A green background with a logo

Description automatically generated

**Split & Settle**

**Debt Simplifier**

Team 6

Ayush Sheth, Dhriti Vivek, Ahmed Elghazi, Nigel Kudowor,

Benjamin Mejia, Huzaifa Nabeel, Dat Nguyen, Veer Waje

Delivered to

Professor Srimathi Srinivasan

10 March 2024

**Table of Contents**

Homepage 1

Table of Contents 2

Introduction of Document 3

Purpose of this Document 3

Audience 3

Structure 3

Requirements Summary 3

Requirements 4

Functional Requirements 4

Non-Functional Requirements 5

Use Cases 5

Architecture Design 7

UML 10

Site Map 11

System Modeling 12

Wire Frames 24

Implementation 28

Software Testing 29

Software Evolution 37

Glossary of terms 37

Use Case Traceability Matrix 38

**Introduction of Document**

Purpose of this Document:

This document's purpose is to generate a thorough documentation of how Split and Settle was created. In the case that Split and Settle require additional changes to be made by a third party, this document should be able to clarify exactly how this application was implemented so that changes can be smoothly made. Another purpose of this document is to create a fully formed user manual. This document should have various levels of insight, outlined in the table of contents, so creating a user manual is simply a matter of translating the information to a user-understandable level.

Audience:

The targeted audience for this application is users who wish to have a more streamlined cost-splitting service. This niche lies especially within college students, where shared living and expenses may often create the extra task of calculating debts and settling them. Other users may also lie outside of college students and can range from all ages and demographics, as long as there is a necessity to eliminate the extra effort needed to calculate costs.

Structure:

The structure of Split and Settle surrounds interactions between a user, their groups, and their friends. A user has an interface through which they can connect with other users by friending or creating groups. There exists a set of interactions a user can have with other users. Each user also has a set of actions they can take to personalize their end, not dealing with other users. The structure of this application has been decided to best minimize the hurdle of splitting costs the traditional way and offers an elegant solution to this problem.

Requirements:

The requirement process was divided into functional and non-functional requirements. We then started brainstorming how the application would start up with simple requirements, such as creating an account, and what the user would be able to modify from their account. Afterward, we moved on to how the user would use the app—adding or deleting simple transactions and how the user could invite and connect with other people. From there, we kept expanding onto features and the mechanics of the app, listing the necessary requirements into functional and if it was not necessary, we moved it onto non-functional.

**Requirements:**

Functional Requirements  
  
The Users can:

1. (Ayush) create an account.

2. (Ayush) log into the account.

3. (Ayush) reset password if the person can’t login.

4. (Ayush) change password by entering new and current password at same time if a person regularly updates his/her passwords.

5. (Ayush) view and edit account details like email, phone number and name.

6. (Veer) “settle” costs with a specific person or with a specific group.

7. (Veer) be in a maximum of 10 groups.

8. (Ahmed) should be able to create groups.

9. (Ahmed) only leave a group if that specific person has all debts cleared in that group.

10. (Ahmed) allow users to divide expenses in 3 ways – equal, exact, percentage.

11. (Ahmed) upload pictures of receipt in the transaction at the same time of creating transaction or later as well.

12. (Ahmed) see the date and time for transactions posted, and when settlements are cleared.

13. (Nigel) add other valid users into groups.

14. (Nigel) see which user owes how much to each user.

15. (Dhriti) view activity in an activity tab upon their debts being paid or settled.

16. (Dhriti) pay in full or partial, and this can be recorded by both persons (the guy who owes the money, and the guy being owed the money)

The App:

17. (Veer) should allow all users to see all user's transaction history and settlement history.

18. (Veer) should allow user to delete account only if their debts have been settled.

19. (Veer) should allow all users to add/delete/update expenses including name and amount.

20. (Nigel) allows users to split an expense in a group or individually

21. (Ben) should allow users to contact admin in case any error happens.

Miscellaneous:

22. (Nigel) Person 1 cannot remove Person 2 from the same group if Person 2 has unsettled debt(s).

23. (Nigel) A whiteboard should be there for each group to add important details like payment date, emergency contact, etc.

24. (Huzaifa) Users should be able to categorize expenses (rent, food, subscription, etc.)

25. (Huzaifa) The homepage should have separate tabs for friends, groups, transaction history and account profile.

26. (Huzaifa) There should be an add an expense button in the bottom center of the homepage, and user can select the friend or group to add the expense to.

27. (Dhriti) The homepage should summarize which group owes how much money in total and should list it out individually as well.

28. (Dhriti) On the top of the homepage, there should be a summary of expenses if user owes or is being owed across friends and groups in total.

29. (Dhriti) The friends page on homepage should summarize each friend’s debts overall across groups and individually.

30. (Ben) Response time should be below 5s for displaying anything.

Non-Functional Requirements

1. (Ben) App should have transactions thoroughly detailed

2. (Ben) Be able to display all past transactions with no data limit

3. (Ben) App should be secured from unauthorized users.

4. (Huzaifa) App should be suitable for all ages.

5. (Huzaifa) UI should be easy to navigate and visually appealing.

6. (Dat) App should be available on any browser.

7. (Dat) App should be able to handle a growing number of users and volume of transactions.

8. (Dat) Error messages are clear and easy to understand for users.

9. (Dat) App should obtain user consent for permission to handle their data.

10. (Dat) App should be accessible in an ‘online’ mode

**Use Cases:**

TUCBW: The use case begins when, TUCEW: The use case ends when

1. Expense settlement (Huzaifa) TAKEN

TUCBW: The user clicks on “settle” button inside a group and chooses the person he wishes to pay.

TUCEW: The user enters a number (it can be whole or part of the debt he owes), and then he clicks on “Pay” and all balances and activity tab are updated accordingly.

1. Navigate to Activity Tab on Homepage (Dhriti) TAKEN

TUCBW: User clicks on “Activity” tab on

TUCEW: User can see all activity including date and time when an expense was added or deleted, what amount but can only see transactions involving them.

1. Viewing all debts (Nigel) TAKEN

TUCBW: User clicks on. “Groups”.

TUCEW: The system shows how much a person owes or is being owed by the other person for every member in every group, including friends.

1. Delete an expense (Ahmed) TAKEN

TUCBW: User clicks on an already existing expense.

TUCEW: User clicks on delete expense button and the expense is deleted.

1. Create a group (Ayush) TAKEN

TUCBW: User clicks on create a group button.

TUCEW: User creates group after naming it and inviting people.

1. Add an expense (Veer) TAKEN

TUCBW: User clicks on “+” button located in bottom center of homepage or upper right corner in a group.

TUCEW: User can add item description, amount, select the group or friend to split the expense with, add an image of receipt(optional), add the expense will be settled equally.

1. Viewing Transactions Within an Individual Group or Friend (Ben) TAKEN

TUCBW: User clicks on the desired group or friend on homepage.

TUCEW: All transactions a user is involved in between the group or friend is shown.

1. Edit an expense (DAT) TAKEN

TUCBW: User clicks on “edit expense” button

TUCEW: User views their updated expense

1. Invite a friend button (Ahmed)

TUCBW: User clicks on invite friend button.

TUCEW: The user invites a friend using his/her phone number or email.

1. App User Authentication (Ayush)

TUCBW: The user enters login ID and password.  
 TUCEW: The system checks in database if it is a valid input, and if it is, the account is logged in, otherwise “Incorrect details” is displayed.

1. Account Creation for User (Ayush)

TUCBW: The user clicks on create account, and enters name, email, phone number and password to create.  
 TUCEW: If email or phone number already exists in the system, display “Account details already in use” and give a link to reset password. Otherwise, create an account and display the homepage.

1. Edit Account Details Tab on Homepage (Veer)

TUCBW: User clicks on “Account” and account details pop up.

TUCEW: The user can edit/update their account email, name, password, or phone number.

1. Delete account (Veer)

TUCBW: User clicks on “Delete Account” in “Account”.

TUCEW: The user can only delete the account if all debts with friends and groups have been settled, otherwise an error message pops up “You have outstanding debts with one or more friend/s or group/s.

1. Friends Tab on Homepage (Dat)

TUCBW: User clicks on “Friends” on homepage.

TUCEW: User can see how much each friend owes in total, whether it might be across groups or individually. For example, A owes B $10 personally (individually), $40 for electricity in home bill (Home Group) and $50 for food (Travel Group), so that page would show A owes B total $100.

1. Report problem (Dhriti)

TUCBW: User clicks report in settings.

TUCEW: User sends problem descriptions, and that message is sent to support team.

1. Whiteboard Tab in a group (Nigel)

TUCBW: User clicks on “Whiteboard”.

TUCEW: The whiteboard shows details if already entered (like emergency contact, or apt number, etc.), or asks the user to enter something if the user wants to.

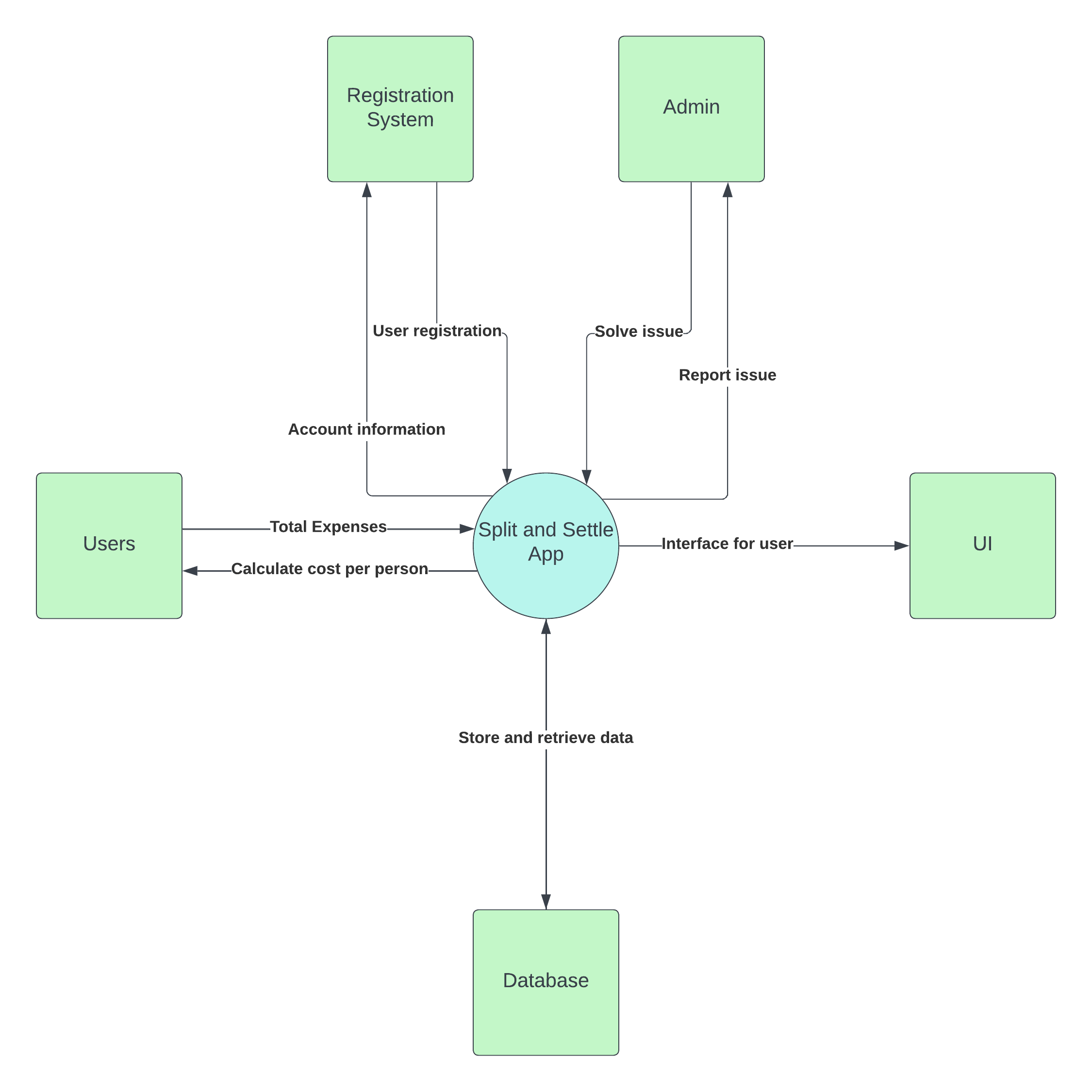
1. Group Settings (Ben)

TUCBW: User clicks on settings on upper right of a group.

TUCEW: Settings shows total amount owed or being owed by each user, and has an option to delete group or remove members. Group deletion can only happen if everyone is settled, and user can remove other user if the user being removed doesn’t have outstanding debts.

**Architecture Design:**

Context Diagram (Dat and Veer)

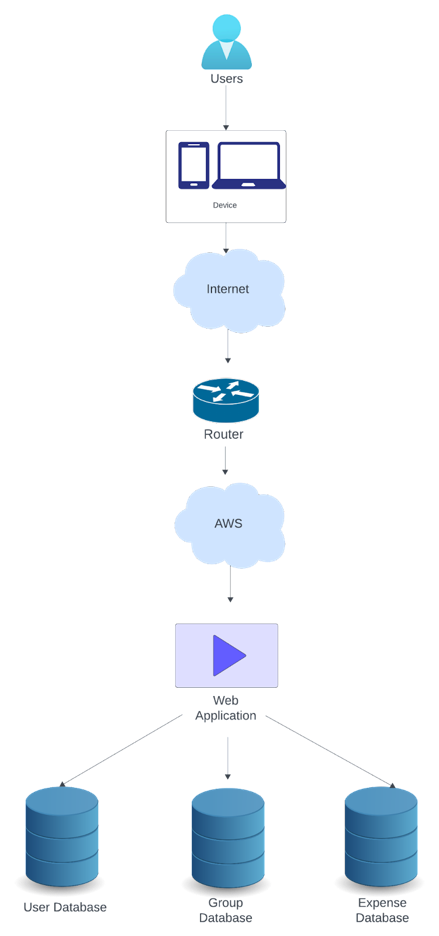


System Architecture (Logical View) - Ben, Huzaifa

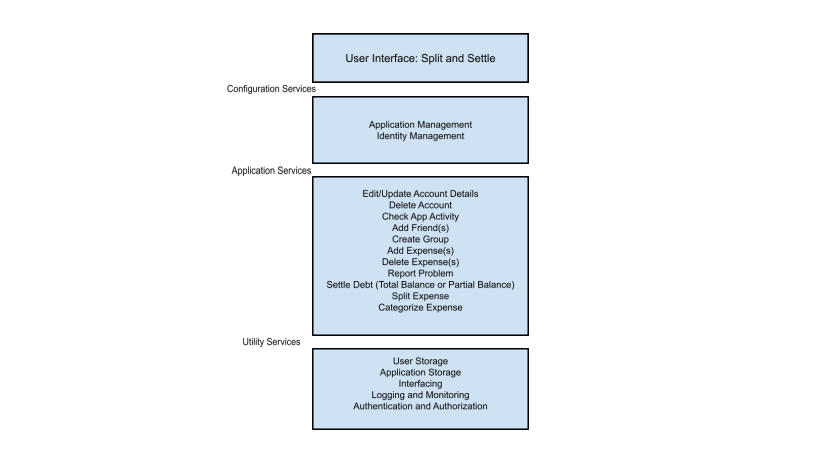
A diagram of a software

Description automatically generated

System Architecture (Physical View) - Nigel, Dhriti



Architecture Design (Layered Architecture) - Nigel



UML

A diagram of a diagram

Description automatically generated

Site Map

A diagram of a company

Description automatically generated

System Modeling

1. Expense settlement (Huzaifa)

TUCBW: The user clicks on “settle” button inside a group and chooses the person he wishes to pay.

TUCEW: The user enters a number (it can be whole or part of the debt he owes), and then he clicks on “Pay” and all balances and activity tab are updated accordingly.

High level use case:

A diagram of a software application

Description automatically generated

Expanded use case:

|  |  |
| --- | --- |
| Actor: User | System: App |
|  | 1. App displays ‘settle’ button within group page. |
| 1. TUCBW: The user clicks on “settle” button inside a group page | 1. App displays the list of group members to choose from. |
| 1. User selects group member to pay. | 1. App prompts user to enter amount to pay the selected member. |
| 1. User enters amount to pay and clicks ‘Pay’ button. | 1. System updates balances accordingly. |
| 1. TUCEW: The user successfully pays selected member, and all balances are updated accordingly. |  |

Sequence diagram:

A diagram of a system

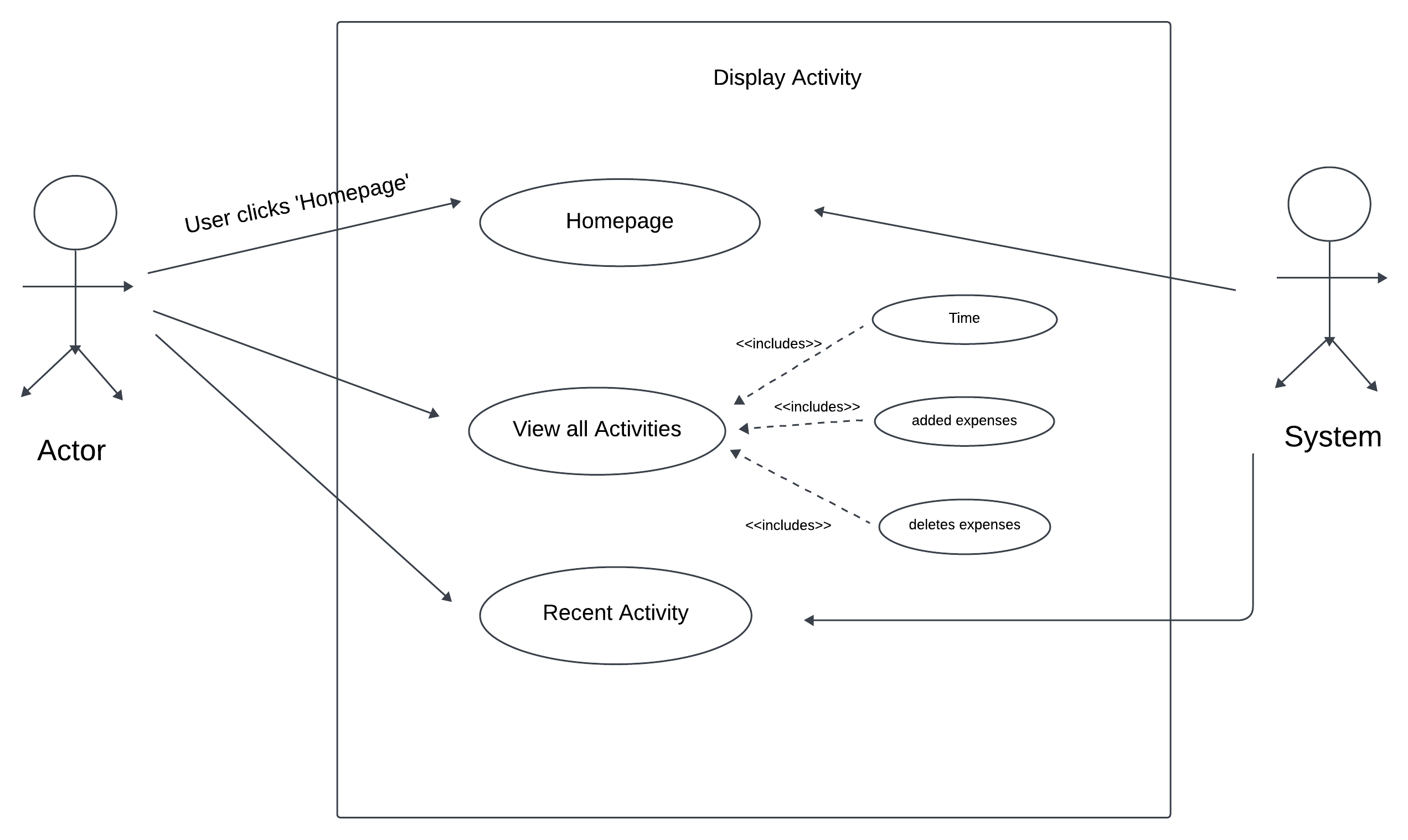
Description automatically generated

2. Navigate to Activity Tab on Homepage (Dhriti)

**High Level Use Case:**

TUCBW: User clicks on “Activity” tab on

TUCEW: User can see all activity including date and time when an expense was added or deleted, what amount but can only see transactions involving them.

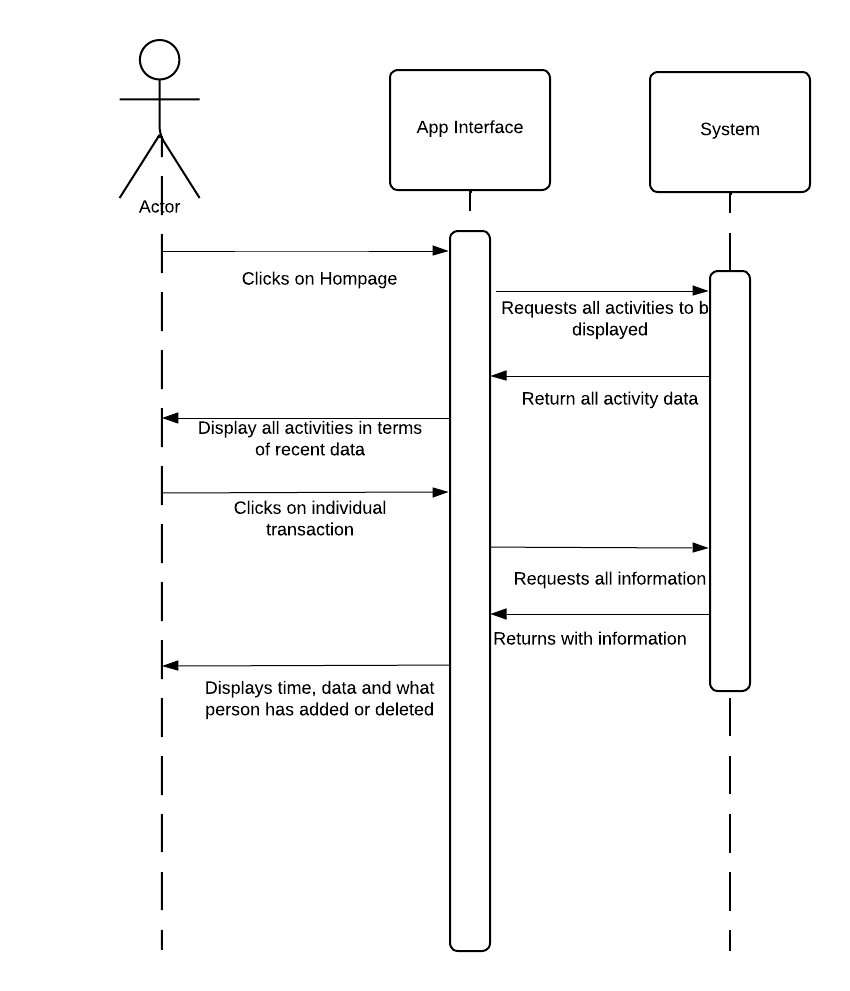


Expanded Use Case:

Precondition: The User has clicked on the Activity Tab on the Homepage to access all the transactions.

|  |  |
| --- | --- |
| Actor: User | System |
|  | 0.The System displays the homepage |
| 1.TUCBW: User clicks on “Activity” Tab | 2.The System displays the most recent activities by date and who made changes |
| 3.TUCEW: User can see all activity including date and time when an expense was added or deleted, what amount but can only see transactions involving them. |  |

Sequence Diagram:

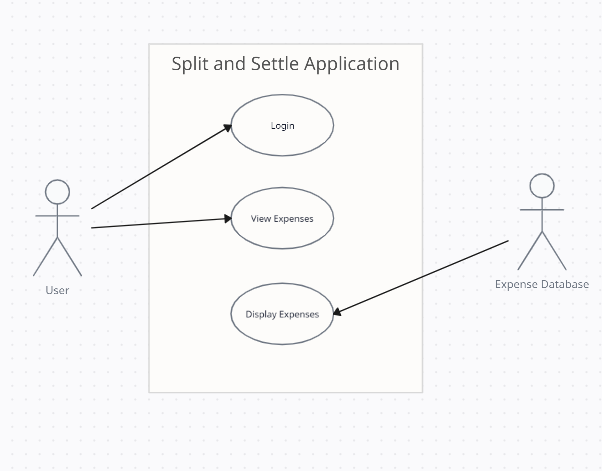


3. Viewing all debts – The user can view the debts associated with their account (Nigel)

**High Level Use Case:**

TUCBW: User clicks on. “Your Expenses”.

TUCEW: The system displays how much a person owes or is owed by other application users.

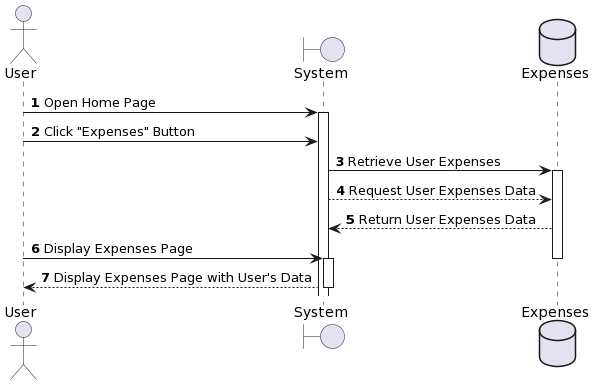


**Expanded Use Case:**

Precondition: The user has logged into the application via a valid account and is on the homepage

|  |  |
| --- | --- |
| Actor: User | System: Split and Settle app |
|  | 0. The system displays a home page with a dashboard/menu of options. |
| 1. TUCBW The user clicks on the “Expenses” button on the home page. | 2. TUCEW The system displays a new page that lists expenses associated with the user across all the user's groups/friends. This includes debts owed and debts to be paid. |

**Sequence Diagram:**



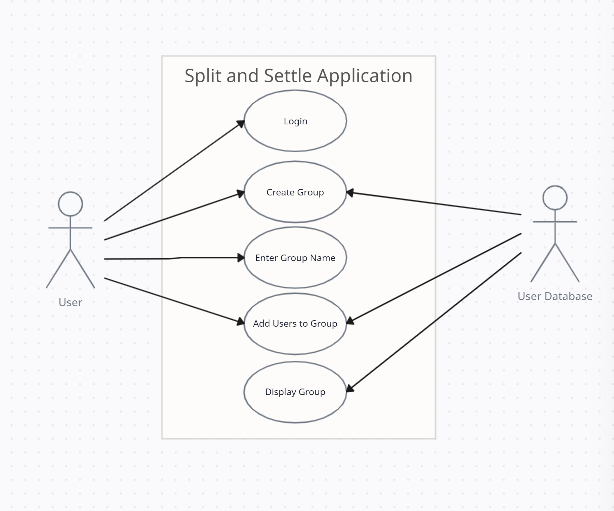
4.

5. Create a group – The user can create a group for expense settlement (Ayush)

**High Level Use Case:**

TUCBW: User clicks on the create a group button.

TUCEW: The system creates a new debt settlement group.

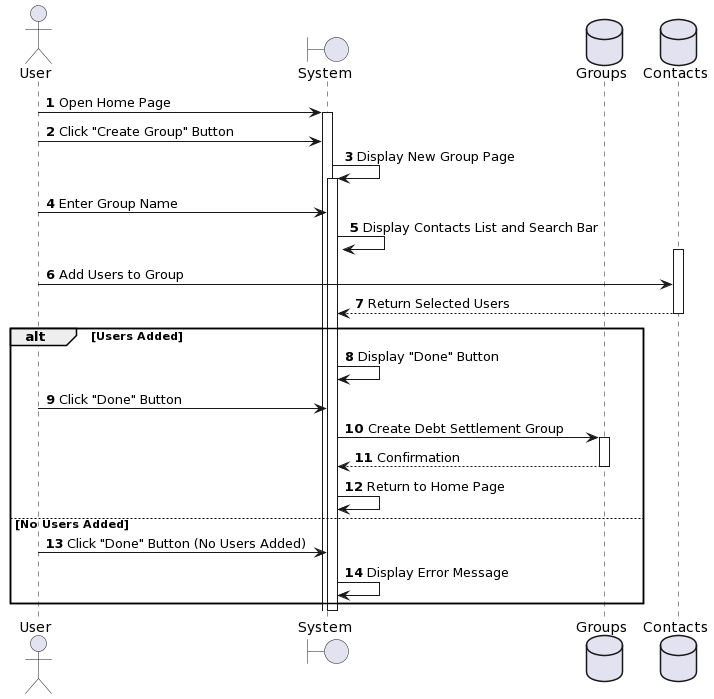


**Expanded Use Case:**

Precondition: The user has logged into the application via a valid account and is on the homepage

|  |  |
| --- | --- |
| Actor: User | System: Split and Settle app |
|  | 0. The system displays a home page with a dashboard/menu of options. |
| 1. TUCBW The user clicks on the “Create Group” button on the home page. | 2. The system displays a new page that prompts users for the name of the group. |
| 3. The user enters their desired name for the new debt settlement group. | 4. The system now displays a page that contains a list of all the user’s contacts within the app along with an additional search/text bar that allows users to add other valid users to the group by manually inputting the email associated with their account. |
| 5. The user adds users to the group, either from the contact list or by manually typing their name. | 6. The system displays a “Done” button after at least one other valid user has been selected for the new group. |
| 7. The user clicks the “Done” button. | 8. TUCEW The system creates the debt settlement group under the name that user provided and returns the user back to the home page. |

**Sequence Diagram:**

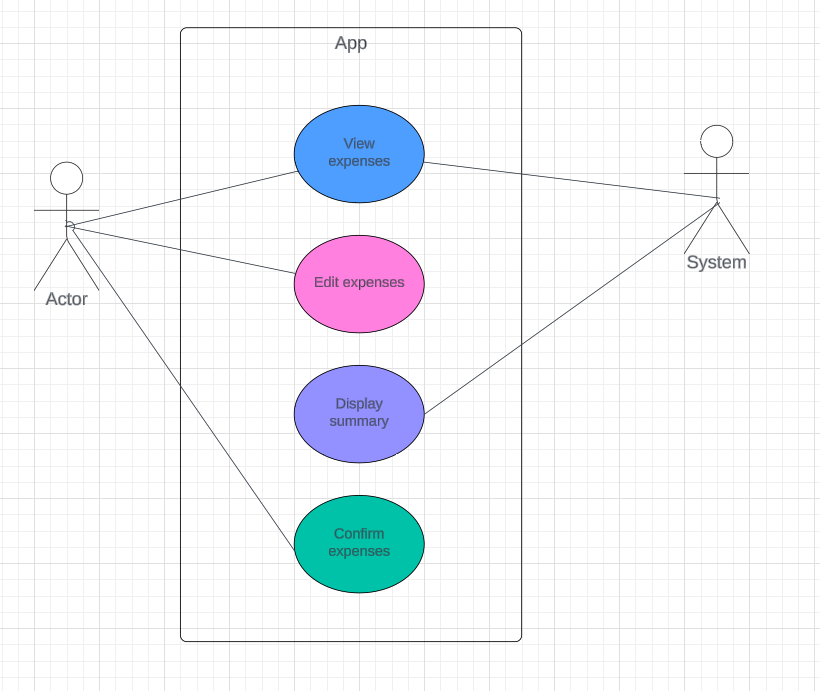


6. Edit an expense – The user can update an expense on their account (Dat)

TUCBW: User clicks on “edit expense” button

TUCEW: User views their updated expense

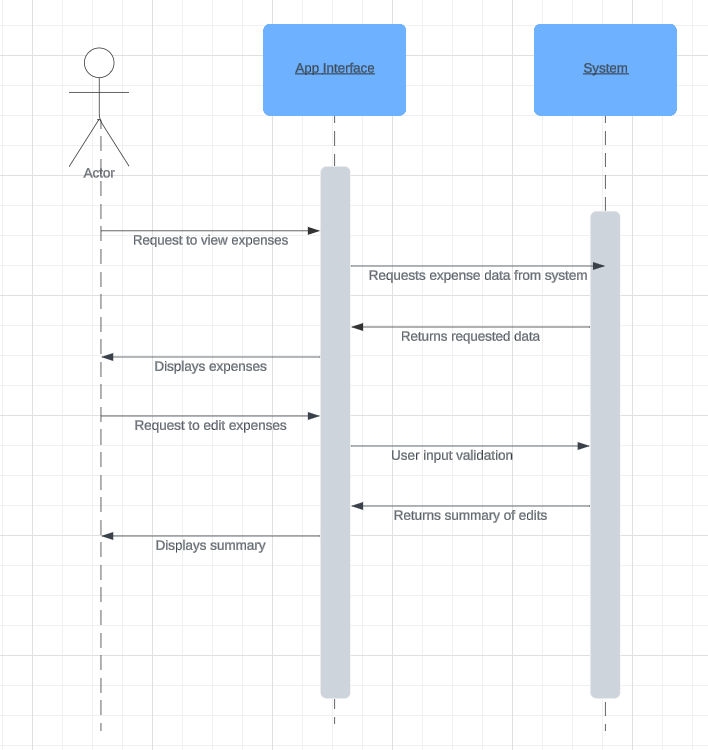
High Level Use Case:



Expanded Use Case:

|  |  |
| --- | --- |
| Actor: User | System: App |
|  | 0. App displays expenses |
| 1. TUCBW The user clicks on edit expenses button | 1. The app asks the user which expense they would like to edit |
| 1. The user clicks on the expense | 1. The app displays all possible actions to user |
| 1. The user selects on an action and makes valid inputs | 1. The app displays a summary of all the edits |
| 1. TUCEW The user clicks on the confirm button |  |

Sequence Diagram:



7. Viewing Transactions Within an Individual Group (Ben)

TUCBW: User clicks on the desired group or friend on homepage.

TUCEW: All transactions a user is involved in between the group or friend is shown

High Level Use Case:

A diagram of a diagram

Description automatically generated

Expanded Use Case:

|  |  |
| --- | --- |
| Actor: User | System: App |
|  | 1. App displays group tab |
| 1. TUCBW User clicks on spending summary within a group or friend | 1. App displays past month transactions in detail |
| 1. User chooses to view past transactions | 1. App displays all past transactions done by the user within the group |
| 1. User decides to download the transactions page | 1. App exports transactions as a pdf |
| 1. TUCEW user decides to click the return button and goes back to homepage |  |

Sequence Diagram:

A diagram of a process

Description automatically generated

1. Add an expense (Veer)

TUCBW: User clicks on “+” button located in bottom center of homepage or upper right corner in a group.

TUCEW: User can add item description, amount, select the group or friend to split the expense with, add an image of receipt(optional), add the expense will be settled equally.

|  |  |
| --- | --- |
| Actor: User | System: Split and Settle app |
|  | 1. App displays homepage |
| 1. TUCBW User clicks on plus button on the homepage | 1. App displays the add expense page, showing empty slots for description, amount, item name, and group to split with |
| 1. User fills in all the information required | 1. System displays the filled out expense page. |
| 1. User clicks “done” | 1. System uploads the expense and makes it viewable to the group. It then displays the expense information on the homepage |
| 1. TUCEW: user can see the expense they just added. |  |

Use Case diagram

A diagram of a person's process

Description automatically generated

Sequence Diagram:

A screenshot of a computer program

Description automatically generated

Wire Frames

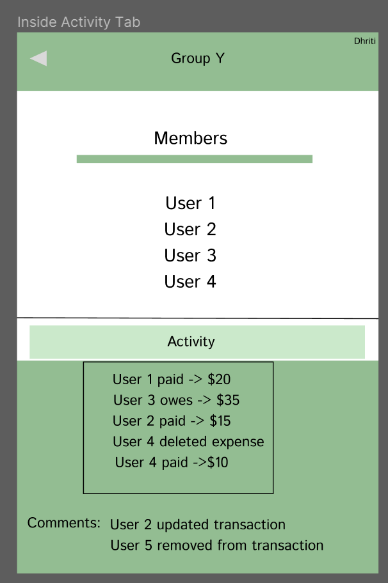
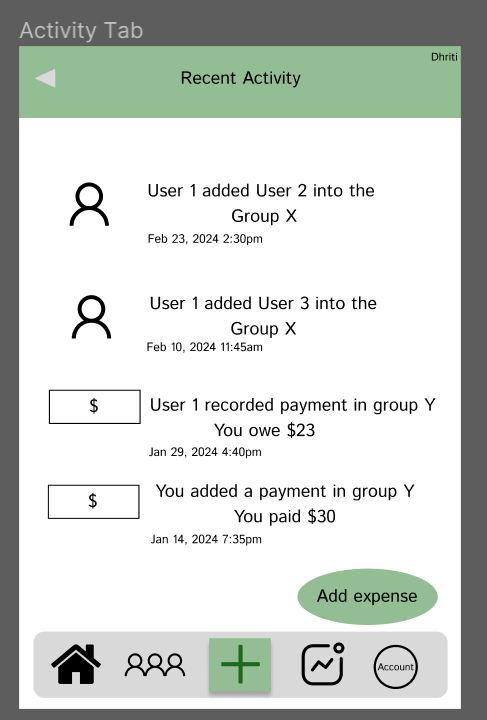
[Link to Figma website for group](https://www.figma.com/file/LdzbxXcr4wCvAKpb4Q3dtb/Untitled?type=design&node-id=0%3A1&mode=design&t=AKSHHKmFg0OKHCUe-1)

Group Tab Wireframes (Ben + Veer)

A screenshot of a group application

Description automatically generated

Activity Tab Wireframes (Dhriti)



Add a Group (Ayush)

A screenshot of a group

Description automatically generated

Pay Settlement to Group Member Wireframes (Huzaifa)

A screenshot of a group

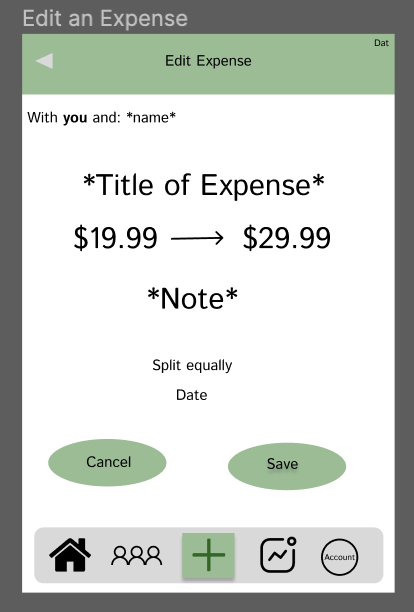
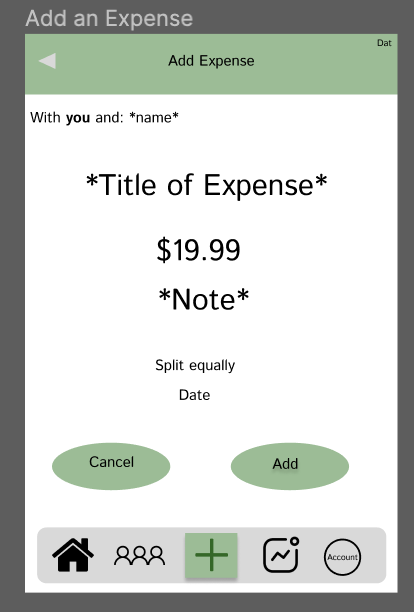
Description automatically generatedA screenshot of a group member

Description automatically generated

A screenshot of a phone

Description automatically generated

Add Expense and Edit Expense (Dat)



View All Debts: (Nigel)

Screens screenshot of a screenshot of a phone

Description automatically generated

Implementation

**Research Development Strategies based on your chosen Design and Delivery methodology**

Software development strategies provide software engineering teams with a structured and coordinated process as to how to design, implement, test, and deploy a software system. Among the many software processes that exist, Group 6 decided to employ a combination of prototyping and incremental software development processes. We felt these software engineering processes allowed us to best implement our solution by allowing us as a team to break up the software development process into small, manageable portions. We could then develop and deliver these portions in increments until we can deliver a desirable software product. The use of the prototype software development model in conjunction with incremental software development, allowed us to ensure that we could build and deliver a basic working version as fast as possible to receive feedback to validate our requirements, before continuing with full-scale development. Additionally, utilizing these specific software development models allowed our group to meet priority deadlines throughout our solution’s software development life cycle.

Based on our software’s design, we chose to implement our solution using Express.js framework for the backend of our application, Mongo DB to handle the data, and React to provide the frontend for our application. Collaboration on our solution was made possible by Git and GitHub. GitHub allowed members of our team to incrementally implement parts to our software solution until a functioning prototype was delivered, all while maintaining stability and keeping track of contributions.

**Design Class Diagram**

A screenshot of a computer

Description automatically generated

Software Testing

The various strategies or approaches used to test an application to ensure it functions as expected are known as software testing methodologies. Software testing is an imperative part of the software engineering process as it identifies any issues, defects, and/or shortcomings of the code so that they can be fixed before the software is deployed as a final product. For our purposes, Group 6 decided to utilize a combination of different software testing methods that encompass both functional and non-functional test cases. We employed unit testing, integration testing, acceptance testing, and usability testing.

**Functional Test Cases:**

Test Case for “Activity Tab” Use Case (Dhriti)

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Scenario | User in a Group | Expected Results |
| TC1 | User clicks on Activity | Valid | Members get the notification |
| TC2 | User clicks on Activity | Invalid | User isn’t shown notifications of a group he/she isn’t a part of. |

Test Case for “Account Creation” Use Case (Nigel)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Email | Username/ User ID | Phone Number | Expected Results |
| TC1 | Valid Creation | Valid | Valid | Valid | New Account created |
| TC2 | Missing Required Field(s) | Blank | Valid/Invalid/Blank | Valid/Invalid/Blank | Error Message: “Blank Field(s)” |
| TC3 | Missing Required Field(s) | Valid/Invalid/Blank | Blank | Valid/Invalid/Blank | Error Message: “Blank Field(s)” |
| TC4 | Missing Required Field(s) | Valid/Invalid/Blank | Valid/Invalid/Blank | Blank | Error Message: “Blank Field(s)” |
| TC5 | Invalid Email | Invalid | Valid/Invalid | Valid/Invalid | Error Message: “Email is already used” |
| TC6 | Invalid Username | Valid/Invalid | Invalid | Valid/Invalid | Error Message: “username is not unique/ already in use |
| TC7 | Invalid Phone Number | Valid/Invalid | Valid/Invalid | Invalid | Error Message: “phone number already associated” |

Test Case for “User create group” Use Case (Ayush)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Group Name | Email  (for invitation) | Phone number  (for invitation) | Expected Results |
| TC1 | User clicks on create group | Valid | Valid | Valid | Successful Group created |
| TC2 | Group name isn’t valid | Invalid | Valid | Valid | Error displays, Group isn’t created |
| TC3 | Email isn’t valid | Valid | Invalid | Valid | Error displays, Group isn’t created |
| TC4 | Phone number isn’t valid | Valid | Valid | Invalid | Error displays, Group isn’t created |
| TC5 | No group name is entered | Invalid | Valid | Valid | Error displays, Group isn’t created |
| TC6 | No email is entered | Valid | Invalid | Valid | Error displays, Group isn’t created |
| TC7 | No phone number is entered | Valid | Valid | Invalid | Error displays, Group isn’t created |

Test Case for “Settling Debt with other User” (Huzaifa)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Amount Owed | Amount Paid | Receiver | Expected Result |
| TC1 | User (Payer) selects Receiver and amount to pay | Valid | Valid | Valid | Debt Settled |
| TC2 | Invalid Receiver Name | Valid | Valid | User Doesn’t Exist | Error Message |
| TC3 | Invalid Payment Amount | Valid | Invalid | Valid | Error Message |
| TC4 | Settlement of Previously Settled Transaction | Invalid | Invalid | Valid | Error message |
| TC5 | Settlement for Non-existent Expense | Invalid | Valid | Valid | Error Message |

Test Case for “Edit an expense” (Dat)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Expense ID | Group ID | Amount | Expected Result |
| TC1 | User edits an expense | Valid | Valid | Valid | Expense is successfully edited with new amount. |
| TC2 | Group does not exist | Valid | Invalid | N/A | Error message for group ID |
| TC3 | Expense does not exist | Invalid | Valid | N/A | Error message for expense ID |
| TC4 | Invalid amount input | Valid | Valid | Invalid | Error message for amount |

Test case for “Delete Account” (Dat)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | User ID | Email | Phone Number | Expected  Result |
| TC1 | User enters correct input | Valid | Valid | Valid | Instructions sent to email to confirm deletion. |
| TC2 | User ID Invalid | Invalid | N/A | N/A | Error message is displayed for invalid user id. |
| TC3 | Email address Invalid | Valid | Invalid | N/A | Error message is displayed for email address |
| TC4 | Phone Number Invalid | Valid | Valid | Invalid | Error message is displayed for phone number |
| TC5 | Invalid input | Invalid | Invalid | Invalid | Error message |

Test Case for “App User Authentication” Use Case (Dhriti)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Scenario | Login ID | Password | Expected Result |
| TC1 | Successful user authentication | Valid | Valid | Display the confirmation |
| TC2 | Invalid login ID | Invalid | Valid | Incorrect Details |
| TC3 | Invalid password | Valid | Invalid | Incorrect Details |

Test Case for “Viewing Transactions Withing an individual Group or Friend.” (Ben)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | User ID has transactions the past 30 days | User ID has made any transactions | Friend ID has made transactions the past 30 days | Friend ID has made any transactions | Expected Result |
| TC1 | User selects past 30 days transactions with a friend | Valid | Valid | Invalid | Invalid | Display past 30 days transactions that match the user ID and friend ID |
| TC2 | User selects all past transactions with a friend | Invalid | Valid | Invalid | Invalid | Display all transactions that match the user ID |
| TC3 | User selects past 30 days transactions with a group | Valid | Valid | Valid | Valid | Display past 30 days transactions that match the user ID and friend ID |
| TC4 | User selects all past transactions with a group | Invalid | Valid | Invalid | Valid | Display all transactions that match |
| TC5 | User selects transactions with a friend or a group | Invalid | Invalid | N/A | N/a | Display “User has not made any transactions” |

Test Case for “Delete an Expense” Use Case (Veer)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Expense exists | User exists | Expense settled | Expected Result |
| TC1 | Expense doesn’t exist | invalid | N/A | N/a | System displays “expense doesn’t exist” |
| TC2 | Expense already deleted | invalid | N/A | N/A | System displays “expense already deleted” |
| TC3 | User doesn’t exist | N/A | invalid | N/A | System displays “user doesn’t exist” |
| TC4 | Expense not settled | N/A | N/A | invalid | System displays “expense not settled” |

Test Case for “Invite a friend” Use Case (Ahmed)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Group Name (to be invited to) | Email  (for invitation) | Phone number (for invitation) | Expected Results |
| TC1 | Valid Email, Phone #, and group name | Valid | Valid | Valid | “Invitation successfully sent” |
| TC2 | Invalid Group (User is not a member) | Invalid | N/A | N/A | “Error: invalid group” |
| TC3 | Invalid Email (no ‘@’ or valid domain) | Valid | Invalid | N/A | “Error: invalid email” |
| TC4 | Invalid Phone Number (invalid amount of #s) | Valid | Valid | Invalid | “Error: invalid phone number” |
| TC5 | Invalid group name and email | Invalid | Invalid | Valid | “Error: please enter valid group and email” |
| TC6 | Invalid group name and phone number | Invalid | valid | invalid | “Error: please enter valid phone number and group name” |

Deployment Plan

The testing strategy we chose is the blue-green deployment. The newer version is the blue version, and the stable tested version is the green one. The users use the green version that is known to work while the blue version is being worked and tested on, once the blue version has passed all tests, it is swapped out with the green version while the previous green version is stored in case an error comes up and if so, is swapped back in.

Results of Test Cases

Test case result for “App User Authentication” (Dhriti)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Scenario | Login ID | Password | Expected Results |
| TC1 | Successful user authentication | ABC@gmail.com | Snowfy24! | Display the confirmation |
| TC2 | Invalid login ID | ABC@.gmail | Icecream! | Incorrect Details |
| TC3 | Invalid password | DEF2403@gmail.com | 12345 | Incorrect Details |
| TC4 | Both Invalid | ADC$gmailcom | Elephant | Incorrect Details |

Test case result for “invite a friend” (Ahmed)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Group Name (to be invited to) | Email  (for invitation) | Phone number (for invitation) | Expected Results |
| TC1 | Valid Email, Phone #, and group name | Mountain Hikers | ABC@gmail.com | 936-483-9235 | “Invitation successfully sent” |
| TC2 | Invalid Group (User is not a member) | \_!@#$% | N/A | N/A | “Error: invalid group” |
| TC3 | Invalid Email (no ‘@’ or valid domain) | Summer Plans | 123.@gmail | N/A | “Error: invalid email” |
| TC4 | Invalid Phone Number (invalid amount of #s) | Trip to California | DEF@gmail.com | 92-345-4853 | “Error: invalid phone number” |

Test Case for “Viewing Transactions Withing an individual Group or Friend.” (Ben)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Scenario | Input: User has transactions | Expected Result | Actual Result |
| TC1 | User selects past 30 days transactions with a friend | Valid transaction withing 30 days in database | Past 30 days data with friend displayed | N/A |
| TC2 | User selects all past transactions with a friend | Invalid, user has not made any transactions with friend | No transactions with x friend | N/A |
| TC3 | User selects past 30 days transactions with a group | Valid transaction withing 30 days in database | Past 30 days transactions data within the group | N/A |
| TC4 | User selects all past transactions with a group | Invalid, there have been no transactions in group | No transactions have been made in this group | N/A |

Test case for “Delete Account” (Dat)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | User ID | Email | Phone Number | Expected  Result |
| TC1 | User enters correct input | username123 | ABC@gmail.com | 123-456-7890 | Instructions sent to email to confirm deletion. |
| TC2 | User ID Invalid | username@ | N/A | N/A | Error message is displayed for invalid user id. |
| TC3 | Email address Invalid | username123 | ABC@gmail.co@@ | N/A | Error message is displayed for email address |
| TC4 | Phone Number Invalid | username123 | ABC@gmail.com | 123-456-78 | Error message is displayed for phone number |

Test Case for “Group Creation” Use Case (Ayush)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Group Name | Email (for invitation) | Phone Number | Expected Results |
| TC1 | Valid Creation | Europe 2024 | [abc@gmail.com](mailto:abc@gmail.com) | 123-456-7890 | Successful Creation |
| TC2 | Missing Required Field (s) | Europe Trip | Blank | Blank | Error Message: “Blank Field(s) – Enter at least 1 phone number or email for invitation” |
| TC3 | Invalid Group Name | 123 | [abc@gmail.com](mailto:abc@gmail.com) | 123-456-7890 | Error Messages “Enter valid group name” |
| TC4 | Invalid Phone Number | Europe Trip | [abc@gmail.com](mailto:abc@gmail.com) | 123 | Successfully invited “Email”  Error Message: “Enter correct phone number” |

Test Case for “Account Creation” Use Case (Nigel)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Email | Username/ User ID | Phone Number | Expected Results |
| TC1 | Valid Creation | ABC@gmail.com | username123 | 123-456-7890 | New Account created |
| TC2 | Missing Required Field(s) | Blank | username123 | 123-456-7890 | Error Message: “Blank Field(s)” |
| TC3 | Invalid Email | ABC@.gmail | username123 | 123-456-7890 | Error Message: “Email is already used” |
| TC4 | Invalid Username | ABC@gmail.com | username@ | 123-456-7890 | Error Message: “username is not unique/ already in use |
| TC5 | Invalid Phone Number | ABC@gmail.com | username123 | 123-456-78 | Error Message: “phone number already associated” |

Test Case for “Delete an Expense” Use Case (Veer)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Expense ID | User ID | Expense settled | Expected Result |
| TC1 | Expense doesn’t exist | Ajdd92 [invalid] | User123 | True | System displays “expense doesn’t exist” |
| TC2 | Expense already deleted | 12398 [deleted] | User123 | True | System displays “expense already deleted” |
| TC3 | User doesn’t exist | 123 | NonexistantUser | True | System displays “user doesn’t exist” |
| TC4 | Expense not settled | 123 | User123 | False | System displays “expense not settled” |

Test Case results for “Settling Debt with other User” (Huzaifa)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Scenario | Amount Owed | Amount Paid | Receiver | Expected Result |
| TC1 | User (Payer) selects Receiver and amount to pay | 5$ | 5$ | Username123 | Debt Settled “You successfully paid [receiver] $5.” |
| TC2 | Invalid Receiver Name | 5$ | 5$ | Fakeusername123 | Error Message “[receiver] is not a registered user.” |
| TC3 | Invalid Payment Amount | 5$ | 15$ | Usename123 | Error Message “payment amount must be less than or equal to amount owed.” |
| TC4 | Settlement of Previously Settled Transaction | 0$ | 5$ | Username123 | Error message “you have already settled this debt” |
| TC5 | Settlement for Non-existent Expense | 0$ | 5$ | Username123 | Error Message “payment amount must be less than or equal to amount owed.” |

Glossary of Terms:

**AWS**: Amazon Web Services is a comprehensive, evolving cloud computing platform provided by Amazon.

**Router**: A networking device that forwards data packets between computer networks

**Authentication**: The process of verifying the identity of a user in an application

**Utility Services**: Backend of server managing of all data and processes

**Database**: An organized collection of structured information, or data, typically stored electronically in a computer system

**Layered Architecture:** Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request

­­­