Undo Functionality Documentation

Overview

This document outlines the design and implementation of the undo functionality in the Expense Tracker App The undo function empowers users to reverse specific actions, such as deleting a transaction, and revert the application to its previous state.

Table of Contents

1. Design Considerations

The undo feature should adhere to the MVC architecture pattern, ensuring clear separation between Model, View, and Controller components. Key design considerations include:

- **Undoable Actions:** Define which actions are undoable, such as removing transactions.
- **Previous State Storage**: Maintain a stack or list to store previous states.
- **User Interface Updates:** Communicate to the user when an action is undoable.

2. Implementation Steps

Model (MVC) (FILE: ExpenseTrackerModel.Java)

The Model component manages data and state. Implement the undo functionality in the Model:

- Maintain a stack to store previous states of transactions.
- On an undoable action (e.g., transaction removal), push the current state onto the stack.
- When undo is triggered, pop the previous state from the stack and update the model.

Controller (MVC) (FILE: ExpenseTrackerControllerl.Java)

The Controller handles user input and interactions. Update the Controller component:

- Add an event listener or method to handle the undo action.
- Instruct the Model to revert to the previous state when undo is triggered.

View (MVC) (FILE: ExpenseTrackerView.Java)

The View component is responsible for the user interface. Update the View:

- Provide user feedback when an action is undoable (e.g., display a message).
- Reflect the current state of transactions after an undo operation.

3. User Instructions

To use the undo feature in the Expense Tracker App:

1. Perform an undoable action (e.g., remove a transaction).
2. After the action, a message or indication will appear to confirm the action is undoable.
3. To undo the action, click the "Undo" button or use the specified keyboard shortcut.
4. The previous state of transactions will be restored.

4. Testing

Testing the undo functionality is essential to ensure it works as expected:

- Test various scenarios involving undoable actions.
- Verify that the application's state is correctly restored after undo.
- Check for edge cases and potential issues.

Undo Functionality Design Document: Remove Selected Transactions

The objective of this document is to outline the design for implementing the "Remove Selected Transactions" functionality in the ExpenseTracker application. This feature allows users to conveniently remove specific transactions from the list by selecting rows and using the "Remove" button.

ExpenseTrackerController Modifications:

1. **removeSelectedTransaction(List<int> selectedRowIndexes):**
   A new method, **removeSelectedTransaction**, is introduced in the controller to handlethe removal of transactions.

Key Steps:

- Check if the selected row indexes are valid (greater than or equal to 0).
- Retrieve the corresponding transaction from the model using the selected row index.
- If a valid transaction is found, remove it from the model.
- Refresh the view to reflect the removal of the selected transaction by updating the table with the current list of transactions in the model.

ExpenseTrackerView Modifications:

1. A **removeTransactionBtn** button is introduced as a new field in the view.
2. **removeTransactionBtn** added to buttonPanel (user interface).
3. New method to get **removeTransactionBtn** added:
   This method retrieves the "Remove" button, allowing access for event handling.

Key Steps:

- The "Remove" button is displayed in the user interface.
- The "Remove" button's click event is handled.

ExpenseTrackerApp Modifications:

1. A new action listener is added to **ExpenseTrackerApp** to handle the "Remove" button's click event, which initiates the removal of selected transactions.

Key Steps:

- When the "Remove" button is clicked, it triggers the removeSelectedTransaction method in the controller.
- The controller identifies the selected rows and proceeds to remove the corresponding transactions from the model.
- The view updates to reflect the changes, and the total cost is adjusted accordingly.

Confirmation Functionality:

Another function worth contemplating is the integration of a confirmation system. When users click the "Remove" button on the user interface, a confirmation dialog could pop up, asking them to either confirm or cancel the removal of chosen transactions. This approach adds an extra level of security, reducing the risk of unintentional data loss. If the user confirms, the selected transactions will be deleted; if not, no action will be taken to ensure the safety of user data.