System Design Document

CRC Cards

Class Name: Event	
Responsibilities: ● Store location, number of people, and description of event, theme (category) of event, event creator, etc ● Displayed in event cards ● Popup on cards for more details	Collaborators: ● User

Class Name: User	
Responsibilities: ● Store username, name, email, and password, friends, enrolled events	Collaborators: ● Event

Class Name: Message	
Responsibilities: ● Contains a message that is sent in a group chat or user	Collaborators: ● User ● Event

Class Name: Find Event Page	
Responsibilities: ● Shows event details ● User can enroll or un-enroll through here ● Can search for events with various filters ● Main/Landing page ● User can waitlist for full events to auto-enroll if someone unenrolls	Collaborators: ● Event ● User

Class Name: Messages Page	
Responsibilities: ● Shows messages ● Associated to an event or 2 users. ● Stores history of Messages	Collaborators: ● Message ● User ● Event

Class Name: Navbar	
Responsibilities: ● Present on all major pages ● The user will navigate between pages using this	Collaborators: ● Find event page ● User profile ● Friend page ● Messages page ● Add event

Class Name: Event list	
Responsibilities: ● Shows events that can be joined ● Users can view event details they are interested in.	Collaborators: ● Event page ● Event ● User

Class Name: Edit Profile page	
Responsibilities: ● Allows user to edit the settings of their profile ● Accessed through the profile page	Collaborators: ● Profile page ● User

Class Name: Profile page	
Responsibilities: ● Shows the current user’s information ● Shows users created and enrolled events	Collaborators: ● User ● events

Class Name: Login/signup page	
Responsibilities: ● Verifies and distinguishes different users	Collaborators: ● User

• Starting point. After logging in or registering it goes to the find event page	

Class Name: Add event	
Responsibilities: • Allows a user to create a new event to be posted	Collaborators: • Event • User

Class Name: Friend page	
Responsibilities: • Allows users to add friends/remove, accept friend requests, and see a list of their friends.	Collaborators: • User

System Description

The project is web based so the OS, programming language compiler, virtual machine, etc should not be relevant so long as the IP address isn't blacklisted from the database and the user has a stable internet connection.
From a development perspective, any OS should be fine as long as it can run a code editor like VScode. The project uses the MERN stack, so languages include Javascript, HTML, CSS and will need to have React and Node installed. It is a three tier architecture and the database is using MongoDB, making it a NoSQL document database.
Currently we are using the following libraries: Axios, Material UI, Moment, Redux, and Cors, Antd, Socket.io, react-paginate, country-state-city, react-select, Twilio.

System Architecture

We are using Mongoose to structure our data in schemas for requests with the backend. All of our page components can make requests with the appropriate schema to the appropriate endpoint, or for other components they will be structured appropriately as schema.

Components

Our project can be broken down into 4 main components: user, events, messages and conversations, along with 7 pages. The following is a rough breakdown of all the pages we expect, and fields for users, events, and passwords:
Pages

- Profile page
    - Events user has joined.
    - Events user has made

- Edit Profile page
- Find events
- Messages page
    - Specific Group Chat or direct message.
- Add Event
- Login/Signup
- Friend page

User with following info:

- User ID
- Name (first and last)
- Email (unique)
- Password
- Phone number
- User Image (optional)
- Preferred Locations
- Themes/categories
- Enrolled events: [events, category]
- Friends: [users]
- Verified (boolean)
- Sent friend requests
- Received Friend Requests

Event with following info:

- Event ID
- Event Name
- Event Description
- Event Image (optional)
- Event creator (email + phone number)
- Link (optional)
- Themes
- Time, Date
- Location (4 fields: Country, Region, City, Address (optional))
- Max number of people
- Price
- Waitlist

Messages

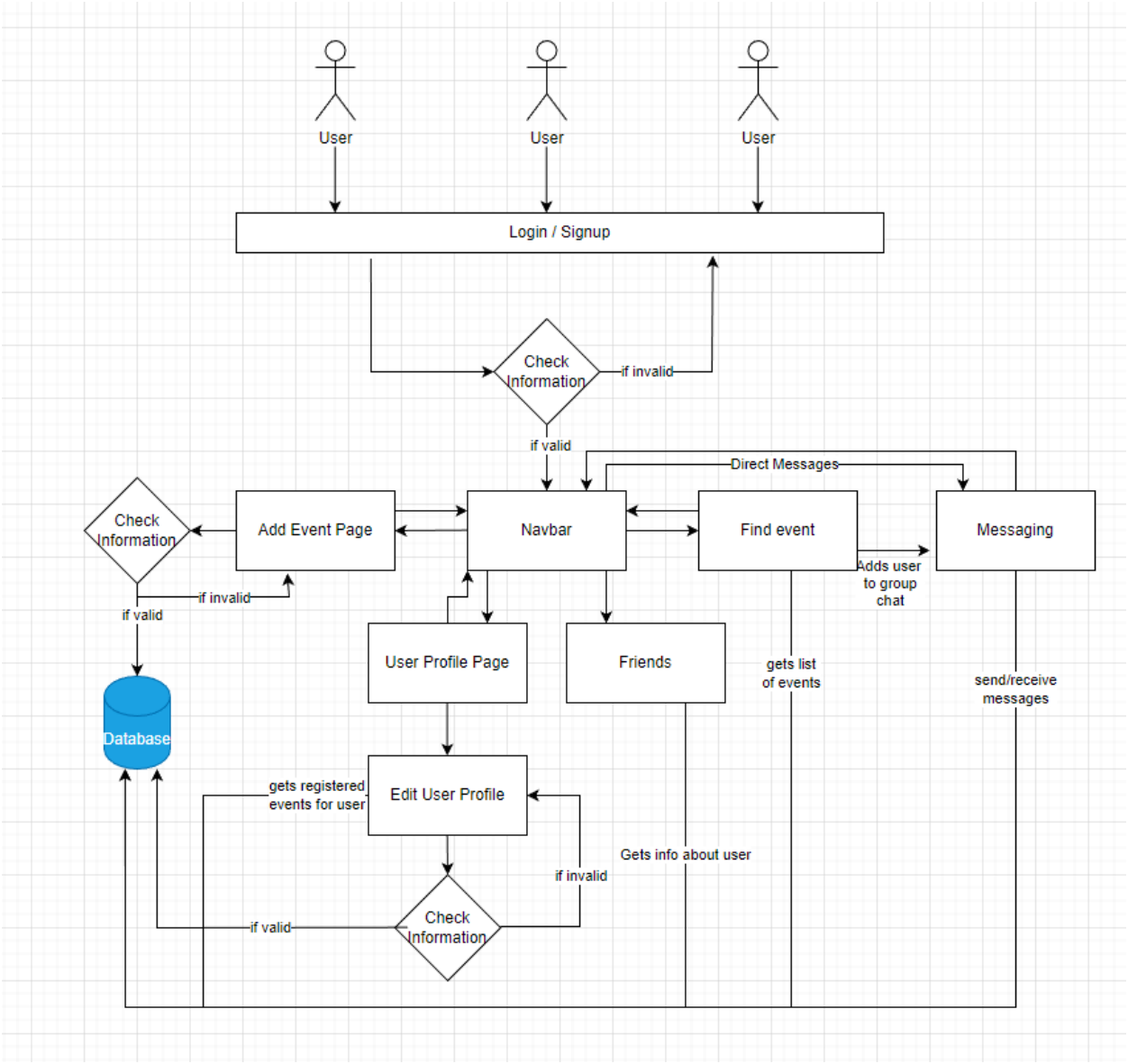
- Message ID
- Associated Conversation ID
- Sender
- Text
- Created time

Conversations - group chats for events

- Conversation ID
- Event name
- Event ID

- Created time

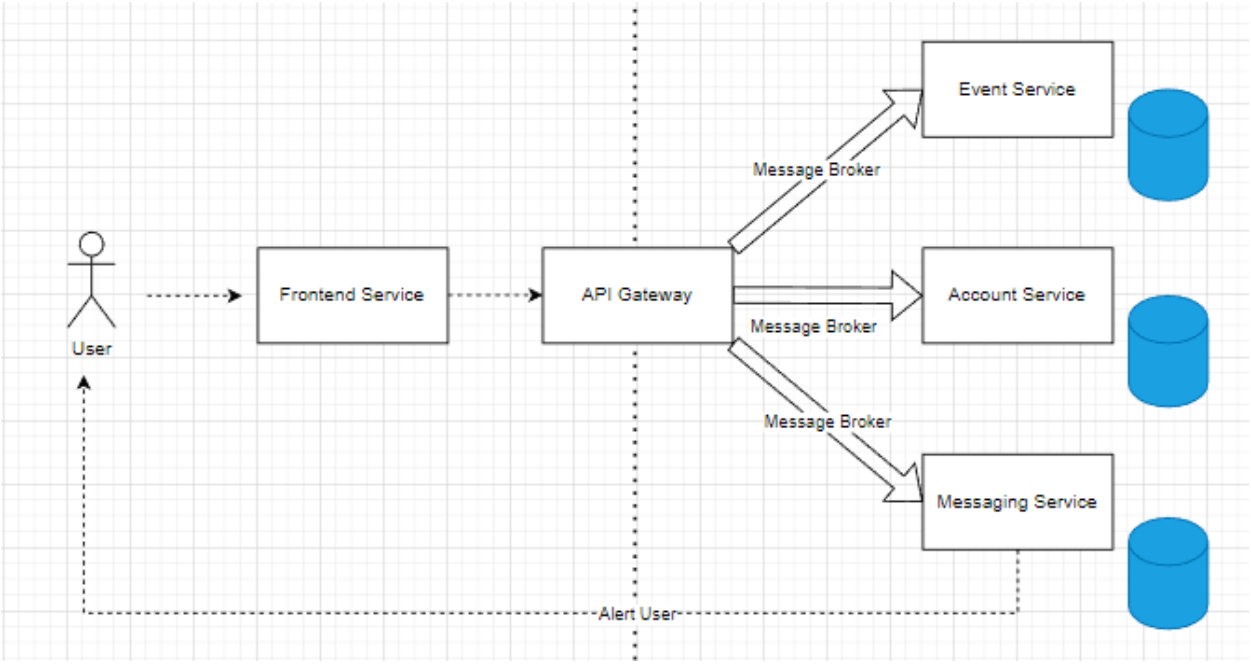
Below is a diagram of how the components connect to each other.



- The architecture of the system is fairly simple, we first have an authentication component responsible for signing up and logging in users.
- Once a user is logged in, they are taken to the find event page, but they can navigate to any of the pages with the navbar as shown.
- The add event page allows them to add an event by filling in the required fields. Once complete, it will post the event details to the database.
- The user profile page allows the user to review their own profile, where they can also navigate to the edit user profile page, where they can edit their name, password, and

email, preferences, etc. The info shown will be the fields as detailed above for the user component.

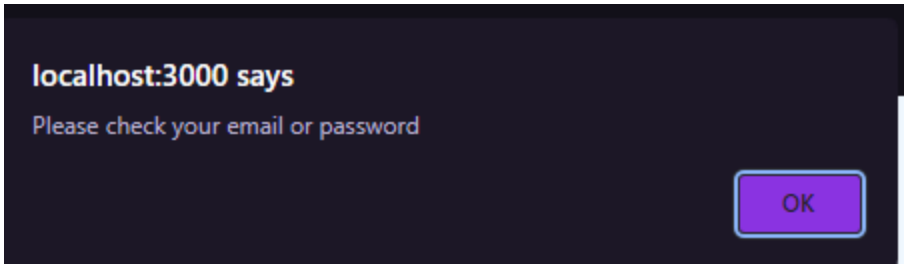
- In profile they can also see the events they are enrolled in and events they created
- The event list will fetch all the events posted to the database based and can be filtered on various things
- From the find event page or wherever there are event cards the user can view the details of a specific event in another page. The user will be able to enroll and leave events from this page. The info will be the fields as detailed for the events component.
- If a user enrolls to an event, it will show on their profile and they will be able to message the group from the messaging page.
- In friends the user can send, accept, and decline friend requests, remove friends, see a list of their friends, and search users.
- Users can message in group chats or to individual users.
- The user can also signout via the navbar.



The above is a functional architecture diagram showing the event driven architecture for the project.

Error Handling

For errors and exceptional cases, all cases are expected to be handled in a similar fashion. When a request is made there are appropriate responses for any errors. The caller will receive the response and have an appropriate response. For what we currently anticipate, an alert will pop up on screen along with an appropriate message for why there is an error like shown below



We anticipate the following errors:

- Invalid/lack of user input
- Network or external system failure