

## CRC Cards

<b>Class Name: Accounts</b>
<b>Parent Class: n/a Subclasses: n/a</b>
<b>Responsibilities:</b> <ul style="list-style-type: none"><li>• Store account information</li><li>• Allow accounts to add each other as friends</li></ul>
<b>Collaborators:</b> <ul style="list-style-type: none"><li>• Events</li></ul>

<b>Class Name: Events</b>
<b>Parent Class: n/a Subclasses: n/a</b>
<b>Responsibilities:</b> <ul style="list-style-type: none"><li>• Store event information</li><li>• Allow accounts (users) to sign up for events</li></ul>
<b>Collaborators:</b> <ul style="list-style-type: none"><li>• Accounts</li></ul>

<b>Class Name: Clubs</b>
<b>Parent Class: Events Subclasses: n/a</b>
<b>Responsibilities:</b> <ul style="list-style-type: none"><li>• Store club information</li><li>• Allow accounts (users) to sign up for clubs</li></ul>
<b>Collaborators:</b> <ul style="list-style-type: none"><li>• Accounts</li></ul>

<b>Class Name: Study Groups</b>
<b>Parent Class: Events Subclasses: n/a</b>
<b>Responsibilities:</b> <ul style="list-style-type: none"><li>• Store study group information</li><li>• Allow accounts (users) to sign up for study groups</li></ul>
<b>Collaborators:</b> <ul style="list-style-type: none"><li>• Accounts</li></ul>

## System Interaction with the environment:

Operating System:

- Any machine with Docker.

Programming languages:

- The back end is developed in the latest version of Python.
- The front end is developed in React.js.

Virtual Machines and Containers:

- We're gonna use Docker for containerization.

Databases:

- We will use MySQL databases for most of our storage, since we have determined that most of the data makes sense to be stored in this fashion. SQLAlchemy will be used with Python backend to interact with said database.

Network Configuration:

- MySQL database is accessed with port 42069.
- Web application is accessed with port 42070.

External Services and APIs:

- FastAPI will be used for frontend-backend connection. I think that's how it works.

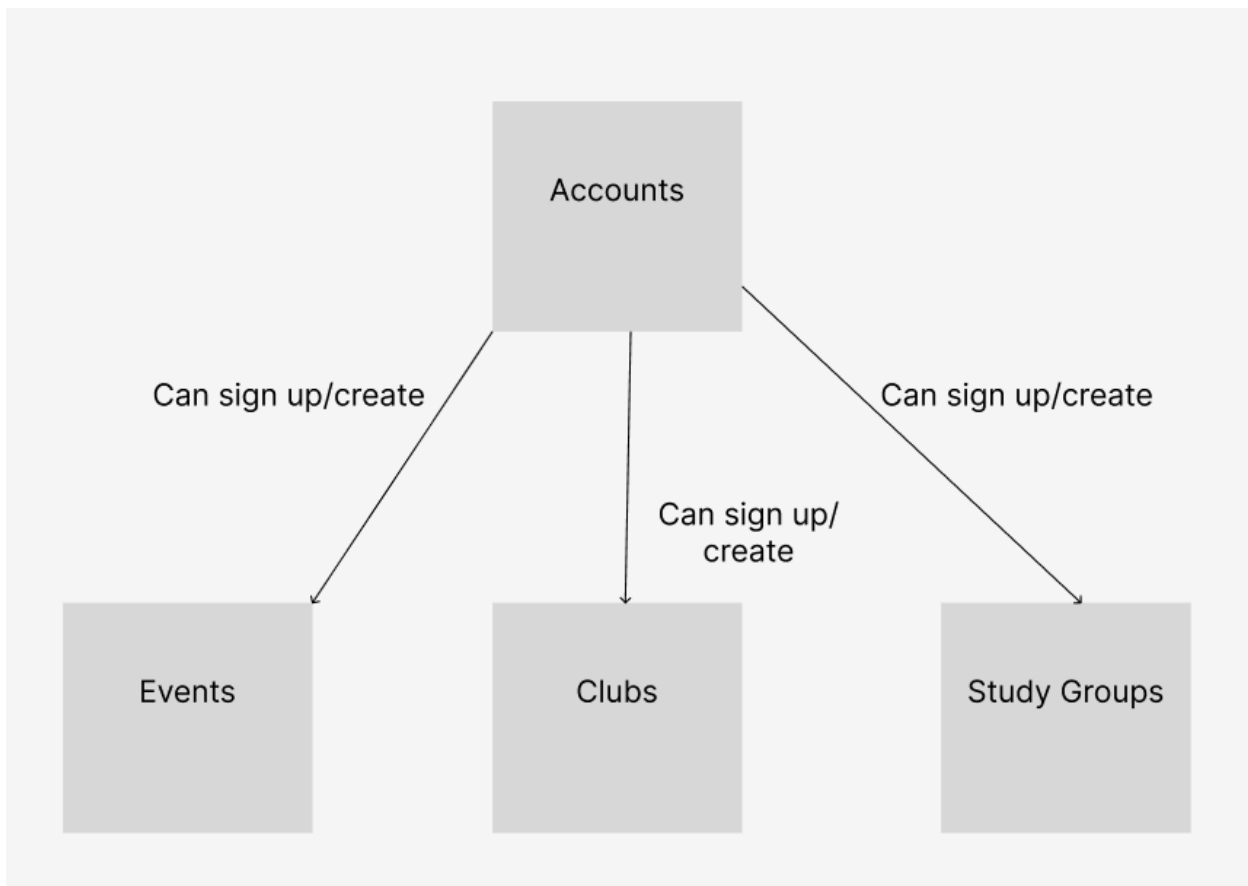
Hardware Requirements

- Enough to do a "docker compose up". Shouldn't be that much.

Assumptions

- Able to use Google, or any other browser.
- Servers involved in this system have internet access.

## System Architecture Diagram:



## **System Decomposition and Error Handling**

1) System Decomposition: Each of the components shown in the diagram would have their own webpage on our website (except accounts, unless you count the profile page), each with their own databases for storing their respective information. Accounts will be able to sign up for and/or create any of these, but I imagine we will not store which of them each account has signed up for/created. A simple "Username created this Event/Club Opening/Study Group" on the display is good enough.

### **2) Error Handling:**

- Perhaps the user logs in incorrectly, then display something like "login failed". Same deal for registration, if they somehow register incorrectly, like forgetting to enter a password. In general, most invalid user input should result in an error message the user can see.
- If a system failure occurs, ideally we have a way to make sure not too much is lost, but not sure how to do so.
- If a network failure occurs, we imagine the system will continue to retry to reconnect until then. Perhaps some way of informing the user of the network issue as well.