

Ethan Robinson

Timmy Frederiksen

Trae Stevens

Will Henderson

Bear Budget Project Documentation

Abstract

Bear Budget is a personal budgeting tool that allows the user to categorize their spending and add any additional notes or recurring transactions. It contains a login and transaction system, as well as appropriate submenus. It is also convenient to use and very user friendly.

Background

One day, a student at Baylor University was spending all their money in the “grease pit”, eating out every night instead of using their meal plans at dining halls. That night, the student was paying their credit card bill when they noticed they had spent \$300 on fast food. The student was very sad, as they realized all of their savings from high school was almost depleted. “If only I had a system that could have tracked my spending and helped me realize I was spending too much money on fast food...” Thus, the idea of Bear Budget was born. A small team of Computer Science majors began working together to build the perfect system that can organize an individual’s spending so that they can budget better. The small team also included features such as a transaction log system, login system, and much more.

Project Vision

The group members are Ethan Robinson, Timmy Frederiksen, Trae Stevens, and Will Henderson, with Ethan Robinson being the defined leader. Our vision for the project is a personal budgeting tool. We plan to implement features such as allowing users to categorize spending, subdivide the categories, identify trends in their spending, and comment on each section of spending. The system will also allow the user to edit the amount of money they currently have available to spend, as well as display the amount of money they will have left over after expenses. The user will be able to differentiate between essential & non-essential expenses. The system will also be able to track how much money has been spent on a given category over a given amount of time. There will be a heavy emphasis on a user-friendly visual interface with simple terminology.

Gantt Diagram

Tasks

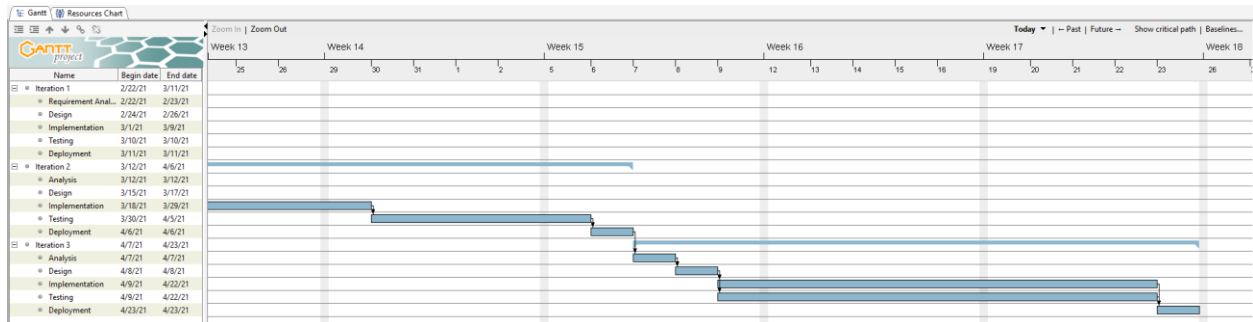
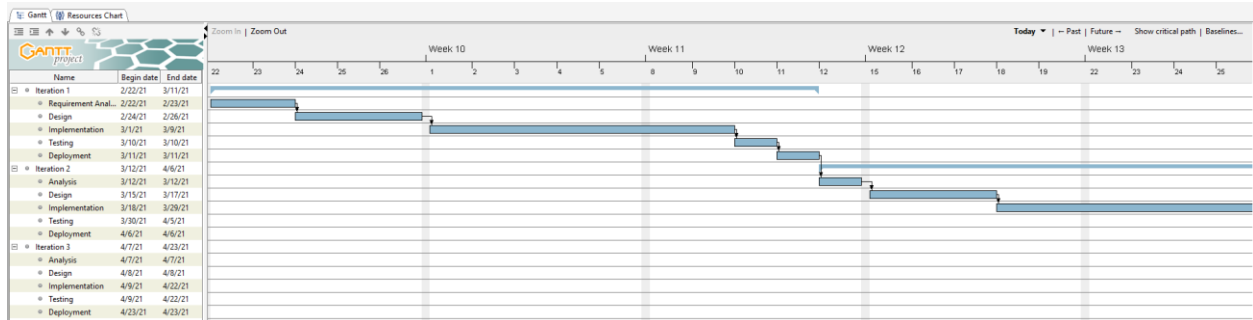
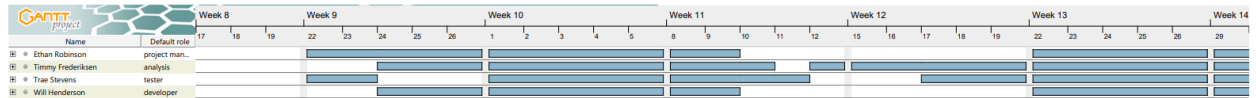
Name	Begin date	End date
Iteration I	2/22/21	3/11/21
requirement analysis	2/22/21	2/23/21
design	2/24/21	2/26/21
implementation	3/1/21	3/9/21
testing	3/10/21	3/10/21
deployment	3/11/21	3/11/21
Iteration II	3/12/21	4/7/21
analysis	3/12/21	3/16/21
design	3/17/21	3/19/21
implementation	3/22/21	3/31/21
testing	4/1/21	4/5/21
deployment	4/6/21	4/7/21
Iteration III	4/8/21	4/30/21
analysis	4/8/21	4/12/21
design	4/13/21	4/16/21
implementation	4/19/21	4/26/21
testing	4/27/21	4/28/21
deployment	4/29/21	4/30/21

Untitled Gantt Project

Mar 3, 2021

Resources Chart

5



Requirements

req Requirements

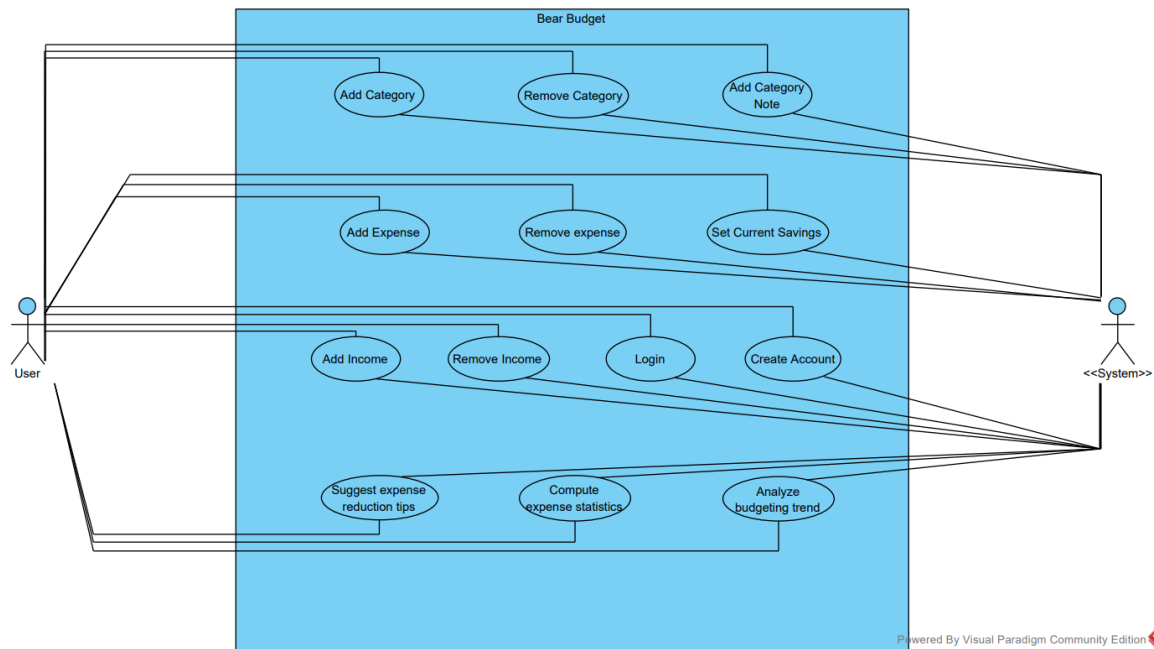
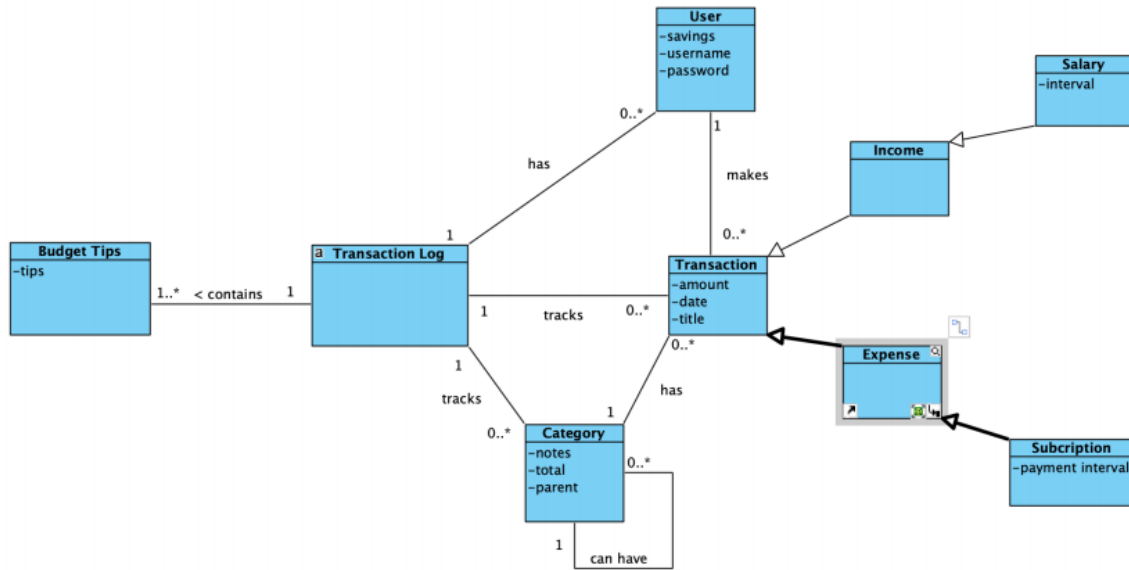
<<requirement>> Create Account Text = "The user can create an account" ID = "REQ001" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = ""	<<requirement>> Log In Text = "The user can log in" ID = "REQ002" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Log Out Text = "The user can log out" ID = "REQ003" source = "" kind = "Functional" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Run Continuously Text = "Maintain minimal usage if kept running in background" ID = "REQ004" source = "" kind = "Performance" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Create Categories Text = "User can create categories for transactions" ID = "REQ005" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Sort by Categories Text = "System can sort expenses and income by category" ID = "REQ006" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Delete Category Text = "User can delete custom made categories" ID = "REQ007" source = "" kind = "Functional" verifyMethod = "" risk = "Medium" status = ""
<<requirement>> Edit Category Text = "User can change name of category and change will be made system wide" ID = "REQ008" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Display by Category Text = "User expenses and incomes can be displayed by category" ID = "REQ009" source = "" kind = "Interface" verifyMethod = "" risk = "High" status = ""	<<requirement>> Find Common Expenses Text = "System can determine common expenses" ID = "REQ010" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Display Common Expenses Text = "User can see all of their most common expenses" ID = "REQ011" source = "" kind = "Interface" verifyMethod = "" risk = "High" status = ""	<<requirement>> Recurring Transactions Text = "The system can handle calculations involving recurring transactions" ID = "REQ012" source = "" kind = "" verifyMethod = "" risk = "" status = ""		
<<requirement>> Display Transactions Text = "The system can display all of the users transactions" ID = "REQ013" source = "" kind = "" verifyMethod = "" risk = "High" status = ""	<<requirement>> Add User Note Text = "User can add custom note to transactions" ID = "REQ014" source = "" kind = "Functional" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Display User Note Text = "System can display users custom notes" ID = "REQ015" source = "" kind = "Interface" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Remove User Note Text = "User can remove their custom notes" ID = "REQ016" source = "" kind = "Functional" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Essential Expenditures Text = "The system can sort between essential and non-essential transactions" ID = "REQ017" source = "" kind = "Functional" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Historical Trends Text = "The system can display historical trends for transactions" ID = "REQ018" source = "" kind = "Interface" verifyMethod = "" risk = "Low" status = ""	
<<requirement>> Subscriptions Text = "The system can calculate total expenditures on subscriptions over time and sort them by cost and occurrence" ID = "REQ019" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Show Subscriptions Text = "The user can see their subscriptions sorted by cost and occurrence" ID = "REQ020" source = "" kind = "Interface" verifyMethod = "" risk = "High" status = ""	<<requirement>> Tips Text = "The system can offer the user tips to reduce their spending" ID = "REQ021" source = "" kind = "Functional" verifyMethod = "" risk = "Low" status = ""	<<requirement>> Add Transaction Text = "The user can add transactions to the system" ID = "REQ022" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Remove Transaction Text = "The user can remove transactions from the system" ID = "REQ023" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	<<requirement>> Set Current Savings Text = "The user can set their current savings to be used for calculations" ID = "REQ024" source = "" kind = "Functional" verifyMethod = "" risk = "High" status = ""	

Powered By Instar Paradigm Community Edition



	REQ001	REQ002	REQ003	REQ004	REQ005	REQ006	REQ007	REQ008	REQ009	REQ010	REQ011	REQ012	REQ013	REQ014	REQ015	REQ016	REQ017	REQ018	REQ019	REQ020	REQ021	REQ022	REQ023	REQ024
Create Account	X																							
Log In		X																						
Log Out			X																					
Add Expense											X	X						X	X			X		
Remove Expense																X							X	
Set Current Savings																								X
Add Category				X	X			X																
Add Category Notes								X						X	X									
Remove Category							X		X															
Add Income																					X			
Remove Income																							X	
Trend Analysis					X					X								X						
Expense Statistics										X	X					X		X	X					
Saving Tips										X											X			

Analysis (Excluding documentation related to the use cases)



```

+setNote(note : String) : void
+getSubCategories() : Array<ICategory>
+getSubCategory(id : Int, category : ICCategory) : ICCategory
+getSuperCategory() : ICCategory
+setSuperCategory(superCategory : ICCategory) : void

```



Ethan's Use Case Documentation

ID: Add user notes to an existing category

Scope: Personal budgeting application

Level: User goal

Stakeholders and interests:

User

- person that is adding notes to the existing category.

Preconditions:

A budget category already exists.

User wants to add new user notes to the budget category.

Postcondition: New user notes will be added to the existing budget category.

Main Success Scenario:

1. User wants to add notes to an existing category.
2. User selects the existing category that they want to add notes to.
3. User writes the message for the new note.
4. System creates the new note with the specified message by the user.
5. User saves the new note to the category in the system.

Exceptions:

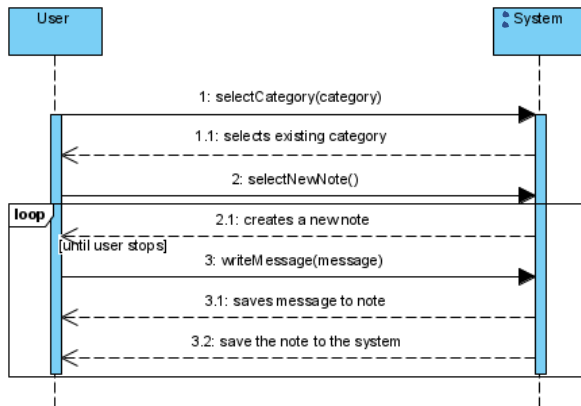
a.* anytime system does not respond

1. user will restart the application

4. a If the user wants to add multiple notes

1. Repeat step 4. As many times as necessary.

Add User Notes Use Case - SSD



Operation Contracts:

Name: selectCategory(category : integer)

Cross References: Use Cases: Remove Category

Output: The category will be currently selected.

Pre-Conditions:

User already has an existing account.

User wants to remove a category.

Post-Conditions:

A category will be currently selected.

Name: selectNewNote()

Cross References: None.

Output: A new note will be created associated with the selected category.

Pre-Conditions:

A category is currently selected by the user.

Post-Conditions:

A new note object will be created.

Name: writeMessage(message : String)

Cross References: None.

Output: None.

Pre-Conditions:

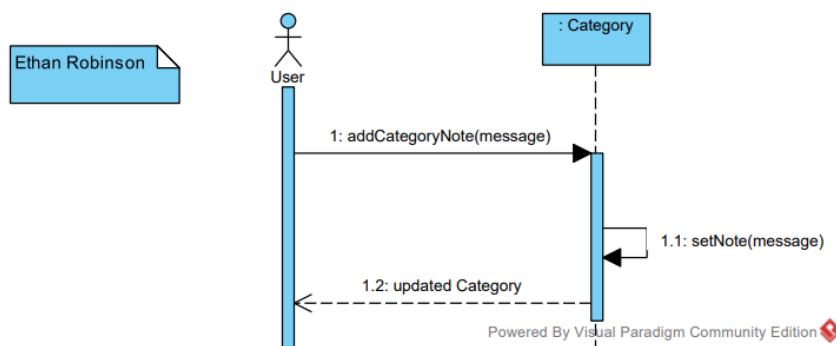
A new note object has been created and is currently selected.

Post-Conditions:

The message on the note object will be changed to the specified message.

The new note will be saved with the associated category to the system.

Add Category Note Use Case - SD



Ethan Robinson

ID: Add Category of spending

Scope: Personal Budgeting Application

Level: User goal

Stakeholders and interests:

User

- person that is adding a category to manage their spending.

Preconditions:

User already has an existing account.

User wants to add a new category to help manage their budget.

Postcondition: A new category is created where the user can add expenses under it.

Main Success Scenario:

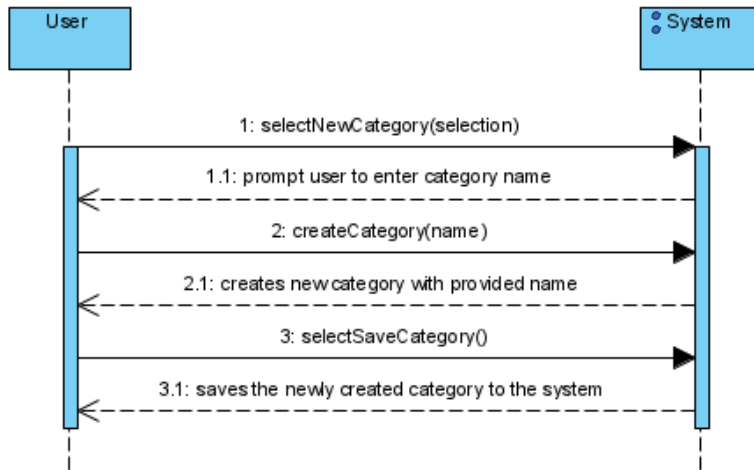
1. User wants to create a new category of spending.
2. User selects the option to add a new category.
3. System prompts user to enter a name for the category.
4. User enters a name for the new category.
5. System creates a new category with the user's specified name.
6. User saves the new category to the system.

Extensions:

a.* anytime system does not respond

1. user will restart the application

Add Category Use Case – SSD



Operation Contracts:

Name: `selectNewCategory(selection : integer)`

Cross References: None.

Output: New category with specified name.

Pre-Conditions:

User already has an existing account.

User wants to add a new category to the system.

Post-Conditions:

System prompts user to enter a name for the category.

Name: createCategory(name : string)

Cross References: None.

Output: New category with provided name.

Pre-Conditions:

User selected the option to create a new category.

User entered in a name for the category.

Post-Conditions:

New category object was created with the specified name.

Name: selectSaveCategory(selection : integer)

Cross References: None.

Output: None.

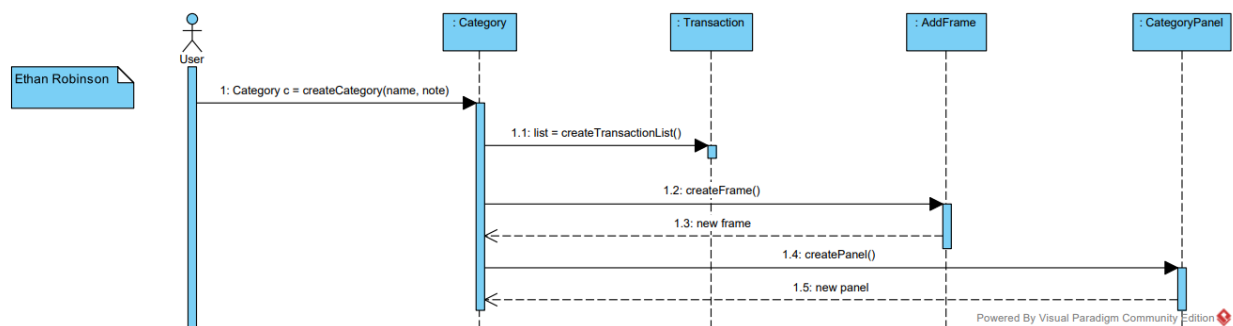
Pre-Conditions:

A new category was created by the user.

Post-Conditions:

The new category is saved to the system.

Add Category Use Case - SD



Ethan Robinson

ID: Remove Category of spending

Scope: Personal budgeting application

Level: User goal

Stakeholders and interests:

User

- person that is removing the category to manage their spending.

Precondition: User wants to remove an existing category in the system.

Postcondition: An existing category, any subcategories, expenditures, and any associated notes are removed from the system.

Main Success Scenario:

1. User wants to remove a category of spending.
2. User will navigate to the category submenu.
3. System displays the category submenu.
4. User selects the category of spending they want to remove.
5. User selects the delete button.
6. System will delete the category and any associations with it.

Extensions:

a.* anytime system does not respond

1. user will restart the application

6. a If the category contains any subcategories

1. the system will delete all subcategories associated with the category

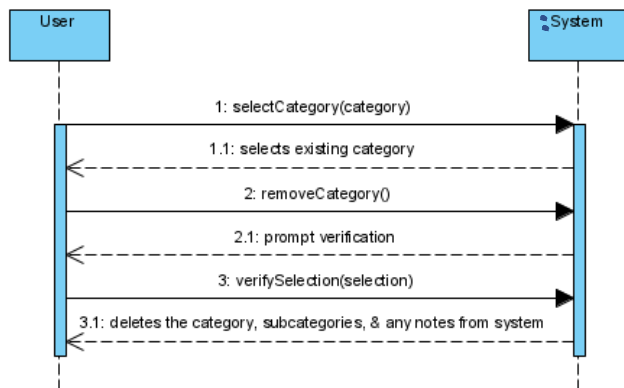
6. b If the category contains any expenditures

1. the system will delete all expenditures associated with the category

6. c If the category contains any notes

1. the system will delete all notes associated with the category

Remove Category Use Case - SSD



Operation Contracts:

Name: selectCategory(category : category)

Cross References: Use Cases: Add Category Note

Output: The category will be currently selected.

Pre-Conditions:

User already has an existing account.

User wants to remove a category.

Post-Conditions:

A category will be currently selected.

Name: removeCategory()

Output: The category will be removed.

Pre-Conditions:

User has selected an existing category.

Post-Conditions:

The system will prompt the user with a verification message.

Name: verifySelection(selection : integer)

Output: The selection to remove the category will either be confirmed or denied.

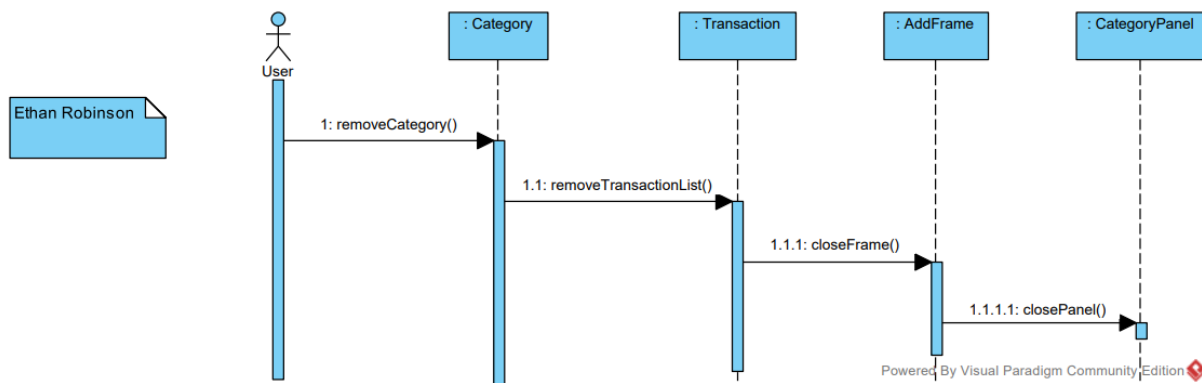
Pre-Conditions:

The user has selected the option to remove a category.

Post-Conditions:

The category will either be removed or not.

Remove Category Use Case – SD



Timmy's Use Case Documentation

Timmy Frederiksen

ID: Add Income Transaction

Scope: Personal Budgeting App

Level: User Goal

Stakeholders and Interests:

User

- person who is adding an income transaction to their budget.

Precondition: User has an income that they would like to add to their budget.

Postcondition: User's budget reflects this income and can calculate accordingly.

Main Success Scenario:

1. User wants to add a new income to their budget
2. User selects "add income" from the interface
3. User adds a title to this transaction
4. User enters the amount of income
5. User enters the date of the transaction
6. User selects whether this is a recurring income or not
7. User selects transaction category from existing categories
8. User saves the new transaction to the system

Extensions:

a.* Anytime the system doesn't respond

User will restart the application

b.* Anytime the user decides not to save their changes

User will cancel the new transaction creation process

6.a If the user selects a recurring payment

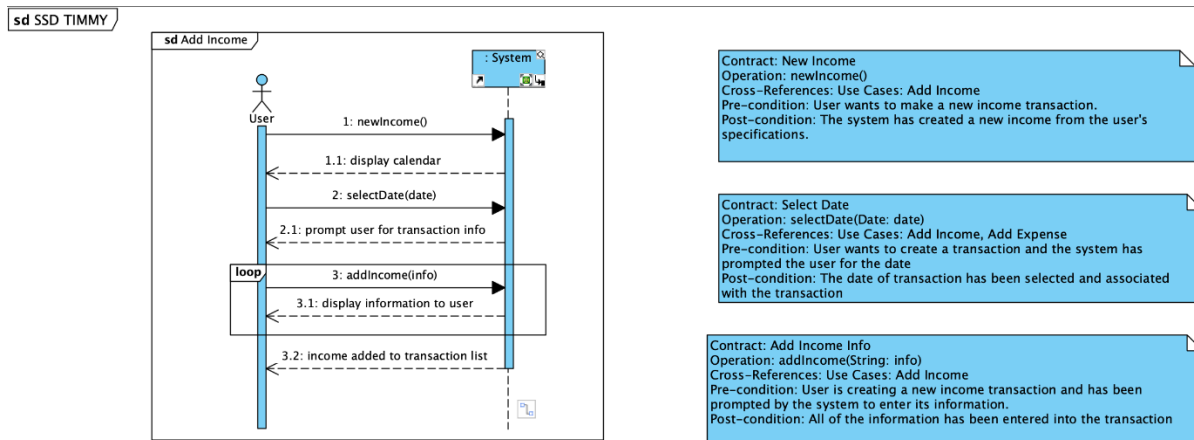
1. The system will prompt for the recurrence interval for this payment

7.a If the desired category does not exist

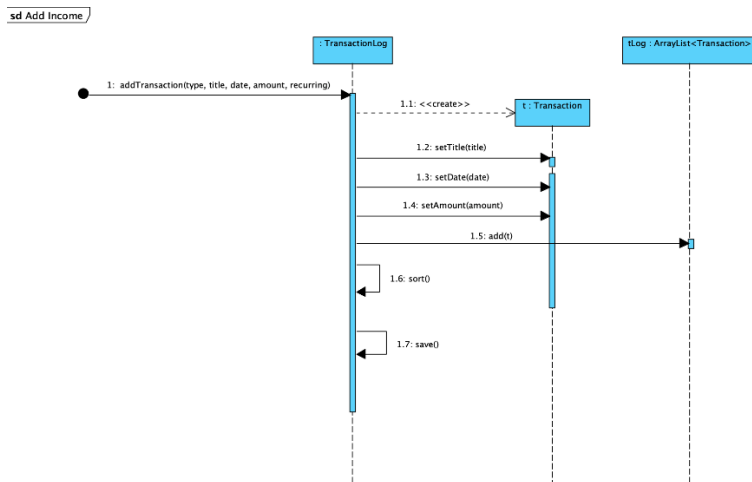
1. The user should cancel the transaction

2. The user should then create a new transaction category

3. The user then should return to create their transaction and will be able to select the new category



Add Income – SD



Timmy Frederiksen

ID: Remove Income Transaction

Scope: Personal Budgeting App

Level: User Goal

Stakeholders and Interests:

User

- person who is removing an income transaction from their budget.

Precondition: User has an income that they would like to remove from their budget.

Postcondition: User's budget no longer reflects this income and can calculate accordingly.

Main Success Scenario:

1. User wants to remove an income from their budget
2. User selects "remove income"
3. User selects which income to remove from the list of existing incomes
4. User saves their changes
5. System removes the income from their budget

Extensions:

a.* Anytime the system doesn't respond

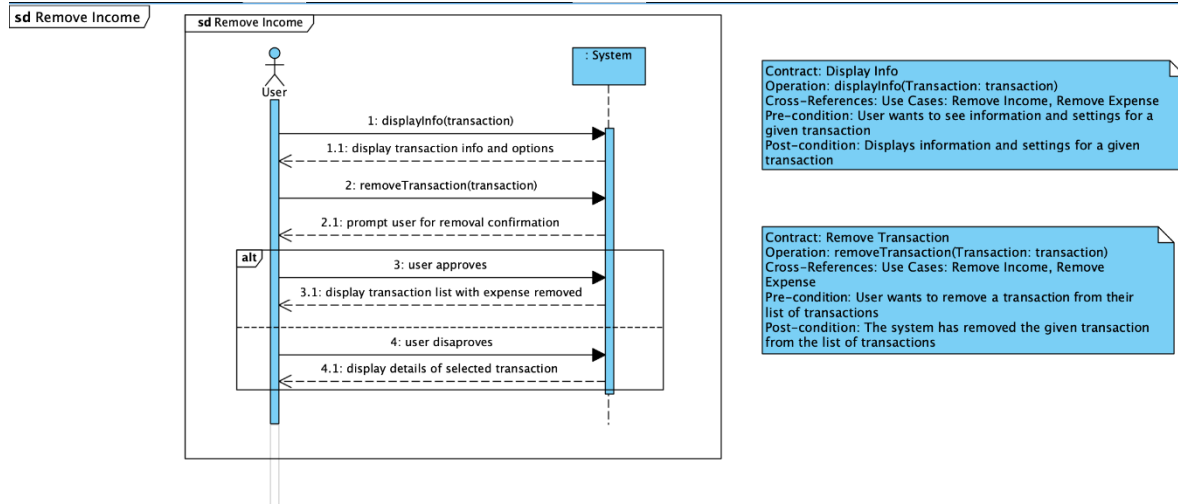
1. *User will restart the application*

b.* Anytime the user decides not to save their changes

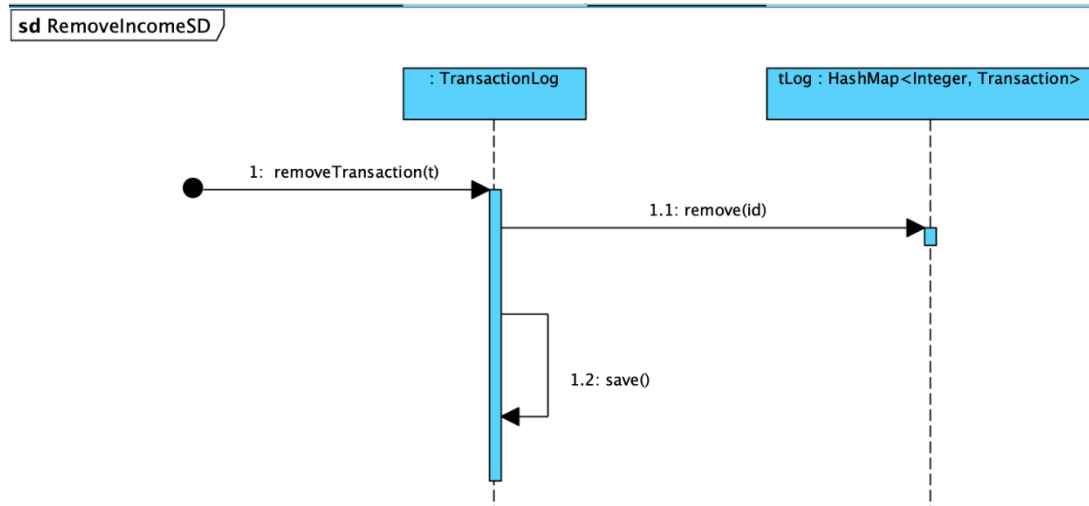
1. *User will cancel the transaction removal process*

3.a The income that they wish to remove does not exist

1. *The user will cancel the transaction removal process*



Remove Income Use Case – SD



Timmy Frederiksen

ID: Create Account

Scope: Personal Budgeting App

Level: User Goal

Stakeholders and Interests:

User

- person who wishes to begin tracking their budget using our system.

Precondition: User wants to begin tracking their budget using our system.

Postcondition: User has created a new account and can now begin tracking their budget

Main Success Scenario:

1. User wants to create a new account
2. User selects “create new account”
3. System prompts user for a username and password
4. User enters their desired username and password
5. User selects “done”
6. The system creates the account

Extensions:

a.* Anytime the system doesn’t respond

1. *User will restart the application*

1.a User already has an account

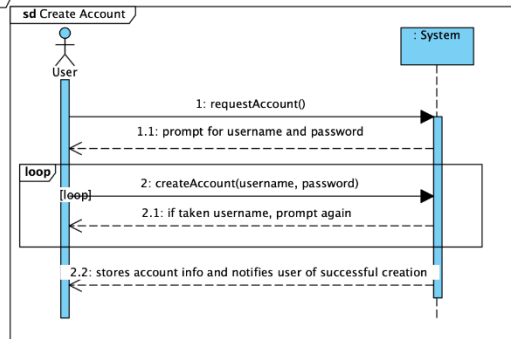
User selects “sign in” instead of “create new account”

5.* The given username is taken

1. *System informs the user that the provided username is already taken*

2. System prompts user again for a new username

sd Create Account – SSD

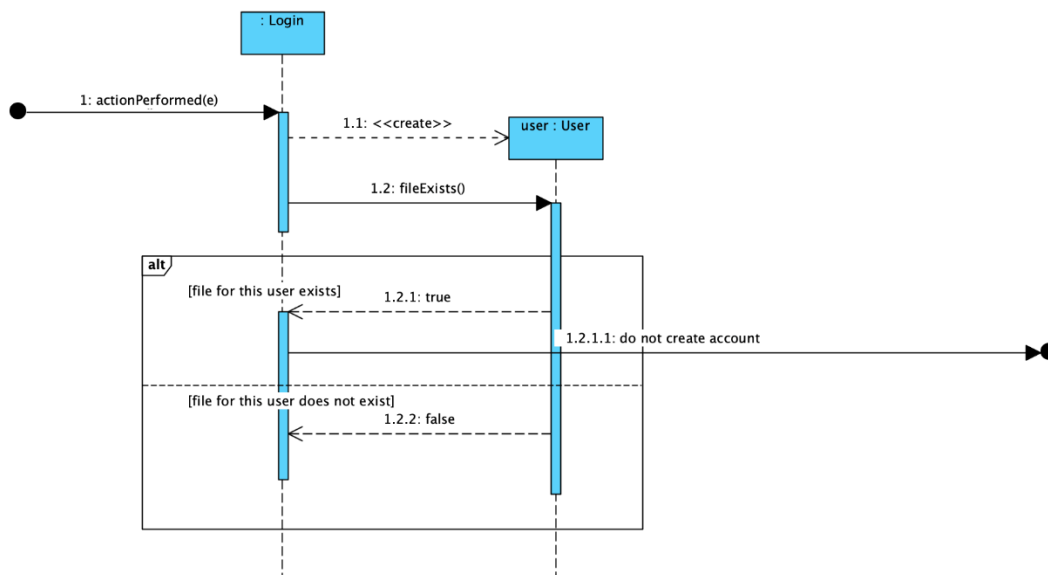


Contract: Request Account
Operation: requestAccount()
Cross-References: Use Cases: Create Account
Pre-condition: User would like to create a new account
Post-condition: User begins the account creation process

Contract: Create Account
Operation: createAccount(username: String, password: String)
Cross-References: Use Cases: Create Account
Pre-condition: User would like to create a new account with this username and password
Post-condition: System checks for availability of the username

Create Account Use Case – SD

sd CreateAccountSD



Trae's Use Case Documentation

Trae Stevens

ID: Add Transaction

Scope: Personal Budgeting App

Level: User goal

Stakeholders and interests:

User

- Wants an expense to be accounted for by the system

Precondition: The user has an expense currently unaccounted for in the system

Postcondition: The system will account for a new expense in calculations of spending habits and totals

Main Success Scenario:

1. The user wants to add an expense
2. The user will select the “add expense” option
3. The system will ask the user to select the date
4. The user will select the date
5. The system will prompt the user for the expense amount, source, and category
6. The user will enter the information
7. The system will prompt the user to confirm the information
8. The user will confirm the information
9. The system will add the expense to the transaction list

Extensions

a*. The system stops functioning

1. The user will restart the system

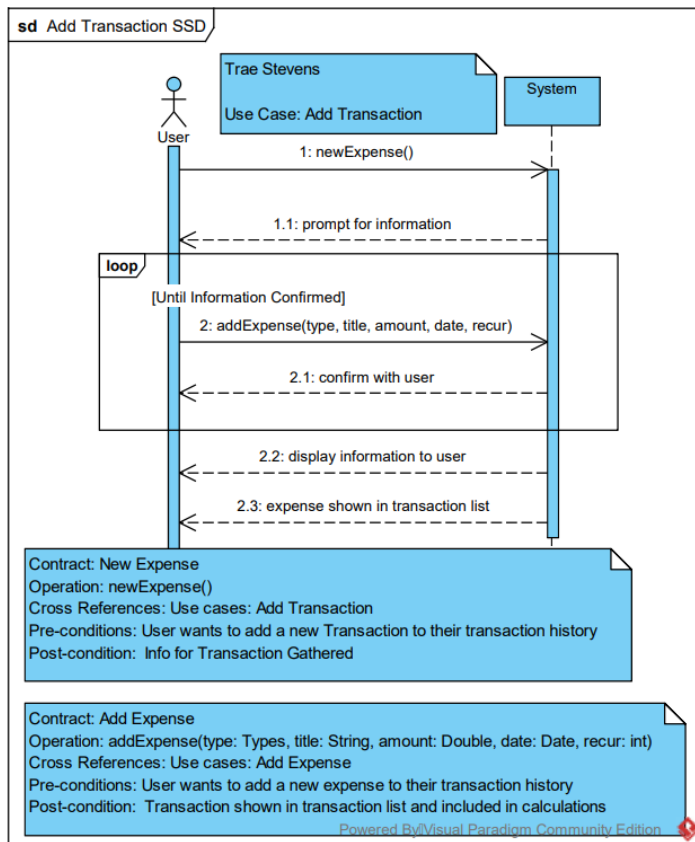
3a. The date is not available in the immediately visible calendar

1. The user will search for the specific date

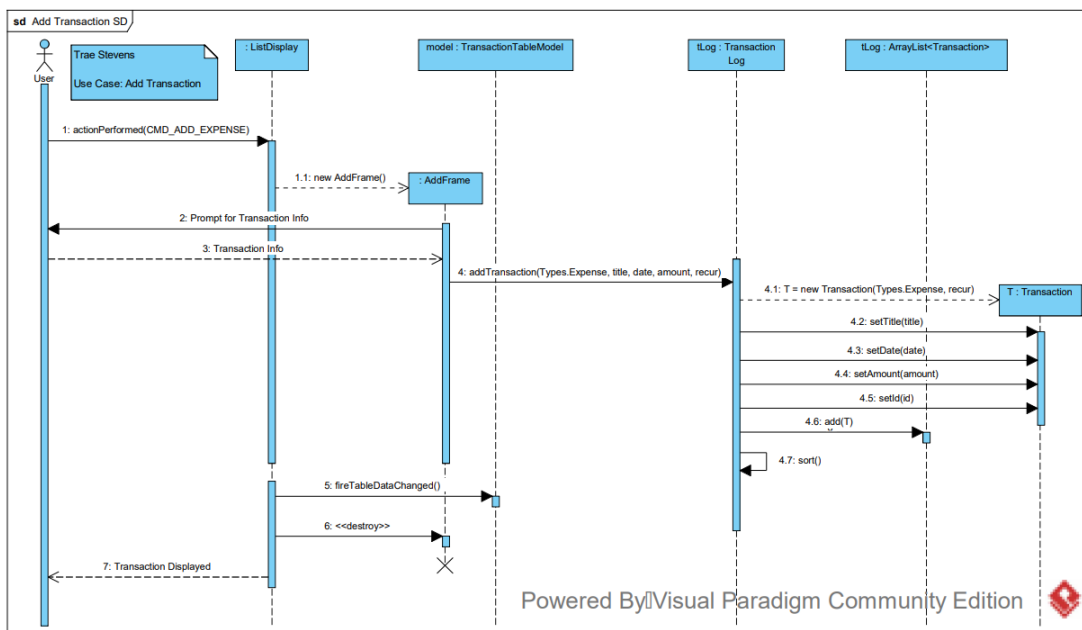
7a. The user does not approve the information

1. The system will return to allow the user to change the information

Add Transaction Use Case – SSD & Operation Contracts



Add Transaction Use Case – SD



Trae Stevens

ID: Remove Transaction

Scope: Personal Budgeting App

Level: User goal

Stakeholders and interests:

User

- Wants to remove an expense from the system

Precondition: The user has an expense they want to be removed from the system

Postcondition: The system will no longer consider the removed expense in calculations of spending habits and totals

Main Success Scenario:

1. The user wants to remove an expense
2. The user will select the expense from their list of transactions
3. The system will display detailed information and settings regarding that expense
4. The user will select “remove expense”
5. The system will confirm the removal with the user
6. The user will confirm the removal
7. The system will remove the expense from the transaction list

Extensions

a*. The system stops functioning

1. The user will restart the system

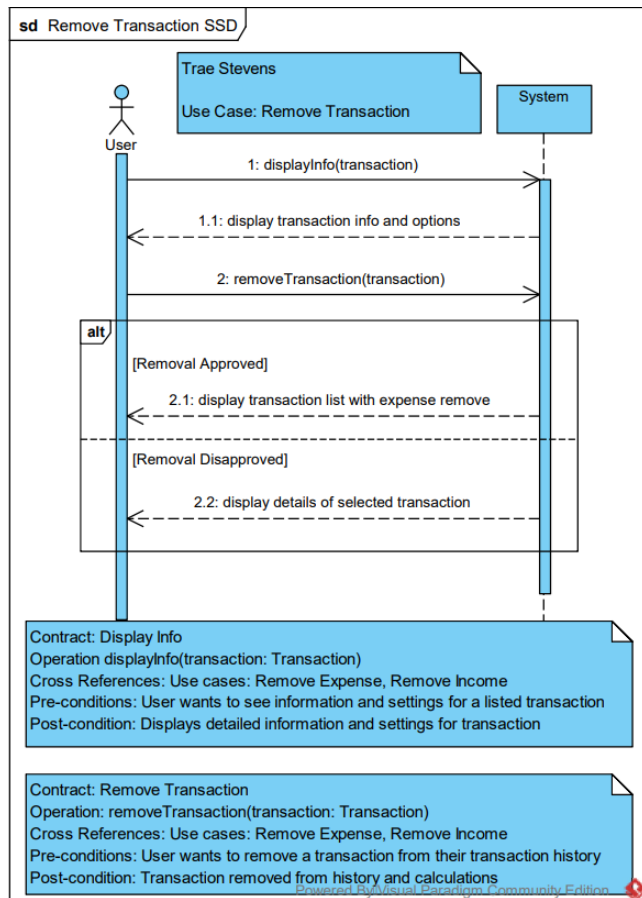
2a. The date is not available in the immediately visible calendar

1. The user will scroll through the calendar to the date

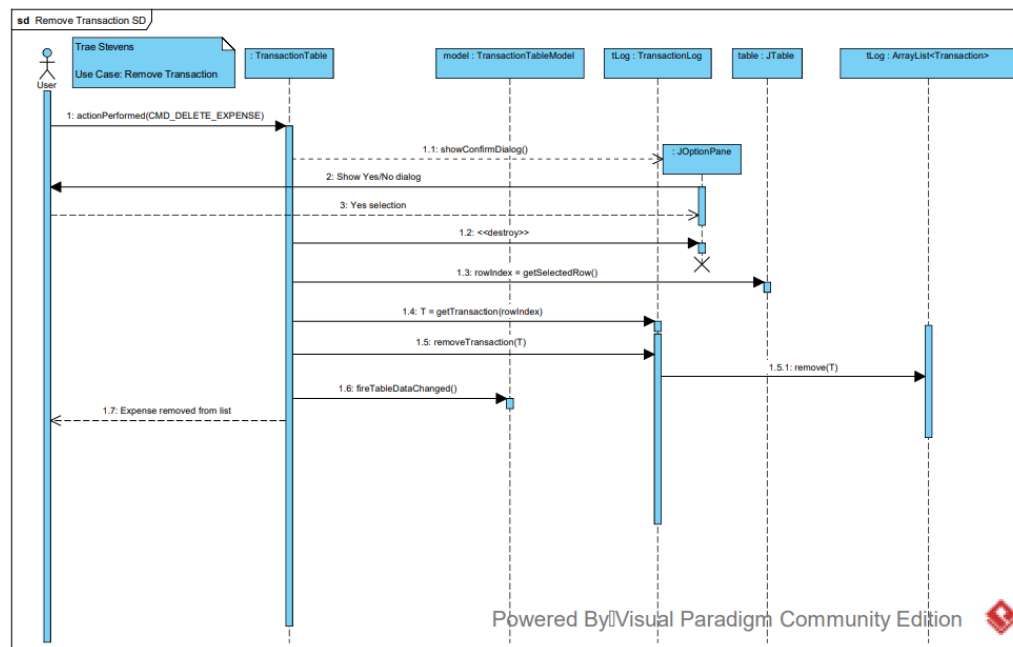
6a. The user does not approve the removal

1. The system will return to the expenses information screen

Remove Transaction Use Case – SSD & Operation Contracts



Remove Transaction Use Case – SD



Trae Stevens

ID: Set Current Savings

Scope: Personal Budgeting App

Level: User goal

Stakeholders and interests:

User

- Wants to update the current savings information in the system

Precondition: The user wants to update their existing savings at a certain date

Postcondition: The system will calculate remaining savings based on the date of the updated value

Main Success Scenario:

1. The user wants to set their current savings in the system
2. The user will choose the update savings option
3. The system will prompt the user for the date from which their savings amount changed
4. The user will select the date at which their savings changed
5. The system will prompt the user for the new savings amount
6. The user will enter their new total savings
7. The system will confirm the new value
8. The user will approve the new value
9. The system will display the savings in the transaction list at the date of the change

Extensions

a*. The system stops functioning

1. The user will restart the system

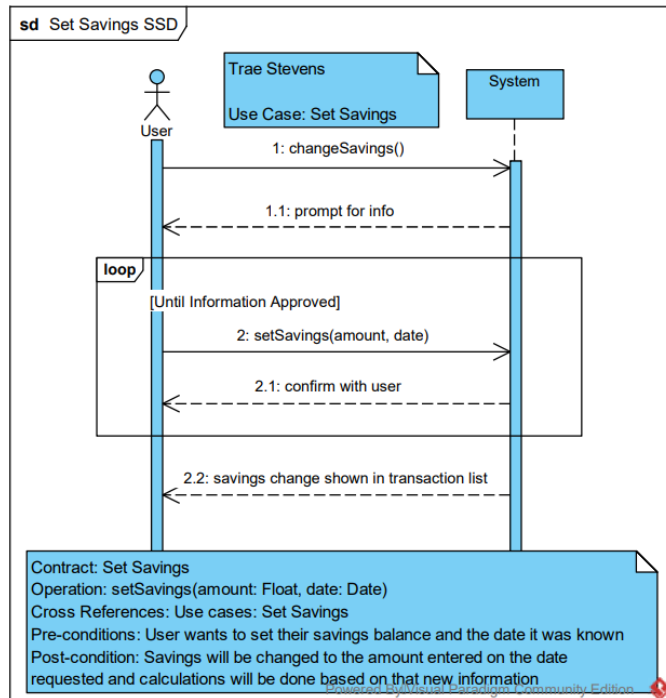
2a. The date is not available in the immediately visible calendar

1. The user will scroll through the calendar to the date

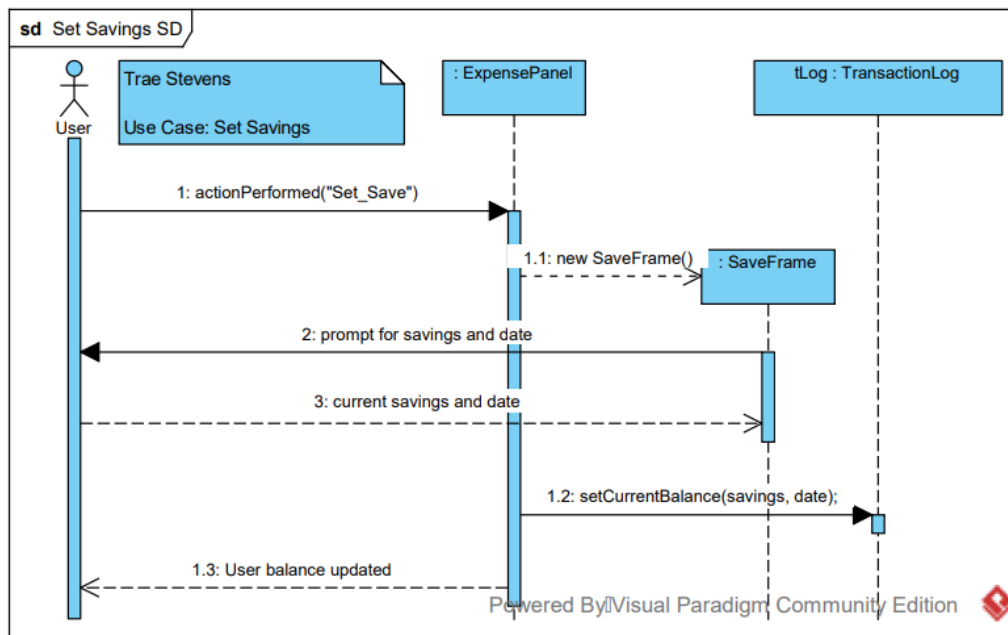
7a. The user does not approve the new savings amount

1. The system will again prompt for the savings amount

Set Savings Use Case – SSD & Operation Contracts



Set Savings Use Case – SD



Will's Use Case Documentation

Author: Will Henderson

ID: Budget Trend Analysis

Scope: Personal Budgeting App

Level: user-goal

Primary Actor: User

Stakeholders:

- User: wants illustration of financial trends to control spending.

Preconditions:

- Transactions are recorded in the app.

Postconditions:

- Data displayed indicating directionality of budget.
- Graphical visualization of extrapolated budget trend displayed.
- Comparisons to previously computed trends displayed.

Main flow:

1. User opened "Trends" tab of the application.
2. Trend report is generated by extrapolating from net savings/spendings.
3. Report graph generated illustrating recent transactions for current period and trend vector.
4. Current report compared to previously generated reports.
5. Report displayed to the user.

6. Application congratulates/reprimands user on behavior based on comparison to previous reports.

Alternate flow:

2-4a. A report was already generated within the current period.

1. Do not generate, display previous report.

2a. No transactions are recorded in the application.

1. Notify user of the lack of transactions and inability to generate report.

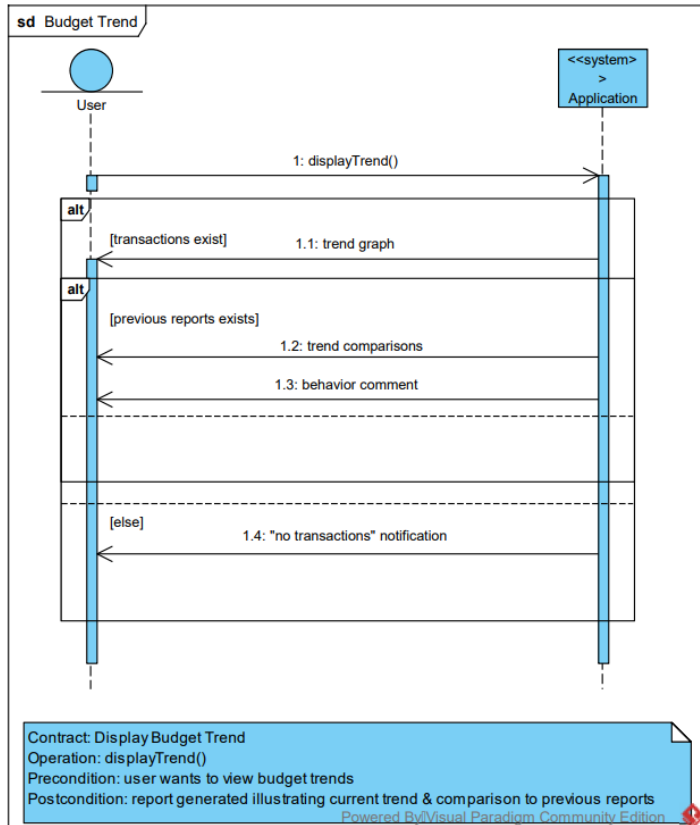
4-6. No other reports were generated.

1. Skip step 6.

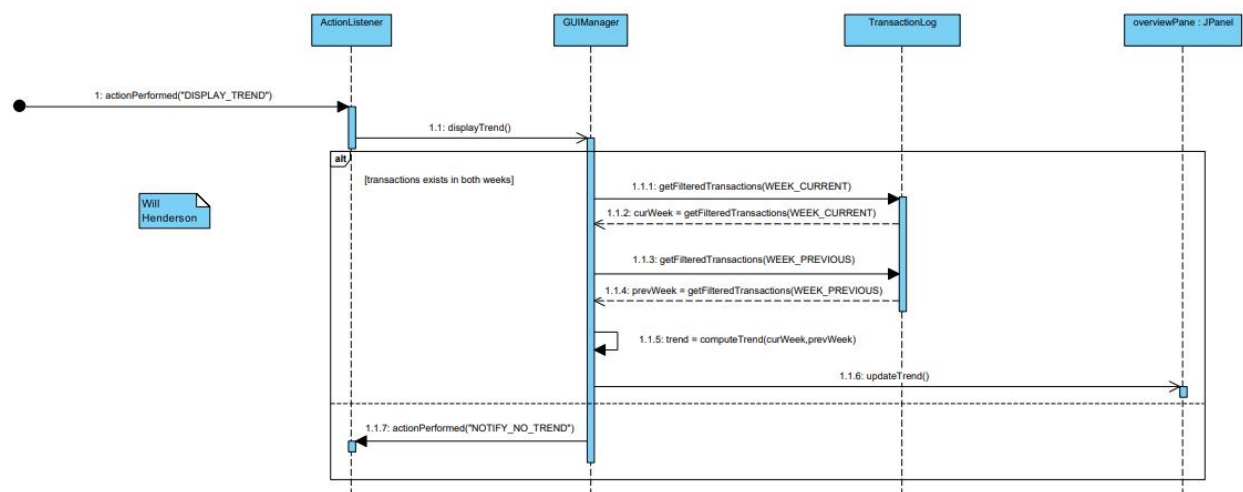
Special requirements:

- Color-coded transactions in trend graph.
- Information presented to the user is kept simplified as possible.
- Congratulations/reprimanding phrased to psychologically motivate user to budget responsibly.

Budget Trend Use Case – SSD & Operation Contracts



Budget Trend Use Case - SD



Author: Will Henderson

ID: Calculate Expense Statistics

Scope: Personal Budgeting App

Level: user-goal

Primary Actor: User

Stakeholders:

- User: wants spending statistics data per transaction category to identify problematic expenses.

Preconditions:

- Expenses are recorded in the app.

Postconditions:

- Graph with few highest-ranked expense categories displayed.
- Chart featuring varying stats per expense category displayed.
- Lists compiled with every individual recorded expense generated.

Main flow:

1. User opens "Expenses" tab of application.
2. All recorded expenses are totaled per category.
3. Graph generated displaying few categories with highest totals.
4. Chart generated with totals, averages, medians, etc. per category generated.
5. Expense transactions compiled into list and associated with respective category entry in chart.
6. Stats displayed to user.

Alternate flow:

2-5a. No recorded expenses.

1. Notify user of inability to generate statistics.

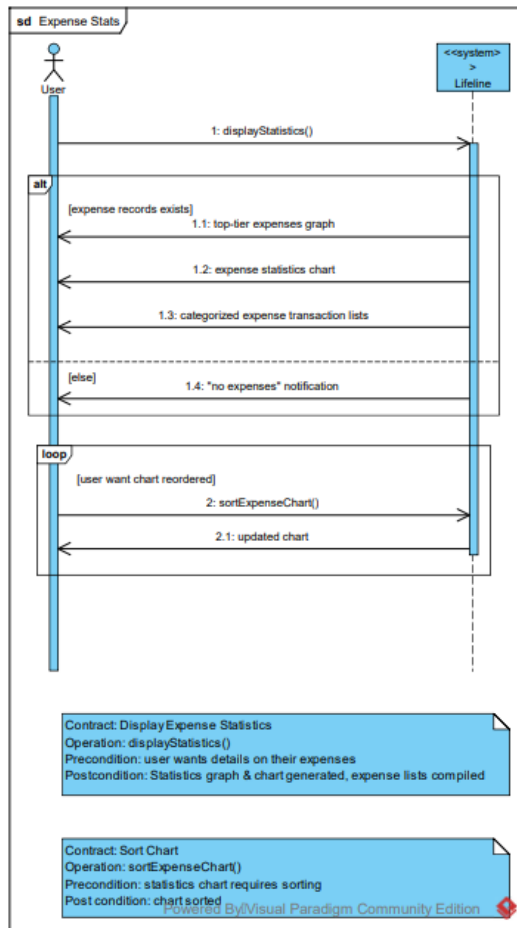
3a. Number of categories below display limit.

1. Display categories up to existing amount.

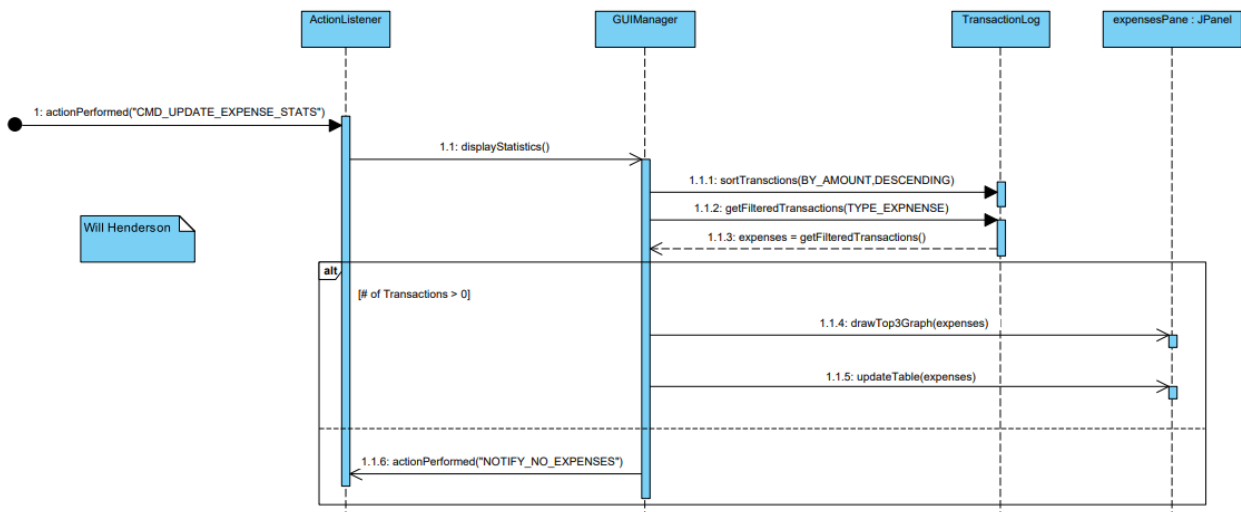
4. User wants data sorted differently.

1. User selects label of data column to sort by.
2. Chart toggles between ascending/descending.

Calculate Expense Stats Use Case – SSD & Operation Contracts



Calculate Expense Stats Use Case – SD



Author: Will Henderson

ID: Savings Suggestions

Scope: Personal Budgeting App

Level: user-goal

Primary Actor: User

Stakeholders:

- User: wants advice on saving money by controlling expenses.

Preconditions:

- Transactions are recorded in the app.

Postconditions:

- Advice displayed to user.
- Category-specific advice highlights related category.

Main flow:

1. User request savings advice or triggers advice generation during other action.
2. All recent recorded transactions, statistics, and display context are applied to an advice rule table.
3. Localized advice string displayed to user from rule table.

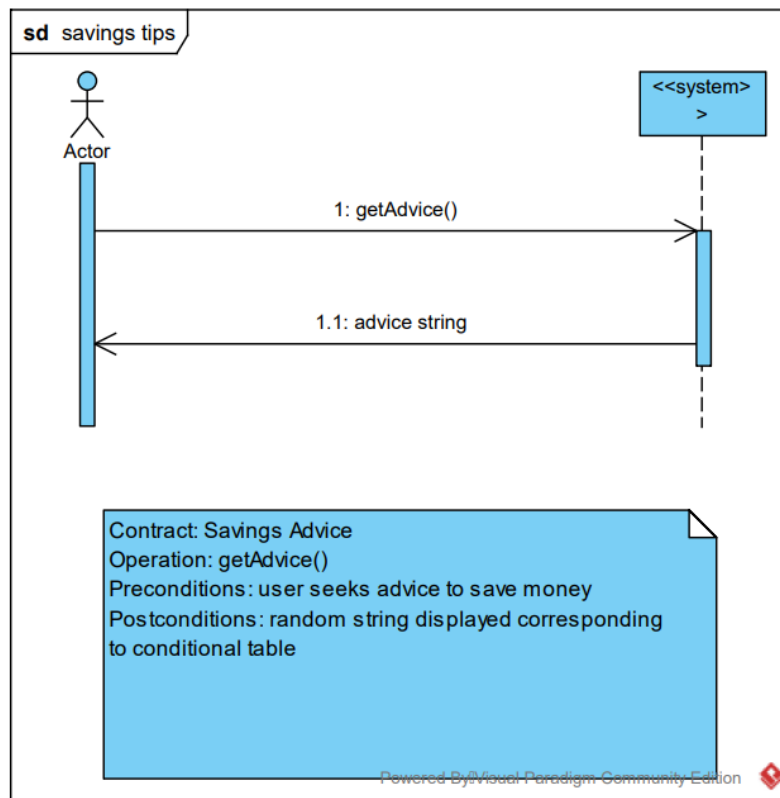
Alternate flow:

- 2a. No table entry matches the current conditions.
 1. No advice is displayed

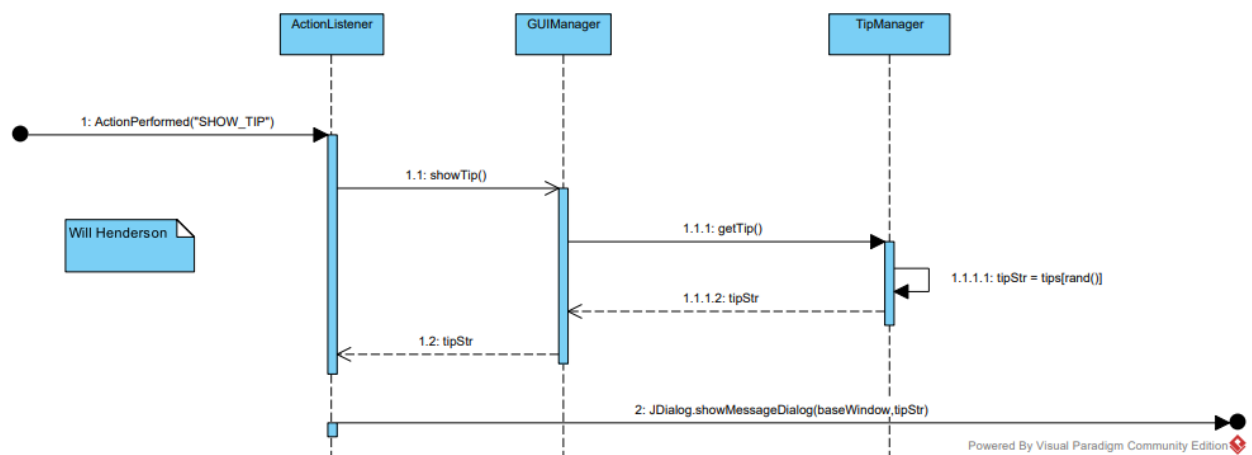
2b. Multiple entries match current conditions.

1. Select an advice string at random.

Savings Tips Use Case – SSD & Operation Contracts



Savings Tips Use Case – SD



Test Coverage Plan

To test Budget Bear, we will use Junit to test each function of the transaction log and login system. The transaction system is not fully integrated yet, including the expense menu.

For the GUI Manager, we plan to test:

- The creation of frames that will be used by various menu systems.
- The creation of panels for Expenses, Transactions, and Categories.

For the Expense Menu, we plan to test:

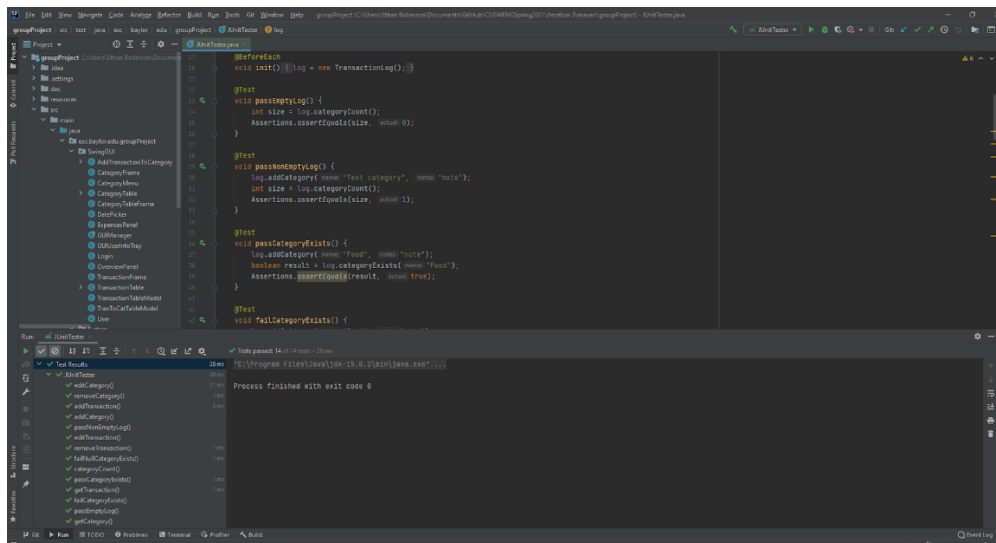
- Add Expense, through bounds checking.
- Remove Expense, through bounds checking.
- Initialization of the Edit Expense menu.

For the TransactionLog, we plan to test:

- The addition/removal of Transactions to the sorting functionality
- The addition of categories to keep track of the expenses
- The ability to group Transactions by their attributes

For the Categories, we plan to test:

- The addition of subcategories with their own transactions
- The addition of transactions within the category object.
- The removal of categories, including their corresponding transactions, subcategories, and their notes.
- The addition of category notes.
- The removal of Transactions from Categories when removed from the Transaction Log
- The addition of Transactions to subcategories



GRASP Identification

TransactionLog:

- Creator and Information Expert
- Handles everything to do with the creation and storage of Transactions and their Categories. Allows easy access to all Transactions from a single class as well as easy categorizing and sorting.

Transaction:

- Information Expert
- Stores the information regarding a transaction. Uses a *Types* enumeration to increase reusability. Creation and modification handled by TransactionLog makes it easily adjustable.

Category:

- Information Expert
- Store a list of their own subcategories as well as a list of transactions within that category. Supports low-coupling by tracking related categories and associated Transactions.

GUIManager:

- Creator.
- Creates the tabbed menu to display all other information panels. Supports high cohesion by letting panels handle their own functionality. Supports low-coupling by displaying various panels because it doesn't handle panel events or rely on other panels to exist.

ExpensesPanel

- Creator and information expert
- Creates the ListDisplay panel and buttons for interacting with the list display. Supports low-coupling and high cohesion by letting the ListDisplay handle the Transaction events itself rather than having to access its data members.

ListDisplay

- Controller and Information expert
- Handles the display of all Transactions and related events such as Transaction creation/deletion. Supports high cohesion by focusing only on the Transactions and their associated functionality.

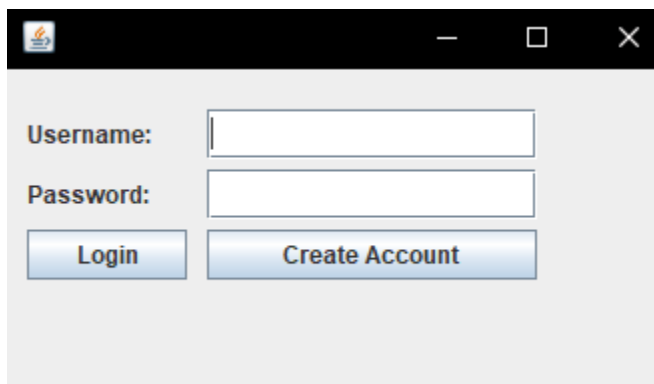
Implementation

For our Bear Budget system, we implemented a Java GUI using Swing that allows users to manage their budget. The system stores users and their transactions even while the application is not running. The user can login to the system to find their transactions waiting there. The user can manage their transaction by adding new ones, deleting transactions, editing, and placing them into categories. The Bear Budget system also determines your current savings for your budget. You can manage categories by adding, deleting, and editing them. The table of transactions displayed by our GUI allows for sorting and filtering for easy, simple access to any given transaction.

User Manual

Bear Budget is a personal budgeting tool that allows the user to categorize their spending and add any additional notes or recurring expenses. It contains a login system, a transaction log, a transaction submenu, a category submenu, and a content page. We have designed Bear Budget so that it is convenient and user friendly.

Login System

A screenshot of a web application's login interface. The interface is displayed within a window with a black title bar containing standard minimize, maximize, and close buttons. The main content area has a light gray background. It features two text input fields: the first is labeled 'Username:' and the second is labeled 'Password:'. Below these fields are two buttons: a blue 'Login' button and a blue 'Create Account' button.

Upon opening the application, the user will be presented with the login screen. If the user has already created an account, they are able to enter their username and password to login. Their information in their account will be automatically loaded when they successfully login.

To login, the user will enter their username and password and click the login button. If their credentials are correct, the system will log them in. If they are not correct, their login attempt is invalid, and the system will not log them in.

To create an account, the user will enter their desired username and password and click the create account button. Then, the default content page of the application will be automatically opened.

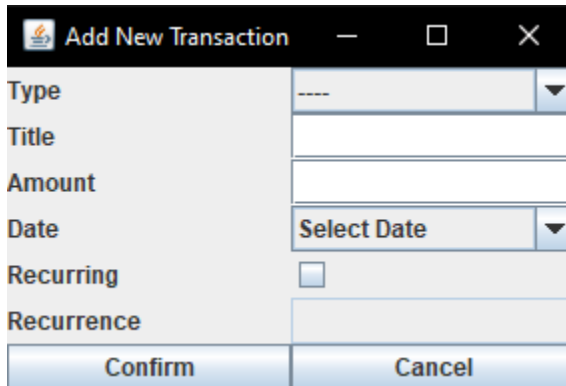
Transaction System

The screenshot shows the BearBudget application window. At the top is a menu bar with 'File' and 'Category'. Below the menu bar is a text area with a '32px' label and two lines of text: 'Sample Text' and 'Hello again'. Below the text area are two tabs: 'Overview' and 'Expenses'. The 'Expenses' tab is active, displaying a table with three columns: 'Title', 'Amount', and 'Date'. The table contains three rows of data: Walmart (-36.79, April 13, 2021), Chilis (-12.78, April 16, 2021), and HEB groceries (-50.00, April 22, 2021). Below the table is a large empty area. At the bottom of the window is a toolbar with five buttons: 'Set Savings', 'Add Expense', 'Edit Expense', 'Delete Expense', and a dropdown menu currently showing 'None'.

Title	Amount	Date
Walmart	-36.79	April 13, 2021
Chilis	-12.78	April 16, 2021
HEB groceries	-50.00	April 22, 2021

To locate the transaction submenu, the user can click on the expenses tab on the content page. Any entered expenses or income will be displayed in the transaction log, sorted by date by default.

To filter the transactions by category, the user can select the combo box in the bottom right corner of the menu. There, all the user's categories are listed. The user can select a category, then all the transactions under that category will be displayed. "None" is selected by default, so all the transactions are displayed in the table. To update the list of categories in the combo box, then the user can select "Refresh List".



Add New Transaction	
Type	---
Title	
Amount	
Date	Select Date
Recurring	<input type="checkbox"/>
Recurrence	
Confirm Cancel	

To add an expense or income to the transaction log, the user can select “Add Expense” button. The add expense frame will be opened and the user will be able to specify whether the transaction is an expense or income, its title, the amount, the date, whether it is recurring, and the number of days it is recurring.

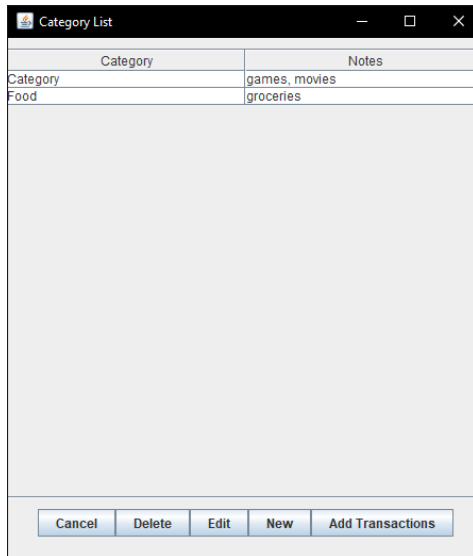
To edit a transaction, the user can select “Edit Expense” button and a similar window to “Add Expense” will open. The transaction’s information will already be filled into the fields. The user can edit any field and then select confirm to save the changes.

To delete a transaction, the user can select a transaction from the transaction log and select “Delete Expense”. Then, a confirmation window will open prompting the user to confirm their decision. If the user confirms, then the selected transaction will be deleted from the transaction log and all associated categories.

To update the user’s current savings, the user can select “Set Savings” button and they will be prompted to enter the amount of savings they currently have. The updated savings will appear near the top of the application, on the “Current balance:” line.

Category System

To view the category list, the user can select “Category” tab at the top of the application and then “View Category List” to open the category submenu.

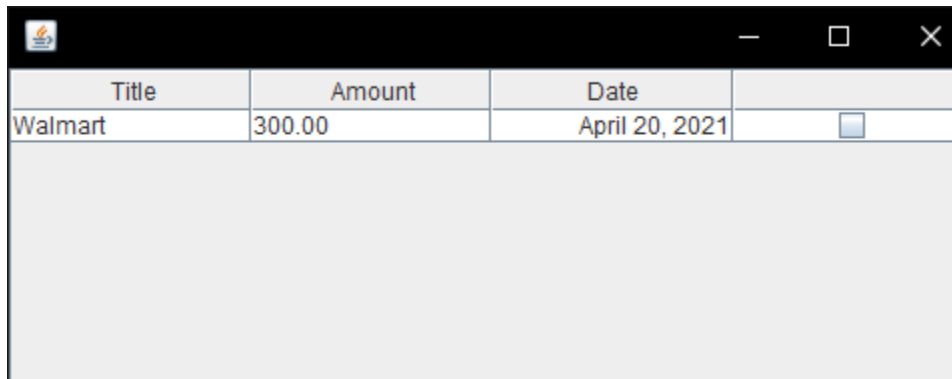


To add a category, the user can select the “New” button. This will open a menu where the user can enter the category name and any associated notes that will be displayed along with the category. To save the changes, the user can select the confirm button.




To edit a category, the user can select a category from the list and select the “Edit” button. This will bring up a similar window to the add category button, but with the selected category’s information already filled into the fields. The user can edit the name and any note to the category and can select confirm to save the changes.

To delete a category, the user can select the “Delete” button, which will open a confirmation window where the user will be prompted to confirm their decision to delete the category. If the user confirms their decision, the category will be deleted from the list.



The screenshot shows a software window with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window is a table with four columns: 'Title', 'Amount', 'Date', and an empty column. The first row of data contains 'Walmart', '300.00', 'April 20, 2021', and a small blue square icon. Below the table is a large, empty light gray rectangular area.

Title	Amount	Date	
Walmart	300.00	April 20, 2021	

To add a transaction to a category, the user can select the desired category from the list. Then, the user can select “Add Transactions” button and a menu will open with the list of transactions. To add an association between a transaction and the selected category, the user can select the check box on the right side of the window.

To return to the default menu of the program, the user can select the “Back” button. Their changes to the category list will be saved.

Saving Changes

Finally, to save all changes made to the user’s information, including the transaction system, the category list, and their savings, the user can select the “File” tab at the top of the application. From there, the user can select “Save”. This will automatically save all of their updated information to the user’s information files. This will not close the application.

Issue Tracking

<https://github.com/someuntakenusername/CSISWENGSpring2021/issues>

<input type="checkbox"/>	4 Open ✓ 31 Closed	Author ▾	Label ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	Complete application with User Interface and user input validation, test documentation and JUnit testing, user guide, live demonstration, #37 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	presentation #36 by timmyFrederiksen was closed 2 days ago						
<input type="checkbox"/>	video of 3 distinct use cases by each member #35 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	results from git analysis (how many commits, each member commits) #34 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	time tracking linking issue tracking #33 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	reporting the number of issues per member in the issue tracking #32 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	7 design patterns highlighted in the code #31 by timmyFrederiksen was closed 2 days ago						
<input type="checkbox"/>	timecards report #28 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	code quality matters (e.g., comments/formatting/coupling/cohesion) #27 by timmyFrederiksen was closed 1 hour ago						
<input type="checkbox"/>	Fix Package Diagram #24 by timmyFrederiksen was closed 26 days ago						
<input type="checkbox"/>	Presentation to-do #23 by timmyFrederiksen was closed 26 days ago						
<input type="checkbox"/>	package diagram to-do #22 by timmyFrederiksen was closed 28 days ago						
<input type="checkbox"/>	Design Class diagram to-do #21 by timmyFrederiksen was closed 26 days ago						
<input type="checkbox"/>	timecards report update #20 by timmyFrederiksen was closed 26 days ago						

Git contributions

Commits on Apr 27, 2021		
static modifications	TheLunchTrae committed 22 hours ago	85d04c1
Big Merge	TheLunchTrae committed 2 days ago	78d78f8
Checking static modifications	TheLunchTrae committed 2 days ago	6f21216
Made TransactionLog static in table for easier universal access	TheLunchTrae committed 2 days ago	503e387
fixed remove category	Retro5050 authored and Retro5050 committed 2 days ago	41da209
Merge branch 'main' into Trae	TheLunchTrae committed 2 days ago	054dbc1
Test Case Updates	TheLunchTrae committed 2 days ago	03519e4
Commits on Apr 26, 2021		
added user for testing	Retro5050 authored and Retro5050 committed 2 days ago	c02ed88
created account	Retro5050 authored and Retro5050 committed 2 days ago	51849d4
Merge branch 'main' of https://github.com/someuntakenusername/CSISWEN...	Will Henderson committed 2 days ago	0fe2d40
ort	Will Henderson committed 2 days ago	44fefd1
Set Savings	TheLunchTrae committed 2 days ago	f6e146d

Number of Commits by member:

Trae – 124 commits

Timmy – 80 commits

Ethan – 53 commits

Will – 50 commits



Roster of Hours

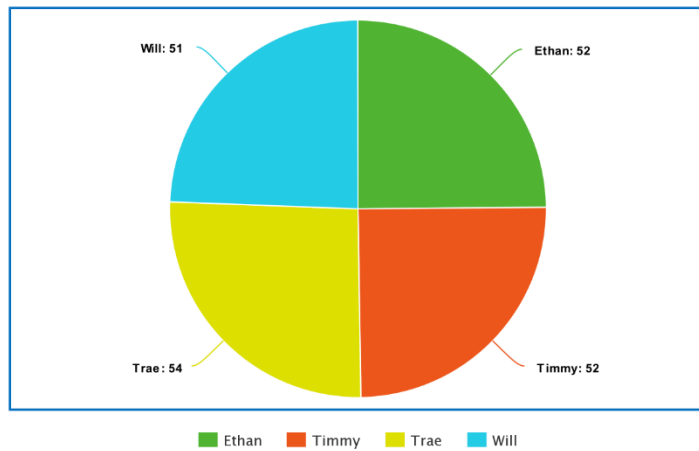
Hours Worked by member:

Ethan – 52 Hours

Timmy – 52 Hours

Trae – 54 Hours

Will – 51 Hours



meta-chart.com

Conclusion

In conclusion, Bear Budget is a very useful personal budgeting tool. It provides the user with the ability to make their own account to store their personal expenditures. With Bear Budget, the user is able to categorize their expenses and any income in a user friendly table that can be filtered and sorted by column. Through the use of GRASP we are able to provide the user with a simple interface that can clearly display the user's financial decisions.