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.

Untitled Gantt Project

Mar 3, 2021

http://

Project manager	Ethan Robinson
Project dates	Feb 22, 2021 - May 1, 2021
Completion	0%
Tasks	18
Resources	4

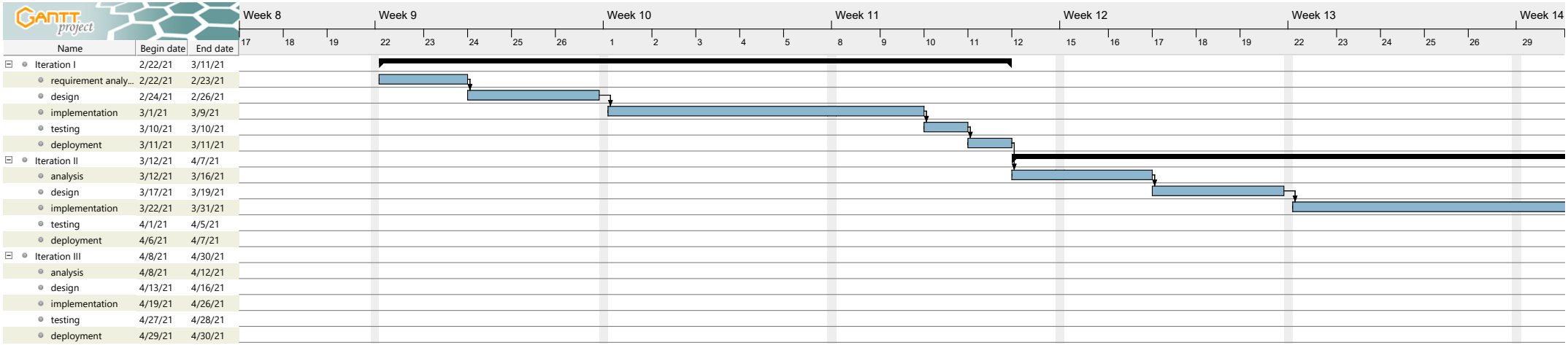
Tasks

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

Resources

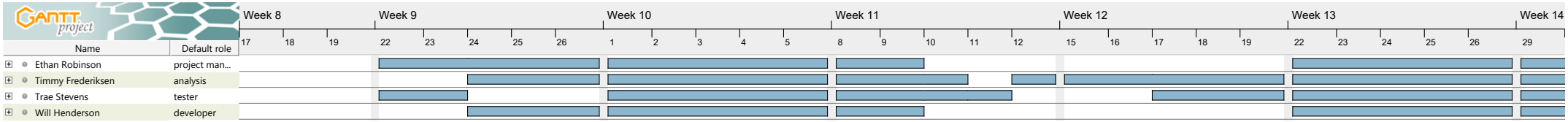
Name	Default role
Ethan Robinson	project manager
Timmy Frederiksen	analysis
Trae Stevens	tester
Will Henderson	developer

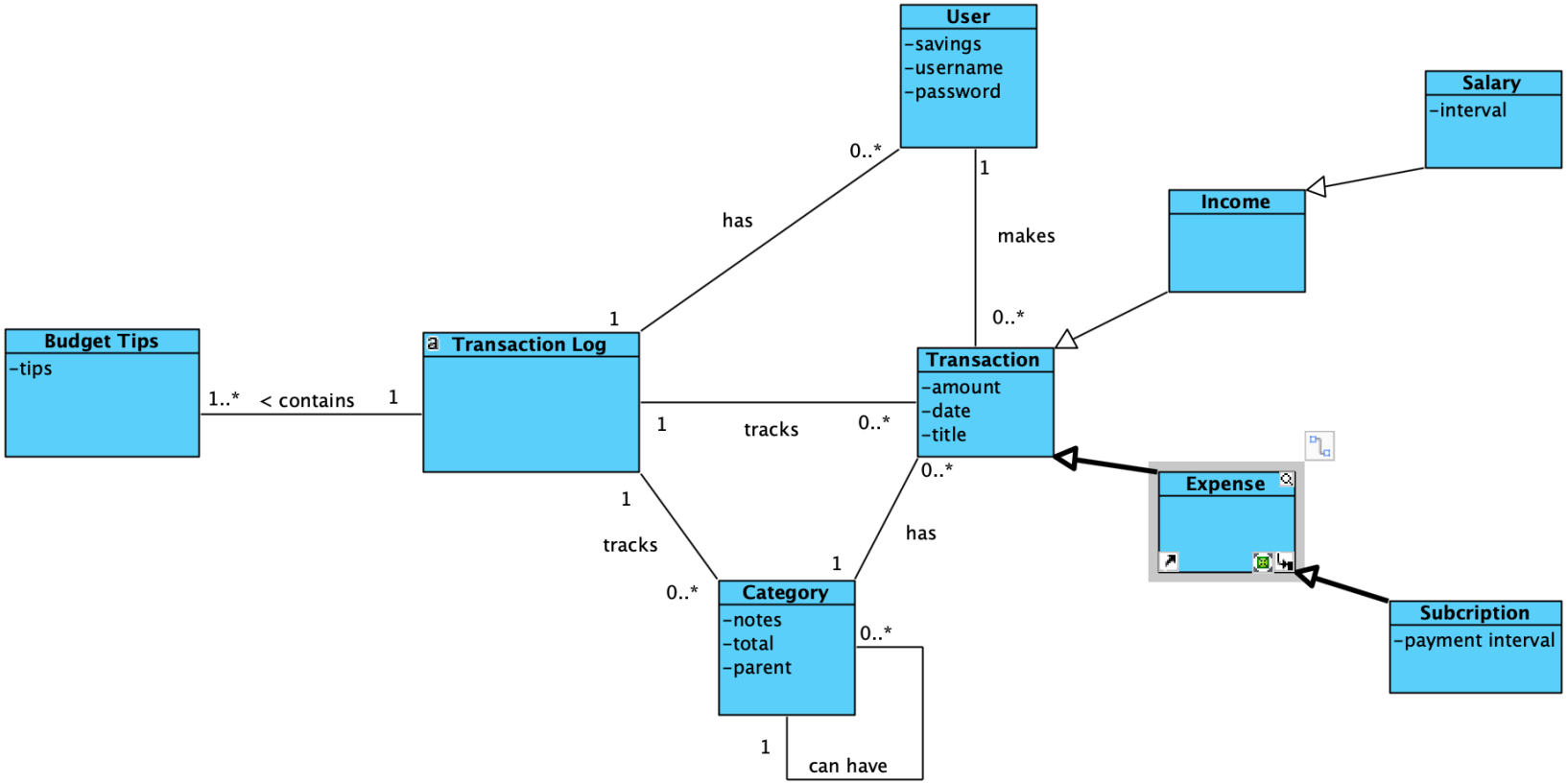
Gantt Chart



Name	Begin date	End date
Iteration I	2/22/21	3/11/21
requirement analy...	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

Resources Chart



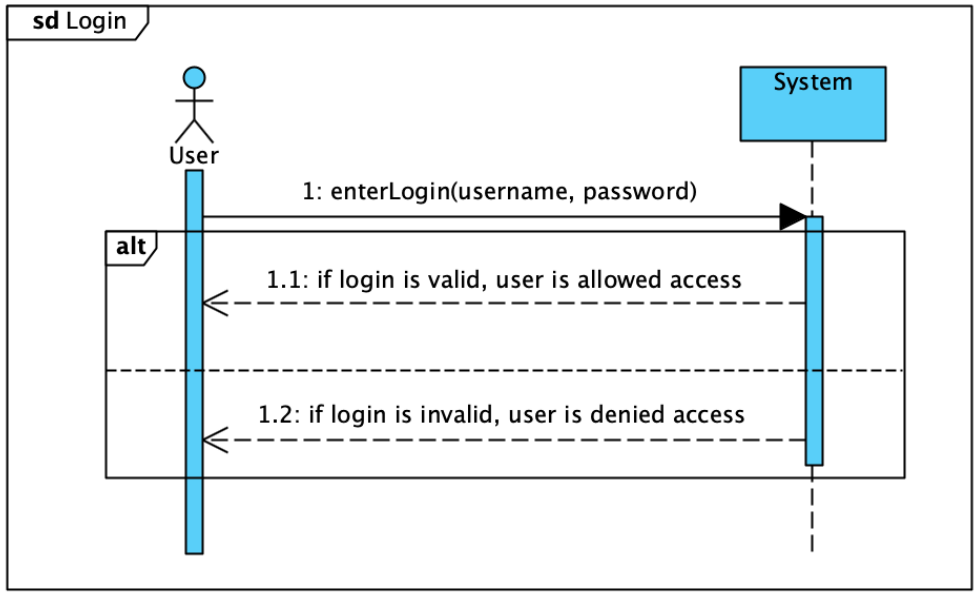

User
-savings
-username
-password
Salary
-interval
Income
Budget Tips
-tips
Transaction Log
Transaction
-amount
-date
-title
Expense
Category
-notes
-total
-parent
Subcription
-payment interval
0..*
has
1
makes
0..*
1
1..*
< contains
1
1
tracks
0..*
0..*
1
tracks
0..*
has
1
0..*
1
can have


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 Continously
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 occurence"
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"
<<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 Visual 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				

Use Case: Add Category

System
+selectCategory(category) +selectNewNote() +writeMessage(message) +selectNewCategory(category) +createCategory(name) +selectSaveCategory() +removeCategory() +verifySelection() +newIncome() +selectDate(date) +addIncome(income) +requestAccount() +createAccount(username, password) +displayInfo(transaction) +removeTransaction(transaction) +newExpense() +addExpense(type, parameter, category, date) +changeSavings() +setSavings(amount, date) +displayTrend() +getAdvice() +displayStatistics() +sortExpenseChart()



Contract: Enter Login
Operation: enterLogin(username: String, password: String)
Cross-References: Use Cases: Login
Pre-condition: User is not logged in and has an account
Post-condition: User is logged in or is told to create an

Ethan Robinson

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.

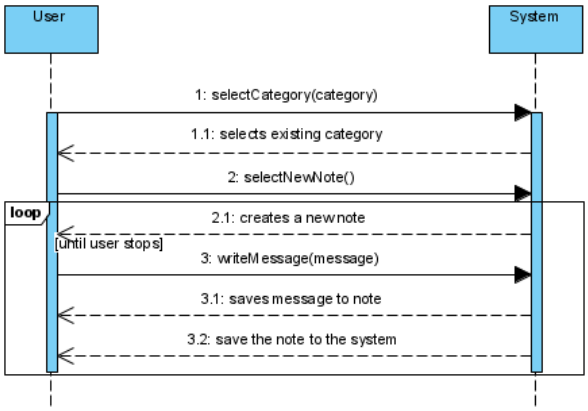
Exceptions:

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.



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.

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.

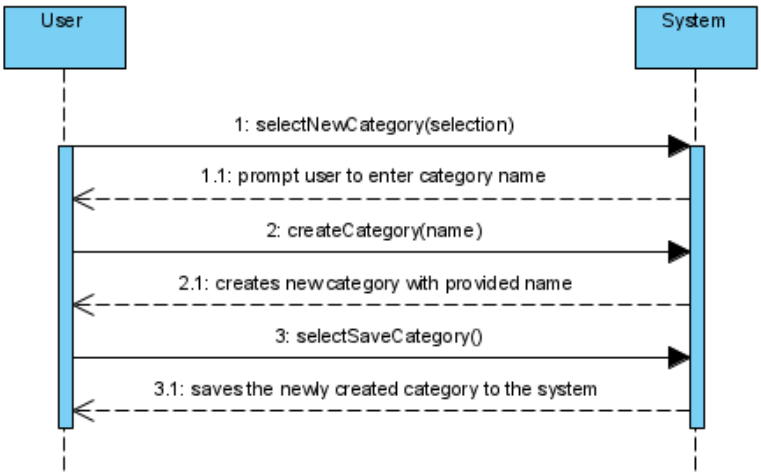
Main Success Scenario:

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



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.

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

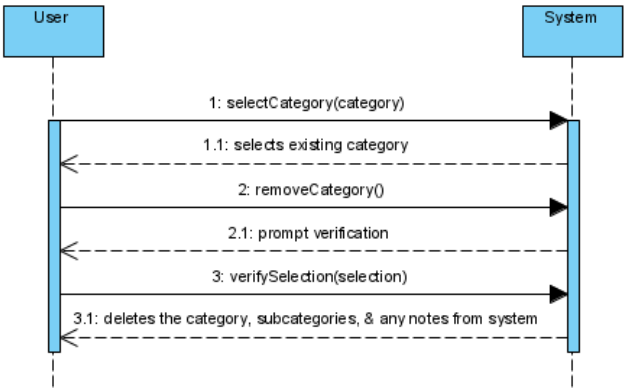
1. the system will delete all subcategories associated with the category

6. b If the cateogory 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



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)

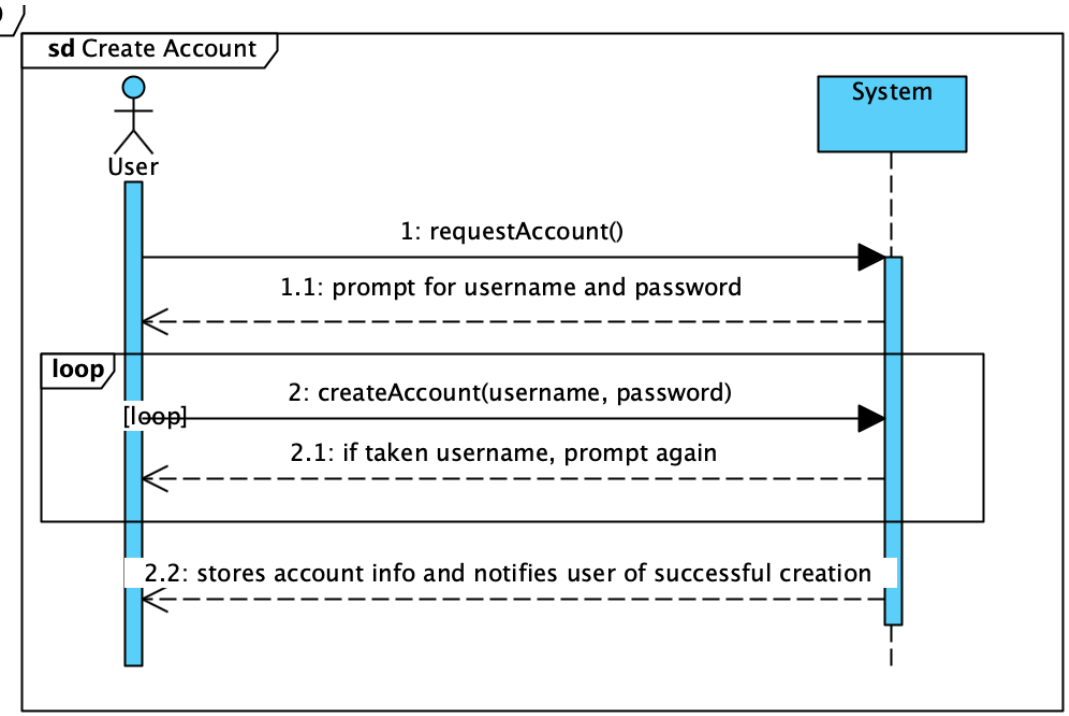
Output: The selection to remove the category will either be confirmed or denied.

Pre-Conditions:

The user has selected the option to remove a category.

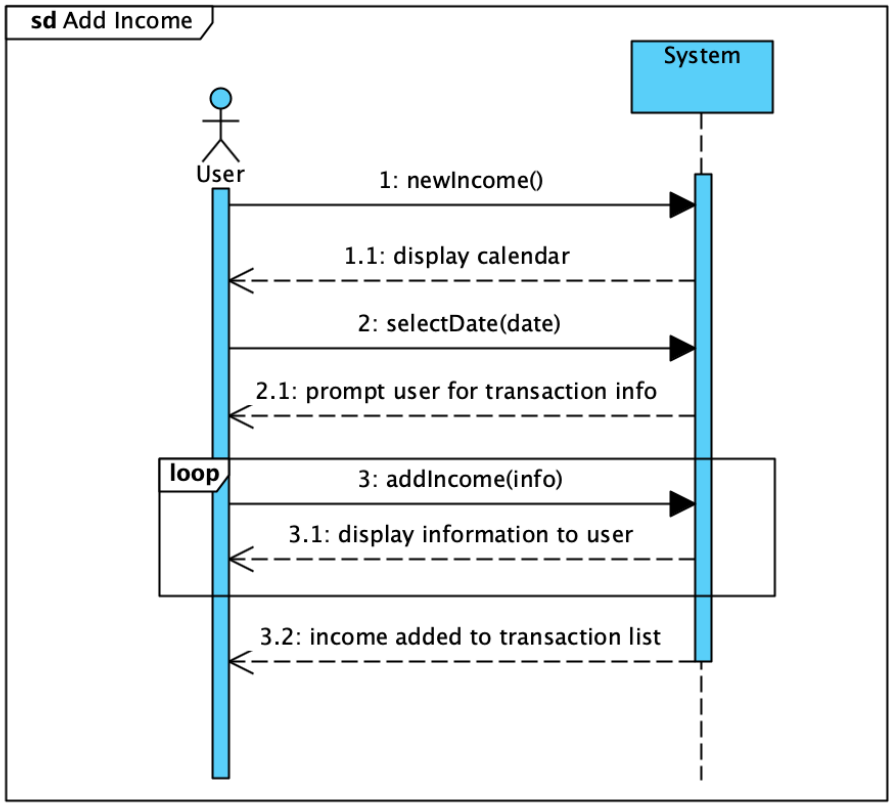
Post-Conditions:

The category will either be removed or not.



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

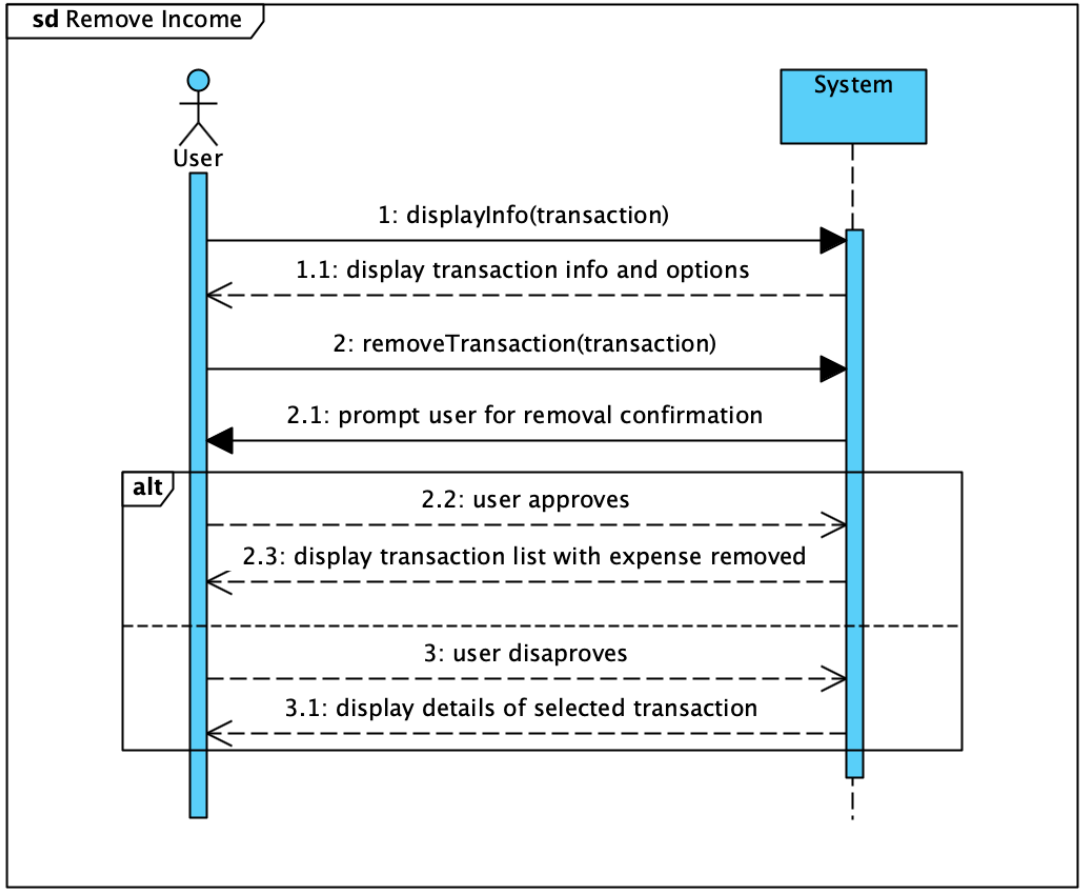
Contract: Create Account
Operation: createAccount(username: String, password: String)
Cross-References: Use Cases: Create Account
Pre-condition: User would like to create a new account with this username and password
Post-condition: System checks for availability of the username



Contract: New Income
Operation: newIncome()
Cross-References: Use Cases: Add Income
Pre-condition: User wants to make a new income transaction.
Post-condition: The system has created a new income from the user's specifications.

Contract: Select Date
Operation: selectDate(Date: date)
Cross-References: Use Cases: Add Income, Add Expense
Pre-condition: User wants to create a transaction and the system has prompted the user for the date
Post-condition: The date of transaction has been selected and associated with the transaction

Contract: Add Income Info
Operation: addIncome(String: info)
Cross-References: Use Cases: Add Income
Pre-condition: User is creating a new income transaction and has been prompted by the system to enter its information.
Post-condition: All of the information has been entered into the transaction



Contract: Display Info
Operation: displayInfo(Transaction: transaction)
Cross-References: Use Cases: Remove Income, Remove Expense
Pre-condition: User wants to see information and settings for a given transaction
Post-condition: Displays information and settings for a given transaction

Contract: Remove Transaction
Operation: removeTransaction(Transaction: transaction)
Cross-References: Use Cases: Remove Income, Remove Expense
Pre-condition: User wants to remove a transaction from their list of transactions
Post-condition: The system has removed the given transaction from the list of transactions

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

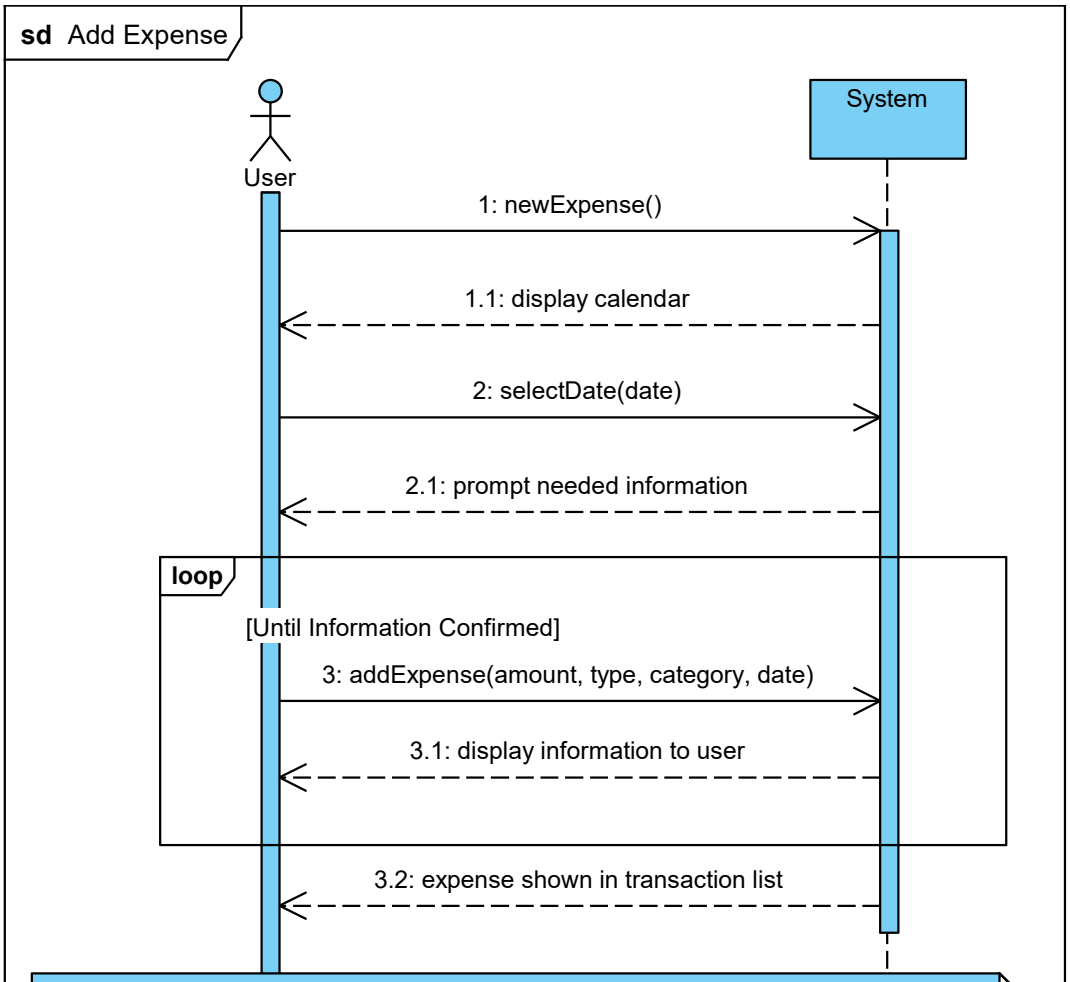
1. The user will restart the system

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



Contract: New Expense
Operation: newExpense()
Cross References: Use cases: Add Expense
Pre-conditions: User wants to add a new expense to their transaction history
Post-condition: Transaction shown in transaction list and included in calculations

Contract: Select Date
Operation: selectDate(date: Date)
Cross References: Use cases: Add Expense, Add Income
Pre-conditions: User needs to select a date for another operation
Post-condition: Date will be set for operation

Contract: Add Expense
Operation: addExpense(amount: Float, type: String, category: String, date: Date)
Cross References: Use cases: Add Expense
Pre-conditions: User wants to add a new expense to their transaction history
Post-condition: Transaction shown in transaction list and included in

ID: Add Expense

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 system will prompt the user for the expense amount and source
5. The user will enter all of the requested information
6. The system will prompt the user to confirm the information
7. The user will confirm the information
8. The system will add the expense to the transaction list

Extensions

a*. The system stops functioning

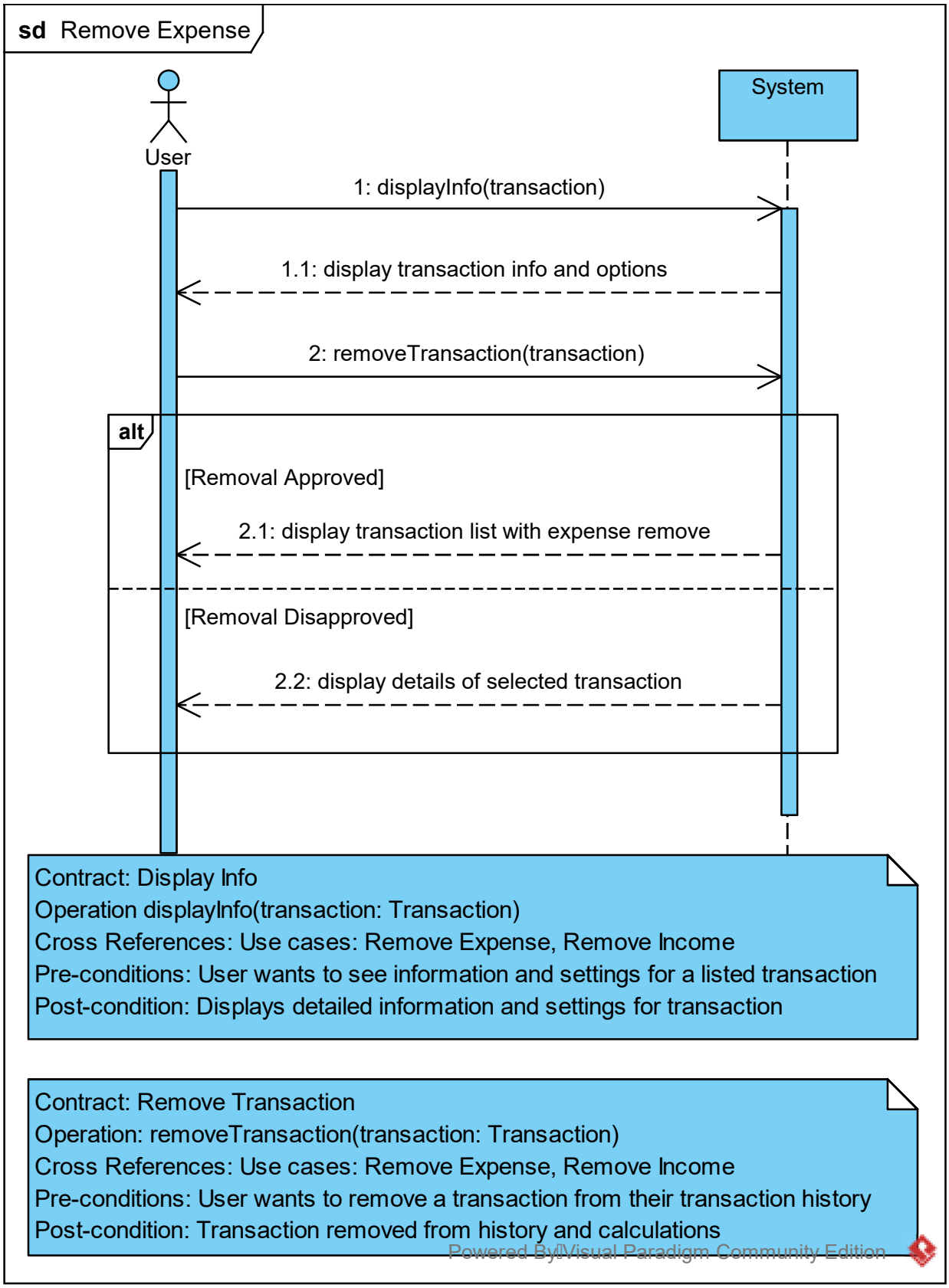
1. The user will restart the system

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 reenter the information


sd Remove Expense
User
System
1: displayInfo(transaction)
1.1: display transaction info and options
2: removeTransaction(transaction)
alt
[Removal Approved]
2.1: display transaction list with expense remove
[Removal Disapproved]
2.2: display details of selected transaction
Contract: Display Info
Operation displayInfo(transaction: Transaction)
Cross References: Use cases: Remove Expense, Remove Income
Pre-conditions: User wants to see information and settings for a listed transaction
Post-condition: Displays detailed information and settings for transaction
Contract: Remove Transaction
Operation: removeTransaction(transaction: Transaction)
Cross References: Use cases: Remove Expense, Remove Income
Pre-conditions: User wants to remove a transaction from their transaction history
Post-condition: Transaction removed from history and calculations
Powered By Visual Paradigm Community Edition

ID: Remove Expense

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

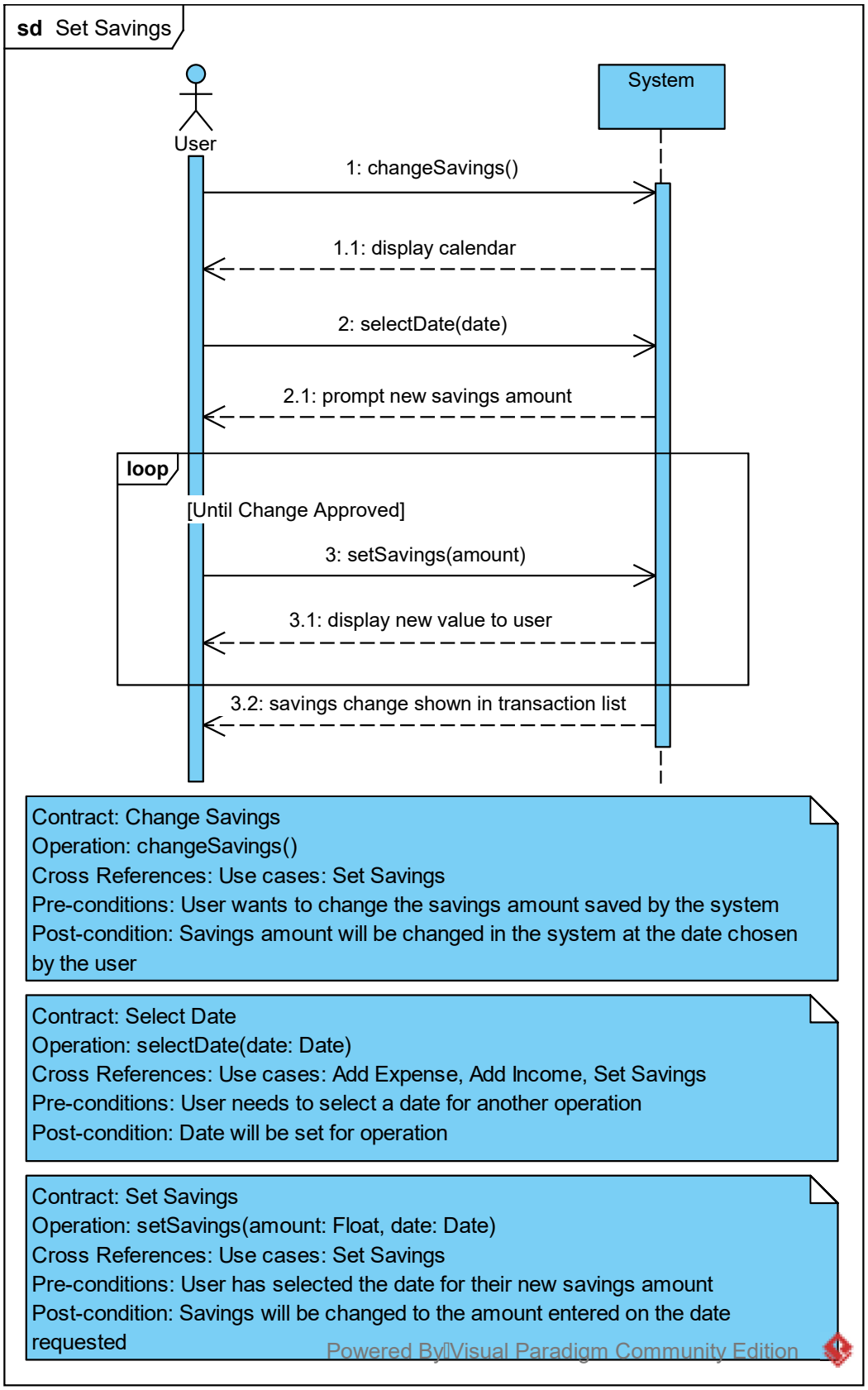
1. The user will restart the system

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


sd Set Savings
User
System
1: changeSavings()
1.1: display calendar
2: selectDate(date)
2.1: prompt new savings amount
loop
[Until Change Approved]
3: setSavings(amount)
3.1: display new value to user
3.2: savings change shown in transaction list
Contract: Change Savings
Operation: changeSavings()
Cross References: Use cases: Set Savings
Pre-conditions: User wants to change the savings amount saved by the system
Post-condition: Savings amount will be changed in the system at the date chosen by the user
Contract: Select Date
Operation: selectDate(date: Date)
Cross References: Use cases: Add Expense, Add Income, Set Savings
Pre-conditions: User needs to select a date for another operation
Post-condition: Date will be set for operation
Contract: Set Savings
Operation: setSavings(amount: Float, date: Date)
Cross References: Use cases: Set Savings
Pre-conditions: User has selected the date for their new savings amount
Post-condition: Savings will be changed to the amount entered on the date requested
Powered By Visual Paradigm Community Edition

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

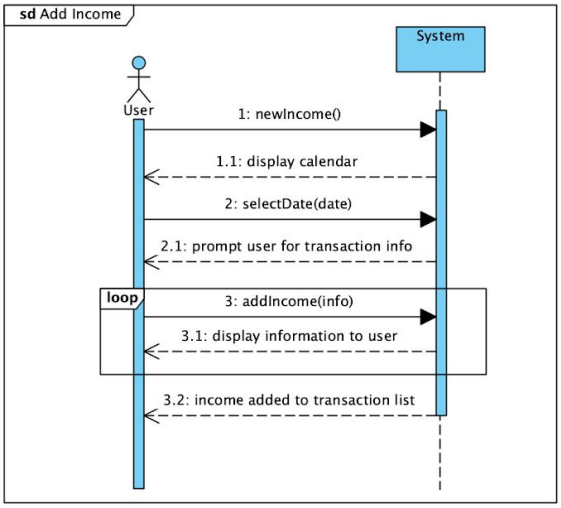
6.a If the user selects a recurring payment

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*



Contract: New Income
Operation: newIncome()
Cross-References: Use Cases: Add Income
Pre-condition: User wants to make a new income transaction.
Post-condition: The system has created a new income from the user's specifications.

Contract: Select Date
Operation: selectDate(Date: date)
Cross-References: Use Cases: Add Income, Add Expense
Pre-condition: User wants to create a transaction and the system has prompted the user for the date
Post-condition: The date of transaction has been selected and associated with the transaction

Contract: Add Income Info
Operation: addIncome(String: info)
Cross-References: Use Cases: Add Income
Pre-condition: User is creating a new income transaction and has been prompted by the system to enter its information.
Post-condition: All of the information has been entered into the transaction

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

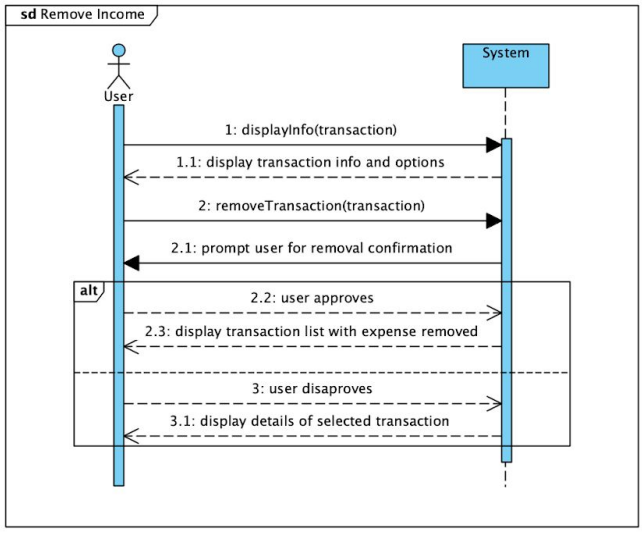
1. *User will restart the application*

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*



Contract: Display Info
Operation: displayInfo(Transaction: transaction)
Cross-References: Use Cases: Remove Income, Remove Expense
Pre-condition: User wants to see information and settings for a given transaction
Post-condition: Displays information and settings for a given transaction

Contract: Remove Transaction
Operation: removeTransaction(Transaction: transaction)
Cross-References: Use Cases: Remove Income, Remove Expense
Pre-condition: User wants to remove a transaction from their list of transactions
Post-condition: The system has removed the given transaction from the list of transactions

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

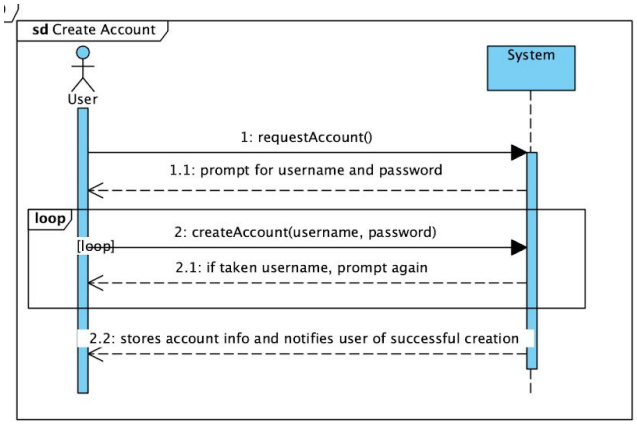
1. *User will restart the application*

1.a User already has an account

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

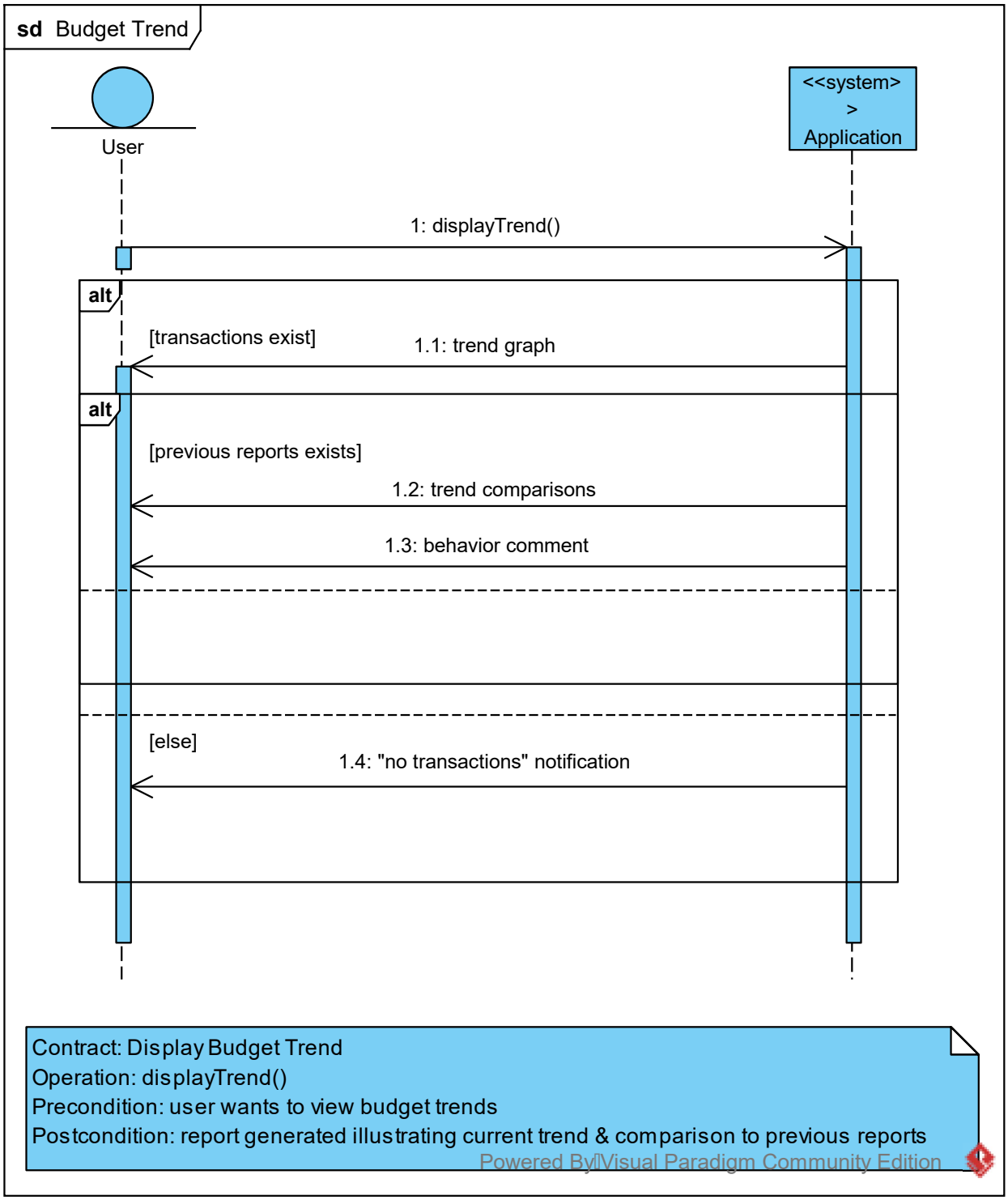
5.* The given username is taken

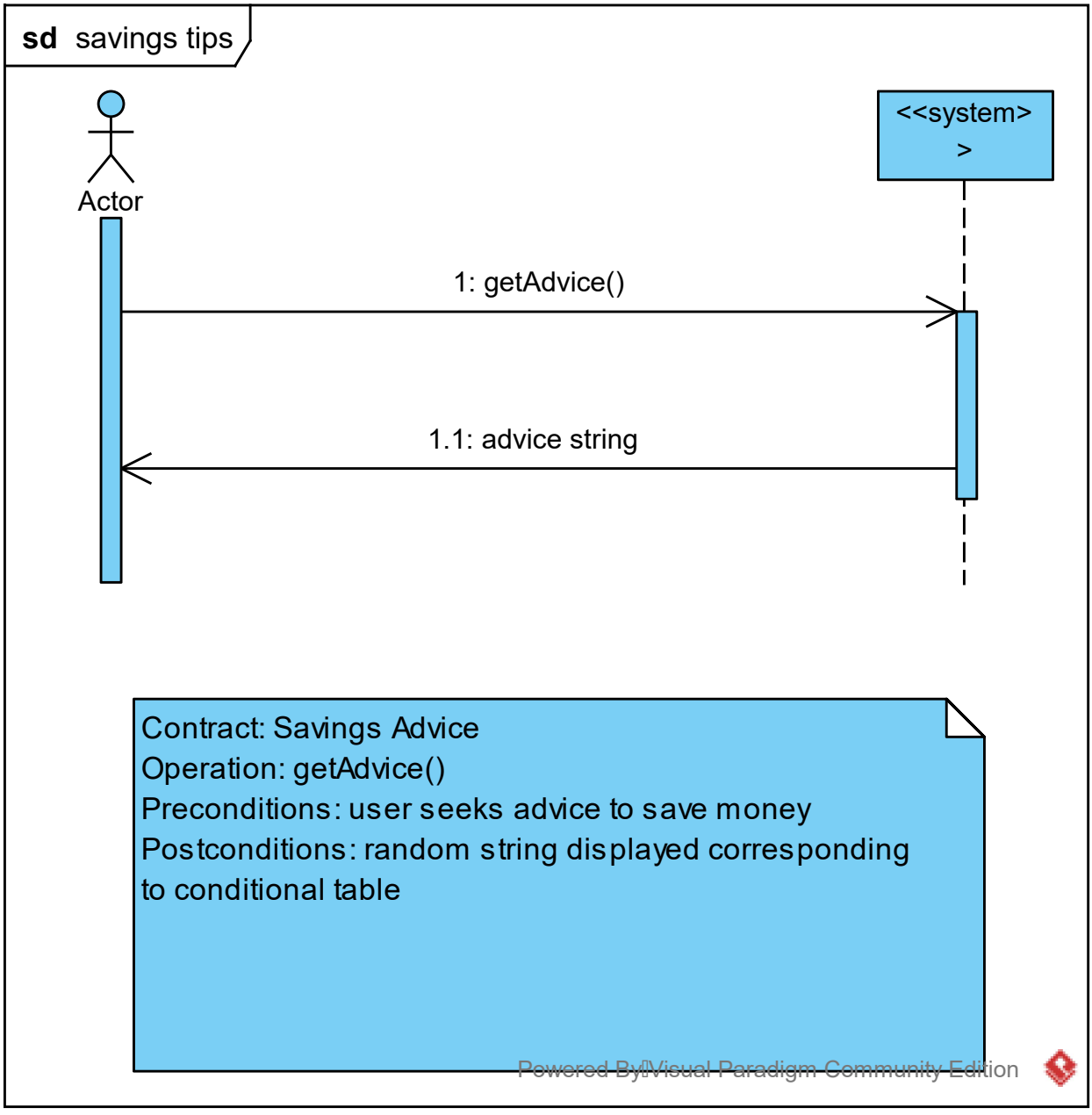
1. *System informs the user that the provided username is already taken*
2. *System prompts user again for a new username*

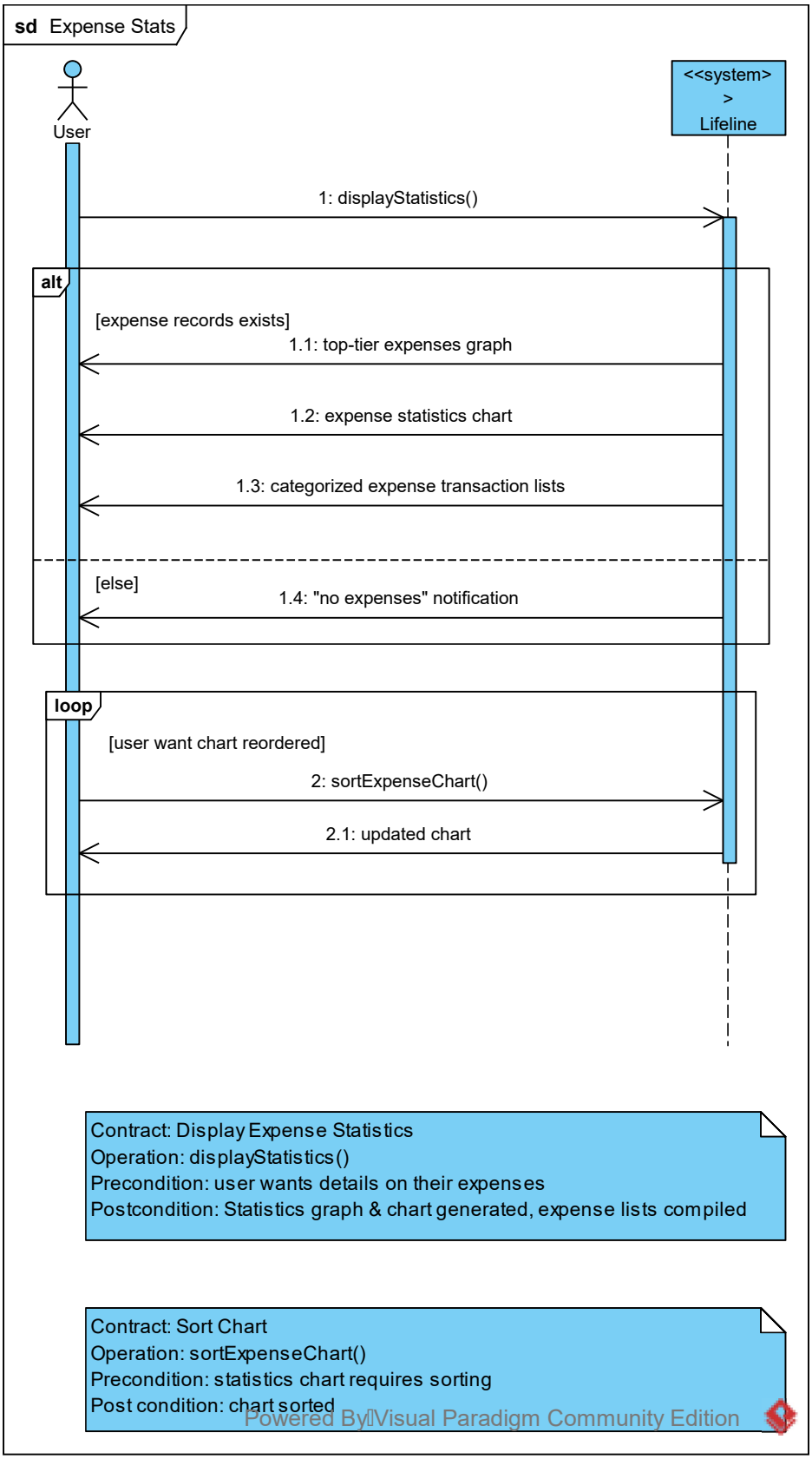


Contract: Request Account
Operation: requestAccount()
Cross-References: Use Cases: Create Account
Pre-condition: User would like to create a new account
Post-condition: User begins the account creation process

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


sd Budget Trend
User
<<system>>
Application
1: displayTrend()
alt
[transactions exist]
1.1: trend graph
alt
[previous reports exists]
1.2: trend comparisons
1.3: behavior comment
[else]
1.4: "no transactions" notification
Contract: Display Budget Trend
Operation: displayTrend()
Precondition: user wants to view budget trends
Postcondition: report generated illustrating current trend & comparison to previous reports
Powered By Visual Paradigm Community Edition


sd savings tips
Actor
<<system>>
1: getAdvice()
1.1: advice string
Contract: Savings Advice
Operation: getAdvice()
Preconditions: user seeks advice to save money
Postconditions: random string displayed corresponding to conditional table
Powered By Visual Paradigm Community Edition


sd Expense Stats
User
<<system>>
Lifeline
1: displayStatistics()
alt
[expense records exists]
1.1: top-tier expenses graph
1.2: expense statistics chart
1.3: categorized expense transaction lists
[else]
1.4: "no expenses" notification
loop
[user want chart reordered]
2: sortExpenseChart()
2.1: updated chart
Contract: Display Expense Statistics
Operation: displayStatistics()
Precondition: user wants details on their expenses
Postcondition: Statistics graph & chart generated, expense lists compiled
Contract: Sort Chart
Operation: sortExpenseChart()
Precondition: statistics chart requires sorting
Post condition: chart sorted
Powered By Visual Paradigm Community Edition

author: will henderson

Budget Trend Analysis

Scope: Personal Budgeting App
Level: user-goal
Primary Actor: User
Stakeholders:
- User: wants illustration of finnancial 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.

author: will henderson

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 catagories displayed.
- Chart featuring varying stats per expense category displayed.
- Lists compiled with every individual recored 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 catagory 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.

author: will henderson

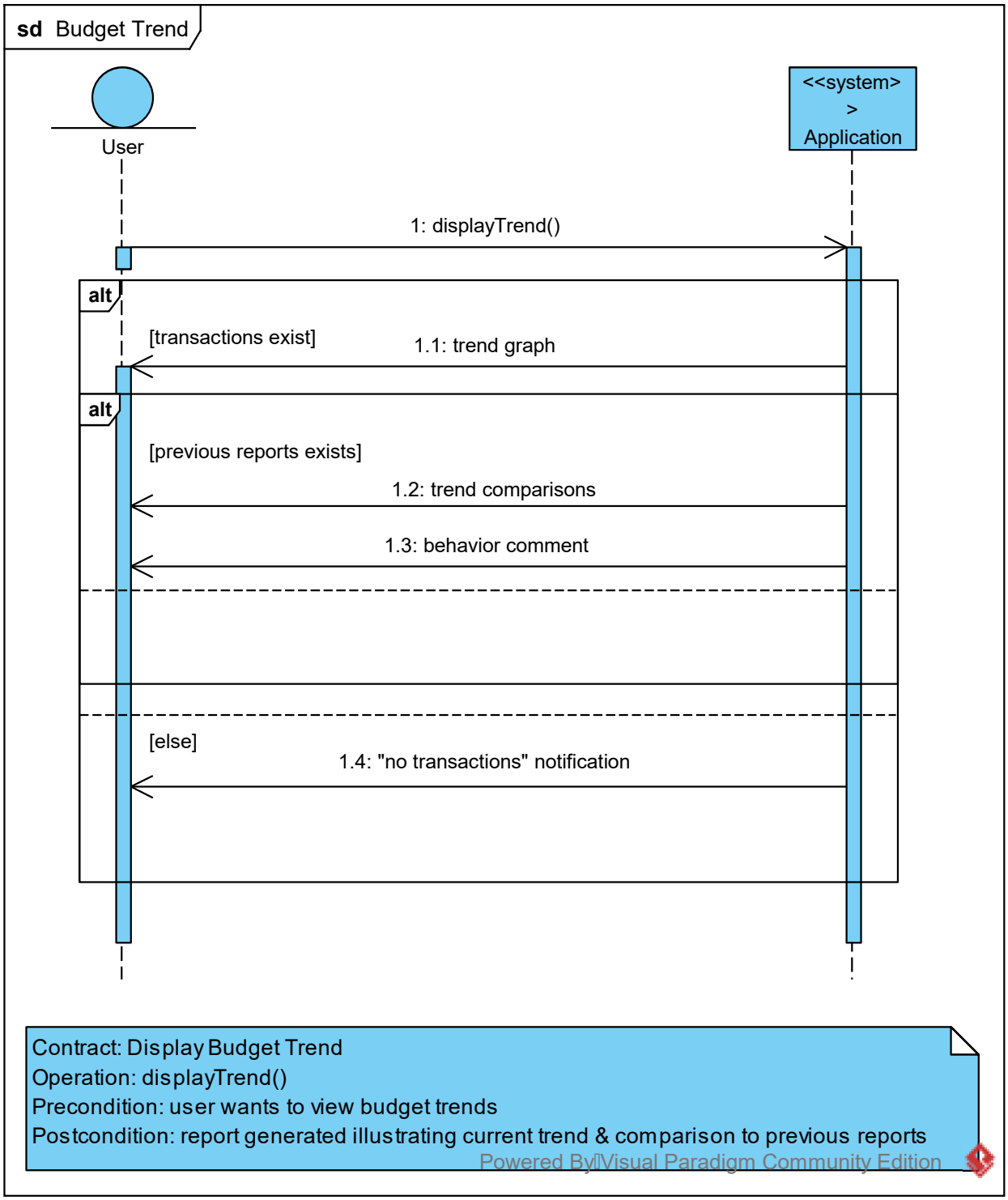
Savings Suggestions

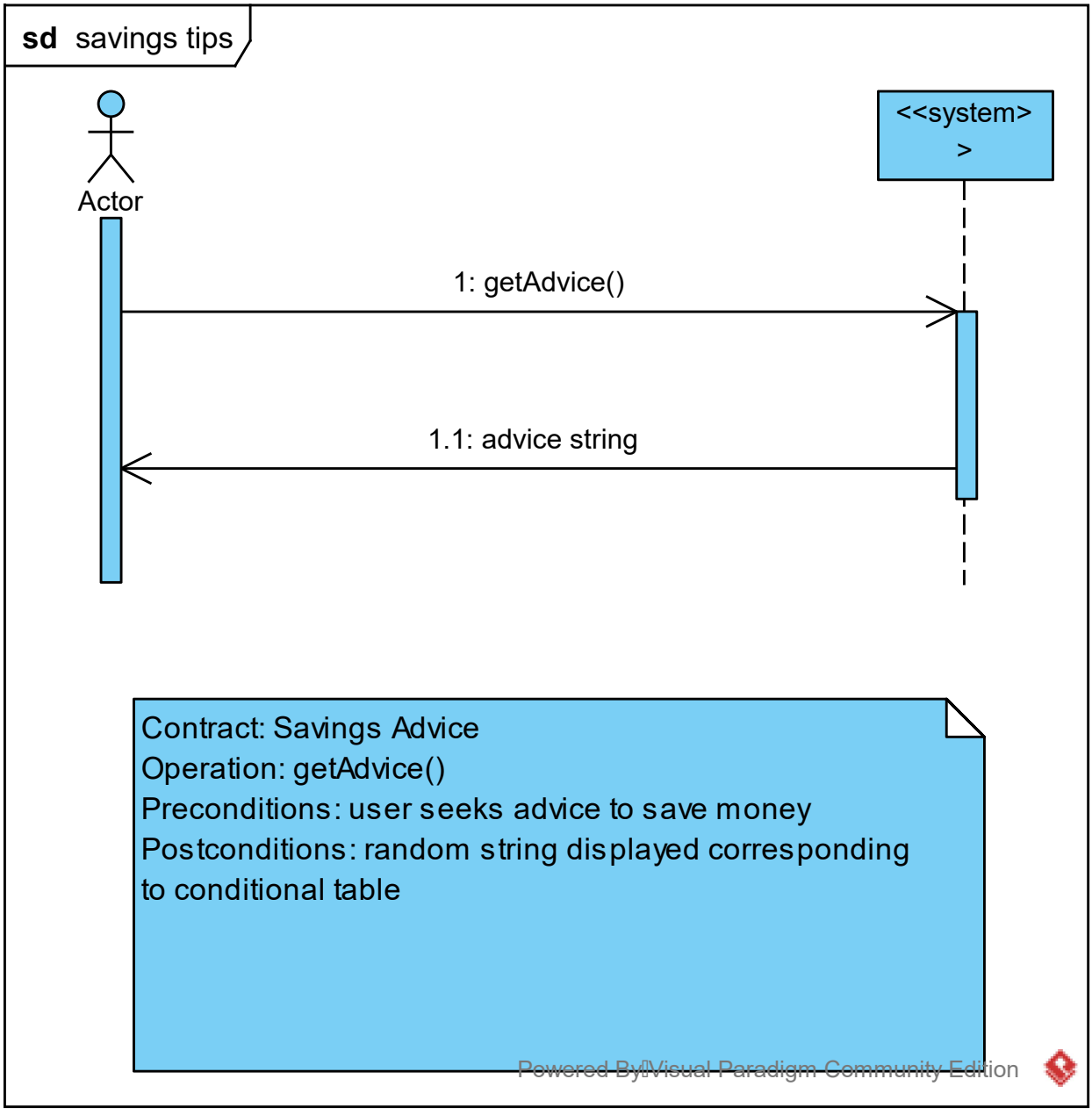
Scope: Personal Budgeting App
Level: user-goal
Primary Actor: User
Stakeholders:
- User: wants advice on saving money by controlling expenses.

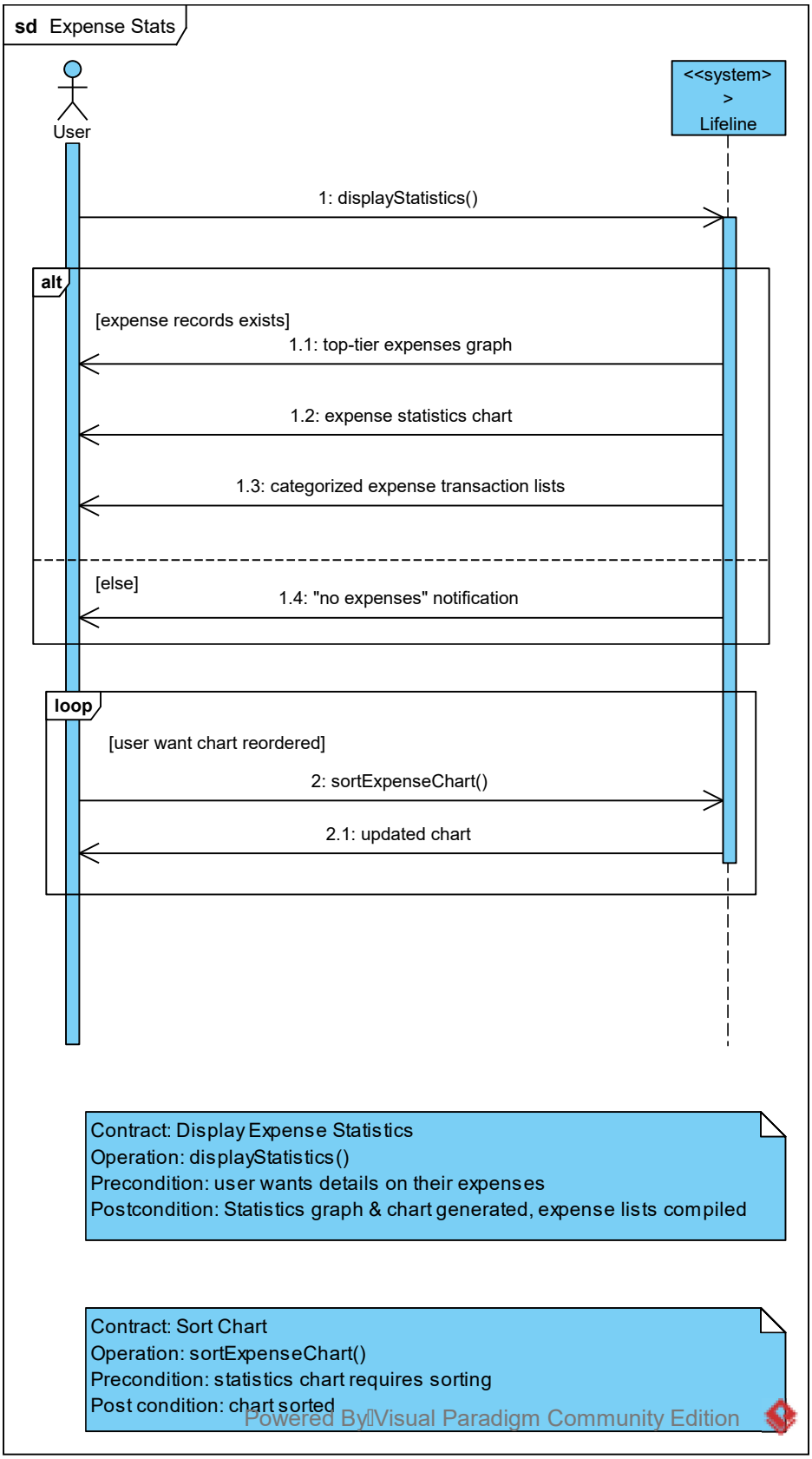
Preconditions:
- Transactions are recorded in the app.
Postconditions:
- Adivce displayed to user.
- Category-specific advice highlights related category.

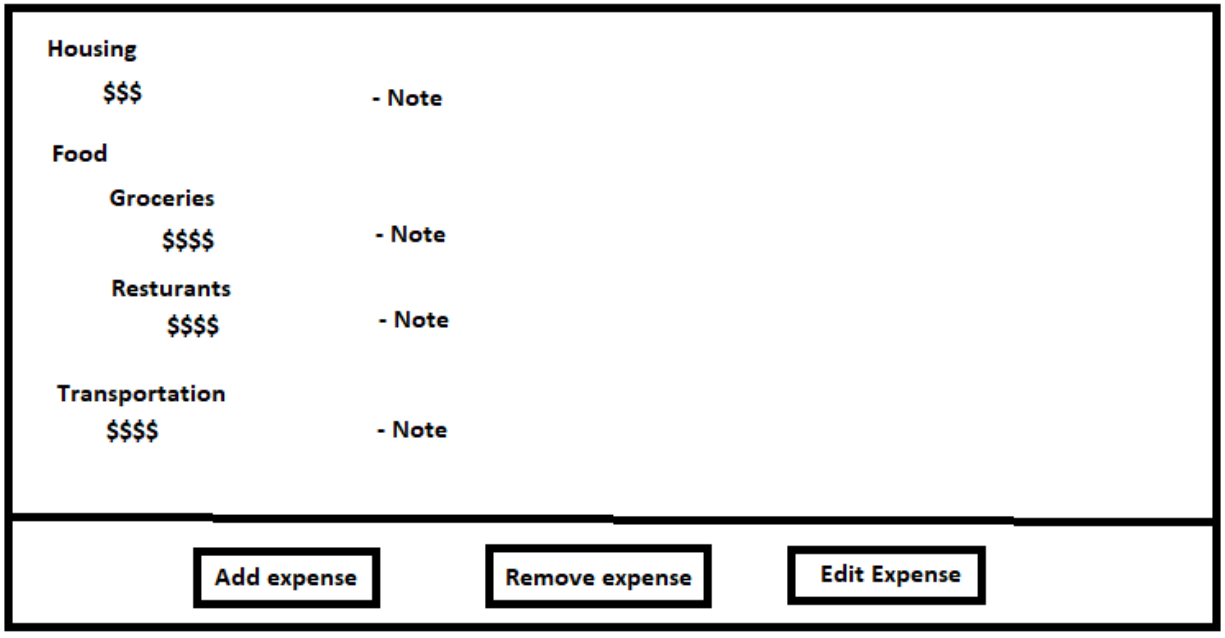
Main flow:
1. User request savings advice or triggers advice generation during other action.
2. All recent recorded transactions, statistics, and display context are applied to an advice rule table.
3. Localized advice string displayed to user from rule table.

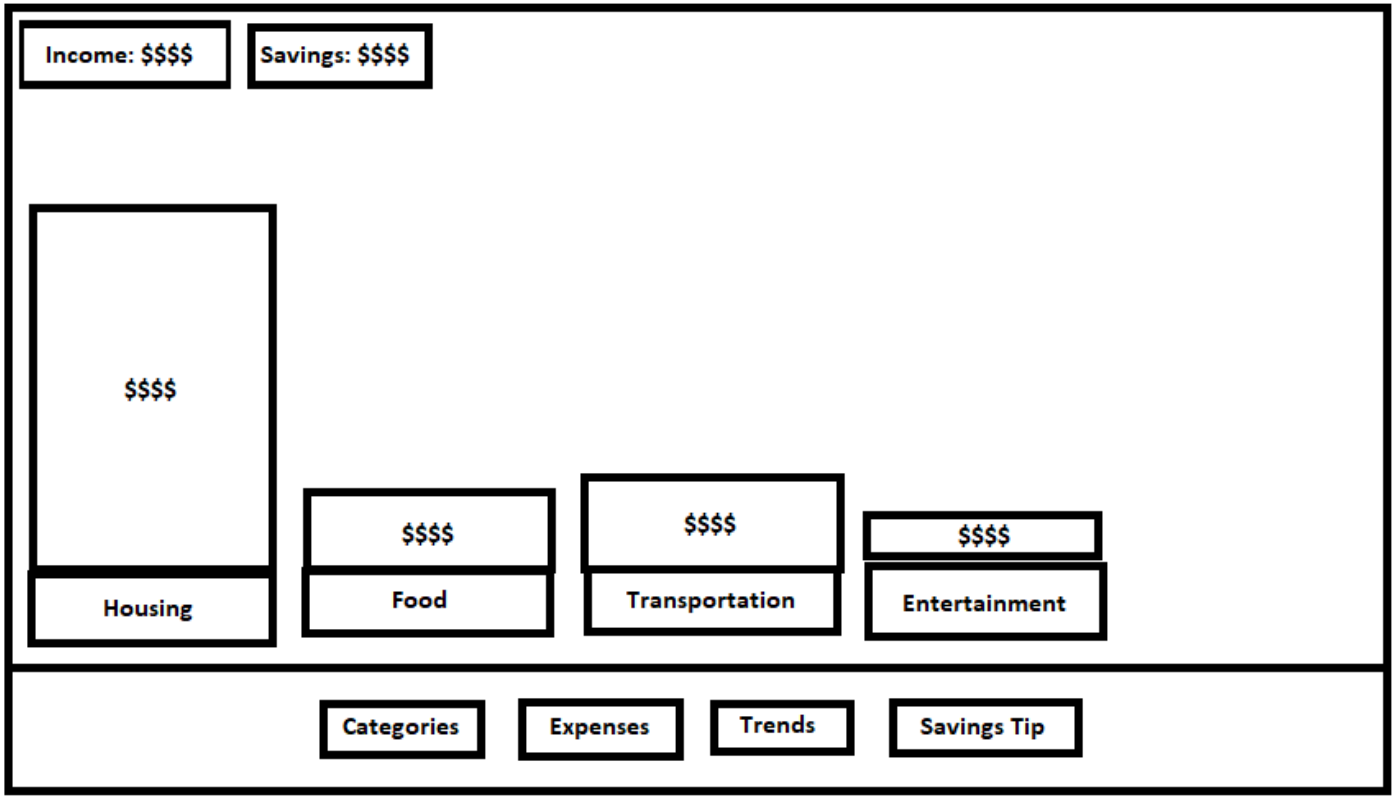
Alternate flow:
2a. No table entry matches the current conditions.
    1. No advice is displayed
2b. Multiple entries match current conditions.
    1. Select an advice string at random.


sd Budget Trend
User
<<system>>
Application
1: displayTrend()
alt
[transactions exist]
1.1: trend graph
alt
[previous reports exists]
1.2: trend comparisons
1.3: behavior comment
[else]
1.4: "no transactions" notification
Contract: Display Budget Trend
Operation: displayTrend()
Precondition: user wants to view budget trends
Postcondition: report generated illustrating current trend & comparison to previous reports
Powered By Visual Paradigm Community Edition

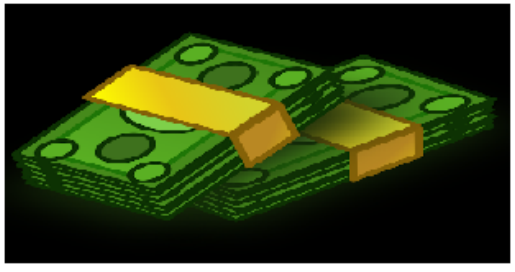

sd savings tips
Actor
<<system>>
1: getAdvice()
1.1: advice string
Contract: Savings Advice
Operation: getAdvice()
Preconditions: user seeks advice to save money
Postconditions: random string displayed corresponding to conditional table
Powered By Visual Paradigm Community Edition


sd Expense Stats
User
<<system>>
Lifeline
1: displayStatistics()
alt
[expense records exists]
1.1: top-tier expenses graph
1.2: expense statistics chart
1.3: categorized expense transaction lists
[else]
1.4: "no expenses" notification
loop
[user want chart reordered]
2: sortExpenseChart()
2.1: updated chart
Contract: Display Expense Statistics
Operation: displayStatistics()
Precondition: user wants details on their expenses
Postcondition: Statistics graph & chart generated, expense lists compiled
Contract: Sort Chart
Operation: sortExpenseChart()
Precondition: statistics chart requires sorting
Post condition: chart sorted
Powered By Visual Paradigm Community Edition


Housing
$$$
- Note
Food
Groceries
$$$$
- Note
Resturants
$$$$
- Note
Transportation
$$$$
- Note
Add expense
Remove expense
Edit Expense


Income: $$$$
Savings: $$$$
$$$$
Housing
$$$$
Food
$$$$
Transportation
$$$$
Entertainment
Categories
Expenses
Trends
Savings Tip

Bear Budget



Username

Password

Create an Account