COSC 310 - 101

M3: Formal Analysis and Architecture

Team Accord

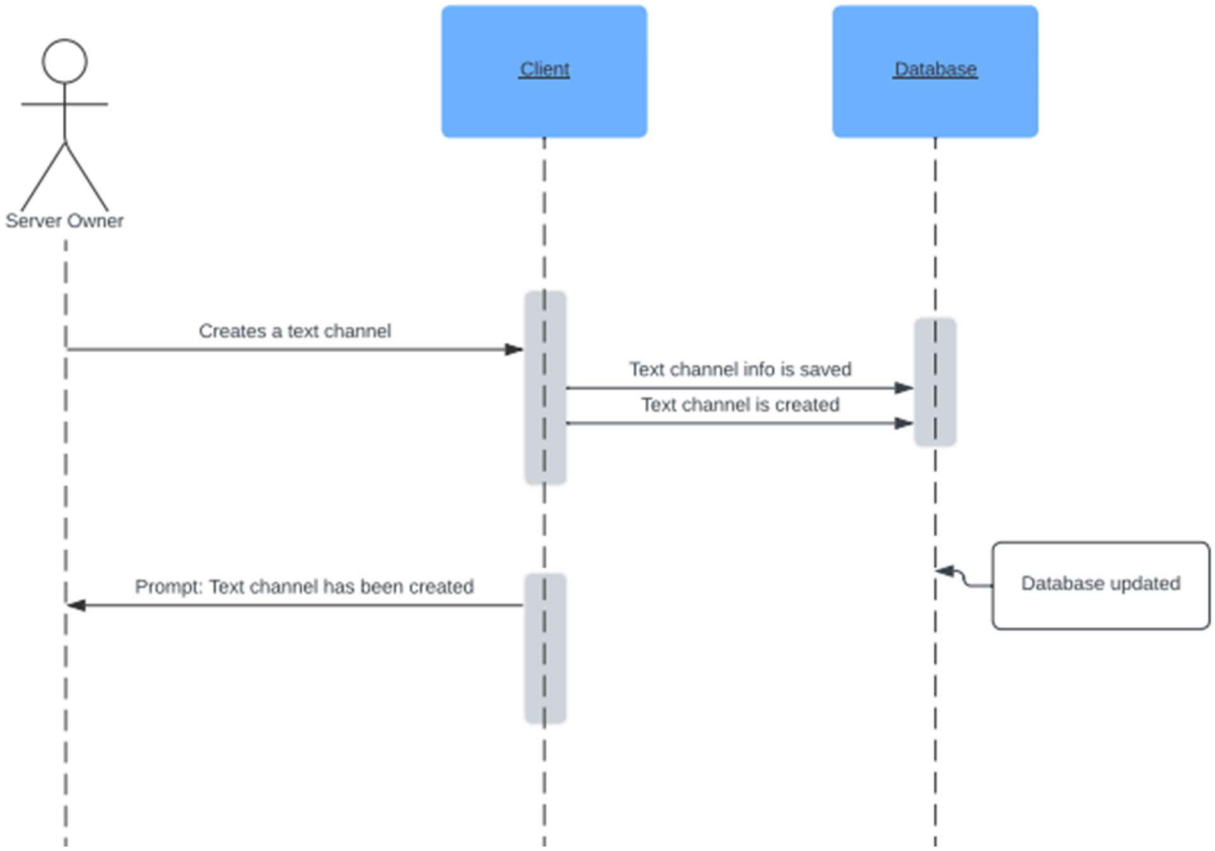
Hoc Nguyen
Colin Lefter
Toby Nguyen
Bao Pham
Immanuel Wiessler

Table of Contents

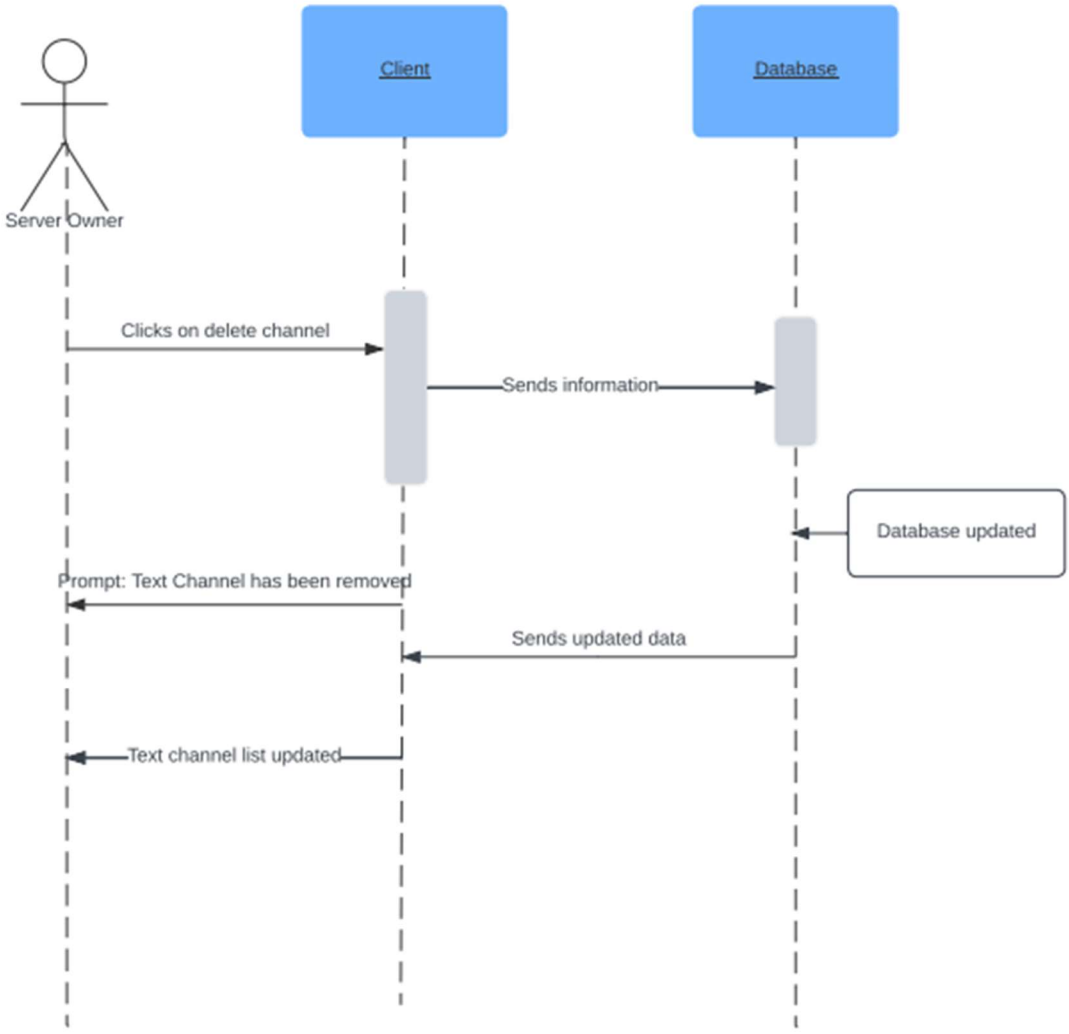
Sequence Diagrams

For Group Chat Owner + Server Owner:

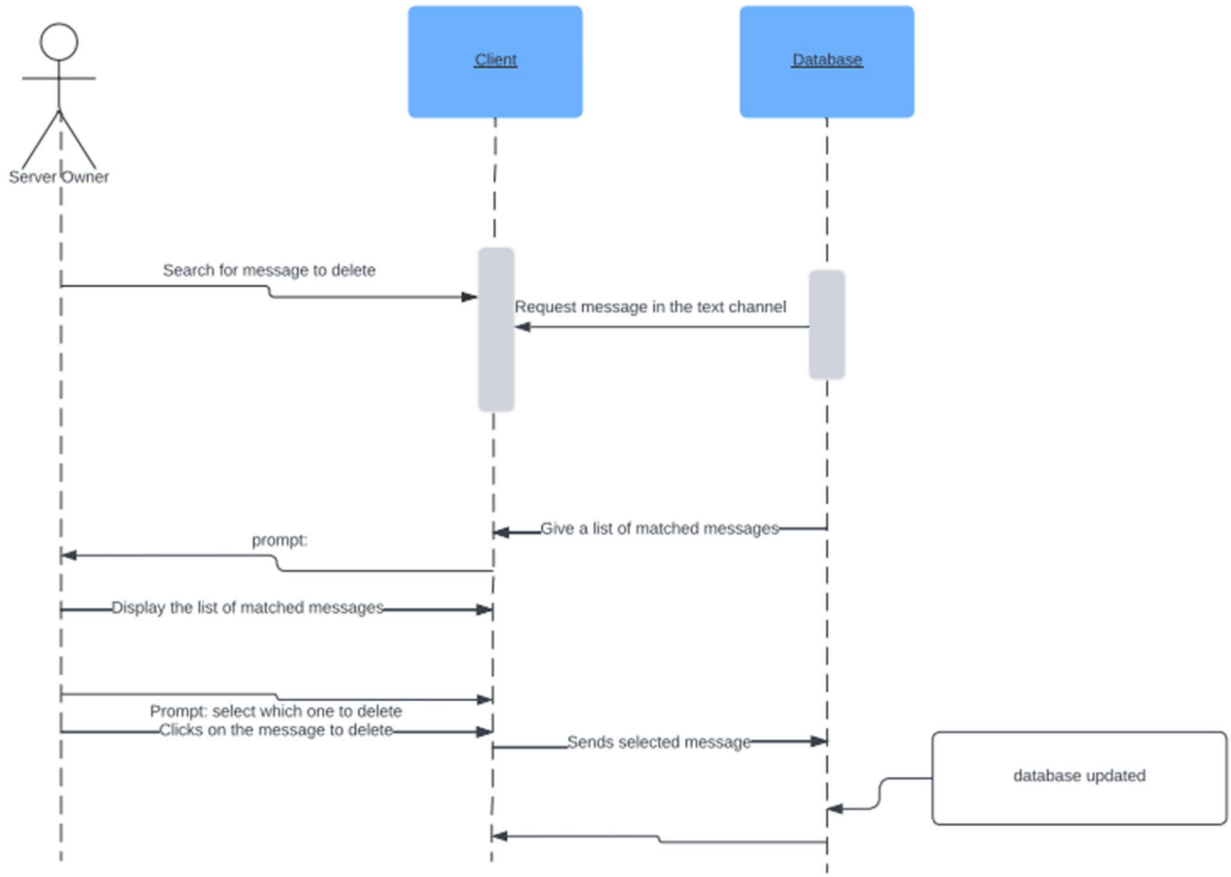
- Creating Text Channel:



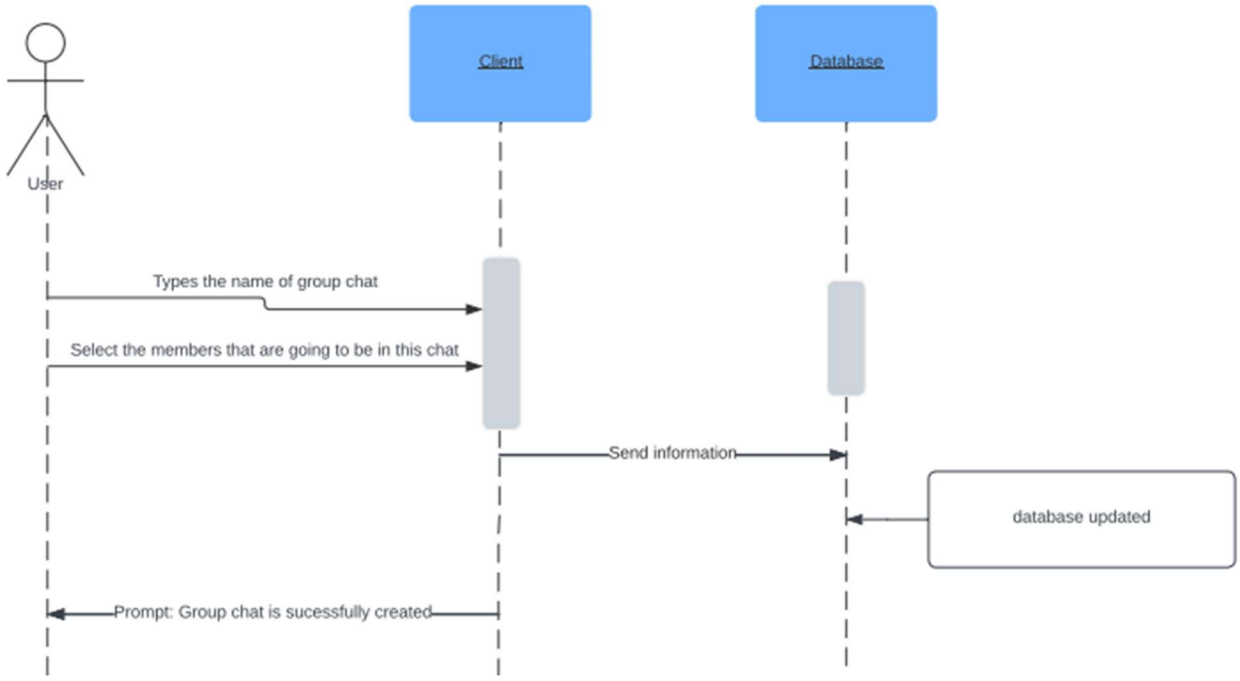
- Deleting Text Channel:



- Deleting Message:

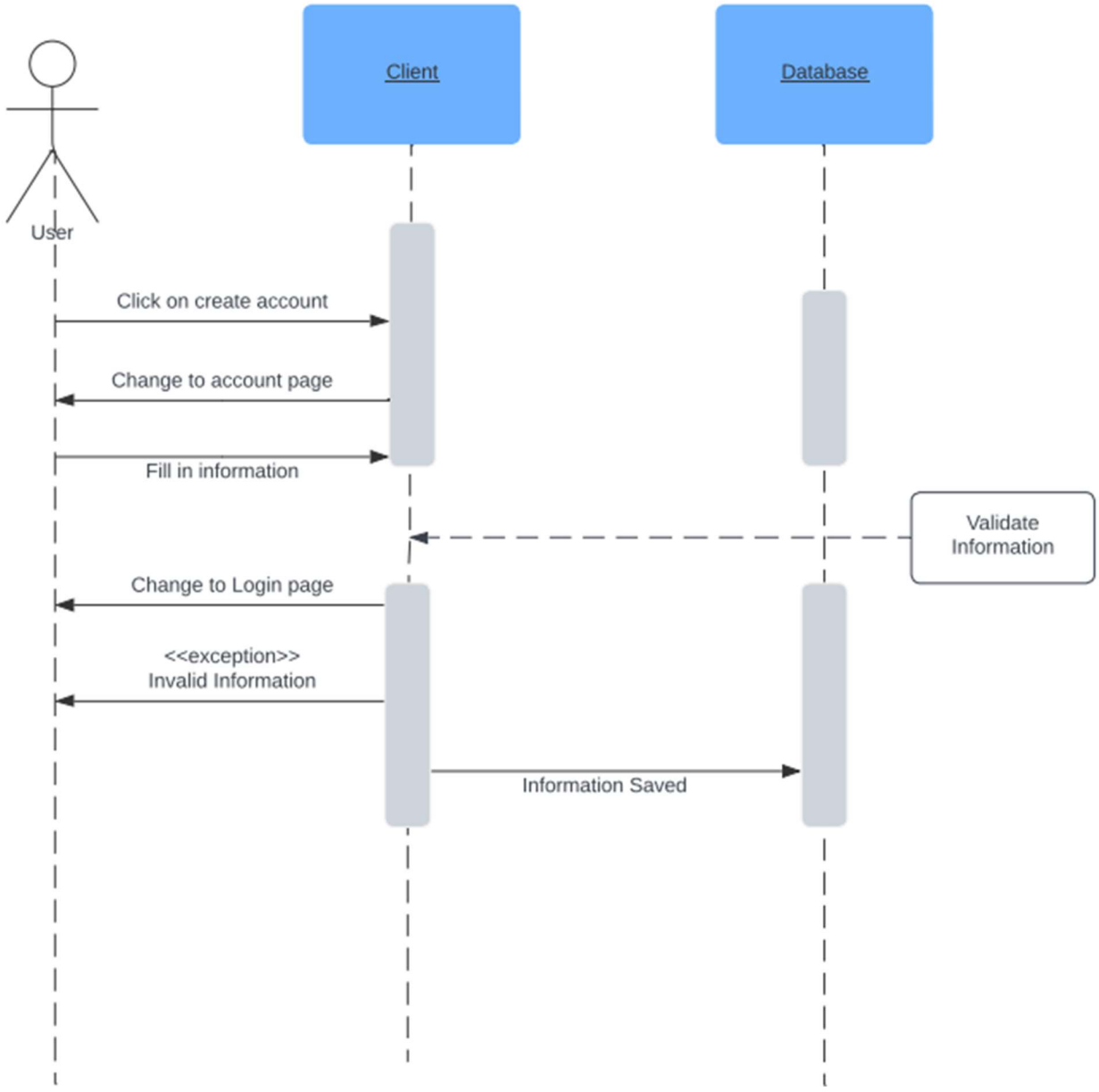


- Creating Group Chat:

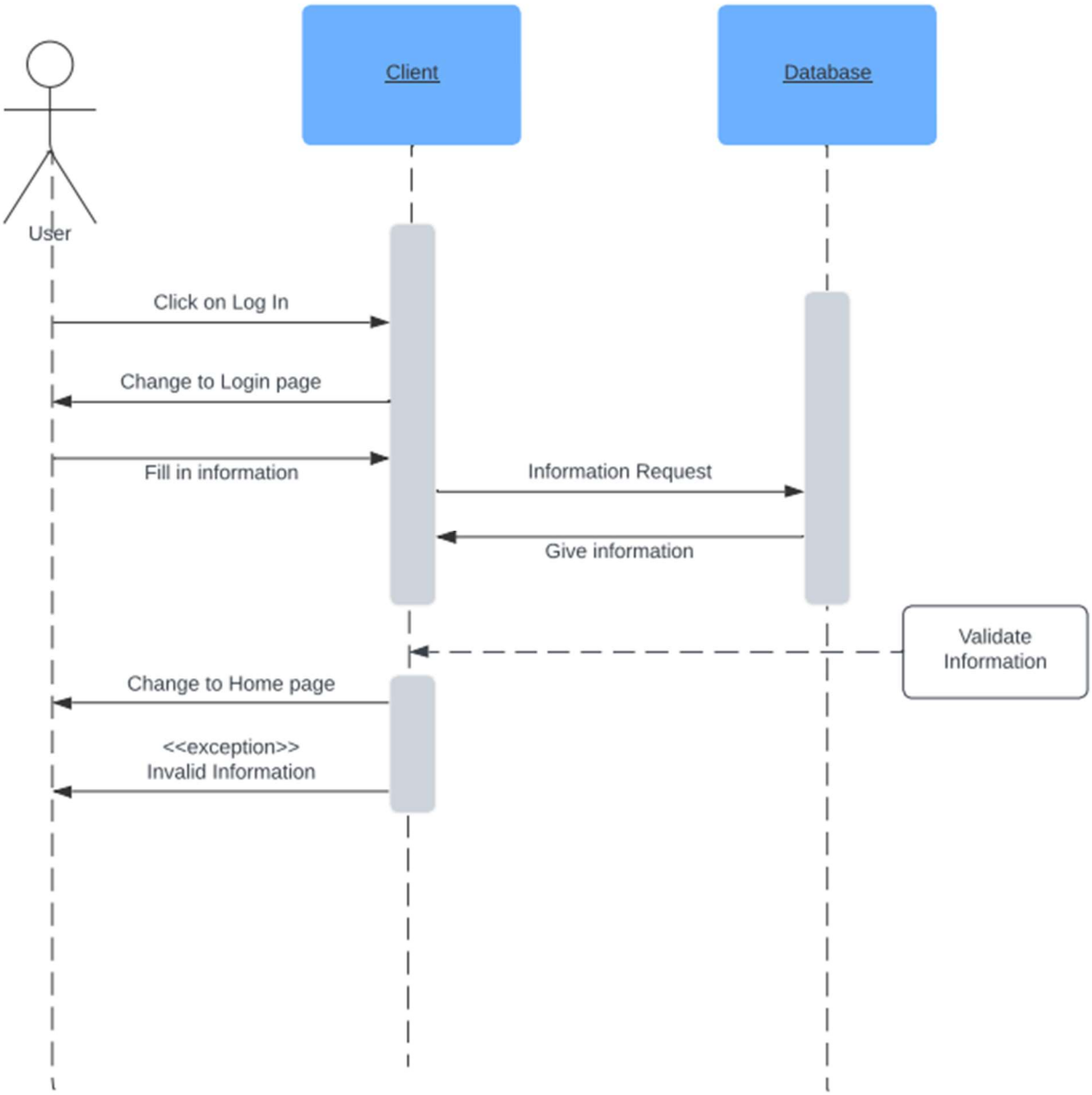


For general users:

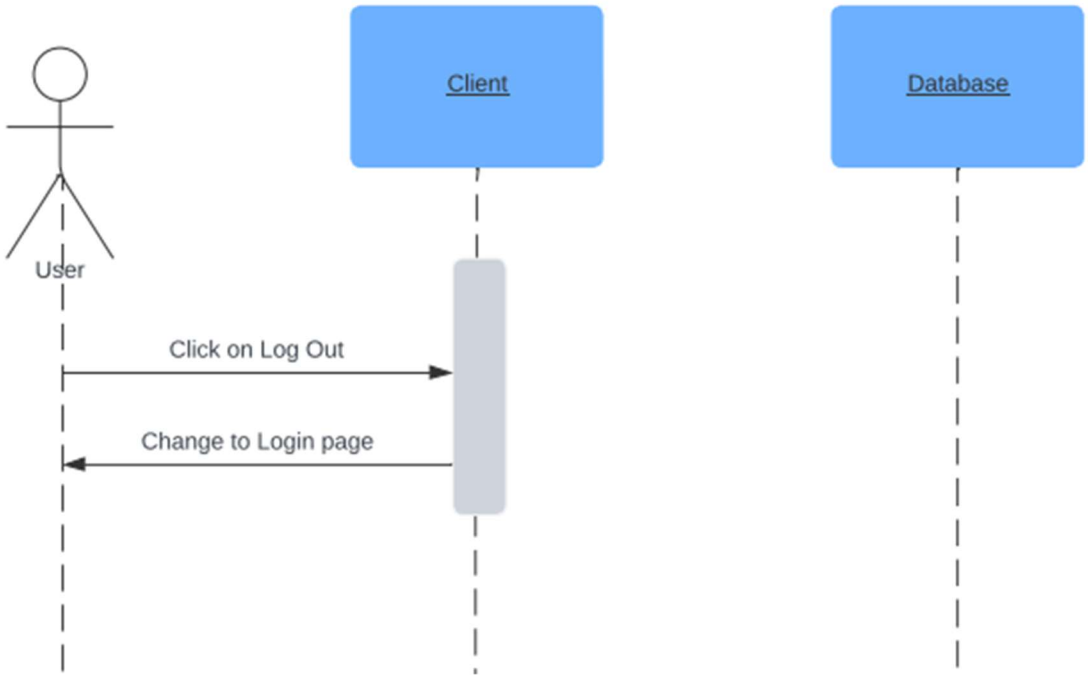
- Creating an Account:



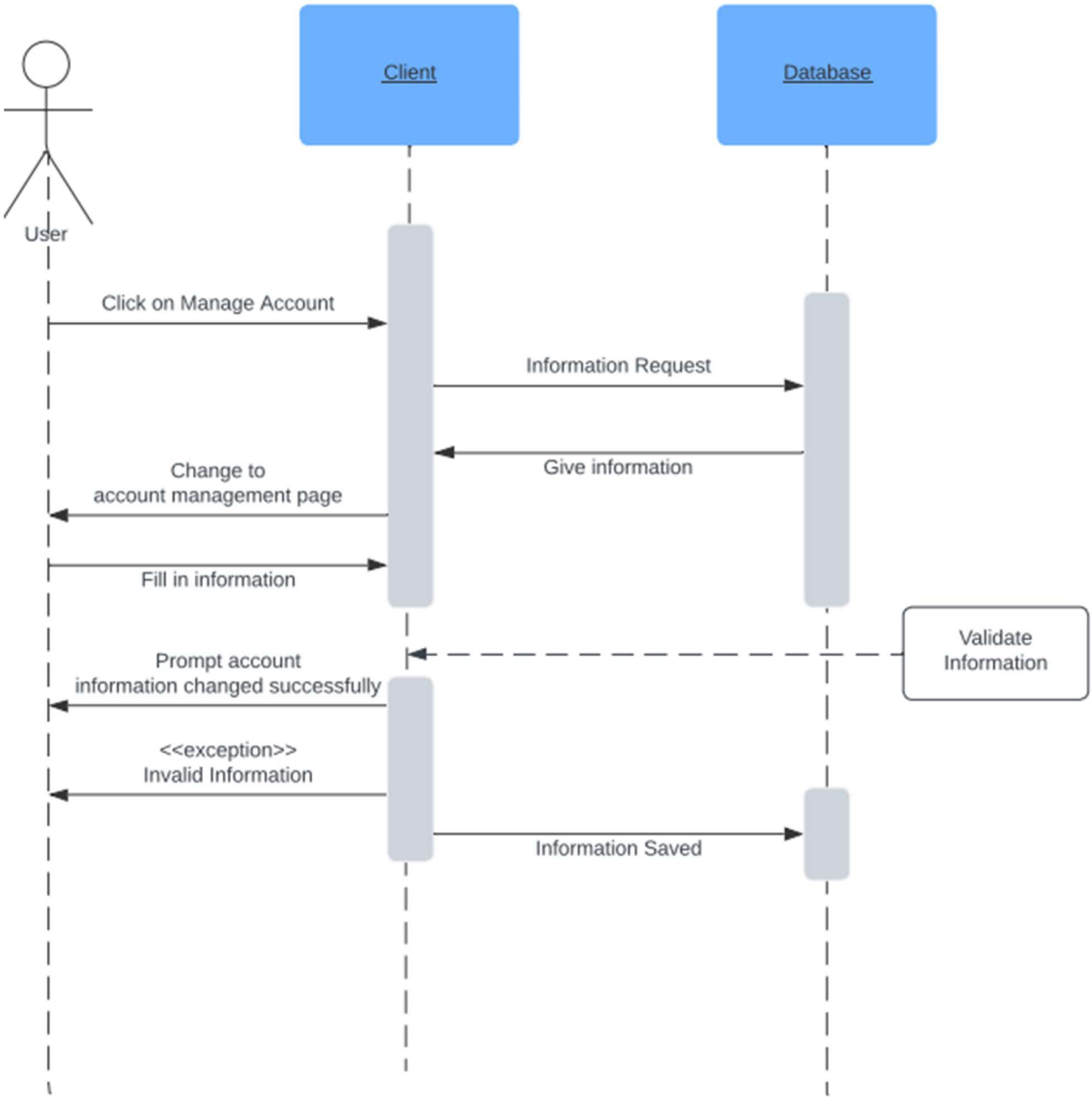
- Log in:


Client
Database
User
Click on Log In
Change to Login page
Fill in information
Information Request
Give information
Validate Information
Change to Home page
<<exception>>
Invalid Information

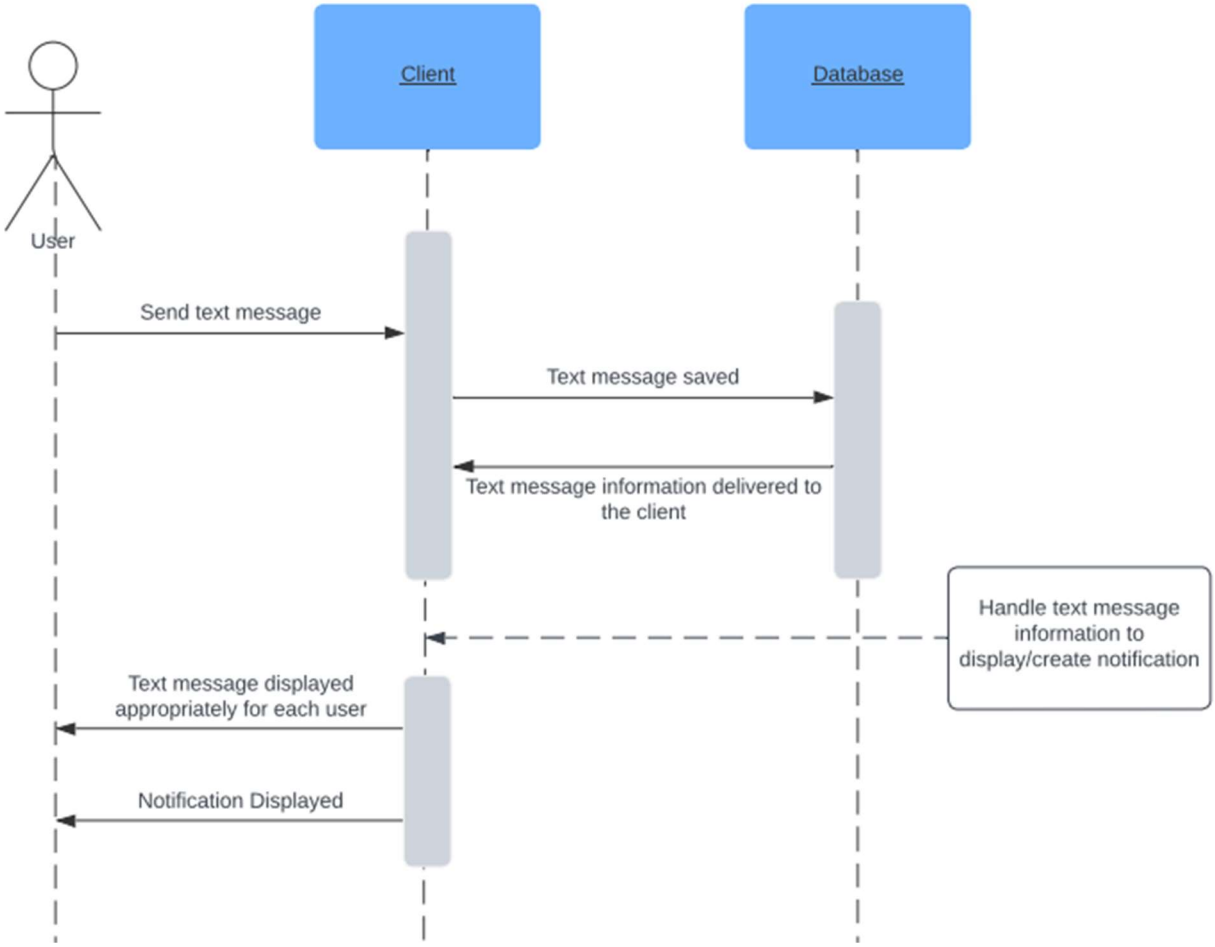
- Log out:



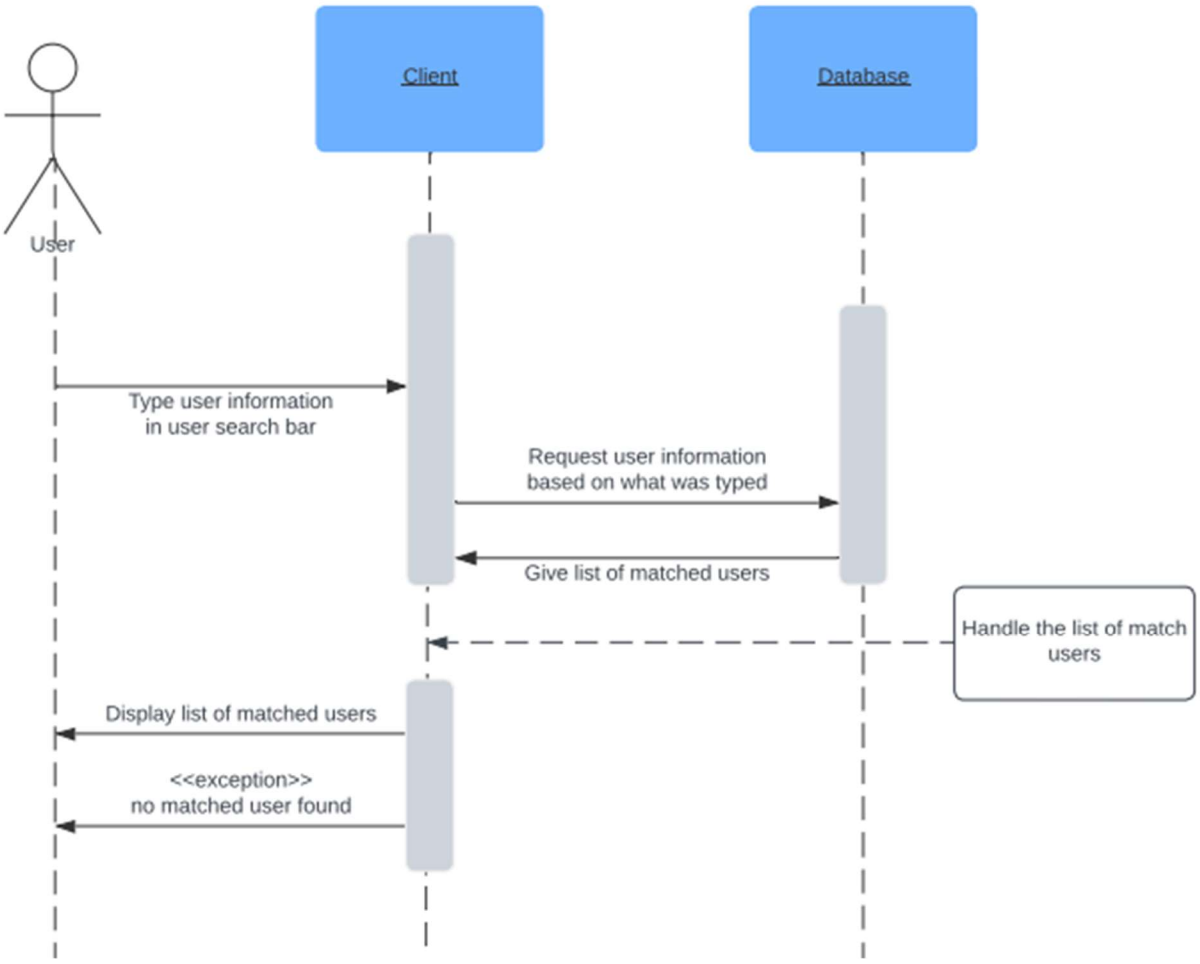
- Manage Account:



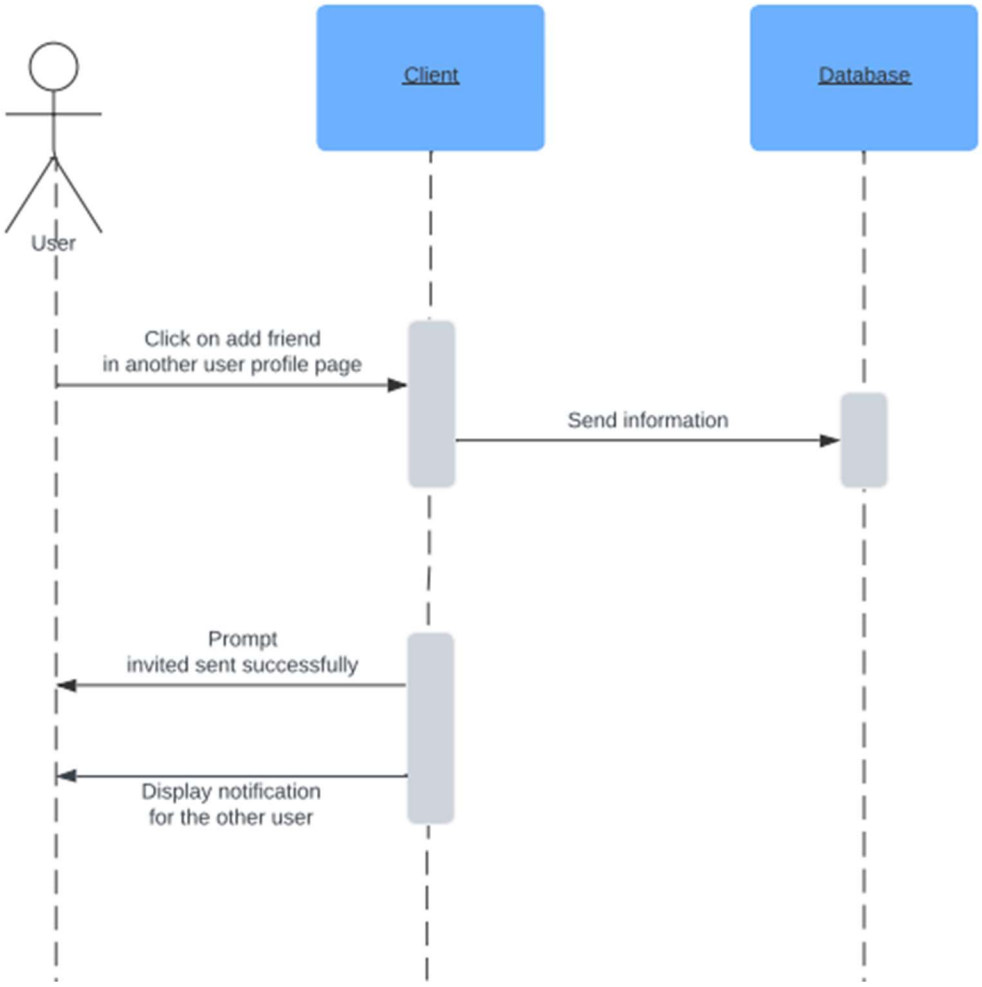
- Send Message:


Client
Database
User
Send text message
Text message saved
Text message information delivered to the client
Handle text message information to display/create notification
Text message displayed appropriately for each user
Notification Displayed

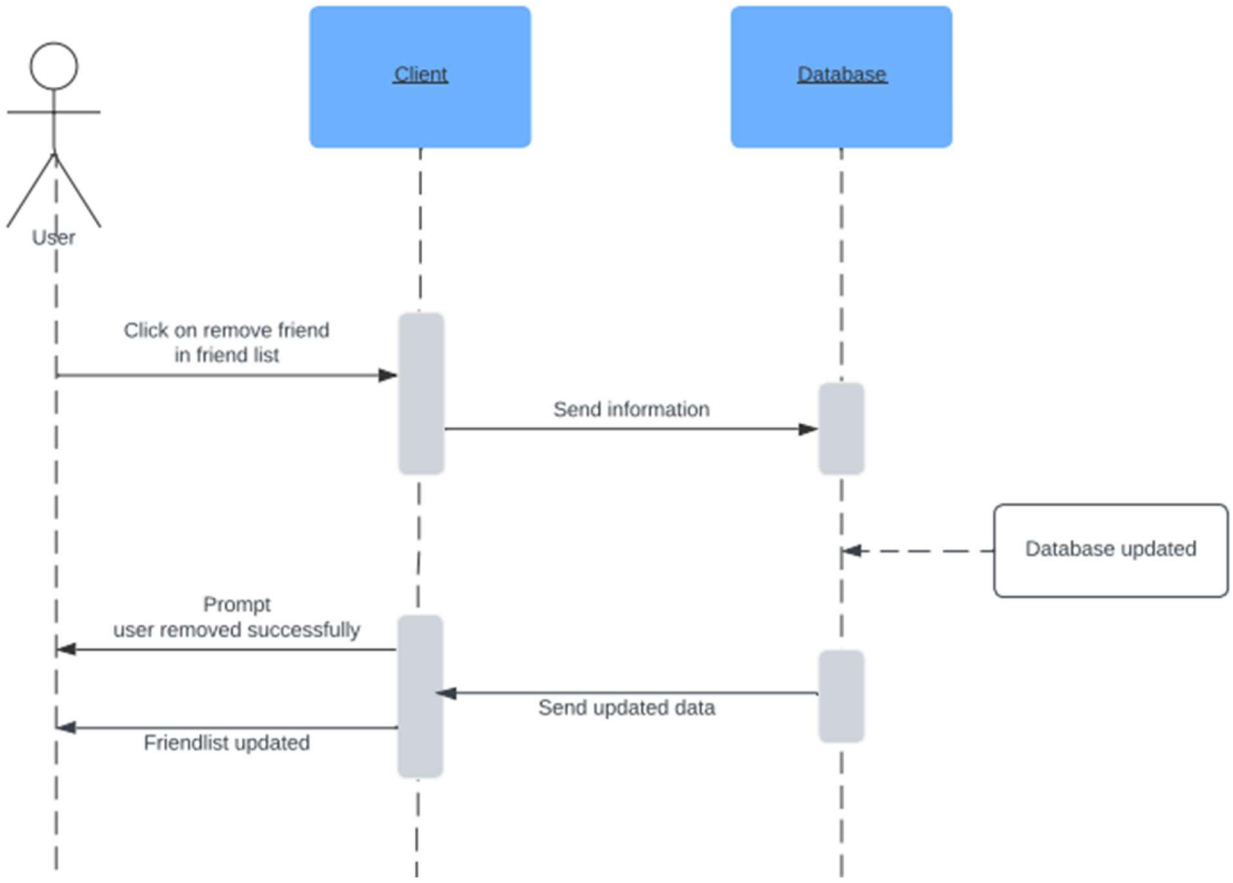
- Search User:



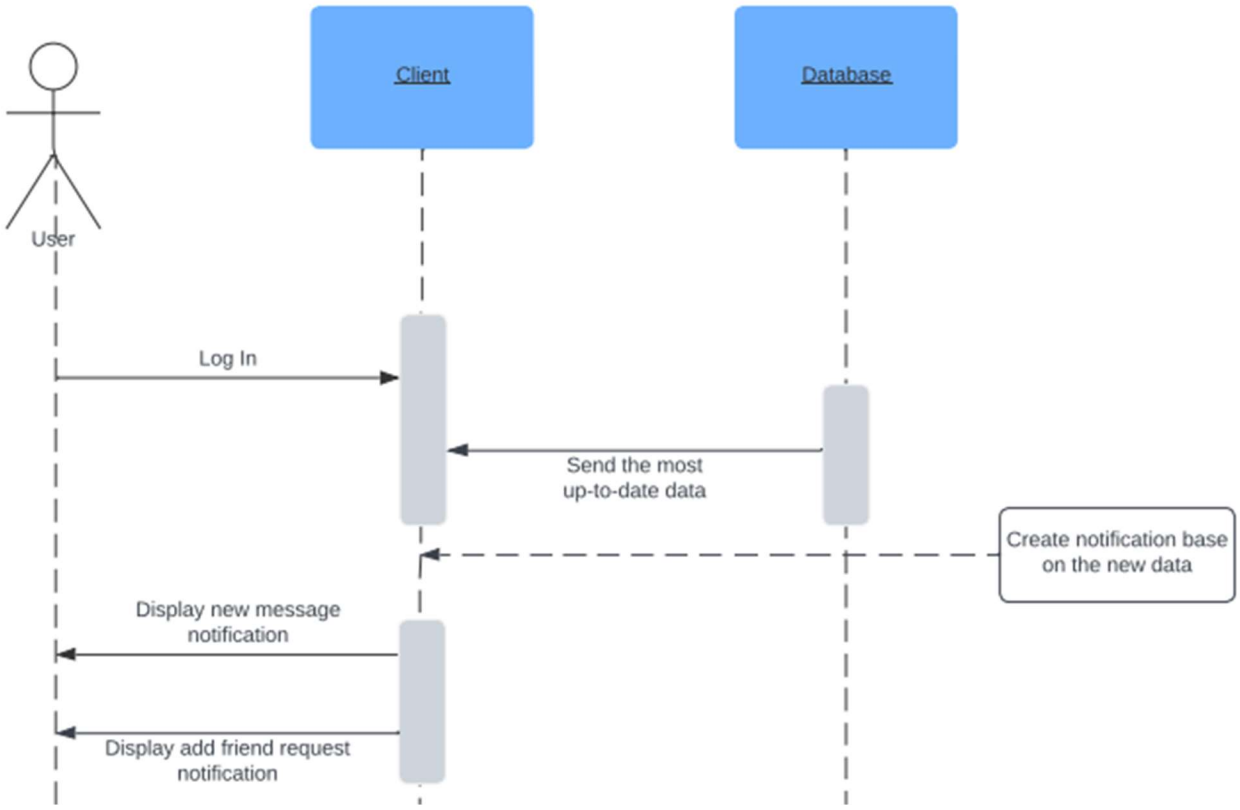
- Add a user as friend:



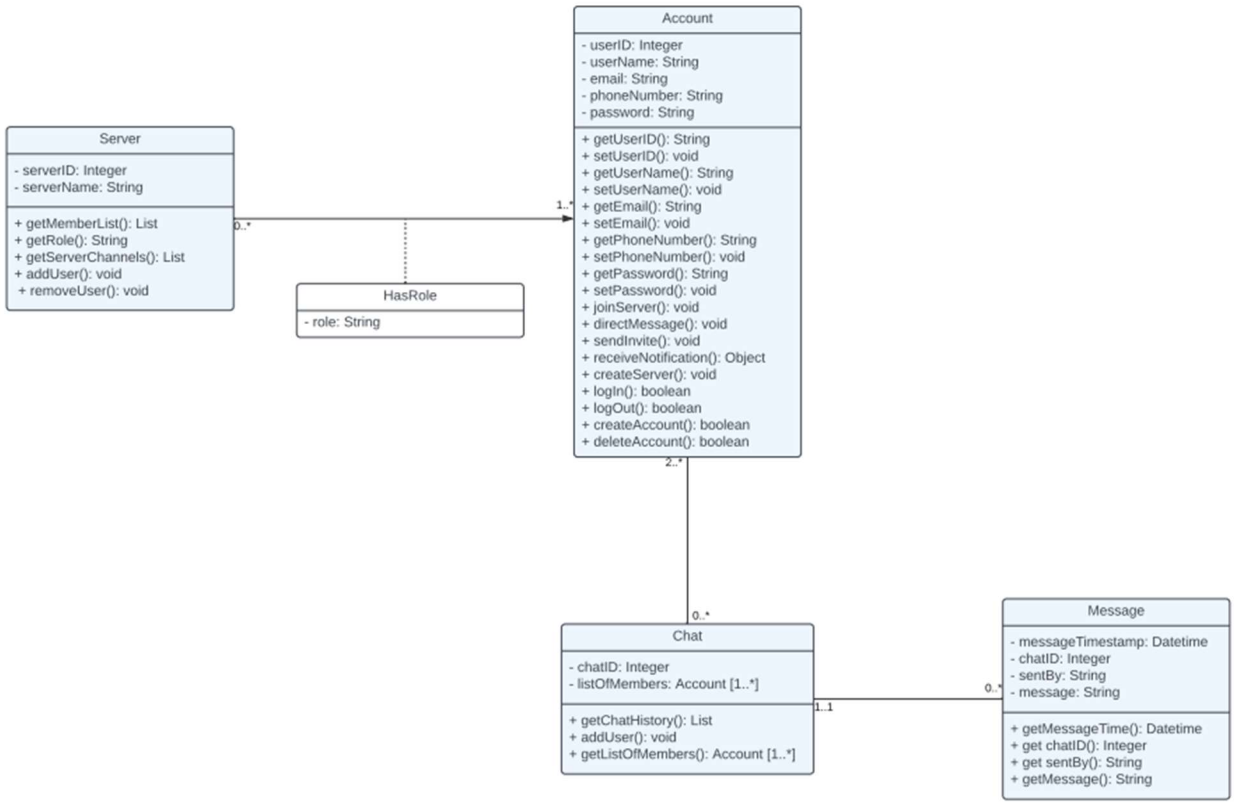
- Remove a user from friend list:



- Receive Notification:



Class Diagram



Testing Plan

Software Development Process and Testing Methodology

- We will follow a Test-Driven Development (TDD) approach whereby initial test cases are written before application components are developed. As our database is central to our feature development, test cases rely on mock user data supplied to our MongoDB Atlas database on the cloud. As we ease into the development of application components, we will proceed with black-box testing as we begin to finalize components.

Software Testing Approaches

- We are using a mix of ad hoc, unit and black-box testing. Our general approach is to first conduct adhoc testing for newly introduced backend processes, such as MongoDB queries. Such tests are discretionary and do not have any accompanying documentation, design or test cases—they are quick and done when necessary to test code as it is written.

- Our subsequent approach is to conduct unit testing for backend processes. These processes follow Test-Driven Development as these tests are written before components are included and only target database queries in isolation. In contrast to ad hoc tests, they are formally documented as test cases. For example, such tests would cover parsing JSON data from MongoDB queries and checking if the outputs are as expected. In addition, this may also involve checking if a query parameter always returns a unique object.
- Our final approach is to conduct black-box testing. Such testing targets specific components and mocks the process of receiving user input from an isolated frontend component followed by the handling of user input by REST API endpoints. As the final level of testing, this approach is particularly rigorous as the test is three-fold:

1. Testing that all form data is captured in State variables

2. Testing that every `onSubmit` function successfully sends an HTTP request to the corresponding API endpoints that handles that request

3. Testing that the API endpoint successfully receives the data from the HTTP request and processes it correctly with the database

Our Current Approaches

Our current approaches are strictly focused on adhoc and unit testing. These are suitable for TDD as all test cases are written before components are integrated.

Our Future Approaches

Once we finalize existing components and begin introducing more components to the application, we will proceed to black-box testing whereby we test the complete pipeline of receiving and processing user input across frontend components and API endpoints.

Test Frameworks

As we are using Next.js as our React framework, we are using Jest as our testing framework. For every component that we build, we will write the equivalent test cases that cover all aspects of its functionality. We also intend to set up GitHub actions to facilitate automatic testing and ensure that all prior test cases pass as new functionality is added. Jest provides a variety of key features including automated testing, built-in assertions, mocking, code coverage and asynchronous testing that makes it highly beneficial for our project.

Test Workflow

All tests are written under the `__tests__` directory of our `frontend` folder. Each test follows the convention of `testName.test.tsx` and should only test one feature or component. A group of tests can be compiled under each component test but they need to be isolated to that particular feature.

Specific Details

To run all tests, execute `npm run test:jest` while in the `application/frontend` directory.

Design patterns

Observer Pattern (Behavioural)

- **Why we chose it**: For our particular project (Discord clone), it is crucial to have real-time updates and notifications whenever a user receives a message. This pattern will establish the 1-to-many dependency between objects that is necessary for updates to be reflected in real-time.
- **How we will implement it**: We first create a **Subject** object that is in charge of sending the correct notifications to users. Then we create an **Observer List** object that will be the parent to all the other observer children (such as **group chat observer**, **server chat observer**, and **direct-message chat observer**). Whenever a message is received from any of these sources, it is immediately observed and updates are reflected through in-app notifications.

Singleton Pattern (Creational)

- **Why we chose it**: For our particular project (Discord clone), it is crucial to have a **Manager** class to handle all project functionality, including user authentication (log in, account creation), server connections and site-wide user information.
- **How we will implement it**: We create a singleton class called **MessagingHandler** that will manage all of the above mentioned functionality (user authentication, Web Socket connection, etc.).