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"

> *** 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: To access app features, a user must Sign In using Google Authentication Service first. New users should Sign Up before Signing In. An authenticated user can Sign Out. Users can also remove their account. This authentication requires using Google's external API which will allow users to authenticate in our app with their Google accounts. We will use this to fulfill the "implement an authentication feature using an external authentication service" requirement via Google.

Manage Project: User can create a project and become its owner. When creating a project, the project owner provides the project name. After creating the project, the project owner can view project details such as the project name. The group owner can also delete projects they own. Any user can view a list of active projects they own or have joined. A user can further view detailed information of groups where they are the owner or a member. If a new user joins the group while it is being viewed by another user, the list of members is automatically updated.

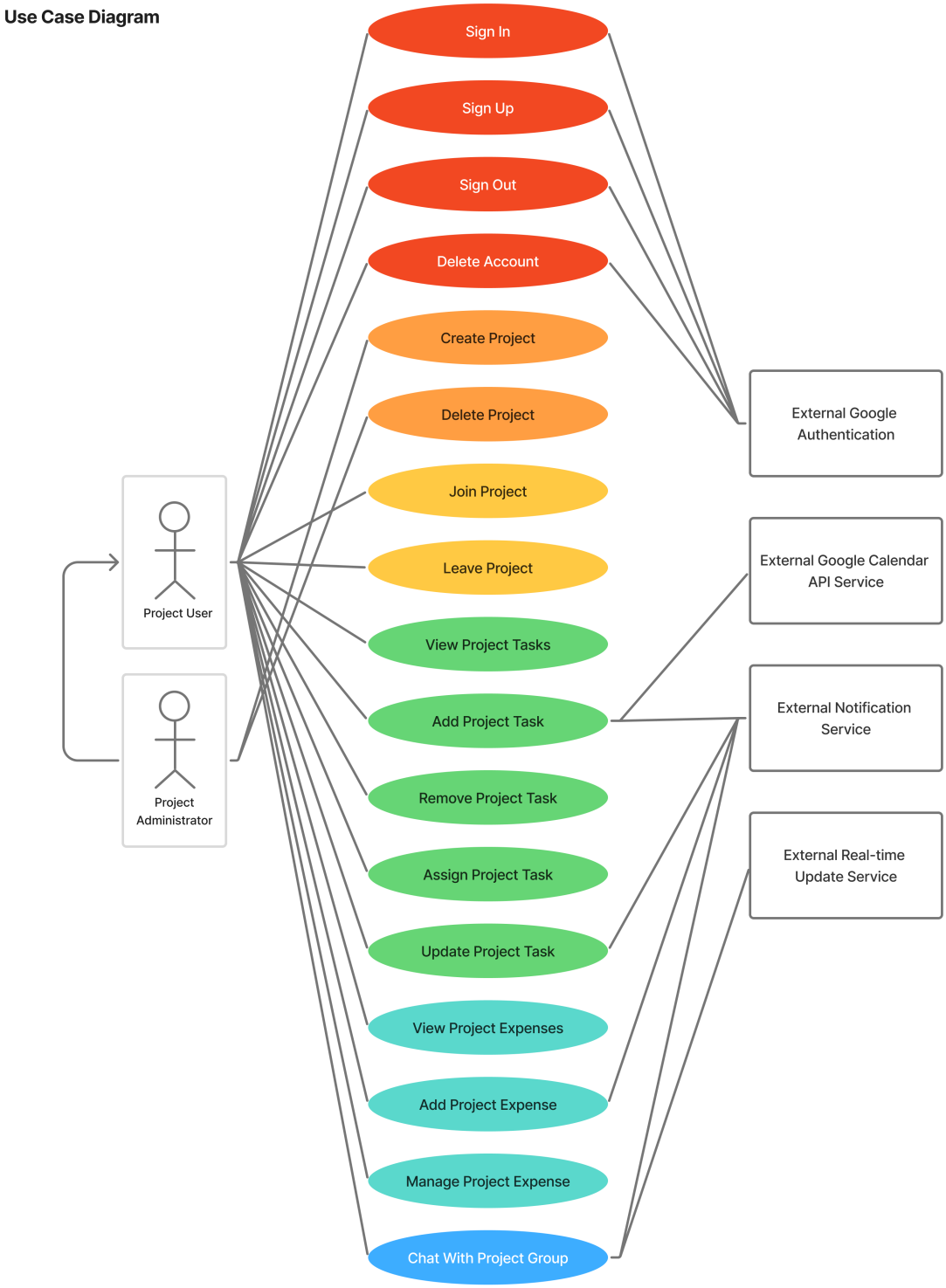
Manage Project Tasks: Users can synchronize tasks with everyone else in the project. Users can add deadlines to tasks and assign them to specific users. Each task can have a status (e.g. "Not started", "In progress", "Completed", "Blocked", or "Backlog") so team members are aware of the progress on each task.

Sync Project Deadlines Across all Members: Users can sync their Google Calendar to tasks so everyone has the same deadlines synced. This will go through Google Calendar's API for syncing, and to also support live notifications. This will fulfill 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. When splitting, users can choose how many people they want to split the expense with. Users can also keep track of current expenses through a visual aid. Users can also add common links/items to a "resources tab" which is a tab that all members can use to access common resources. This feature will satisfy the "at least one of the features must involve computations implemented by your group" requirement, as all of the expense splitting logic will be handled by us, rather than an external API.

Communicate With Members: Enable chatting for projects: Users can use the chat feature for live communications with their teammates in the group. This feature will fulfill 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.

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, can assign "Project User" or "Project Administrator" role 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.

> MVP Implementation Difference: The implementation uses Google OAuth exclusively with a unified authentication flow rather than separate sign-in and sign-up buttons. Both actions use the same Google credential retrieval mechanism, eliminating the distinct "Create a new Google account" button described in the document.

1. **User registers/signs up for the first time**: The user opens the app and clicks sign up to create an account. We will utilize the Google authentication provided in M1, so users will only have the option to create an account through their existing Google account.

2. **Returning user signs in**: The user opens the app and clicks sign in to log into our app with their registered credentials. We will utilize the Google authentication provided in M1, so users will only have the option to log in through their existing Google account.

3. **Returning user signs out**: The user is already logged into our app and clicks the "Sign Out" button to sign out of their account. They are redirected to the sign in/sign up page, where they can follow use case 1 or 2 in this section.

4. **Returning user deletes account**: The user is already logged into our app and clicks the "Delete Account" button to delete their account. They are redirected to the sign in/sign up page, where they can follow use case 1 or 2 in this section. In addition, 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 in our application, a user clicks "Create Project" to create a new project. They then enter project specific details by filling in the "Project Title", before the project is created. After submitting this, a POST endpoint is called to save this project data into the database.

> MVP Implementation Difference: Projects are created immediately with only name and description, and invitation codes must be shared manually. Member roles and invitations are managed after project creation rather than during the creation process.

2. **Joining an existing project**: Users join existing projects (created by other users) through email invitations or project codes.

3. **Access/view projects**: From the home screen in our application, a user views the projects they are currently a part of and clicks 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 click "Delete project" to delete this project.

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.

> MVP Implementation Difference: Task creation uses a single-form dialog. All task details including name, assignee, status, and deadline are entered in one dialog before submission. Tasks display with color-coded status indicators (green for completed, gold for in progress, purple for backlog, red for blocked, blue for not started) for enhanced visual feedback.

1. **Add a new task with a deadline**: Inside a specific project, a user clicks "Add task" to add a new task to the project task screen. They are prompted to add a name, description, assignees, and deadline. By default, the task status will be set to "In progress".

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 dropdown to change the status of a task (in progress, complete, backlog). 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 the tasks.

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

1. **Allow app access to your Google Calendar**: After creating your account, users are prompted to allow the app access to your Google Calendar. You are also able to change this through the Settings tab at a later time.

2. **Sync project tasks assigned to you to your Google Calendar**: Assuming you allow access, any project tasks assigned to you are synced with your Google Calendar through the Google Calendar external API and appear in your Google 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.

> MVP Implementation Difference: Expenses are automatically split equally among selected members rather than allowing custom split amounts. Users select members via checkboxes but cannot specify different amounts per person. The expense view displays a detailed table format showing all expense information including per-person amounts.

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 to add a new project expense. They are prompted to enter the expense name, description, and amount. This expense then appears in all the teammates' apps. The expense added is equally split among all project members. For example, if

a project has 4 members and a project member purchased an item worth $10, each project member will owe $2.50 to the project member that purchased it.

2. **View outstanding balances owed to other teammates**: Inside a specific project, users click on the "Expenses" tab where they view all their outstanding expenses owed to other teammates. They see the amount and teammate name for each expense. This information is shown in a table.

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. It should be noted that 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 with notifications, allowing dynamic communication that updates the app's state based on external events.

> MVP Implementation Difference: The chat system uses WebSocket connections (Socket.IO) for real-time messaging instead of HTTP polling. Messages are delivered instantly through persistent connections. The "sent" indicator replaced with a loading spinner during message transmission.

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".

2. **Receive chat notifications**: If a user is using the app and someone messages a chat in a project that they are part of, they receive a pop-up notification to notify them.

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 tab with a Google sign-in button and a "Create a new Google account" button.
2. User clicks the "Google sign-in" button and is prompted to enter their email and password.
3. User enters this information and clicks "Next".
4. User successfully logs in with Google, and lands on the home page.

> MVP Implementation Difference: Steps 5-7 are not implemented. Projects are created immediately after step 4 without an invite page. Users share the generated invitation code manually, and member management occurs after project creation through the Project Settings tab (available only to admins).

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 send a message
    - 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

> MVP Implementation Difference: Failure scenario 4a is not implemented because direct email invitation functionality has not been integrated. Users join projects using invitation codes only.

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. Other relevant project features should also be accessible (such as the chat and expenses tab, though both would still be empty at this stage).

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

Main success scenario:

1. User opens the app and navigates to the home screen.
2. User clicks on the "Create Project" button.
3. A form appears prompting the user to enter the project name and optional project description.
4. User types in the project name and description in the corresponding text fields and clicks "Submit".
5. User is taken to an invite page, where they can enter the email addresses of other users they want to invite to the project. Added email addresses by default will be a "Project user", but they can be changed to a "Project Administrator" by clicking the role dropdown next to the corresponding email address.
6. Once user is done entering all emails and adjusting roles as necessary, user clicks "Send Invites".
7. User sees a success message confirming the project has been created and that invitations have been sent to the added members.
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, members, and tabs for tasks, chat, and expenses (which are initially empty).

> MVP Implementation Difference: Steps 5-7 are not implemented. Projects are created immediately after step 4 without an invite page. Users share the generated invitation code manually, and member management occurs after project creation through the Project Settings tab (available only to admins).

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 invalid email
    - 4a1. User types an email address that is not valid
    - 4a2. The app shows a warning next to the email address indicating that the email is invalid

    - 4a3. If the user tries to send an invite with invalid emails, they have the option to select "Ignore Invalid Users" and proceed with the valid ones
- 6a. Network error
    - 6a1. If the app cannot connect to the server, it shows an error message letting the user know there is a network issue

> MVP Implementation Difference: Failure scenario 4a is not implemented because direct email invitation functionality has not been integrated. Users join projects using invitation codes only.

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.

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 taps on the "Chat" tab to open the chat room.
4. User types a message in the input field.
5. If the message is empty, the app shows a warning and prevents sending.
6. User clicks "Send". A successfully delivered message will have "sent" indicator underneath the last successfully sent message.
7. All project members see the message immediately and can react or reply.
8. If there is a network problem, the app displays an error indicating it cannot send the message.

> MVP Implementation Difference: Step 6 is implemented differently. Instead of a "sent" indicator underneath messages, the system displays a loading spinner on the send button during message transmission. Messages appear instantly for all users through WebSocket connections rather than HTTP polling.

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 taps the "Expense" tab.
4. User clicks "Add Expense" button.
5. User enters expense information including amount and description.
6. User clicks on "Submit".
7. User gets brought back to the expenses tab.
8. User views the updated expenses tab which includes the new expense that was just added. Other group members can view the updated expenses on the expenses tab too.

> MVP Implementation Difference: Step 5 includes additional fields not mentioned in the description. Users must select who paid for the expense and use checkboxes to select which members to split the expense between. The expense is automatically split equally among selected members rather than allowing custom split amounts. Step 8 displays expenses in a detailed table format showing description, amount, paid by, split between, and per-person amounts.

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
- 5a. Invalid input
  - 5a1. User adds a non-positive amount or text string
  - 5a2. System prompts user to input a valid numeric positive amount
- 6a. Server network error (can't connect to server)
  - 6a1. 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 "Existing Project" button to view this specific project.
3. User enters the project dashboard that features project details (name, members, chat, expenses, etc.).
4. User selects the "Add task" button.
5. User fills in task name and the task description fields, then clicks "Next".
6. User sees boxes on which group members to assign the task to, and what the deadline is.
7. User enters this information and clicks on "Submit", and returns back to the project dashboard page.
8. User clicks on the task board page of the project.

9. User sees the task board added with a new task, with its corresponding name, description, deadline, and assigned users.

> MVP Implementation Difference: Steps 4-7 are implemented as a single dialog form instead of a two-step process. The "Create Task" button is accessible from the main project view, and users enter task name, select a single assignee from a dropdown (not multiple via checkboxes), choose status from a dropdown, and pick a deadline using a date picker all in one dialog before clicking "Create". Step 2 references an "Existing Project" button that doesn't exist in the implementation. Step 9 displays tasks with color-coded status indicators (green for completed, gold for in progress, purple for backlog, red for blocked, blue for not started) which provides enhanced visual feedback beyond the basic display described.

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. Invalid input
  - 7a1. User adds an invalid date
  - 7a2. System prompts user to input a valid future date
- 7b. Server network error (can't connect to server)
  - 7b1. 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**: 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. **Usability**
   - **Description**: Users should be able to find desired information/actions within 3 clicks from the project dashboard. The most frequently used features must be easily accessible and reached with minimal navigation. For example, accessing the UI for creating a task should be in a minimal number of clicks from the dashboard.

- **Justification**: Easy to use and pleasant application features are important in positive user experience.
- **Source(s)**: https://www.nngroup.com/articles/usability-101-introduction-to-usability/, https://www.statista.com/statistics/433871/daily-social-media-usage-worldwide/?srsltid=AfmBOore4jK-WKJfo1M9W6h1PtNfaS07bVAFjEfdrizsx1wlx8lZ8Qio

---

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 (potentially 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

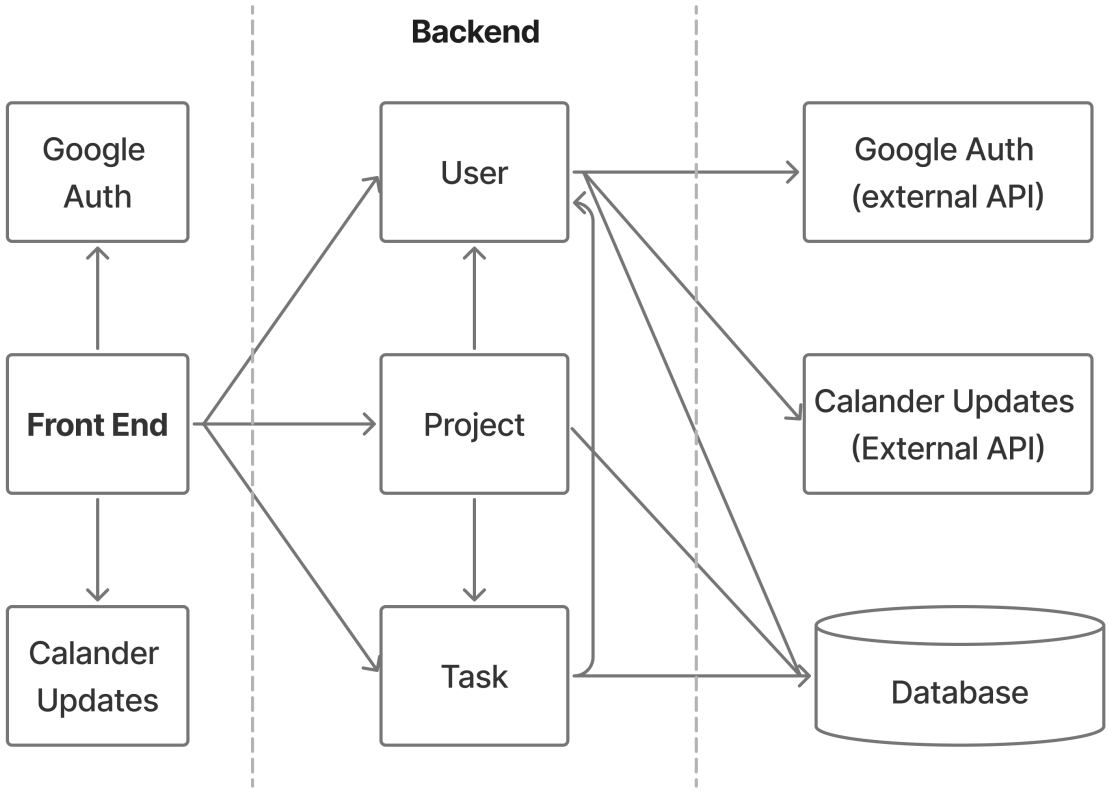
1. **Google Authentication API**
   - **Purpose**: To handle user sign-up and sign-in securely.
2. **Google Calendar API**
   - **Purpose**: To sync project task deadlines to users' personal calendars.

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

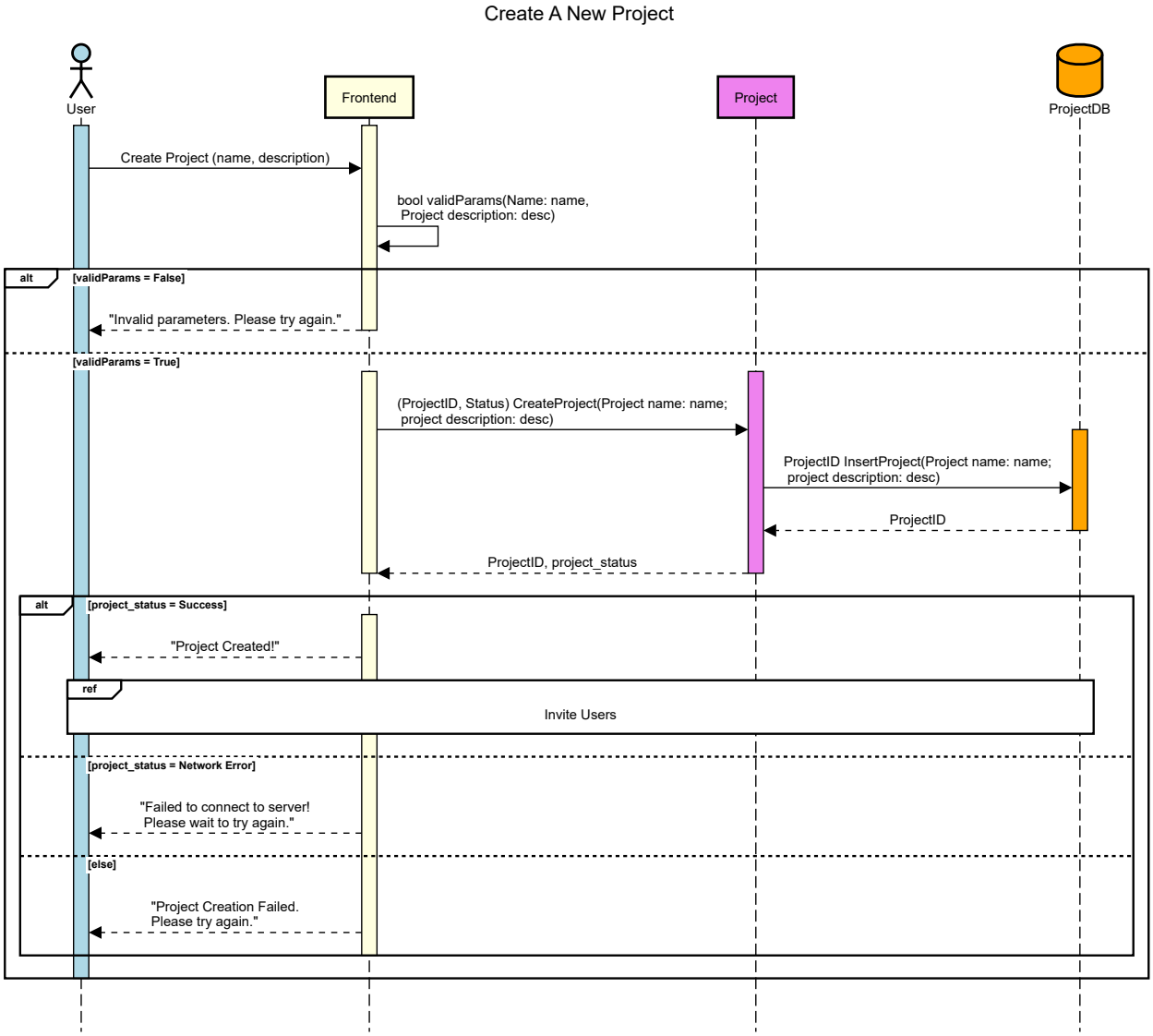
4.5. Dependencies Diagram

Dependencies Diagram



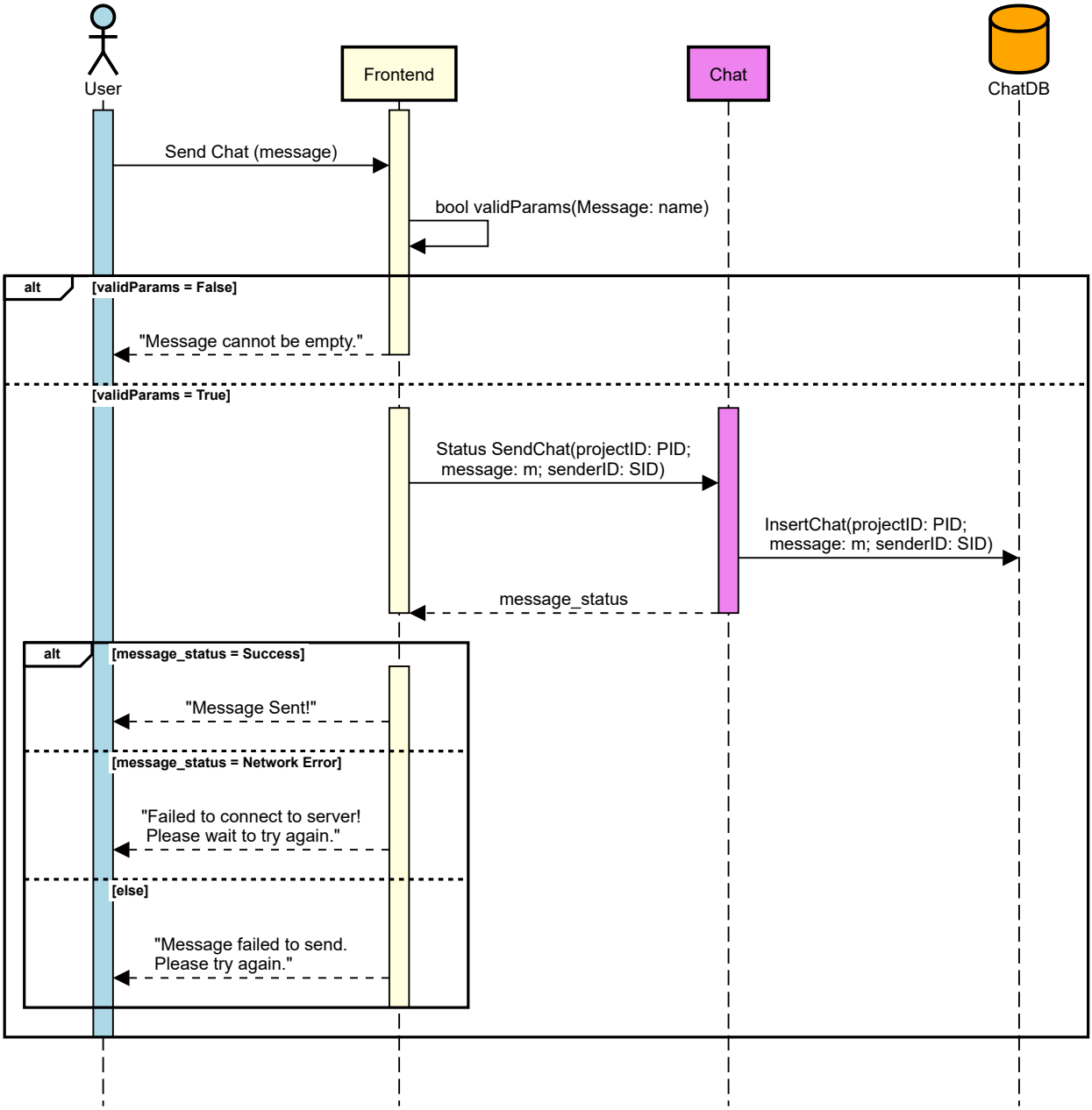
4.6. Use Case Sequence Diagram (5 Most Major Use Cases)

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

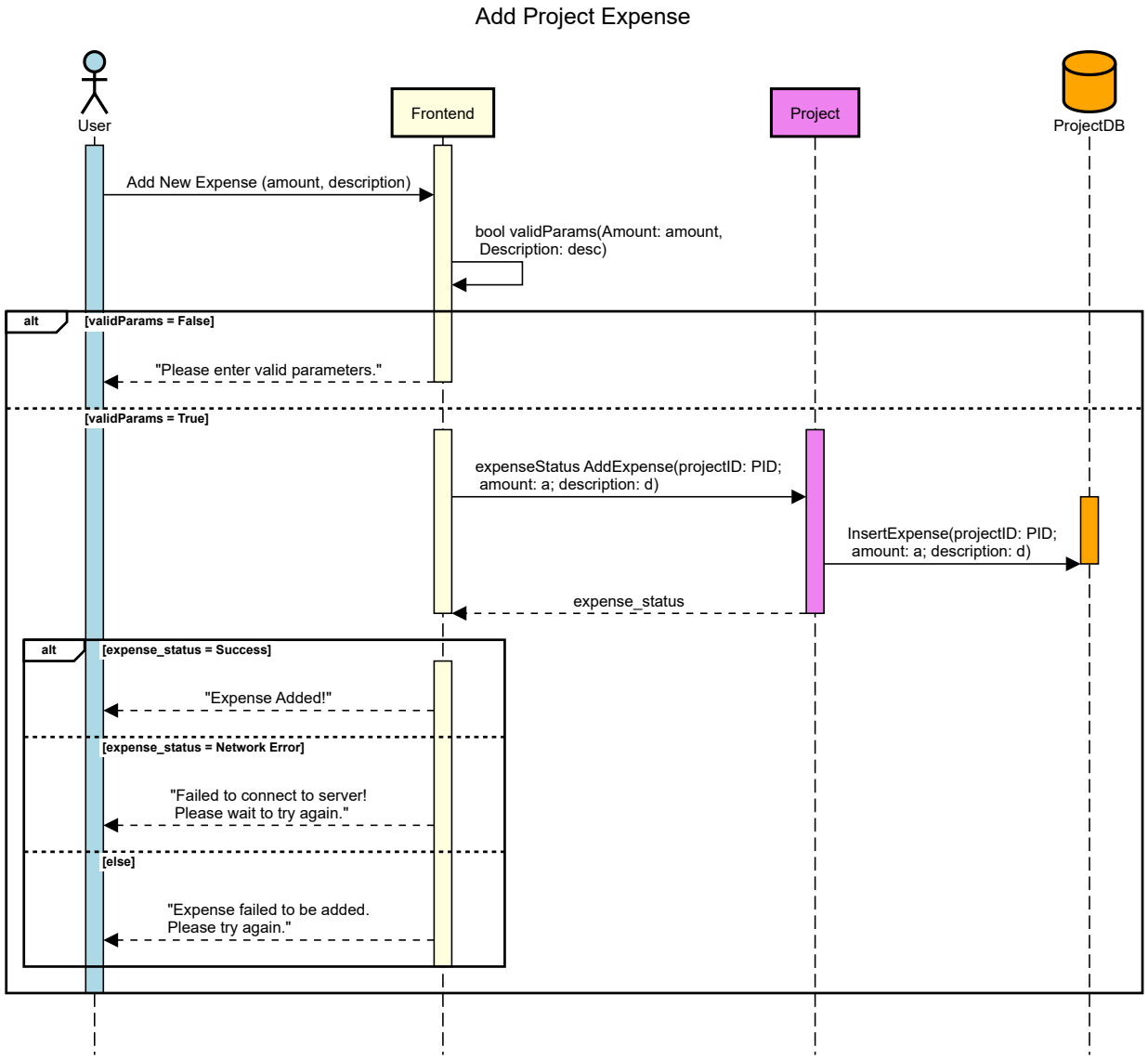


2. [**Communicate/Chat With Group Members In An Existing Project**]

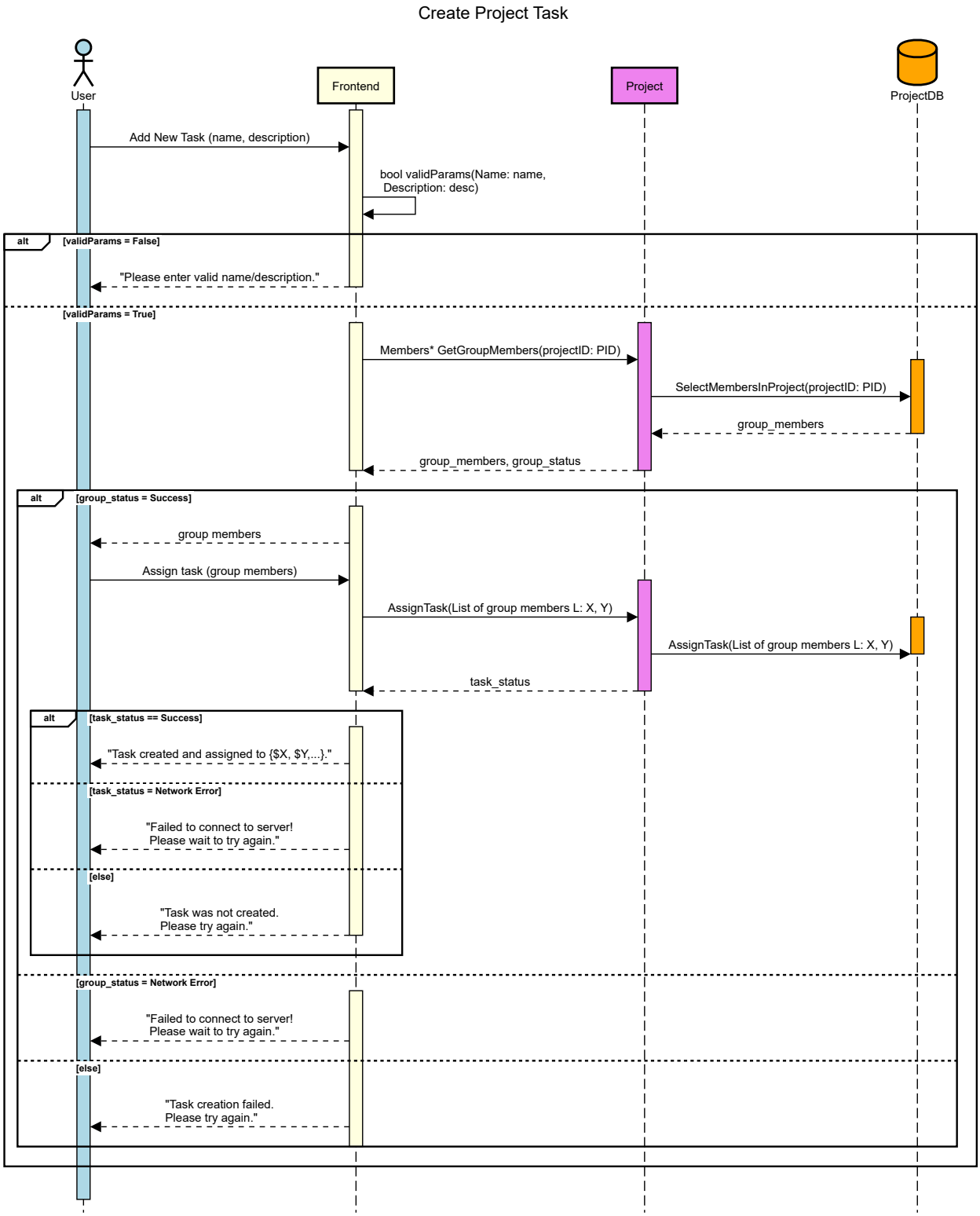
Communicate/chat with group members in an existing project



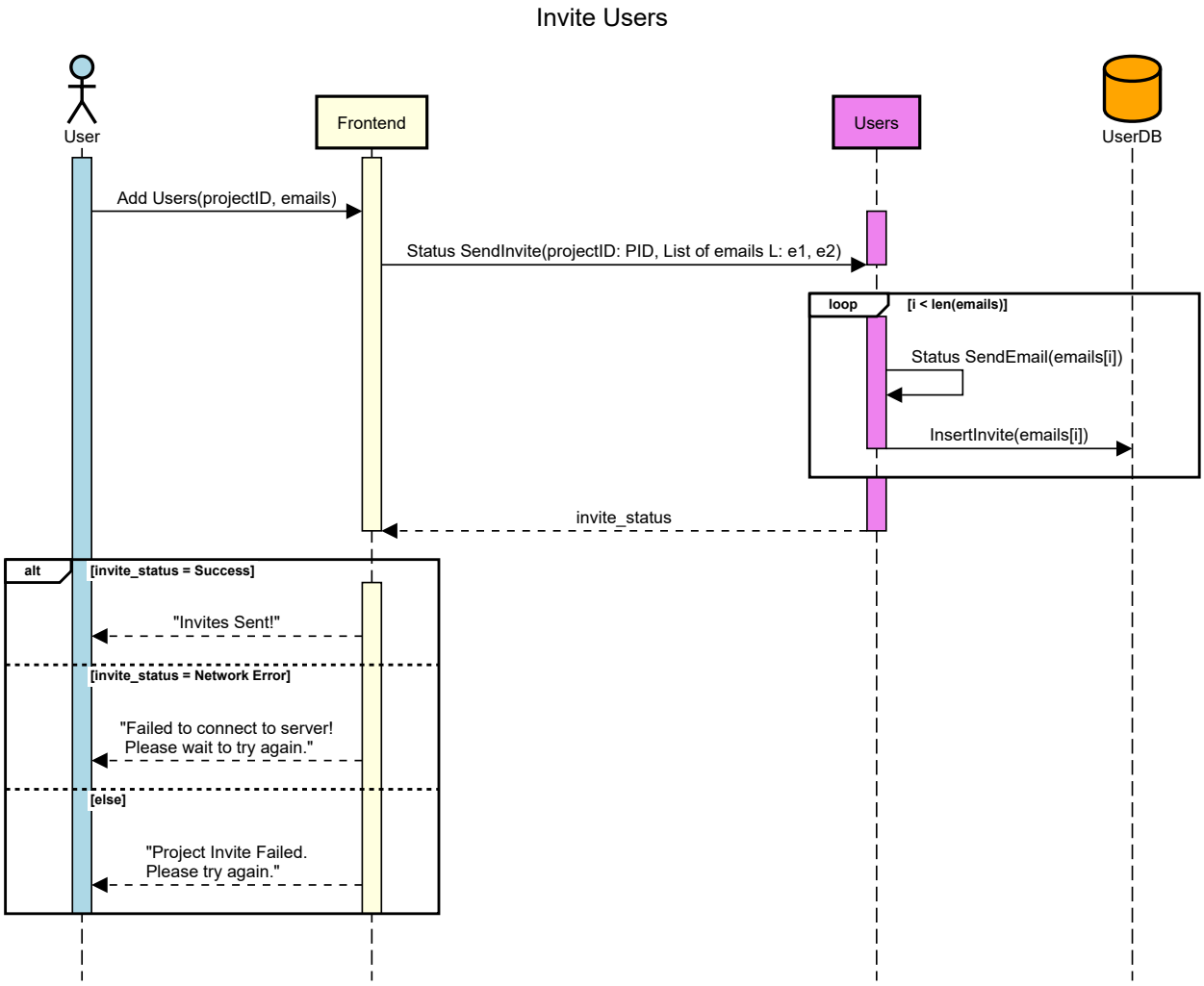
3. [**Add Project Expenses**]



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. Wait 300ms after user stops typing before searching.
   - **Testing Approach**: Time how long it takes to load a project page (should be under 1 second). Have multiple people use the app at the same time and see if it slows down. Check that switching between tabs feels instant.

2. **Usability**

   - **Implementation**: Keep the same menu in the same place on every screen. Put "Create Task" and "Send Message" buttons where they're easy to find. Make buttons big enough to tap easily. Show loading spinners when something is processing.
   - **Testing Approac**h: Ask friends to try creating a task - count how many clicks it takes. See if people can figure out how to add expenses without instructions. Test if everything works using only keyboard navigation. Watch someone use the app for 5 minutes and see where they get confused.