2023

Necessary UML Diagrams for Software Development

FINAL REPORT

HISHAM ODEH

Contents

Software Description

SplitSmart is an all-in-one software application designed to streamline expense sharing among individuals or groups of people. It offers a comprehensive set of features, enabling users to easily create and manage their accounts, groups, and expenses. It contains functions that allows the user to create an account, log in to an existing account recover a forgotten password and deactivate an account.

A key feature of SplitSmart is group management, which allows users to create and manage groups of people to efficiently track shared expenses and balances. Expense management is also made simple, as users can create new expenses and manage existing ones effortlessly. This feature enables users to specify the involved parties in the expense so that appropriate balance adjustments can be made.

Another essential feature of SplitSmart is its notification system. This system keeps everyone in the group informed about the latest expenses and balances by notifying users when other group members create a new expense, a user creates a new group, or any adjustments are made to the group or expense. The notification system is automatically invoked when a user uses the group, expense, or payment feature.

SplitSmart also offers reporting functionality, generating reports and summaries of users' expenses, balances, and payments. Users can create custom reports with specified filters to meet their reporting needs. This functionality allows users to gain a quick and efficient overview of their expenses, balances, and payments.

Planned Functional Requirements

1. User registration and login: The software should allow users to create their own accounts and log in securely.

2. Password reset: The system should allow users to reset their password if they forget it or if their account is locked out due to too many failed login attempts.

3. User roles and permissions: The system should allow administrators to control who can perform certain actions, such as creating or approving expenses, viewing or editing group information, or managing other users within the group. This functionality should be secure and reliable, ensuring that users are only able to perform actions that are within their assigned role and permissions, and that their actions are properly recorded and audited by the system.

4. User profile management: The system should allow users to manage their own profiles, including their contact information, payment methods, and notification settings.

5. User authentication and authorization: The system should require users to authenticate themselves before accessing the system, and should enforce appropriate access controls based on user roles and permissions.

6. Group creation: Users should be able to create groups and include members for each group.

7. Group deletion: The system should allow group owners (admins) or administrators to delete groups.

8. Group member management: The system should allow group owners or administrators to manage the members of a group, including adding or removing users, changing user roles, and adjusting user permissions.

9. Expense creation: The software should allow users to create new expenses, including the amount, date, description, shared manner (equally split or split by %), and an optional receipt image.

10. Expense allocation: Users should be able to specify which other users were involved in an expense, so that the appropriate balance adjustments can be made.

11. Expense editing: The system should allow users to edit existing expenses, with appropriate balance adjustments made based on the changes.

12. Expense deletion: The system should allow users to delete expenses, with appropriate balance adjustments made based on the deletion.

13. Approval workflow: The software should have a system for reviewing and approving expenses before they are added to the system and used to adjust balances.

14. Notification system: The software should have a way to notify users when other group members create a new expense.

15. Balance tracking: The software should track the balance owed by each user to every other user, taking into account all expenses that have been entered.

16. Payment tracking: The software should allow users to record when payments have been made to settle balances owed, so that the balances are accurately reflected in the system.

17. Reporting and summaries: The software should provide users with reports and summaries of their expenses, balances, and payments, possibly including the ability to generate custom reports with specified date ranges and other parameters.

18. Security and privacy: The software should have appropriate security measures in place to protect user data, and user privacy should be respected throughout the system.

19. User support: The software should provide user support, such as FAQs, user guides, and customer service, to help users navigate the system and resolve any issues they encounter.

20. Scalability: The software should be able to handle a large number of users and groups, with performance and reliability that do not degrade with increasing load.

21. Ease of use: The software should be easy to use, with intuitive interfaces and clear navigation, so that users can quickly and easily perform the actions they need to manage their expenses and balances.

22. Multiple currencies: The system should allow users to enter expenses and balances in multiple currencies, and should be able to convert between currencies based on current exchange rates.

23. Multiple payment methods: The system should allow users to record payments made to settle balances using multiple payment methods, such as credit cards, bank transfers, or cash.

Planned Non-Functional Requirements

Usability:

- The system should have an intuitive and user-friendly interface to minimize the learning curve for users.
- The system should support multiple languages to accommodate users who speak languages other than English.
- The system should provide clear and concise instructions to guide users through complex tasks.

Reliability:

- The system should be available 24/7, with a target uptime of at least 99.9%.
- The system should be able to recover from unexpected errors or failures without data loss or corruption.
- The system should be able to handle a large number of simultaneous users without performance degradation.

Robustness:

- The system should be able to handle unexpected inputs or user actions without crashing or producing incorrect results.
- The system should be able to handle network interruptions or other external events that may affect its operation.
- The system should have proper error handling and logging mechanisms to help diagnose and debug issues.

Safety:

- The system should have appropriate security measures to protect user data and prevent unauthorized access.
- The system should comply with relevant data privacy laws and regulations.
- The system should have a backup and disaster recovery plan in case of system failure or data loss.

Performance:

- Response time: The system should respond to user requests within a reasonable time frame, with an average response time of no more than 2 seconds.
- Throughput: The system should be able to handle a large number of simultaneous requests, with a target throughput of at least 100 requests per second.
- Availability: The system should be available at all times, with scheduled downtime limited to a maximum of 4 hours per month.
- Accuracy: The system should accurately calculate and display expenses, balances, and payments, with an error rate of less than 0.1%.

Supportability:

- The system should have clear documentation and user guides to help users troubleshoot common issues.
- The system should have a bug reporting and tracking system to help users report and track issues.

Adaptability:

- The system should be able to adapt to changing business needs, such as adding new features or supporting new payment methods.
- The system should be able to scale up or down depending on the number of users or groups using the system.
- The system should support integration with third-party services, such as payment gateways or accounting software.

Maintainability:

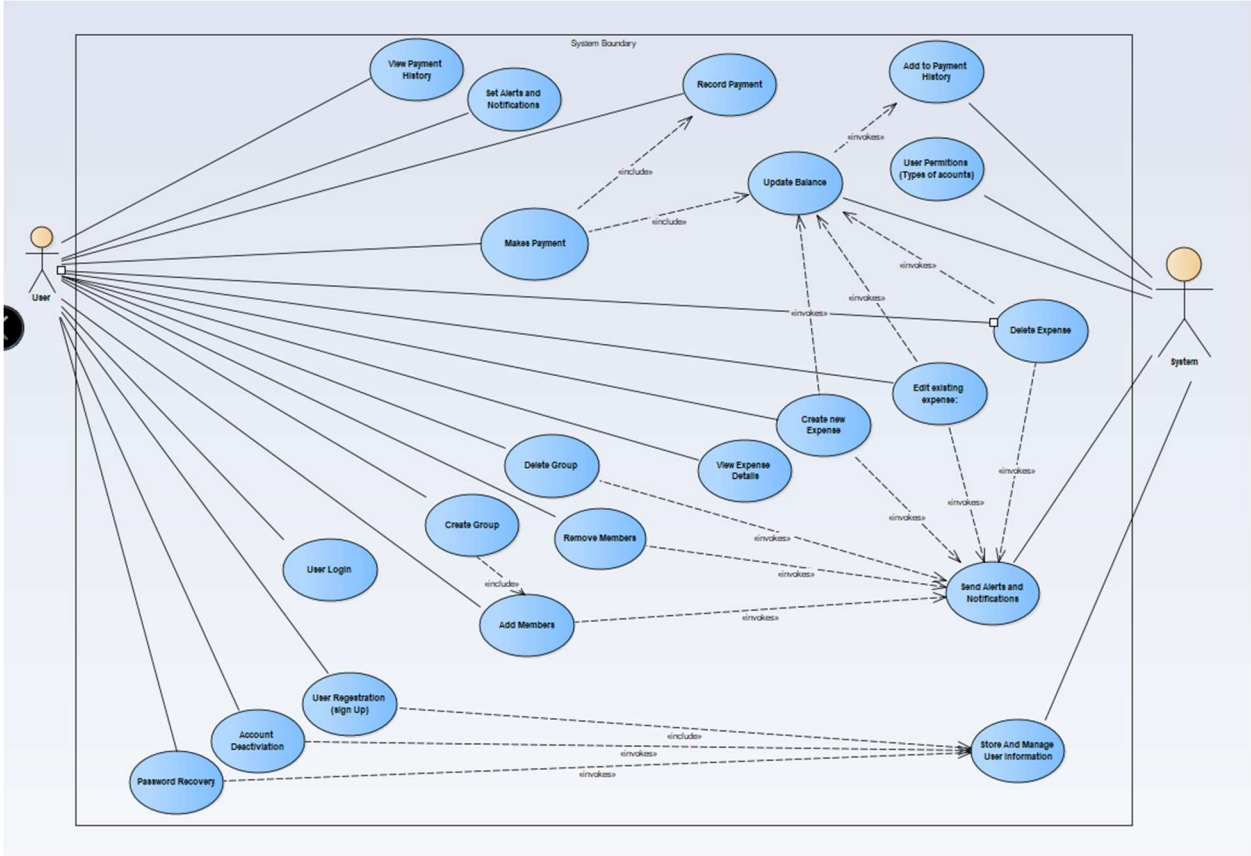
- The system should be designed to minimize the impact of changes or upgrades, with a focus on backward compatibility.

Portability:

- The system should be able to run on multiple platforms, such as Windows, Mac, or Linux.
- The system should be compatible with different web browsers, such as Chrome, Firefox, or Safari.
- The system should be designed to minimize dependencies on specific hardware or software configurations.

UML Diagrams

Use Case Diagram

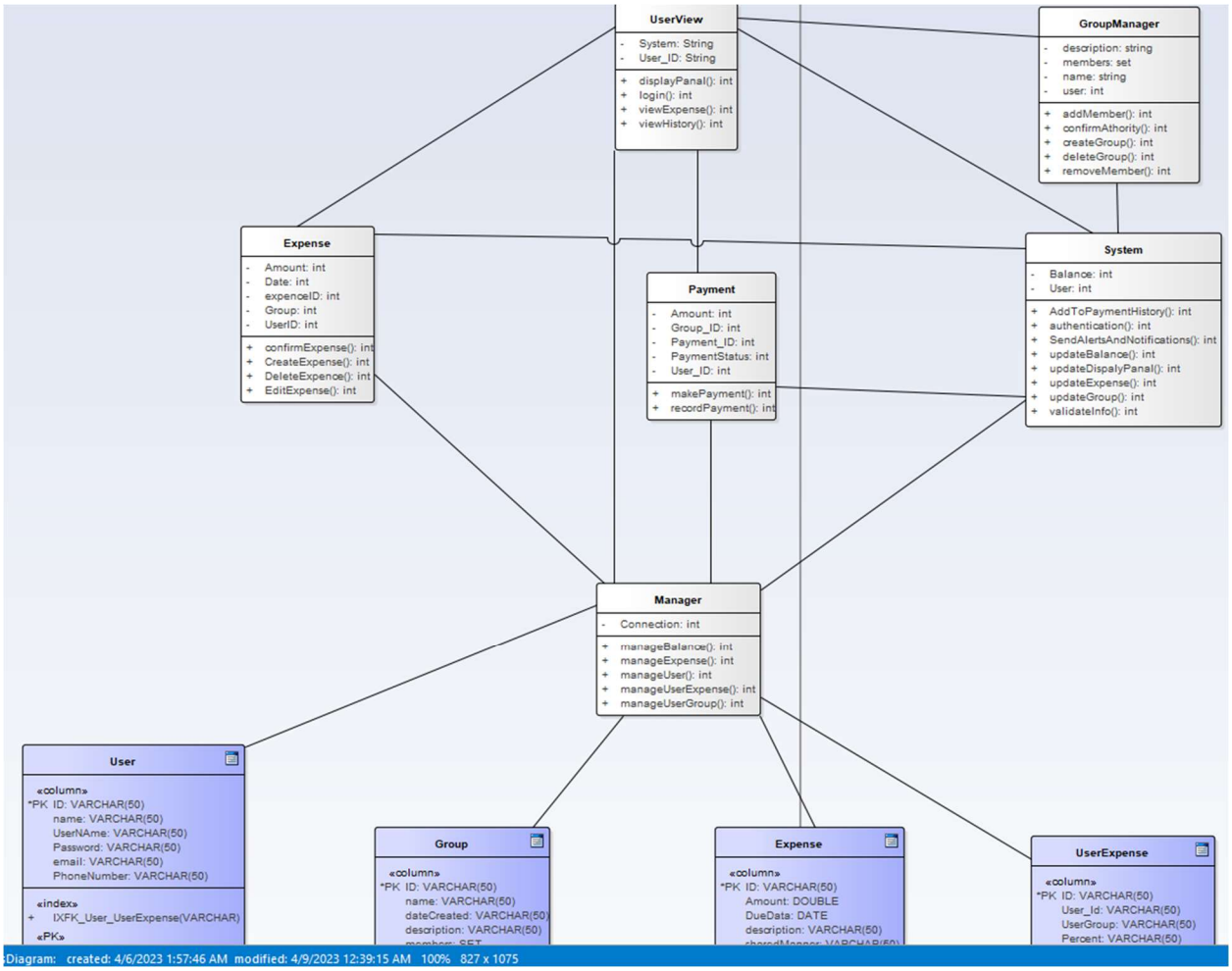


This UML Diagram presents over 20 use cases, each with a description and a flow of events in EA. As mapped out some use cases are only related to one actor, the user, or the system. Some are related to both, as some are related to the User that internally invoke use cases involved with the System.

The use cases related to the **User** are, related to the group, expense account, or payment.

The use cases related to the **System** are regarding updating balance, storing, and updating user information or sending alerts and notifications.

Class Diagram



This is the class diagram for out software product, this class diagram includes more than one flow of events.

The flow of the Diagram stars with User view grabbing information from the Manager, the flow either goes to the expense, groupMager, or payment classes, all these classes are connected to the System class the responds to the action/method invoked in the previous class, the system class invokes methods in the manger class and invokes it to update its information are reflect the change the user made, the finally the manager and system class cause the Userview class to update its view and renavigate to the userView class being both the start and end state.

Below are the three main paths (classes) that the user could chose in general and how it reflects on the diagram

Expense – create Expense:

- User logs into the software through the UserView class

- User navigates to the Expense section
- User click on one of the Expense related methods in the Expense class (createExpense())
- Software prompts the user for information such as amount, date, and if its associated with a group or not.
- System uses the validateInfo method in the System class to validate user input
- System uses the authentication method in the System class to authenticate the user
- System invokes UpdateBalance / updateExpense/ updateGroup to update the info and sends it to the manger table
- The manager table updates all the info for the user, group and expenses
- System invokes updatedisplayPanal(), as a resultof the updated info, so the userView is updated
- System send notifications and alerts

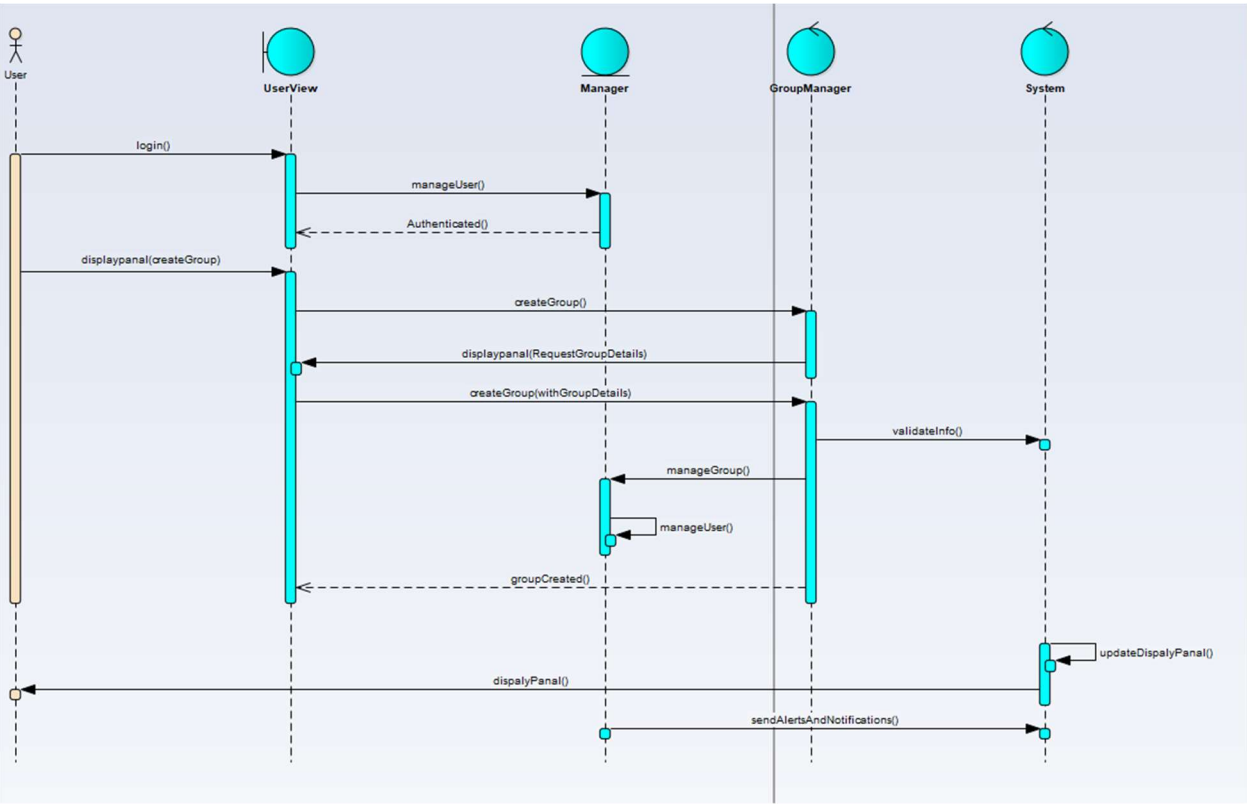
Manage group – create Group

- User logs into the software through the UserView class
- User navigates to the groupManger section
- User click on one of the groupManger related methods in the Expense class (createGroup())
- Software prompts the user for information such as members, and if its associated with a eexpense or not.
- System uses the validateInfo method in the System class to validate user input
- System uses the authentication method in the System class to authenticate the group members
- System invokes UpdateBalance/updateExpense/updateGroup/addtopaymentyhistory to update the info and sends it to the manger table
- System gives the user the option to record the payment
- The manager table updates all the info for the user, group and expenses
- System invokes updatedisplayPanal(), as a resultof the updated info, so the userView is updated
- System send notifications and alerts

Payment – make Payment

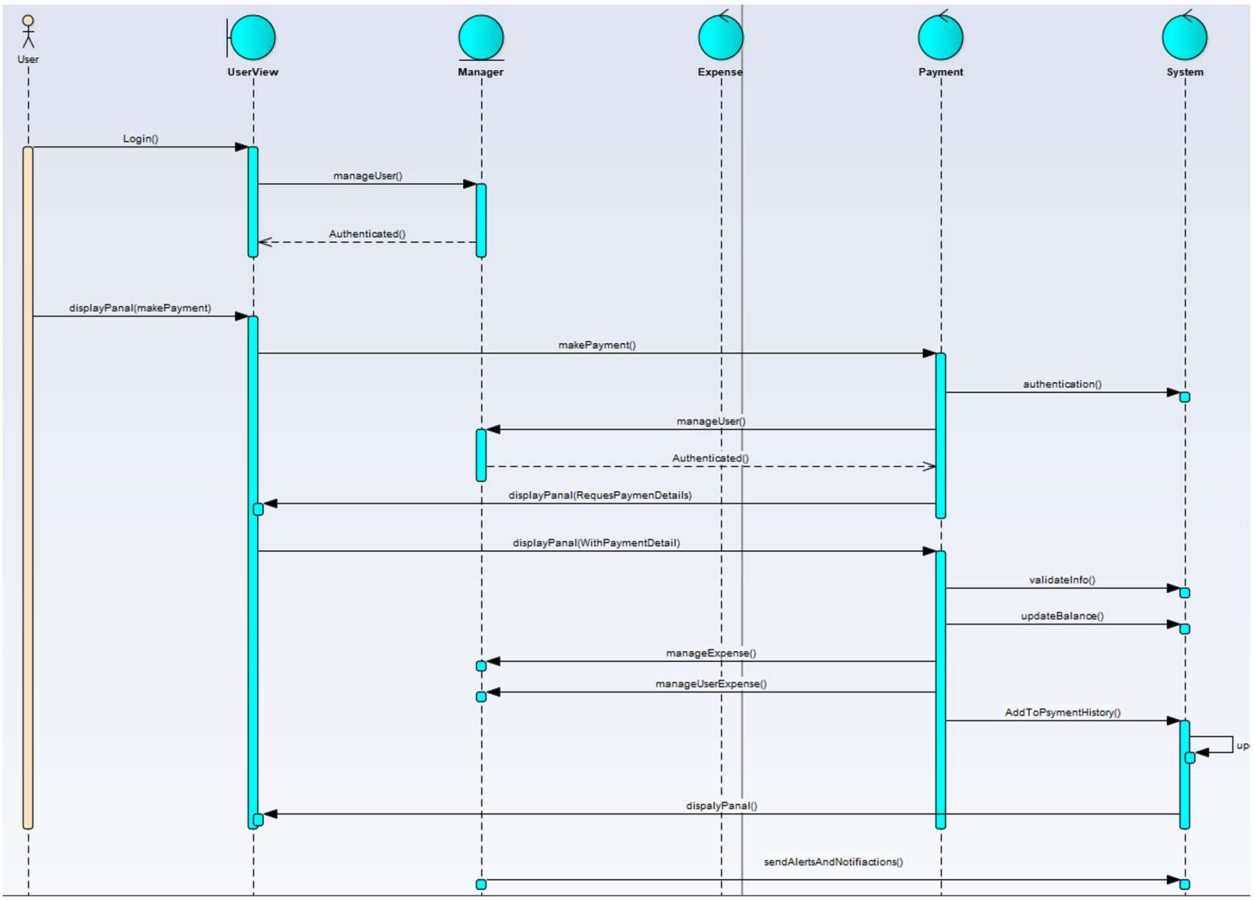
- User logs into the software through the UserView class
- User navigates to the payment section
- User click on one of the payment related methods in the payment class (makepayment())
- Software prompts the user to select a payment
- System uses the validateInfo method in the System class to validate user input
- System uses the authentication method in the System class to authenticate the user and payment method
- System invokes UpdateBalance/updateExpense/updateGroup to update the info and sends it to the manger table
- The manager table updates all the info for the user, group and expenses paymetns
- System invokes updatedisplayPanal(), as a resultof the updated info, so the userView is updated
- System send notifications and alerts

Sequence Diagrams



Create Group Sequence Diagram

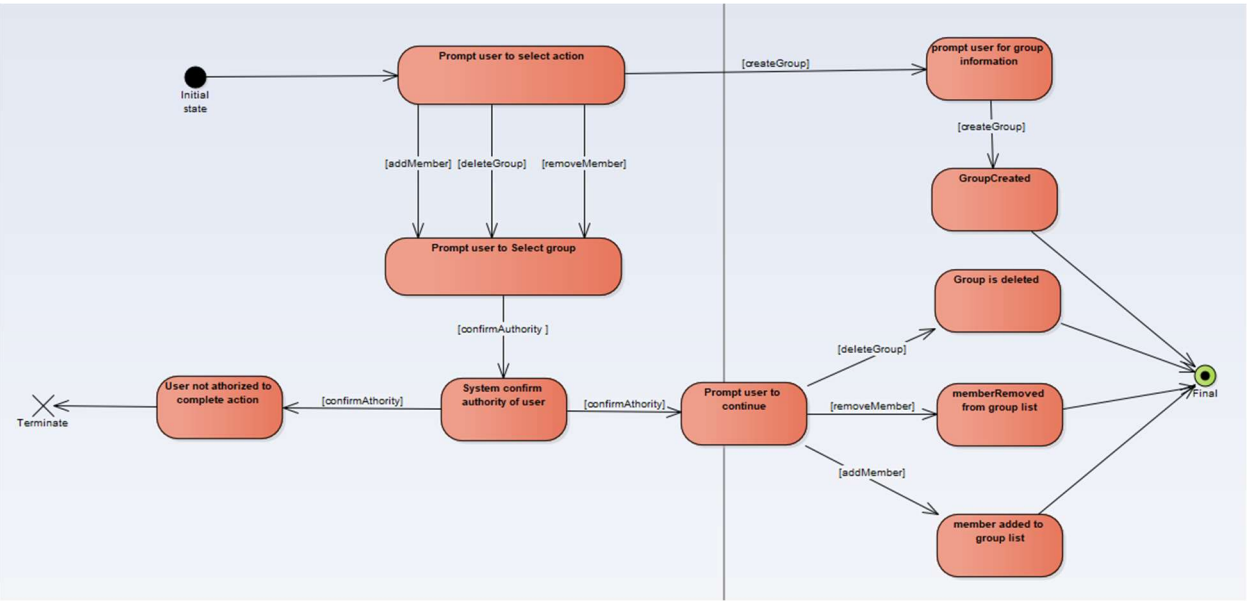
This sequence diagram describes how the user interacts with the userView class as a **Boundary**, the manager class as an **Entity** that stores and manages information in the system regarding the user and the group, and the expense. And groupManager and System as **Control** types that are used to executes activities that make creating a group possible.



Make Payment Sequence Diagram

This sequence diagram describes how the user interacts with the userView class as a **Boundary**, the manager class as an **Entity** that stores and manages information in the system regarding the user and the payment. And groupManager, payment, and System as **Control** types are used to executes activities that make the payment process possible.

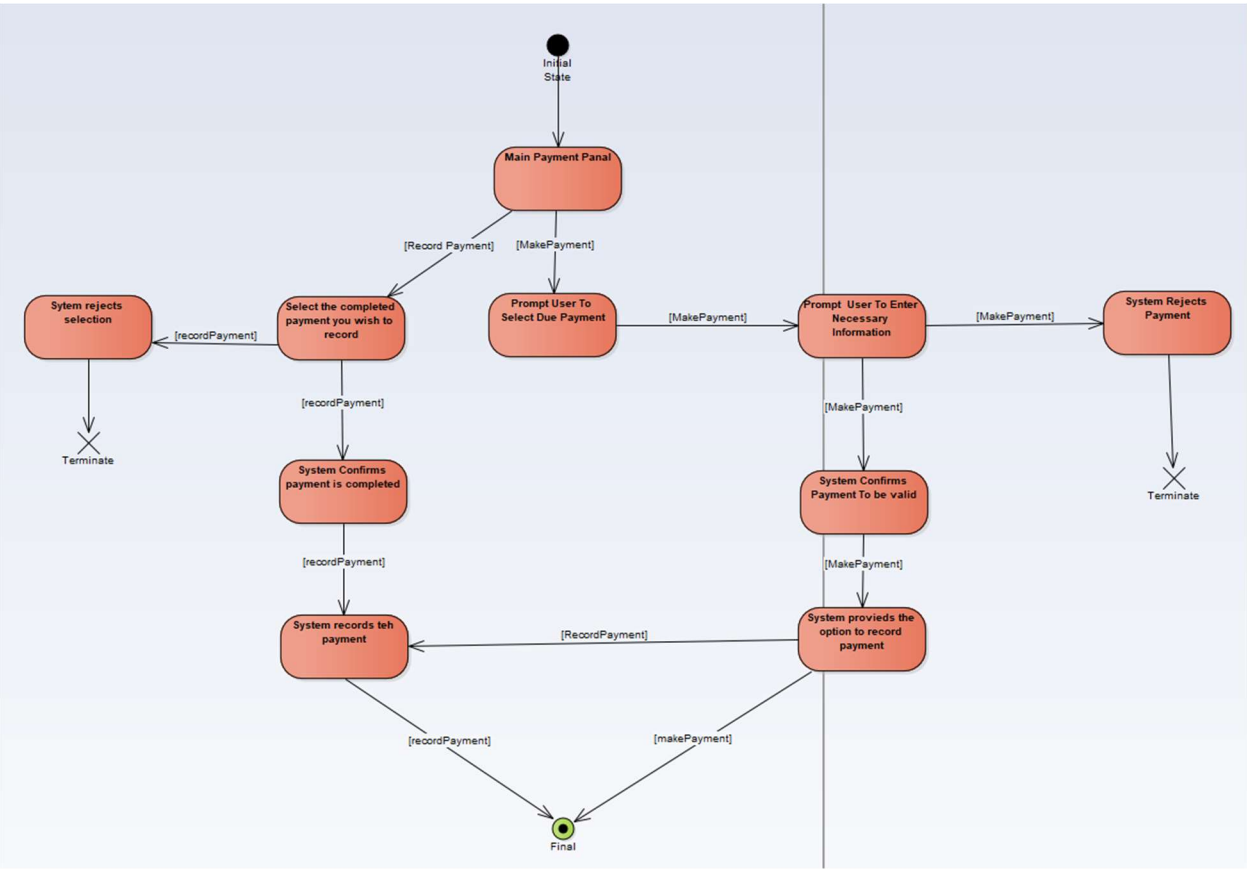
State Diagrams



State Diagram – GroupManager Class

This state diagram represents the groupManger class, and how the state of the main panel regarding the group changes as a response to the methods in the class. Note how the state of the system and the interface change as a response to what method or patch is chosen.

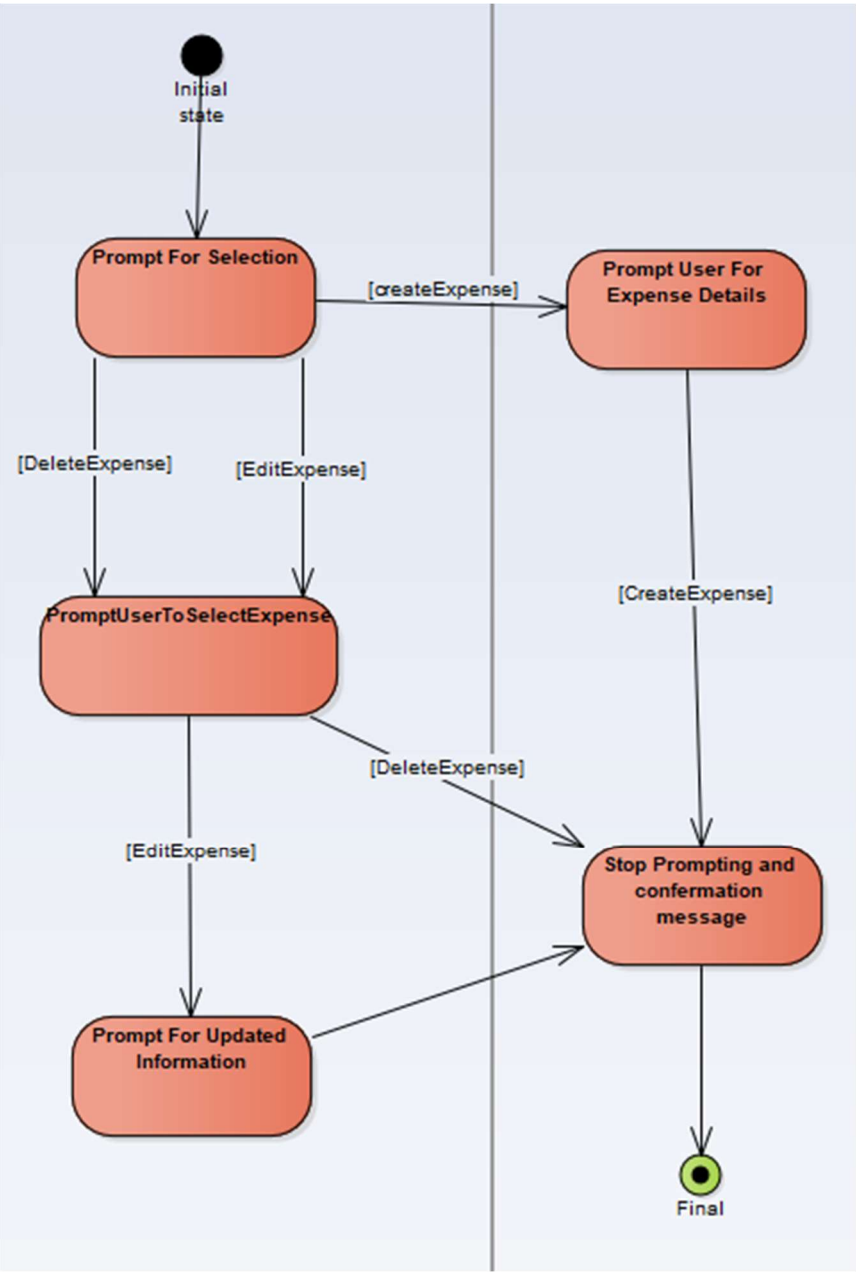
This explains the options and paths you have in the groupManger class, this also shows that the users authority to do an action is authorized during the process.



State Diagram – Payment Class

This state diagram represents the Payment class, and how the state of the main panel regarding the stages of the payment changes as a response to the methods in the class. Note how the state of the system and the interface change as a response to what method or patch is chosen.

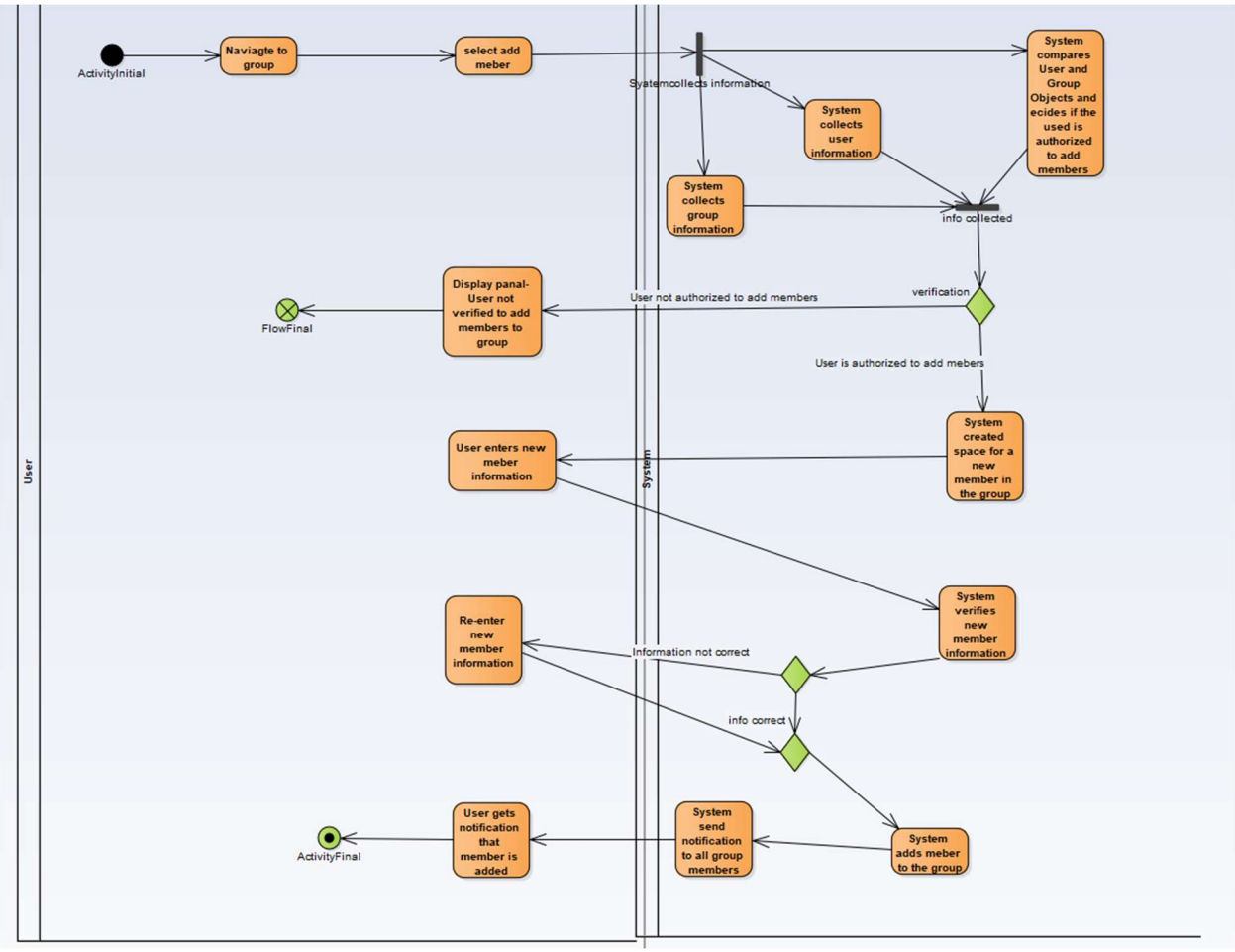
This diagram explains how the payment is verified during the process of the payment as well as the option of recording a payment invoked when a payment is made.



State Diagram – Expense Class

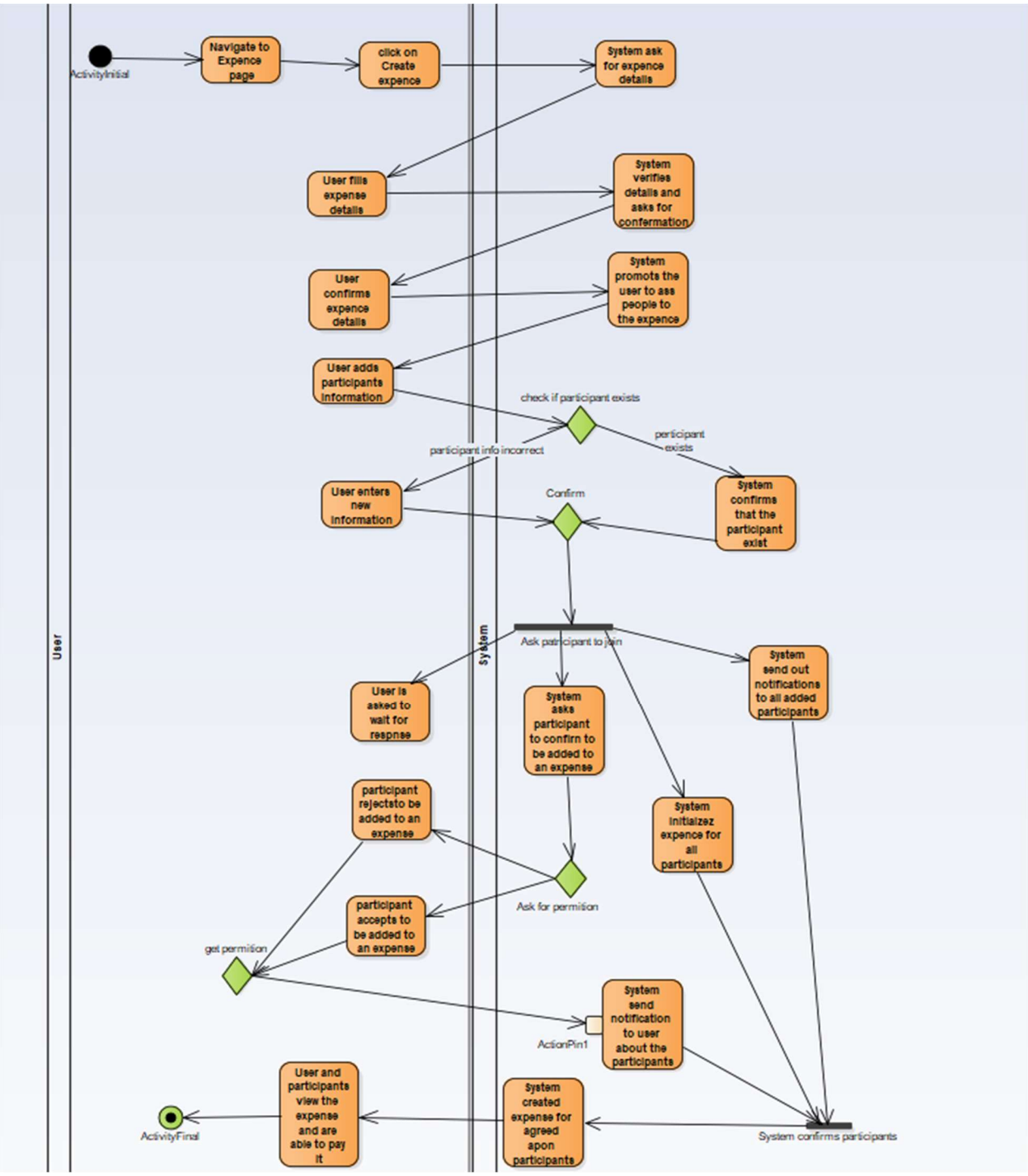
This state diagram represents the Expense class, and how the state of the main panel regarding the option of managing and creating expenses. Note how the expense info change as a response to what method is chosen.

Activity Diagrams



Activity Diagram – Add a member to an existing group

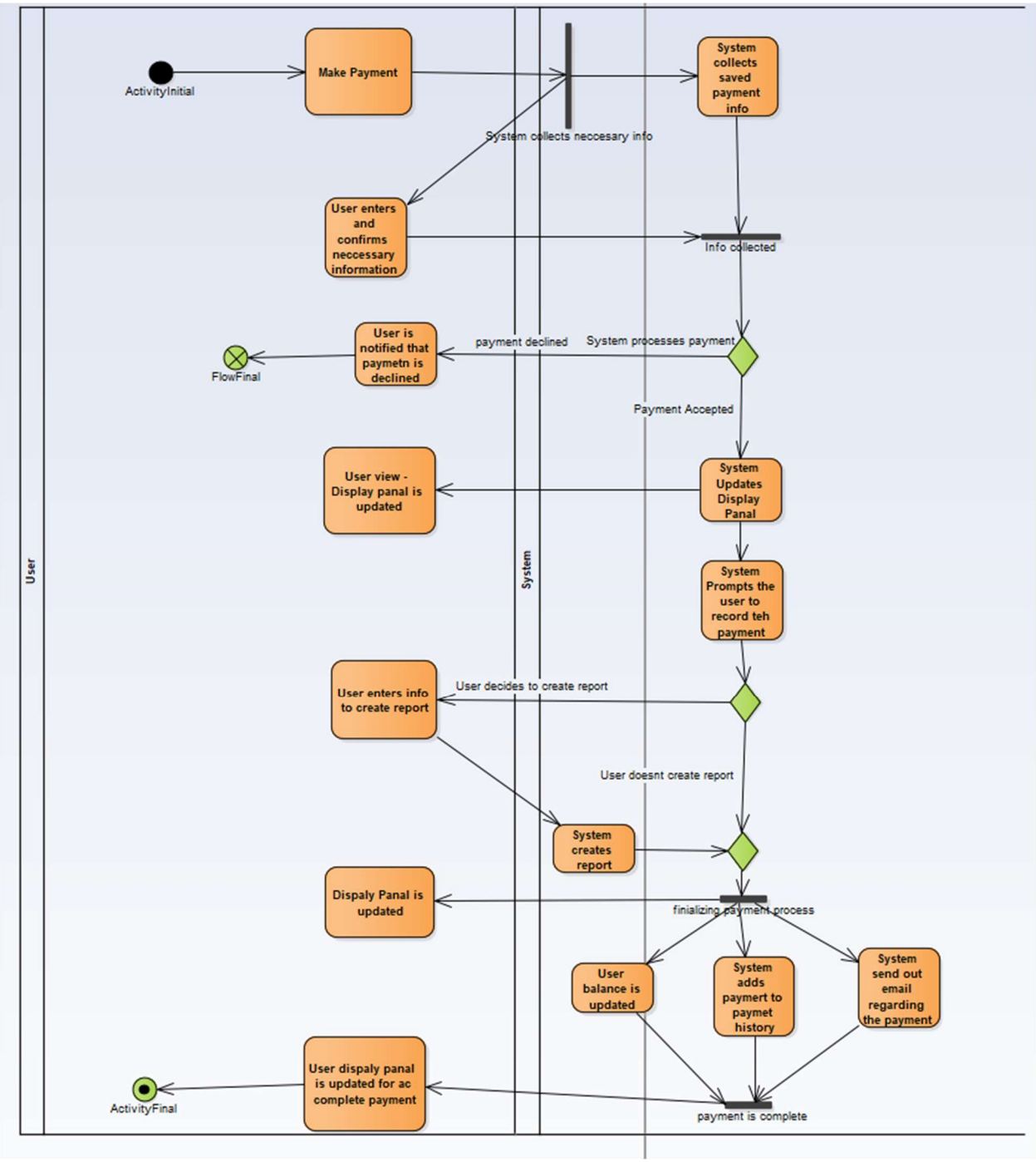
This activity diagram shows the user interacts with the system to add a member to an existing group. The system does a number of things, some of which use a merge such as comparing groups to users. Others use a fork such as the verification and authorization process. And after each process we see that the user gets a response either in the UI or as a email notification.



Activity Diagram – Create Expense

This activity diagram shows the user interacts with the system to create and construct an expense. The system does a number of things, some of which use a merge such as a process of asking participants to join. A fork is used for the process of getting permission from a participant.

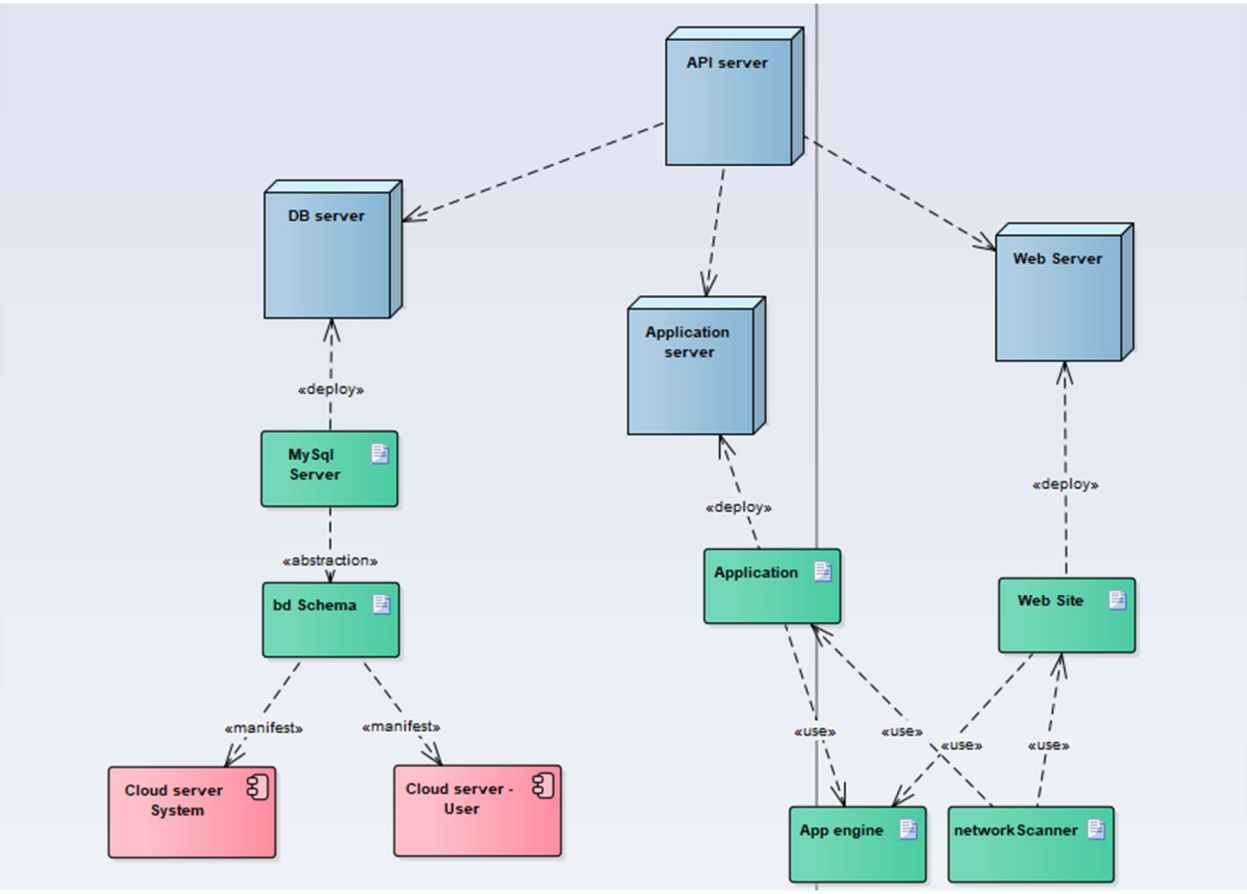
The user provides the system with info through the UI and receives it through the UI or a notification



Activity Diagram – Make payment

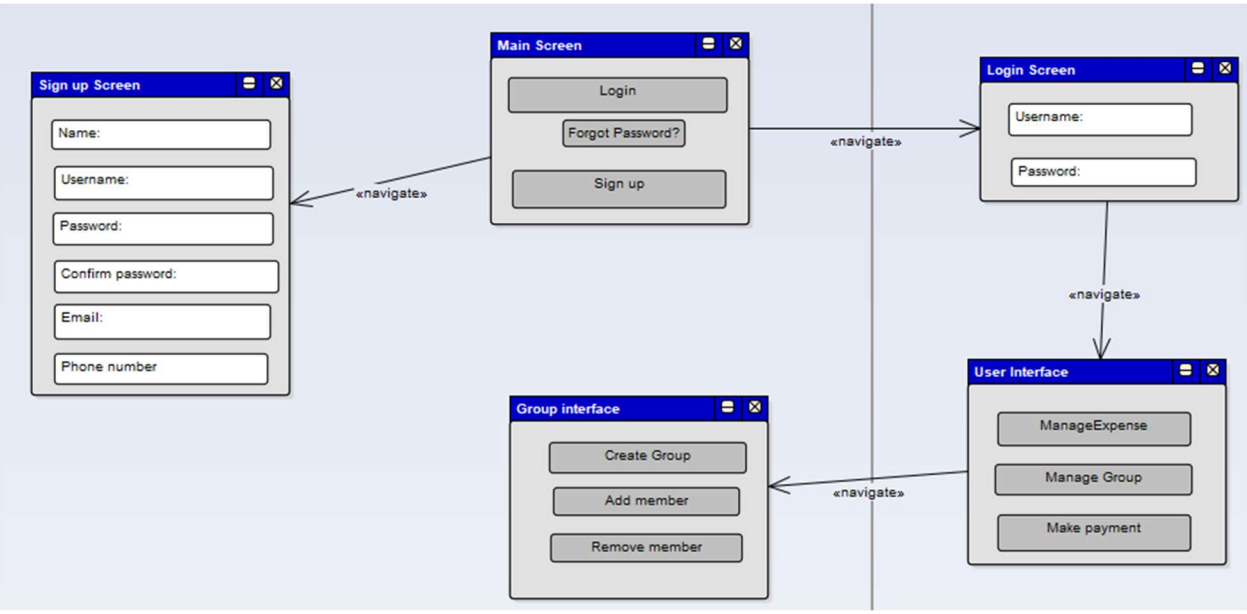
This activity diagram shows the user interacts with the system to make a payment for an existing expense. The system uses a merge in a few instances, one of which is to collect information. Forks are used to verify payment in one instance, as the system terminates the process if the payments didn't go through. The user provides the system with info through the UI and receives it through the UI or a notification

Deployment Diagram



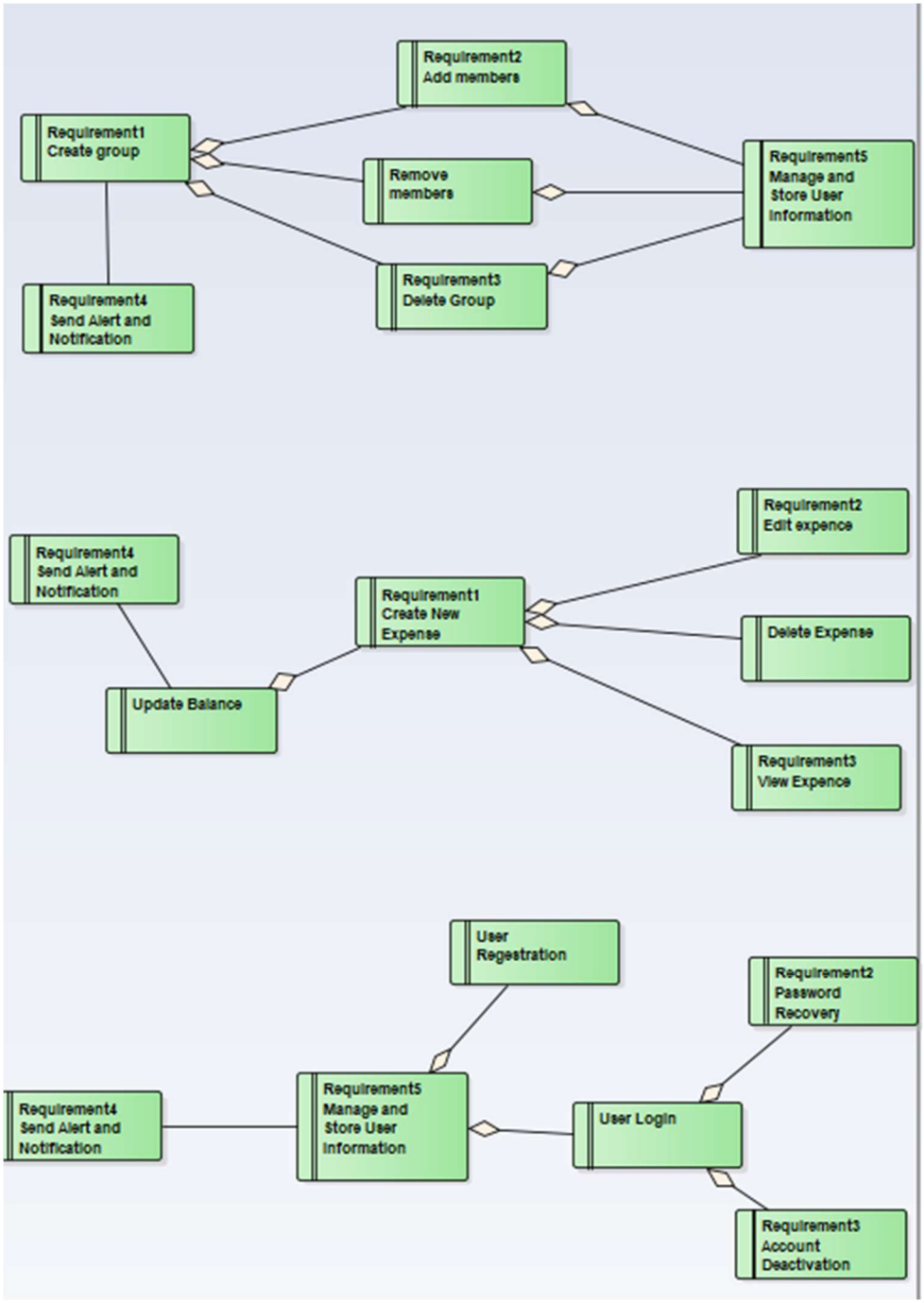
This deployment diagram explains how the different aspects of software, packages, services, and servers are connected and how they interact. For example, you can see in the application server, an application is deployed that uses an app engine at the very bottom of things.

User Interface Model



This UI navigates the user from the main login page to a feature such create group, and navigates every page to another. We see things like the login page and the reset password option, we also notice the main panel after we login.

Requirement Models



Here we have three requirement UML models

Each requirement model shows how a group of use cases are related and how they depend on each other.

The first is regarding the use cases related to the groupManager class, the second is regarding the expenses class and the third is concerning user accounts.