**CS3354 Software Engineering**

**Final Project Deliverable 2**

**Split Squad**

By: Tan Vo, Daniel Nguyen, Stephen George, Quan Pham,

Mir Ali, Samuel Preston, Amaan Babul, Shriniketh Mukundan

**GitHub Repository**: <https://github.com/md-y/3354-splitsquad>

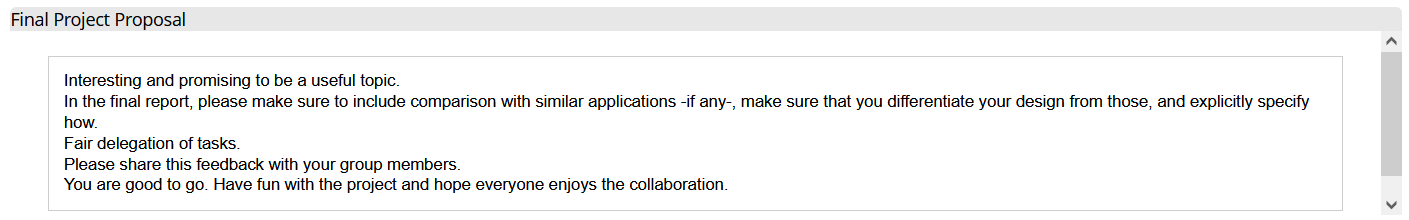
**1. [5 POINTS]** Well described delegation of tasks, i.e. who did what in the project. Now that your project is complete, you are required to submit the delegation of tasks from beginning of the project until the end. Please make sure to fairly distribute tasks in the team and remember that in the end of the semester, each member of a team will receive the same grade. See grading policy below for more detail. If no/poor contribution by a member, please specify clearly so that we can grade each student fairly.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task Description** | **Task Owner** | **Task Description** | **Task Owner** |
| GitHub Invitation | Samuel Preston | Project Scheduling | Daniel Nguyen |
| README | Tan Vo | Cost, Effort, Pricing | Shriniketh Mukundan |
| Project Scope | Stephen George | Hardware Cost | Stephen George |
| Process Model | Quan Pham | Software Cost | Tan Vo |
| Functional Requirements | Daniel Nguyen | Personnel Cost | Quan Pham |
| Nonfunctional Requirements | Mir Ali | Unit Testing | Samuel Preston |
| Use case Diagram | Samuel Preston | Comparison | Mir Ali, Amaan Babul |
| Sequence Diagram | Tan Vo |  |  |
| Class Diagram | Shriniketh Mukundan |  |  |
| Architecture Design | Amaan Babul |  |  |

**2. [10 POINTS]** Everything required and already submitted in Final Project Deliverable 1. Please specify this part as “Project Deliverable 1 content”.

**(D1) 1. [5 POINTS]** Please attach here the Final Project draft description (that contains the instructor feedback). It is ok to include a picture of the original document. Address the feedback provided for your proposal by listing what you did / plan to do to comply with those proposed changes and or requests for additions to your project.

|  |
| --- |
| **Description** |
| We'll be creating a mobile expense-sharing app to streamline the process of splitting costs among multiple people. Our goal is to simplify shared financial management for roommates, households, travel groups, teams, and event organizers to promote transparency and reduce conflicts in real-life scenarios. |



|  |
| --- |
| In compliance with the feedback, we will include the comparison with similar applications to make sure that our design is unique from our competitor.  Venmo is a similar application that also allows friends to send money to each other electronically. However, Venmo does not have native functionality to create groups and split money between them. This means our application has an advantage to Venmo in this regard.  Zell is a similar application to Venmo that allows friends to send money to each other and to split bills. Its main advantage over Venmo is that it directly deposits money it users’ bank accounts instead of holding money in a separate account. However, only certain banks support this ability, so Zell is not available to every possible user. Our application does not have this problem because it will support multiple payment systems, including Zell. This means that our application allows for direct deposit, but it does not require having an account at certain banks. |

**(D1) 2. [10 POINTS]** Setting up a GitHub repository. Please use your utdallas email accounts only for each group member.

1.1. Each team member should create a GitHub account if you don’t already have one.

1.2. Create a GitHub repository named 3354-teamName. (whatever your team’s name will be).

1.3. Add all team members, and the TA as collaborators. Our TA will post his GitHub info on

1.4. Make the first commit to the repository (i.e., a README file with [team name] as its content).

1.5. Make another commit including a pdf/txt/doc file named “project\_scope”. If you choose a predefined topic (one of the 4 topics described in the “Project Topic Ideas” section of this document), the contents of the file should be identical to the corresponding project in this section. If you choose other topics, the contents should follow a similar structure.

1.6. Keep all your project-related files in your repository as we will check them. Include the URL of your team project repository into your project deliverable 1 report.

**Important Note:**

* Tasks 1.3 - 1.5 should be performed by different team members. We will check the commit history for these activities.
* Do not include credentials (e.g., UTD ID) in the repository.
* Only commits performed before the deadline will be considered. Do not forget to push your changes after you have done the work!

**GitHub Usernames:**

Samuel Preston – md-y

Daniel Nguyen – CrazEpic

Tan Vo – tenvoltz

Quan Pham – 11Quan10

Shriniketh Mukundan – 23Shrin1977

Stephan George – StephenGeorgeUTD

Mir Ali – mirhali

Amaan Babul – amaan1540

**(D1) 3. [5 POINTS]** Delegation of tasks: Who is doing what. If no contribution, please specify as it will help us grade each group member fairly.

[This is also included in question 1 of Deliverable 2]

|  |  |  |
| --- | --- | --- |
| **Task** | **Task Description** | **Task Owner** |
| 2.1.3 | GitHub Invitation | Samuel Preston |
| 2.1.4 | README | Tan Vo |
| 2.1.5 | Project Scope | Stephen George |
| 4 | Process Model | Quan Pham |
| 5.a | Functional Requirements | Daniel Nguyen |
| 5.b | Nonfunctional Requirements | Mir Ali |
| 6 | Use case Diagram | Samuel Preston |
| 7 | Sequence Diagram | Tan Vo |
| 8 | Class Diagram | Shriniketh Mukundan |
| 9 | Architecture Design | Amaan Babul |

**(D1) 4. [5 POINTS]** Which software process model is employed in the project and why. (Ch 2)

|  |
| --- |
| The agile process model is best suited for the development of our expense sharing app. Agile methodologies, such as Scrum, offer the flexibility for us to create new requirements/functionalities and make changes to the application per user feedback. Any development ideas can be addressed in a scrum meeting, and the development team would focus and work on each topic on a daily basis. In the first scrum meeting, the team will come together with the customers and lay out the basic requirements and functionalities for the app (can be seen in project\_scope and functional/non-functional requirements specified in other sections of this deliverable) and begin the development process. For every meeting afterwards, the team will attempt to implement each functionality and requirement and have the customers review them. Each feature should be kept simple and adaptable for the development process to take after the agile process model. The increment phase and development process will end when customers are mostly satisfied with the final product (every requested feature is implemented with little to no issues). |

**(D1) 5. [15 POINTS]** Software Requirements including

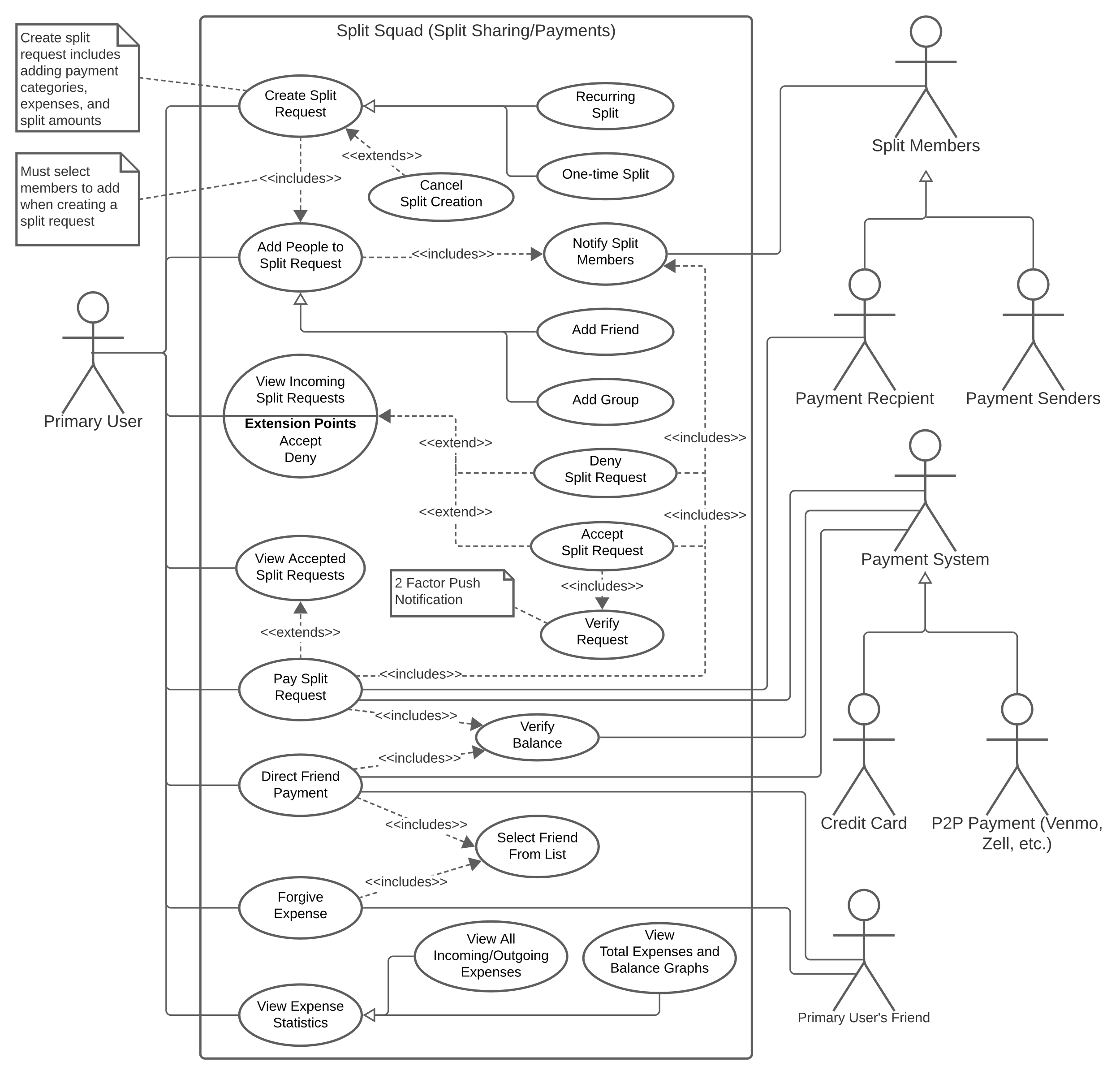
**(D1) 5.a.) [5 POINTS]** Functional requirements. To simplify your design, please keep your functional requirements in the range of minimum 5 (five) to maximum 7 (seven). (Ch 4)

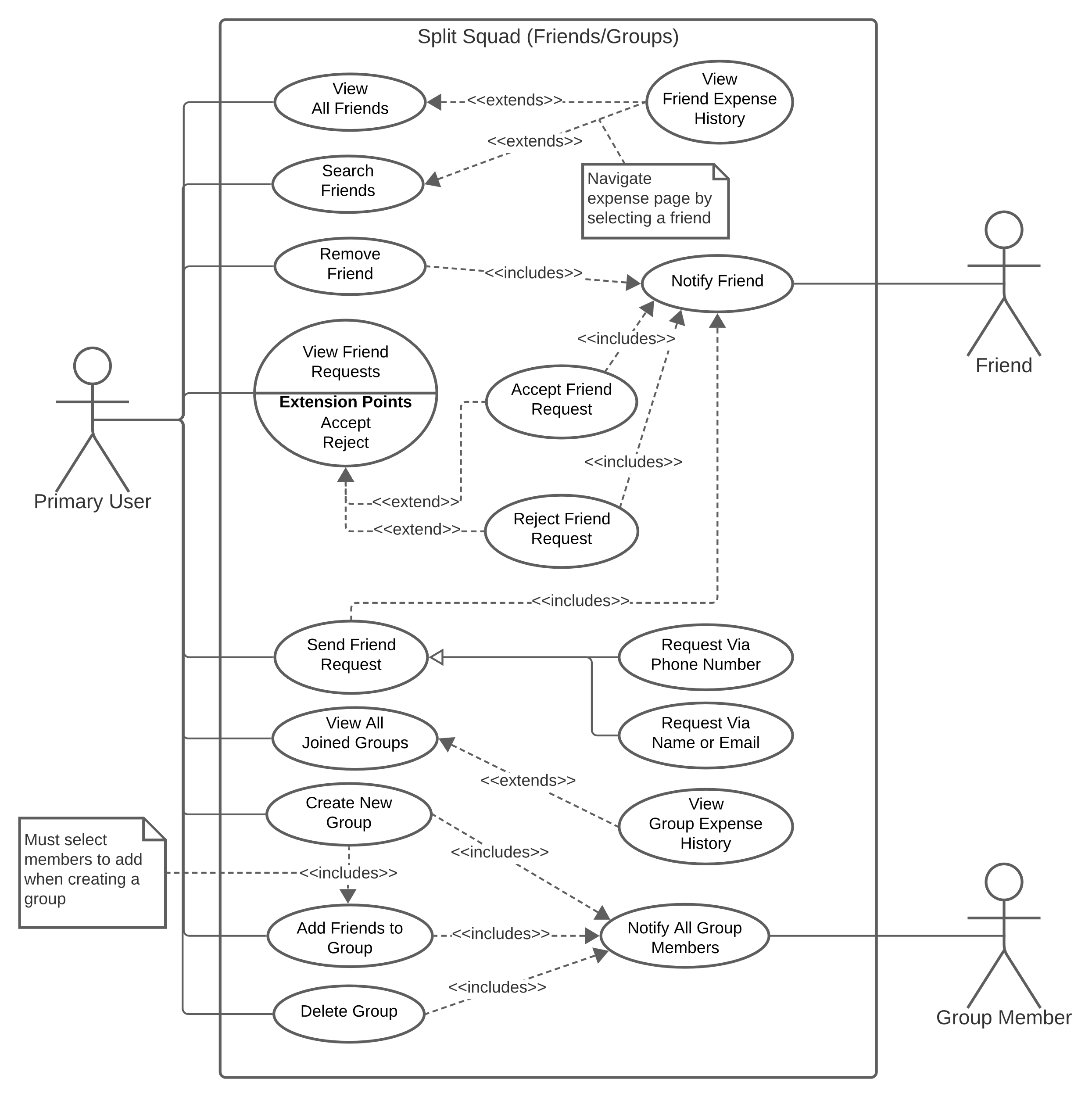
|  |
| --- |
| 1. The user shall be able to create a split request among one or more other users with fields to input total expense, to split total expense in the split request equally, by percentage, by custom amount, or by a mix of percentages and custom amount with options to add periodicity (weekly/monthly) or categories (food, rent, trips, other) to the split request. 2. The user shall be able to see and accept or deny a pending split request and, if accepted, pay the expense back partially or in full via a selected payment integration such as debit card, credit card, or a supported Peer-to-Peer payment system such as Paypal or Venmo. 3. The user shall be able to view expense organization such as history of all incoming/outgoing expenses (“I owe you”/”You owe me”) and current unpaid incoming/outgoing expenses or statistics such as total incoming/outgoing expenses, net balance of expenses, graph of net balance over time, and expenses by category (food, rent, trips, other). 4. The user shall be able to forgive or partially forgive an outgoing expense with an optional note for adding a reason such as cash payment or favor. 5. The user shall be able to add/view/search for/remove friends and create/view/search for/delete groups and see all related expense information for the friend or group. 6. The user shall be able to view and change account settings such as name, email, phone, profile picture, payment information, password, language, dark/light view mode and notification settings such as toggles for text/email/mobile push notifications, friend requests notifications, pending split request notifications, and incoming payment notifications. 7. The system shall store all user information, friend/group information, expense records in server database(s) belonging to the app. |

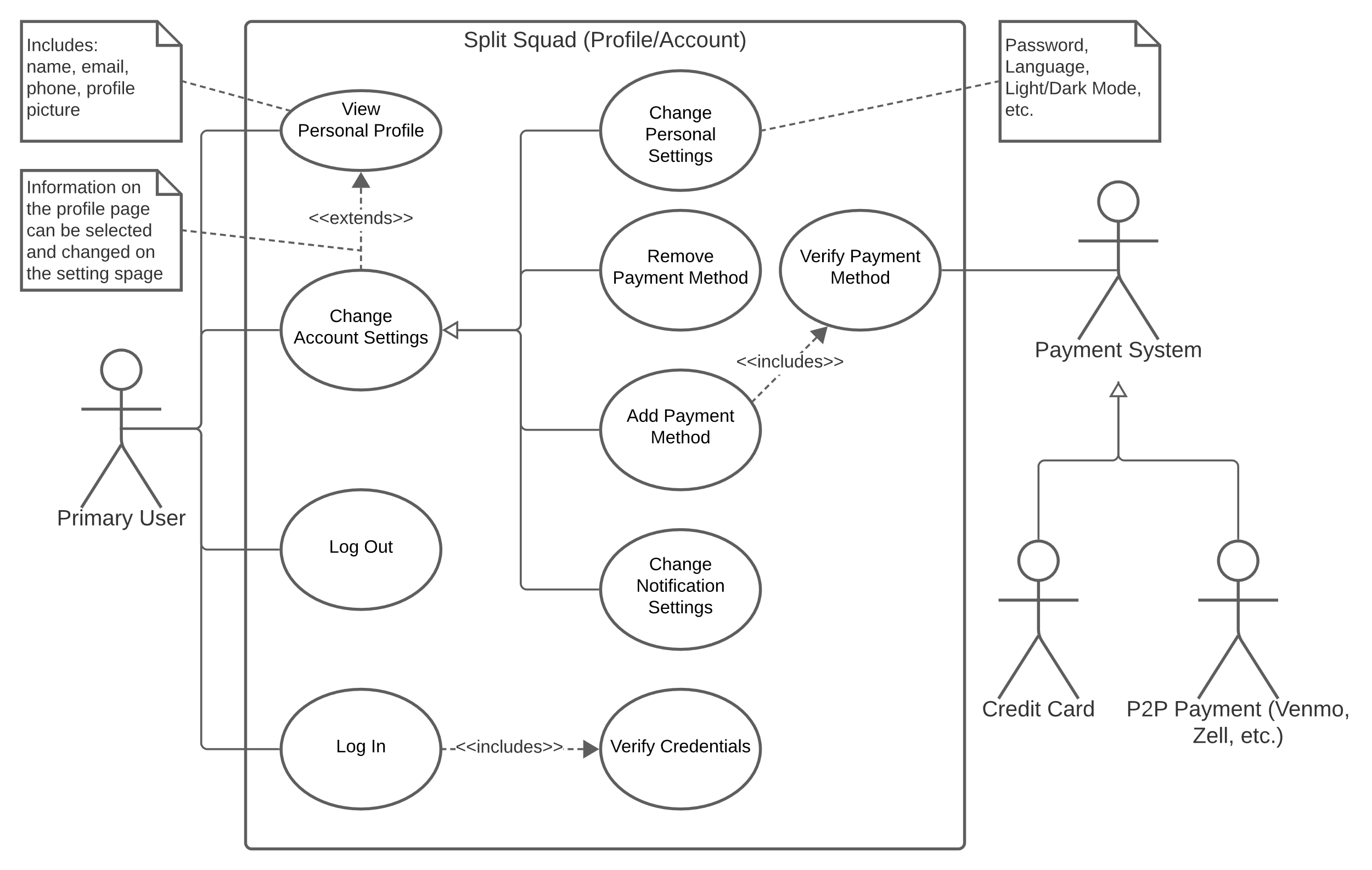
**(D1) 5.b.) [10 POINTS]** Non-functional requirements (use all non-functional requirement types listed in Figure 4.3 - Ch 4. This means provide one non-functional requirement for each of the leaves of Figure 4.3. You can certainly make assumptions, even make up government/country-based rules, requirements to be able to provide one for each. Please explicitly specify if you are considering such assumptions.)

|  |
| --- |
| **Usability Requirements:**   * The user interface should be intuitive and easy to navigate to ensure a positive user experience. * Users should be able to perform basic tasks (e.g., creating a group, initiating a split) within 1 minute of interaction with the user interface. * User satisfaction surveys should indicate an average satisfaction score of at least 4 out of 5.   **Performance Requirements:**   * The system should provide a responsive user interface, with pages loading within 2 seconds under typical user loads. * The software should be capable of handling concurrent user loads of up to 1000 users without any request response time exceeding 3 seconds. * Backend server response time for split requests and expense updates should be less than 100 milliseconds.   **Space Requirements:**   * The software should efficiently manage and store user data, ensuring that the database storage requirements are optimized. * The database should support a minimum of 100,000 user accounts and associated transaction records. * Data storage should not exceed 1 GB for every 10,000 user accounts.   **Security Requirements:**   * User data, including personal and financial information, should be encrypted both in transit and at rest. * All data in transit should be encrypted using industry-standard TLS protocols. * Data at rest should be encrypted using AES-256 encryption. * Access control and authentication mechanisms should be in place to prevent unauthorized access to user accounts. * Multi-factor authentication (MFA) should be enforced for all user accounts. * Failed login attempts should trigger account lockouts for a defined period. * The software should have mechanisms to detect and respond to security threats, such as intrusion detection and prevention systems. * Intrusion attempts should be logged, and security alerts should be generated for the operations team. * Regular security audits and updates should be performed to address vulnerabilities and compliance with security standards. * Security audits should be conducted quarterly, and patches for critical vulnerabilities should be applied within 30 days of release.   **Environmental Requirements:**   * The software should be designed to work in various network environments, including mobile data and Wi-Fi. * It should be compatible with a wide range of devices, including smartphones, tablets, and computers.   **Operational Requirements:**   * The software should have high availability, with a target uptime of 99.9%. * The service should be available 99.9% of the time (approximately 8 hours and 45 minutes of downtime per year). * The system should be easy to maintain and update without significant downtime. * Software updates should be deployed during off-peak hours, with scheduled maintenance windows limited to 2 hours.   **Development Requirements:**   * The software development process should adhere to coding and testing standards. * Version control and collaborative development tools should be in place. * Adequate documentation for the software architecture and codebase should be maintained. * The development team should follow secure coding practices to mitigate vulnerabilities.   **Regulatory Requirements:**   * The software should adhere to data privacy regulations, such as GDPR or HIPAA, depending on the user data handled. * User data should be anonymized or pseudonymized to comply with GDPR data protection requirements. * Compliance with financial regulations and payment processing standards should be maintained if financial transactions are involved. * All financial transactions must adhere to PCI DSS compliance standards.   **Ethical Requirements:**   * The software should prioritize user privacy and data protection, and user consent should be obtained for any data processing. * The system should discourage unethical financial practices or encourage fair expense sharing.   **Accounting Requirements:**   * If the software handles financial transactions, it should provide accurate and auditable financial records. * Users should be able to view their transaction history, including payments and expenses.   **Safety/Security Requirements:**   * The system should implement security measures to protect against fraudulent activities. * Safety measures should be in place to prevent accidental financial transactions or data loss. |

**(D1) 6. [15 POINTS]** Use case diagram – Provide a use case diagram (like Figure 5.5) for your project. Please note that there can be more than one use case diagram as your project might be very comprehensive. (Ch 5 and Ch 7)

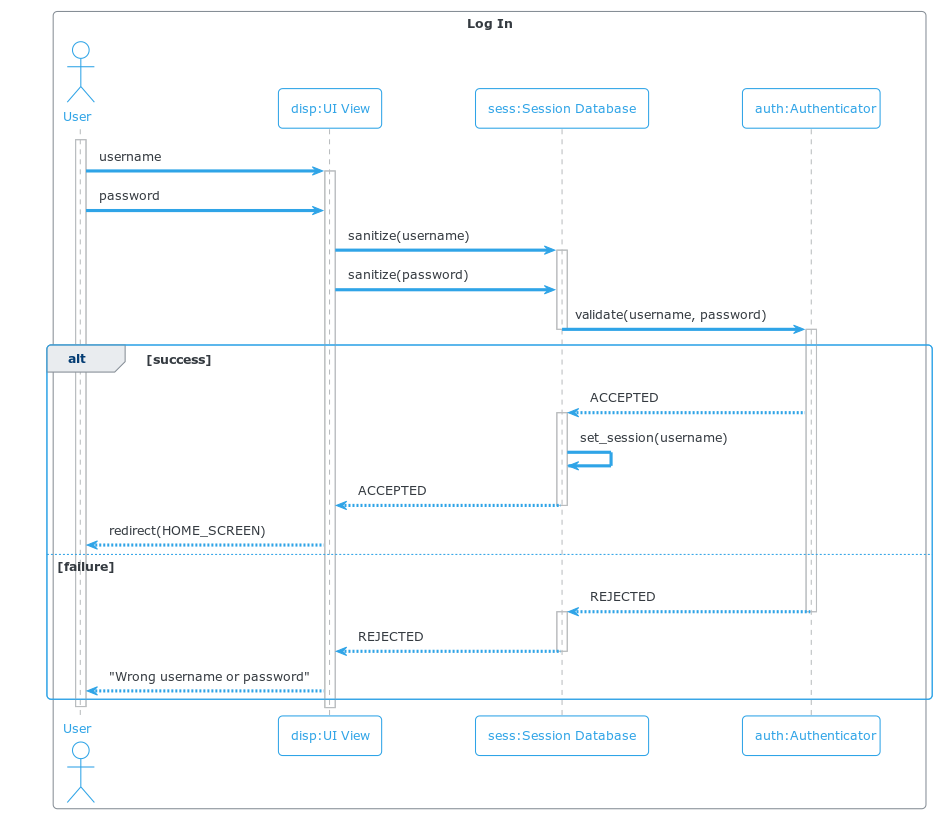


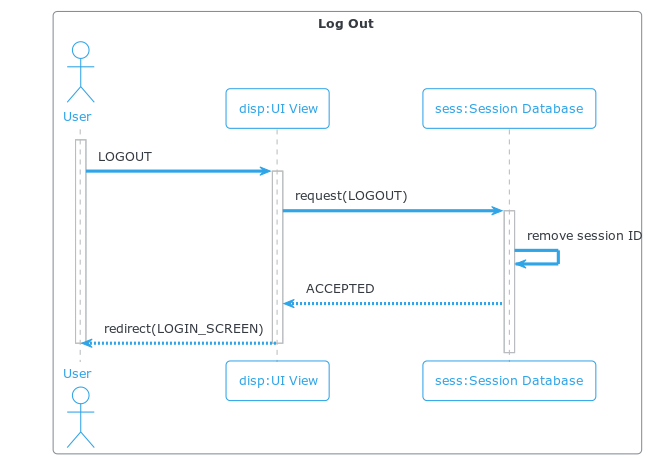


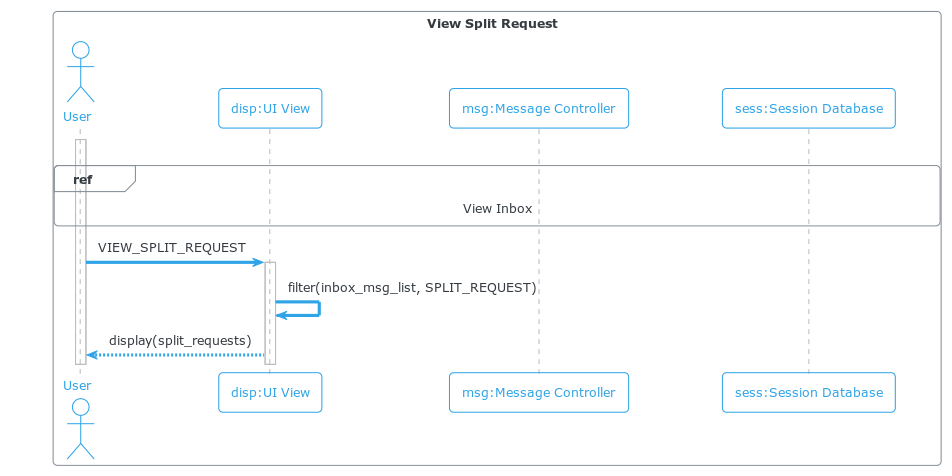
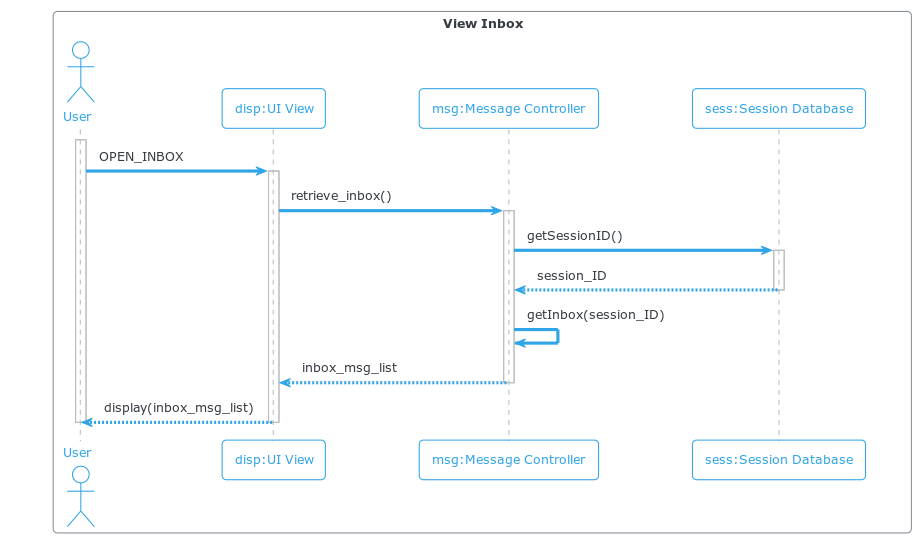
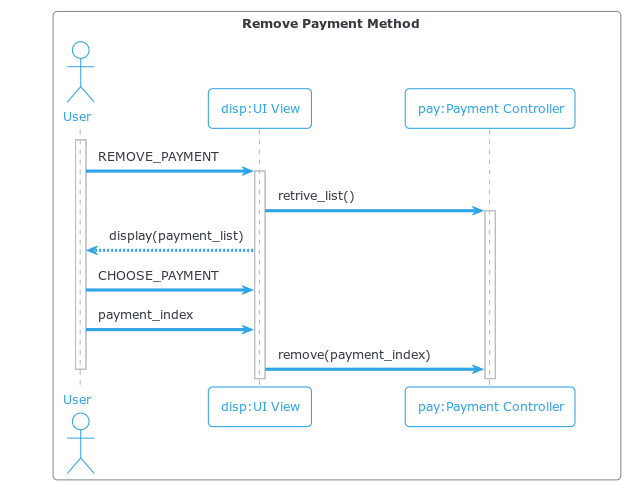
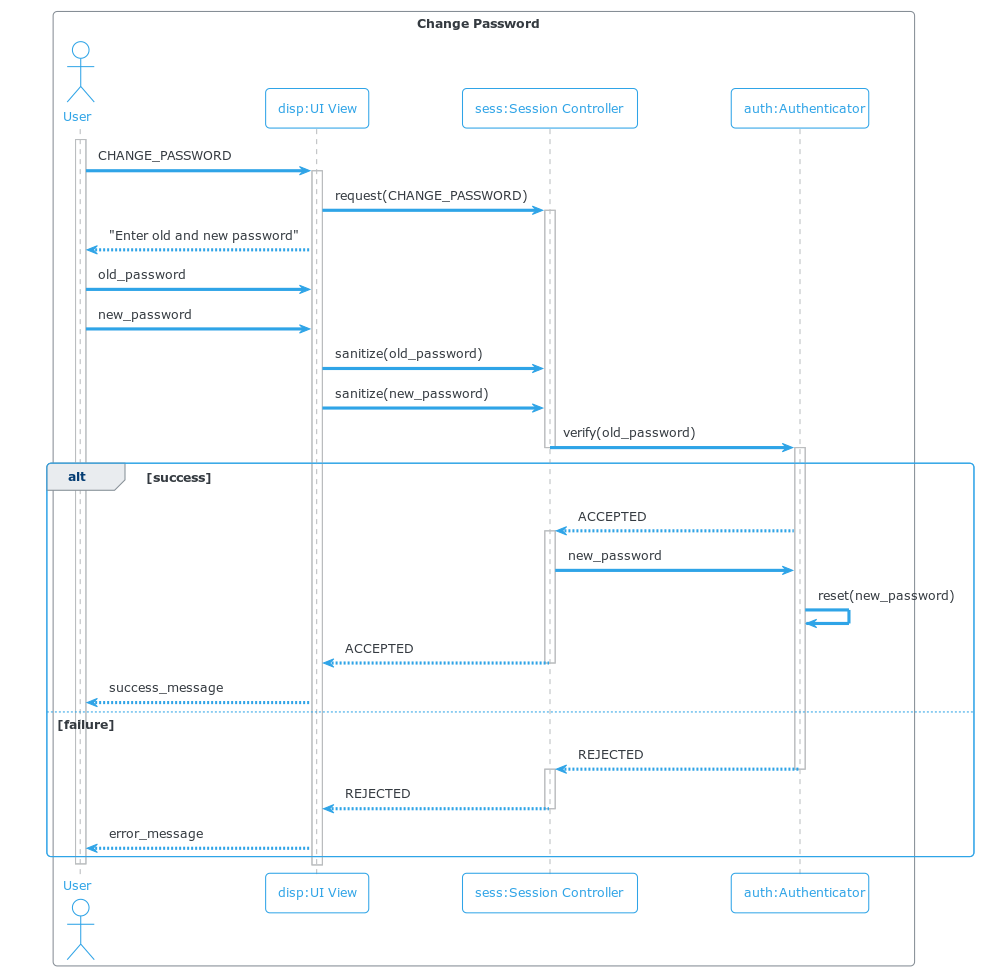
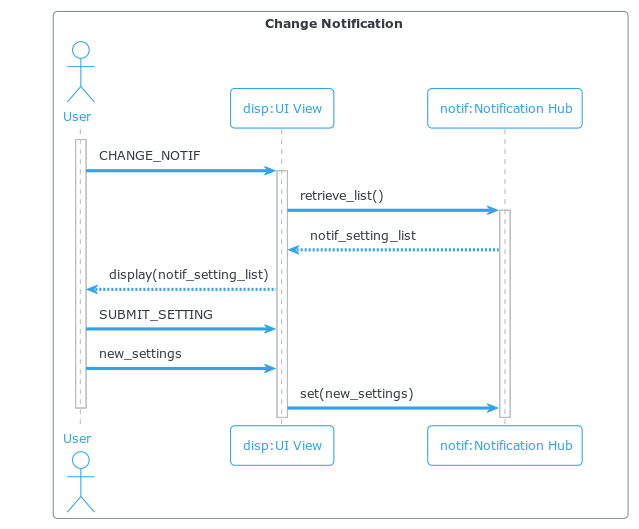
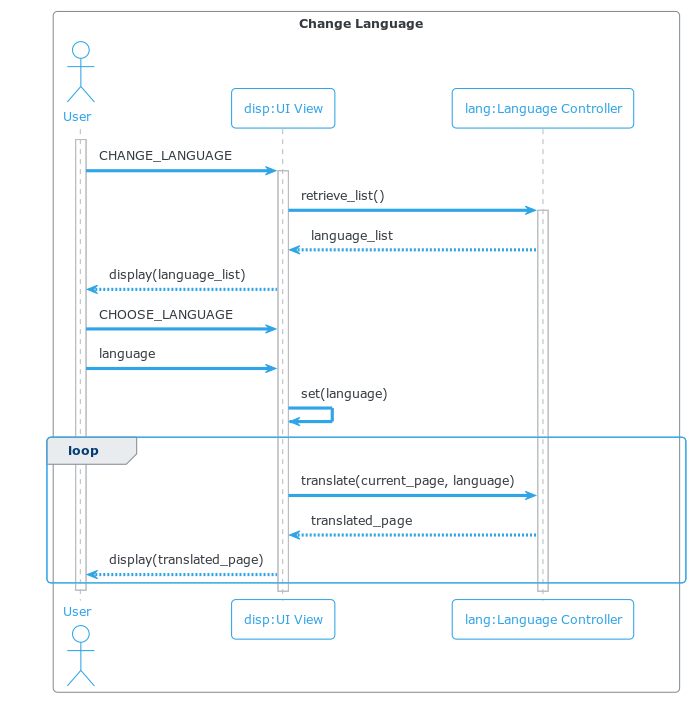
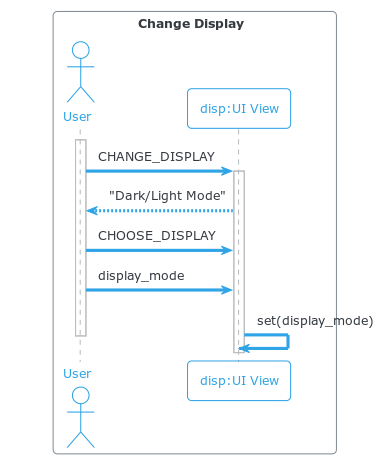
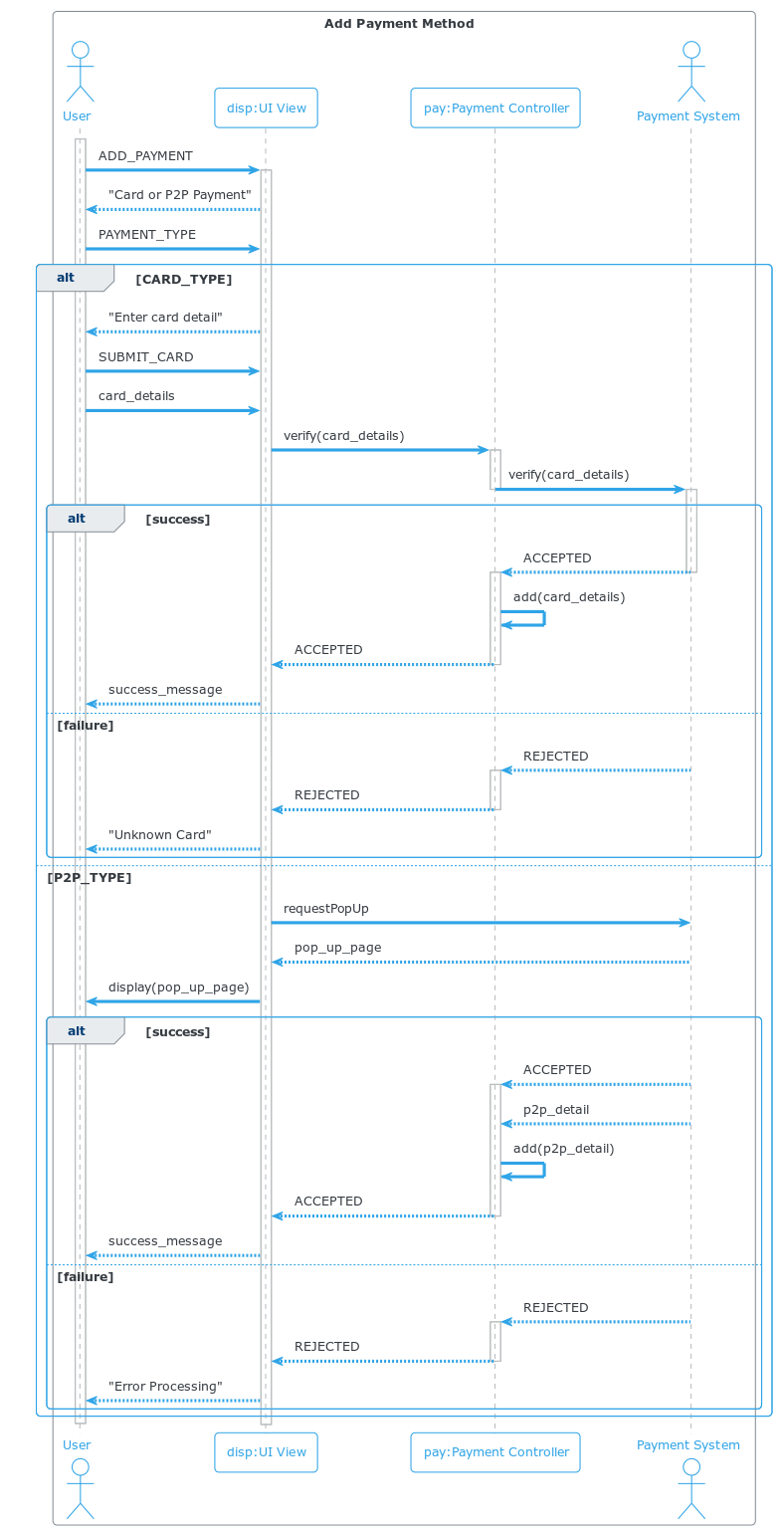


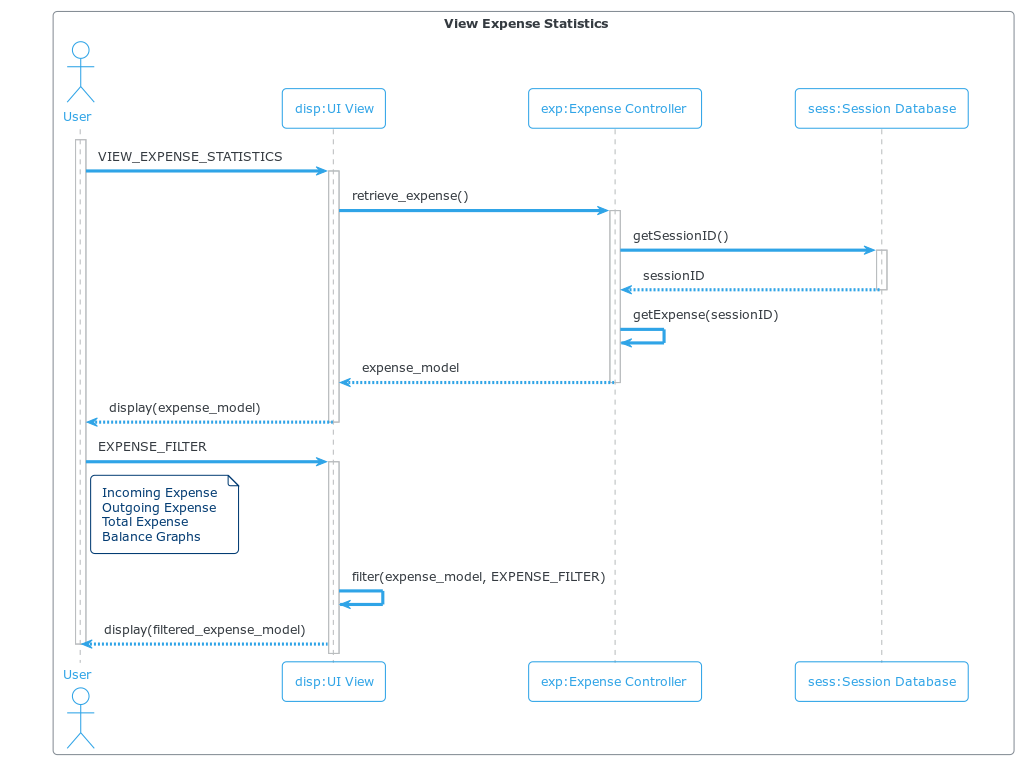
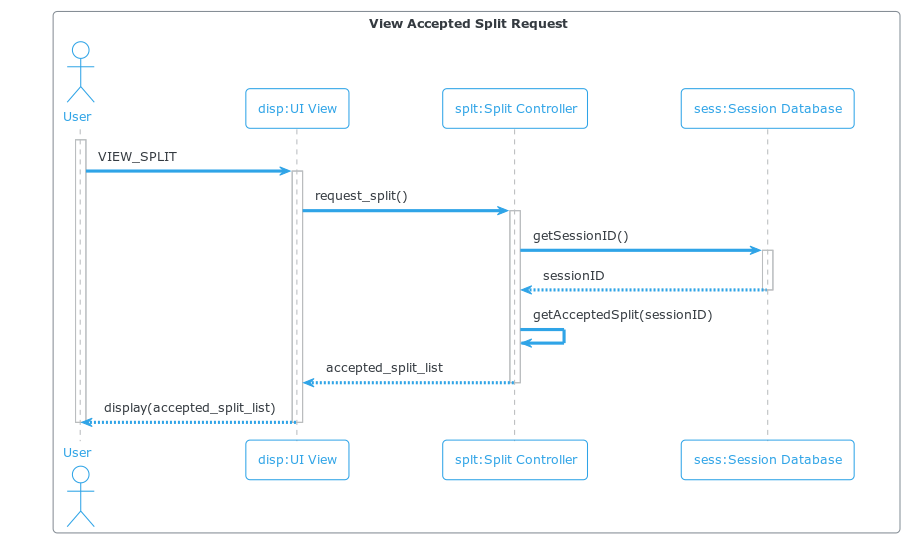
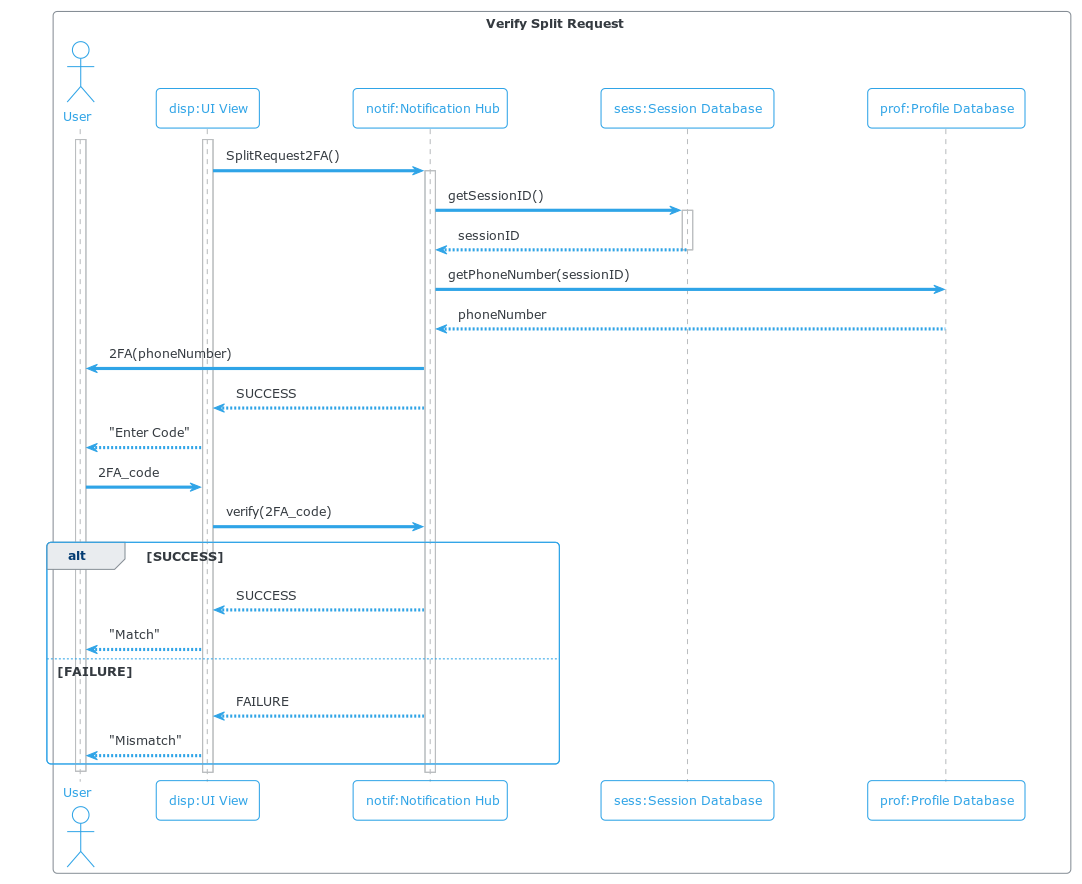
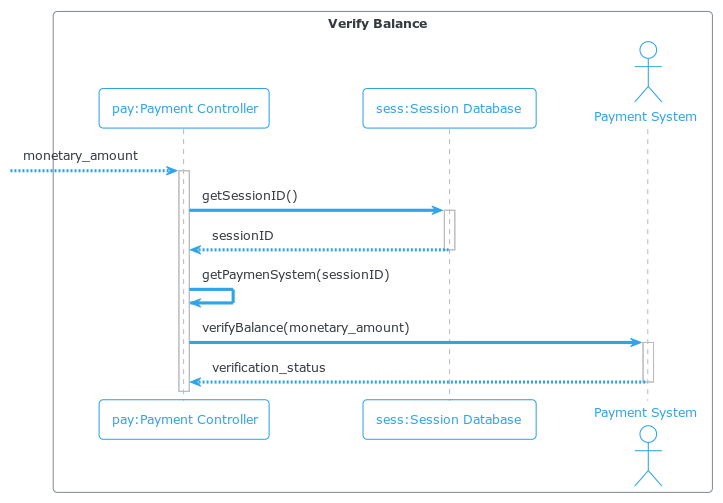
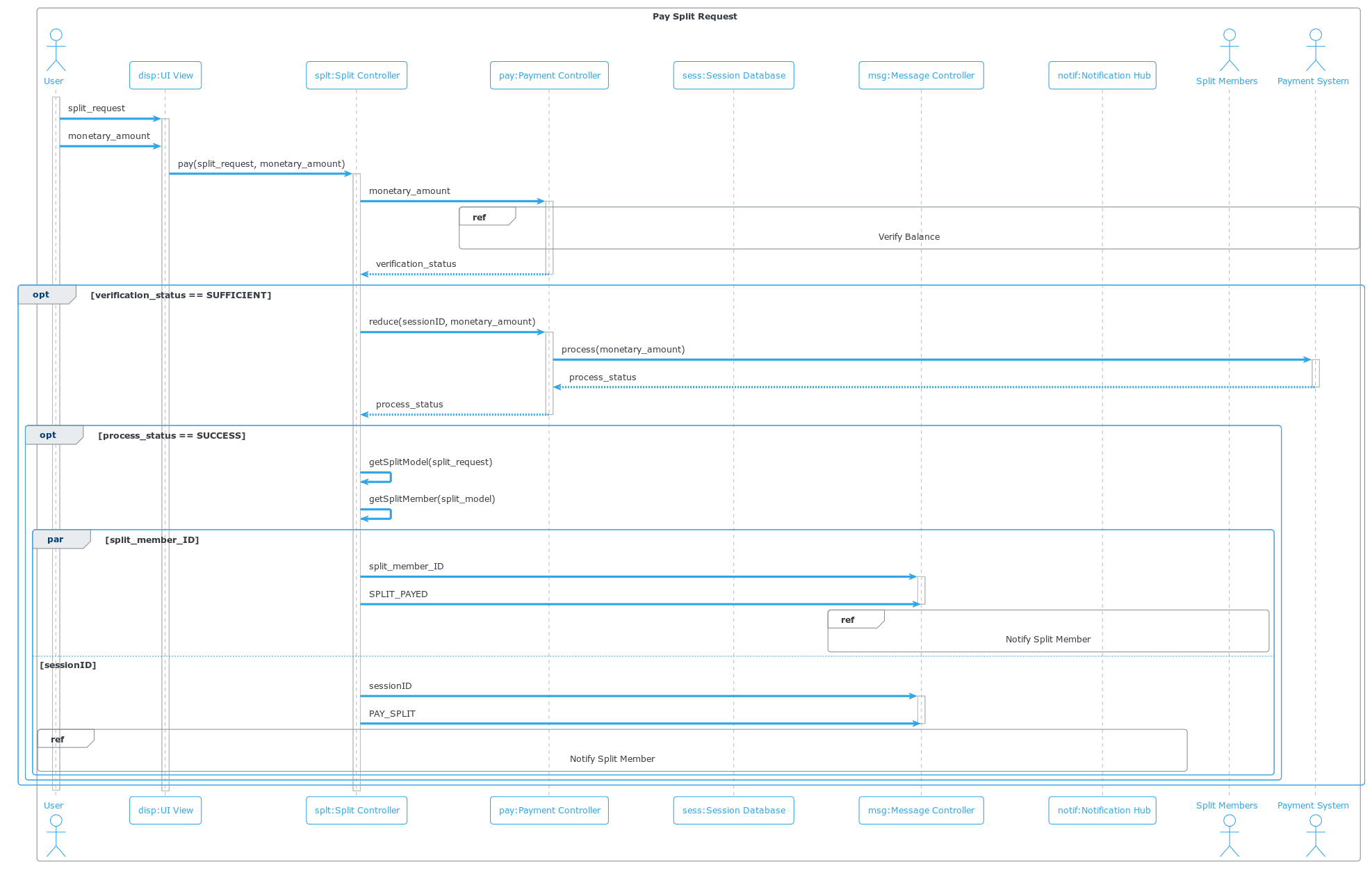
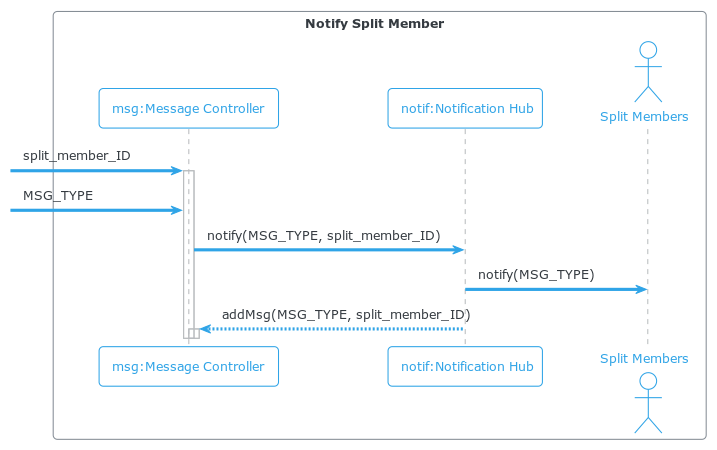
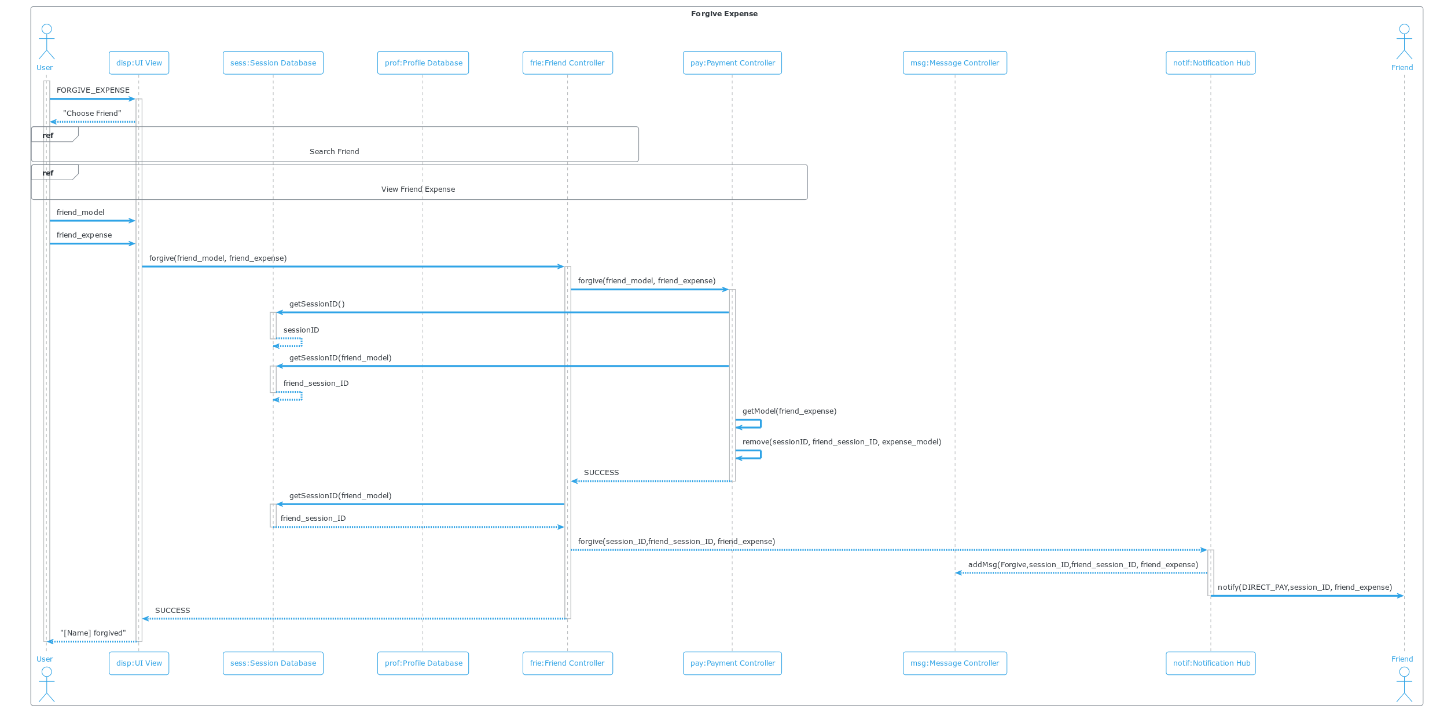
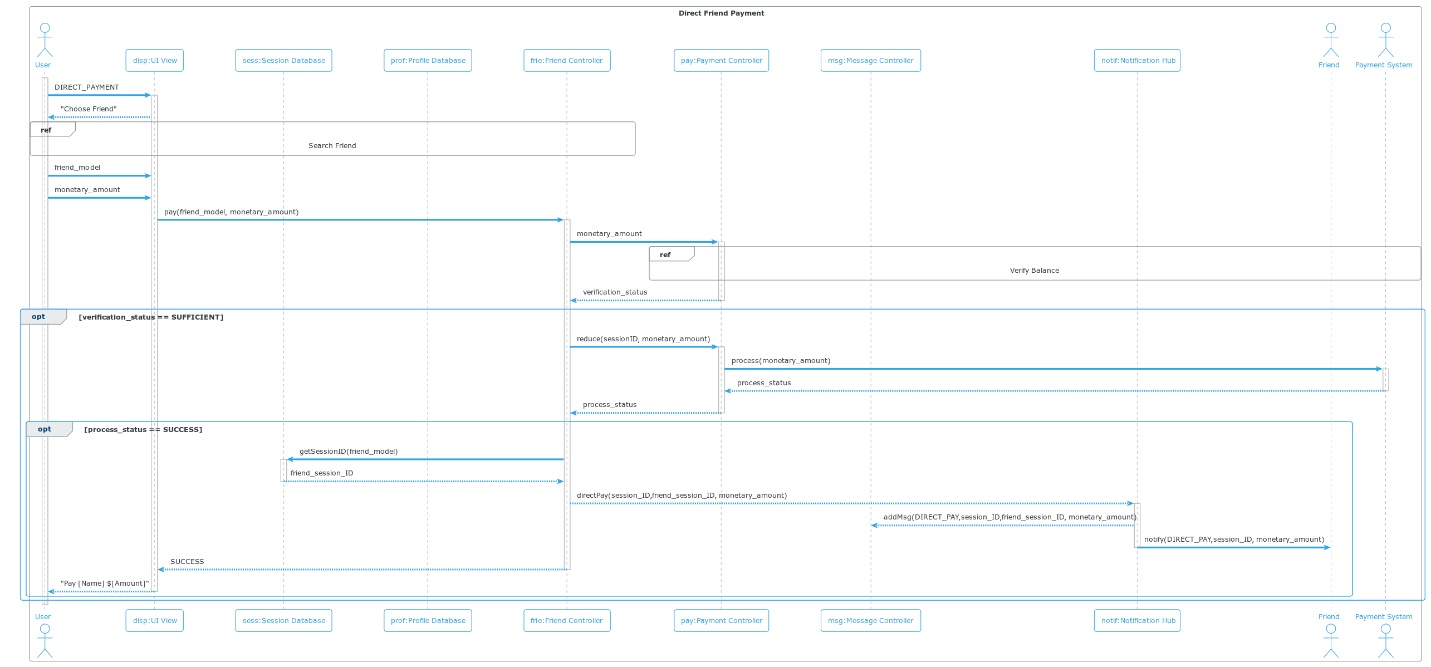
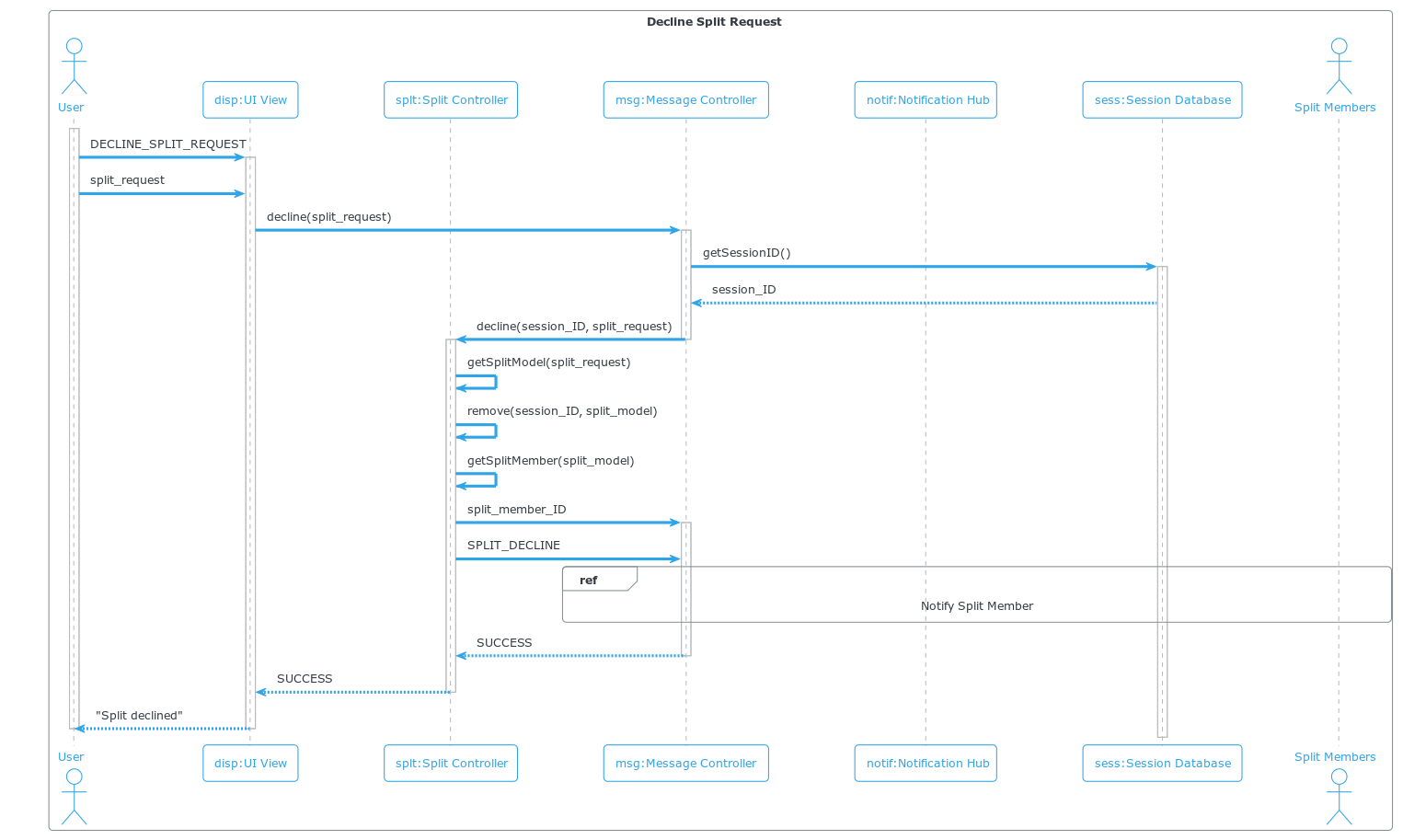
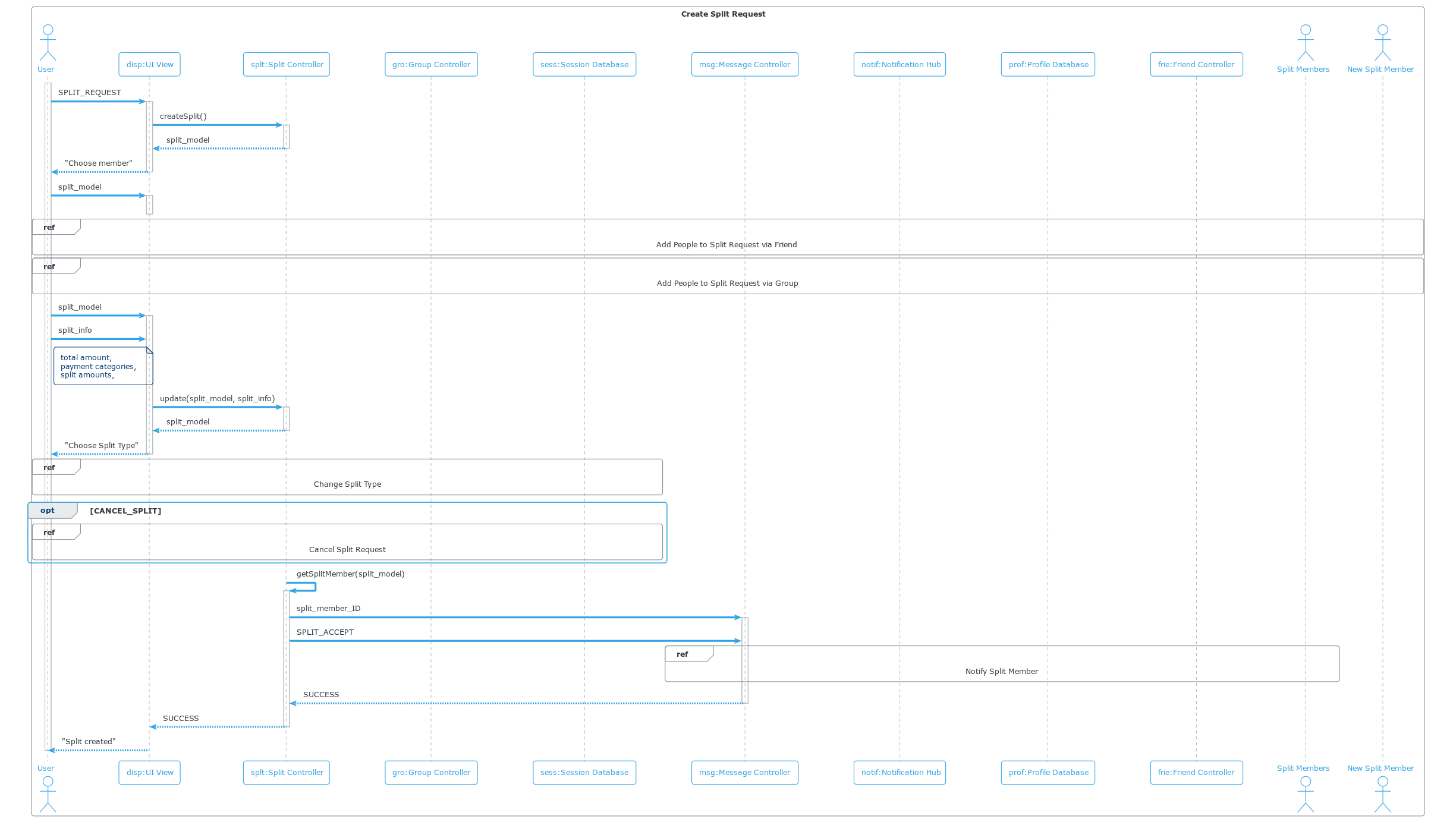
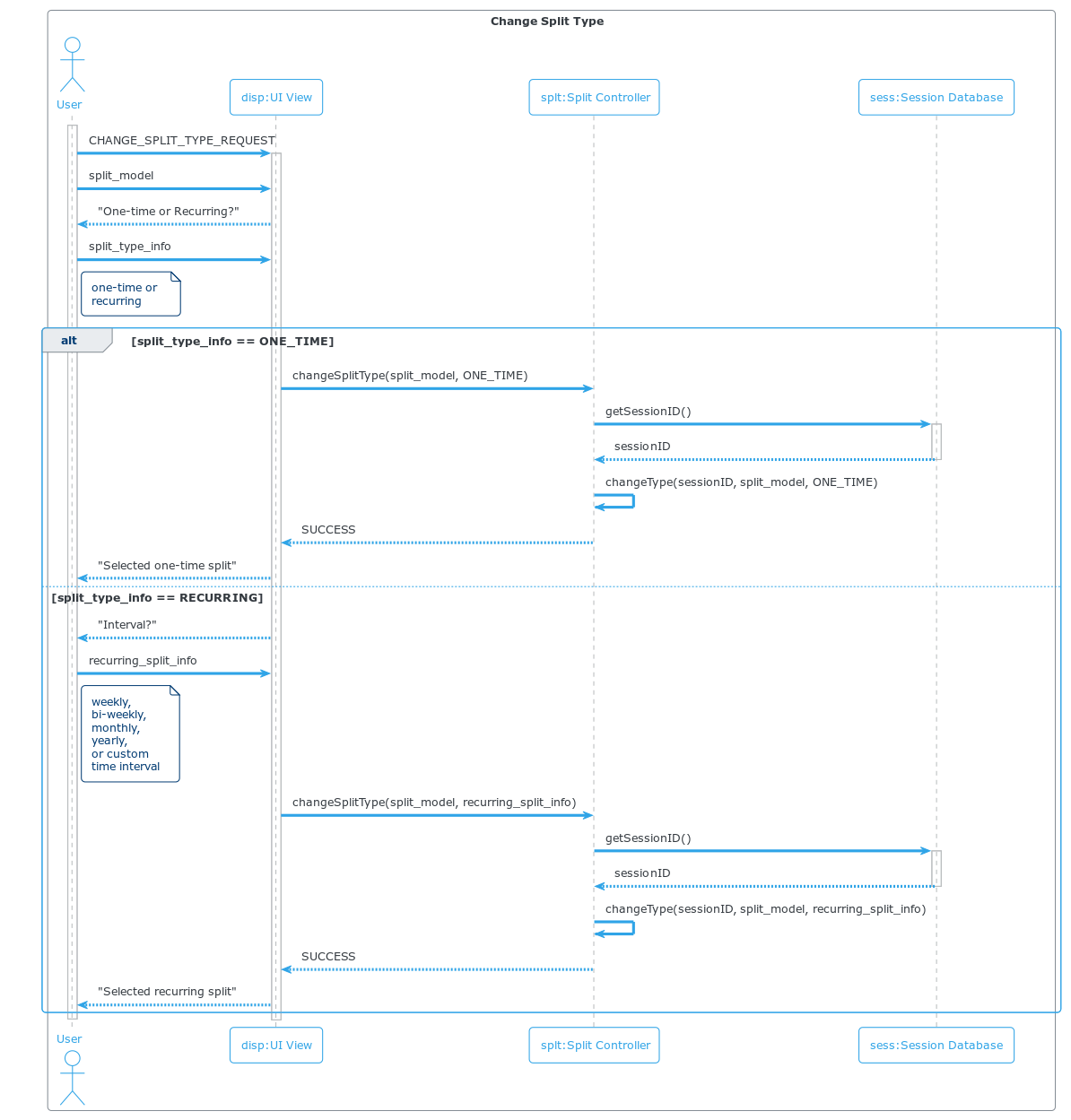
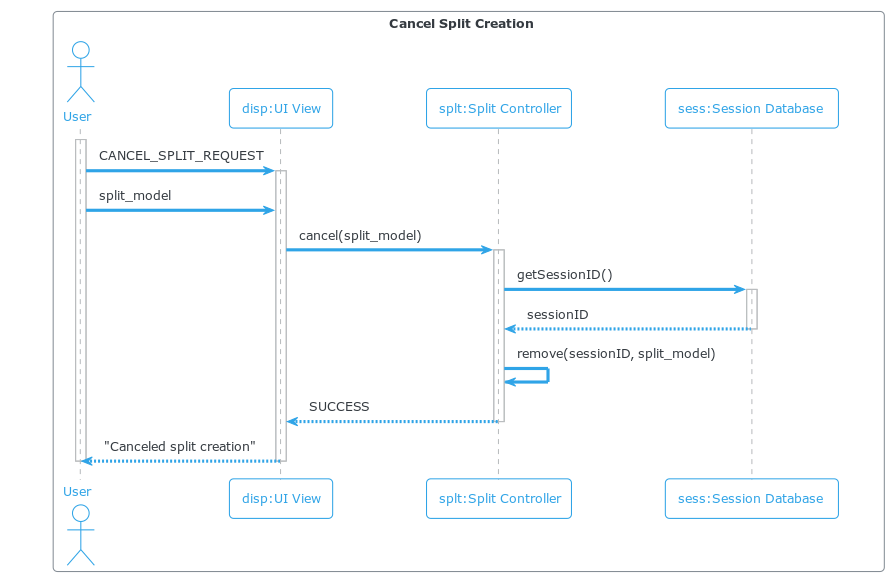
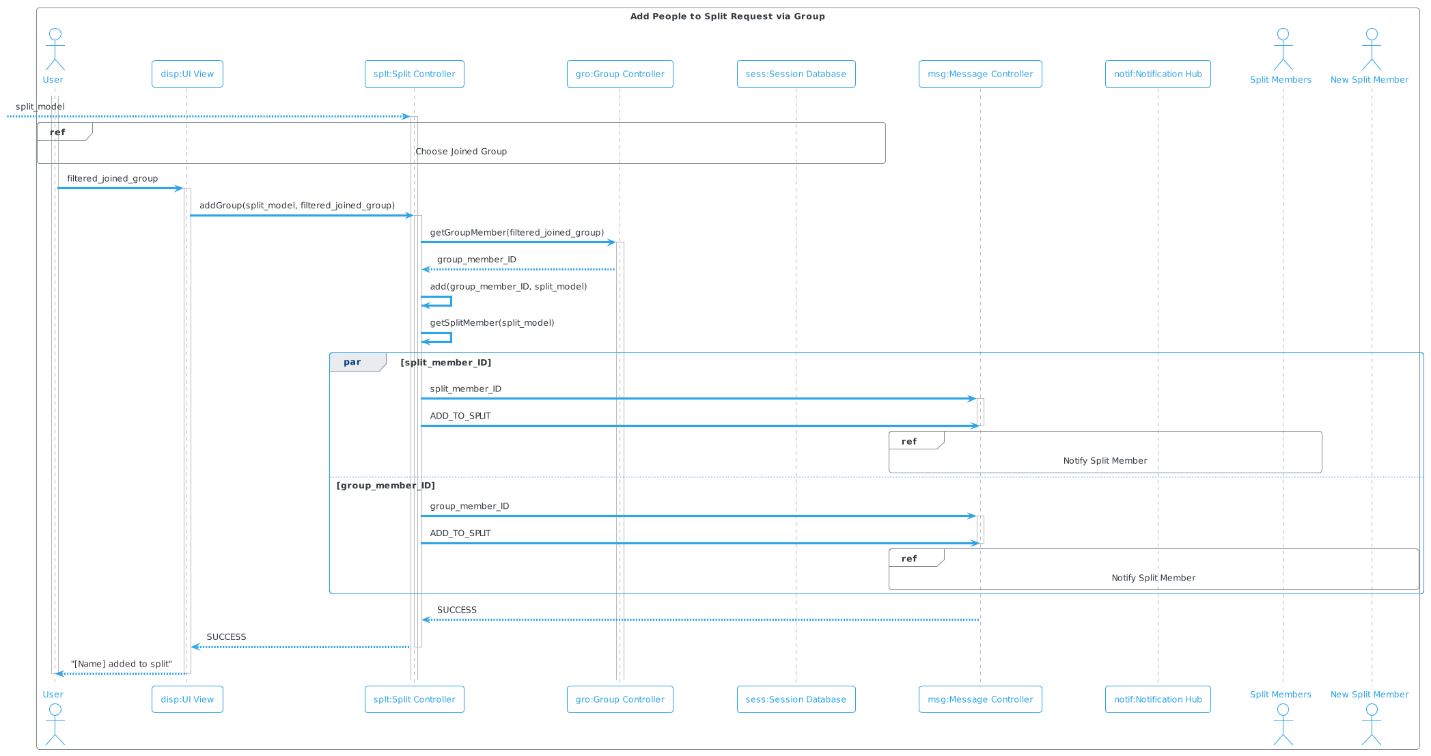
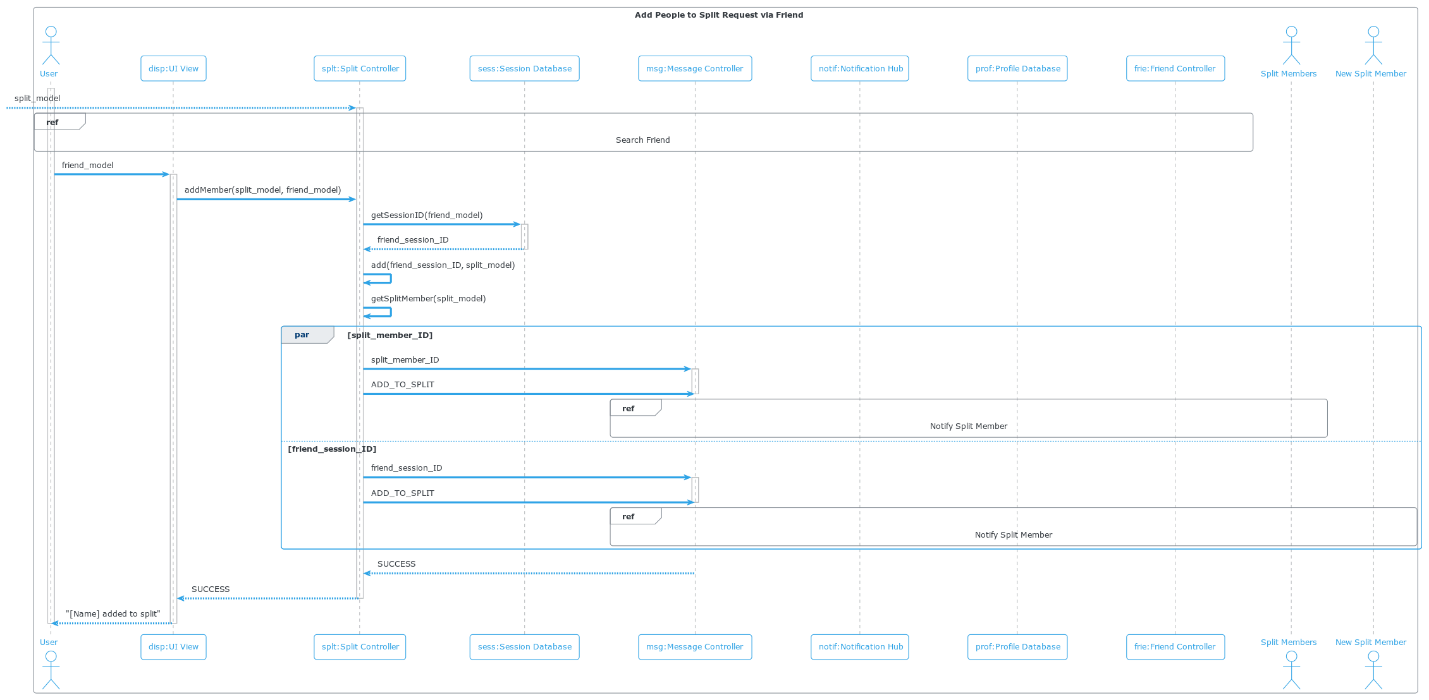
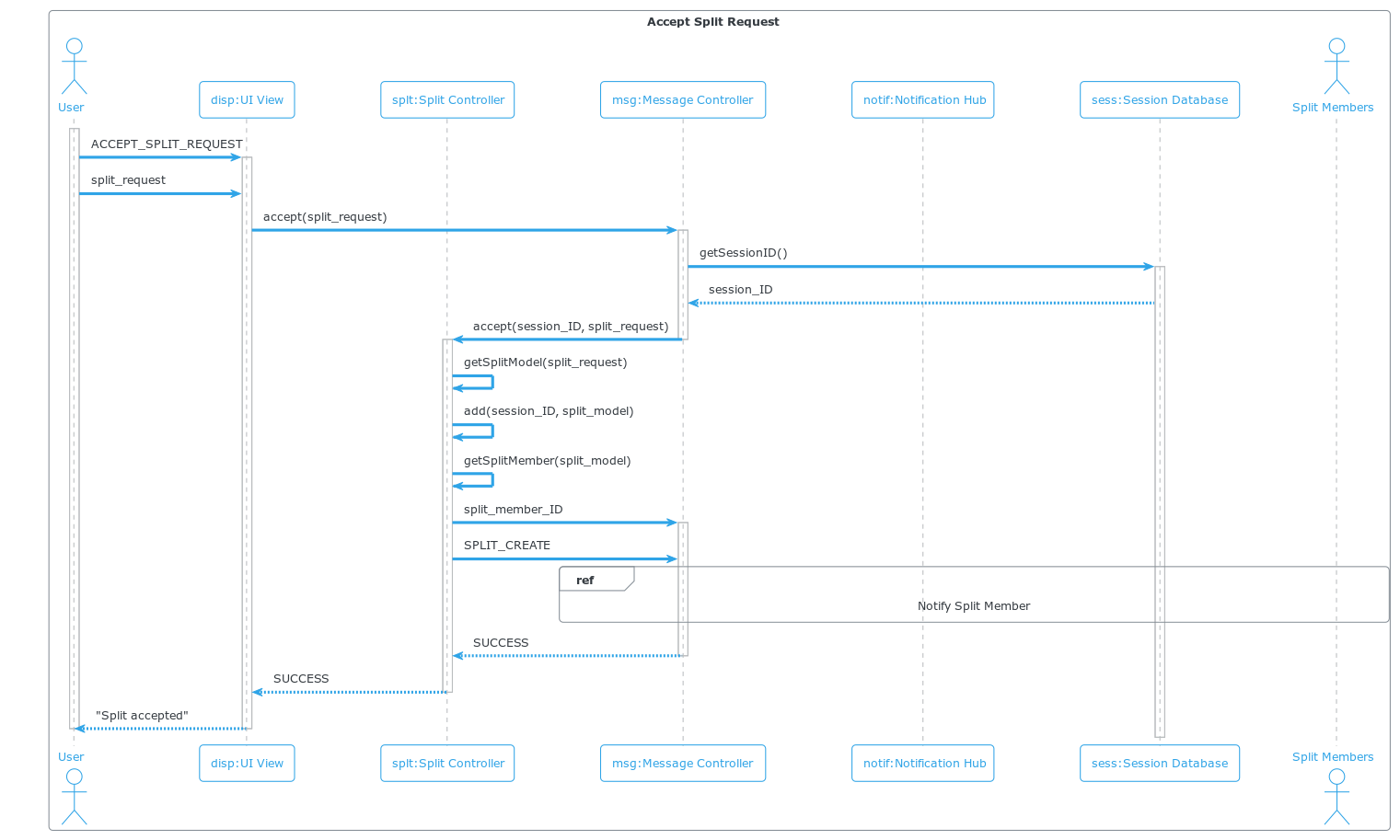
**(D1) 7. [15 POINTS]** Sequence diagram – Provide sequence diagrams (like Figure 5.6 and Figure 5.7) for each use case of your project. Please note that there should be an individual sequence diagram for each use case of your project. (Ch 5 and Ch 7)

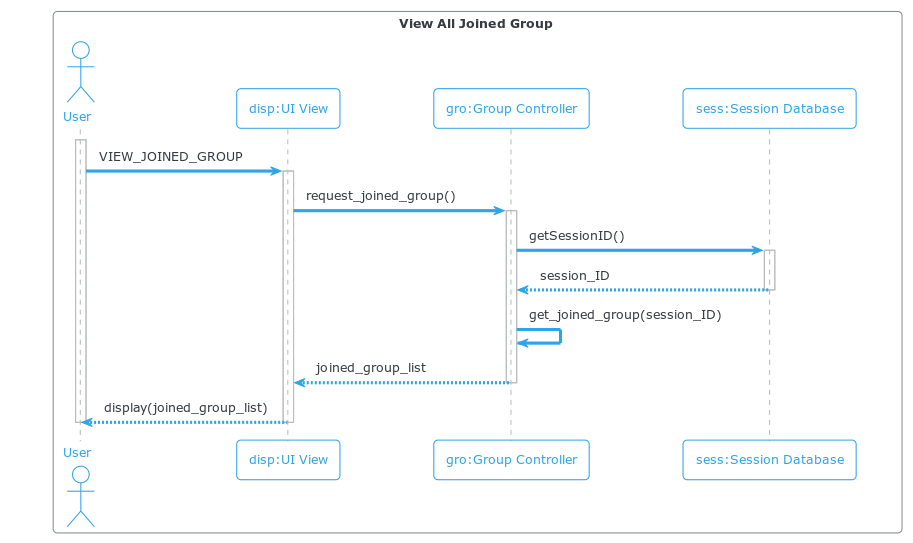
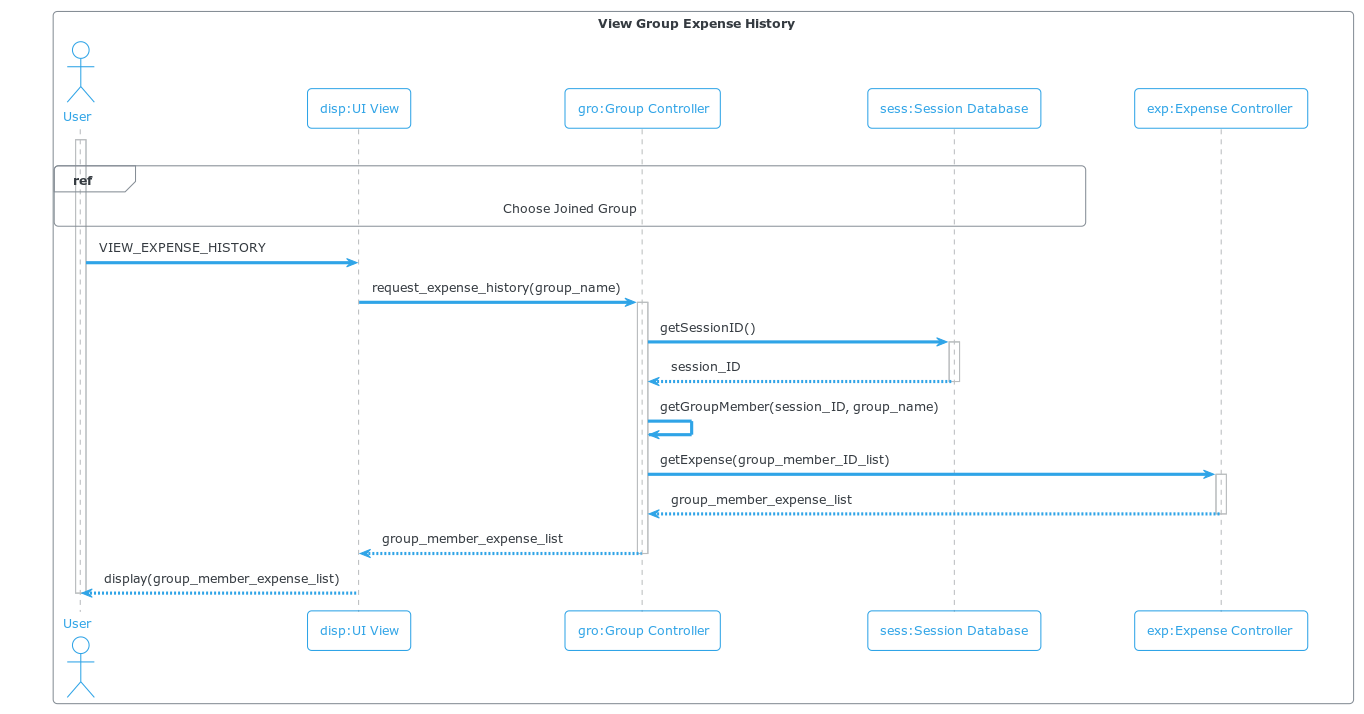
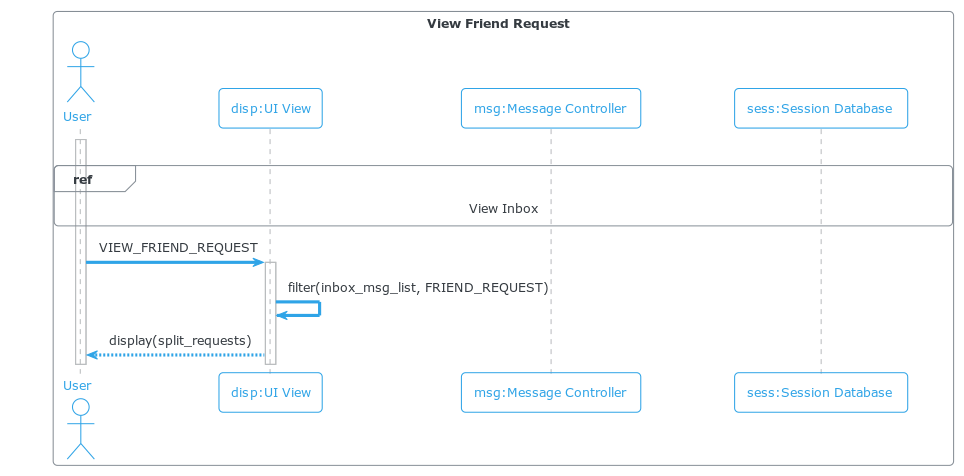
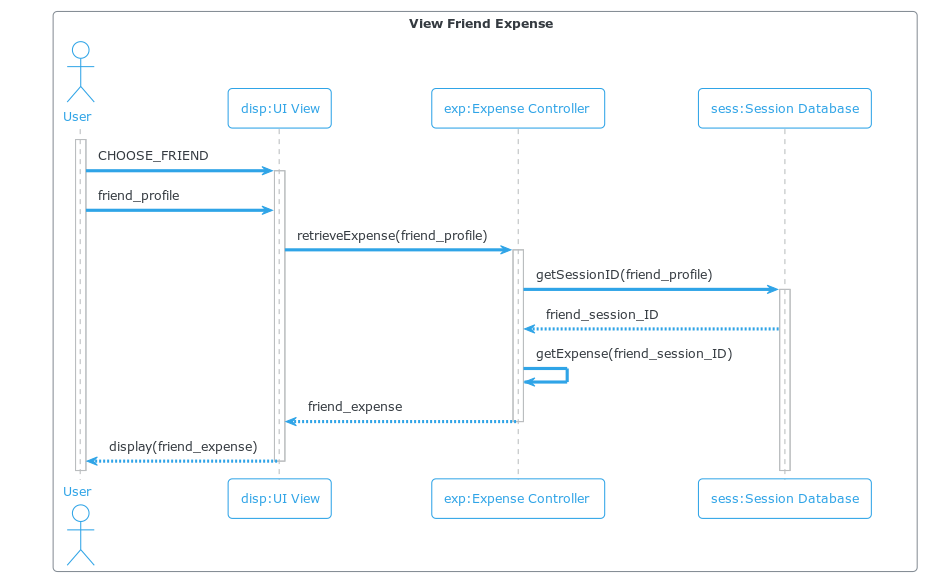
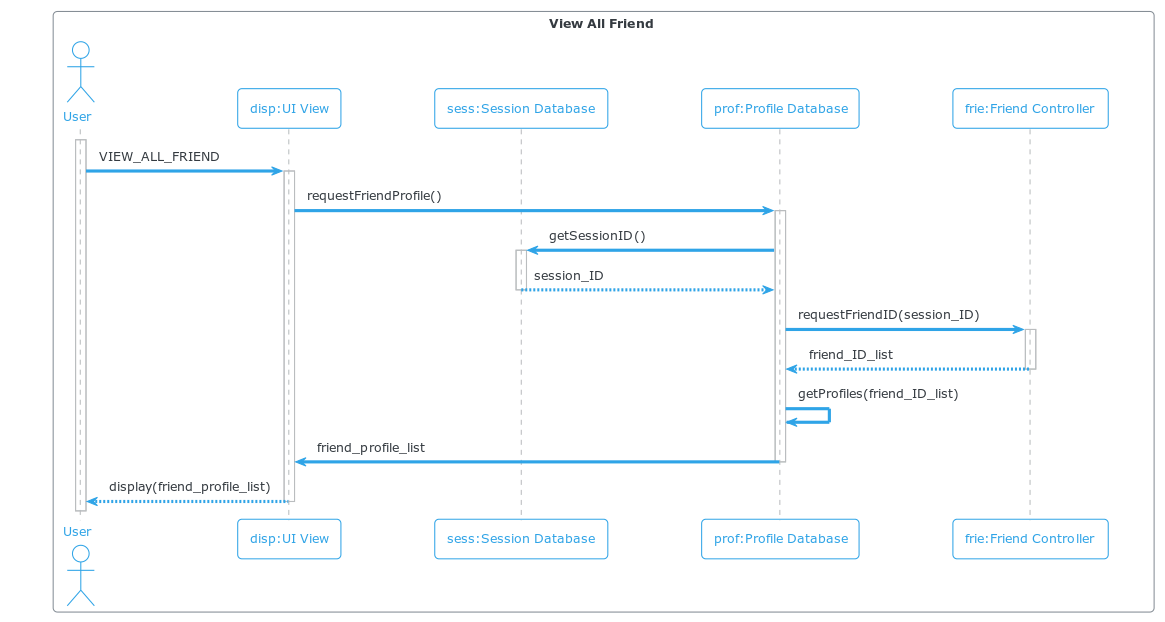
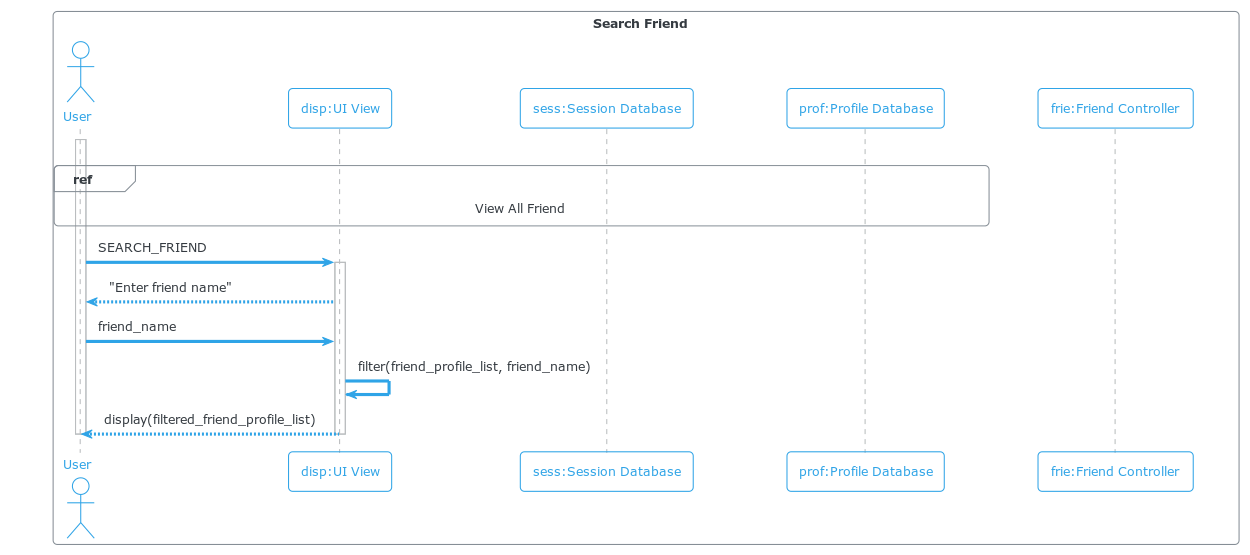
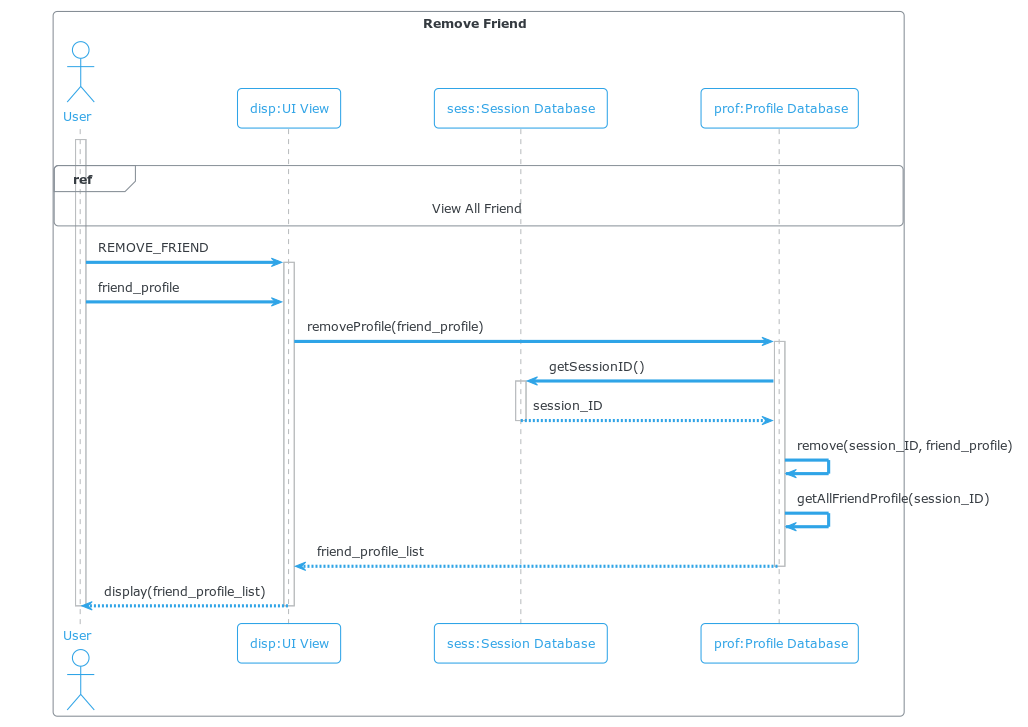
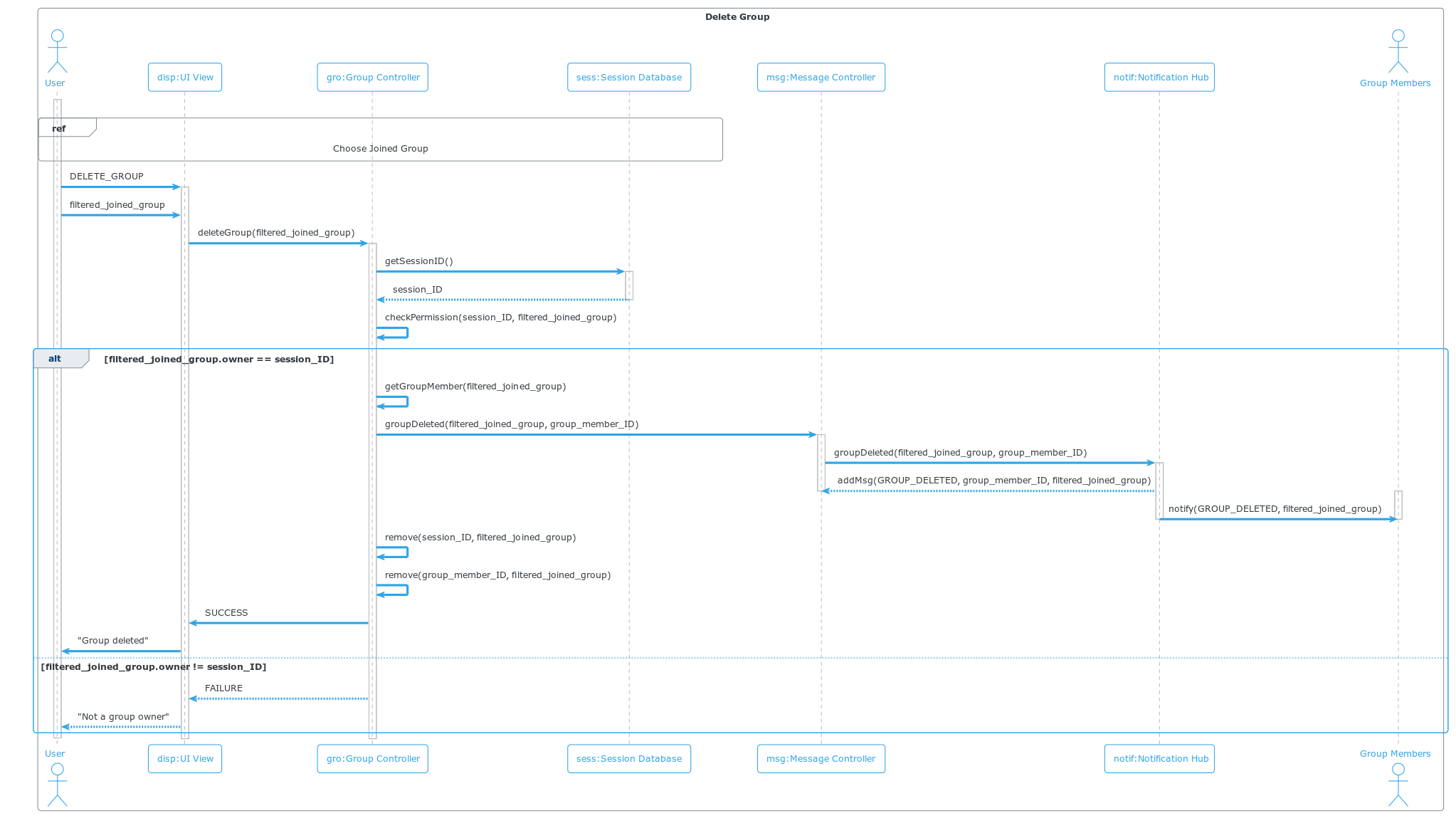
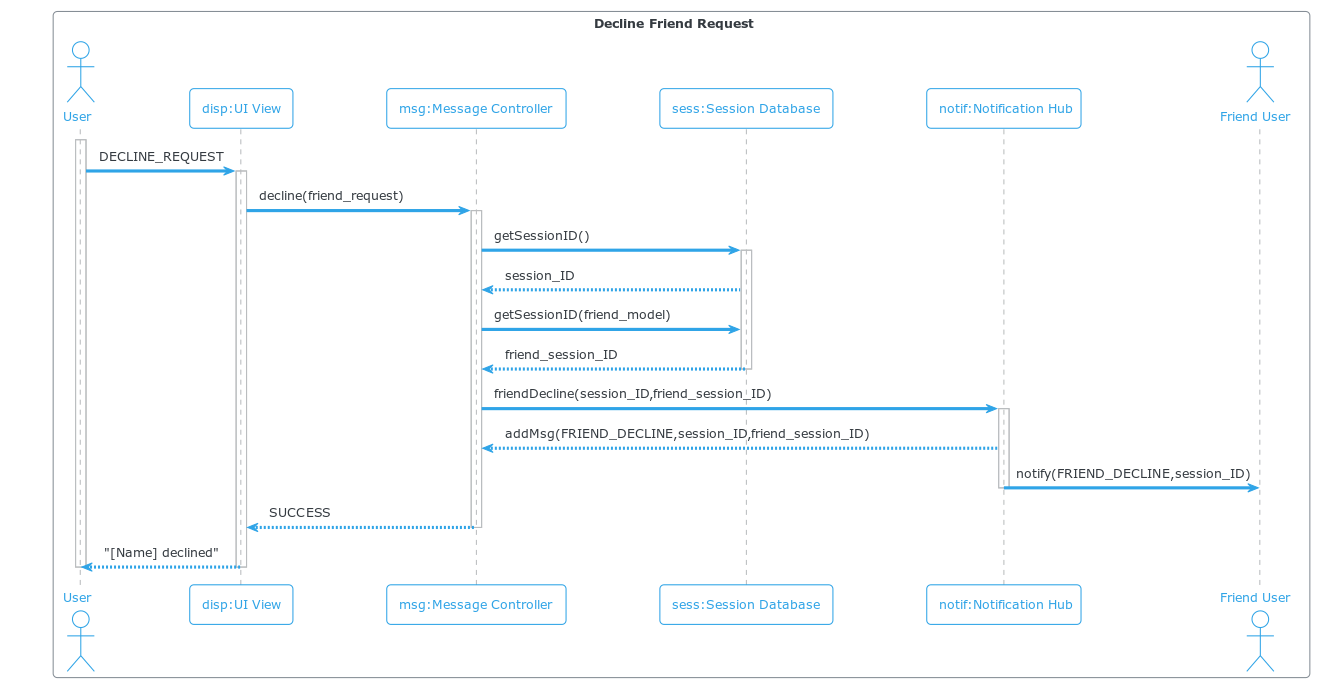
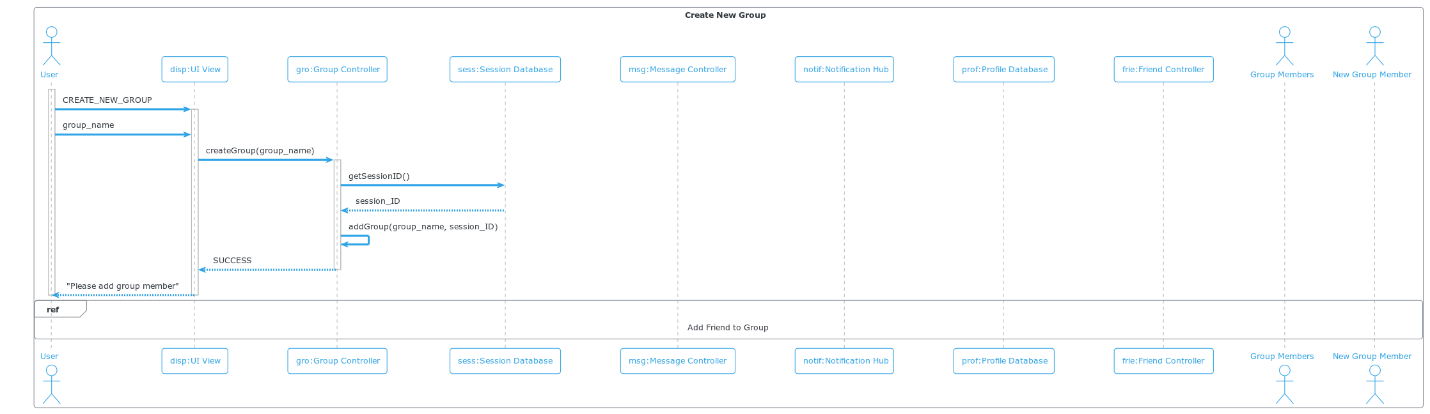
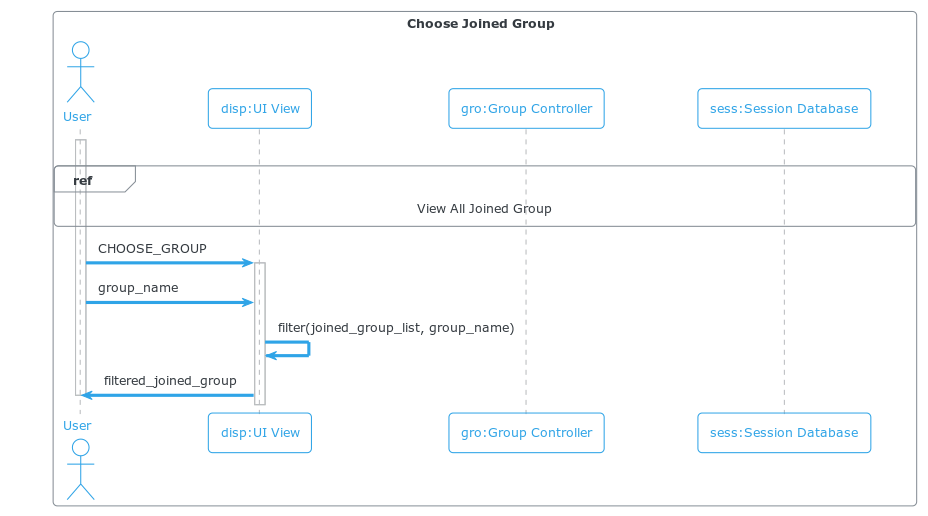
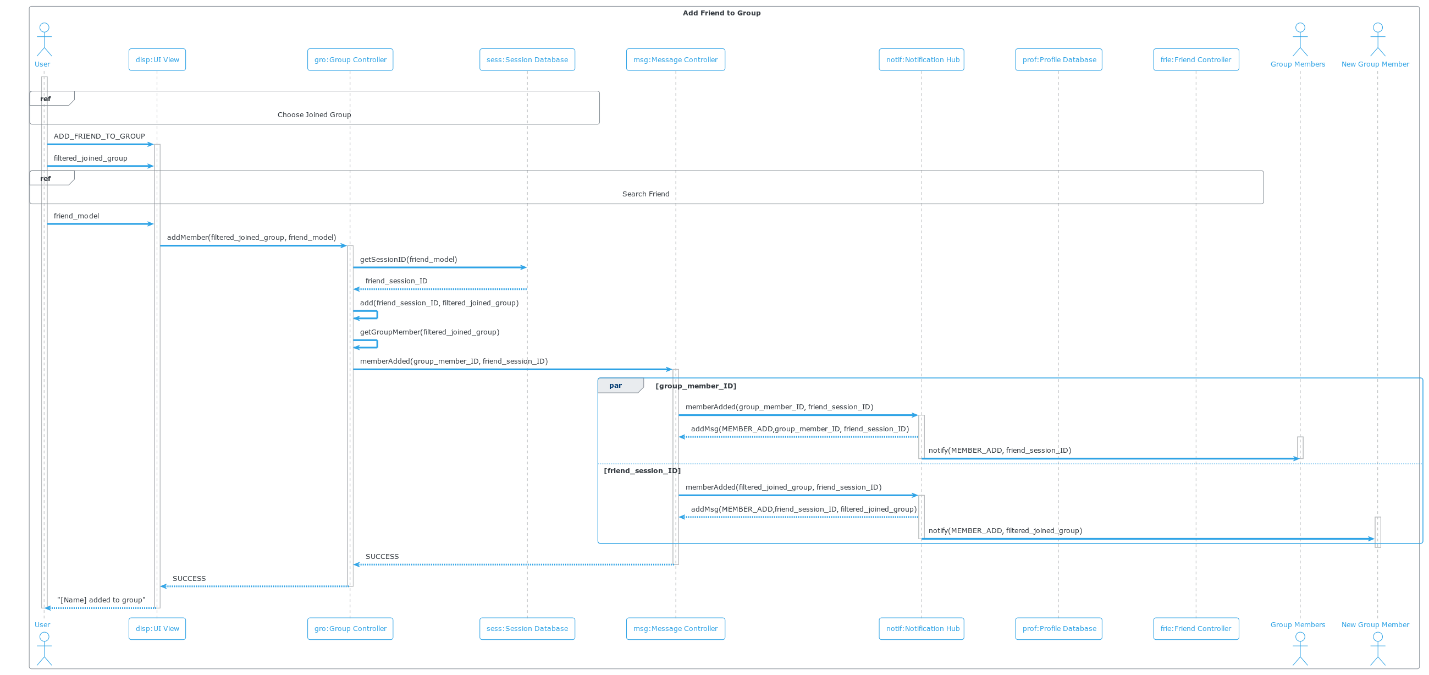
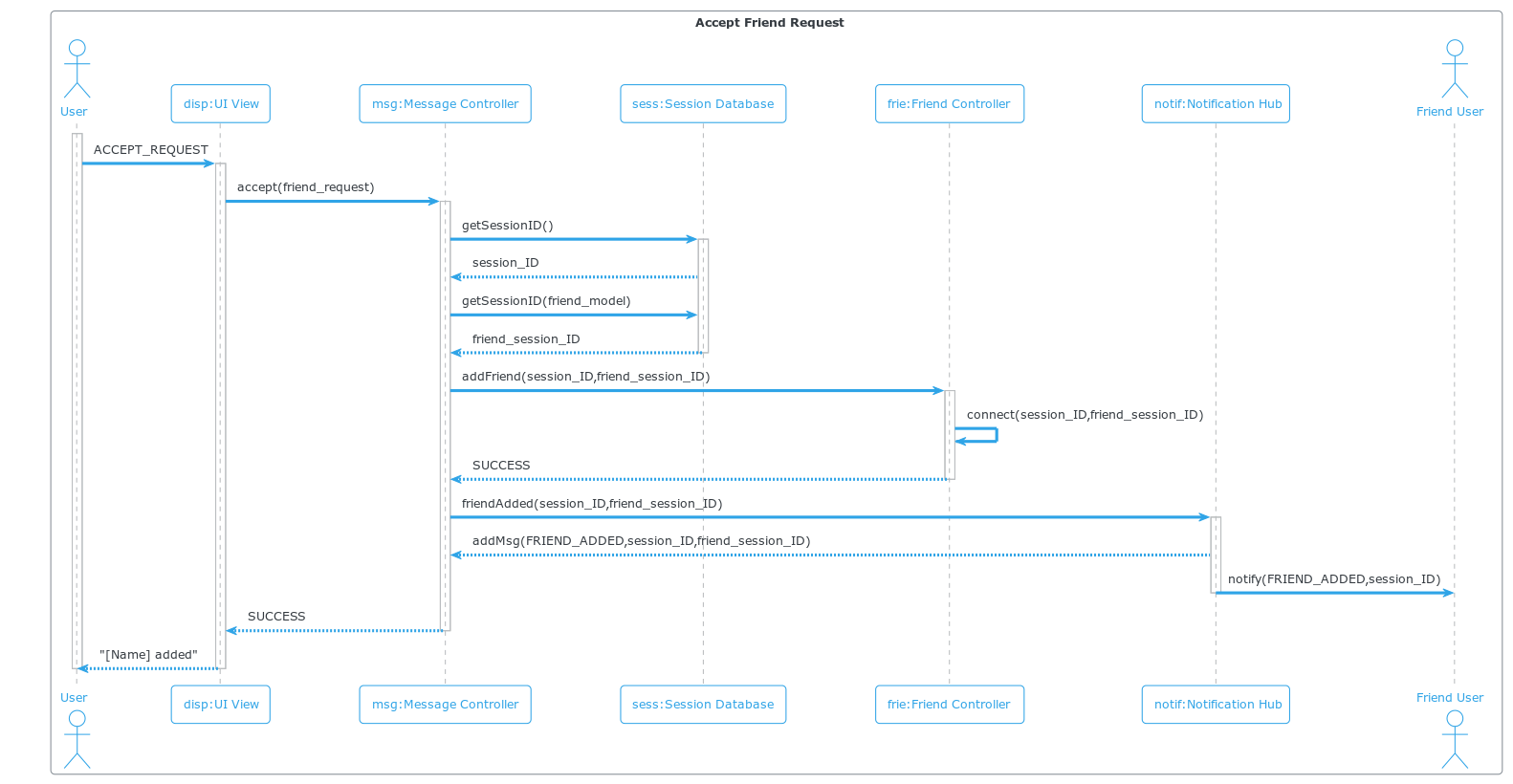
|  |
| --- |
| The sequence diagrams are included on the following pages. However, some are hard to read due to their large width. That is why it may be easier to access them as markdown files in the GitHub repo:   * [Profile.md](https://github.com/md-y/3354-splitsquad/blob/master/Sequence%20Diagrams/Profile.md) * [Split.md](https://github.com/md-y/3354-splitsquad/blob/master/Sequence%20Diagrams/Split.md) * [Squad.md](https://github.com/md-y/3354-splitsquad/blob/master/Sequence%20Diagrams/Squad.md) |

****

****

****

****

****

**(D1) 8. [15 POINTS]** Class diagram – Provide a class diagram (like Figure 5.9) of your project. The class diagram should be unique (only one) and should include all classes of your project. Please make sure to include cardinalities, and relationship types (such as generalization and aggregation) between classes in your class diagram. Also make sure that each class has a class name, attributes, and methods named (Ch 5).

A diagram of a computer

Description automatically generated

**(D1) 9. [15 POINTS]** Architectural design – Provide an architectural design of your project. Based on the characteristics of your project, choose and apply only one appropriate architectural pattern from the following list:

(Ch 6 section 6.3)

**(D1) 9.1. Model-View-Controller (MVC) pattern (like Figure 6.6)**

(D1) 9.2. Layered architecture pattern (like Figure 6.9)

(D1) 9.3. Repository architecture pattern (like Figure 6.11)

(D1) 9.4. Client-server architecture pattern (like Figure 6.13)

(D1) 9.5. Pipe and filter architecture pattern (like Figure 6.15)

A diagram of a user flow

Description automatically generated

*IMPORTANT NOTE: The following items will all need to be calculated / worked on based on the project you are designing. As an example, if a team of 7 students in CS3354 class is working on the development of a hospital information system, this group will prepare the project scheduling, cost, effort and pricing estimation calculations based on the hospital information system design, NOT based on their 7 student team. Think of the analogy to the “Inception” movie: What you will be working on is the dream in a dream, i.e. the dream in the second level, NOT in the first level.*

**3. [35 POINTS] Project Scheduling, Cost, Effort and Pricing Estimation, Project duration and staffing:** Include a detailed study of project scheduling, cost and pricing estimation for your project. Please include the following for scheduling and estimation studies:

**3.1. [5 POINTS] Project Scheduling.** Make an estimation on the schedule of your project. Please provide start date, end date by giving justifications about your estimation. Also provide the details for:

- Whether weekends will be counted in your schedule or not

- What is the number of working hours per day for the project

|  |
| --- |
| **Assumptions:**   * Weekends will not be counted * Working hours per day is 8 * Schedule will follow agile methodology using a scrum approach with sprints * Effort estimation is measured in story points (SP), which is a measure of effort to fully implement a backlog item scaled relative to complexity, risk, and knowledge * Software team size is 10. Average velocity is 2 SP per sprint, so the average team velocity is 20 SP per sprint * Sprints are 2-week intervals   **Schedule Estimation**  For convenience, the simplified product backlog and task dependencies are shown below. Estimating effort for each task with “Planning Poker”, we estimate a total of 70.75 SP to implement.  In one sprint, the team can implement 20 SP. To implement the product, we estimate a minimum of 70.75 SP / (20 SP per sprint) = 3.5375 sprints = 1.76875 months  Accounting for an agreed upon 75% uncertainty for factors such as volatile requirements, testing, holidays, and unexpected hurdles, we adjust our minimum time to be 1.76875 \* 1.75 = 3.0953125 months ~ 3 months for flexibility  Because December month has a lot of holidays and vacation leaves, we estimate   * Project start date: January 2, 2024 * Project end date: March 31, 2024 * Project duration: 3 months   **Schedule Overview**  Following the scrum framework, each sprint will consist of sprint planning, daily stand ups, sprint review, and sprint retrospective.   * Though it is difficult to provide a detailed picture of the task schedule for each sprint due to the flexible nature of our software process, the product backlog shows a general picture that a “minimum viable product” will be implemented first with higher priority items being done first. * Priority is based on if a task has dependents and the task’s importance   **Product Backlog**   * Note that the backlog shown below is a simplified version * Detailed backlog available in Project Scheduling file on GitHub     **Task Dependencies**   * Task dependencies are in the detailed product backlog and visualized in an activity-on-node diagram. Those can be found in the Project Scheduling folder on Github. |

**3.2. [15 POINTS] Cost, Effort and Pricing Estimation**. Describe in detail which method you use to calculate the estimated cost and in turn the price for your project. Please choose one of the two alternative cost modeling techniques and apply that only:

- Function Point (FP)

- Application composition

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| We all decided that using the FP model would be the algorithmic technique for our group project itself due to the fact that there are many benefits when pertaining to implementing this method. For example, we thought this method would be useful since this method has a very standard approach that helps to address all the different factors including the inputs, outputs, and other aspects of the program that contribute to the overall functionality of the project. Additionally, we thought that the FP method would be really useful from a scalability perspective due to the fact that there are many instances during the duration of projects where new resources need to be added for enhancing the overall structure and functionality of the system and as a result, this method would be really useful when it came to understanding how those changes can impact the overall time needed for the project along with the change in price and cost.  **Number of user input:**   * Login Information (Username/Email & Password) * Add Friend (Username/Email Information) * Remove Friend (Username/Email Information) * Update Friend Request Status (Accept/Decline) * Add Group (Group Name & Initial Members) * Add Friend to Group (Group Name & Username/Email) * Delete Group (Group Name w/ Authority) * Create Split Request (Split Info & Members’ Usernames/Emails) * Update Split Request (Split Info & Accept/Decline) * Pay Split Request (Split Info & Payment Info) * Pay Friend Directly (Member Username/Email & Payment Info) * Forgive Expense (Username/Email) * Update Account Settings (New Account Settings)   **Number of user output:**   * Friend Request Notification * Group Invite Notification * Split Request Notification * Payment Notification * Friends view * Profile/Settings View * Transaction History View * Split Creation View * Group View   **Number of user queries:**   * Search Friends * List All Friends * List All Friend Requests * List All Joined Groups * List All Split Requests * View Expense Statistics * View Personal Profile * View Transaction History   **Number of data files and relational tables:**   * User Account Information: * Friends * Payment Methods * Split Requests: * Split Transactions * Split Members * Groups * Group Members * System Configurations * Encryption Keys   **Number of external interfaces:**   * Payment-to-Payment System * Verify Balance w/ Payment System * Verify New Payment Method with Payment System  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | Function Category | Count | Complexity | | | Count Complexity | | Simple | Average | Complex | | 1 | Number of user input | 13 | 3 | 4 | 6 | 52 | | 2 | Number of user output | 9 | 4 | 5 | 7 | 45 | | 3 | Number of user queries | 8 | 3 | 4 | 6 | 32 | | 4 | Number of data files and relational tables | 10 | 7 | 10 | 15 | 150 | | 5 | Number of external interfaces | 3 | 5 | 7 | 10 | 30 | |  |  |  |  |  | GFP | 309 |   GFP=52+45+32+150+30=309  PC1=5 PC2=3 PC3=3 PC4=3 PC5=2 PC6=5 PC7=5  PC8=3 PC9=3 PC10=5 PC11=4 PC12=3 PC13=4 PC14=4  Total= 52  PCA=0.65+(0.01\*52) = 1.17  FP=1.17\*309= 361.53  Assuming there’s 40 FP/person weeks in terms of productivity:  => **361.53/40=9.03825 person weeks**  **Price Estimation**:  Estimated Project Duration: 3 months  Total Hardware Cost: $469.27\*3= $1407.81  Total Software Cost: ($547.5\*3)+25= $1642.5  Total Personnel Cost: ($67066.67\*3)+100= $201,300.01  Total Overall Project Cost: $204,350.32  Selling Price Per User:  Expected Initial Number of Users (within first few months of release): 247.5K  Break Even Cost: $204,350.32/247,500= $0.8257  If we are trying to have a profit margin of 20%, the corresponding selling price per user would be $0.8257\*1.2=$0.99 per user. |

**3.3. [5 POINTS]** Estimated cost of hardware products (such as servers

|  |  |
| --- | --- |
| **Cloud Services** | **Cost** |
| **Azure Cloud Services Virtual Machine (with IIS Web Server)**  A2m v2 processor (2 Cores, 16 GB RAM): $0.195/hour [7]  Considering a 730 hour month: (730)(0.195) = $142.35/month | $142.35 / month |
| **Database** |  |
| **Azure SQL Database**  Standard-Series (Gen 5) Single Database, vCore model (2):   $0.36532/hour and $0.138/GB [8]  Considering a 730 hour month: (730)(0.36532) = $266.68/month  Considering the size of the logs to be about 30%: (0.138)**(**128**)**(1.3) **= $**22.96/month | $289.64 / month |
| **Database Backup**  Monthly Long Term Backup Retention: $0.06/GB/month [8]  Backing up the entire database: (128)(0.06) = $7.68/month | $7.68 / month |
| **Security** |  |
| **Microsoft Defender for Cloud**  Server Plan 2: $0.02/hour [9]  Azure SQL Database Service: $15/server/month [9]  Considering a 730 hour month: (730)(0.02) + (1)(15) = $29.60/month | $29.60 / month |
| **Total Cost:** $469.27/month |  |

**3.4. [5 POINTS]** Estimated cost of software products (such as licensed software, etc.)

|  |  |
| --- | --- |
| **Development Tools** | **Cost** |
| **Visual Studio Code Professional (include Azure DevOps):** $45/months/users. We consider a 10 person-team, which allowed for a discount, with the price listed on the right [10] | $438.75 / month |
| **Third-Party APIs** |  |
| **Stripe**: Stripe is free to set up and the company does not charge a monthly or annual fee for its services. [11] | $0 |
| **Microsoft Notification Hubs**: The first 10 million pushes are included for 200,000 active devices, which we can assume our apps have the user base below that range. Thus, we only need to pay the default rate of $10/month for the Basic plan. [12] | $10.00 / month |
| **Testing Tools** |  |
| **Vite & Vitest**: Free | $0 |
| **Playwright**: $0.01/minute testing on Linux. We can roughly assume approximately one test for one function point. Thus, a function points score of 361.53 roughly translated to 362 tests in the test suite. We can also assume roughly one second per test. Given that we have 10 developers who will be running a test suite roughly 5 per day, we will run roughly 1500 test runs per month. In total, Microsoft Playwright Testing will incurs $90.50/month [12]. | $90.50 / month |
| **App Store Fees** |  |
| **Apple App Store**: $99/year. [13] | $8.25 / month |
| **Google Play Store**: $25 one-time registration fee [13] | $25 |
| **Total Cost:** $547.5/month + $25 one-time fee | |

**3.5. [5 POINTS]** Estimated cost of personnel (number of people to code the end product, training cost after installation)

|  |  |
| --- | --- |
| **Premises** | |
| 1. The development team consists of 10 members, all of whom are developers with nominal experience. 2. Estimated salary for a developer with nominal experience is $80,000/year (~$6,667/month or ~$1,539/week). 3. All members of the development team are experienced in Visual Studio Code. | |
| **Training** | **Cost** |
| **Visual Studio Code:** Since all members are experienced in this tool, no training is necessary. | $0 |
| **Azure DevOps:** A training path is provided by Microsoft, and it’s free. No advanced experience is truly required, so a beginner’s course is sufficient [15]. | $0 |
| **Vite & Playwright:** A course guide for Vite and Playwright can be offered at udemy.com [16] [17]. Using the team plan that the website offers, the team can afford each of these courses with each member being billed at a rate of $30/month [18]. | $300 / month |
| **Vitest:** A comprehensive guide is already provided on the official website, so minimal training is required (we’ll assume an extra total $100 to cover any additional resources needed) [19]. | $100 |
| **Achievement Rewards** |  |
| **Employee of the Month:** A team member is recognized for their outstanding work each month and rewarded with a bonus! | $100 / month |
| **Total Cost:** $67066.67/month + $100 | |

**4. [10 POINTS]** A test plan for your software: Describe the test plan for testing minimum one unit of your software. As an evidence, write a code for one unit (a method for example) of your software in a programming language of your choice, then use an automated testing tool (such as JUnit for a Java unit) to test your unit and present results. Clearly define what test case(s) are provided for testing purposes and what results are obtained (Ch 8). Include your test code as additional document in your zip file submitted.

|  |
| --- |
| For our project, we implemented two types of testing: unit and integration testing. Unit testing is done with Vitest which tests individual functions and components. Integration testing is done using Microsoft Playwright which tests the entire system within a browser environment. Both methods are key to ensuring the reliability of our entire system. By writing test cases for individual methods, we ensure that each method performs exactly what it is supposed to do. By writing test cases that test entire webpages, we ensure that all subsystems work together correctly.  Our example test plan focusses on the login subsystem. This subsystem is one of the most critical in the entire system since users cannot use the system without it. Furthermore, we must test this system to ensure it is secure. That is why we implemented both unit and integration tests for it.  For our example of unit testing, we used Vitest to test the password hash generation function. The function accepts a password and salt argument and returns a SHA256 hash of these two strings. To fully test these two inputs, we wrote four test cases that simulate the full variety of potential user inputs. The first case inputs a normal password and salt such as “password123” and “salt123.” The next case inputs empty strings as the password and salt. Conversely, the following case inputs 200-character long strings as arguments. These two cases test inputs way below and above the expected string lengths, but it is still important to test these values to ensure all possible user inputs are tested. The final test case provides UTF-8 characters as the password and hash. Specifically, Emojis and Hiragana characters are used as string values. These characters are outside of the normal ASCII range, but they are still accepted passwords, so it is important to ensure no characters are invalid. Within each case, the return value is compared to a manually calculated hash string based on the test inputs. If the return value and string match, the test case succeeds. Thankfully, our function works as intended, and all the test cases pass.  For our example of integration testing, we used Playwright to test the login page in a browser. For each test case, Playwright creates a new Chromium instance that fills in our login form with user data and records the result. For our first test case, the system inputs correct user info. The case then waits for a response from the server, and then it compares it to the expected result. The response should be a positive one with a valid session key, so it verifies that a good session key was returned. If it cannot do so, the test case fails. For our other test case, the system inputs incorrect user info. The expected response is therefore a negative one. That is why the test case waits for the response and ensures that the response is an error. Thankfully, this is what happened, and both of our test cases passed. Because of this, it is safe to say that our login page works correctly.  The test scripts are included in the submitted zip file, but they are also linked below. More information about how to run each script is in [this](https://github.com/md-y/3354-splitsquad/tree/master/Implementation#splitsquad) Readme.   * Vitest script: <https://github.com/md-y/3354-splitsquad/blob/master/Implementation/src/lib/util/server.test.ts> * Playwright script: <https://github.com/md-y/3354-splitsquad/blob/master/Implementation/tests/login.spec.ts> |

**5. [10 POINTS]** Comparison of your work with similar designs. This step requires a thorough search in the field of your project domain. Please cite any references you make.

|  |
| --- |
| In the realm of cost-sharing apps, our platform stands out as a streamlined facilitator for splitting expenses among groups, particularly focusing on meals and various group purchases. When we look at existing apps in this field, there are a few that come close to what we offer. Venmo, for example, lets users split bills and expenses, but it's more geared toward individual transactions rather than group purchases [20] [21]. Split wise is another app that's commonly used for splitting expenses among groups [22]. However, it doesn't have a direct payment feature and is more of an expense tracker. It's great for keeping tabs on shared costs but doesn't handle payments directly. [23]  Our app, on the other hand, merges the features of splitting costs among groups and facilitating direct payments, providing a comprehensive solution for communal spending. This sets us apart by offering a more user-friendly interface and efficient payment processing, which isn't fully covered by the existing options available. |

**6. [10 POINTS]** Conclusion - Please make an evaluation of your work, describe any changes that you needed to make (if any), if things have deviated from what you had originally planned for and try to give justification for such changes.

|  |
| --- |
| Overall, our work for this project was comprehensive and thorough. We considered an initial project scope that could implement features that would be competitive in the current financial markets, especially with Venmo and Splitwise as competitors.  To give a clearer idea of our product, we used structured and visually appealing diagrams such as use case diagrams, sequence diagrams, class diagrams, and MVC pattern diagram along with functional and non-functional software requirements, all of which were clear and precise.  To accommodate volatile requirements, we chose Agile methodology with a Scrum framework. With our methodology in mind, we could construct a flexible project timeline with a duration of approximately 3 months. Furthermore, we intensively researched factors that affect hardware, software, and personnel cost to construct a representative estimate for total cost of approximately $204,350.32. To turn a profit margin of about 20%, we calculated a selling price per user of $0.99. Considering our “imaginary” software team size of 10 developers with average skill, these estimates are reasonably within the time and budget constraints.  For testing our software, we considered unit testing with Vitest to test individual functions and components and integration testing with Playwright to test pages and routes.  For working on our project and demo implementation, we collaborated well using GitHub for version control and Discord for communication, and we have a beautiful set of presentation slides to show for it.  One notable improvement that we could make was following Agile methodology and Scrum sprints more closely to better assess risks, changing requirements, and customer feedback. Though, since we had no real-life customers to receive feedback from, it was hard to adopt Agile methodology during the project planning and implementation.  Because our initial project planning was thorough and comprehensive, we did not have to adjust much. Additionally, without customer feedback, our requirements stayed static and therefore our project planning and estimates remain unchanged.  The only changes we made were adding and fixing keywords to the sequence diagrams to improve clarity.  All in all, our attention to detail and research made our work in-depth and exhaustive, and our demo and presentation speaks for the market viability of our product. |

**7. [5 POINTS]** References: Please include properly cited references in IEEE paper referencing format. Please review the IEEE referencing format document at the URL: <https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf>). It means that your references should be numbered, and these numbers properly cited in your project report.

|  |
| --- |
| [1] W. Vige, “Story points: Estimate user stories in agile [2023],” Asana, https://asana.com/resources/story-points (accessed Nov. 17, 2023).  [2] D. Radigan, “What are story points in agile and how do you estimate them?,” Atlassian, https://www.atlassian.com/agile/project-management/estimation (accessed Nov. 17, 2023).  [3] M. Rehkopf, “User stories: Examples and template,” Atlassian, https://www.atlassian.com/agile/project-management/user-stories (accessed Nov. 17, 2023).  [4] A. of us at monday.com, “Scrum: The definitive guide,” monday.com Blog, https://monday.com/blog/project-management/scrum/ (accessed Nov. 17, 2023).  [5] Webgraphviz, http://www.webgraphviz.com/ (accessed Nov. 17, 2023).  [6] “Monday,” monday.com, https://monday.com/ (accessed Nov. 17, 2023).  [7] Microsoft, “Pricing-Microsoft Defender: Microsoft Azure,” Pricing-Microsoft Defender | Microsoft Azure, https://azure.microsoft.com/en-us/pricing/details/defender-for-cloud/ [Accessed Nov. 16, 2023].  [8] Microsoft, “Pricing - Azure SQL Database Single Database: Microsoft Azure,” Pricing - Azure SQL Database Single Database | Microsoft Azure, https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/single/#pricing [Accessed Nov. 16, 2023].  [9] Microsoft, “Pricing - Cloud Services: Microsoft Azure,” Pricing - Cloud Services | Microsoft Azure, https://azure.microsoft.com/en-us/pricing/details/cloud-services/ [Accessed Nov. 16, 2023].  [10] Microsoft, “Visual Studio Professional - monthly subscription - Visual Studio Marketplace,” *marketplace.visualstudio.com*. https://marketplace.visualstudio.com/items?itemName=ms.vs-professional-monthly#pricing (accessed Nov. 16, 2023).  [11] J. Simonson and K. Main, “Stripe Pricing And Fees (2023 Guide) – Forbes Advisor,” *www.forbes.com*, 2023. https://www.forbes.com/advisor/business/services/stripe-pricing-fees/ (accessed Nov. 16, 2023).  [12] Microsoft, “Pricing Calculator | Microsoft Azure,” *Microsoft.com*, 2023. https://azure.microsoft.com/en-us/pricing/calculator/ (accessed Nov. 16, 2023).  [13] Swing2App, “How much does it cost to publish an app on the app store? | Blog,” *Swing2App*, Aug. 18, 2022. https://www.swing2app.com/blog/how-much-does-it-cost-to-publish-an-app-on-the-app-store/ (accessed Nov. 16, 2023).  [14] “Pricing calculator: Microsoft Azure,” Pricing Calculator | Microsoft Azure, https://azure.microsoft.com/en-us/pricing/calculator/?service=playwright-testing [Accessed Nov. 15, 2023].  [15] Chcomley, “Get started with Azure DevOps,” Learning path - Training | Microsoft Learn, https://learn.microsoft.com/en-us/training/paths/evolve-your-devops-practices/ [Accessed Nov. 16, 2023].  [16] J. Charlesworth, “Vite: The Complete Developer’s Guide,” Vite: The Complete Developer’s Guide | Udemy, https://www.udemy.com/course/vite-developers-guide/ [Accessed Nov. 16, 2023].  [17] B. Alam, “Web Automation and Testing using Playwright,” Web Automation and Testing using Playwright | Udemy, https://www.udemy.com/course/playwright/ [Accessed Nov. 16, 2023].  [18] “Scalable learning for organizations of every size,” Plans - Udemy Business, https://business.udemy.com/plans/ [Accessed Nov. 16, 2023].  [19] Vladimir, “Getting Started,” Getting Started | Guide | Vitest, https://vitest.dev/guide/ [Accessed Nov. 16, 2023].  [20] What Is Venmo? – Venmo, help.venmo.com/hc/en-us/articles/221011388-What-is-Venmo-. [Accessed 15 Nov. 2023].  [21] “What Is Venmo: How It Works.” Money, money.com/what-is-venmo-how-it-works/. [Accessed 15 Nov. 2023].  [22] “About.” Splitwise, www.splitwise.com/about. [Accessed 15 Nov. 2023].  [23] Reiff, Nathan. “How Splitwise Makes Money.” Investopedia, Investopedia, www.investopedia.com/articles/company-insights/090816/how-splitwise-works-and-makes-money.asp. [Accessed 15 Nov. 2023]. |

**8. [10 POINTS]** Presentation slides. No min/max number of slides enforced. Please make sure that you can complete presentation within 20 (twenty) minutes. Following template could be a good start to prepare your presentations. As each project topic is different, a variety in presentation style is expected and welcome.

- Title of your project together with participants

- Objective of the project designed

- Cost estimation

- Project timeline (timeline of the project designed, NOT the time you’ve spent on it)

- Functional and non-functional requirements. If too long, select representative items.

- Use case diagram

- Sequence diagram for a selected representative operation of the project.

- Class diagram

- Architectural design

- Model-View-Controller (MVC) pattern (similar to Figure 6.6)

- Layered architecture pattern (similar to Figure 6.9)

- Repository architecture pattern (similar to Figure 6.11)

- Client-server architecture pattern (similar to Figure 6.13)

- Pipe and filter architecture pattern (similar to Figure 6.15)

- Preferably a demo of user interface design that shows screen to screen transitions though no full functionality is required.

- OPTIONAL: IF implemented the project, a demo of your implementation.

**9. OPTIONAL PART [POSSIBLE EXTRA CREDIT UP TO 10 POINTS].** Your program code (if fully implemented the project, not required otherwise). Please note that implementation is not required for the final project. Groups are welcome to implement their work, if they choose to do so. [This part may qualify for extra credit, if you implement and submit the implementation code together with your project. The extra credit will be determined based on the quality of your implementation. Furthermore, any fully implemented project qualifies for scholar publication afterwards. This most probably will involve further commitment to work more an write a scholar paper to send to a Conference for publication.

|  |
| --- |
| We created a fully functional implementation of our project using Sveltekit, a full-stack web-app framework. We chose this framework because it has great performance on any platform and because it works nicely with other libraries. For example, we used Prisma to interface with a Postgres database with a custom schema. Prisma ensures that all data returned from the database has type safety, and it makes it easy to create RESTful endpoints. For our frontend, we used Bootstrap because it allowed us to efficiently create a usable UI.  The implementation also has working and secure authentication using salted and hashed passwords. JSON Web Tokens (JWTs) are used for session authentication.  The code for the project is included in the zip file for this assignment, but it is also available on GitHub [here](https://github.com/md-y/3354-splitsquad/tree/master/Implementation#splitsquad). **We also have a recorded demo** [**here**](https://www.youtube.com/watch?v=_NBJHMUpuRQ)**.**  To run the project, you must install Node.js, NPM, and PgAdmin 4. Information on how to do this is [here](https://github.com/md-y/3354-splitsquad/tree/master/Implementation#how-to-build). |

**10. [5 POINTS] GitHub requirement:**

Make sure at least one member of your group commits everything for project deliverable 2 to your GitHub repository, i.e.

- Your final project deliverable2 report

- Unit test code for a sample unit of your project

- Implementation code (if you have implemented your project)

- Presentation slides

Still, one member of your team should also submit the required project deliverable 2 materials to eLearning.