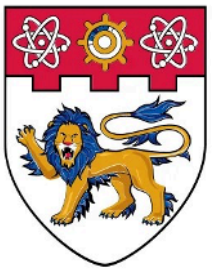


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006 – Software Engineering

Lab 3 Deliverables

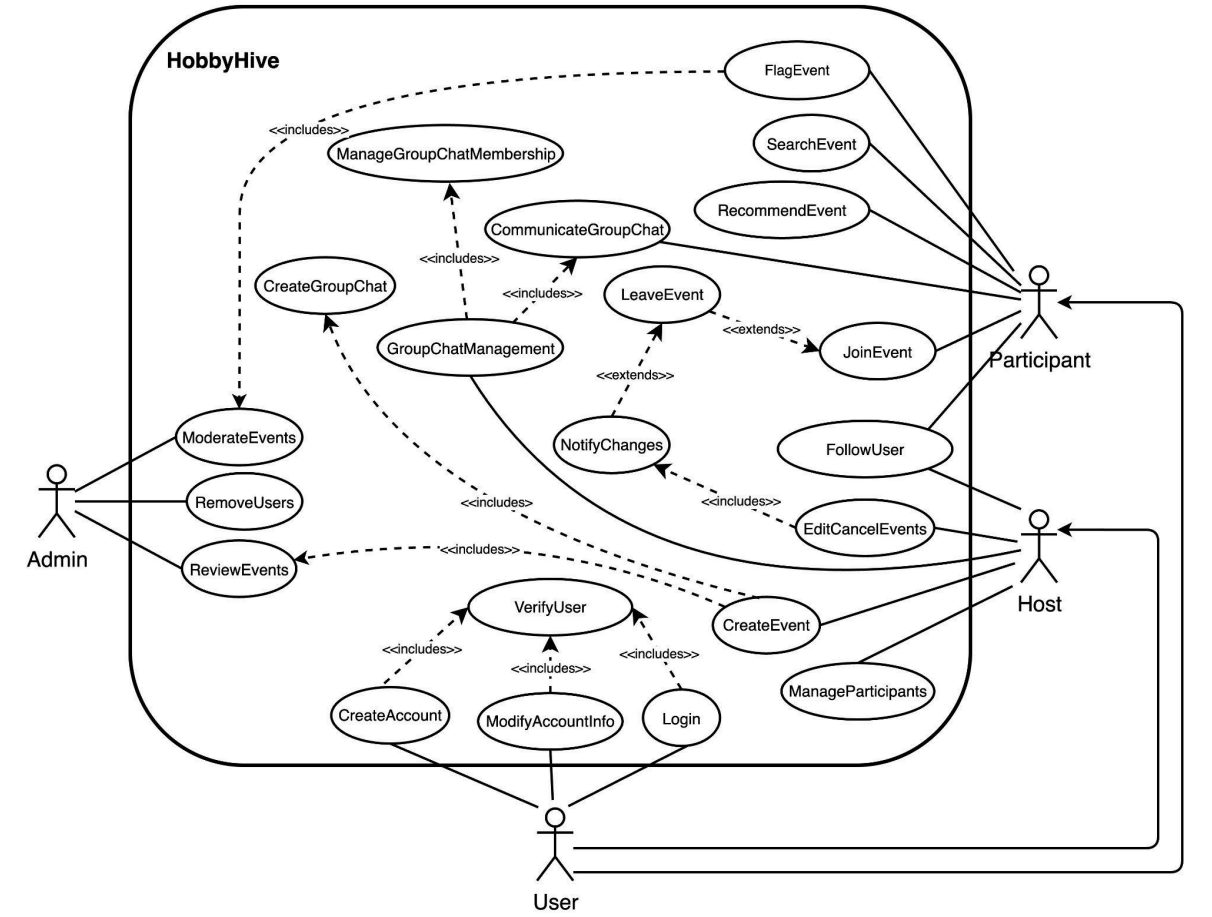
Lab Group	SCSC
Team	HobbyHive
Members	Teo Rong Xuan (U2421554F)
	Jain Divisha (U2423825L)
	Chua Jing Yi Jax (U2421642A)
	Ishita Dhananjaya (U2420909L)
	Palagiri Afreen Mahtaj (U2422632F)
	Swetha Sudhakar (U2420696G)

Table Of Contents

1. Complete Use Case Model..... 3
2. Design Model..... 4
A. Class Diagram..... 4
Views (User Interfaces)..... 4
Controllers (MVC Controller / Application Logic Layer)..... 5
Core Entities..... 5
Data Access Layer (Persistent Data)..... 6
B. Sequence Diagrams..... 7
I. For Use Cases under #1 (Login)..... 7
I.I CreateAccount..... 7
I.II Login..... 8
II. For Use Cases under #2 (Participant)..... 9
II.I SearchEvent..... 9
II.II JoinEvent..... 10
II.III RecommendEvent..... 11
II.IV FollowUser..... 12
II.V LeaveEvent..... 13
II.VI FlagEvent..... 14
III. For Use Cases under #3 (Host)..... 15
III.I CreateEvent..... 15
III.II EditCancelEvent..... 16
III.III ManageParticipants..... 17
III.IV NotifyChanges..... 18
IV. For Use Cases under #4 (Admin)..... 19
IV.I RemoveUsers..... 19
IV.II ReviewEvents..... 20
IV.III ModerateEvents..... 21
V. For Use Cases under #5 (Group Chats)..... 22
V.I GroupChatManagement..... 22
V.II ManageGroupChatMembership..... 23
C. Dialog Map Diagram..... 24
3. System Architecture..... 25
Presentation layer..... 25
App Logic Layer..... 26
Object Layer..... 26
Persistent data..... 27
4. Application Skeleton..... 28
Frontend (With embedded backend logic)..... 28
Backend..... 28
External Backend (Supabase)..... 29

1. Complete Use Case Model

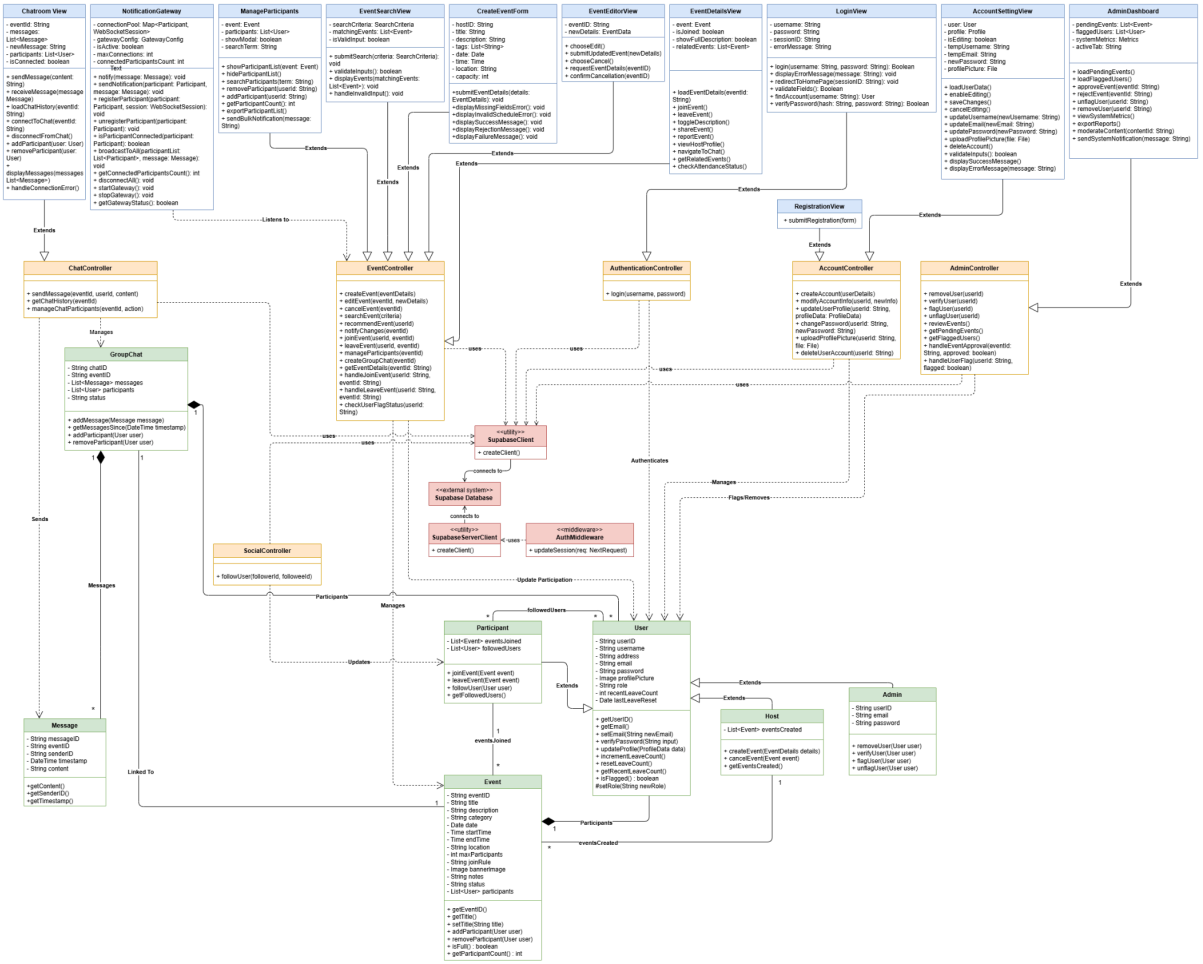
If the image is unclear, please refer to the raw png file that is uploaded together with this document.



2. Design Model

A. Class Diagram

If the image is unclear, please refer to the raw png file that is uploaded together with this document.



Views (User Interfaces)

- **LoginView:** Provides an interface for users and admins to authenticate using their email and password.
- **RegistrationView:** Allows new guests to create an account by submitting their desired registration information.
- **AccountSettingView:** Enables users to update their personal details, including username, password, and profile picture.
- **ChatRoomView:** Displays live group chat messages related to an event and allows message sending
- **NotificationGateway:** Entry point for users to receive notifications on new events, messages, and updates.

- **ManageParticipants:** Allows the event host to view or remove attending event participants.
- **EventSearchView:** Participants can filter down to any available events that can be attended
- **EventDetailsView:** Shows event information, including description, time, list of participants, and chat access.
- **EventEditorView:** Allows the host to edit and update the event's title, date, time, location, and capacity, or cancel the event entirely.
- **CreateEventForm:** Allows users to create an event by inputting a title, date, time, location, and category. The event created is then sent to admins to await approval
- **AdminDashboard:** Main interface for the admin to conduct event approvals, user restrictions, etc.

Each view directly delegates user-triggered logic to the appropriate controller.

Controllers (MVC Controller / Application Logic Layer)

- **AuthenticationController:** Validates and verifies login credentials to allow users to access the platform.
- **AccountController:** Handles updates to user profile details and processes registration of new accounts.
- **AdminController:** Provides elevated system controls, enabling administrators to monitor events, remove abusive users.
- **EventController:** Manages all event-related functionalities, including creating new events, retrieving event details for display, updating event information, and handling participant joining and leaving events.
- **ChatController:** Enables real-time communication by sending and retrieving Message objects within an event's associated group chat.
- **SocialController:** Manages social interactions such as following and unfollowing Users to enhance community engagement.

Core Entities

- **User:** Abstract class representing all system users, storing login and profile data.
- **Host *(inherit from User)*:** User with the capabilities to create and manage their own events.
- **Participant *(inherit from User)*:** Users who are able to join events.
- **Admin *(inherit from User)*:** Users with administrative capabilities, responsible for moderation and control over events posted on the platform and user activities.

- **GroupChat:** Real-time channel linked to each event, storing all associated messages
- **Message:** Represents a single chat message, including sender, timestamp, and text content.

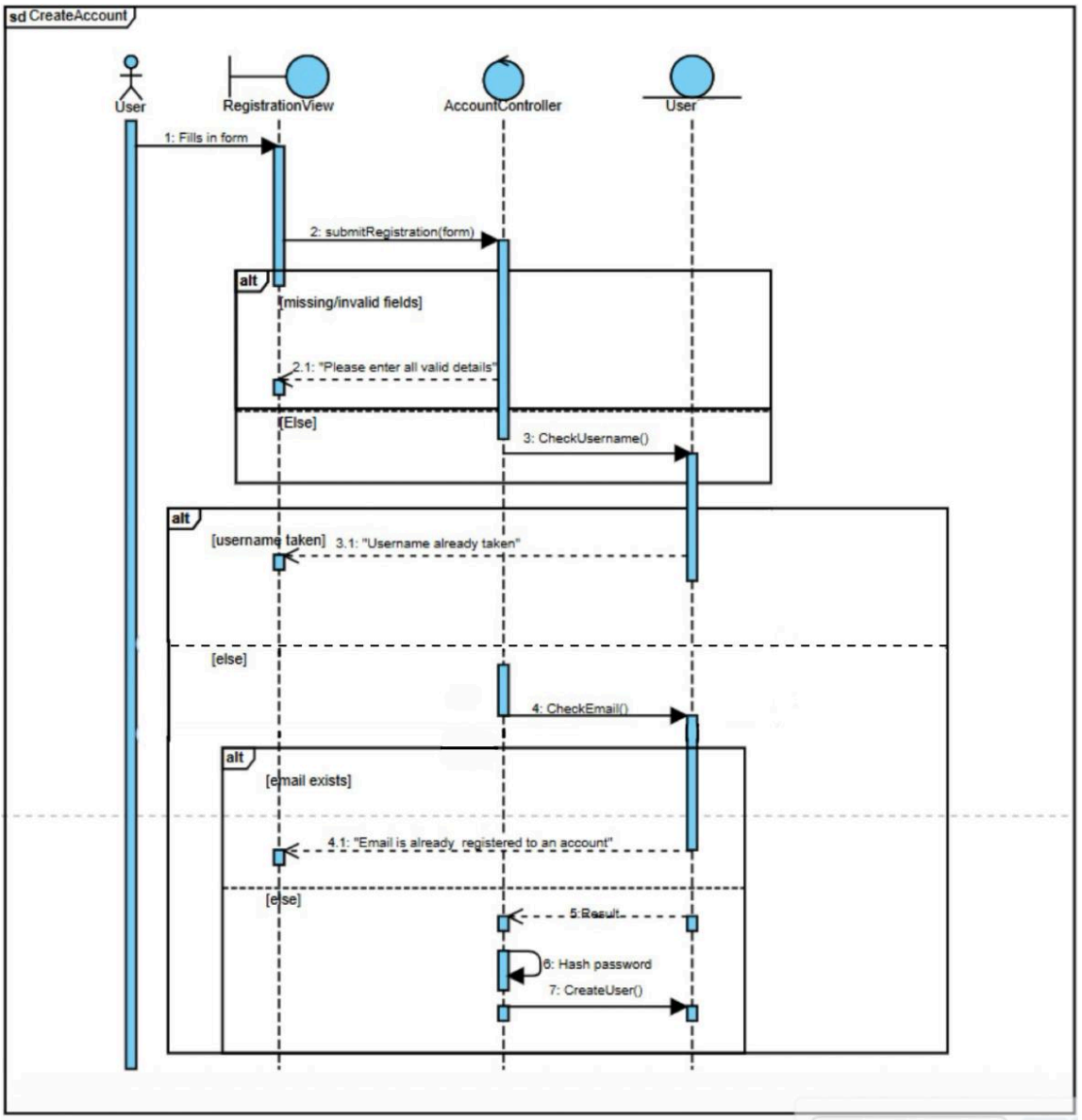
Data Access Layer (Persistent Data)

- **External System – Supabase Database:** Data for all entities and their attributes are stored to ensure persistence.
- **SupabaseClient:** Interacts with Controllers to get, create, update, and remove data. Serves as a gateway to the database.

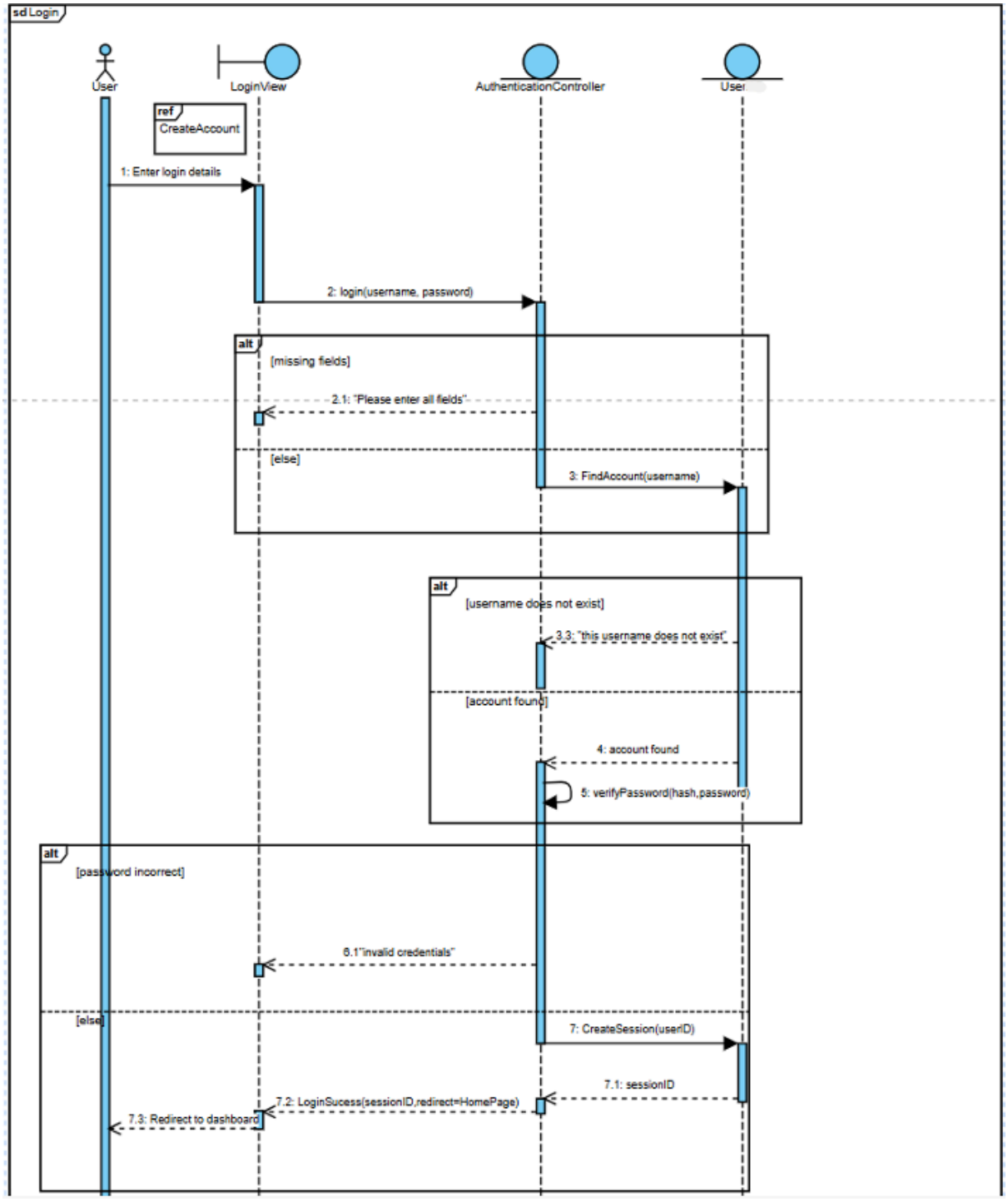
B. Sequence Diagrams

I. For Use Cases under #1 (Login)

I.I CreateAccount

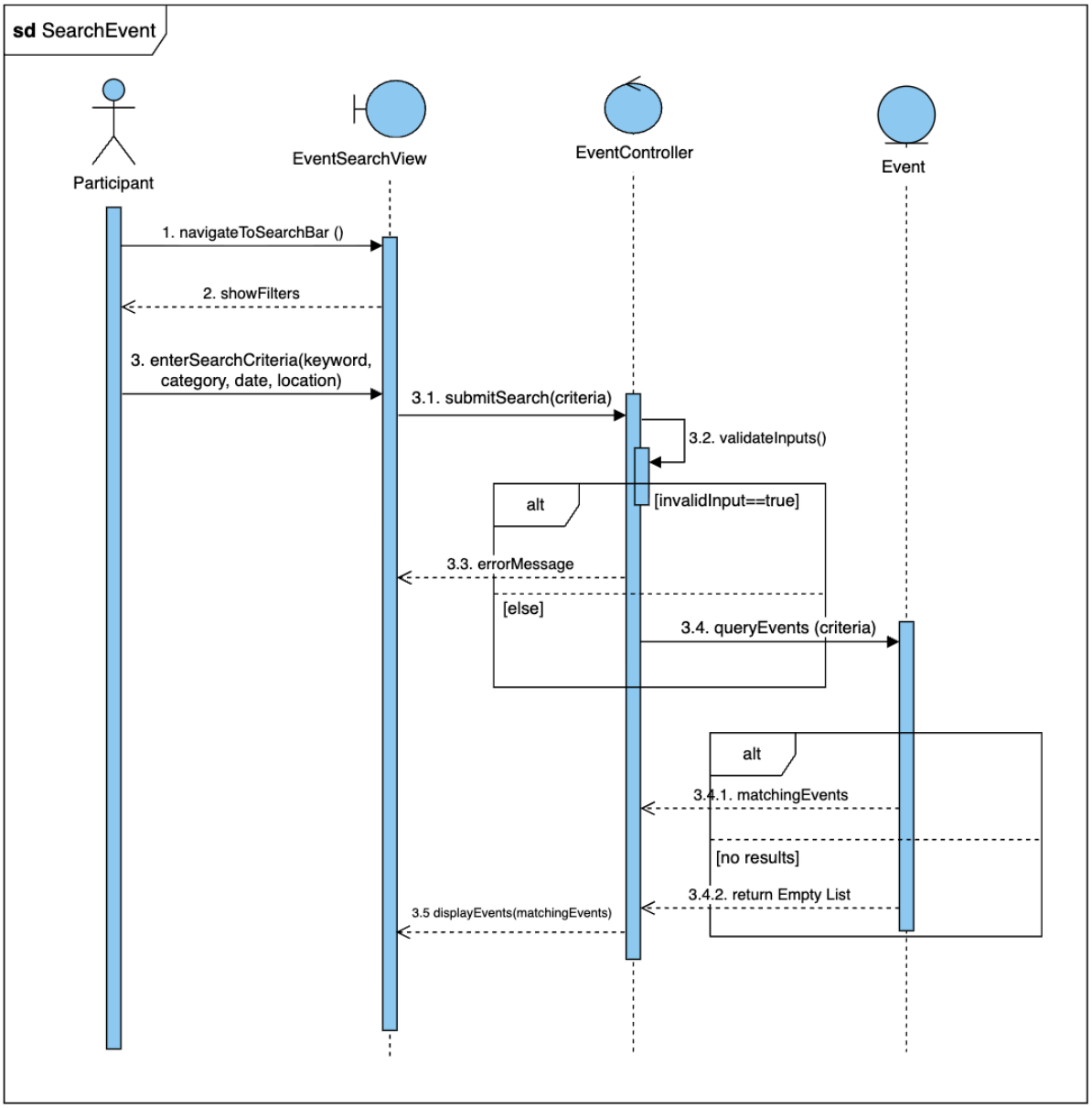


I.II Login

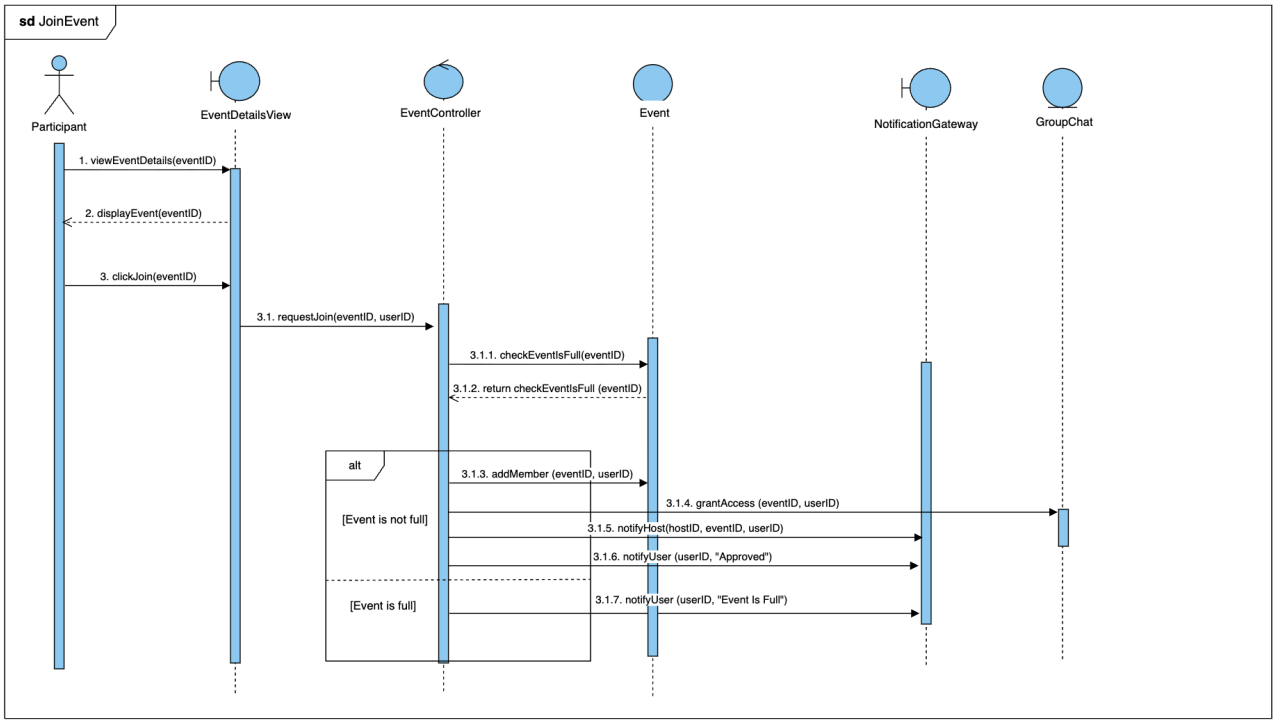

sd Login
User
LoginView
AuthenticationController
User
ref
CreateAccount
1: Enter login details
2: login(username, password)
alt
[missing fields]
2.1: "Please enter all fields"
[else]
3: FindAccount(username)
alt
[username does not exist]
3.3: "this username does not exist"
[account found]
4: account found
5: verifyPassword(hash,password)
alt
[password incorrect]
6.1"invalid credentials"
[else]
7: CreateSession(userID)
7.1: sessionID
7.2: LoginSucess(sessionID,redirect=HomePage)
7.3: Redirect to dashboard

II. For Use Cases under #2 (Participant)

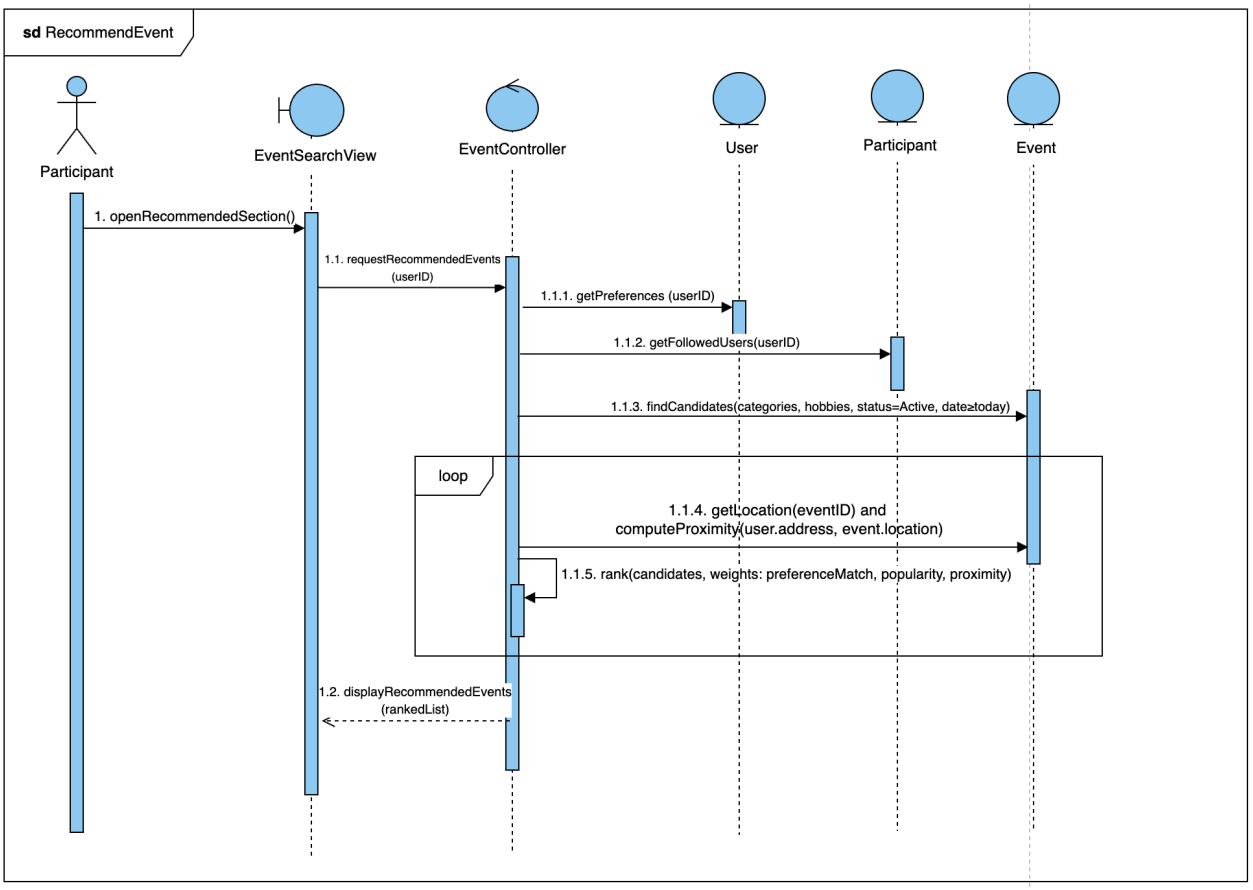
II.I SearchEvent



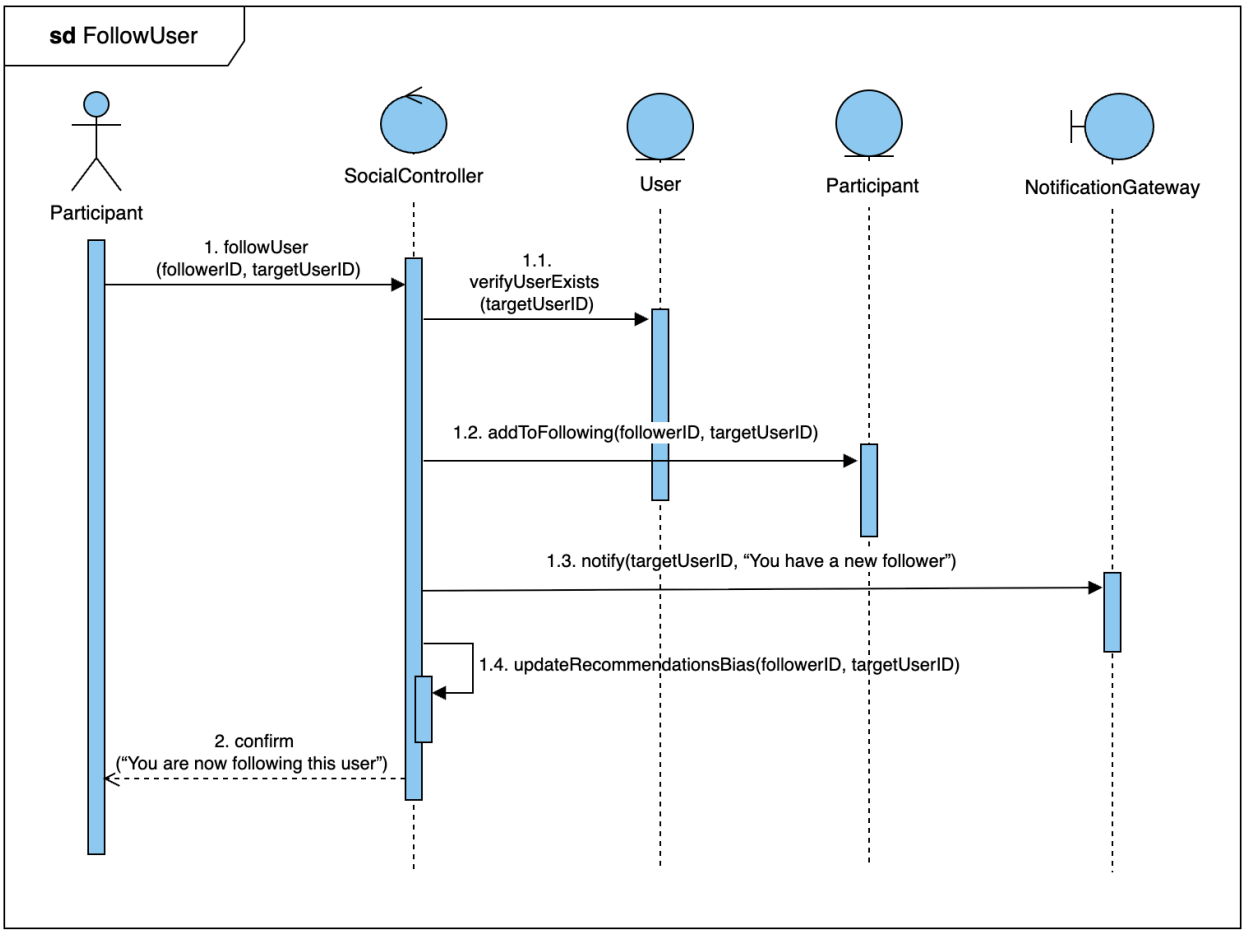
II.II JoinEvent



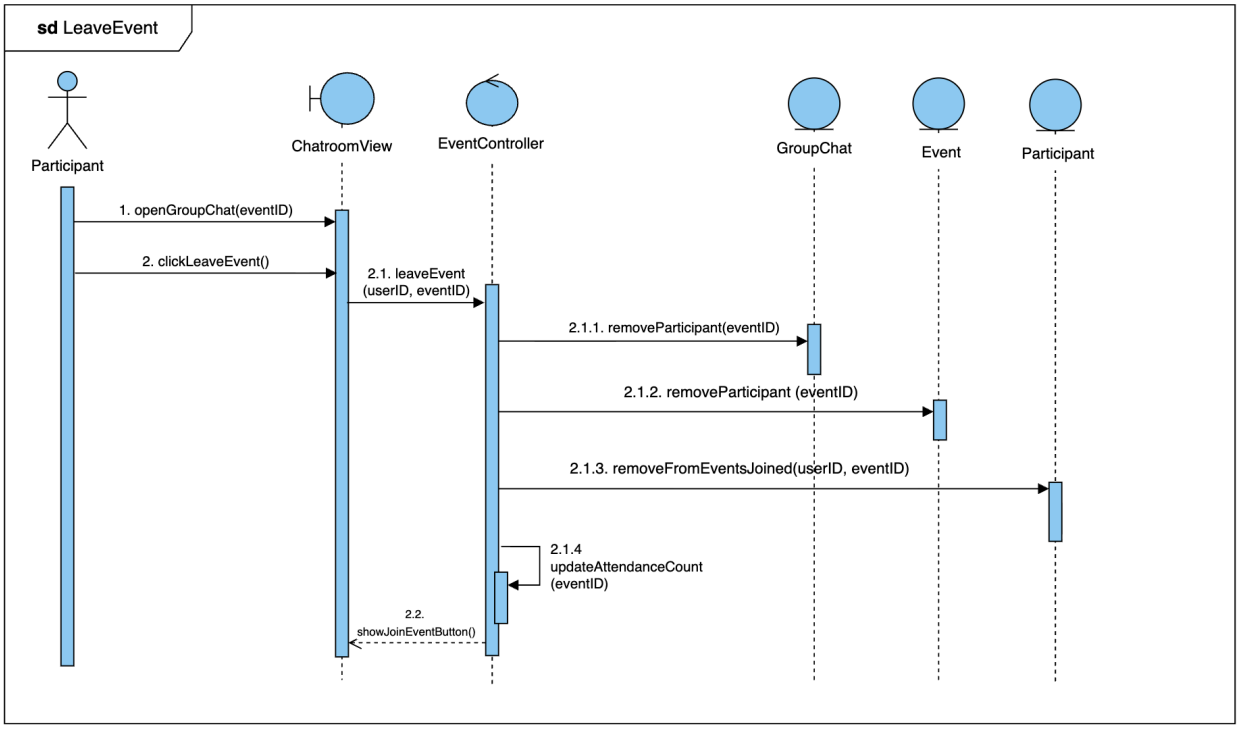
II.III RecommendEvent



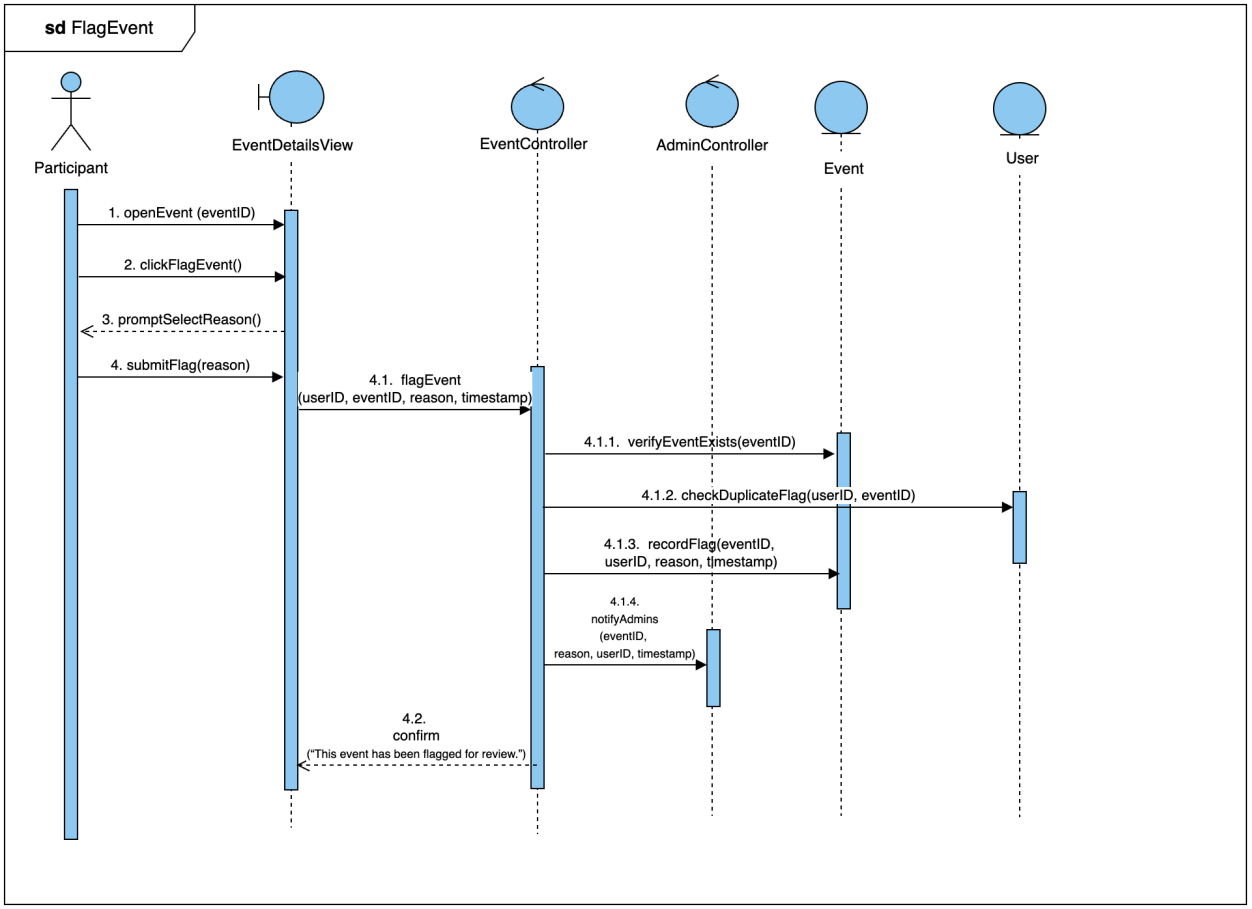
II.IV FollowUser



II.V LeaveEvent

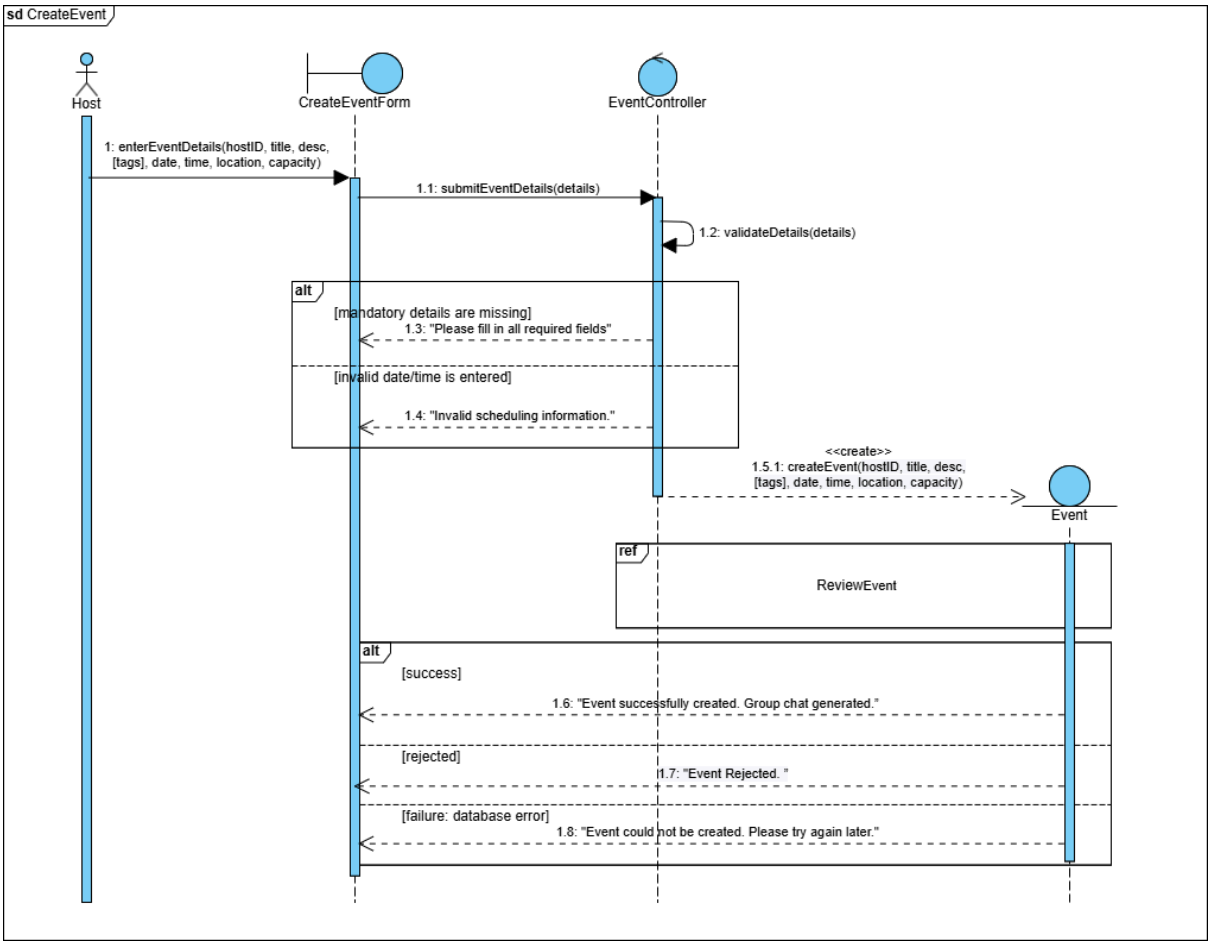


II.VI FlagEvent

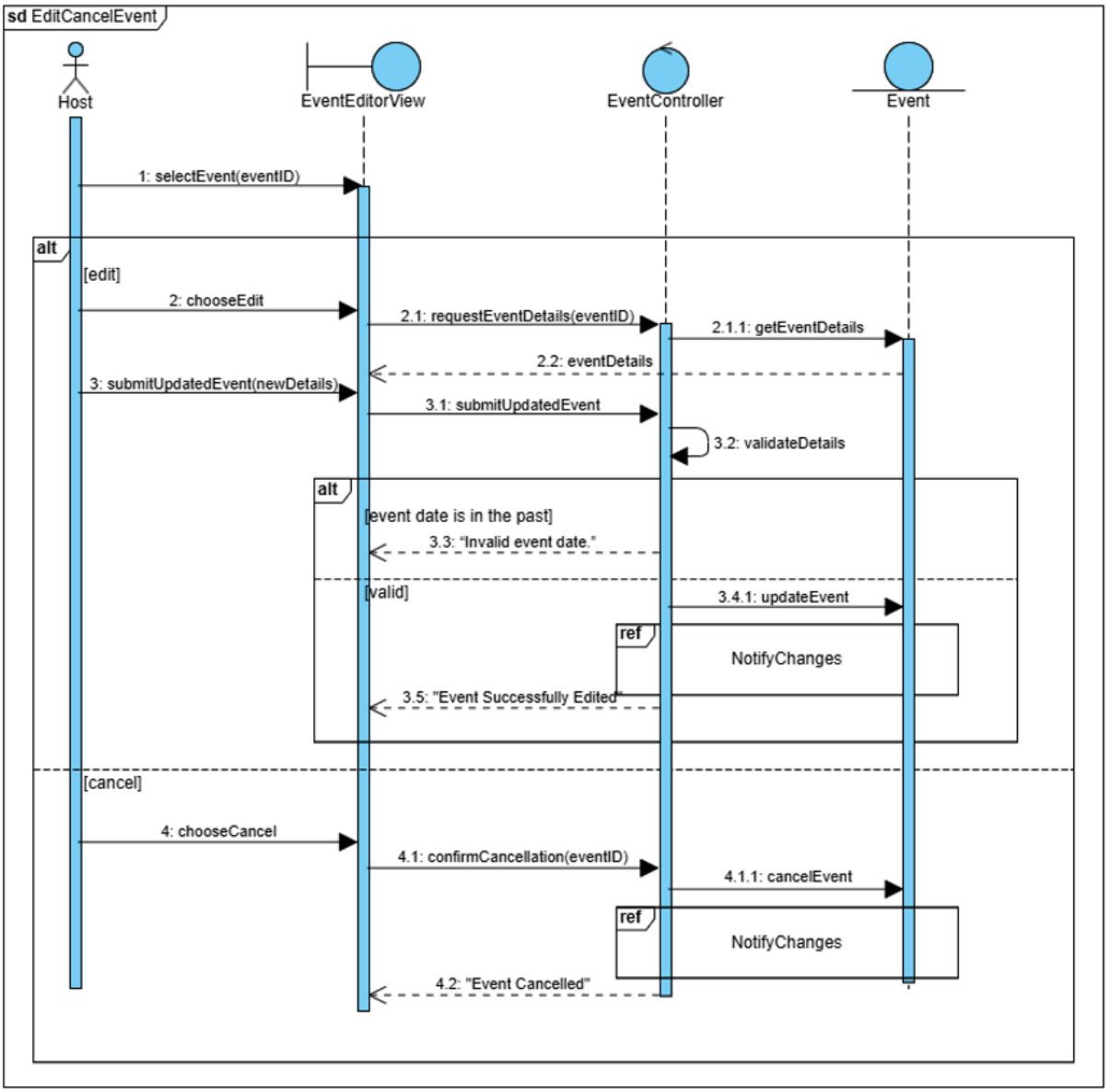


III. For Use Cases under #3 (Host)

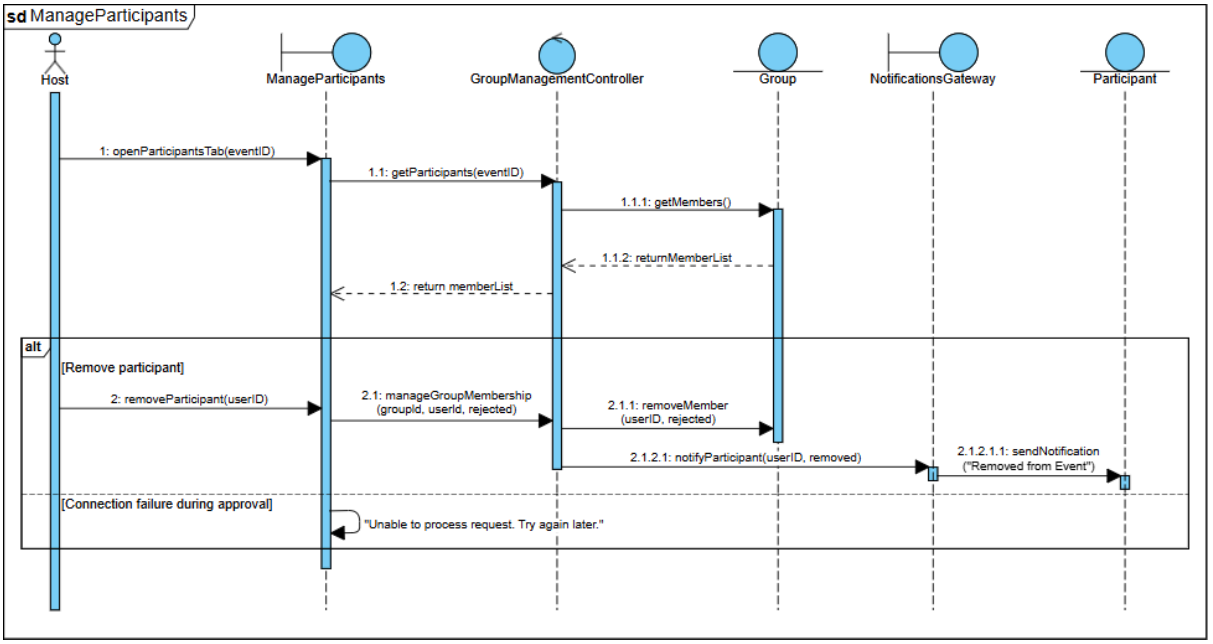
III.I CreateEvent



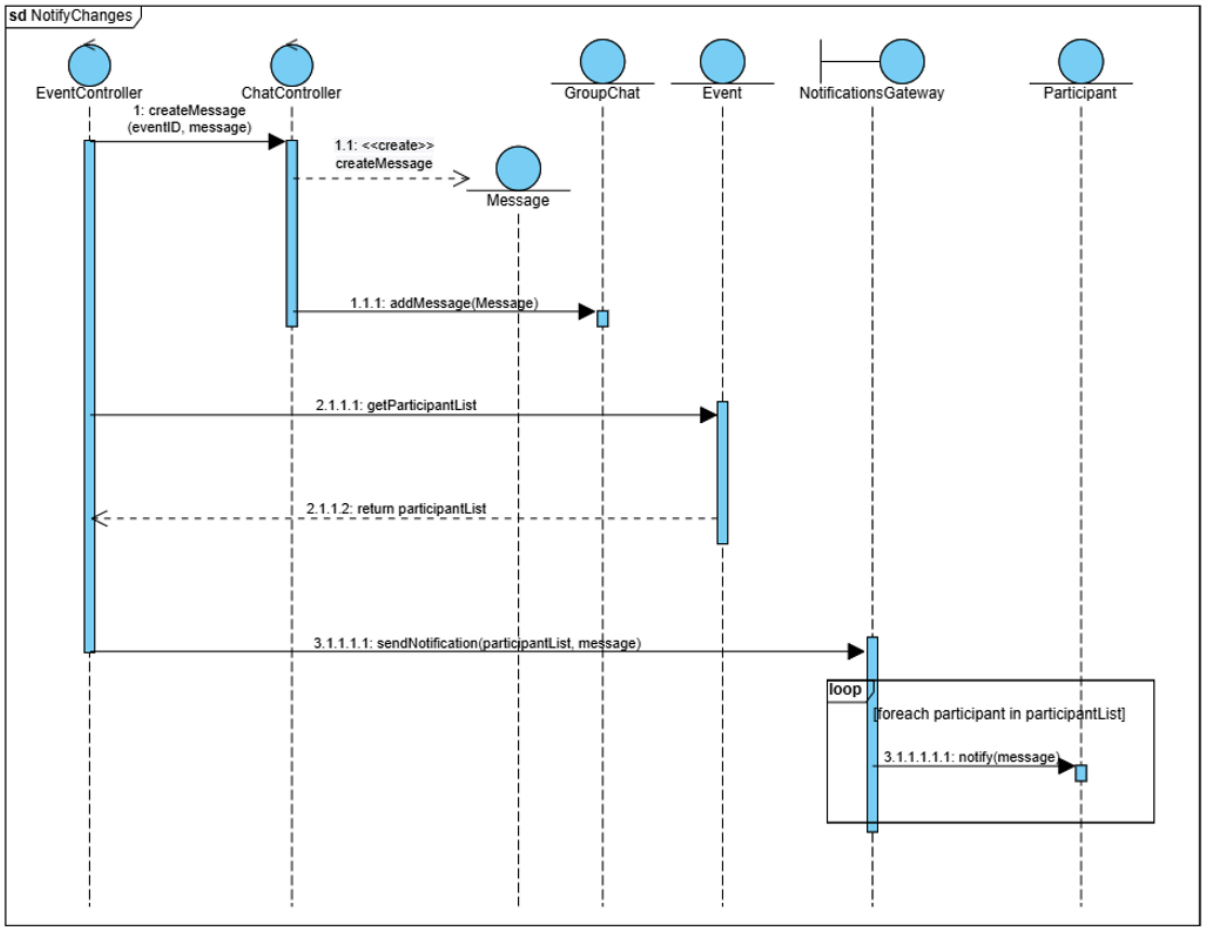
III.II EditCancelEvent



III.III ManageParticipants

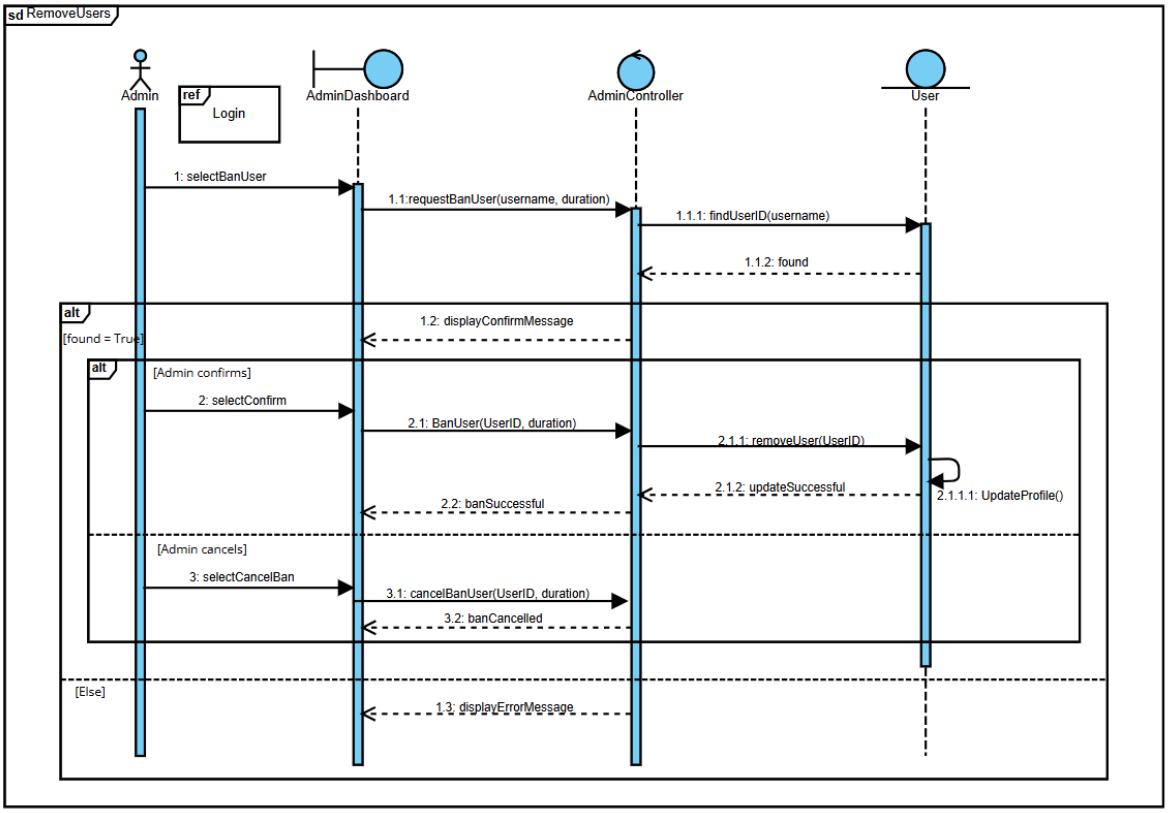


III.IV NotifyChanges

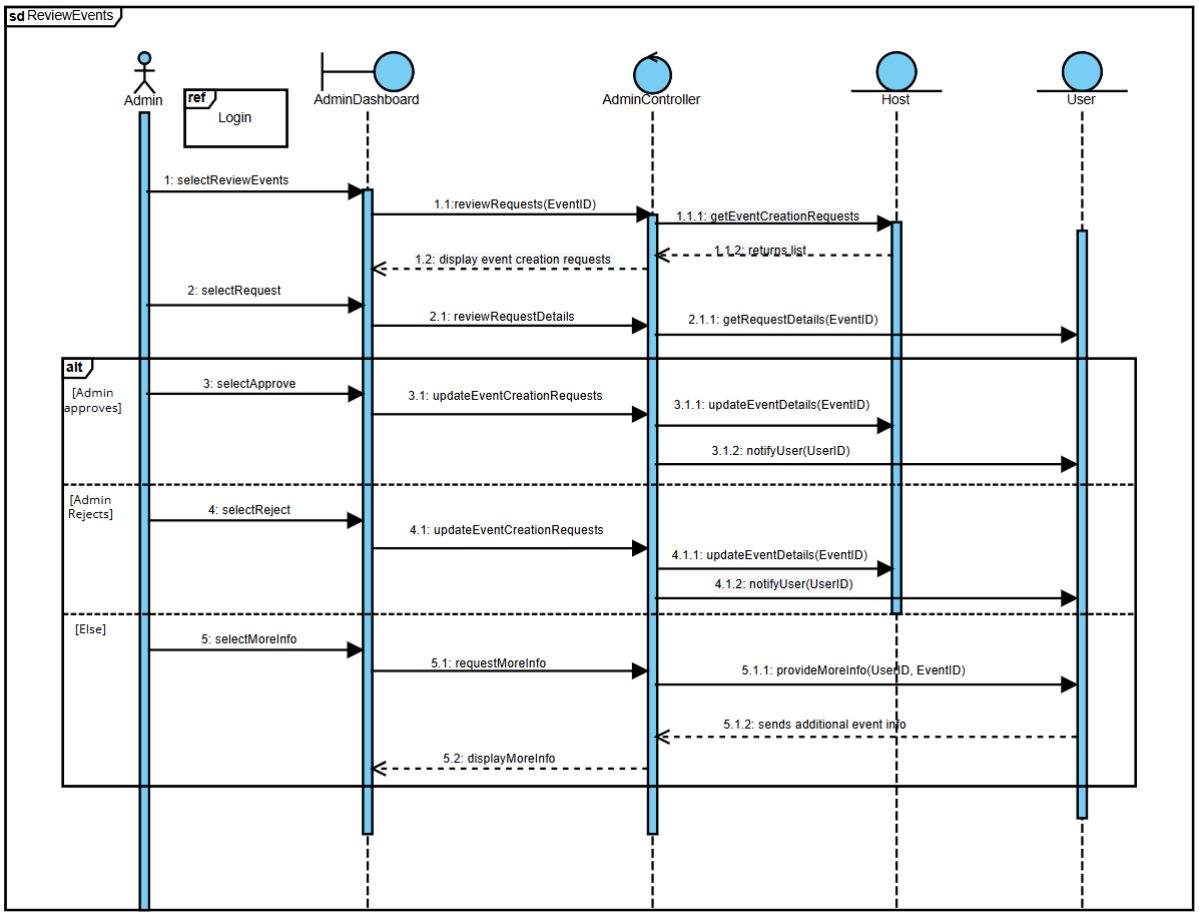


IV. For Use Cases under #4 (Admin)

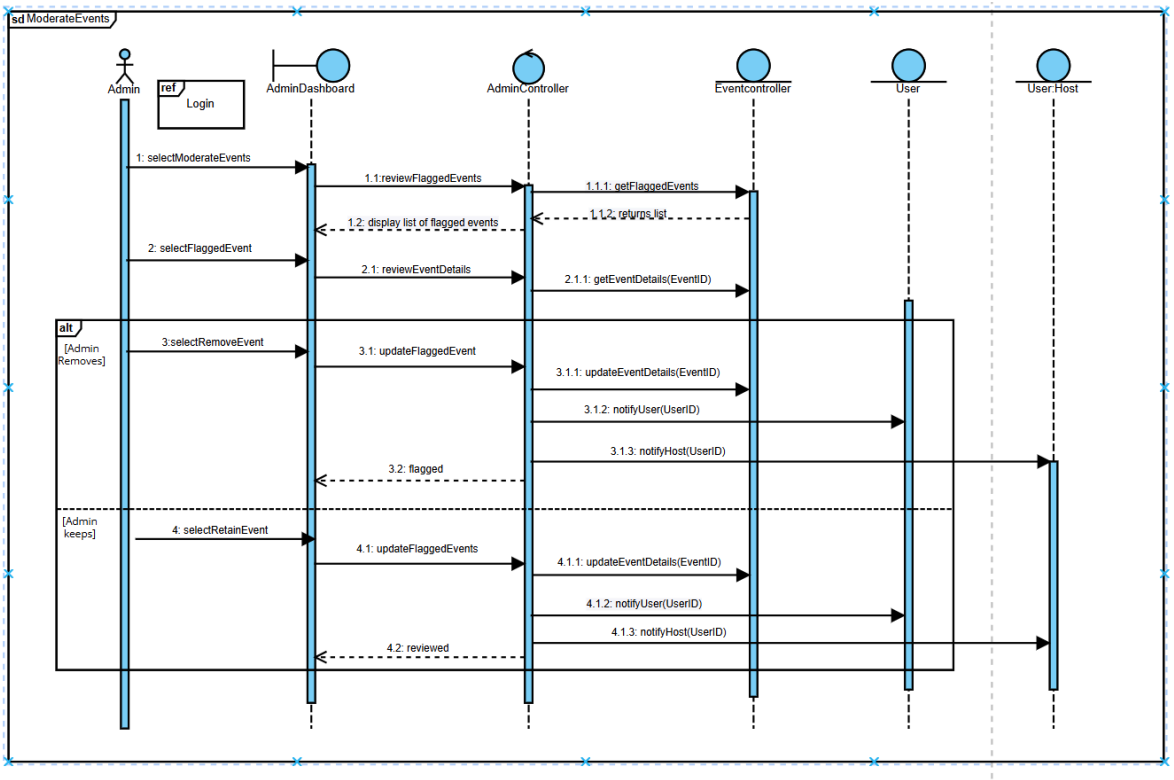
IV.I RemoveUsers



IV.II ReviewEvents

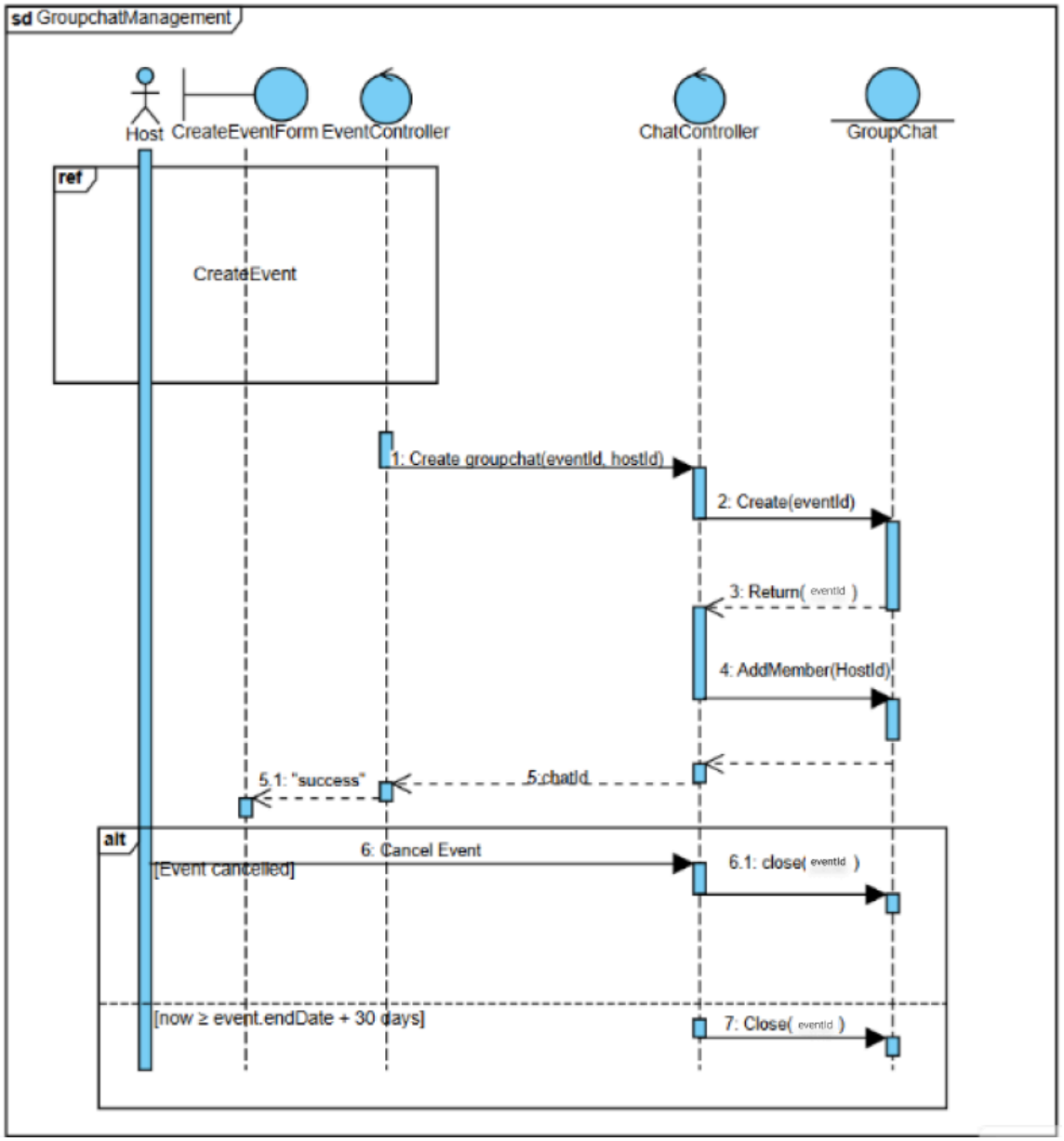


IV.III ModerateEvents

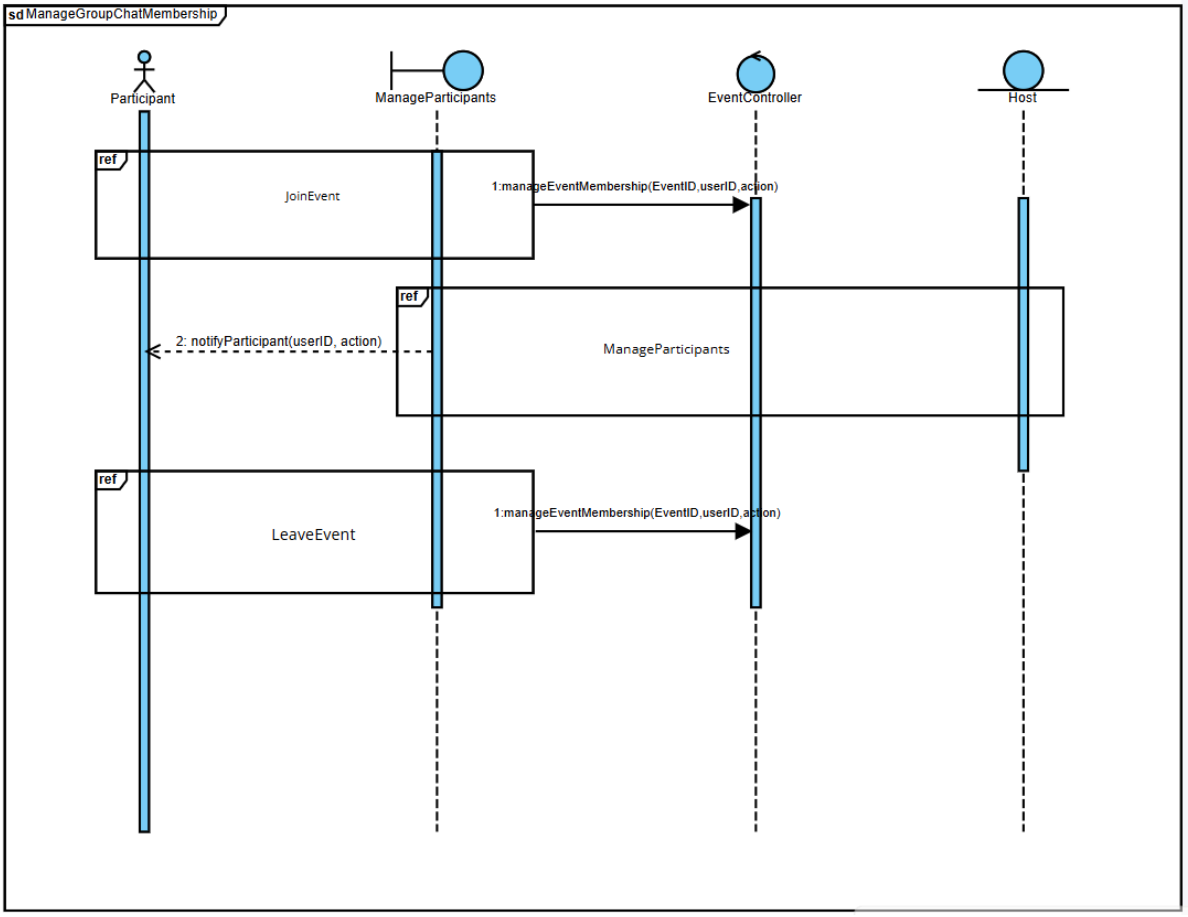


V. For Use Cases under #5 (Group Chats)

V.I GroupChatManagement

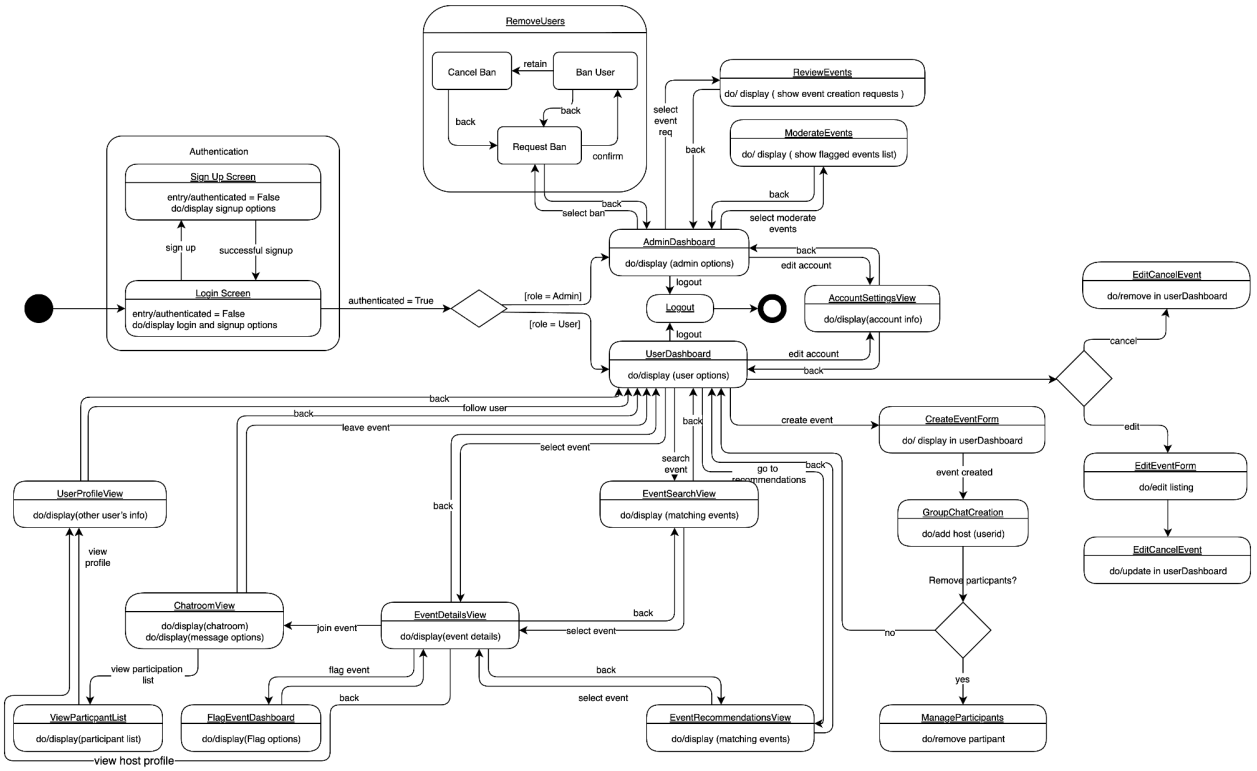


V.II ManageGroupChatMembership



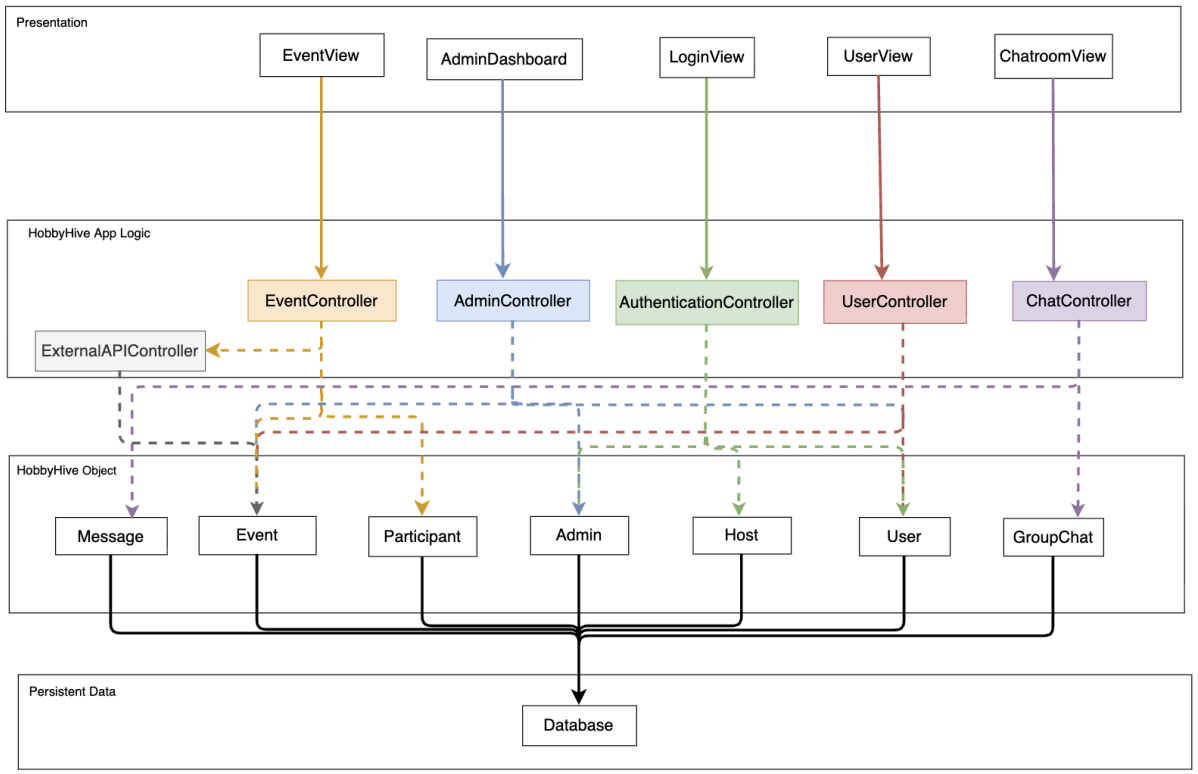
C. Dialog Map Diagram

If the image is unclear, please refer to the raw png file that is uploaded together with this document.



3. System Architecture

If the image is unclear, please refer to the raw png file that is uploaded together with this document.



Presentation layer

This layer is mainly responsible for the interaction between Users and HobbyHive. The different Views/Dashboards will then call for the respective controllers to run the App Logic.

This layer consists of:

1. **LoginView**
   Displays the authentication interface with login and account creation forms. Uses the services of AuthenticationController.
2. **EventView**
   Displays event listings, search interface, and event details. Provides the main interface for browsing and interacting with events. Uses the services of EventController.
3. **AdminDashboard**
   Displays the administrative interface for system moderation and management tasks. Uses the services of AdminController.
4. **UserView**
   Displays user profile information, account settings, and social connections. Uses the services of UserController.
5. **ChatroomView**

Displays the group chat interface for event-based communication. Uses the services of ChatController.

App Logic Layer

This layer contains all the controller classes that will provide the presentation layer with its services. The controller classes will request for entities from the Object Layer if and when necessary.

This layer consists of:

1. **ExternalAPIController**
   ExternalAPIController is called by other controllers when external services are needed. It handles third-party authentication, location-based services and map integration.
2. **AuthenticationCountroller**
   AuthenticationController contains verifying credentials, validating unique usernames/emails, and processing account creation and modifications.
3. **AdminController**
   AdminController contains the logic for system administrators. The logic includes reviewing event creation requests, moderating flagged events, banning users, and logging administrative actions.
4. **UserController**
   UserController contains modifying account information, managing follow/unfollow relationships, and updating user preferences.
5. **EventController**
   EventController contains the logic for the event listings. The logic includes searching, creating, editing, joining and recommending events to the user.
6. **ChatController**
   ChatController contains the logic for the groupchats. It contains the chat history and records the participants that sent messages in the chat

Object Layer

This layer contains entity classes that are used to implement the specific logic for each role. The entity classes are stored in the database of the persistent layer.

This layer consists of:

1. **Groupchat**
   Groupchat contains the chatID, eventID, list of participants and the list of messages sent by each participant.
2. **Message**
   Message contains the senderID, eventID and messageID as well as the content of the message and timestamp.
3. **Participant**

Participant contains the list of users the current user is following and the list of events that the specific user has joined so far.

4. **User**
   User contains all information about the user. It includes the username, password, email, address and profile picture. Each user is given a userID.
5. **Host**
   Host contains the list of events created by that specific user so far.
6. **Event**
   Event contains the event details like date, time, location, etc. It represents a scheduled activity that hosts can create and participants can join and interact in, closely linked to a Groupchat for communication.
7. **Admin**
   Admin contains the admin's details.

Persistent data

This layer contains the database that will store all of the entities.

4. Application Skeleton

Integrated Frontend–Backend Architecture:

HobbyHive is built using Next.js 15 (App Router) and Supabase, enabling a co-located full-stack structure where both user interfaces and server logic reside within the same application.

Frontend (With embedded backend logic)

- **app/**
    - **admin/** - Contains pages and components related to administrative functions (e.g., managing events, users).
    - **api/** - Contains API route handlers that serve as middleware between frontend requests and Supabase backend services.
    - **components/** - Stores reusable UI components shared across multiple screens (e.g., Navbar, ListingCard, ChatBubble).
    - **config/ -** Holds application-wide configuration files including category-to-location mapping rules for event creation.
    - **error/** - Contains error handling pages and fallback UI for failed operations or invalid routes.
    - **events/** - Manages event listing, creation, and details display for all users.
    - **groupchat/** - Implements real-time group chat interfaces using Supabase Realtime channels.
    - **host/** - Contains screens for hosts to create and manage events they organize.
    - **login/** and **register/** - Provide authentication pages connected to Supabase Auth.
    - **participant/** - Displays joined events and participation status for logged-in users.
    - **profile/** - User profile management, including personal details and preferences.
    - **MyEvents/** - Lists all events a user has joined or hosted.

Other frontend files include:

- **layout.tsx / page.tsx** - Core layout and root routing files for Next.js App Router.
- **globals.css** - Global stylesheet controlling the theme, colors, and layout.
- **utils/** - Helper functions (e.g., Supabase client, formatting utilities) used across multiple modules.
- **public/** - Static assets such as icons, images, and logos.

Backend

- **app/api/ -** Server-side API route handlers for data processing and external service integration:

    - **community-clubs/route.ts -** Fetches and processes Singapore community clubs data from data.gov.sg API for location selection in event creation.
    - **geocode/route.ts -** Provides address-to-coordinate conversion using OneMap geocoding API, enabling location search and autocomplete functionality.
    - **libraries/route.ts -** Fetches and processes Singapore libraries data from data.gov.sg API for location selection in event creation.
    - **mrt-stations/route.ts -** Fetches all MRT/LRT stations with coordinates to calculate nearest MRT to an event.
    - **parks/route.ts -** Fetches and processes Singapore parks data from data.gov.sg API for location selection in event creation.

External Backend (Supabase)

- **Database (PostgreSQL)** – Stores events, users, and chat messages etc.
- **Auth** – Handles secure authentication.
- **Realtime Channels** – Powers live chat updates.
- **Storage** – Hosts images and user assets.