**CS3354 Software Engineering**

**Final Project Deliverable 1**

**Split Squad**

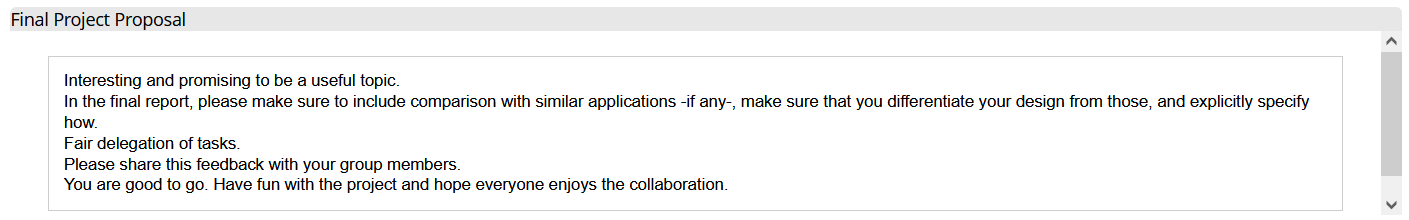
By: Tan Vo, Daniel Nguyen, Stephen George, Quan Pham,

Mir Ali, Samuel Preston, Amaan Babul, Shriniketh Mukundan

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

**1. [5 POINTS]** 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. |

**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

**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.

|  |  |  |
| --- | --- | --- |
| **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 |

**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). |

**5. [15 POINTS]** Software Requirements including

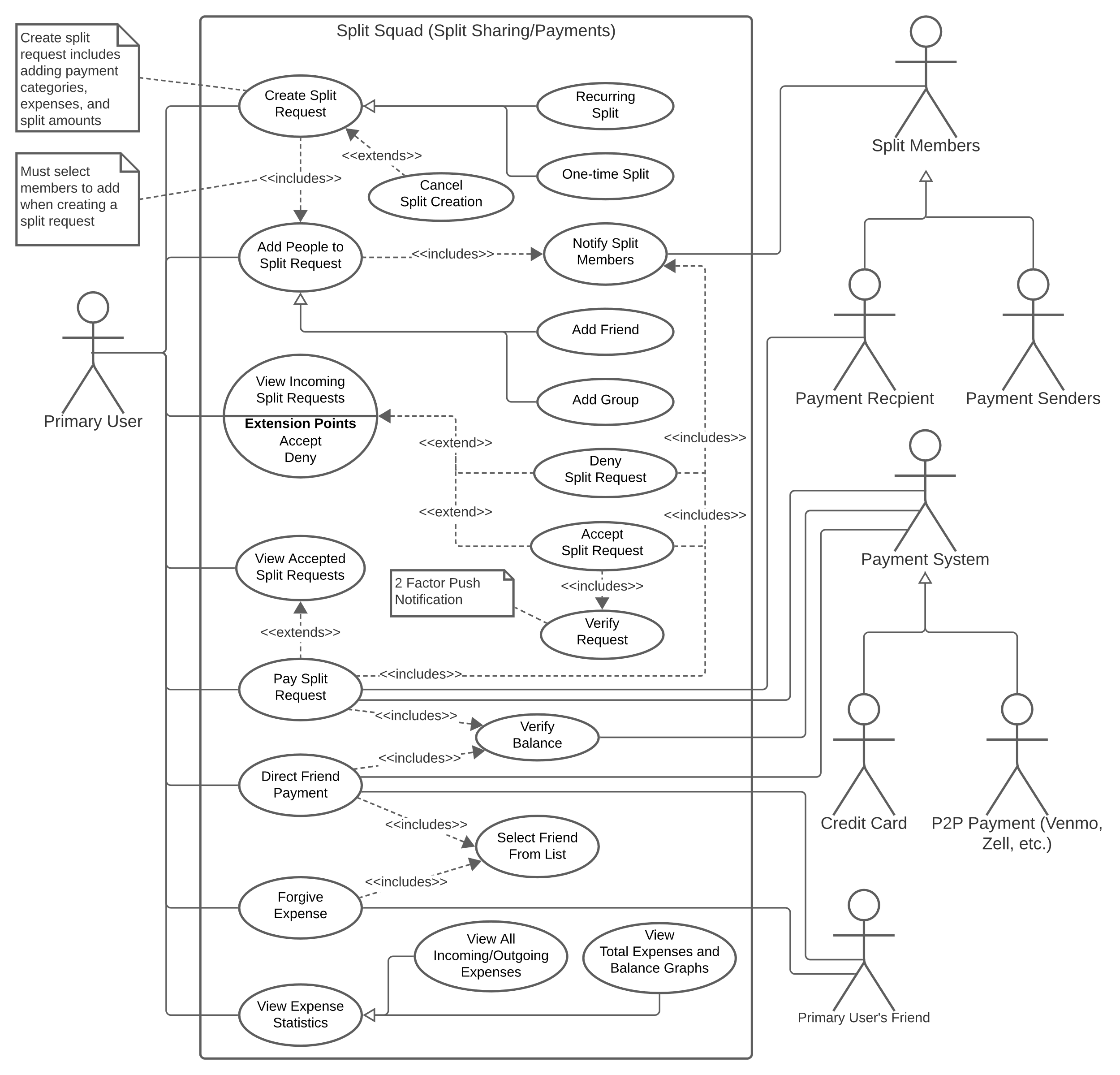
**5.a.) [5 POINTS]** Functional requirements. To simplify your design, please keep your functional requirements in the range of minimum 5 (five) to maximum 7 (seven). (Ch 4)

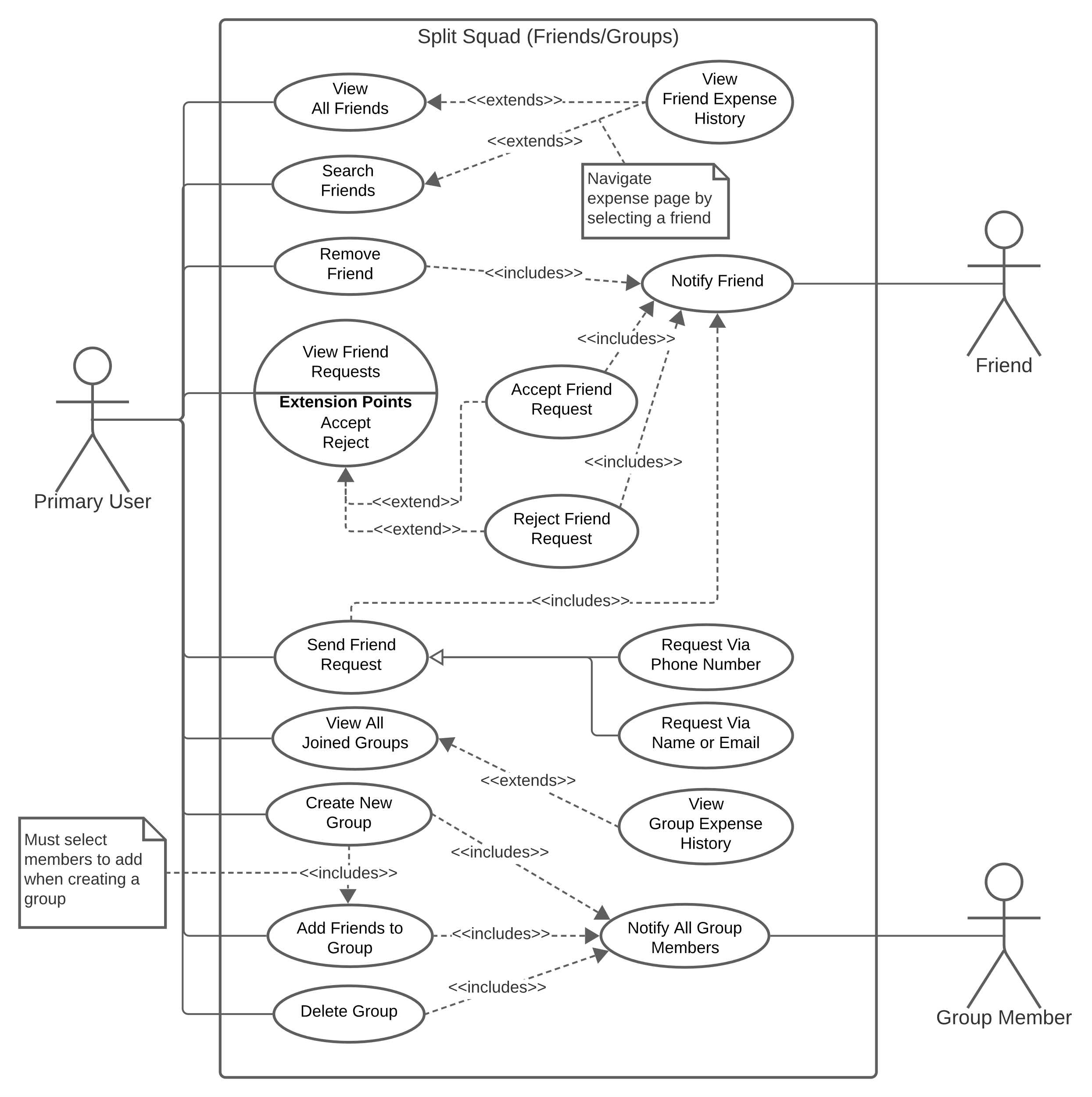
|  |
| --- |
| 1. The user shall be able to create a split request among one or more other users with fields to input total expense, to split total expense in the split request equally, by percentage, by custom amount, or by a mix of percentages and custom amount with options to add periodicity (weekly/monthly) 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. |

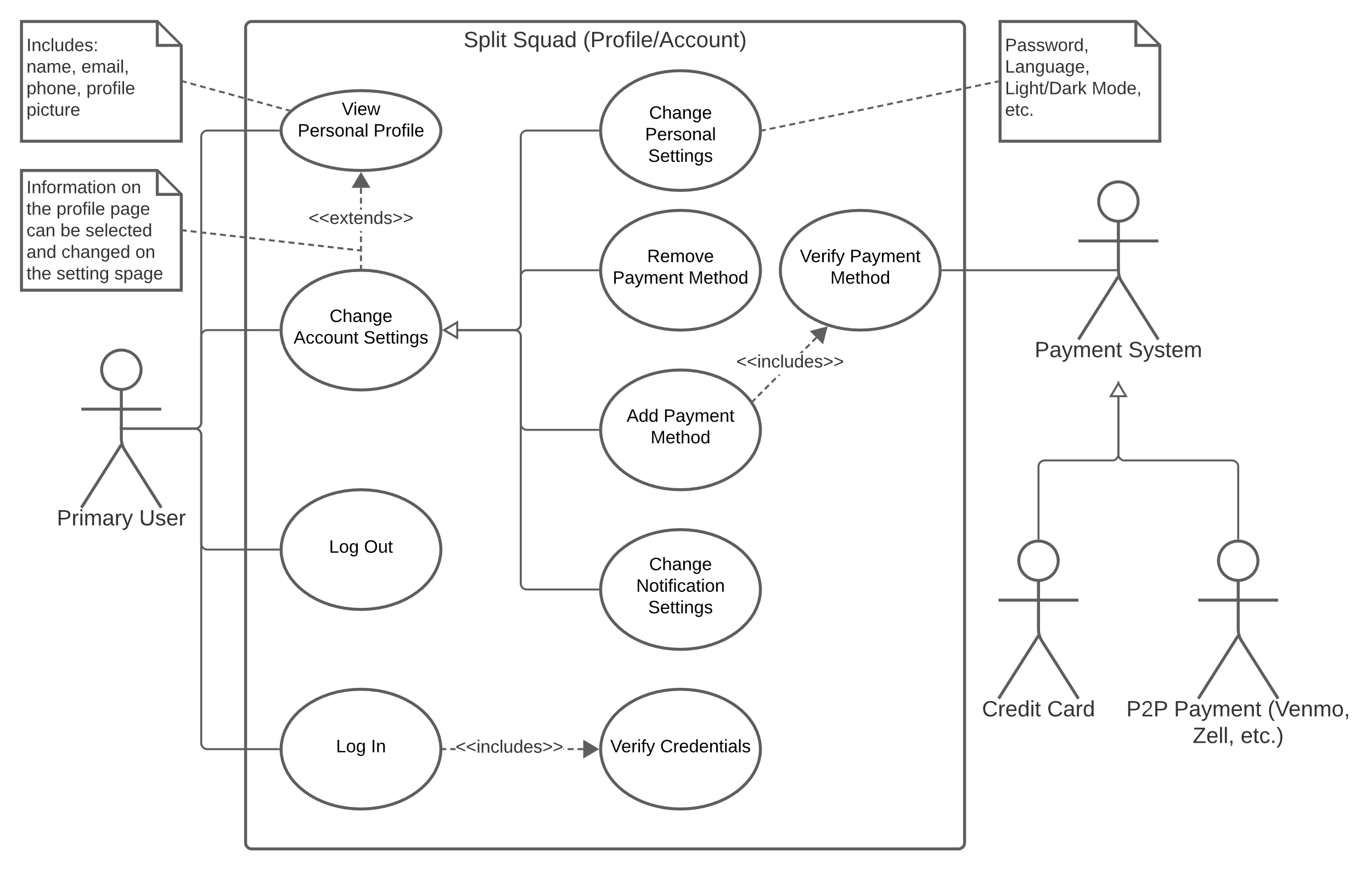
**5.b.) [10 POINTS]** Non-functional requirements (use all non-functional requirement types listed in Figure 4.3 - Ch 4. This means provide one non-functional requirement for each of the leaves of Figure 4.3. You can certainly make assumptions, even make up government/country-based rules, requirements 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. |

**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)

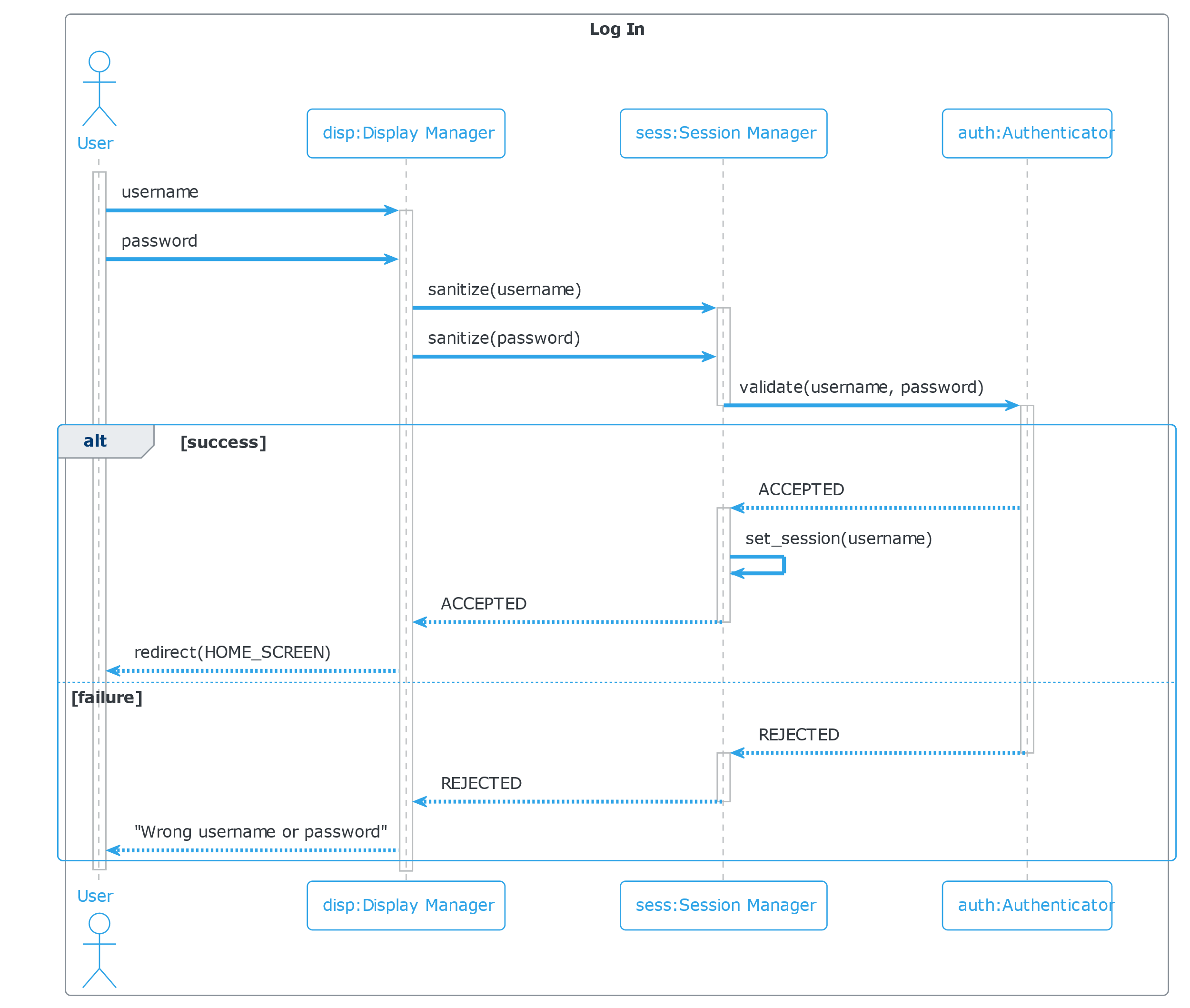


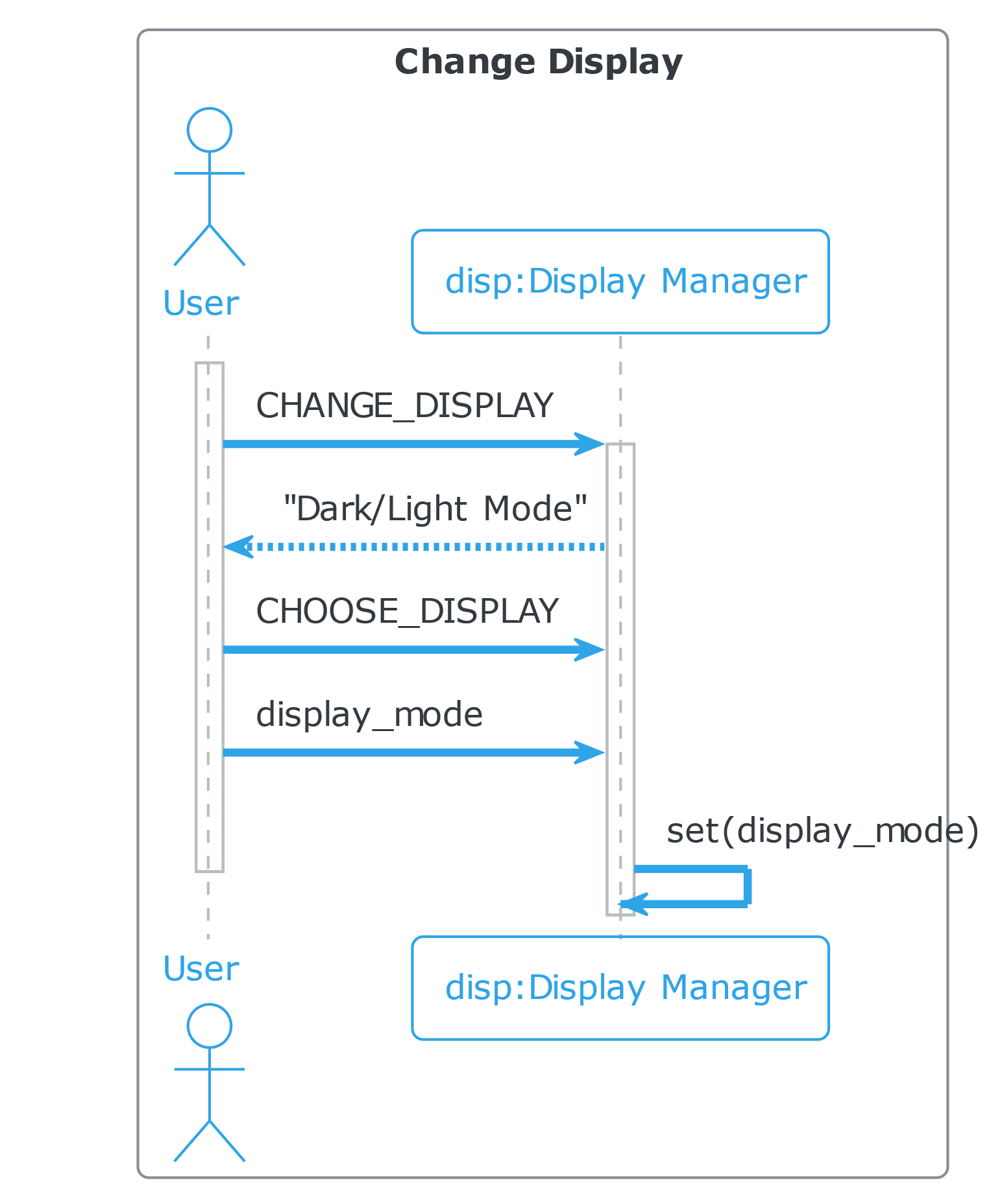
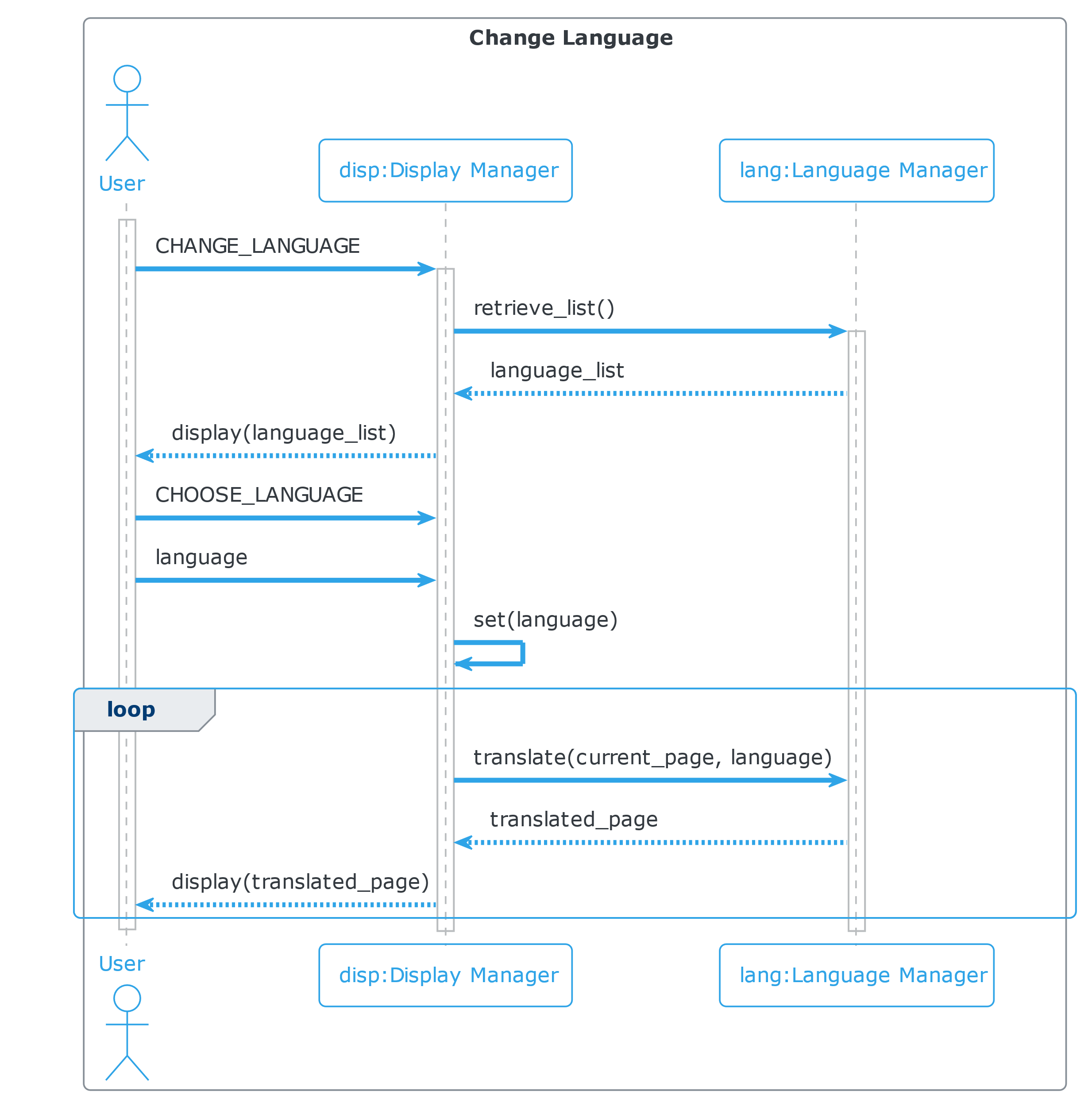
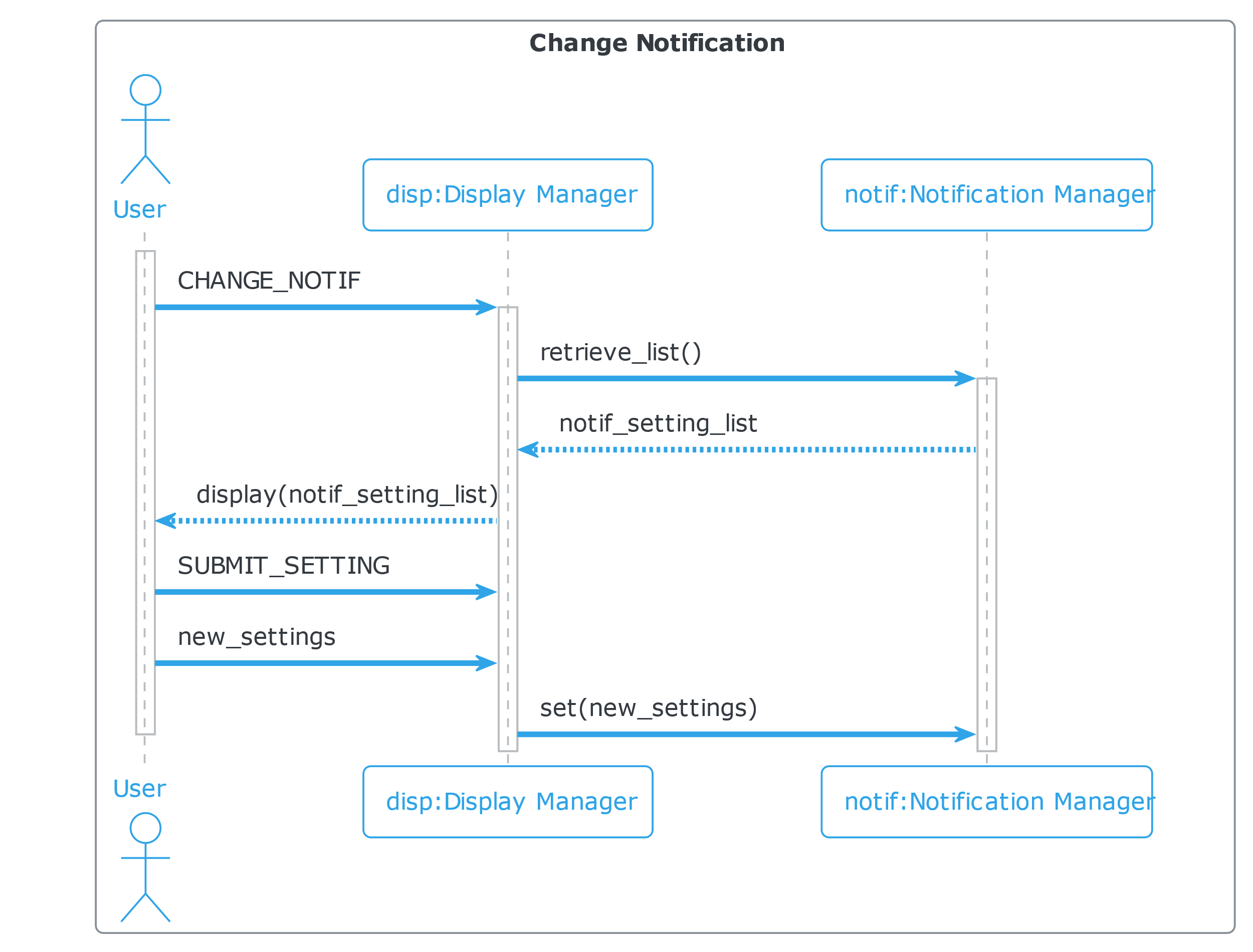
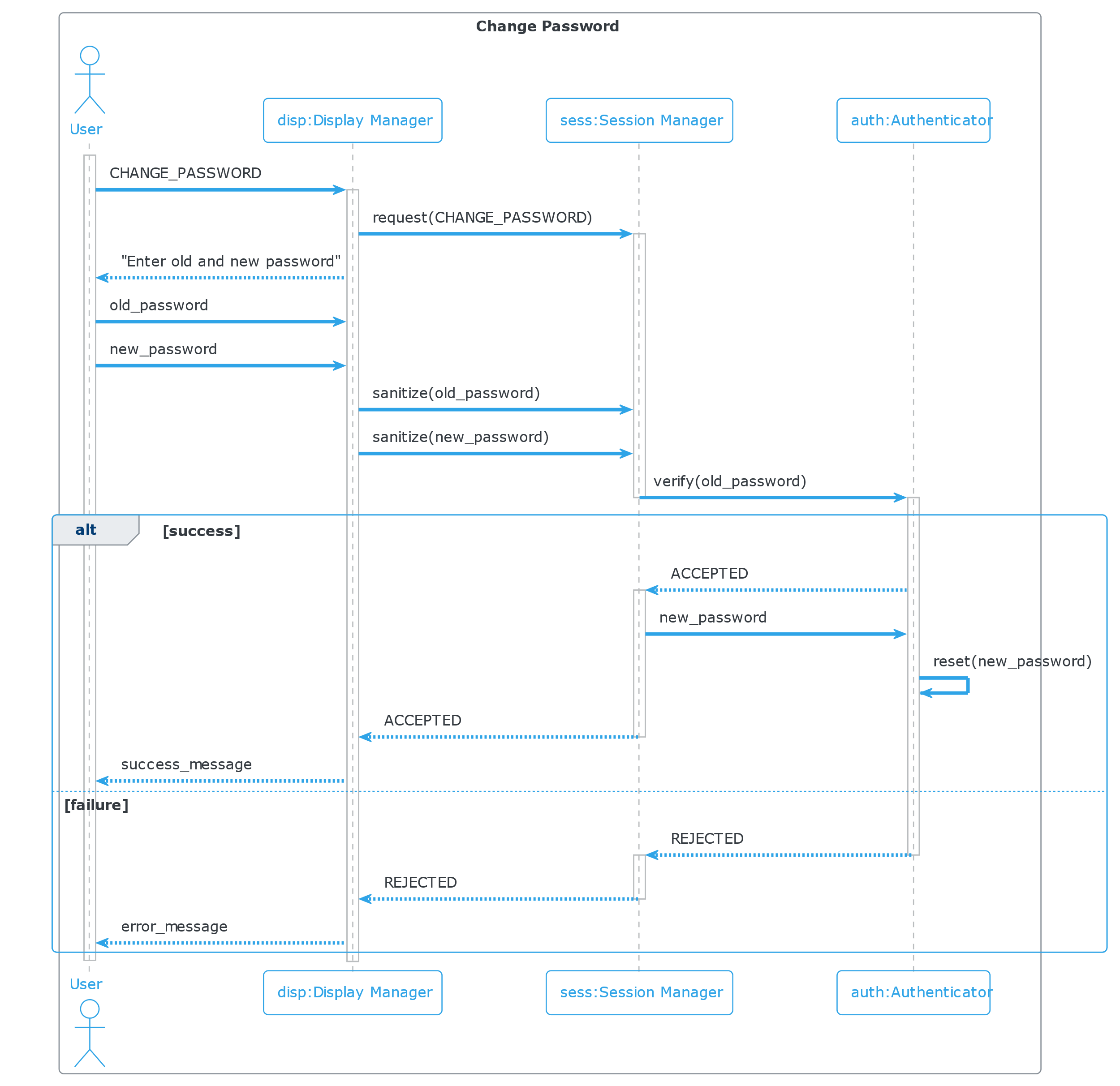
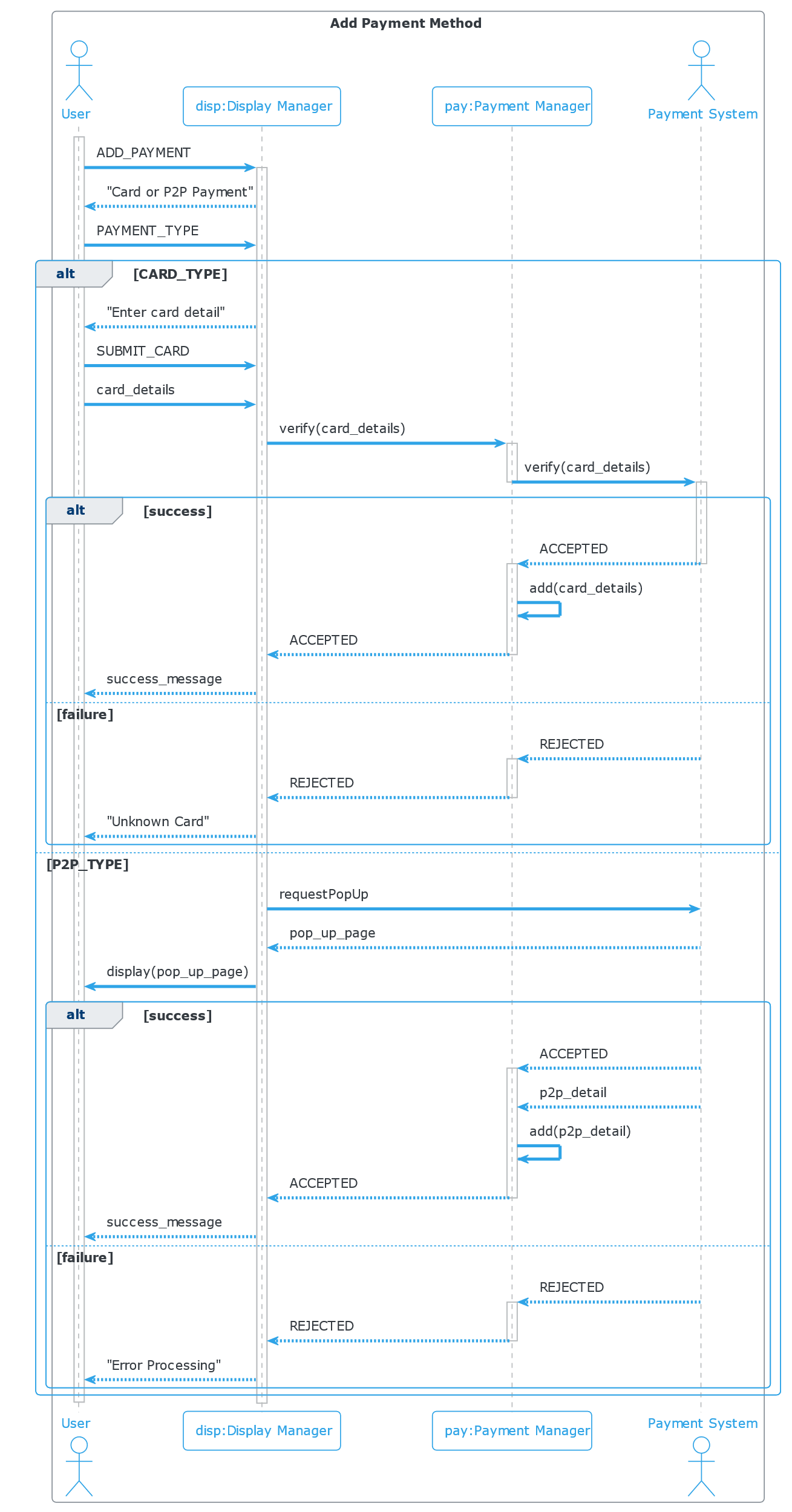
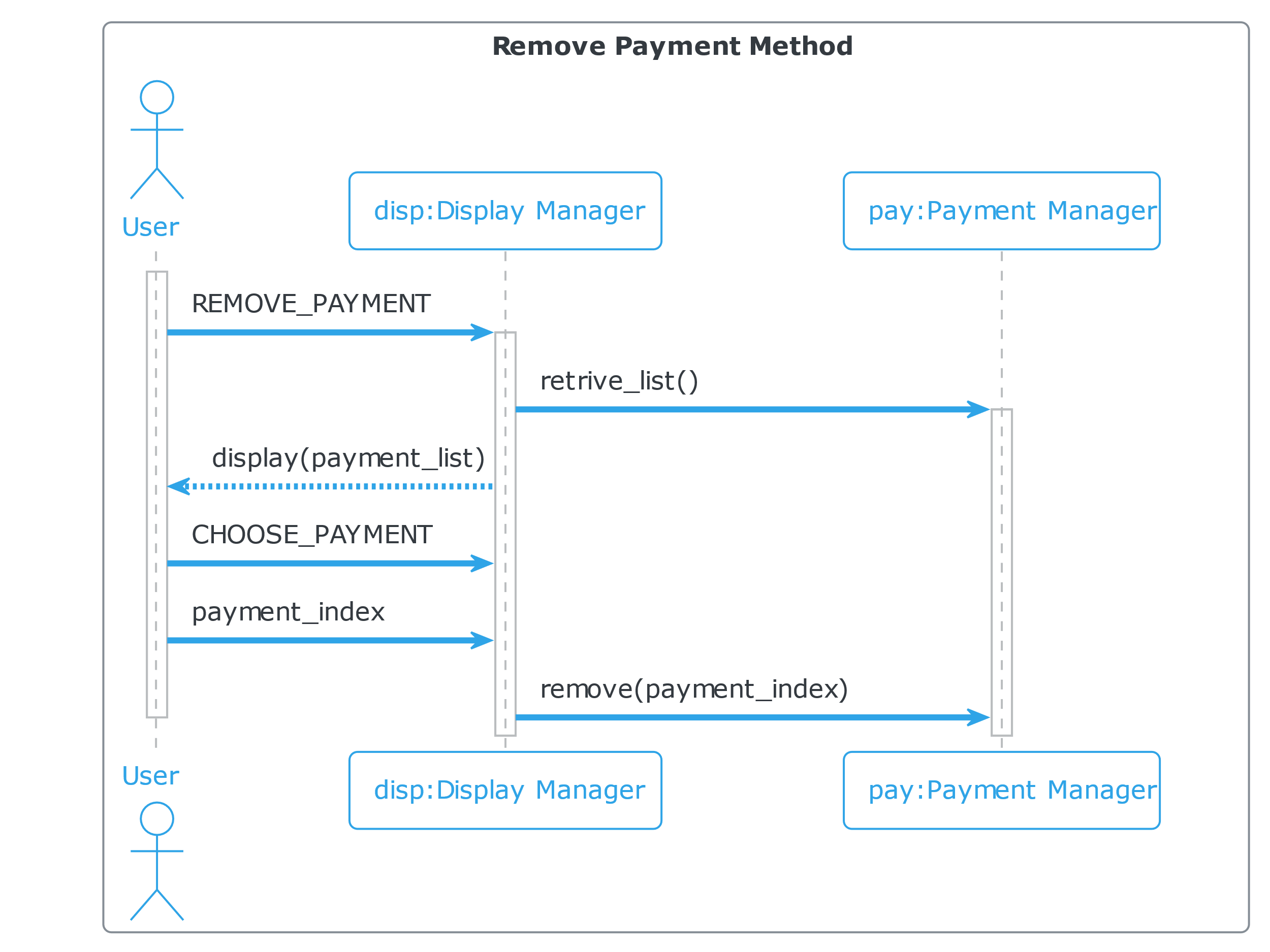
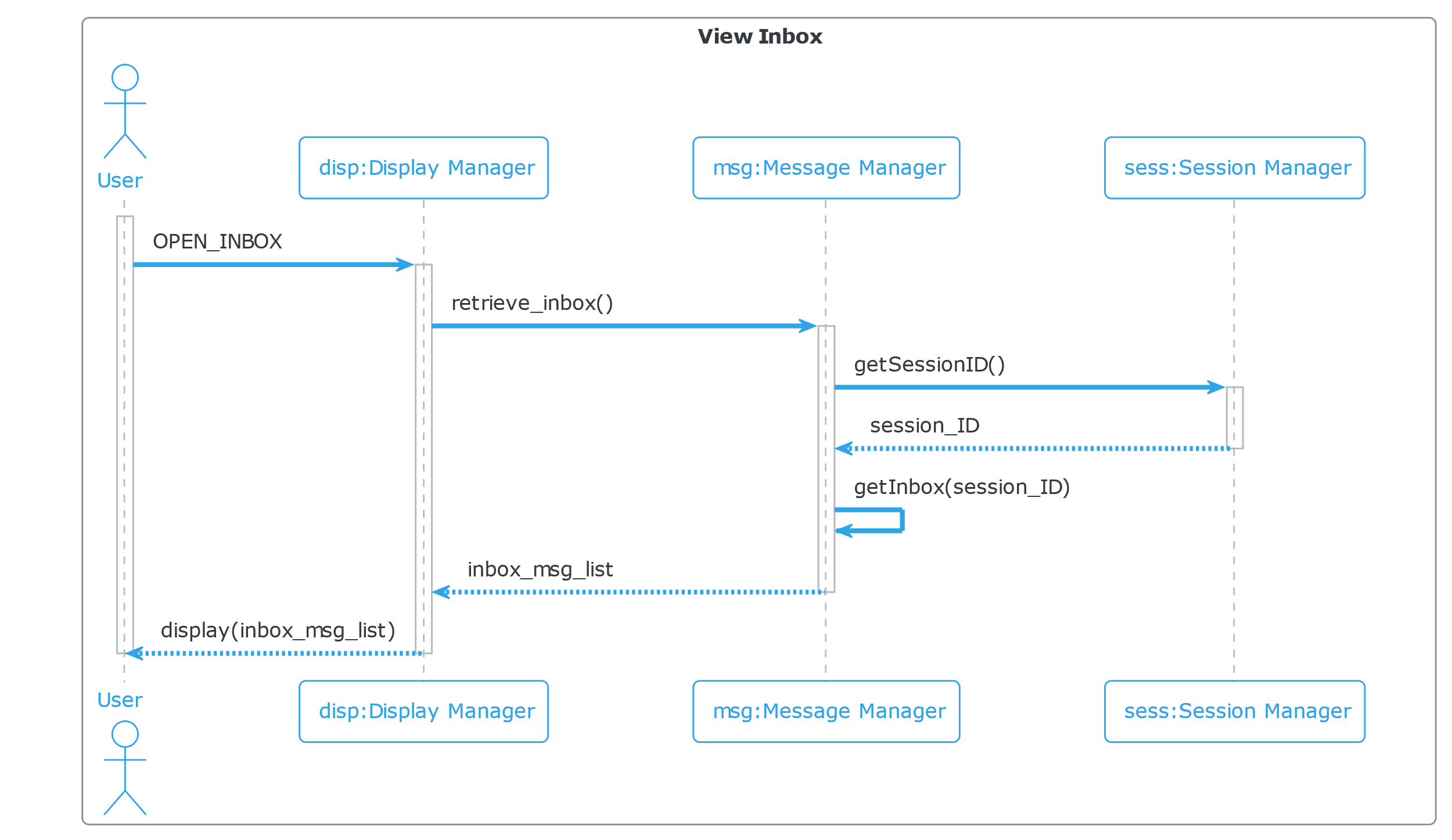


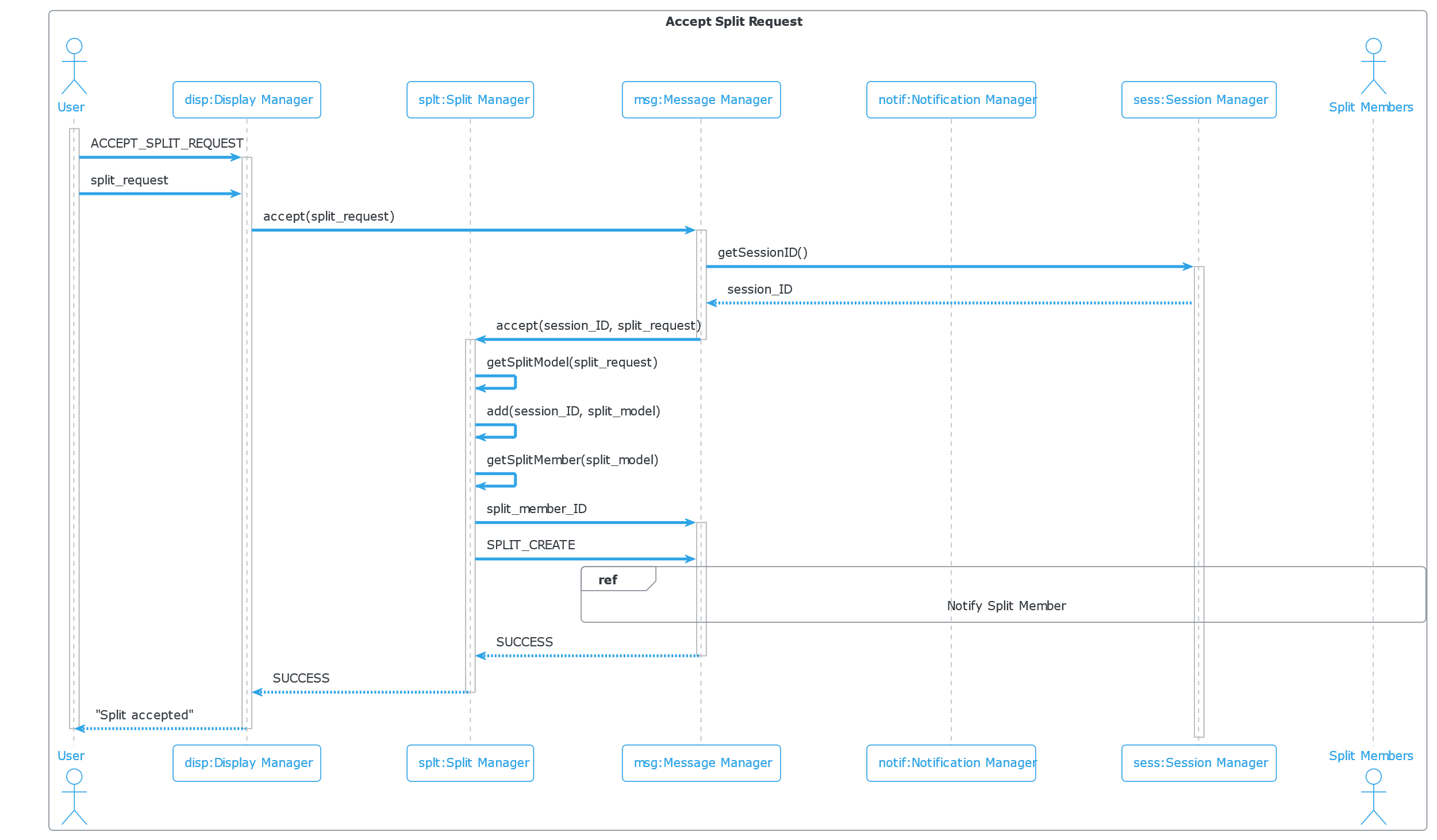
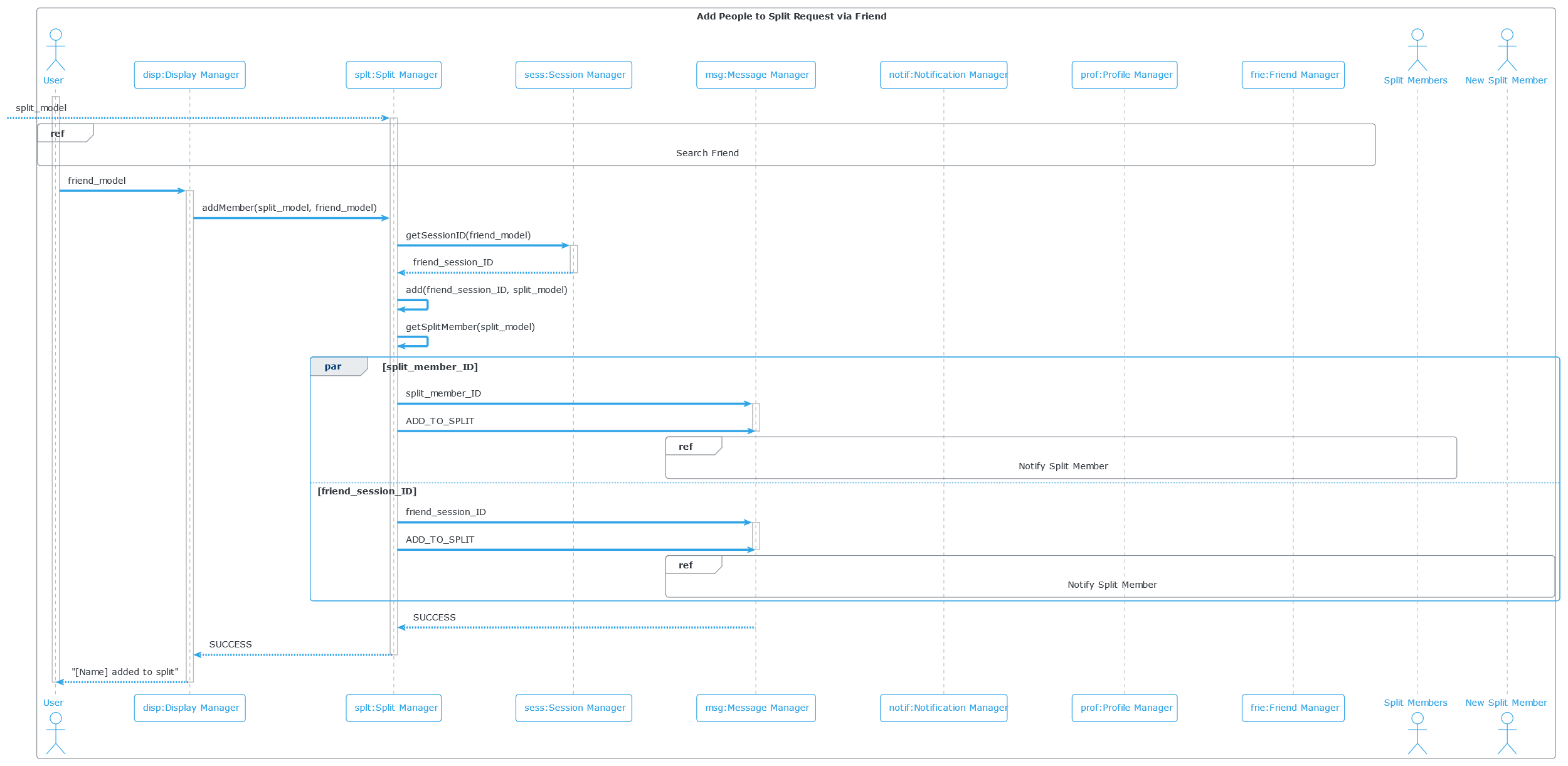
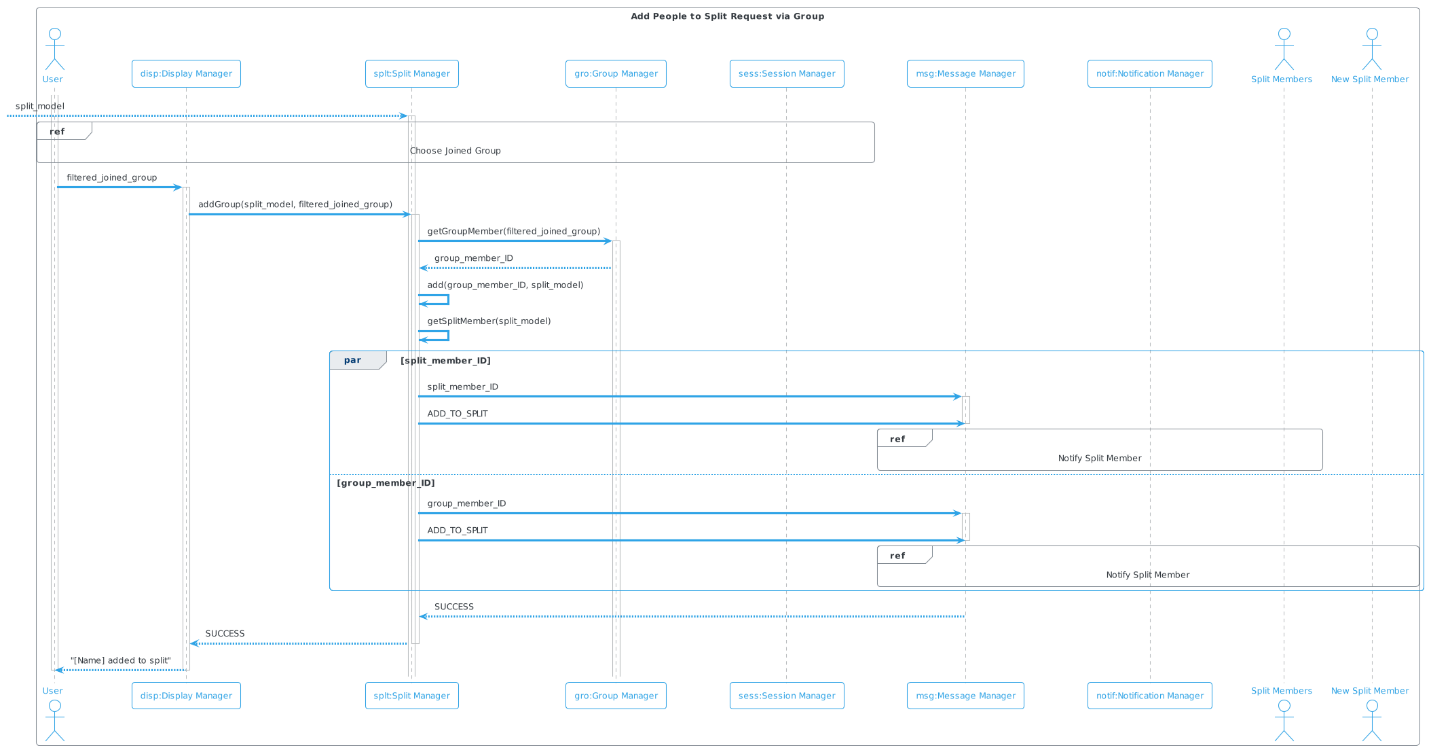
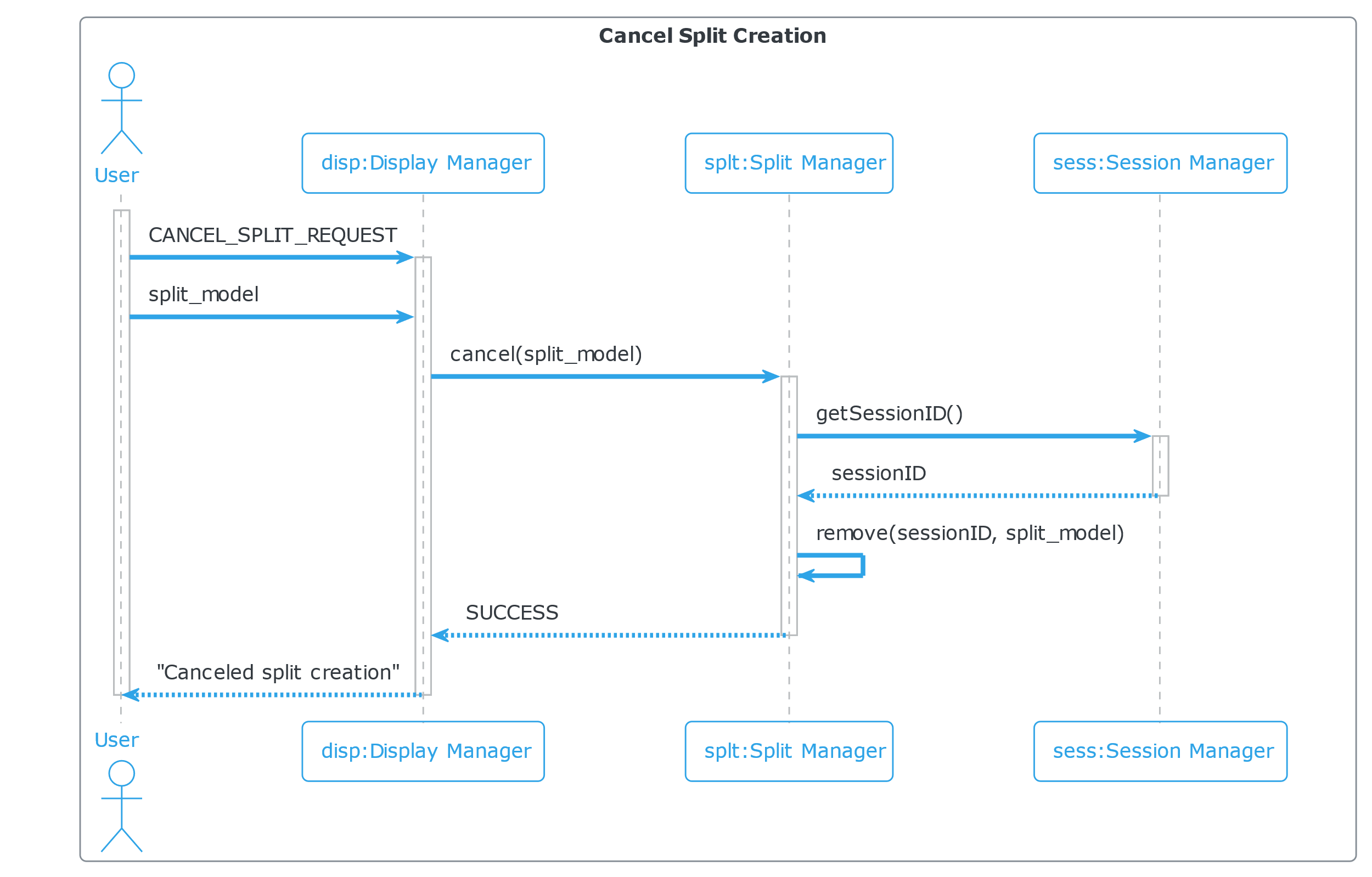
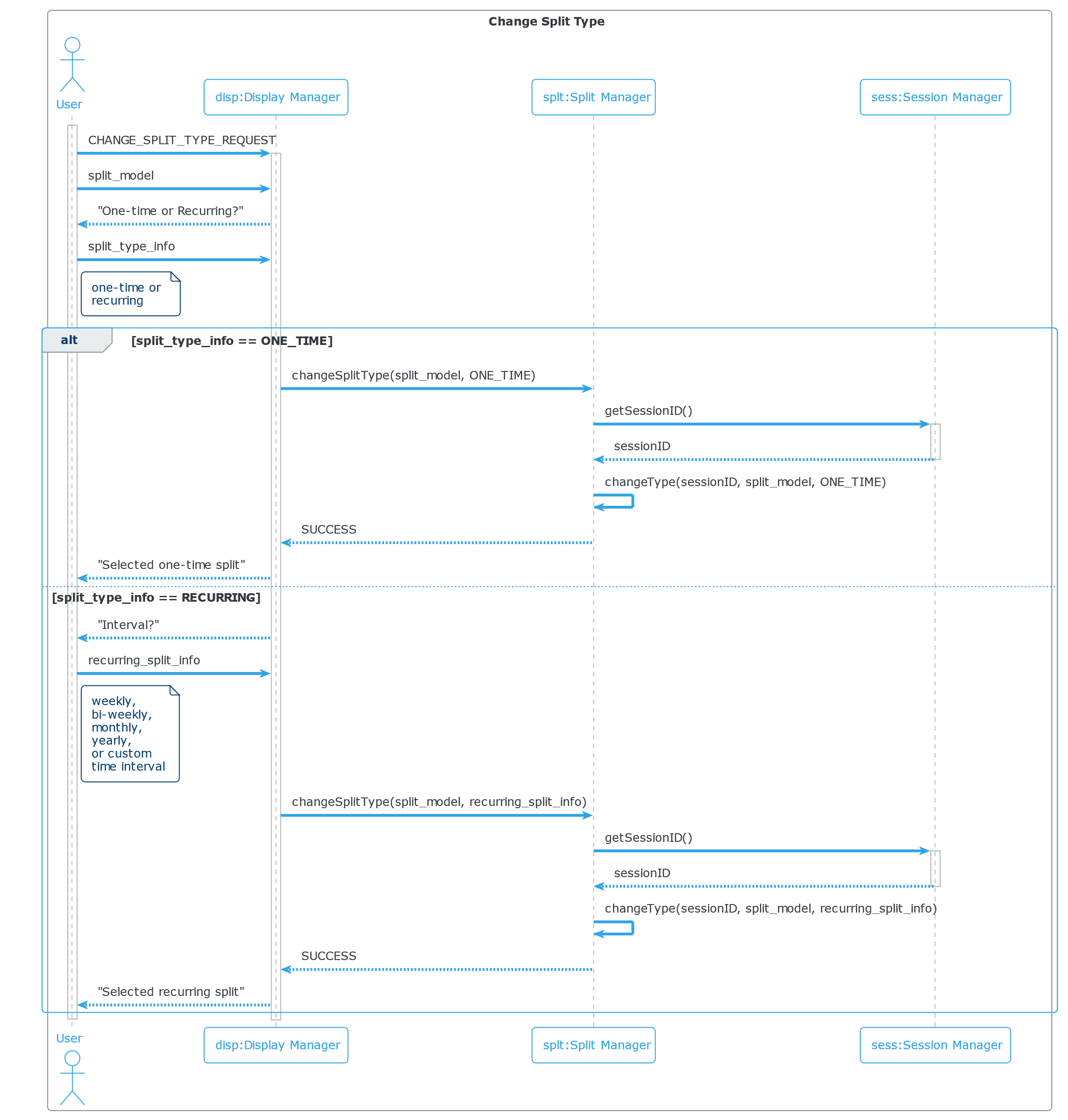
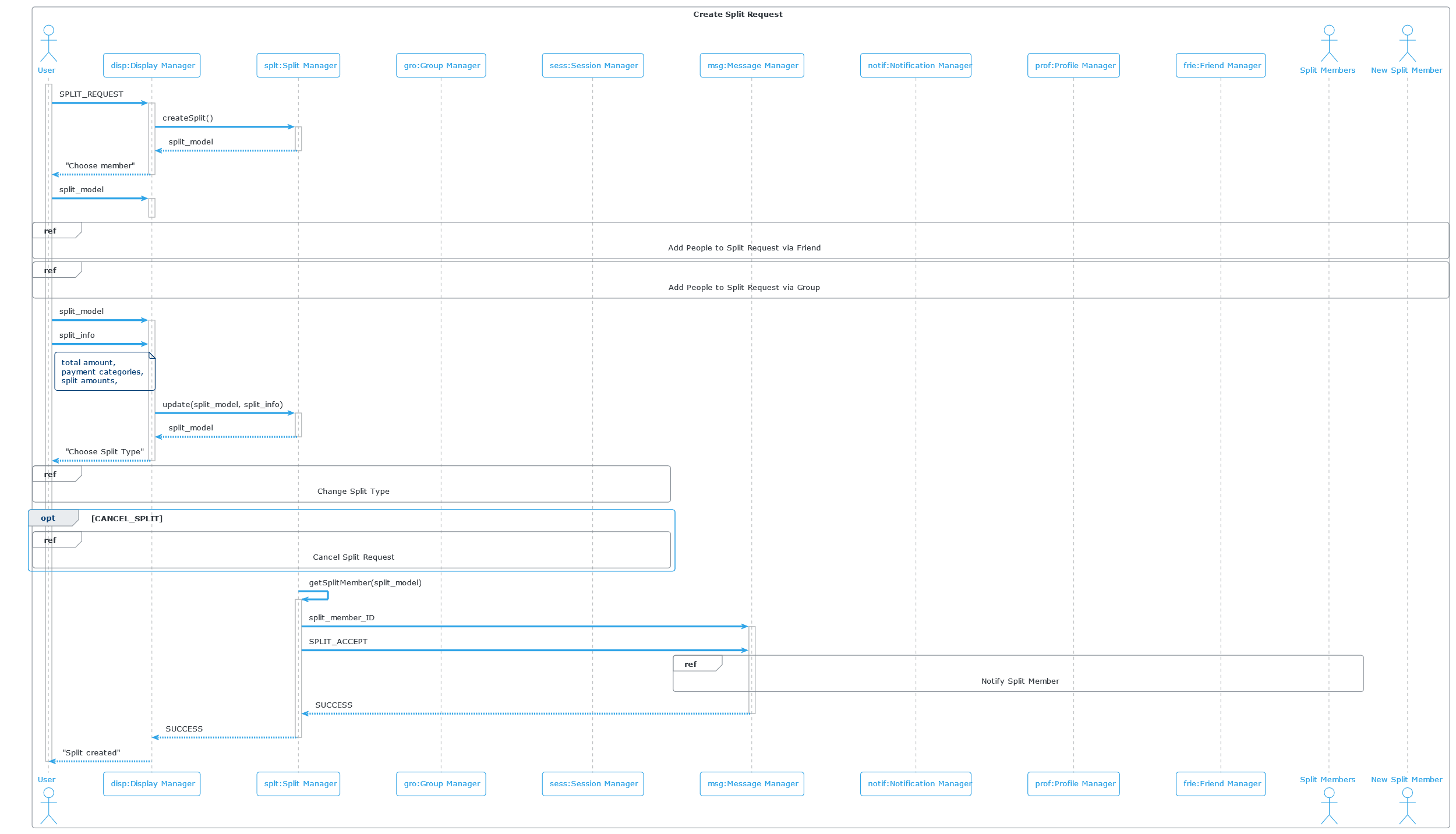
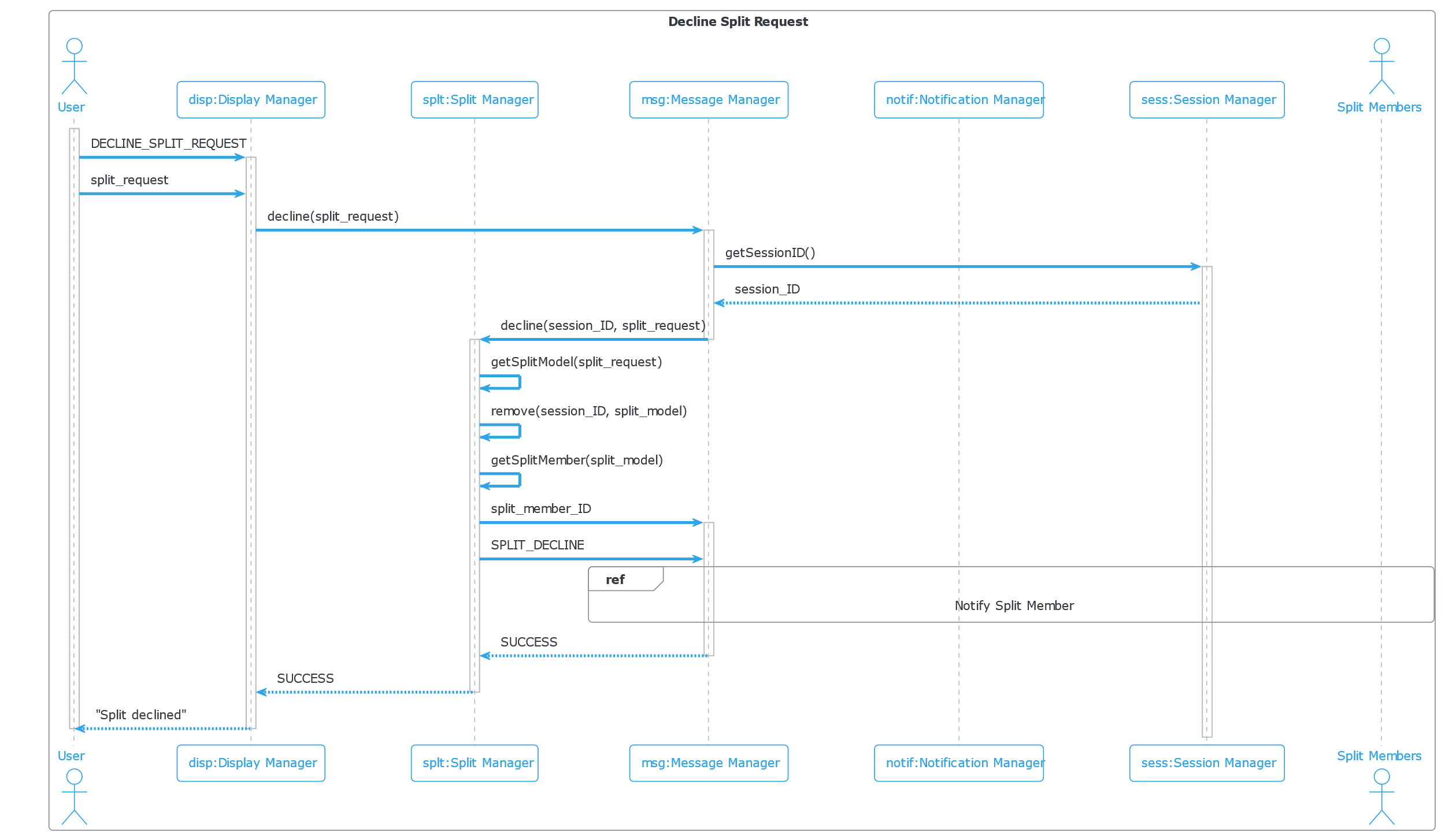
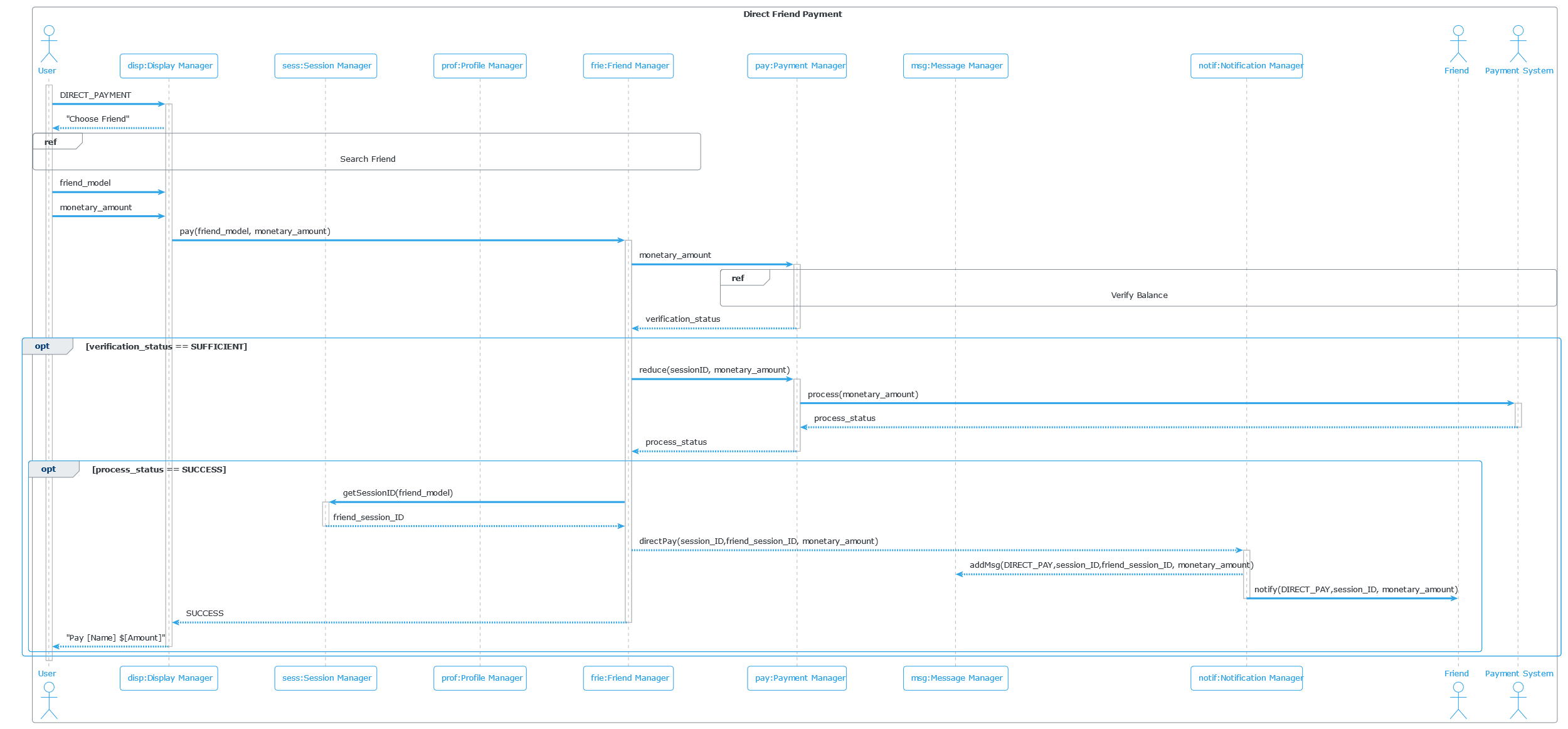
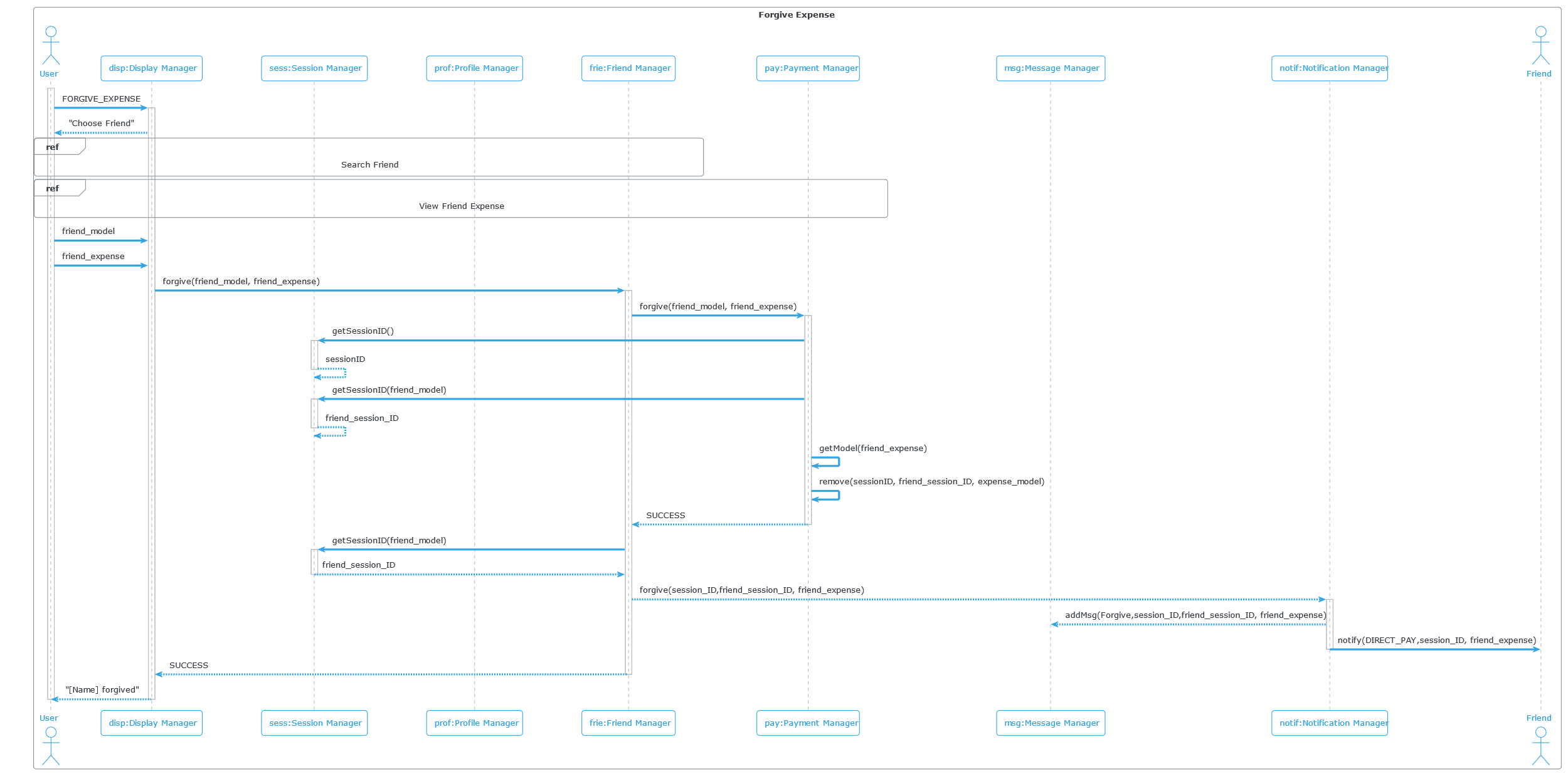
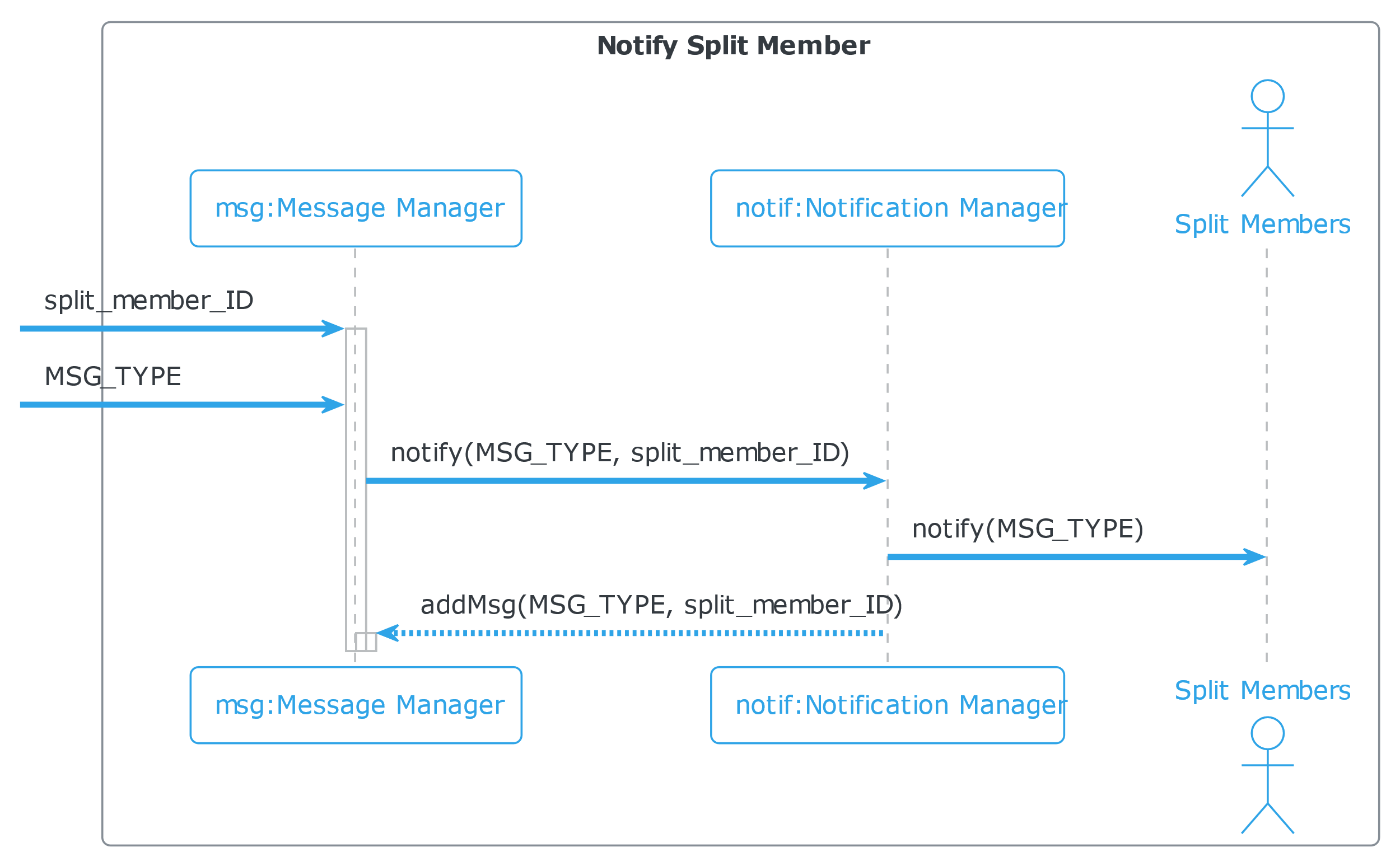
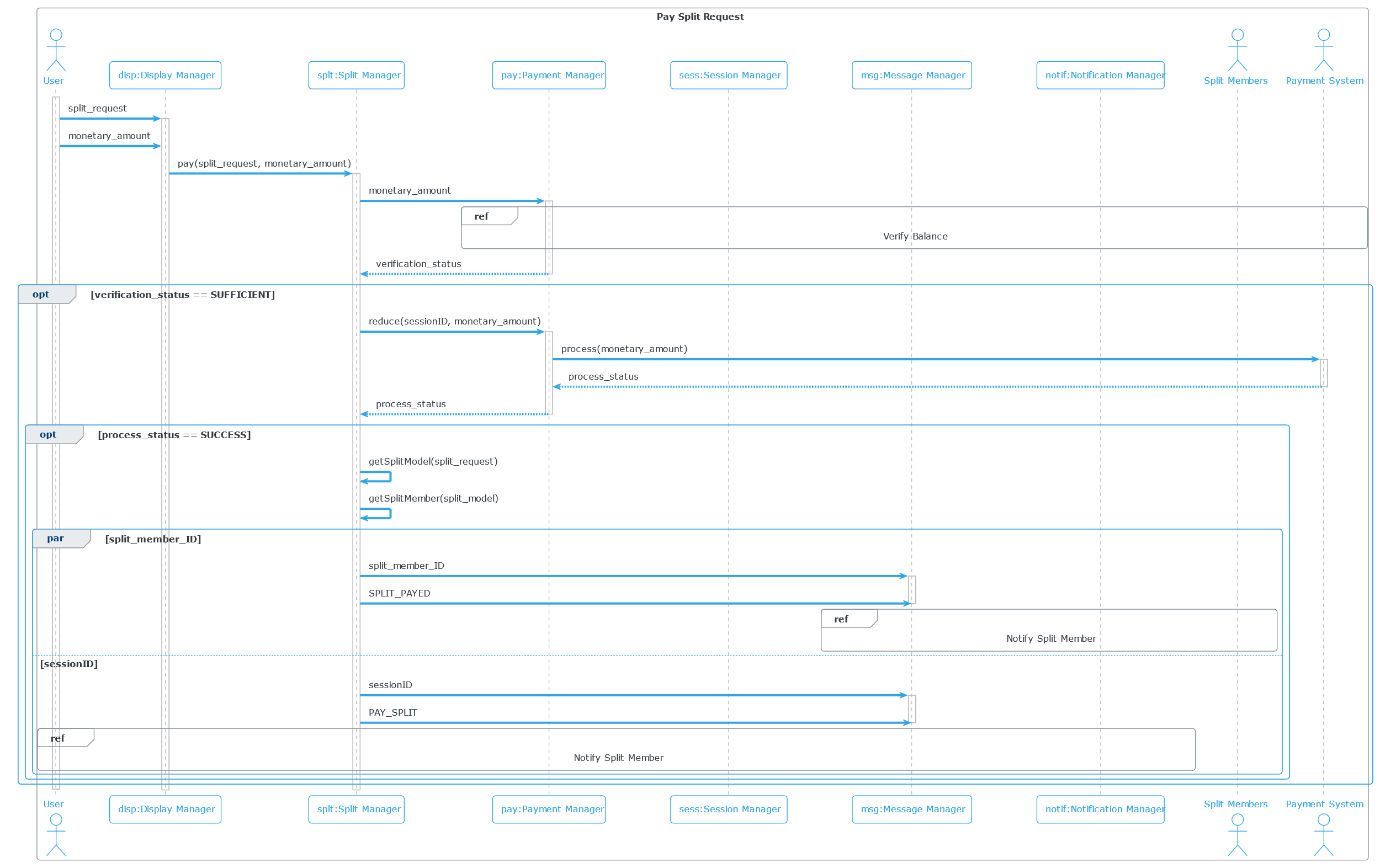
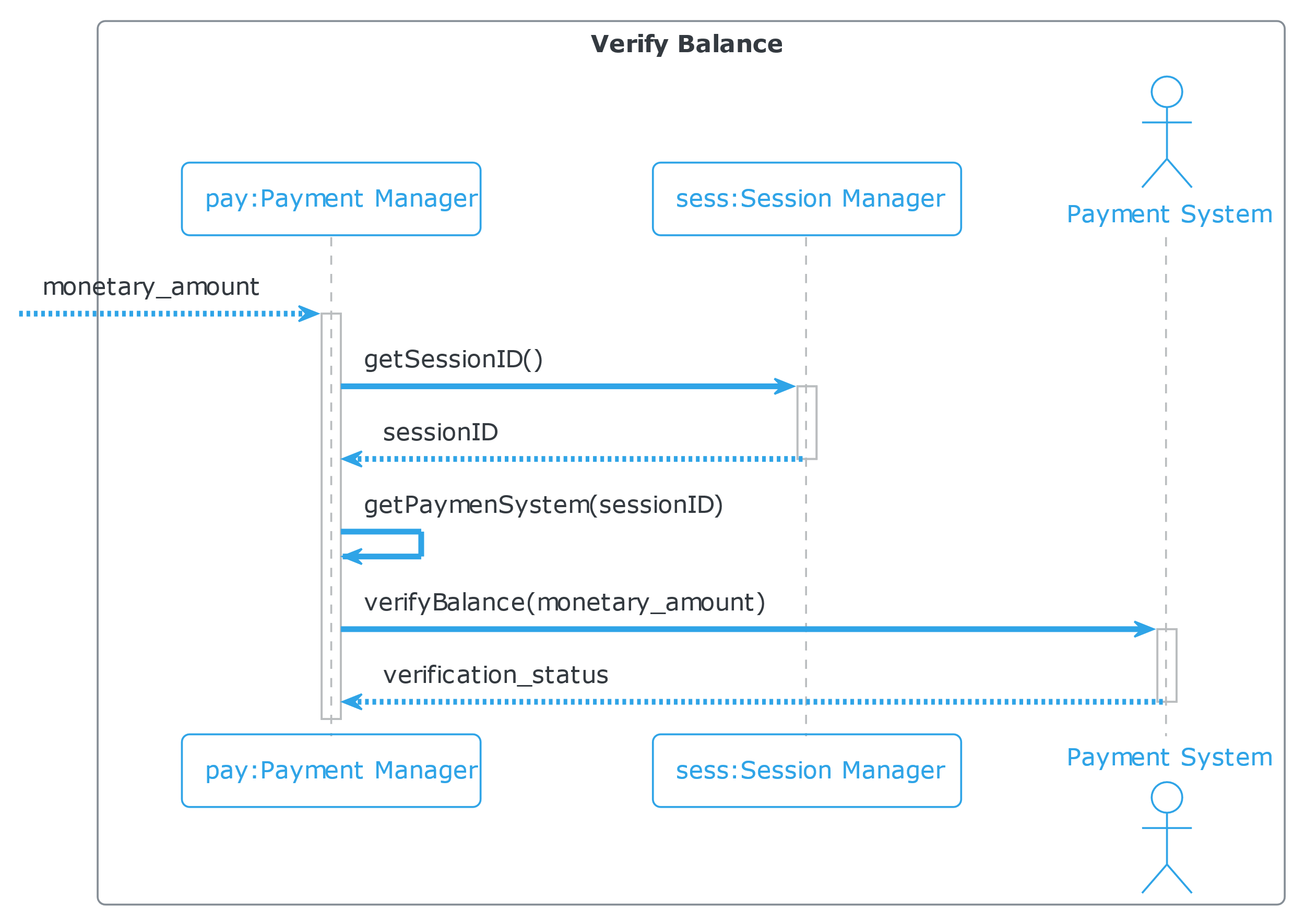
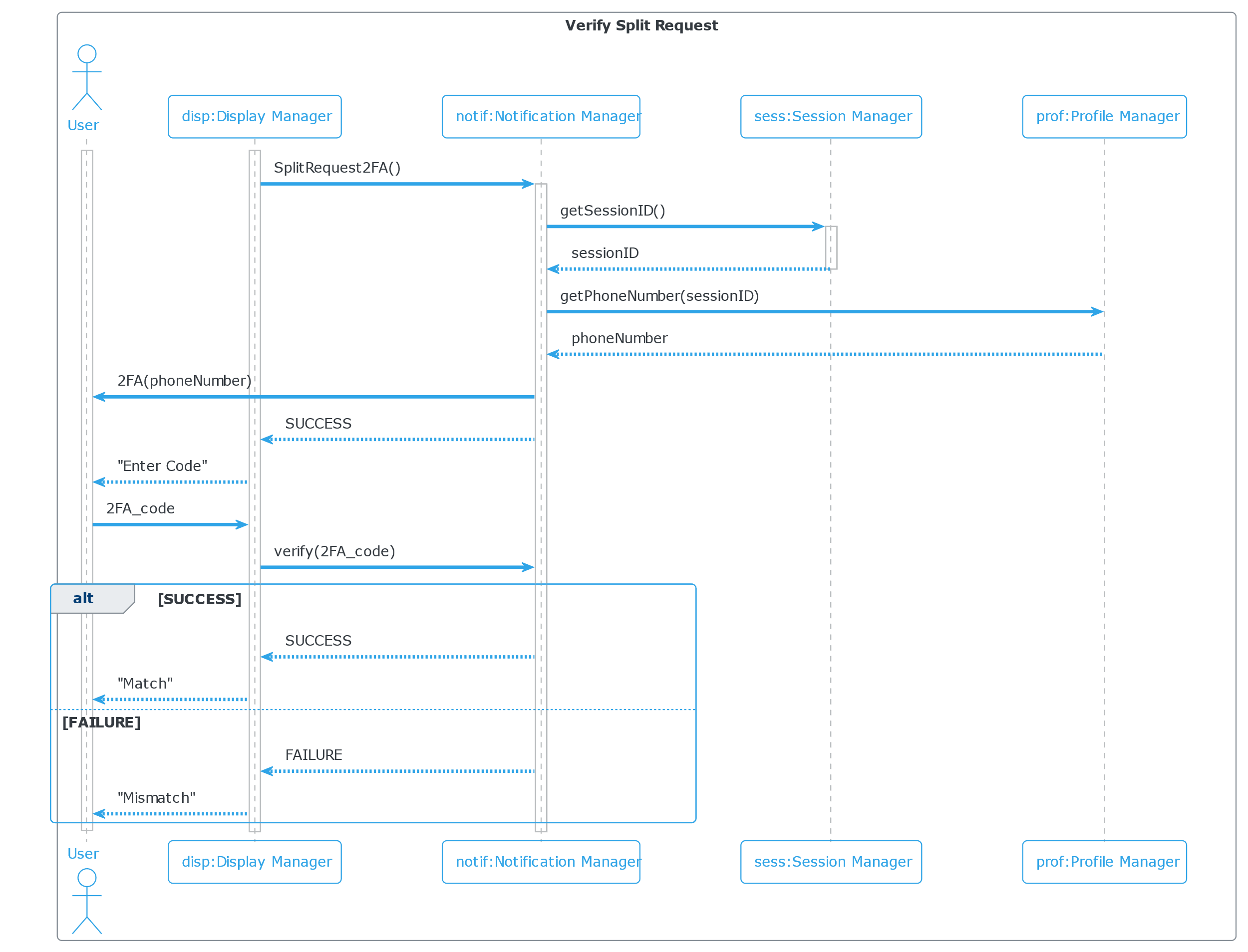
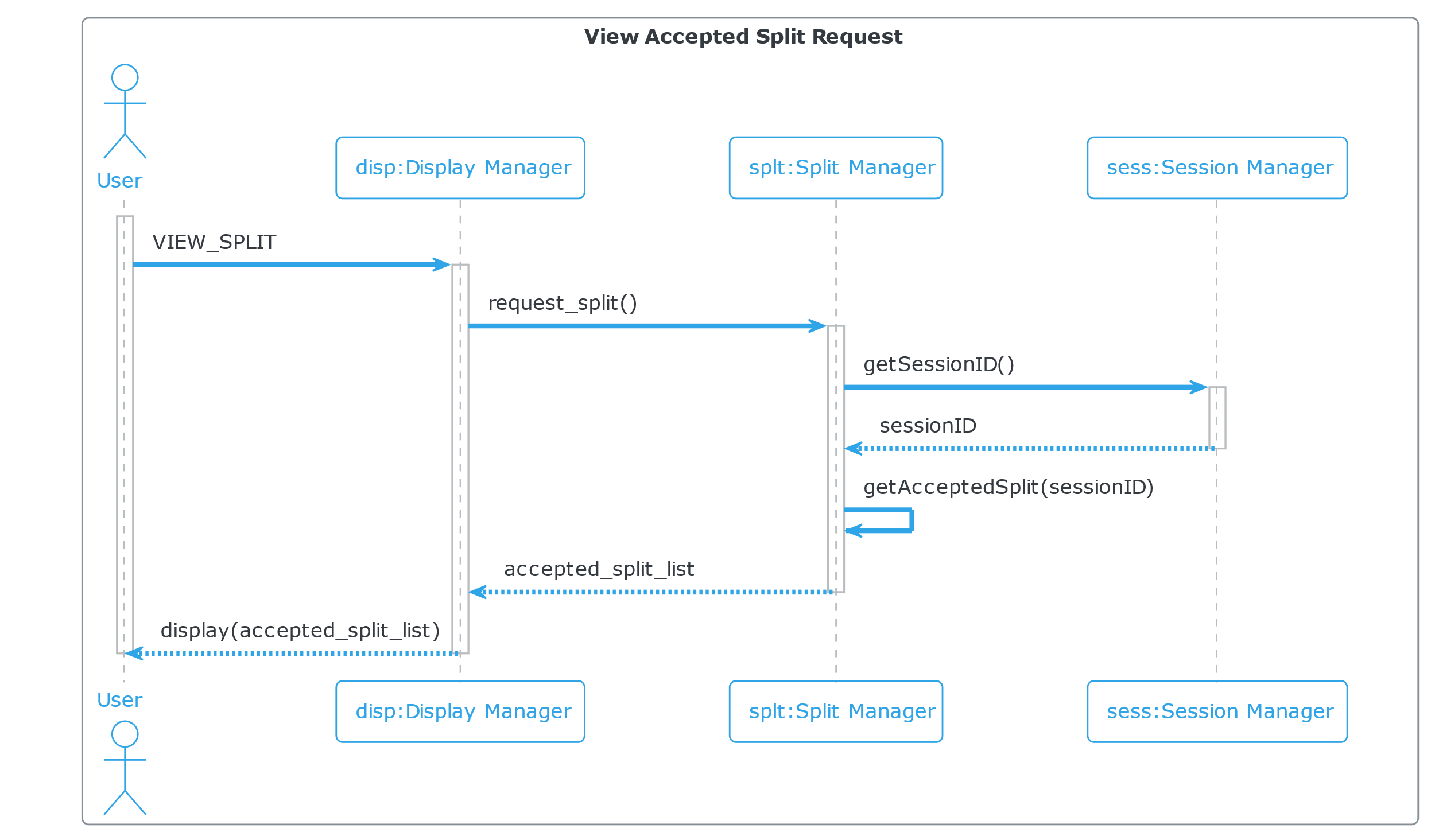
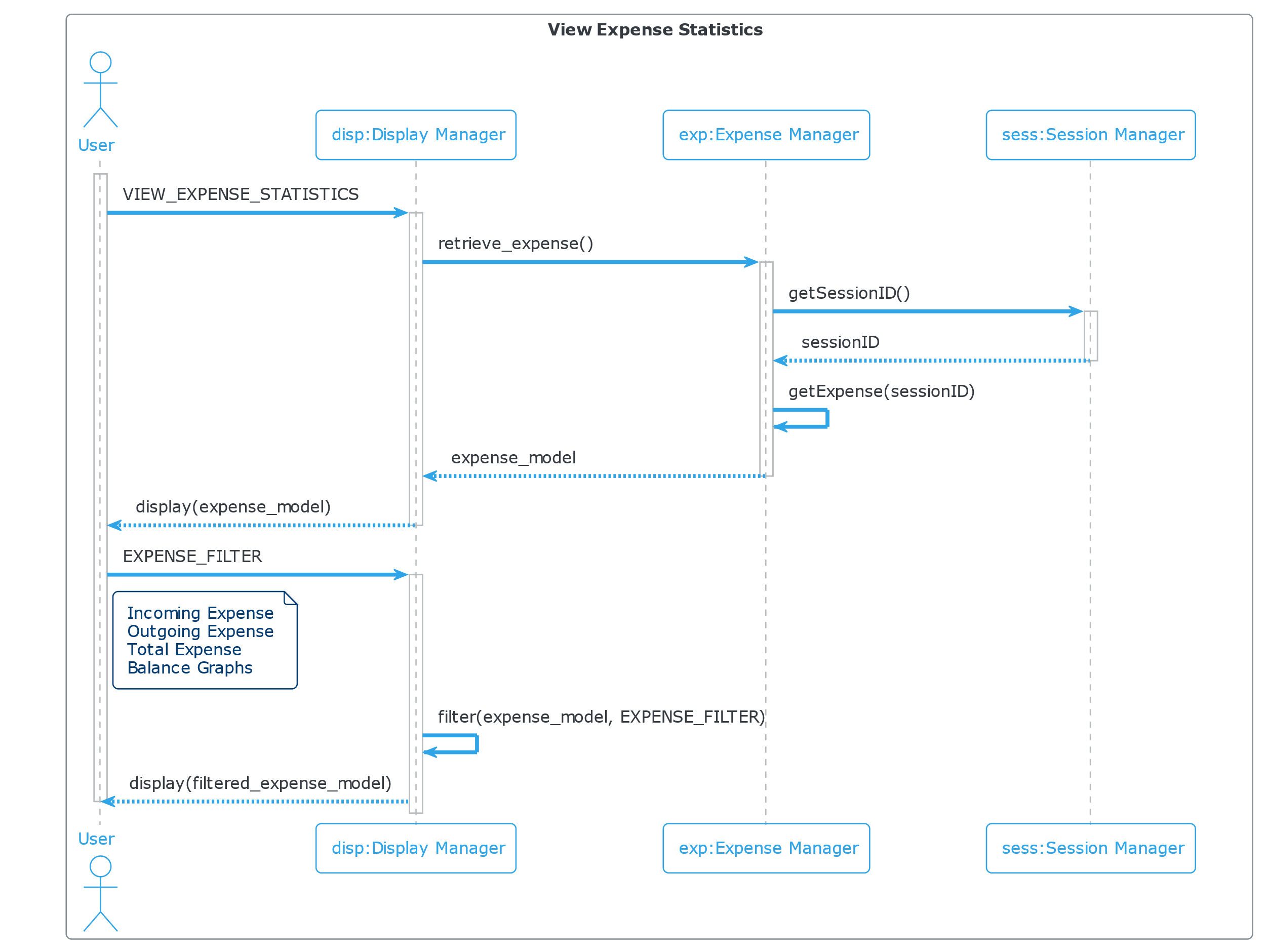
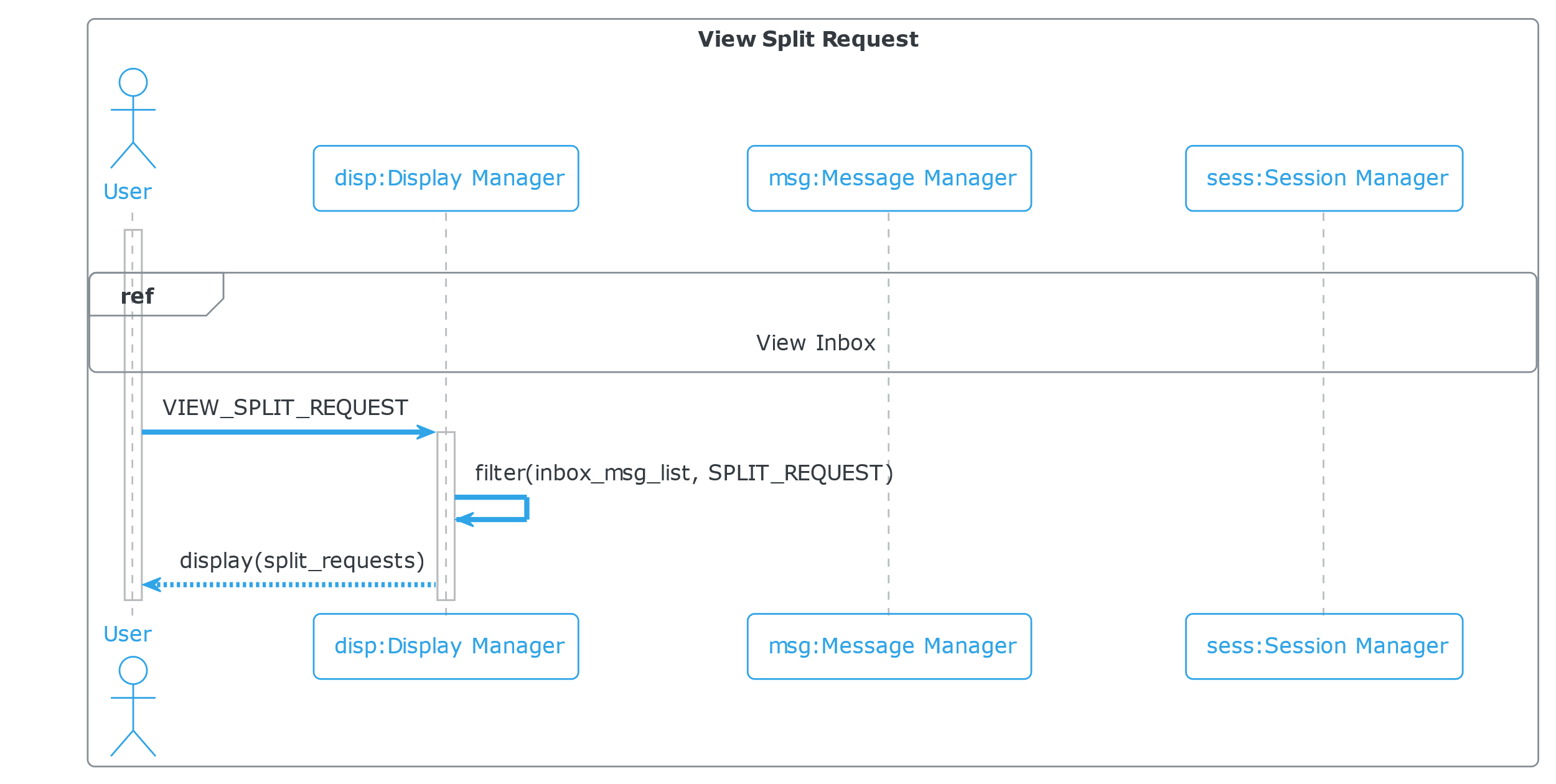


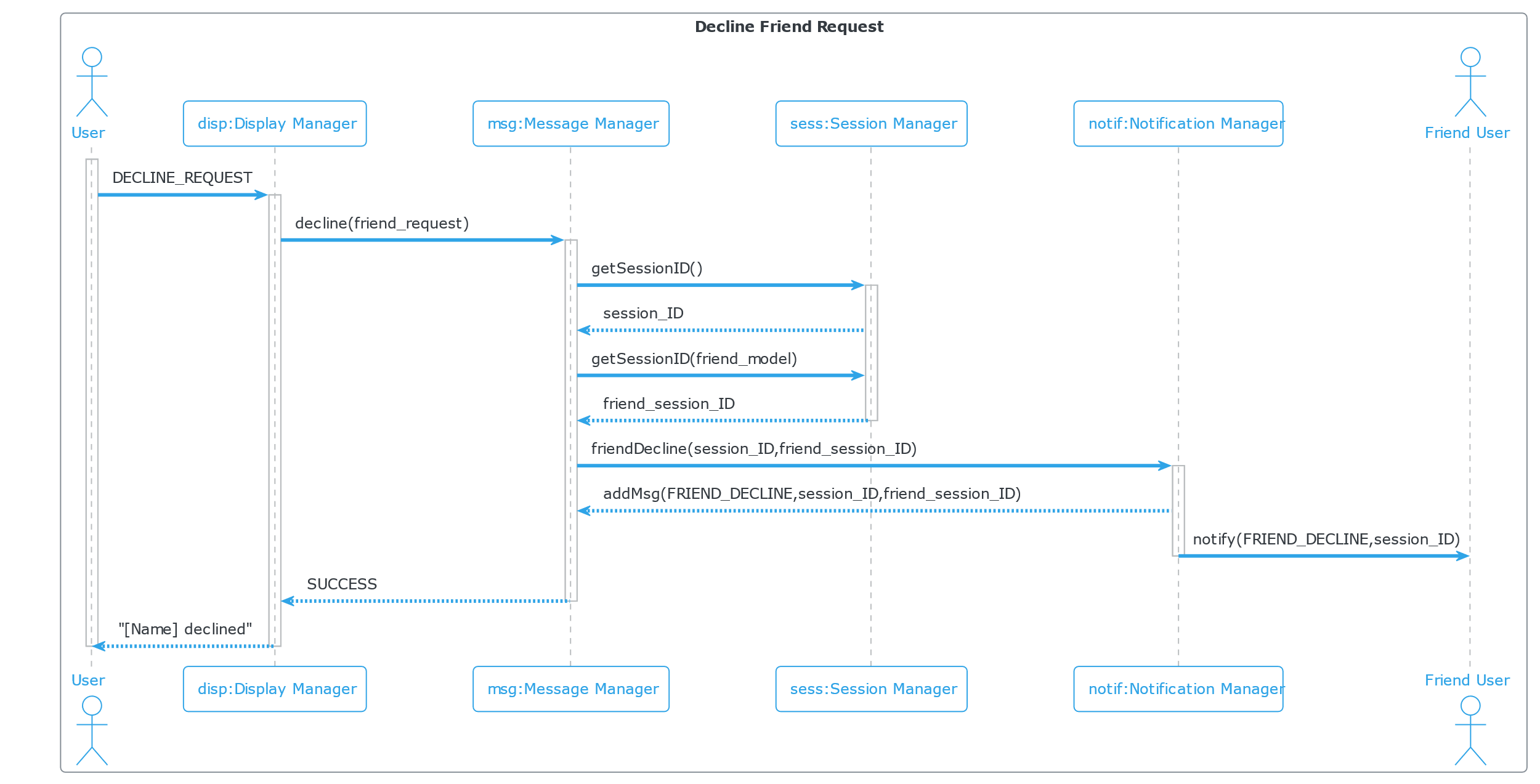
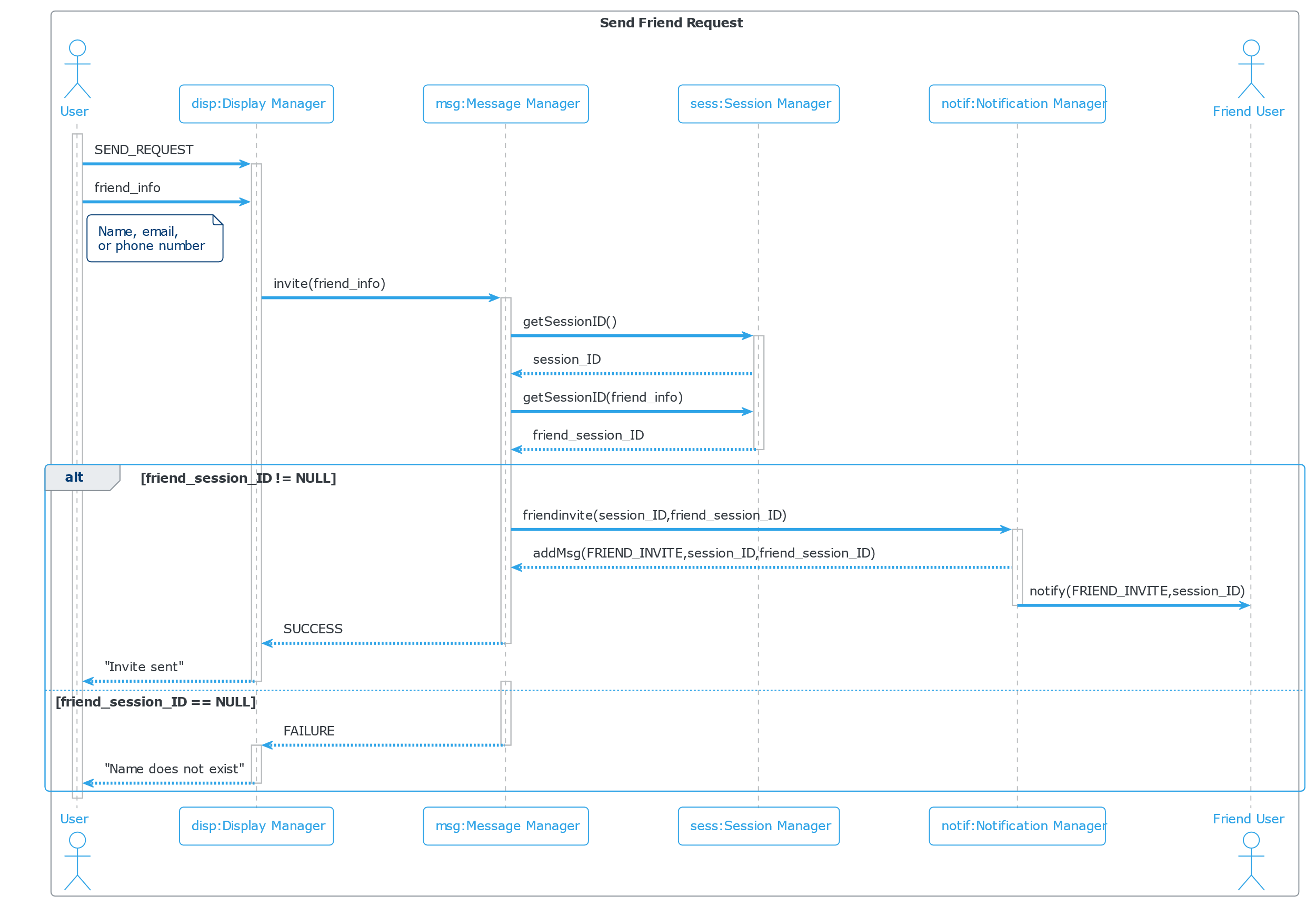
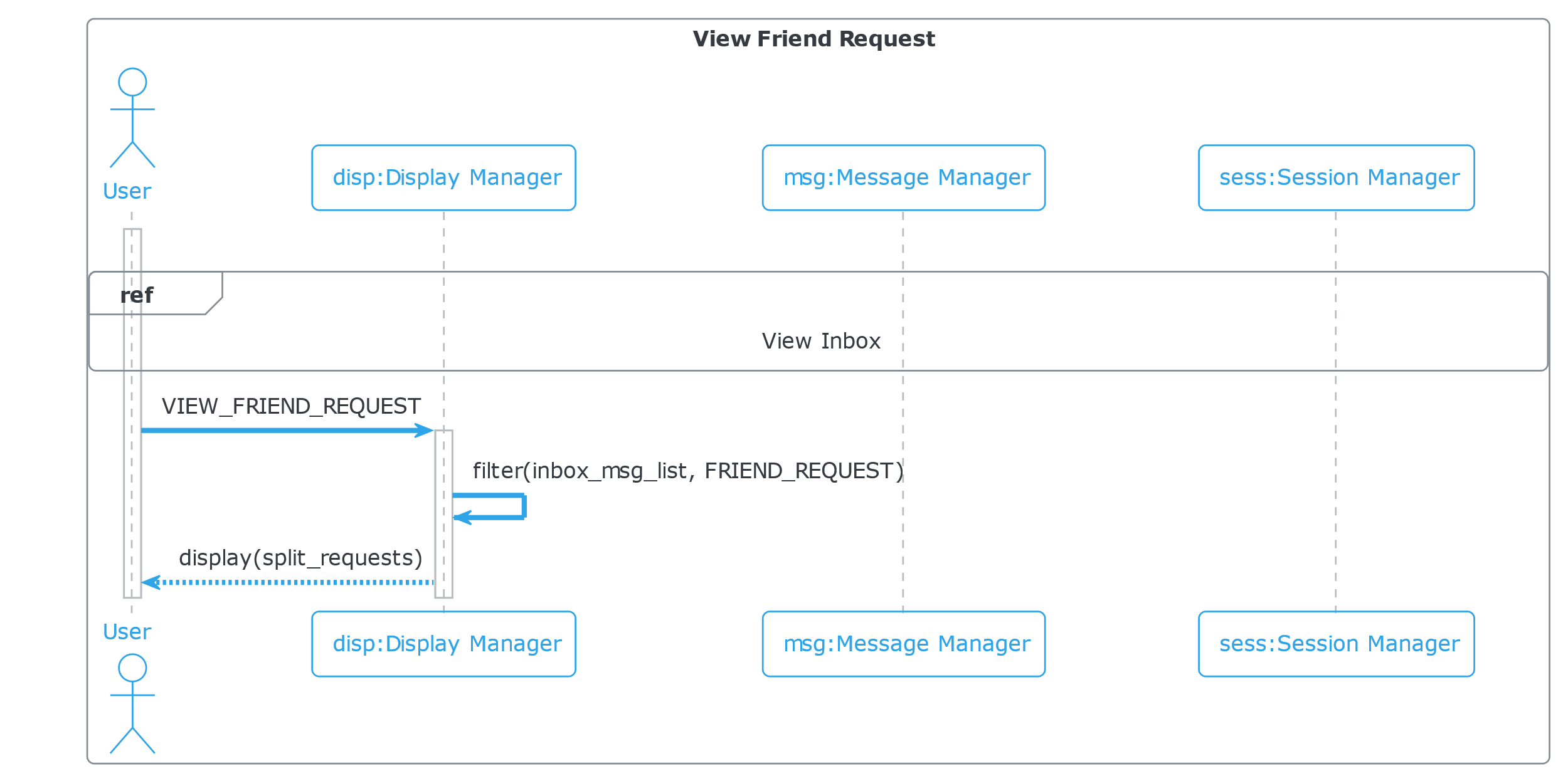
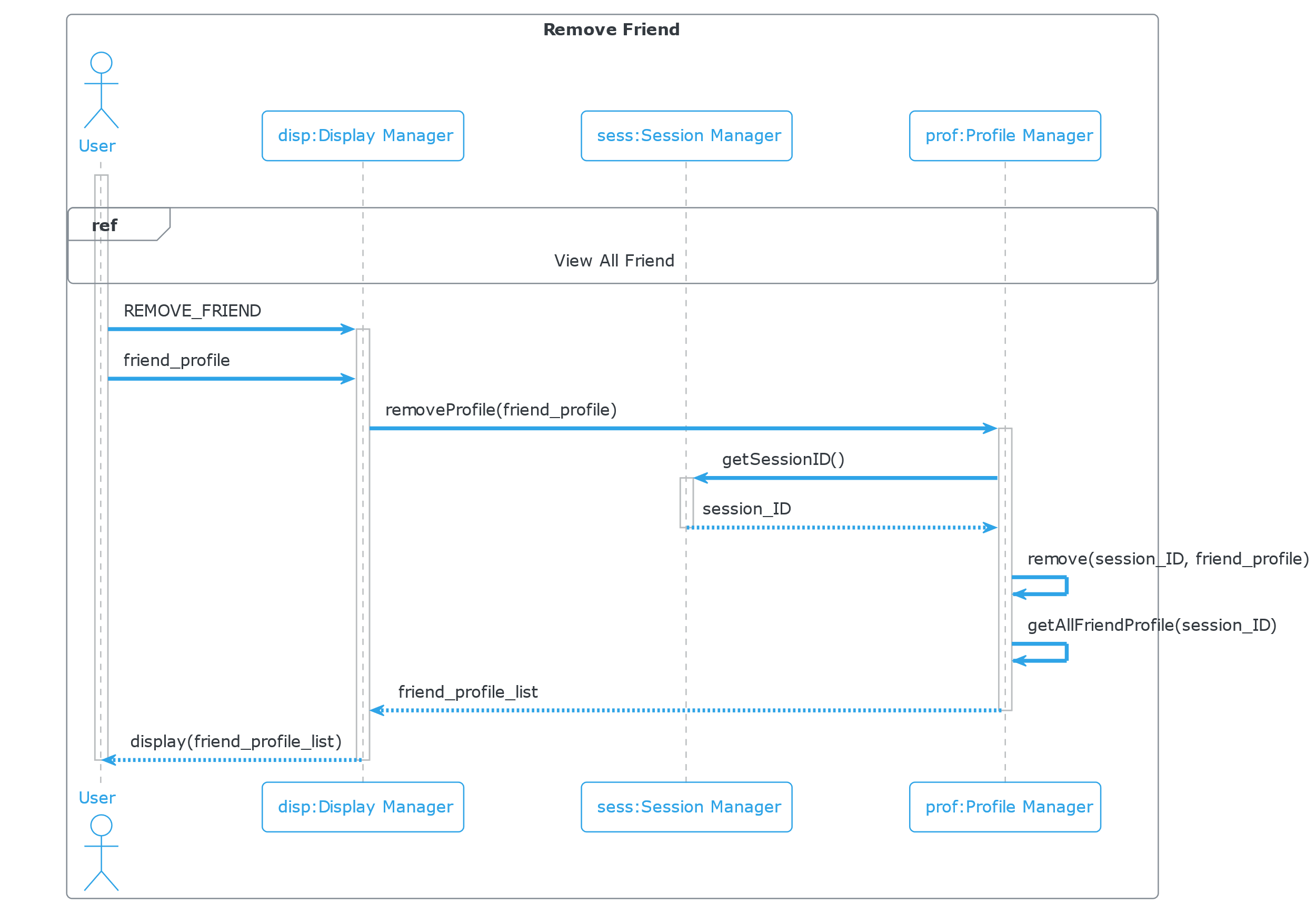
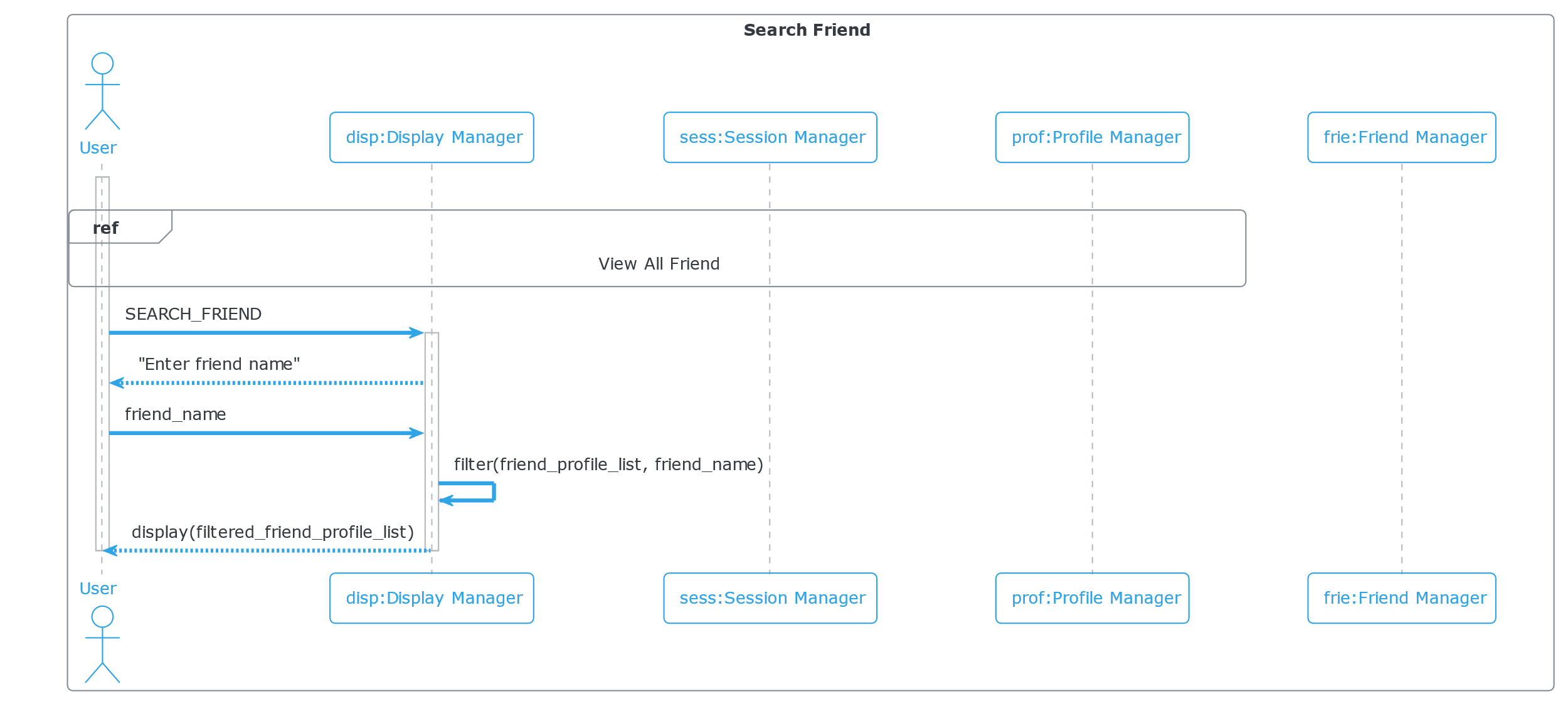
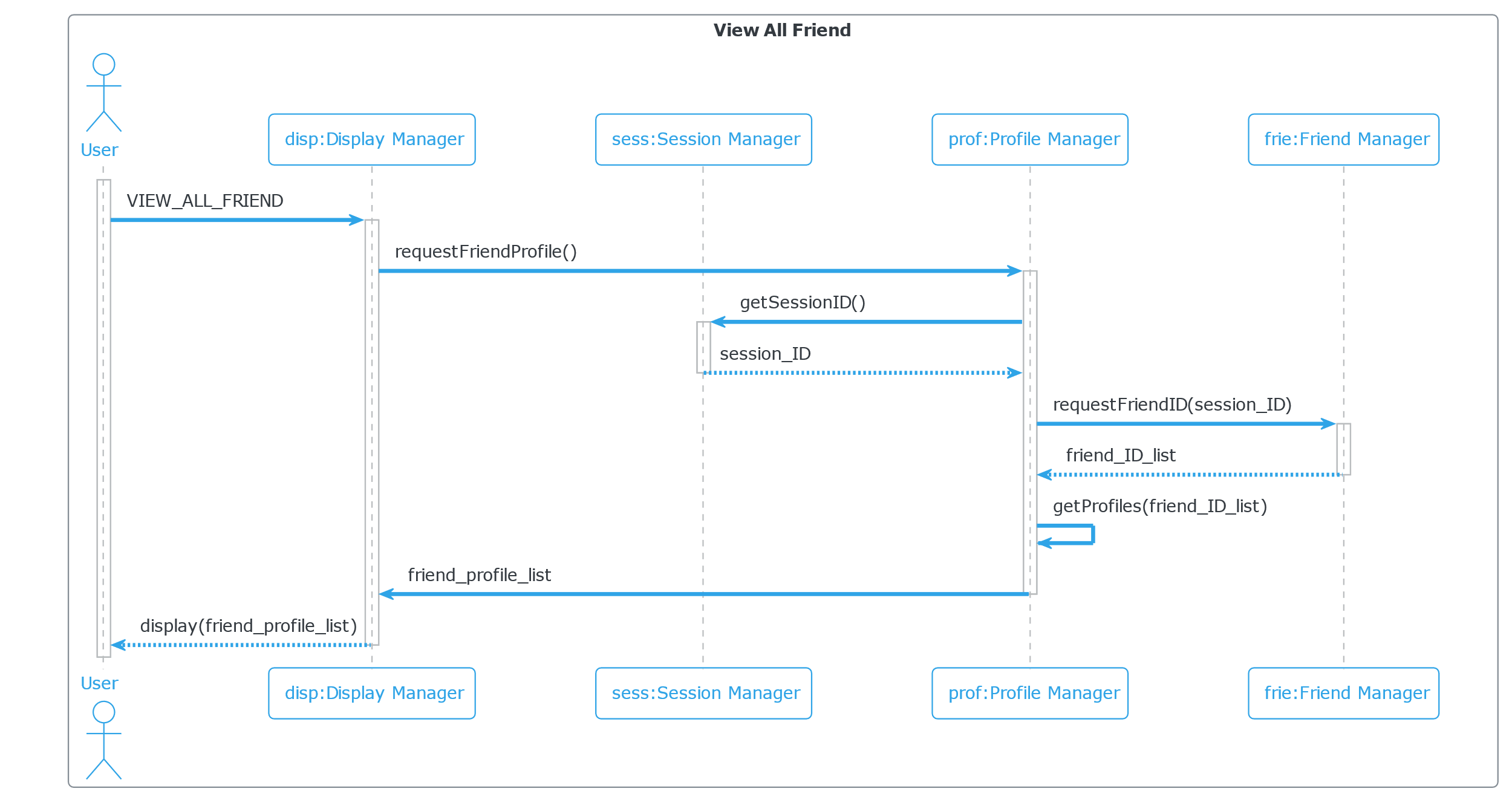
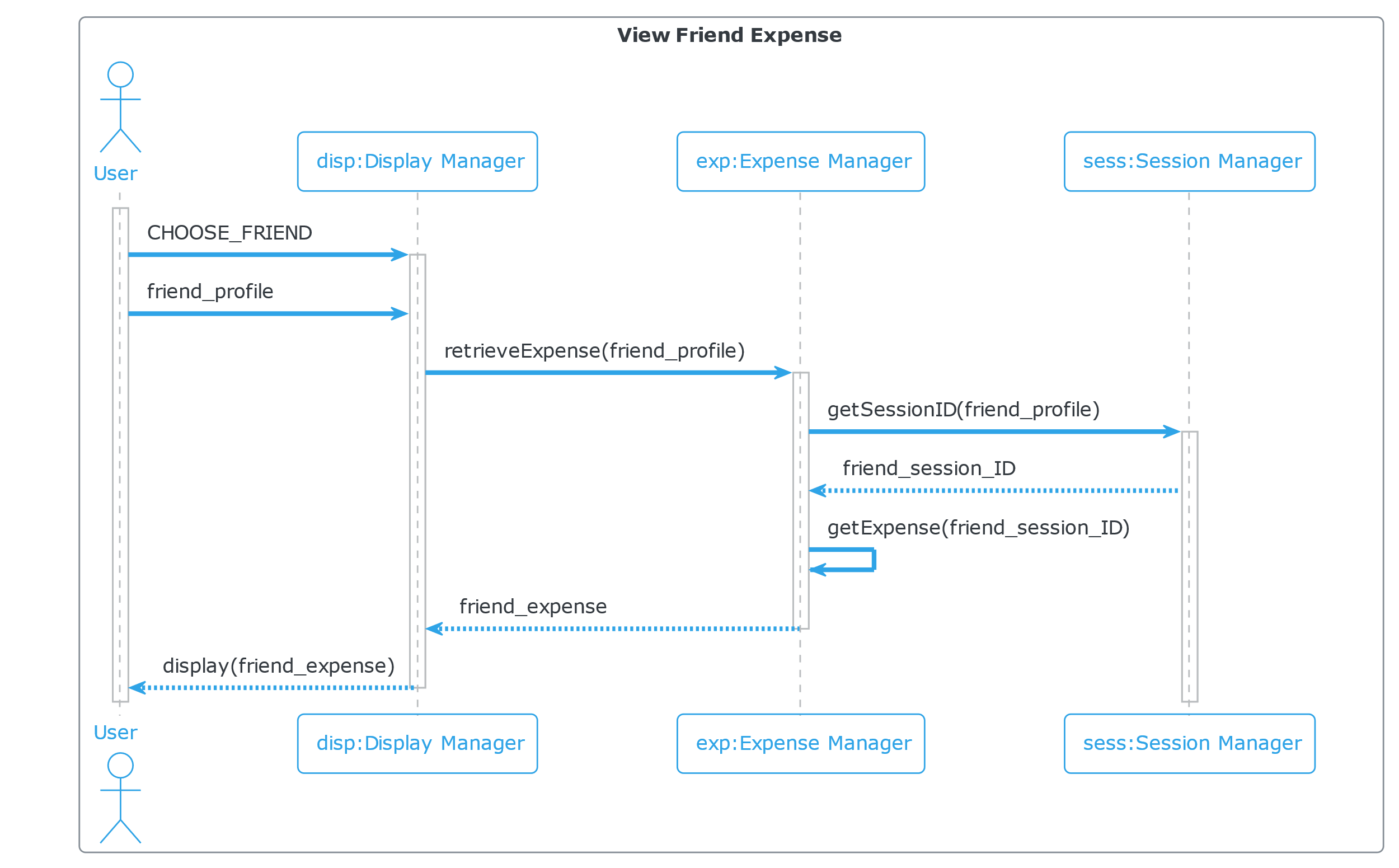
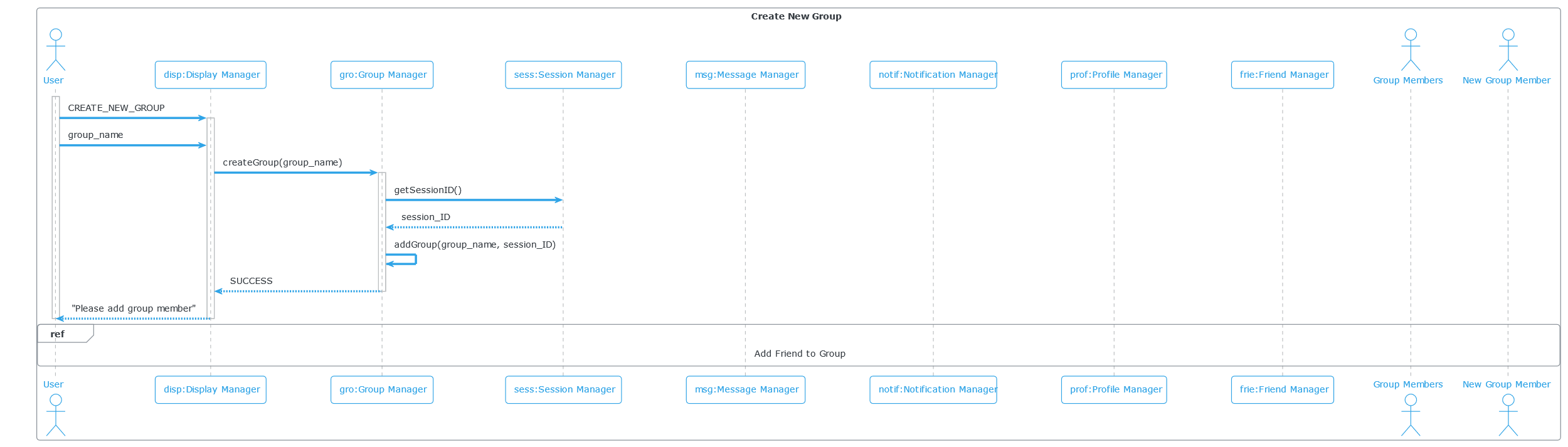
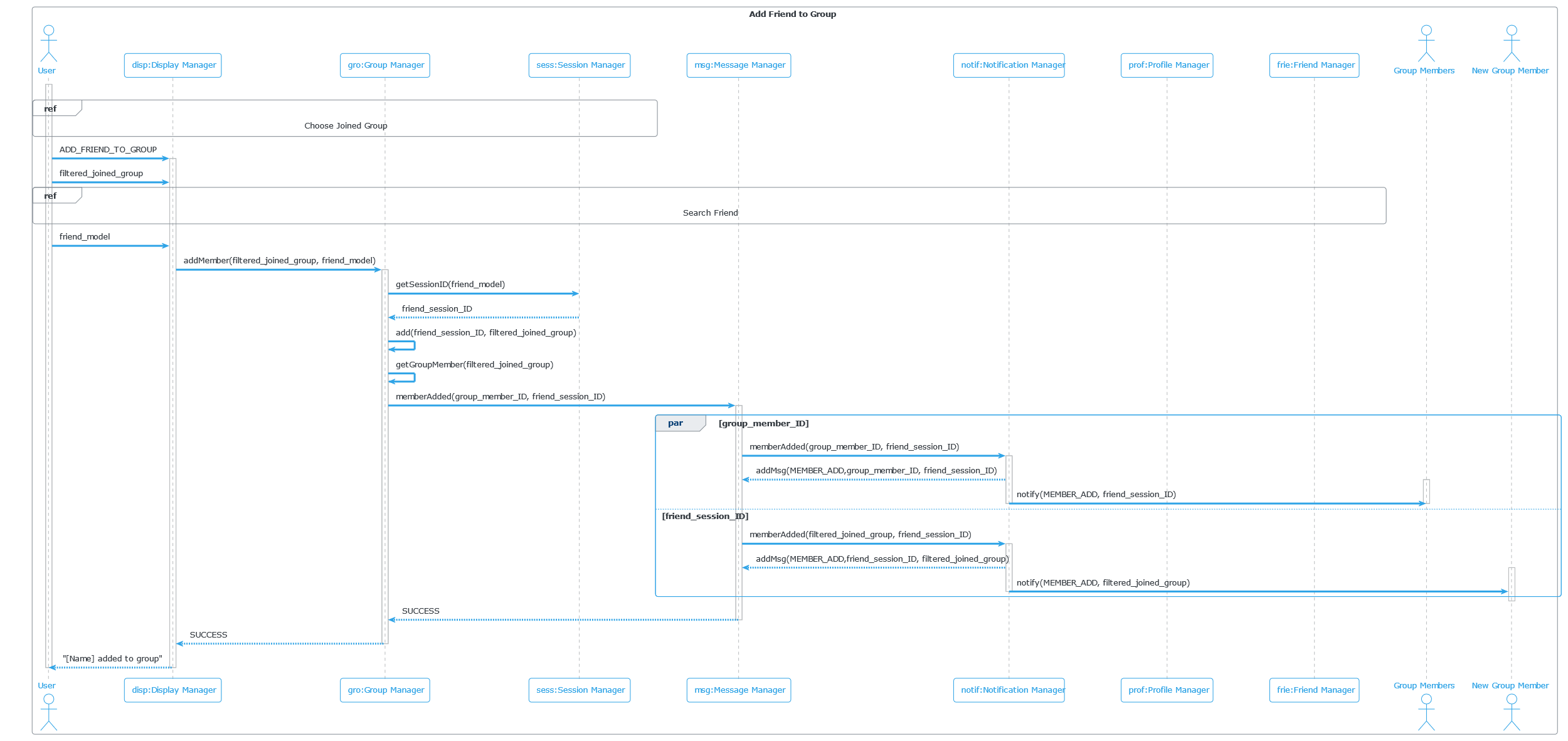
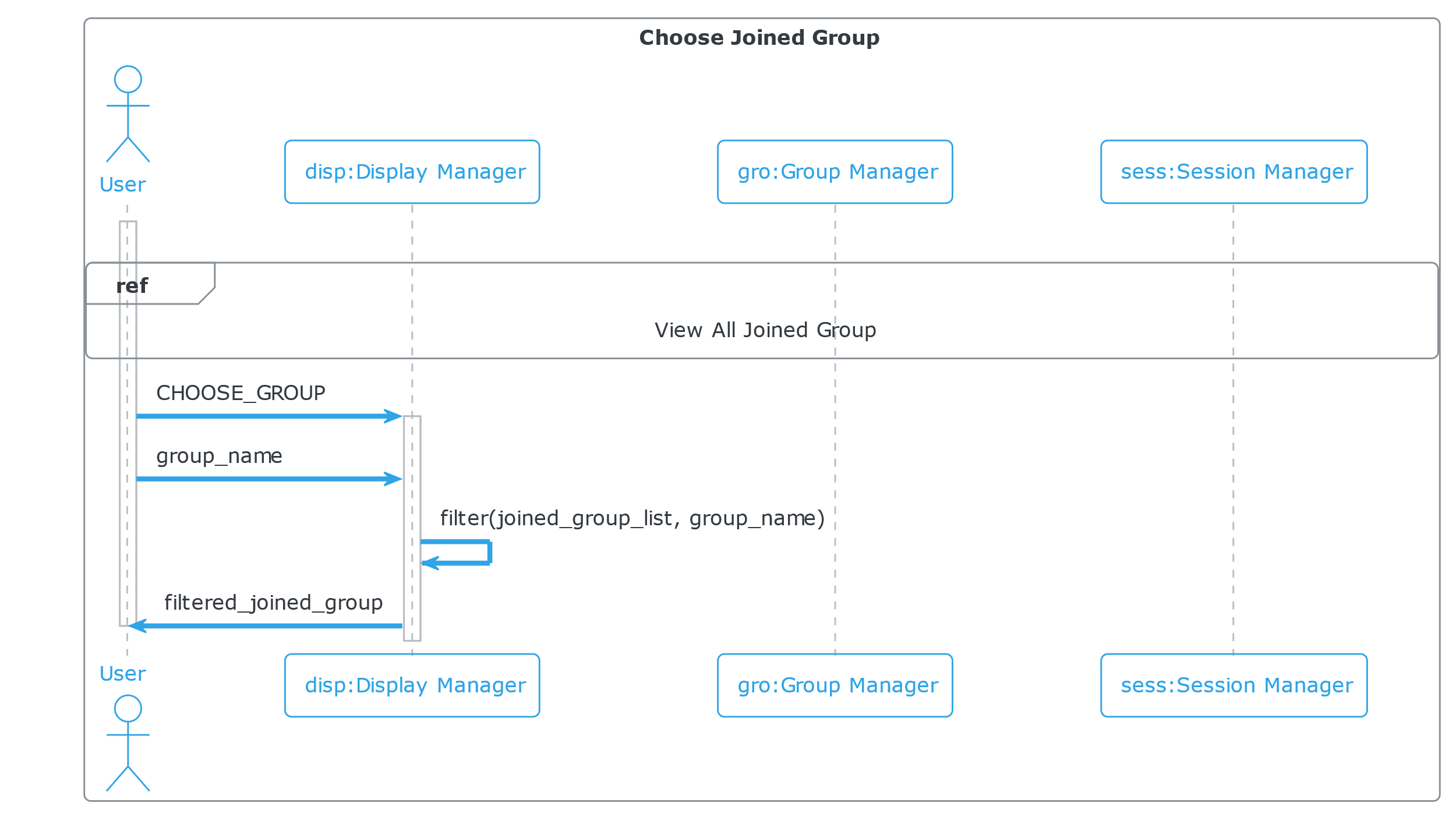
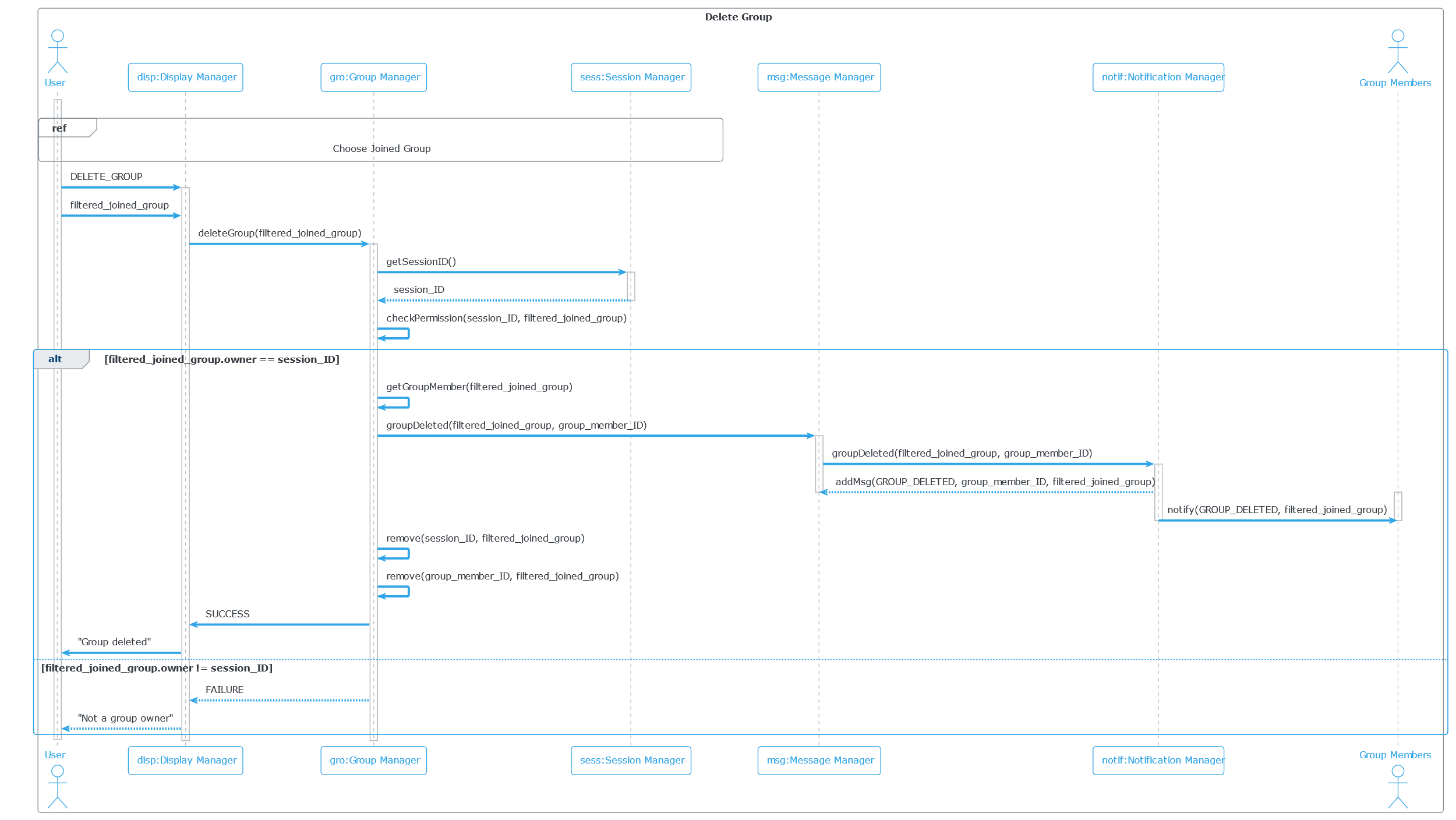
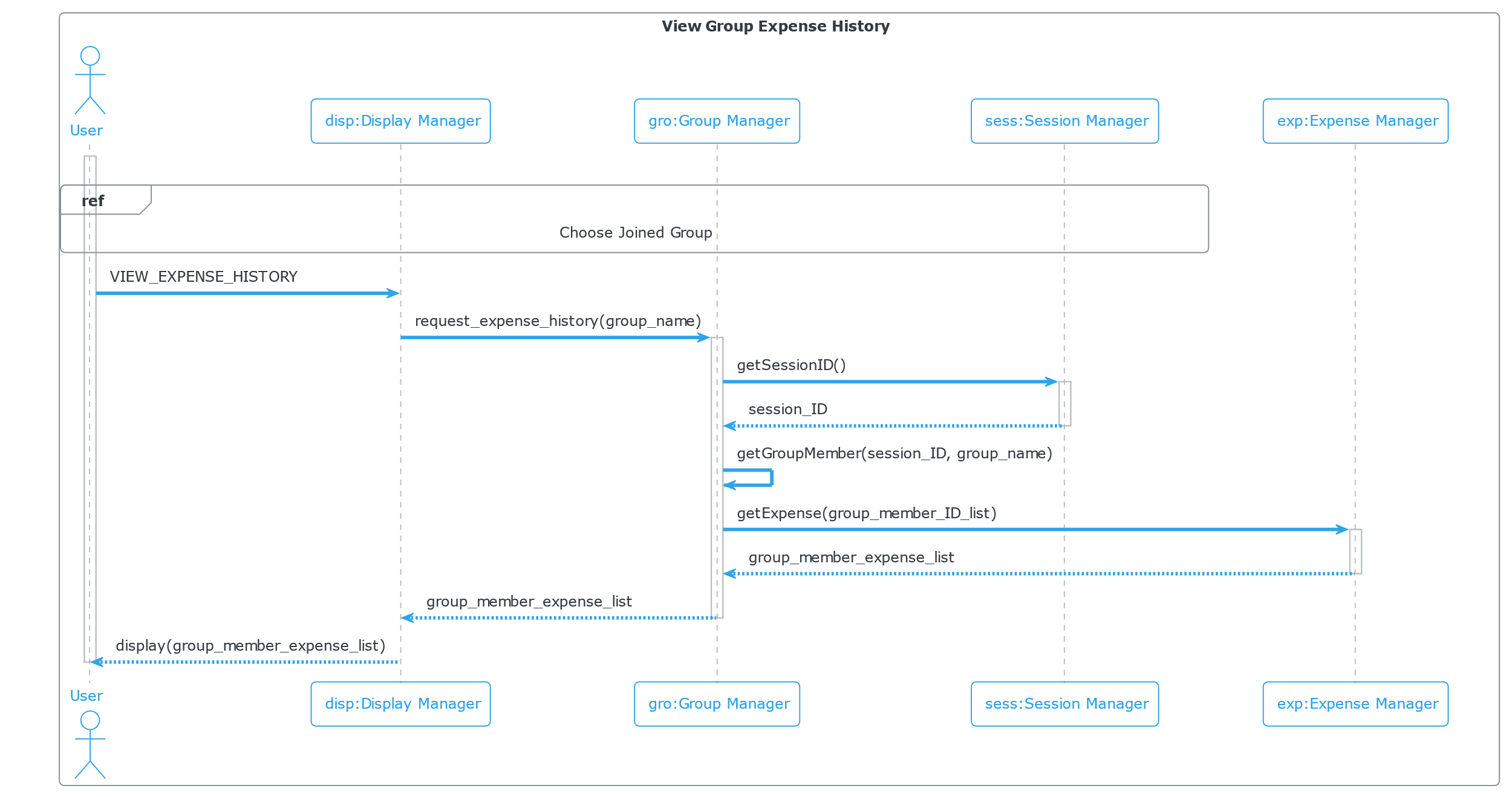
**7. [15 POINTS]** Sequence diagram – Provide sequence diagrams (like Figure 5.6 and Figure 5.7) for each use case of your project. Please note that there should be an individual sequence diagram for each use case of your project. (Ch 5 and Ch 7)

|  |
| --- |
| 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) |

****

****

****

****

**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

**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)

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

9.2. Layered architecture pattern (like Figure 6.9)

9.3. Repository architecture pattern (like Figure 6.11)

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

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

A diagram of a user flow

Description automatically generated