Requirements and Design

1. Change History

Change Date	Modified Sections	Rationale
October 26th, 2026	4.6 Use Case Sequence Diagram	Added sequence diagrams for 5 use cases
October 27th, 2026	4.7 Design and Ways to Test Non-Functional Requirements	Added testing methods for non-functional requirements
October 27th, 2026	3.4. Use Case Description***	Updated wording to all be in active voice
October 27th, 2026	3.5. Formal Use Case Specifications (5 Most Major Use Cases)***	Re-ordered failure scenarios to be chronological
October 27th, 2026	3.7. Non-Functional Requirements***	Removed the incorrect/unecesary non-functional requirements and added more detail to the "Usability" use case
October 27th, 2026	3.2. Use Case Diagram***	Changed to just be "Project User" and "Project Administrator"
November 8th, 2025	3.5. Formal Use Case Specifications	Added frontend implementation notes for use cases
November 27th, 2025	Final Review	Removed implementation notes and refined content for final version

> *** Changes done based on the recommendation of our TA

2. Project Description

The app is designed to help simplify the management of group projects for school by providing a central platform for managing their collaboration, organization, and communication. Users can create projects in which other members can be added. Within these projects, users can track their project progress, add tasks and deadlines, see task statuses, and be updated on all of this in real-time.

The app also integrates with Google Calendar to synchronize deadlines across all members to help keep the entire group on schedule. Teams can also manage their shared resources by having quick access to common links or files, as well as track their expenses for each member. Additionally, users can use a built-in chat feature to communicate with other members of the same project. This app is ideal for anyone looking to keep their group projects organized, on track, and well-coordinated.

3. Requirements Specification

3.1. List of Features

Authentication: Access to the app uses a unified Google OAuth flow: users sign in with the single "Sign In With Google" action which handles both first-time account provisioning and returning sign-ins. Authenticated users can sign out and request account deletion; account removal is performed by calling the backend DELETE endpoint. This uses Google's external API to authenticate users and fulfills the "implement an authentication feature using an external authentication service" requirement.

Manage Project: Users can create projects and become the project owner. When creating a project, the project owner provides the project name and optional description. After creating a project, the owner receives a unique invitation code that must be shared manually with other members. The creating user is automatically added as both the project owner and a member. The project owner can view project details, manage members, and delete projects they own. Any user can view a list of active projects they own or have joined. Users can view detailed information about projects where they are the owner or a member. When new users join a group via invitation code, the list of members is automatically updated for all viewers through WebSocket connections.

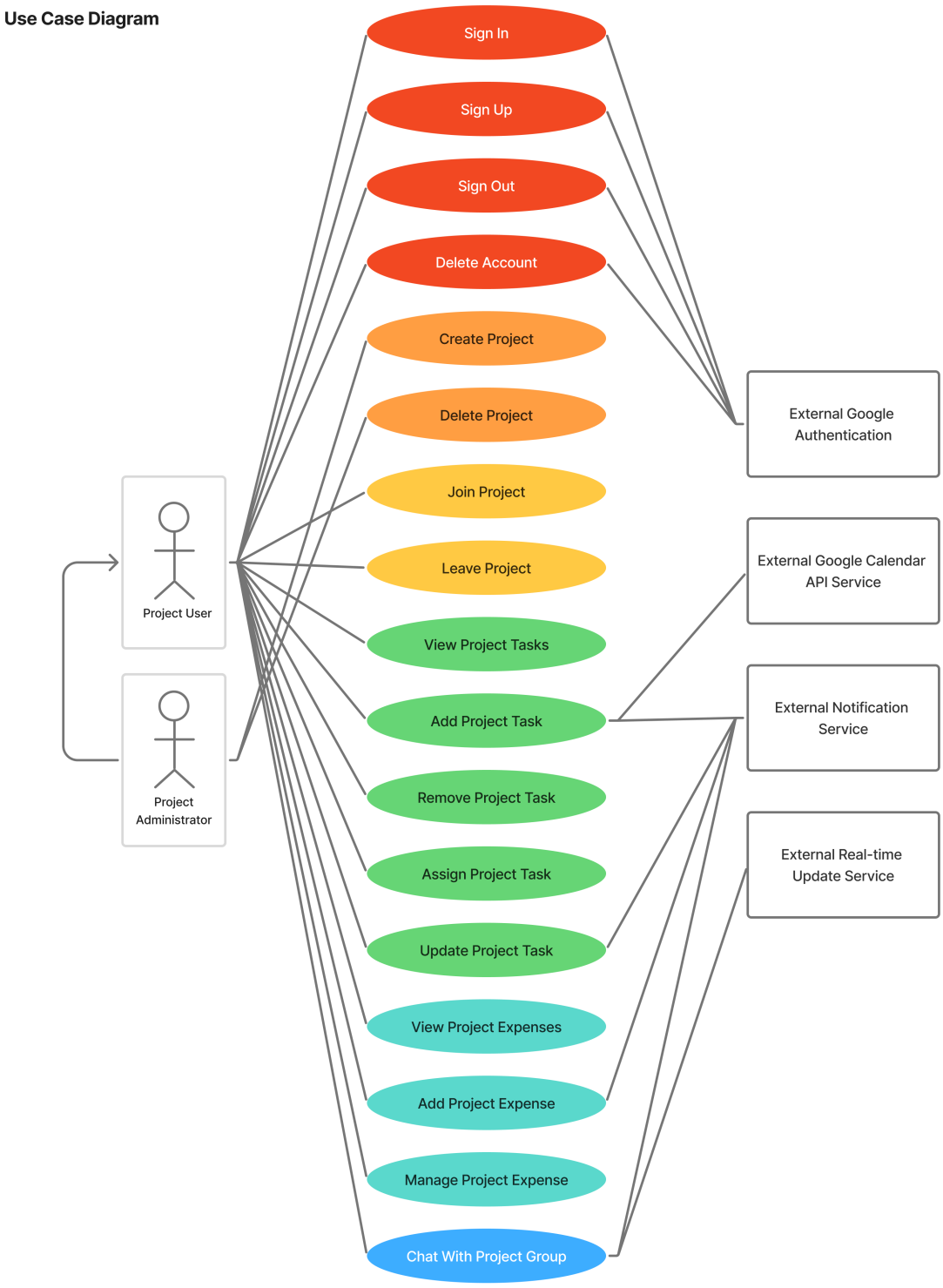
Manage Project Tasks: Users can synchronize tasks with everyone else in the project. Task creation uses a single-form dialog where name, assignee(s), status, and optional deadline are entered before submission. Users can add deadlines to tasks and assign them to specific users. Each task has a status (Not Started, In Progress, Completed, Blocked, or Backlog) so team members are aware of the progress on each task. Task status is displayed with color-coded indicators for enhanced visual feedback.

Sync Project Deadlines Across all Members: Users can sync their Google Calendar to tasks so everyone has the same deadlines synced. Calendar synchronization is performed by backend synchronization services which call the Google Calendar API on behalf of enabled users; users enable calendar sync in Settings. This fulfills the "at least one of the features must use an external service accessible via an API" requirement via Google Calendar API.

Manage Project Resources: Users can split expenses related to the project in an expenses tab. Expenses are automatically split equally among selected members. Users can track current expenses through a detailed table view showing expense information including per-person amounts. Users can also add common links/items to a "resources tab" which all members can access. This feature satisfies the "at least one of the features must involve computations implemented by your group" requirement, as all expense splitting logic is handled internally rather than by an external API.

Communicate With Members: Users can use the chat feature for real-time communication with their teammates in the group. The chat system uses WebSocket connections (Socket.IO) for instant message delivery through persistent connections. This feature fulfills the "one of the features must involve changes happening in the app's content or state as a response to an external event" requirement as it is a multi-user chat with real-time updates.

3.2. Use Case Diagram



3.3. Actors Description

1. **Project User**: User who is invited to a project. Can add and view tasks on the project page. Users can also add resources to the resources tab and add/assign expenses to other users. Users can also chat with other members of a given project in the "chat" tab. Users can update the status of tasks in the project page. A user can also be a part of multiple projects at the same time.

2. **Project Administrator**: User who manages a project. Can add and remove members. Can delete the project. Can edit project name and assign "Project User" or "Project Administrator" roles to any other member of the project.

3.4. Use Case Description

Use cases for feature 1: Authentication This feature enables users to securely sign up, sign in, sign out, and delete their accounts in the app using Google's external authentication service. The system uses a unified Google OAuth flow where both new users and returning users use the same "Sign In With Google" button.

1. **User registers/signs up for the first time**: The user opens the app and clicks "Sign In With Google". The app utilizes Google OAuth with a unified authentication flow that handles both new account creation and existing account login automatically.

2. **Returning user signs in**: The user opens the app and clicks "Sign In With Google" to authenticate with their existing Google credentials.

3. **Returning user signs out**: The user is already logged into the app and clicks the "Sign Out" button to sign out of their account. They are redirected to the sign in/sign up page.

4. **Returning user deletes account**: The user is already logged into the app and clicks the "Delete Account" button to delete their account. They are redirected to the sign in/sign up page. This calls a DELETE endpoint to remove this account from the backend.

Use cases for feature 2: Manage Project This feature allows users to create, join, view, and manage projects, including viewing project details and deleting projects they own, while keeping membership and project information up-to-date in real time.

1. **Creating a new project**: From the home screen, a user clicks "Create Project" to create a new project. They enter the project name and optional description. After submitting, a POST endpoint saves the project data into the database and generates a unique invitation code. The creating user is automatically added as the project owner and member. The user can then share the invitation code manually with other users.

2. **Joining an existing project**: Users join existing projects by entering an invitation code. They obtain the code from a project member (shared manually), enter it in the "Join Project" dialog, and click "Join". The backend validates the code and adds the user to the project's members list. The members list is updated in real-time for all users viewing the project through WebSocket connections.

3. **Access/view projects**: From the home screen, users view the projects they are currently part of and click on each specific project to view additional details, including tasks, chat, and expenses.

4. **Delete project**: A user clicks into a specific project and if they are the creator of this project, they can click "Delete project" to delete it.

Use cases for feature 3: Manage Project Tasks Users can create, edit, assign, and track tasks within a project, including setting deadlines and task status, ensuring all team members stay informed of progress.

1. **Add a new task with a deadline**: Inside a specific project, a user clicks "Add task" to add a new task. They enter task details including name, assignee, status, and optional deadline in a single dialog form.

Tasks display with color-coded status indicators (green for completed, gold for in progress, purple for backlog, red for blocked, blue for not started).

2. **Edit an existing task**: Inside a specific project, a user clicks a task to edit information. After clicking "Save", this data is updated in the database through a PUT request.

3. **Update status of a task**: Inside a specific project, on the project task screen, a user updates a task's status. This is updated in the database through a PUT request.

4. **View all project tasks**: Inside a specific project, users see a project task screen with all current tasks for the project. They are able to scroll down to view all tasks.

Use cases for feature 4: Sync Project Deadlines Across all Members Users can link their Google Calendar to project tasks through backend synchronization services, automatically syncing deadlines and receiving notifications, keeping everyone aligned using the Google Calendar API.

1. **Allow app access to your Google Calendar**: Users can connect their Google Calendar through the Settings tab. The system uses backend synchronization scripts to manage the calendar integration.

2. **Sync project tasks assigned to you to your Google Calendar**: When calendar access is granted, the backend synchronization services automatically sync project task deadlines to the user's Google Calendar through the Google Calendar external API. Tasks appear on their calendar on the deadline date.

Use cases for feature 5: Manage Project Resources Users can manage project expenses by adding, splitting, and tracking payments among team members, as well as share common resources, with all calculations handled internally by the app.

1. **Add a new project expense and split this between project group**: Inside a specific project, users click on the "Expenses" tab where they click the "Add Expense" button. They enter the expense name, description, amount, who paid, and select members to split between. The expense is automatically split equally among selected members. For example, if an expense is $10 split among 4 members, each member owes $2.50.

2. **View outstanding balances owed to other teammates**: Inside a specific project, users click on the "Expenses" tab where they view all expenses in a detailed table format showing all expense information including per-person amounts.

3. **Update an expense as complete if all money is paid, or pending if you require money from other teammates**: Inside a specific project, users click on the "Expenses" tab, where they update expenses as fully paid or pending. Users can only update expenses that they previously created.

Use cases for feature 6: Communicate With Members Users can chat in real time with project teammates, sending and receiving messages, allowing dynamic communication through WebSocket connections.

1. **Communicate/chat with group members in an existing project through the chat tab**: Inside a specific project, users click on the "Chat" tab, where they chat with all members of their team. To send a new message, they enter their message and press "Send". Messages are delivered instantly via WebSocket connection.

2. **Receive chat notifications**: If a user is using the app and someone messages a chat in a project that they are part of, messages appear in real-time through the persistent WebSocket connection.

3.5. Formal Use Case Specifications (5 Most Major Use Cases)

Use Case 1: User Creates a New Account

Description: A brand new user can create a new account and successfully sign into the app. The user can then sign out of the app and sign in again with the newly created account.

Primary actor(s): A new user that wants to sign up for the app.

Main success scenario:

1. The user clicks on the app and sees a Google authentication interface with a "Sign In With Google" button.
2. User clicks the "Sign In With Google" button and is prompted to enter their email and password through Google's OAuth flow.
3. User enters their credentials and completes the Google authentication.
4. User successfully authenticates with Google. If this is a new user, the system automatically creates an account. The user then lands on the home page.

Failure scenario(s):

- 1a. Rate limit
    - 1a1. User sends too many requests at the same time
    - 1a2. System may display messages out of order
- 2a. User is not authenticated
    - 2a1. System rejects user when the user attempts to access protected features
    - 2a2. System prompts user to sign in again before continuing
- 3a. Server network error (can't connect to server)
    - 3a1. System displays an error message to the user indicating it cannot connect to the server

Use Case 2: Creating a New Project

Description: An authenticated user creates a new project, becomes its owner, and receives a unique invitation code that can be shared with other members. The project details (name, owner, code, members) are saved in the backend and displayed in the app. The creating user is automatically added as both the project owner and a member.

Primary actor(s): Authenticated user (already signed in)

Main success scenario:

1. User opens the app and navigates to the home screen.
2. User sees a "Create New Project" button and clicks it.
3. A "Create New Project" form appears prompting the user to enter the project name and optional project description.
4. User types in the project name in the corresponding text field.
5. User optionally enters a project description.

6. User clicks "Create" button (which is enabled when project name is not empty).
7. The backend creates the project, generates a unique invitation code, and adds the user as owner and member.
8. User is redirected to the home screen, where the newly created project is now visible.
9. User clicks on the project to view additional details, including project name, invitation code, members, and tabs for tasks, chat, and expenses.
10. User can share the invitation code manually with other users who wish to join the project.

Failure scenario(s):

- 1a. User is not signed in
  - 1a1. If the user tries to create a project without being signed in, the app prevents them from continuing
  - 1a2. The app prompts the user to sign in before they can create a project
- 4a. User enters empty project name
  - 4a1. The "Create" button remains disabled
  - 4a2. User cannot proceed until a project name is entered
- 7a. Network error
  - 7a1. If the app cannot connect to the server, it shows an error message letting the user know there is a network issue

Use Case 3: Communicate/Chat With Group Members in an Existing Project

Description: An authenticated user can access an existing project and use the chat tab to talk to other members of their project group to communicate updates, roadblocks, and deadlines. Messages are saved to the backend and appear in real time for all group members through WebSocket connections.

Primary actor(s): Authenticated user with access to a specific project

Main success scenario:

1. User opens the app.
2. User selects a specific project from the home screen to enter that project.
3. Inside the project, user sees and clicks on the "Chat" tab button.
4. The chat interface opens where user can view previous messages.
5. User types a message in the input field.
6. User clicks "Send" button.
7. A loading indicator appears on the send button during transmission.
8. The message appears in the chat for all project members immediately via WebSocket connection.
9. Other project members see the message in real-time and can react or reply.

Failure scenario(s):

- 6a. Sending too many messages (Rate limit)
  - 6a1. User tries to send multiple messages very quickly
  - 6a2. Some messages may appear out of order
  - 6a3. Messages that fail to send show a "Message failed to send" indicator
- 8a. Network error

- 8a1. If the app cannot connect to the server, it shows an error message letting the user know messages cannot be sent

Use Case 4: Adding Project Expenses

Description: An authenticated user can access an existing project that they are a part of and use the expense tab to record project expenses. Expenses are saved to the backend and the amount that each group member owes is updated for all members.

Primary actor(s): Authenticated user with access to a specific project

Main success scenario:

1. User is on home page and has already been added to an existing project.
2. User clicks into the desired project they'd like to manage expenses for.
3. User sees and clicks the "Expense" tab button.
4. The expenses page displays with an "Add Expense" button.
5. User clicks "Add Expense" button.
6. An "Add New Expense" form appears on screen.
7. User inputs expense description in the "Description" text field.
8. User inputs a valid numeric amount in the "Amount" text field.
9. User selects who paid for the expense from the "Paid By" dropdown.
10. User selects which members to split the expense between using checkboxes in the "Split Between" section.
11. User clicks "Add Expense" button on the form.
12. User is brought back to the expenses tab.
13. User views the updated expenses in a detailed table format showing:
    - Description of the expense
    - Total amount
    - Who paid
    - Who the expense is split between
    - Per-person amount (automatically calculated as equal split)
14. Other group members can view the updated expenses on the expenses tab in the same table format.

Failure scenario(s):

- 1a. User is not authenticated
  - 1a1. System rejects user when the user attempts to add an expense
  - 1a2. System prompts user to sign in again before continuing
- 7a. User inputs empty description
  - 7a1. User clicks "Add Expense" button
  - 7a2. Dialog opens with error message: "Please fill all fields correctly"
- 8a. User inputs non-numeric amount
  - 8a1. User inputs text like "NON_INTEGER" in the amount field
  - 8a2. User clicks "Add Expense" button
  - 8a3. Dialog opens with error message: "Please fill all fields correctly"
- 9a. User does not select who paid
  - 9a1. User leaves "Paid By" dropdown unselected

    - 9a2. User clicks "Add Expense" button
    - 9a3. Dialog opens with error message: "Please fill all fields correctly"
- 10a. User does not select who to split expense between
    - 10a1. User does not check any boxes in "Split Between" section
    - 10a2. User clicks "Add Expense" button
    - 10a3. Dialog opens with error message: "Please fill all fields correctly"
- 11a. Server network error (can't connect to server)
    - 11a1. System displays an error message to the user indicating it cannot connect to the server

Use Case 5: Creating/Assigning Project Tasks and Deadlines to Group Members

Description: An authenticated user can access a project they are a part of and create/assign project tasks to other group members with a deadline. Tasks are saved to the backend and project tasks are displayed to a task board to all group members, displaying the associated group members and deadline.

Primary actor(s): Authenticated user with access to a specific project

Main success scenario:

1. User is on home page and has already been added to an existing project.
2. User clicks on the project to open the project screen.
3. User sees and clicks the "Task Board" tab button.
4. The task board displays with a "Create Task" button.
5. User clicks "Create Task" button.
6. A "Create New Task" form dialog appears on screen.
7. User inputs task name in the "Task Name" text field.
8. User selects a single assignee from the "Assignee" dropdown.
9. User selects task status from the "Status" dropdown (options: Not started, In progress, Completed, Blocked, Backlog).
10. User selects a future date using the "Deadline" date picker.
11. User clicks "Create" button.
12. User is returned to the task board.
13. User sees the newly created task displayed with:
    - Task name
    - Assigned user
    - Deadline date
    - Status with color-coded indicator (Green: Completed, Gold: In progress, Purple: Backlog, Red: Blocked, Blue: Not started)
14. Other group members can view the task on their task board with the same information.

Failure scenario(s):

- 1a. User is not authenticated
    - 1a1. System rejects user when the user attempts to create a task
    - 1a2. System prompts user to sign in again before continuing
- 7a. User inputs empty task name
    - 7a1. User leaves "Task Name" field empty
    - 7a2. User fills other fields and clicks "Create"

    - 7a3. Dialog opens with error message: "Task name cannot be empty"
- 8a. User does not select an assignee
    - 8a1. User leaves "Assignee" dropdown unselected
    - 8a2. User fills other fields and clicks "Create"
    - 8a3. Dialog opens with error message: "Assignee cannot be empty"
- 10a. User selects a past date
    - 10a1. User selects a date that has already passed using the date picker
    - 10a2. User fills other fields and clicks "Create"
    - 10a3. Dialog opens with error message: "Please enter a future date"
- 11a. Server network error (can't connect to server)
    - 11a1. System displays an error message to the user indicating it cannot connect to the server

3.6. Screen Mock-ups

[Screen mockups to be added if needed]

3.7. Non-Functional Requirements

1. **Performance**

    - **Description**: The system must provide timely, responsive user interactions with appropriate feedback mechanisms. Basic UI navigation such as selecting tasks and navigating between tabs must respond within 0.1 second to feel instantaneous. Slightly more complex interactions, like navigating between project views and retrieving data, must respond within 1 second to preserve the user's flow of thought. More complicated tasks that take longer should provide visual feedback: operations exceeding 10 seconds must display a progress indicator and allow interruption, while tasks between 2 and 10 seconds should at least provide lighter feedback (e.g., a busy cursor or loading animation).
    - **Justification**: Quick response times are essential for maintaining smooth user flow and productivity. Delays in response and feedback would be disruptive and negatively affect user experience.
    - **Source(s)**: https://www.nngroup.com/articles/response-times-3-important-limits/

2. **Security**

    - **Description**: Users should not be able to access data belonging to other users. The system must ensure proper authentication and authorization, preventing unauthorized access to user accounts, projects, tasks, expenses, and chat messages. All API endpoints must validate user permissions before returning sensitive data.
    - **Justification**: Protecting user privacy and data security is critical for maintaining trust and preventing unauthorized access to sensitive project information, financial data, and personal communications.
    - **Source(s)**: OWASP Top 10 Security Risks, Industry standard security practices

---

4. Design Specification

4.1. Main Components

1. **User Management Service**
    - **Purpose**: Handles user authentication (via Google Auth), profile CRUD operations, and managing the list of projects a user owns or is a member of.
    - **Rationale**: Separating user identity from project data allows for secure, independent management of user accounts and simplifies integrating with external authentication providers. This clear separation also makes it easier to manage user-specific preferences in the future.
2. **Project & Collaboration Service**
    - **Purpose**: The core service providing CRUD operations for projects, project members (via invitation codes), tasks, expenses, and chat messages. It manages the entire lifecycle of a collaborative project.
    - **Rationale**: Grouping all project-related entities (tasks, expenses, chat) into a single service reduces complexity and coupling between services. Since these features are tightly interrelated (e.g., a task update might trigger a chat message), having them in one service simplifies data consistency and real-time updates for the project group.
3. **Notification & Real-time Sync Service**
    - **Purpose**: Manages real-time features like live chat and notifications for deadlines or new expenses. It pushes updates to all relevant users' front-ends when project data changes.
    - **Rationale**: Isolating real-time functionality ensures that the main Project Service remains fast for CRUD operations. Using a dedicated service with WebSockets provides a responsive, collaborative experience and allows for independent scaling of the real-time components.
4. **Calendar Integration Service**
    - **Purpose**: Acts as a dedicated bridge between our app and the Google Calendar API. It handles the OAuth flow and translates internal task deadlines into calendar events on users' behalf.
    - **Rationale**: Encapsulating all third-party API interactions for calendar syncing in one service contains the complexity of external API changes and error handling. This prevents the core Project Service from being polluted with Google-specific code, making the system more maintainable.
5. **Expense Tracking Service**
    - **Purpose**: Manages all financial operations within projects, including expense creation, automatic expense splitting calculations among team members and balance tracking.
    - **Rationale**: Financial calculations require high accuracy and specialized validation logic. Separating expense management into its own service ensures data integrity for monetary transactions, provides focused error handling for financial operations, and allows for independent scaling as expense complexity grows. This isolation also makes it easier to implement audit trails and financial reporting features in the future.

4.2. Databases

1. **NoSQL Database**
    - **Purpose**: To store all project data, including user profiles, projects, tasks, expenses, and chat messages. A NoSQL database is suitable for the flexible, nested data structures (like a task with

assignees or an expense with splits) that our project requires.
- **Rationale**: Chosen over a traditional SQL database for its schema flexibility, which is beneficial in the early stages of development when data models may evolve. It also scales well for read/write operations common in collaborative applications.

4.3. External Modules

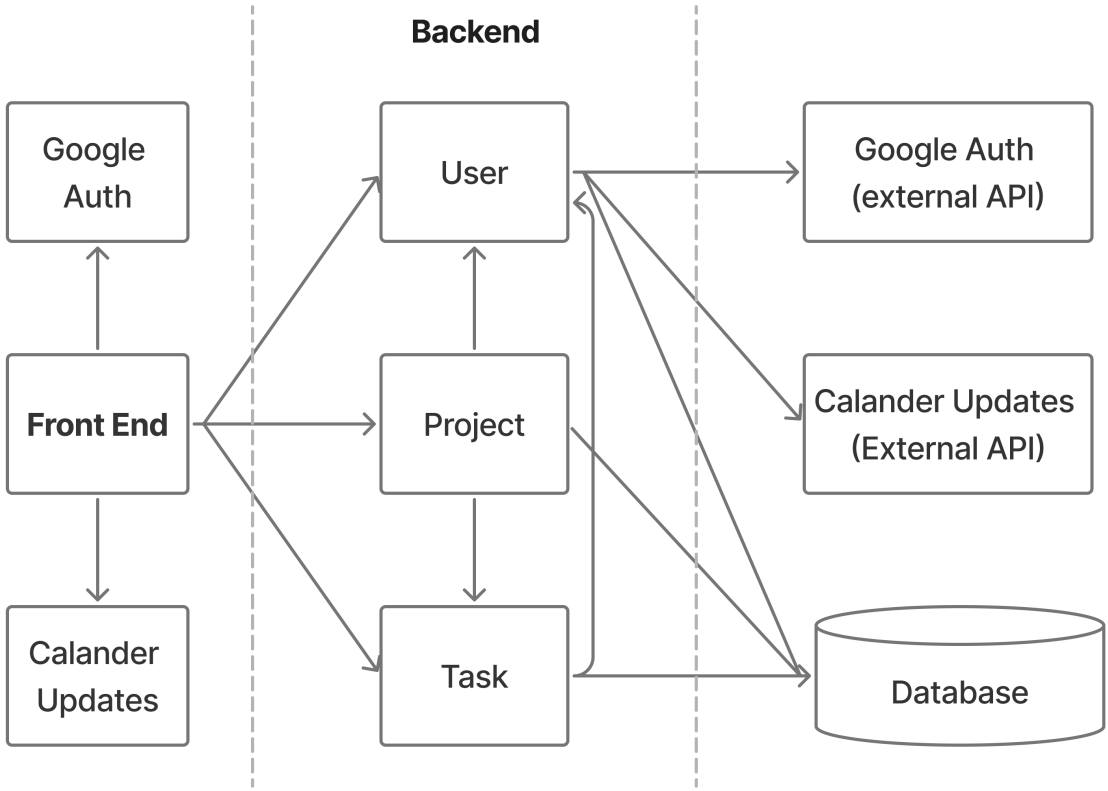
1. **Google Authentication API**
   - **Purpose**: To handle user sign-up and sign-in securely.
2. **Google Calendar API**
   - **Purpose**: To sync project task deadlines to users' personal calendars.

4.4. Frameworks

1. **AWS**
   - **Purpose**: To host our server
   - **Reason**: Course tutorial is easy to follow and group members are familiar with AWS
2. **Node.js**
   - **Purpose**: For developing and running our backend server, which will include endpoints for our APIs
   - **Reason**: Node.js is lightweight and scalable. It is industry standard as well, which will make it very easy to integrate with Azure to run our backend

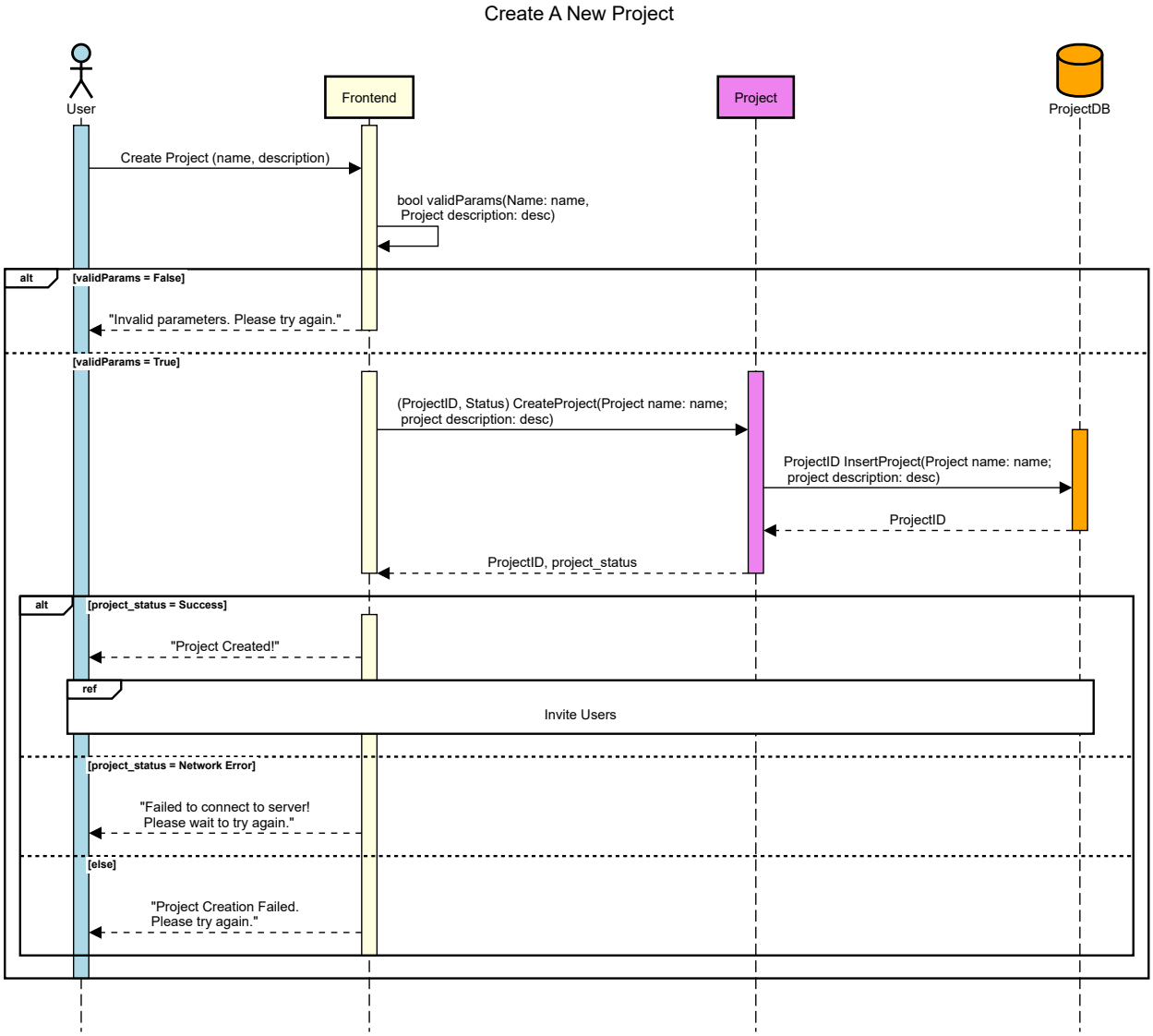
4.5. Dependencies Diagram

Dependencies Diagram



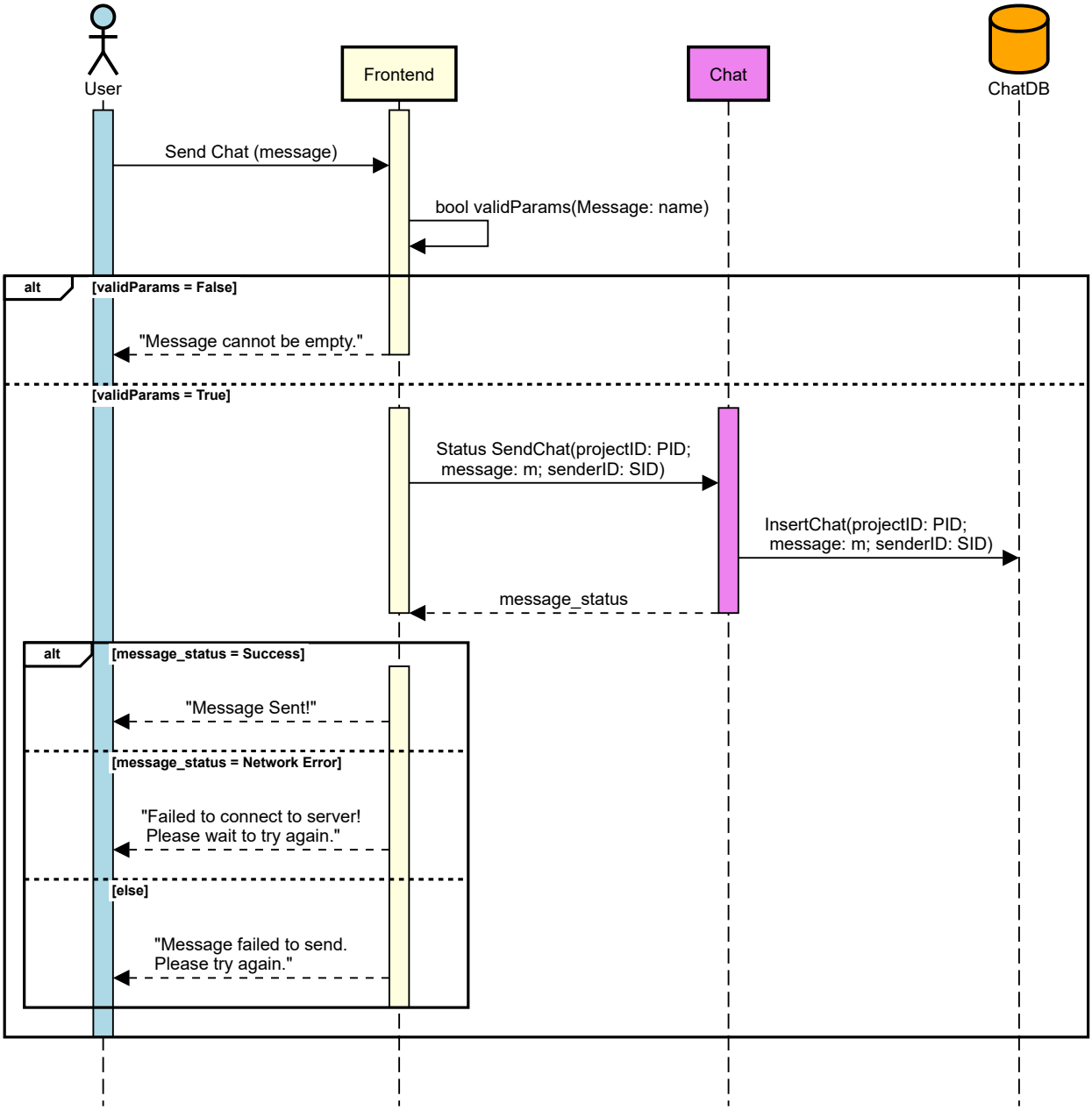
4.6. Use Case Sequence Diagram (5 Most Major Use Cases)

1. [**Create a New Project**]

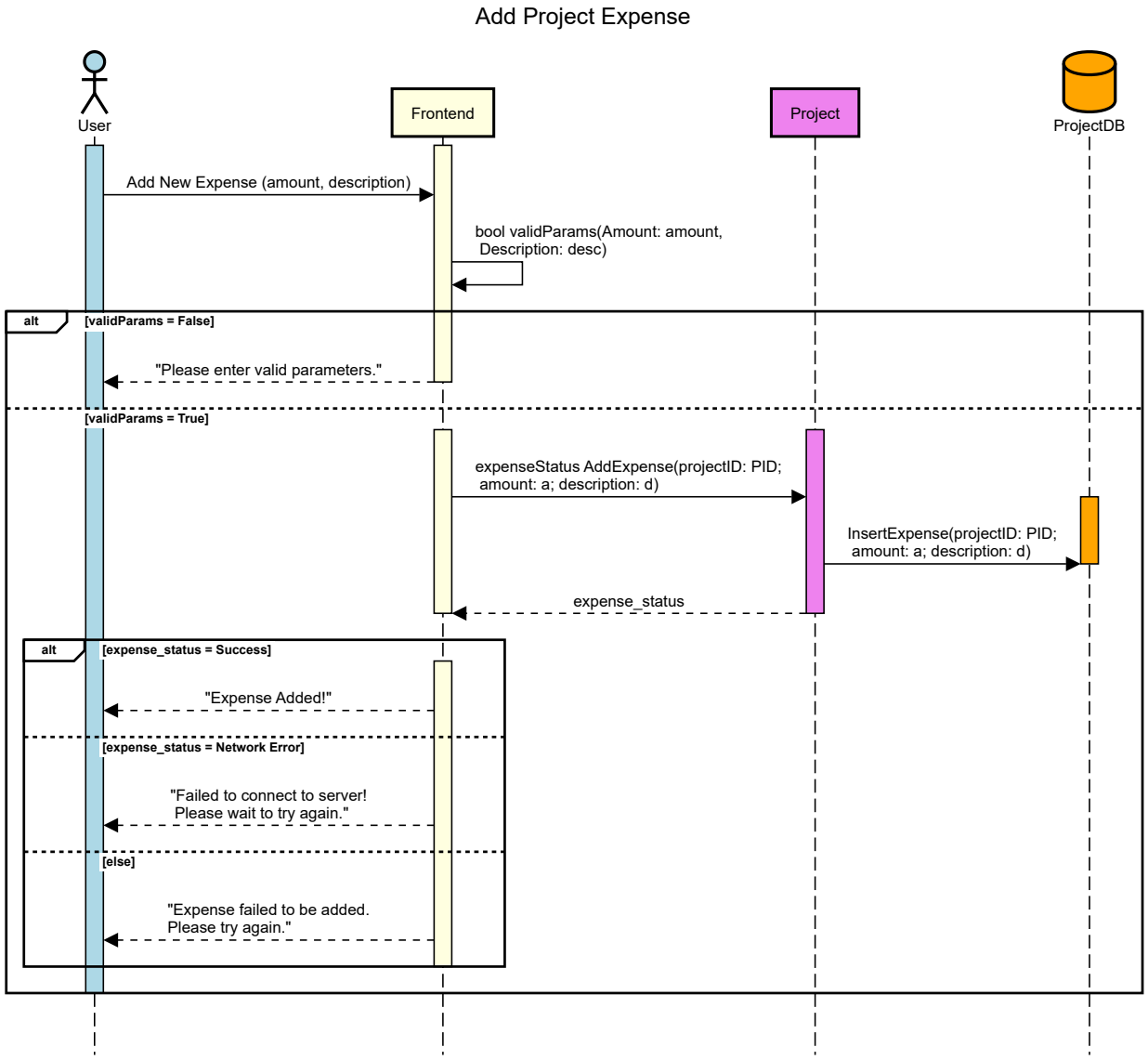


2. [**Communicate/Chat With Group Members In An Existing Project**]

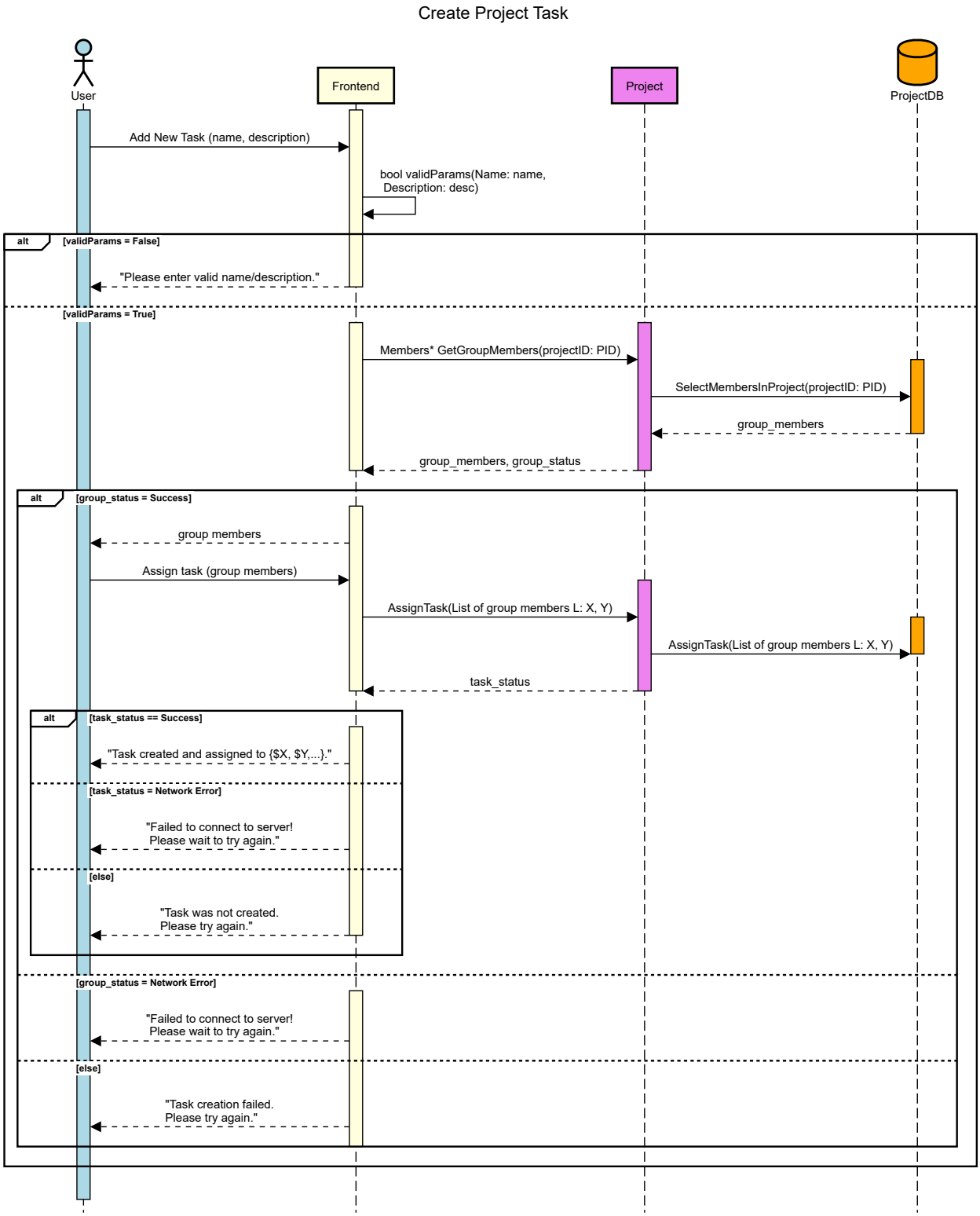
Communicate/chat with group members in an existing project



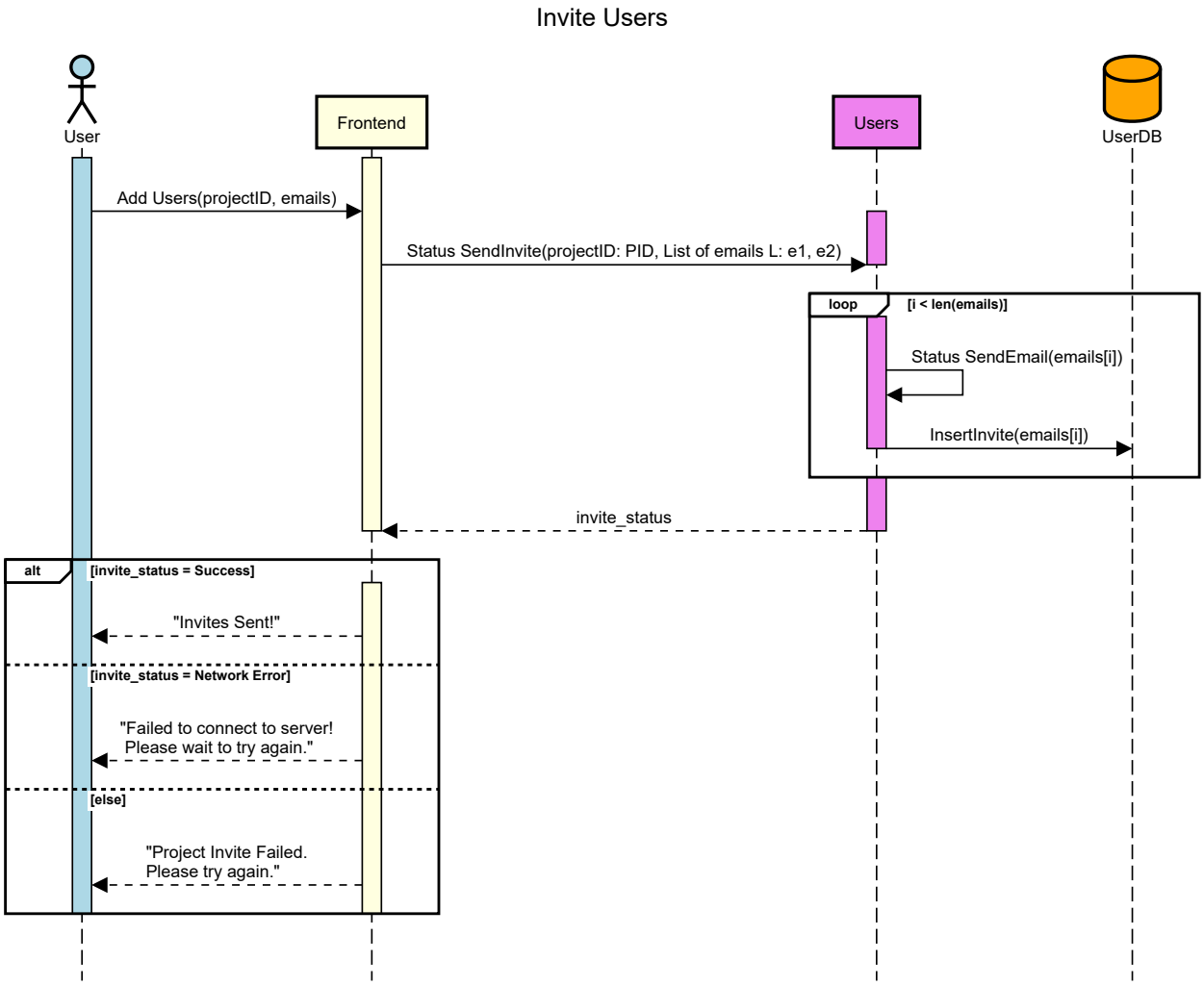
3. [**Add Project Expenses**]



4. [**Create Project Task**]



5. [**Invite Users To Project**]



4.7. Design and Ways to Test Non-Functional Requirements

1. **Performance**

   - **Implementation**: Only load recent chat messages, not the entire history. Implement efficient data loading strategies to minimize response times.
   - **Testing Approach**:
     - Time how long it takes to load a project page (must be under 1 second)
     - Measure the time it takes to perform specific actions (switching tabs, loading tasks, sending messages) to verify they are below the required response times
     - Test with multiple concurrent users to ensure performance doesn't degrade under load
     - Verify that switching between tabs feels instant (under 0.1 seconds)

2. **Security**

   - **Implementation**: Use JWT tokens for authentication with proper expiration. Implement authorization checks at the API level to verify users can only access their own data and projects they are members of. Validate all user permissions before returning sensitive data.
   - **Testing Approach**:
     - Attempt to access other users' data using different user accounts to verify proper isolation
     - Test token reuse after logout to ensure tokens are properly invalidated
     - Try to generate or forge tokens to verify the authentication system's integrity

- Attempt to access API endpoints without proper authentication
- Verify that users cannot view, modify, or delete data from projects they are not members of
- Test that removed project members lose access immediately