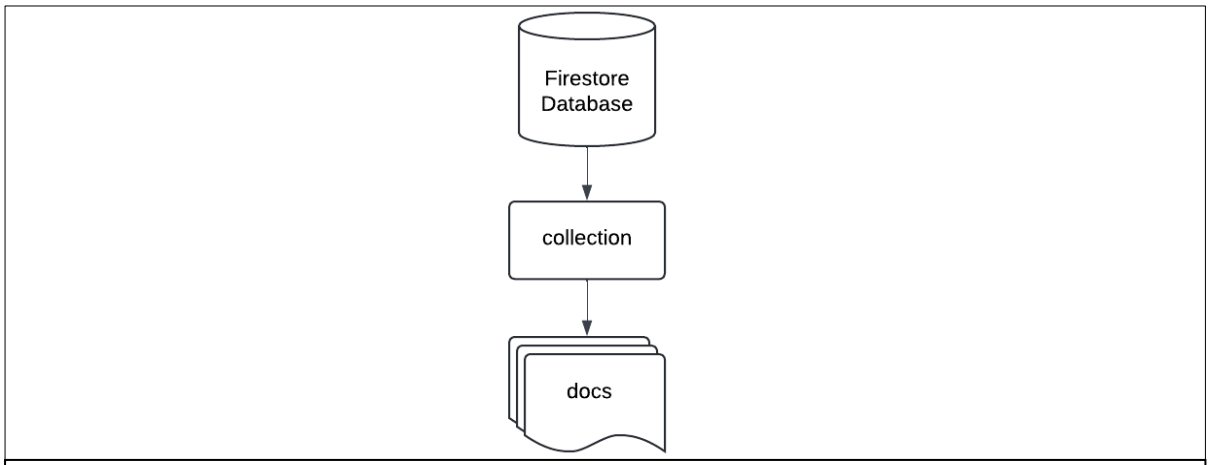


Figure 1: Firestore stores each transaction and category as individual 'docs'. These are organized into separate groups called 'collections' (as exemplified in Figure 6 below).

1.1 Firebase Connection

The product imports key Firebase and Firestore functions and services through the *firebase.js* configuration file (Figure 3). The functions and services are imported through the Firebase npm package *(Firebase)*. Firestore and Firebases' features that are used throughout the application are detailed in Figure 2 below:

auth	googleAuthProvider	db
Used for fulfilling Criteria 1 and 2 as it enables the product to know which Gmail account the client is currently signed in to (in order to display account-specific transactions) and allows application to determine whether the client is signed in or not, thereby preventing unauthorized access respectively.	Used in the sign-in page to allow the client to sign in with this Gmail accounts as per the requirement in Criterion 1 which states that the client must have the ability to manage separate budgets in the application using his two Gmail accounts.	Allows application to access the Firestore database – hence meeting Criteria 5, 6, and 9 as it would allow the app to store, update, and retrieve transactions and transactions categories in both of the client's devices.

Figure 2: Important functions imported from the Firebase npm package (Firebase) and/or defined in the firebase.js configuration file (Figure 3, Lines 3, 23, 26).

src > config > JS firebase.js > ...

```
1   // Importing the required functions from the Firebase npm package.
2   import { initializeApp } from "firebase/app";
3   import { getAuth, GoogleAuthProvider } from "firebase/auth"; // Used for authentication purposes.
4   import { getFirestore } from "firebase/firestore"; // Enables connection to Firestore database.
5
6   // The product's Firebase configuration details which allows Firebase to identify the product.
7   // The contents of the configuration details are hidden as anyone can access the main console otherwise.
8 > const firebaseConfig = {···
16  };
17
18  // Initializing application in Firebase
19  const app = initializeApp(firebaseConfig);
20
21  // Initializing Firebase's Google authentication provider.
22  export const auth = getAuth(app);
23  export const googleAuthProvider = new GoogleAuthProvider();
24
25  // Initializing the application's Firestore database.
26  export const db = getFirestore(app);
27
```

Figure 3: firebase.js configuration file.

1.2 Adding Transactions – IncomeForm Component

```
99          await addDoc(transactionsCollectionReference, {
100           transactionName: transactionName, // Name of transaction
101           amount: income, // The amount of income specified
102           categoryId: transactionCategory, // The category ID that the transaction belongs in
103           dateAdded: new Date(), // The date and time when the transaction was added
104           userId: userId, // User ID
105         });
```

Refers to the 'transactions' collection in the database.

Figure 4: Lines 99-105 from the IncomeForm.js component file.

There are two types of transactions that the client can add – incomes and expenses – which can be added through the IncomeForm and ExpenseForm components respectively. Both components directly fulfil Criterion 3 by enabling the client to add new incomes and expenses in the home page as shown in Figure 5. The *addDoc* function provided by Firestore *(Firebase)* is used for this function (Figure 4, Line 99) as it sends the to-be-created transaction doc's field values (Figure 4, Lines 100-104) from the application frontend to the Firestore database to be stored in the 'transactions' collection as a new doc (sample income doc shown below in Figure 6). Since the *addDoc* function is asynchronous, it provides extra functionality to the client, as elaborated upon in Section 2.

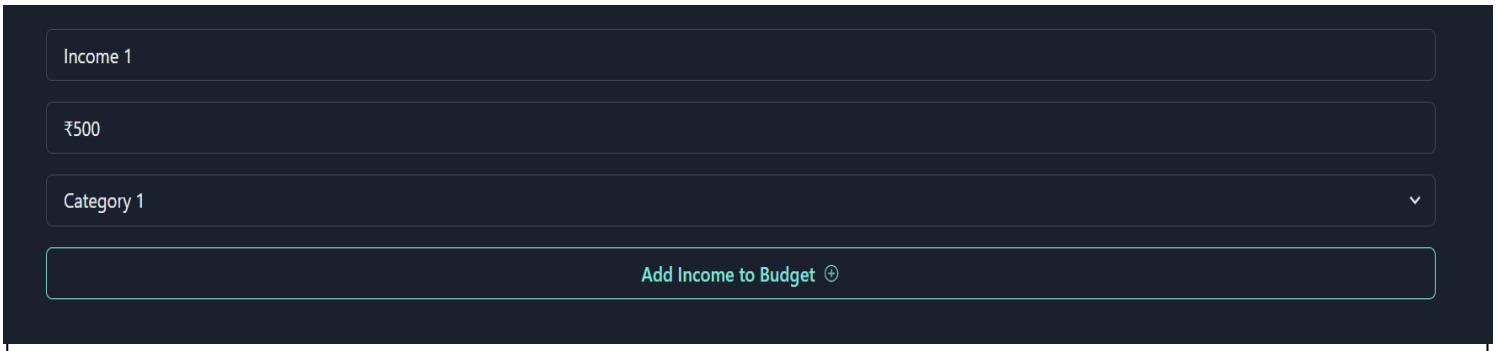


Figure 5: Screen capture of the IncomeForm component with filled-in sample transaction data.

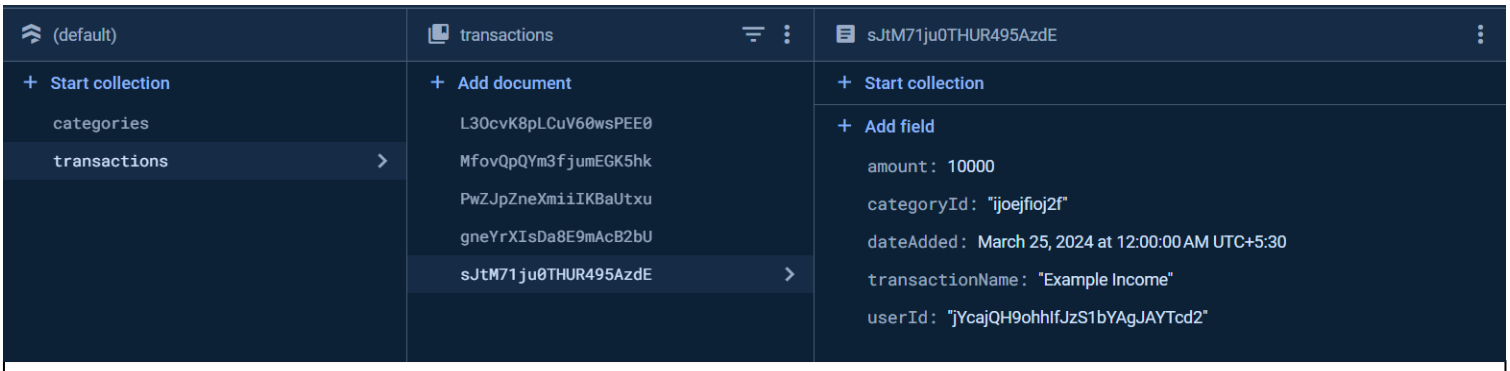


Figure 6: Screenshot of the structure of the Firestore database. The field values of a sample transaction doc from the 'transactions' collection are shown.

1.3 Fetching Transactions - RecentTransactionsTable Component

src > components > home > RecentTransactionsTable.js > RecentTransactionsTable

```
28  export default function RecentTransactionsTable() {
77    useEffect(() => {
78      setIsLoading(true); // Makes the RecentTransactionsTable component display a loading animation
79
80      const transactionsQuery = query(
81        collection(db, "transactions"), // Queries only the documents present in the 'transactions' collection
82        where("userId", "==", userID), // Only selects transactions that belong to the current user
83        orderBy("dateAdded", "desc"), // Sorts all the transactions so that the latest transactions are on the top
84        limit(5) // Only selects the recent 5 transactions
85      );
86
87      const unsubscribe = onSnapshot(
88        transactionsQuery,
89        (snapshot) => {
90          const transactionsWithCategoryNames = snapshot.docs.map((doc) => ({
91            ...doc.data(), // The field values in a transaction doc
92            categoryName: categoriesArray[doc.data().categoryId],
93            dateAdded: doc.data().dateAdded.toDate(), // Ensures that only the date is shown, not the time
94          })); // transactionsWithCategoryNames takes each transaction fetched by transactionsQuery and adds the
95          // category name it belongs to, using the `categoryId` field in the transaction docs.
96          setTransactions(transactionsWithCategoryNames); //
97          setIsLoading(false); // Removes loading animation
98        },
99        (error) => {
100         console.error("Error fetching transactions: ", error); // Displays error message in console for debugging
101         setIsLoading(false); // Removes loading animation
102       }
103     );
104
105     return () => unsubscribe(); // Cleanup function which helps ensure that the transactions are current
106   }, [userID, categoriesArray]); // Dependencies of the useEffect function which cause it to execute when they change
```

Figure 7: Lines 77-106 from the RecentTransactionsTable.js component file.

The RecentTransactionsTable component displays the five most recent transactions created by the client in the Home page, which was a problem for the client in his previous setup using Excel (Appendix A1), which also directly meets the requirements set in Criterion 4.

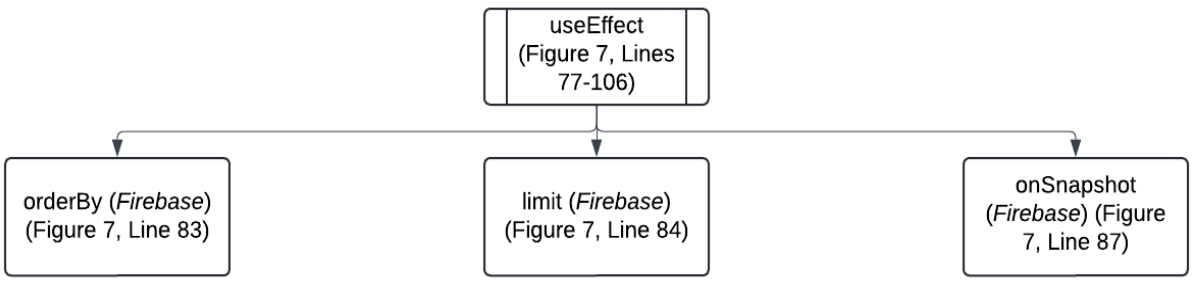


Figure 8: Process flowchart based on code in Figure 7. useEffect function allows for retrieval of transactions from the database to initiate as early as possible, displaying recent transactions faster and more effectively meet Criterion 4.

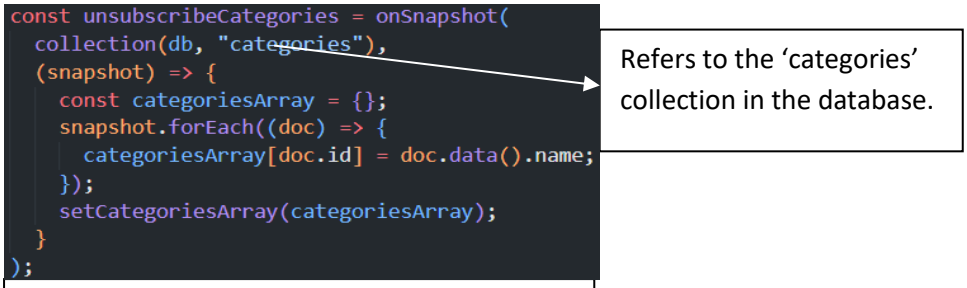


Figure 9: Lines 56-65 from the RecentTransactionsTable.js component file.

However, the RecentTransactionsTable component needs to obtain the category name associated to the categoryId of each transaction (since each transaction doc only stores the ID of the category to which it belongs (Figure 6)). It is not feasible for the client if he does not know the name of the category which each transaction belongs to. Hence, using the categoryId, the category's name is fetched from the categoriesArray (Figure 9, Lines 56-65) and displayed in the RecentTransactionsTable (Figure 10), thus contributing to the fulfilment of Criterion 4 since each transaction record can also display the name of the category that the transaction belongs to – hence providing the client more detailed information about his recent transactions.

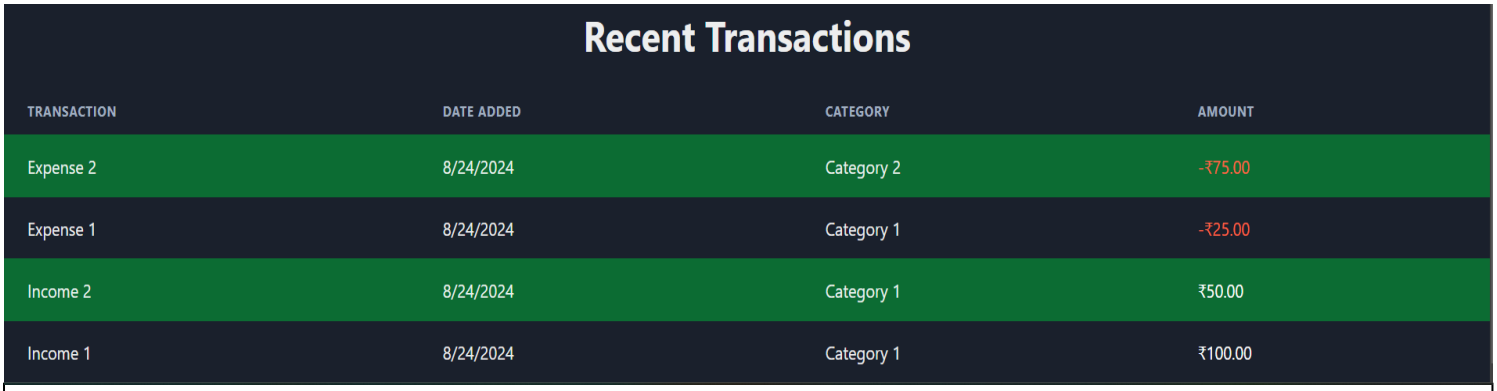


Figure 10: Screen capture of the RecentTransactionsTable component with sample transactions.

1.4 Deleting Categories - CategoryTable Component

```
163  const handleDeleteCategory = async (categoryId) => { // handleDeleteCategory requires the 'categoryId' of the category that is to be deleted
164    const batch = writeBatch(db);
165
166    const transactionsQuery = query(
167      collection(db, "transactions"),
168      where("categoryId", "==", categoryId)
169    );
170
171    const transactionsToBeDeleted0 = await getDocs(transactionsQuery); // The transactions which belong to the category are fetched
172
173    transactionsToBeDeleted0.forEach((transactionDoc) => {
174      const transactionsToBeDeleted1 = doc(db, "transactions", transactionDoc.id);
175      batch.delete(transactionsToBeDeleted1);
176    });
177
178    const categoryToBeDeleted = doc(db, "categories", categoryId); // The category itself is removed from the database
179    batch.delete(categoryToBeDeleted);
```

Figure 11: Lines 163-179 from the CategoryTable.js component file.

When a category is deleted, all the transactions within it should be deleted as well. For this purpose, Firestore's *writeBatch* function *(Firebase)* is used (Figure 11, Line 164). This function performs batched delete operations (Figure 11, Lines 175 and 179) to ensure data integrity, which is very important as it holds the client's vital financial data.

During the batched database operation processes mentioned previously, in the event that the client's system loses internet connection; experiences a failure; or there is an error in the database itself, the entire process is terminated – this ensures data integrity and hence reduces the chance of errors arising in the database.

Thus, through the *writeBatch* function, the CategoryTable component is able to fulfil an aspect of Criterion 6 – specifically that the client should be able to delete transaction categories. This enables the client to efficiently manage his transaction categories and, in extension, his transactions as well.

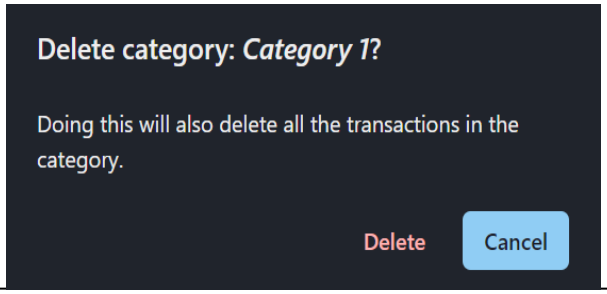


Figure 12: Screen capture of the DeleteTransactionsModal component, which calls the handleDeleteCategory function (Figure 10) in CategoryTable when client clicks on 'Delete'.

2 Exception Handling using Asynchronous JavaScript

```
93      try {
94  >     await addDoc(transactionsCollectionReference, {···
101     }); // addDoc function tries adding a new expense to the database
102 >   toast({···
110     }); // Success alert if new expense created successfully
111   } catch (error) {
112     // Stops the execution of the 'try' block in the event of an error in creating the expense
113     console.error("Error adding document: ", error); // Logs error into console for troubleshooting
114 >   toast({···
123     }); // Error alert if new expense creation was unsuccessful
124   }
125 };
```

Figure 13: Lines 93-125 from the ExpenseForm.js component file. Some functions are only partially shown for better readability.

By using asynchronous JavaScript and try-catch blocks, the application is able to manage errors that may arise during database or authentication operations in a graceful manner. This approach is important for fulfilling Criterion 10 as it ensures the client is informed of any issues that may arise during database/authentication operations when he creates, edits or deletes transactions or when he signs-in using one of his Gmail accounts, hence enhancing the client's accessibility and experience even during events such as network disruptions, system failure, or database/authentication errors.

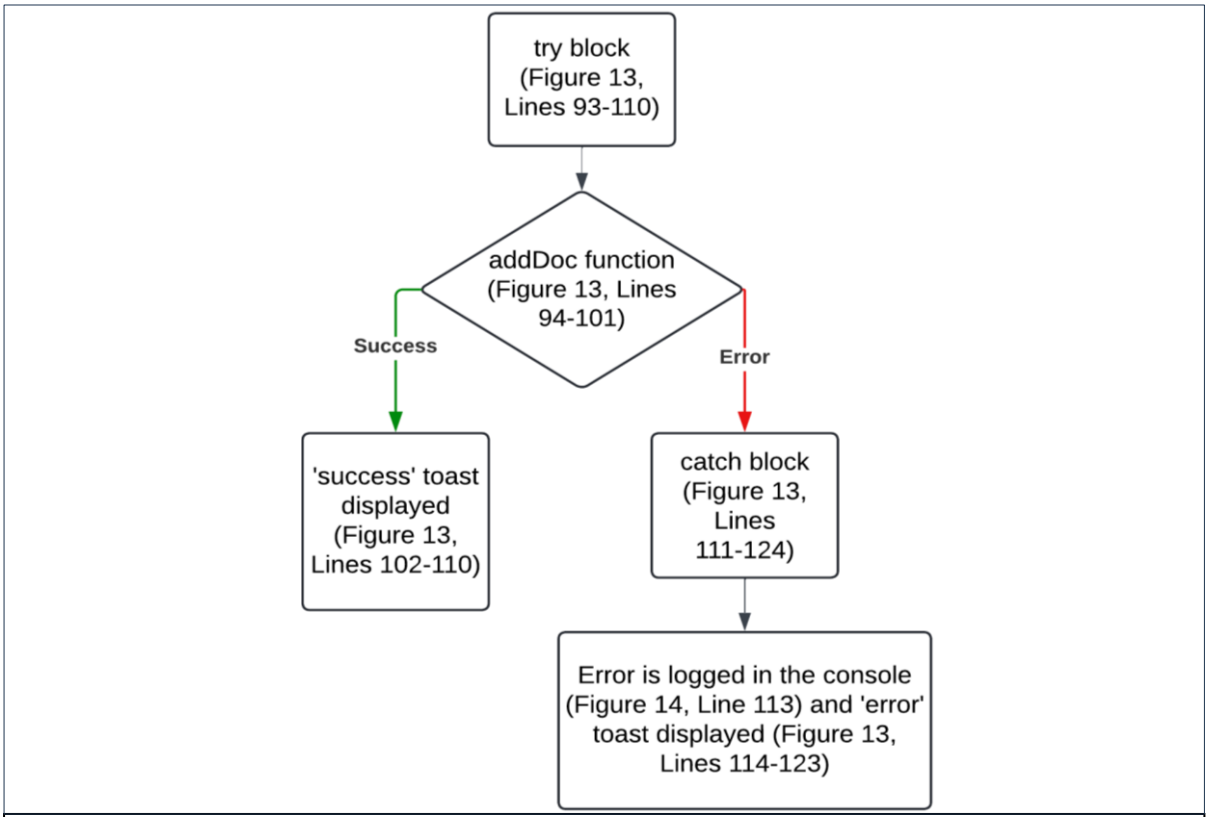


Figure 14: Process flowchart detailing the workings of the exception handling technique (based on the code in Figure 13).

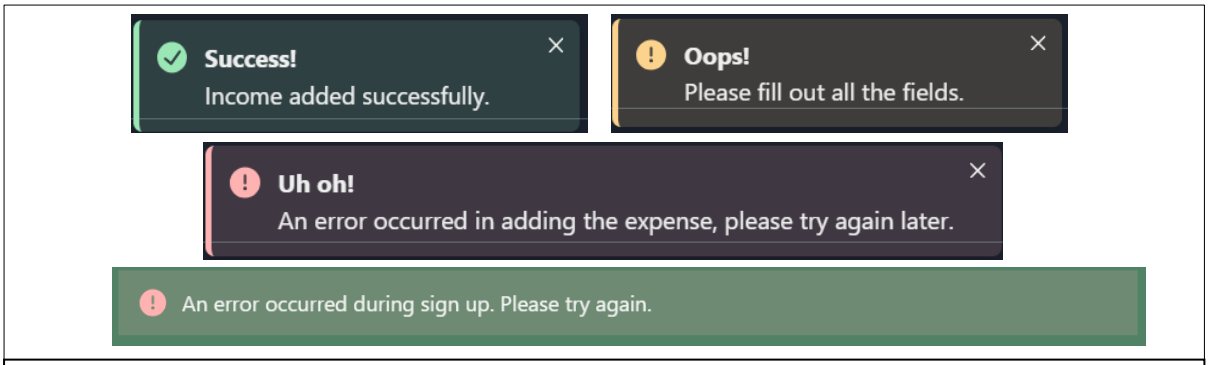


Figure 15: Screen captures of the ChakraUI (ChakraUI) "toast" components in the application: 'success' (top left), 'warning' (top right), and 'error' (middle, bottom) status alerts. They improve client experience as discussed above and fulfil Criterion 10.

3 Input Validation

```
if (updatedTransactionName.length > 20) {
  toast({
    title: "Oops!",
    description:
      "The transaction's name should not exceed 20 characters.",
    status: "warning",
    duration: 5000,
    isClosable: true,
    variant: "left-accent",
  });
  return;
}
```

Figure 16: Lines 100-111 from the EditTransactionsModal.js component file

To maintain data integrity within Firestore, input validation techniques are implemented across all transaction and transaction category creating and editing related components. This technique is essential for Criteria 3, 5, and 6, as it ensures that only valid, normal data is entered into the database docs, preventing potential errors or inconsistencies within the database that could disrupt the client's important finance management processes. Criterion 10 is also fulfilled as client is alerted

if a data validation error occurs (Figure 16, Lines 101-109).

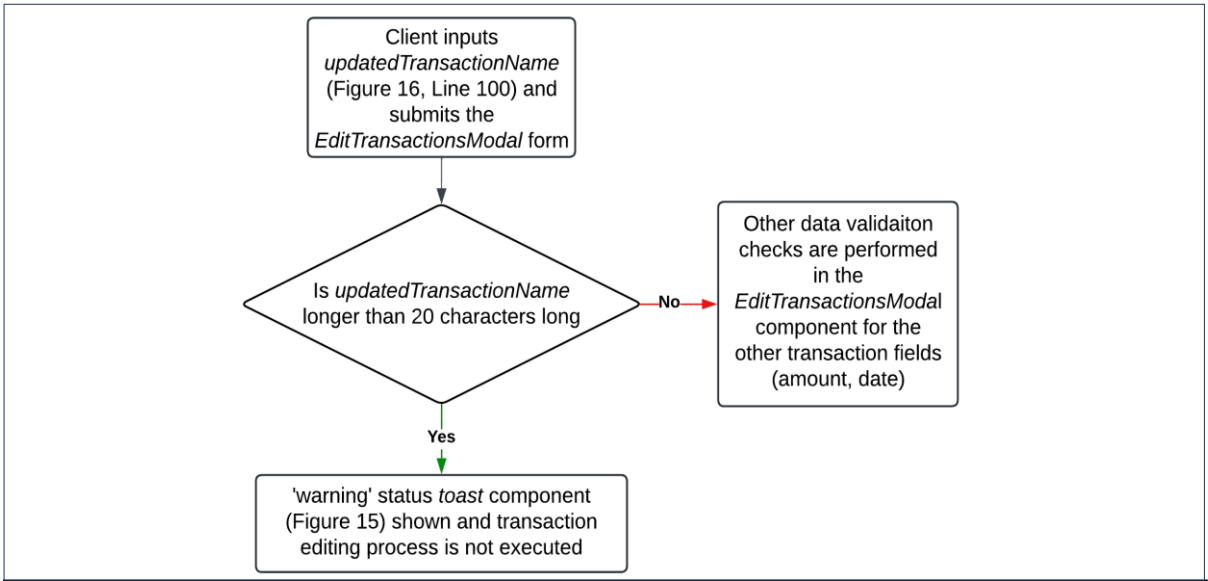


Figure 17: Process flowchart for the input validation of updated transaction names (based on the code in Figure 16).

4 Authentication and Verification

```
function googleSignIn() {
  signInWithPopup(auth, googleAuthProvider)
    .then(() => {
      setIsLoggedIn(true);
      navigate("/home");
    })
    .catch((error) => {
      console.error("Error during Google Sign In: ", error);
      setErrorMessage("An error occurred during sign up. Please try again.");
      setIsLoggedIn(false);
    });
}
```

Redirects the client to the Home page on successful sign in

Client is notified if there is an error and the details of the error are given in the console for troubleshooting.

Figure 18: Lines 26-37 from the SignIn.js page file.

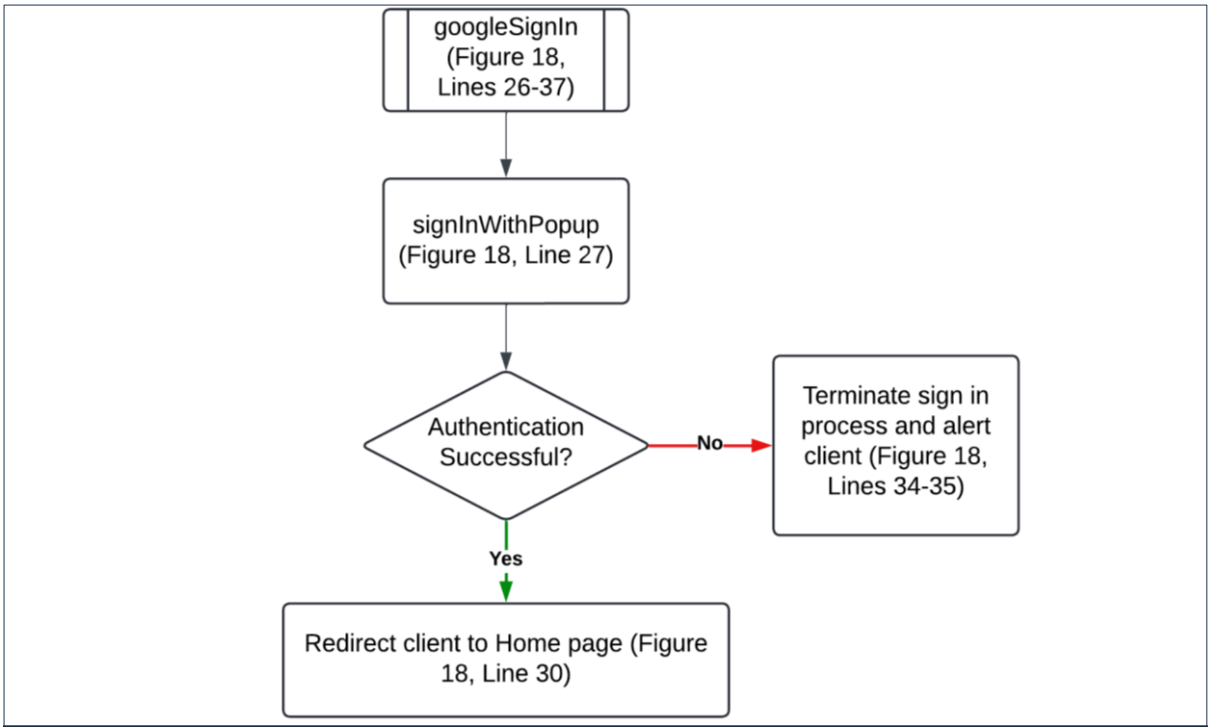


Figure 19: Process flowchart detailing the workings of the googleSignIn function (based on the code in Figure 18).

Since the *googleSignIn* function (Figures 18 and 19) displays the official authentication popup (Figure 19) which is displayed in both his Macbook and Windows systems, Mr. Hari can sign into his accounts with a single click on the screen (without having to input his password every time) – this feature is enabled only on his Macbook and Windows desktop as they are his frequently-used devices (however, this could be disabled on his Macbook as his children may gain unauthorized access to the application otherwise as Mr. Hari mentioned (Appendix A1)). This increases the conveniency with which the client can access the application while maintaining a high degree of security, thus further fulfilling Criterion 1, 2, and 9.

```
const unsubscribeAuth = onAuthStateChanged(auth, (user) => {
  if (!user) {
    // If user not signed in
    navigate("/");
  }
});
```

Figure 20: Lines 27-32 from the Reports.js page file.

Also, verification checks are done in each page through the Firebase *onAuthStateChanged* function *(Firebase)* (Figure 20, Lines 27-32) to enhance security by preventing unauthorized access to the application when the client is not signed into either one of his accounts – thus contributing to the fulfilment of Criterion 2.

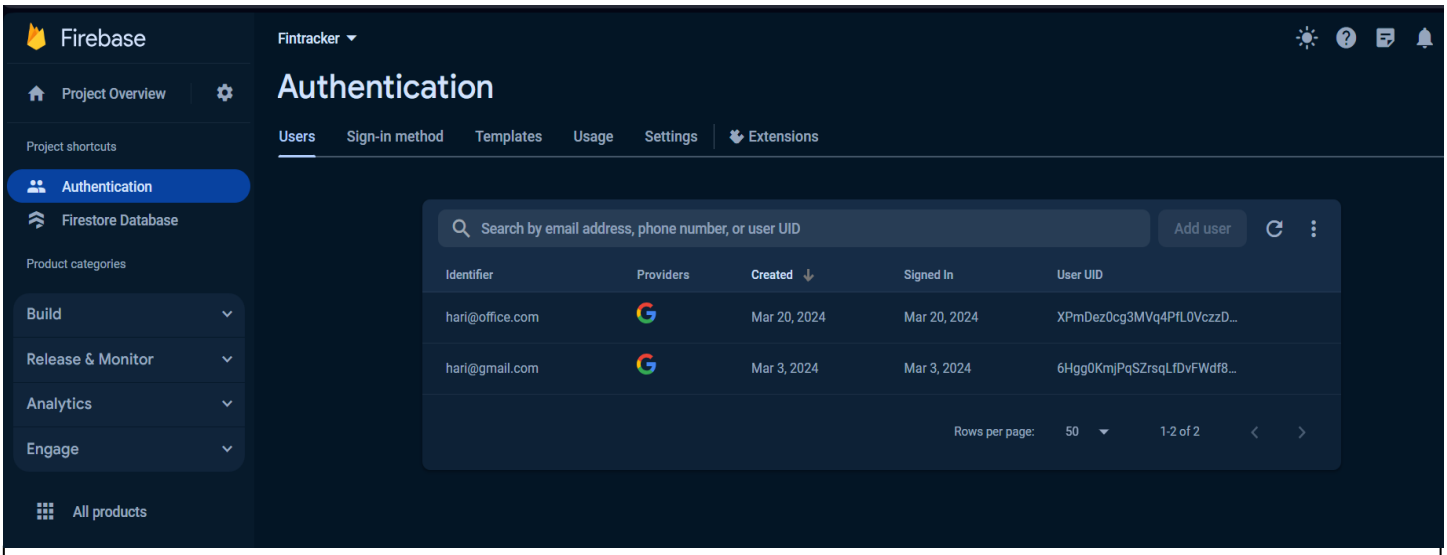


Figure 21: Firebase stores a list of registered users. The client's privacy is secured as third-parties cannot access the Firebase console.

5 Abstraction

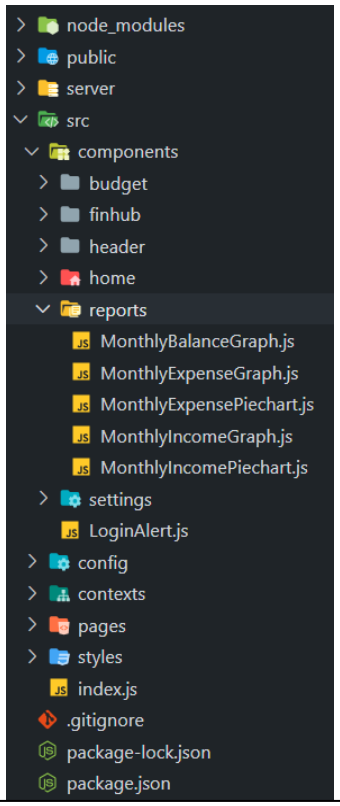


Figure 22: Hierarchal overview of the modular/component-based structure of the product. The graph and chart component files used in the Reports page are shown.

In the Reports page, each graph and piechart is rendered through a separate, dedicated component file instead of in the Reports page directly (Figure 22). The component takes in three parameters from the Reports page (Figure 23, Lines 89, 91, 93). One of them – the 'year' parameter – is set by the client

in the Reports page and is passed on to the MonthlyIncomeGraph component, wherein it is used in the database query (Figure 24, Lines 64-65) to allow the client to see his income reports for specific months within the selected year.

This modular approach allows the client to easily switch the month for each graph independently without having to interact with the entire page, thus simplifying the client interface and enhancing user experience and effectively meeting Criterion 8. Moreover, the maintainability, usability, and scalability of the graph components are improved, hence further contributing to meeting Criterion 8 by enabling easy modifications and updates to the graphical representations of financial data.

```
<MonthlyIncomeGraph
  shouldAnimate={shouldAnimate === "income"} // Variable used for animating the
  // - MonthlyIncomeGraph component if the user wanted to see his monthly incomes
  year={selectedYear} // Passes the 'selectedYear' value from this page
  // - as the 'year' value to the MonthlyIncomeGraph component
  flex={1} // Ensures that each graph component occupies an equal amount of space within the HStack component
/>
```

Figure 23: Lines 88-94 from the Reports.js page file showing the use of abstraction.

```
const startOfMonth = new Date(year, selectedMonth, 1);
const endOfMonth = new Date(year, selectedMonth + 1, 0);
const transactionsRef = query(
  collection(db, "transactions"),
  where("userId", "==", auth.currentUser.uid),
  where("dateAdded", ">=", startOfMonth),
  where("dateAdded", "<=", endOfMonth)
);
```

The aforementioned 'year' parameter that is passed on from the Reports page

Ensures that only the transactions which belong to the currently signed-in account are shown

These variables filter the transactions which have been created at the starting and ending of the selected month in the selected year

Figure 24: Lines 64-71 from the MonthlyIncomeGraph.js component file.

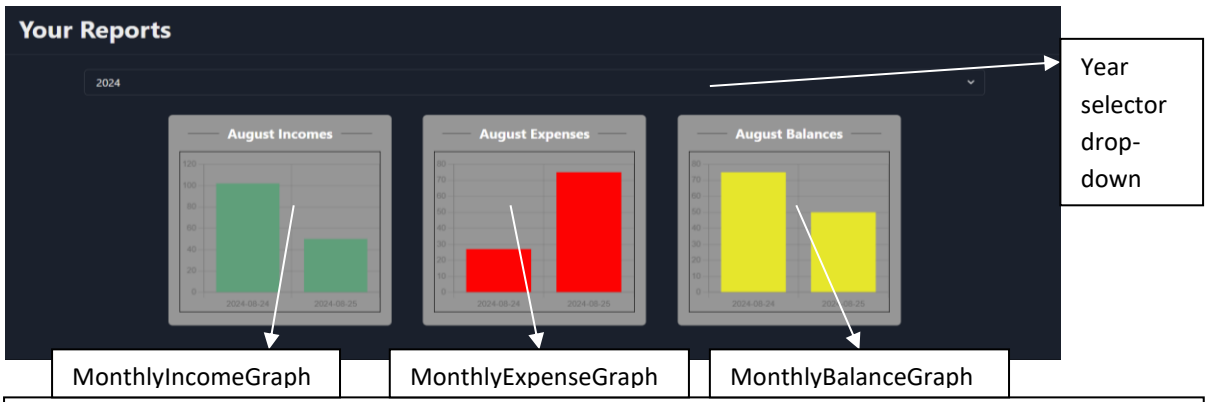


Figure 25: Partial screen capture of the Reports page. Abstraction was used in the aforementioned graph components in a similar manner as that of in MonthlyIncomeGraph as elaborated upon above.

6 Navigation

```jsx
<nav>
  <div className="pill-nav"> {/* Applies consistent styling for all buttons */}
    <NavLink to="/home" activeClassName="active">
      Home
    </NavLink>
    <NavLink to="/budget" activeClassName="active">
      Budget
    </NavLink>
    <NavLink to="/reports" activeClassName="active">
      Reports
    </NavLink>
    <NavLink to="/settings" activeClassName="active">
      Settings
    </NavLink>
  </div>
</nav>
```

Figure 26: Lines 60-75 from the Header.js component file.

Figure 27: Screen capture of Header component in the Home page (which is why the 'Home' button is darkened, improving client experience and application navigability)

```jsx
<Routes>
  <Route path="/" element={<SignIn />} />
  <Route path="home" element={<Home />} />
  <Route path="budget" element={<Budget />} />
  <Route path="reports" element={<Reports />} />
  <Route path="settings" element={<Settings />} />
  <Route path="*" element={<Home />} />
</Routes>
```

Figure 28: Lines 15-22 from the index.js main application file.

The use of the react-router-dom package's NavLink component *(Dorr and Strickland)* in the Header component (Figure 26, Lines 62-73) streamlines navigation across the application. This is because, when the client clicks on each button/link, the NavLink component redirects them to the requested page. For example, if the client clicks on "Budget", he is redirected to the "/budget" link (Figure 26, Line 65), which coresponds to the Budget page (Figure 28, Line 18) in application, thus redirecting him to that page.

This contributes to the overall fulfilment of Criteria 3, 4, 6, 7, and 8 as this technique ensures that the client can easily switch between performing his finance management-related tasks, such as adding transactions, filtering and finding specific transactions, viewing graph reports, or managing transaction categories.

Moreover, when the client clicks on the profile picture icon on the far-left in the Header (Figure 27), he can sign out of his current Gmail account and switch to his other account in the SignIn page, thus

helping fulfil Criterion 1 (which states that he should be able to switch between his personal and professional Gmail accounts).

Word count: 1102

7 Bibliography

ChakraUI. *ChakraUI*. n.d. 10 October 2023. <https://v2.chakra-ui.com/>.

Dorr, Tim and Chance Strickland. *react-router-dom*. Vers. 6.22.3. n.d. 24 October 2023. <https://www.npmjs.com/package/react-router-dom>.

Firebase. *firebase*. Vers. 10.9.0. n.d. 5 October 2023. <https://www.npmjs.com/package/firebase>.

GeeksForGeeks. *Firestore and its advantages*. 19 February 2021. 6 October 2023. <https://www.geeksforgeeks.org/firestore-and-its-advantages/>.

Criterion E: Evaluation

Product Evaluation

An interview was held with the client on 14/11/2023 for gathering his feedback regarding the application. The interview transcript has been included in Appendix E1. Please refer to Appendix E1 for further details.

Success Criteria	Met/Not Met	Evaluation (based on Client Interview in Appendix E1)
The client must be able to switch between his personal and work Gmail accounts without losing any of his financial data. Each account's data should be isolated, ensuring that only the transactions and categories related to the signed-in account are accessible.	Met	The client talked about and praised the ease of use of signing in and using his different Gmail accounts in the product and had no complaints regarding his financial data being mixed between his two accounts.
The application must ensure that sensitive financial data is only accessible when the client is logged in using one of their Gmail accounts. Attempts to access the application without logging in must redirect the user to the sign in page.	Met	The client said he was reassured about the security of the application as it redirected him to the sign-in page when he had forgotten to sign in.
The client should be able to add new incomes and expenses on the Home page. The transactions must also be validated – transaction amounts cannot exceed 100,000 and transaction names cannot be longer than 20 characters long.	Met	The client mentioned and commended the simplicity of adding incomes and expenses in the Home page.
The client must be able to view five of his recently made transactions in the Home page.	Met	The client was said he was contended with the ability to view five of his recent transactions in the home page.

The client must have the ability to edit or delete existing transactions. Editing transactions will carry the same validation limitations (maximum amount being 100,000 and maximum transaction name length being 20 characters).	Met	The client had stated his satisfaction with the transactions editing and deleting features.
In order to segregate transactions, the solution should have the functionality to create, edit, and delete transaction categories, and they should be less than or equal to 25 characters in length.	Met	The client said he was pleased with the transaction categories feature and found it to be time-saving compared to his previous setup.
The client must be able to filter transactions by their type (income/expense), category, date added and the magnitude of the amount.	Met	The client said he had positive feedback regarding the filtering of transactions.
The implemented solution should display monthly transaction data in graphs (daily incomes/expenses, daily balance, and monthly categorical earnings/spendings). This data must be filtered by month and year.	Met	The client said he found all the graphs and pie-charts to be useful for having a holistic view of his financial health.
The application must be fully accessible and functional on both the client's MacBook and Windows desktops. The interface should maintain consistency across devices, ensuring that there is no loss of functionality between platforms.	Met	The client mentioned and liked the flexibility of how he can use the application in both his Windows and Macbook computer systems.
The product should be able to handle data validation errors and provide feedback regarding the completion of database	Met	Mr. Hari expressed his appreciation for the status messages used for the alerting him regarding the status of the various functions of

operations with clear success/warning/error messages.		the application.
All changes (adding, editing, or deleting transactions and categories) must be updated in real-time across the client's devices and reflected immediately in the application.	Met	The client had said that he appreciated this feature.

The product has met client expectations (Appendix E1) through solving the problem identified in the initial interview (Appendix A1) by fulfilling the success criteria. However, a feature that could have been implemented differently in my opinion would be being able to create new categories in the home page using a modal, which would've increased convenience for the client as he wouldn't have to go to a new page to do so.

Recommendations for Further Development

Mr. Hari found the product satisfactory. However, given below are some recommendations for further development that I have come up with based on his feedback and along with my personal suggestions as well.

1. **Multiple categories:** The client expressed his wish for the ability to assign multiple categories to each transaction (one transaction can be assigned multiple categories instead of just one) to enhance transactions organization (Appendix E1). Implementing this requires adjusting the database schema to support many-to-many relationships between transactions and categories instead of the current one-to-many relationship system. This would lead to a more effective categorization system, allowing for more granular transactions filtering and categorization.

2. **Recurring transactions:** I would like to allow the client to set up recurring incomes/expenses (ex. salary, rent) that are automatically added to the 'transactions' collection in the database at the specified time periods. Implementing this would involve creating Firestore Cloud Functions to periodically create these transactions at the specified intervals. I believe this would be beneficial as it reduces the time spent in creating transactions (since Mr. Hari would not have to repeatedly make transactions that are made regularly), hence allowing for more convenient financial management.

Word count: 291

A1 Client Interview Transcript:

Date: 19/09/2023

Mode of Interview: Physical

Interviewer: Hi Mr. Hari, thank you for taking the time to meet with me today as we had discussed earlier. I would like to discuss with you to know more about your nature of work and the issues that you are facing.

Client: Hi, it's a pleasure to discuss with you.

Interviewer: Could you give a background of what responsibilities you have at work?

Client: Sure. **I'm a Regional Manager at Store XXX.** I specifically manage the revenue and costs for the stores in this district. Although I have some staff working under me in the finance department, I prefer handling the transactions by myself.

Interviewer: Ok sir. What are the responsibilities you have at home?

Client: Since I'm the breadwinner of the family, I manage our budgeting and finances. **I have a lot of transactions to handle for my family as well**, my kids' school fees, rent, groceries, loans, and more.

Interviewer: Have you faced any issues regarding finance management for your office and home?

Client: Yes, I've been having issues with my initial and current method of financial management.

Interviewer: Ok sir, could you give me some information about your initial financial management system?

Client: Definitely. I used to manage my finances using a combination of **notebooks and paper spreadsheets**. One of the major challenges I faced was the **sheer number of transactions** I had to handle for both my work and family. It was **difficult to keep everything organized and to get a holistic view of my financial health.** Also, this system required a lot of **time, physical space, and energy.** I had to manually write down each income and expense, which is quite time-wasting and made my wrists hurt after a while. Sometimes, **I mixed up my work and personal finances or made some calculation mistakes**, like one day I put one of my office sales data into my personal finance notebook and became a millionaire for a day! What a headache that was! And as for **accessing existing transactions, it was frustrating to have to spend a lot of time take out the folders from my shelf and go through them one-by-one. Editing transactions could take me hours and drain my energy since I had to redo folders upon folders of transactions.** The only plus-point in using this method I can think of was that I could **easily check what recent transactions I made** in my shelf.

Interviewer: I'm sorry to hear that sir, your past situation must have been stressful for you. What is your current method in dealing with this issue?

Client: I'm **currently using a very large Excel file with multiple sheets** to record my transactions. I am now able to **very quickly and easily make transactions in a few clicks** without having to deal with numerous notebooks and such. But this system has its own list of disadvantages that make it a headache to use. For example, **I can only use it from my home Macbook** so if I have a new work-related transaction to add or calculations to do, I have to wait till I get home since I can't bring my MacBook to the office. And I'm afraid to give my kids my laptop since they may drop it and potentially **corrupt the data, making me lose it forever, or they may open the file and tamper with the data.** I also tried to categorize the transactions by

creating separate sheets for each category and putting the transactions in the respective sheet but I'm **not satisfied by this ad-hoc method at all since it's hard to find previous transactions to edit or delete them.**

Interviewer: It seems like your current situation also comes with its share of problems. In your daily life, what devices do you primarily use?

Client: Like I said, **at home I use my Macbook** and in the **office I use the company-provided Windows desktop.** But I would like to add that I'm **not very adept with technology**, for example, I'm struggling to make graphs in Excel to be able to see a report of my monthly finances. And also, **I use both a personal Gmail account and an office Gmail account regularly.**

Interviewer: I will keep that in mind. Thank you for your time, sir. This information will be invaluable for the planning of the application. I'll get started right away.

Client: Thanks. I'm excited to see the progress!

A2 Client Interview Transcript:

Date: 21/09/2023

Mode of Interview: Physical

Interviewer: Good afternoon, sir. Thank you for meeting with me again. I've developed a plan and I'd like to discuss it with you to make sure it aligns with your needs.

Client: Hi, I'm eager to hear what you've come up with.

Interviewer: Thank you. Based on our previous conversation, I'm planning to develop a **web application that will be accessible from both your Macbook and Windows desktop,** so you won't be restricted to only one device. Your financial data will be **securely stored in a free online database** that uses robust encryption to protect your finance information. Also, I'm going to integrate the **official Google authentication system** into the application, so you should be able to securely log in and out with your Gmail accounts so that you can manage your personal and office budgets separately.

Client: Yes, I'd like to ensure that my data is kept secure, especially considering that I have to manage many business transactions and so I wouldn't want that data to be leaked somehow!

Interviewer: Of course sir. The application will also have all the basic functionalities for your finance management purposes. You can **add incomes and expenses in the home page** and **view five of your recent most transactions at a click of a button** since you mentioned in our last meeting that you liked having access to your recent transactions in your previous pen-and-paper system.

Client: Yes, I recall mentioning that.

Interview: And in order to be able to manage your transactions effectively, **I will implement a category system.** Editing and deleting transactions will no longer involve manually re-calculating finances as the **application will update in real-time for any changes you make for your transactions.**

Client: That sounds great! I see that you've really planned things out.

Interviewer: Thanks. I'll also include a **reports feature that will generate graphs to visualize your monthly finance information for your incomes, expenses and balances. You'll also be able to monitor your monthly transactions category-wise.** I will also include a comprehensive transactions filtering feature that will allow you to **filter transactions by income or expenses, categories, date added, and amount** so that you do not have to spend a lot of time to find a transaction. Of course, each operation you perform like adding or editing transactions will be accompanied by an **appropriate status alert** to inform you if the operation was successful, if you've made a mistake in inputting details, or if there was an error.

Client: That's exactly what I wanted.

Interviewer: One last thing, are there any additional features or adjustments you'd like to request at this stage?

Client: Nothing comes to my mind at the moment. I'm happy with the solution you've proposed.

Interviewer: Great then! I'll proceed with development based on this plan. I'll keep you updated on my progress and we can make adjustments as needed.

Client: Thanks for your hard work on this. I'm looking forward to using the application!

Interviewer: Thank you, sir. I'm excited to develop this solution.

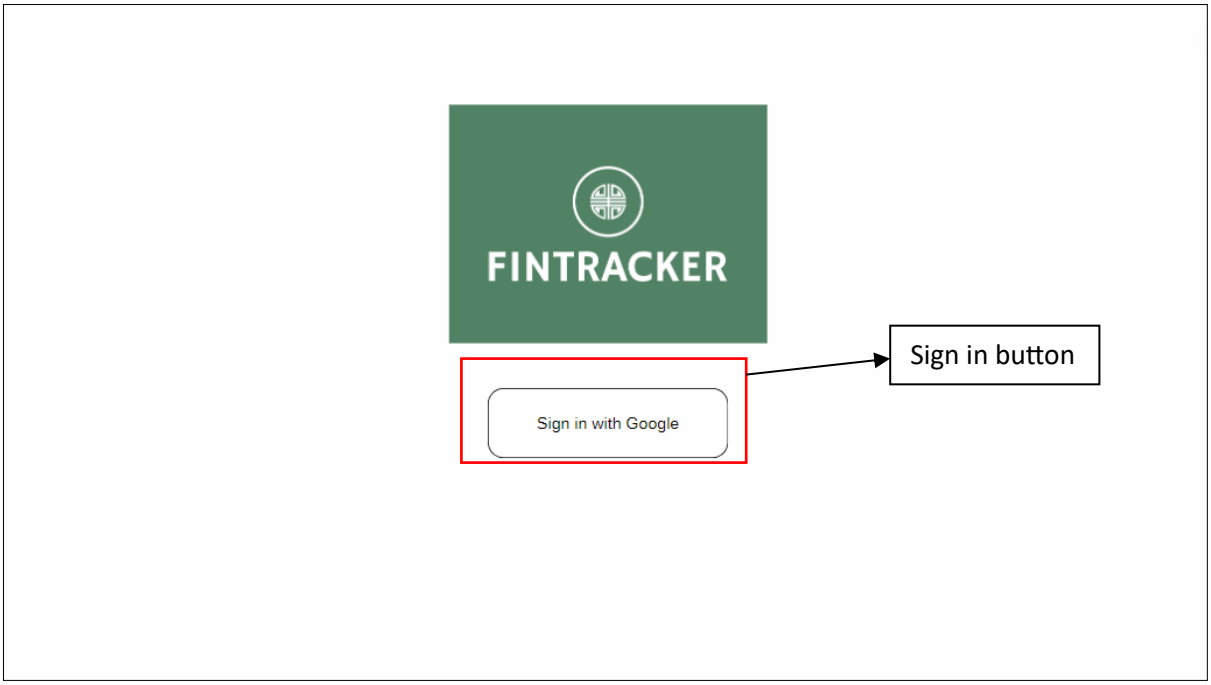
UI Appendix:

Date: 29/09/2023

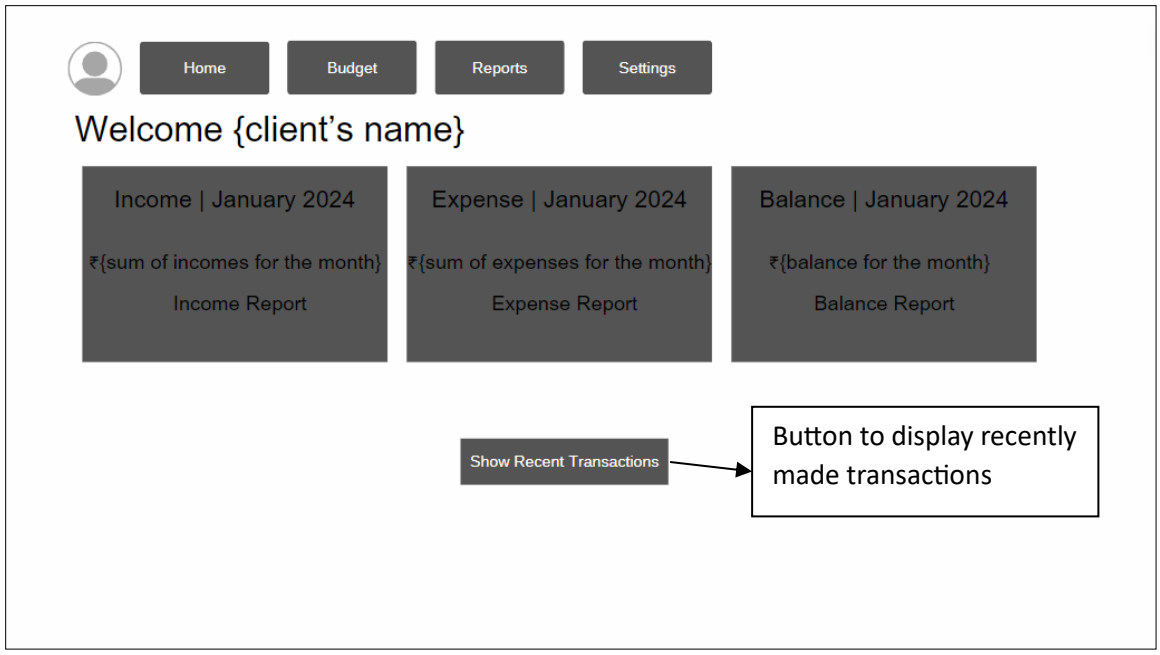
Mode of Interview: Physical

The user interface design outlines created for each page of the application were shown to the client for his feedback and approval before commencement of the development process.

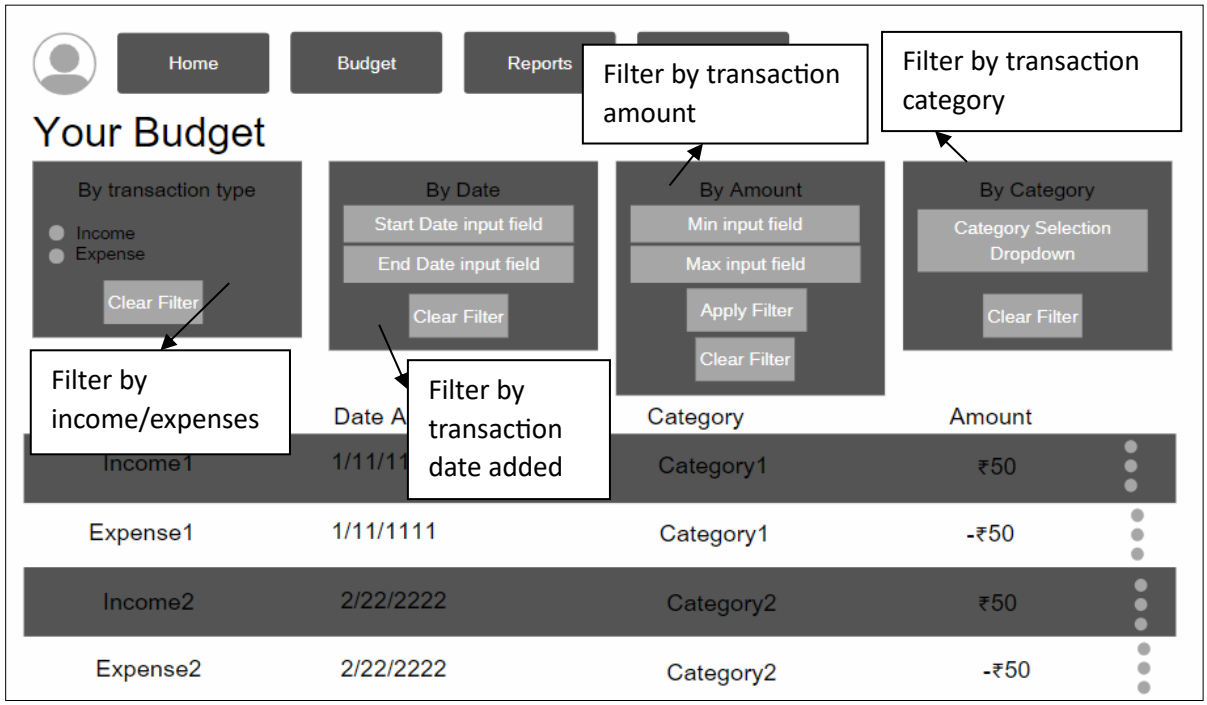
SignIn Page – Design Outline



Home Page – Design Outline

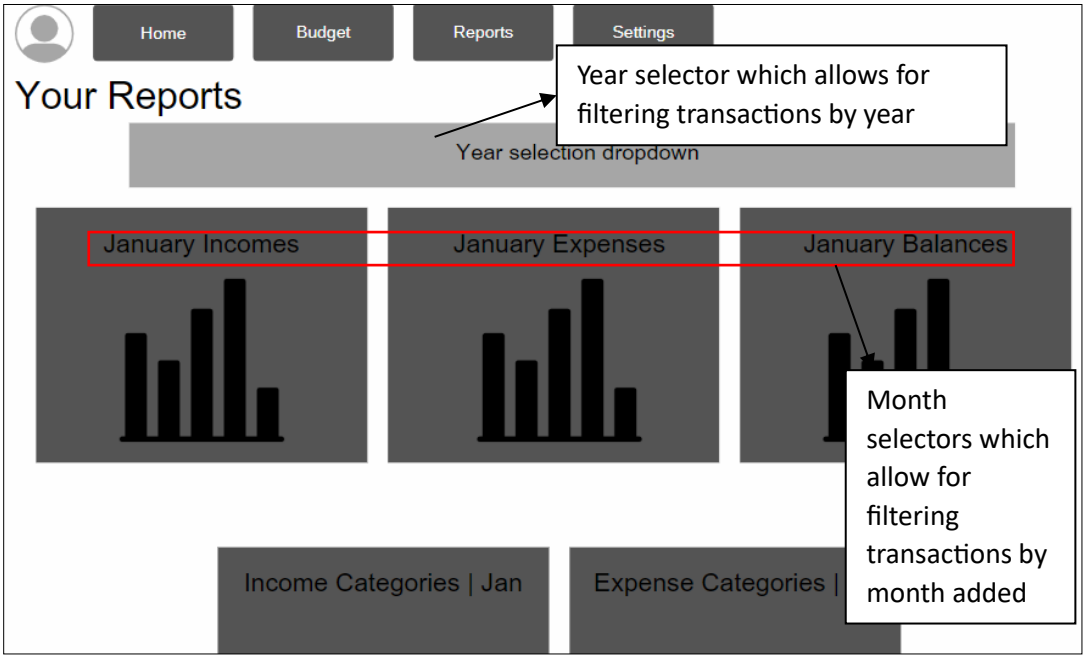


Budget Page – Design Outline

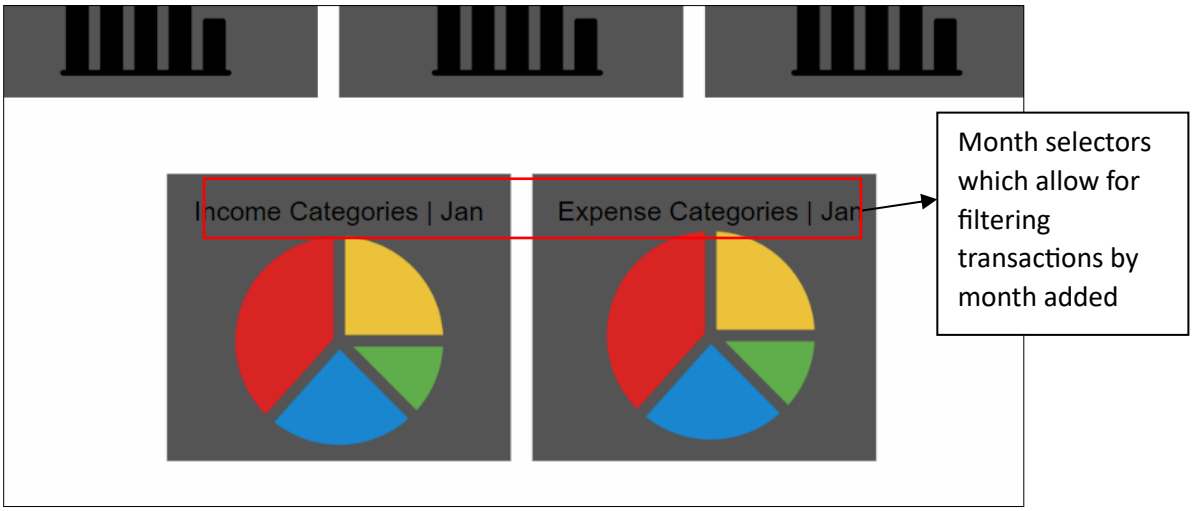


Reports Page – Design Outline

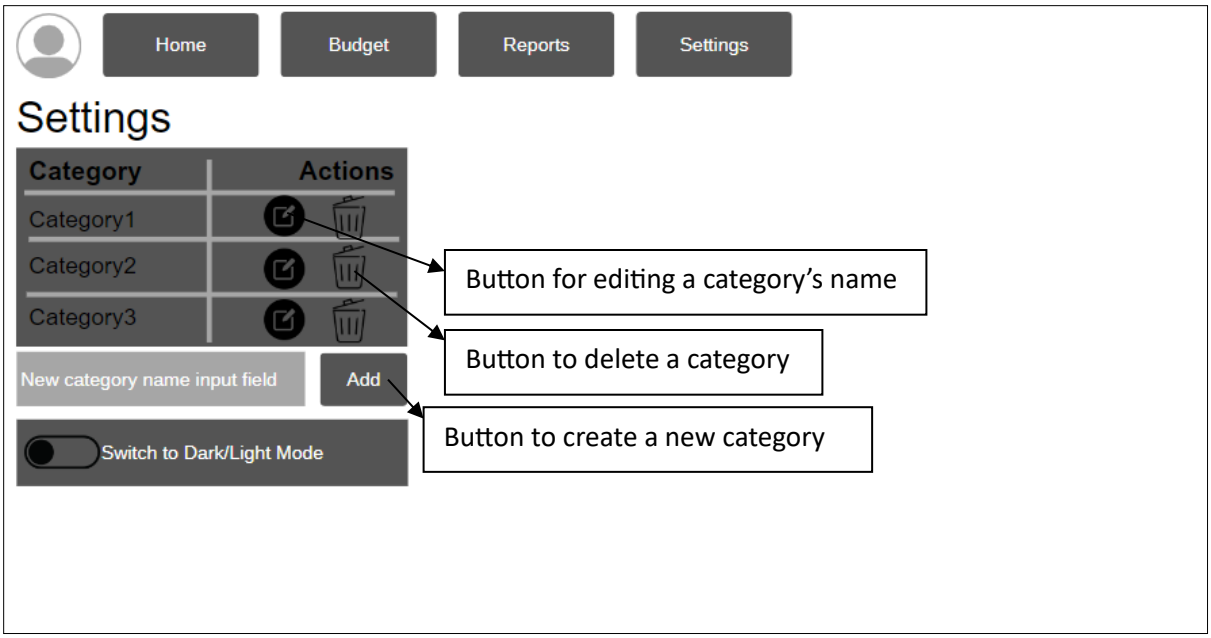
Page Design – 1



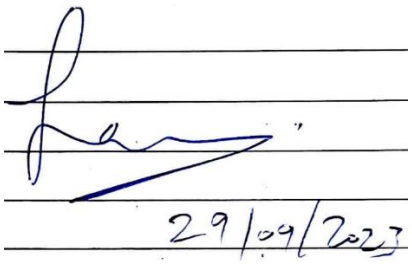
Page Design – 2



Settings Page – Design Outline



Client's Signature of Approval



E1 Client Interview Transcript:

Date: 14/11/2023

Mode of Interview: Physical

Interviewer: Good afternoon sir, I hope that you've been able to use the application?

Client: Yes, I've used the application for the past seven days.

Interviewer: Has it met your expectations?

Client: It has for sure, I really enjoyed using the application.

Interviewer: I'm glad to hear that sir! Can you share your thoughts on switching between your personal and work Gmail accounts while using the application?

Client: Yes, switching between my Gmail accounts has been smooth. I was particularly happy that my financial data for each account stays separate, which was one of my main concerns. I haven't experienced any mix-up of data between the two accounts, which is a relief.

Interviewer: Now, how do you feel about the security of the application?

Client: I feel reassured about it. One time, I forgot to sign in, and the application immediately redirected me to the sign-in page. This made me confident that my financial data is secure from others and only accessible when I'm logged in.

Interviewer: I'm glad to hear that. Can you also tell me your experience with adding new incomes and expenses?

Client: Adding new transactions on the Home page has been pretty straightforward, and I haven't had any issues with it.

Interviewer: Great. Moving on, how do you feel about the ability to view your five most recent transactions on the Home page?

Client: I like that feature. It's convenient to be able to see my most recent transactions in the home screen itself. I can quickly check the last few entries, and it's been very helpful so far.

Interviewer: How about editing or deleting transactions, did the application meet your expectations in that regard?

Client: Yes, I've had no issues with editing or deleting transactions.

Interviewer: What about organizing your transactions by category? Were you able to create, edit, and delete categories easily?

Client: Yes, creating and managing categories has been a simple process, and it has definitely helped me keep everything organized and save time searching for transactions. However, I do wish there was a way to add multiple categories to a single transaction for enhanced organization.

Interviewer: I'll keep a note of that sir. Next, how did the transactions filtering feature work for you?

Client: The filtering options are great. I've been able to quickly find the transactions I'm looking for across all the different criteria, whether it's by date, amount, or category. It's made managing my finances much easier compared to my previous setups.

Interviewer: That's good to hear! What's your opinion on the graphs that display your monthly transaction data, such as daily incomes, expenses, and balances?

Client: The Reports page is also very useful for having a holistic view of my financial health. I find the category piecharts to be particularly useful. But, for the first three graphs, I would prefer a more detailed bar graph with axes instead of the line graphs since it is difficult for me to gain a numerical understanding of my spending patterns.

Interviewer: I will keep that in mind sir. Have you experienced any issues with using the application across your MacBook and Windows systems?

Client: No issues at all. The application works consistently on both my computers. I can switch between the two devices without losing any of the functionality or data.

Interviewer: Lastly, have you found that the application provides clear feedback regarding any data validation errors or database operations, like success or error alerts?

Client: Yes, the status messages have been very clear for me. I appreciate that the application lets me know if something has gone wrong or if an operation was successful.

Interviewer: Is there anything else that you'd like to mention?

Client: I don't think that I have any further suggestions.

Interviewer: In that case, thank you for your feedback on the application sir and I will look into your feedback for the next update.

Client: Thanks for your help.

Client's Signature of Approval