Passion Pals System Design Document

CRC Cards

Class Name: Event	
Responsibilities: ● Store location, number of people, and description of event, theme (category) of event, etc	Collaborators:

Class Name: User	
Responsibilities: ● Store username, name, email, and password, etc	Collaborators:

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

Class Name: Event Page	
Responsibilities: ● Shows event details ● User can enroll or un-enroll through here	Collaborators: ● Event

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

Class Name: Calendar	
Responsibilities: ● Keeps track of date and what events are scheduled when for the user.	Collaborators: ● Event Page ● Event ● User

Class Name: Dashboard	
Responsibilities: ● Main hub for application ● The user can access other page components like profile or calendar	Collaborators: ● Event list ● User profile ● Calendar ● Messages ● 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	Collaborators: ● User

Class Name: Login/signup page	
Responsibilities: ● Verifies and distinguishes different users ● Starting point. After logging in or registering it goes to the dashboard.	Collaborators: ● User ● Dashboard

Class Name: Add event

Responsibilities:	Collaborators:
● Allows a user to create a new event to be posted	● Event ● 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 have the following libraries are Axios, Material UI, Moment, Redux, and Cors, Antd.

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 3 main components: user, events, and messages, along with 9 pages. The following is a rough breakdown of all the pages we expect, and fields for users, events, and passwords:
Pages

- Profile Settings
- Edit Profile settings
- Dashboard
- Event List
- Messages page
    - Specific Group Chat or direct message.
- Add Event
- Calendar
    - Events user has joined.
- Login/Signup
- Event page

User with following info:

- Name
- Email
- Username (unique)
- Password (encrypted)

- User Image (optional)
- Location
- Subscriptions: [events, category]
- Friends: [users]

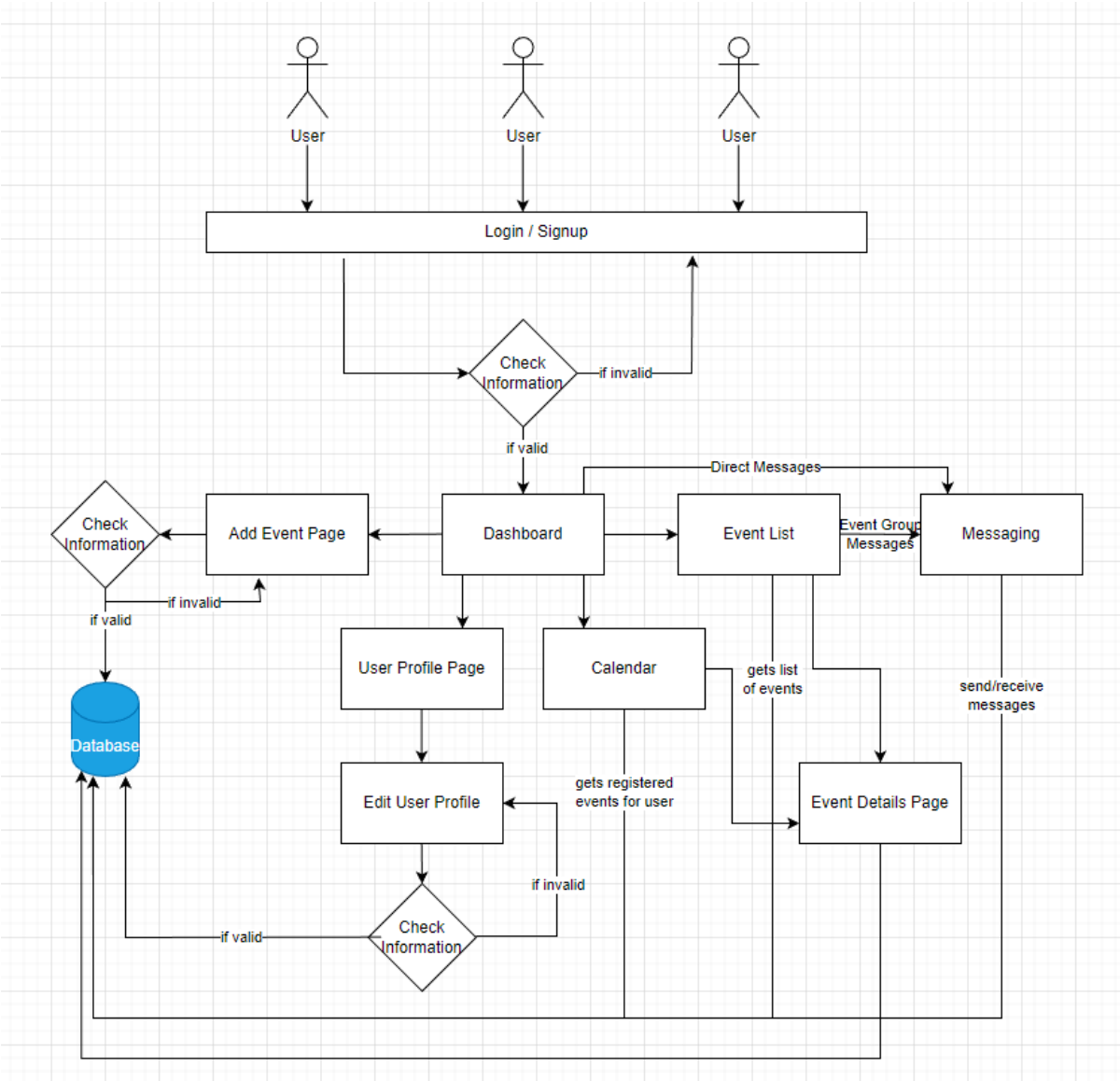
Event with following info:

- Venue Name
- Venue Description (detect hyperlinks)
- Venue Image (optional)
- Time, Date
- Location
- Max number of people
- Price
- Users: {user1, user2} (sort by friends relative to user)
- GroupChat: {messages} (upon enrolment)

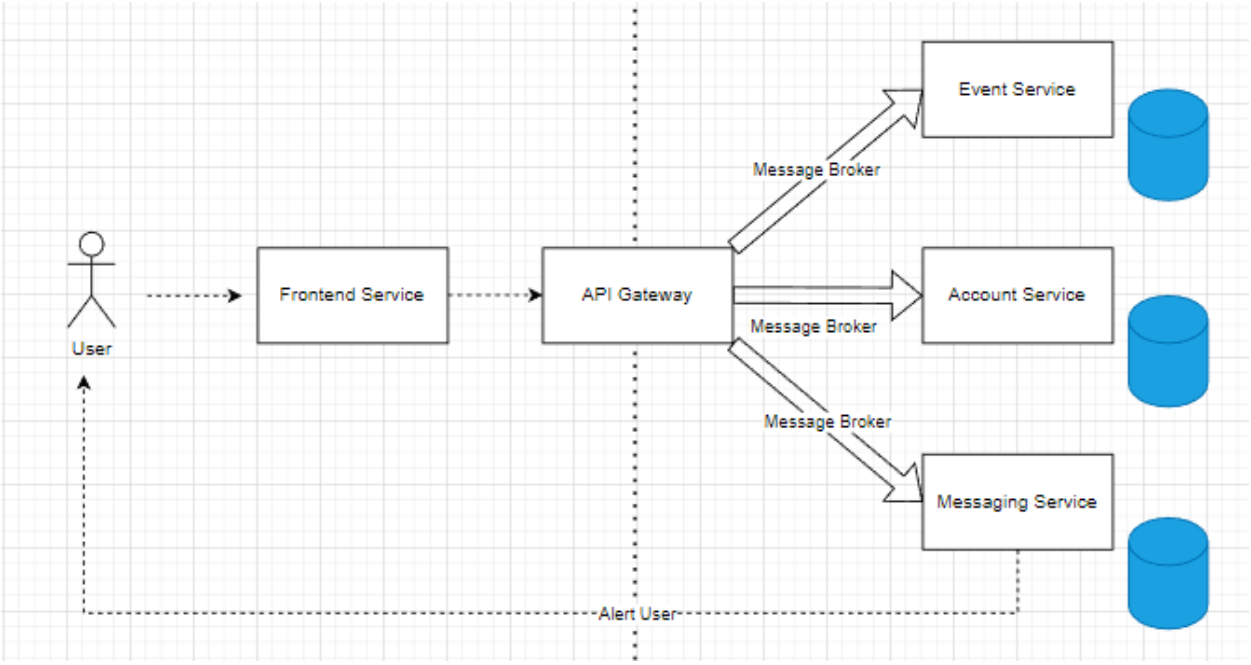
Messages

- Messages sent between users or in groups on the messages page.

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 dashboard, in which they can navigate to a variety of services and pages. The dashboard displays the events they are enrolled in.
- 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. The info shown will be the fields as detailed above for the user component.
- The calendar page shows a calendar dating all of the events the user has registered for.
- The event list will fetch all the events posted to the database based on filters for the user and/or an algorithm, and display them to the user.
- From the event list or calendar 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 calendar and they will be able to message the group from the messaging page.
- Users can also access direct messages in the messaging page from the dashboard. Messages will be the messages component as mentioned above.

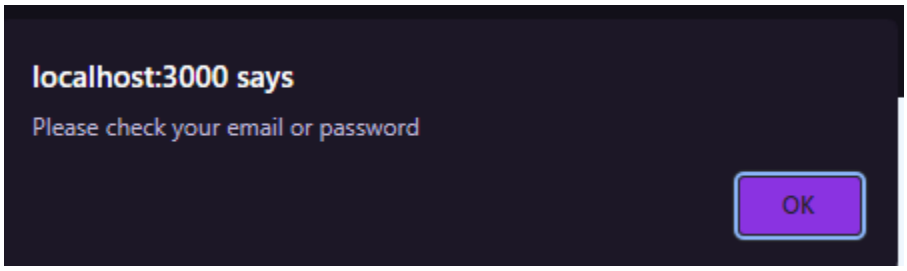


The above is a functional architecture diagram showing the event driven architecture for the project.

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