Clava

Team 11

Hunton, Alex
Lee, Christopher
Leungwattanakij, Kris
Pan, Leonard
Tinkess, Kai

Purpose

At this time, most college clubs and organizations do not have a centralized system in which general information, member data, and funds are managed. This directly impacts their ability to effectively operate by increasing the amount of time it takes to complete basic tasks, such as granting individual permissions or providing reimbursements.

The purpose of our project, Clava, is to provide college organizations with a suite of tools to manage their members, expenses, and documentation. Clava will utilize a user interface designed with functionality and ease-of-use for clubs in mind, with feedback directly pulled from Purdue organizations like the Boiler Book Club. Additionally, Clava will also provide users with access to a variety of managerial tools, such as a financial, documentation, and event hub that will help expedite all executive tasks within an organization. Due to all of these tools being centralized, tasks like managing member permissions for documentation and finances will be streamlined, and organizational information will be much easier to view.

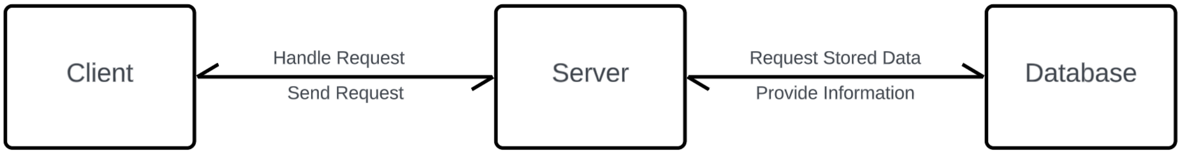
Design Outline

High Level Overview

Clava is a web application that maintains a list of users, all of which are officers of college organizations, and a list of college organizations with related data. Users are able to interface with the organizations they are a part of and edit members, events, finances, and documents. To accomplish this, we will use a client-server model. We will offer a client, the frontend React webapp, and for the backend use an ExpressJS server. The server will maintain a database of all the information Clava keeps using MongoDB. These modules can operate independently without failure but functionality is tied to them working together.

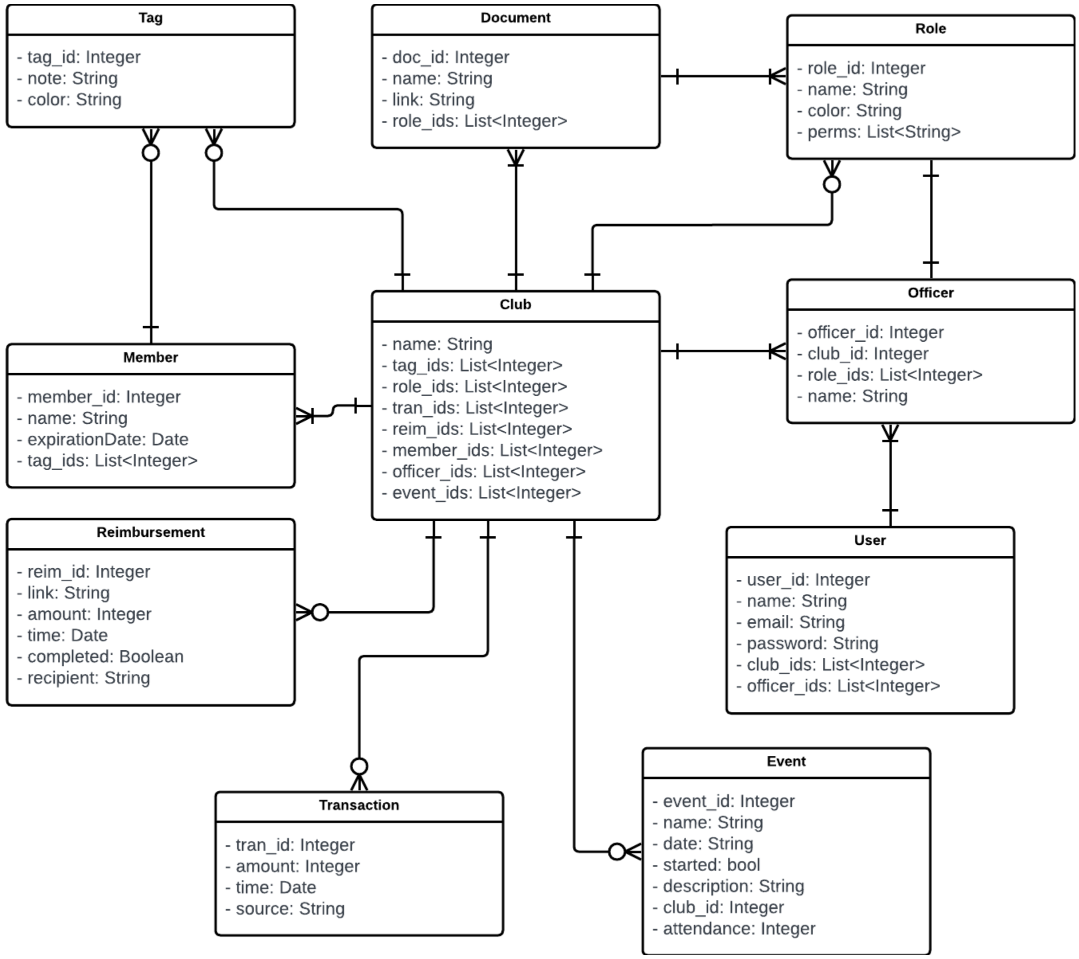
Components

Because our server will be written as a REST API with ExpressJS, we will communicate with it using HTTPS requests. Every user will open their own instance of the client, which will communicate with the server. Data is sent back and forth using a JSON format which will enable efficient, easy parsing. The server will be hosted using Docker and AWS.



1. eb Client
    a. Users will do all of their interaction using the web client.
    b. Whenever data must be displayed, edited, or updated, requests will be sent to the server.
    c. Functionality will be abstracted into various hubs, each of which will have their own web page. This allows for cleaner user interfaces, which is one of the project goals.
2. Server
    a. Whenever a request is received from the client, retrieve the correct resource and send it back with proper JSON formatting.
    b. In order to retrieve information, communicate with the Database.
    c. Follow REST conventions.
3. Database
    a. Whenever the server needs information, provide information.
    b. Follow MongoDB design patterns and make modifying the schema easy.
    c. Provide persistent storage to data regardless of Server status.

Backend UML



Design Issues

Functional Issues

- What information should be kept and displayed for each user?
  - Option 1: User ID, Name, Username, Password, Associated Club IDs, Associated Officer IDs, and Associated Member IDs (if they also happen to be members of other clubs).
  - Option 2: Only User ID, Name, Username, Password, Associated Club IDs, and Associated Officer IDs.

  Decision: **Option 2**. While storing the associated member IDs of each user has its benefits, namely being able to use this information for future features, the extra information will only take up valuable space in our database as Clava grows and is used by more organizations. Additionally, it also presents a possible personal data security risk, as we do not want officers of a club to be associated with their identity as a general member in another club, and complicates our design, as we want officers and members to be treated as separate entities. Furthermore, by only storing what is needed, we can keep the User class light and our design clear.

- How should officers modify member information?
  - Option 1: Allow a spreadsheet styled editing system (editable cells)
  - Option 2: Create a dialog that must be opened to modify information

  Decision: **Option 2**. This determines the member data is represented client side. A spreadsheet styled editing system would allow officers to easily modify member information, similar to google sheets, without having to open dialogs for everything. However, this may result in accidental or unintentional modification of potentially important information. Additionally, we are attempting to distinguish ourselves from google sheets by offering something that specifically is designed as a database, not a spreadsheet. Creating dialogs that allow officers to modify information solves this issue. To prevent having to open too many, we can allow the selection of multiple members and modify them simultaneously.

- How should users login to Clava?
  - Option 1: Through a Clava username and password tied to their account
  - Option 2: Through their university portal/login system
  - Option 3: Through other third party authentication, such as Google

  Decision: **Option 1**. We will use both custom user authentication in order to provide the user as much flexibility as possible while retaining our own control of the data. Using university login, while likely preferred by organizations due to security, is not a feasible option as it would require us to support individual universities, in addition to necessitating access to APIs that may not exist. Using third party authentication may happen in the future but for now we do not want to use services like Firebase that control the data. Doing so would allow for users to use existing accounts, and we would be able to wrap our data through Firebase's security. However, that would give up having complete control of the data, and would introduce additional complexity we don't want at the moment.

Non-Functional Issues

- Which programming language should we use?
    - Option 1: Typescript
    - Option 2: Javascript
    - Option 3: Java

  Decision: **Option 1**. Javascript is a good choice as all modern web browsers have great support for the language. Furthermore, many optimizations have been made for language to be extremely fast. Finally, the language features all center around providing support for web applications. However, it lacks many object oriented support and its class keyword provides very little. Java is extremely object oriented and provides the team extensive ways to collaborate through that design. However, it lacks the web support provided by javascript. As a result, we are choosing to use Typescript for both our frontend and backend as it is the best of both worlds by directly compiling into Javascript and providing necessary object oriented features.

- How do we want to handle events?
    - Option 1: React hooks, one class which allows other classes to send and subscribe (listen for) events.
    - Option 2: Hierarchy system for classes.

  Decision: **Option 1**. A hierarchy system would involve traditional React state variables, passed down from top to bottom. If we were to use this system, it would allow for faster prototyping, as it is often more intuitive to keep track of data this way. However, the longer we spend time developing, the harder keeping track of dependencies will become. To solve this issue, we will use React hooks instead. This will allow for a central "event manager" class that all classes send events to and listen for events from. This allows for organization of dependencies, at the cost of additional time spent setting up such a class and having to "subscribe" for each additional class created.

- How do we want to do CSS and styling?
    - Option 1: Use Bootstrap
    - Option 2: Use Tailwind

  Decision: **Option 2**. Both of these options allow inline CSS styling, which is desirable where possible in order to condense the amount of css we will have to write. When necessary, we will write reusable stylesheets to apply to components. This will have to be done in order to prioritize readability, as doing *all* of the styling solely inline is not feasible. Bootstrap is a competitive option because it is the more popular library, and has more complete documentation and digital resources. Despite this, we will choose Tailwind as we will be using the MUI collection of React components. Bootstrap is essentially composed of two parts, the component library and the styling, and we will be replacing those with two separate libraries that each do their specific job more effectively.

Design Details

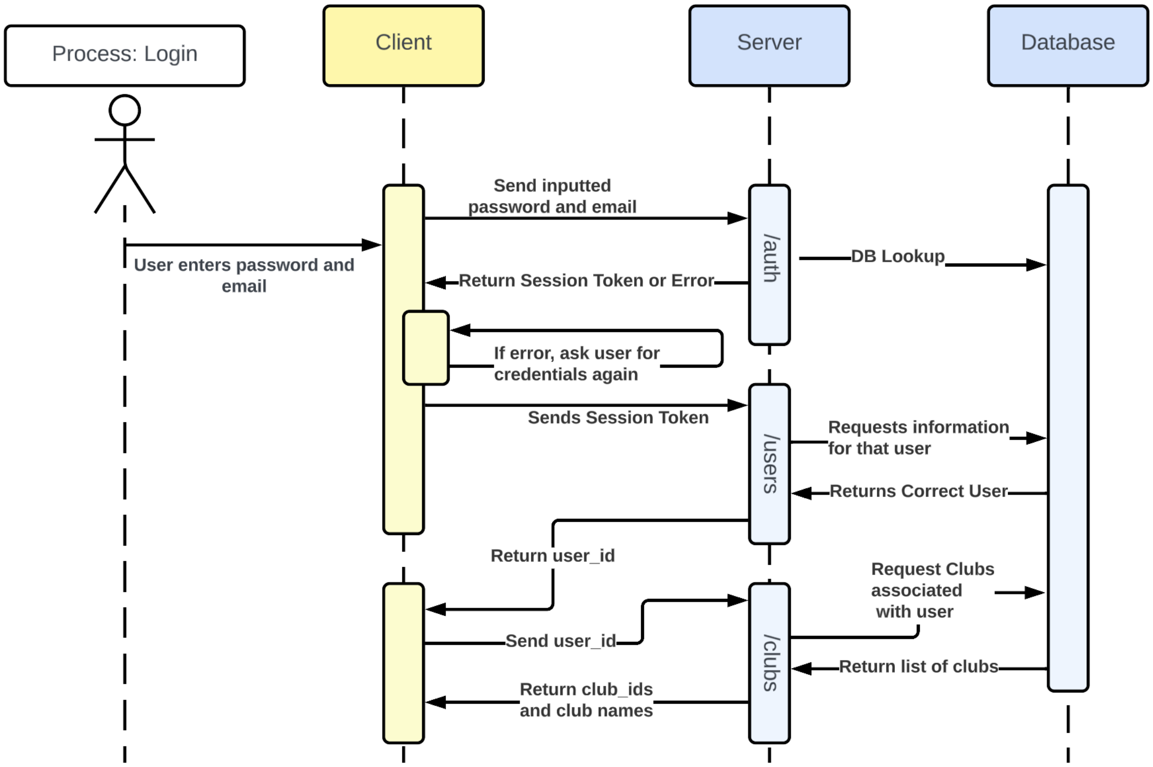
Terminology

User	An account that is registered with Clava. A **User** can be an **Officer** in one or multiple different college organizations.
Officer	An **Officer** of an organization has access to the organization's Clava **Hubs** while holding management permissions. **Users** can become **Officers** for organizations, but there is no interaction between two organizations even if you are an **Officer** for both. If you are an **Officer** for an organization, you are also a **Member** of that organization by default, but there is no interaction between your **Officer** id and your **Member** id. **Officers** have access to the various **Hubs** for an **Organization**, and can be assigned **Roles** to modify their permissions.
Member	A **Member** is a collection of data representing a **Member** of an organization. They do not have accounts and are not associated with accounts. If they are an officer, you cannot access their **Officer** or **User** id from their **Member** id. **Members** can be assigned **Tags**, but not **Roles**.
Organization	Every **Organization** (also referred to as 'club') has its own collection of **Hubs**, with organization-specific data.
Hub	A specific page on the site with a specific purpose. Each **Hub** contains organization-specific data. For example, every **Organization** will have an event hub for managing events, a member hub for managing members, and so on.
Role	A set of permissions you can assign to an officer. Ex: "Treasurer" - can access and edit finance related information.
Tag	A string/note you can assign to a member. Ex: "Paid Dues"
Document	A link to an external resource used by the club for the purposes of documentation. For example, both a youtube link and a google doc link would be considered documents. Each **Document** has a Clava-specific name attached to it with access managed by permissions.

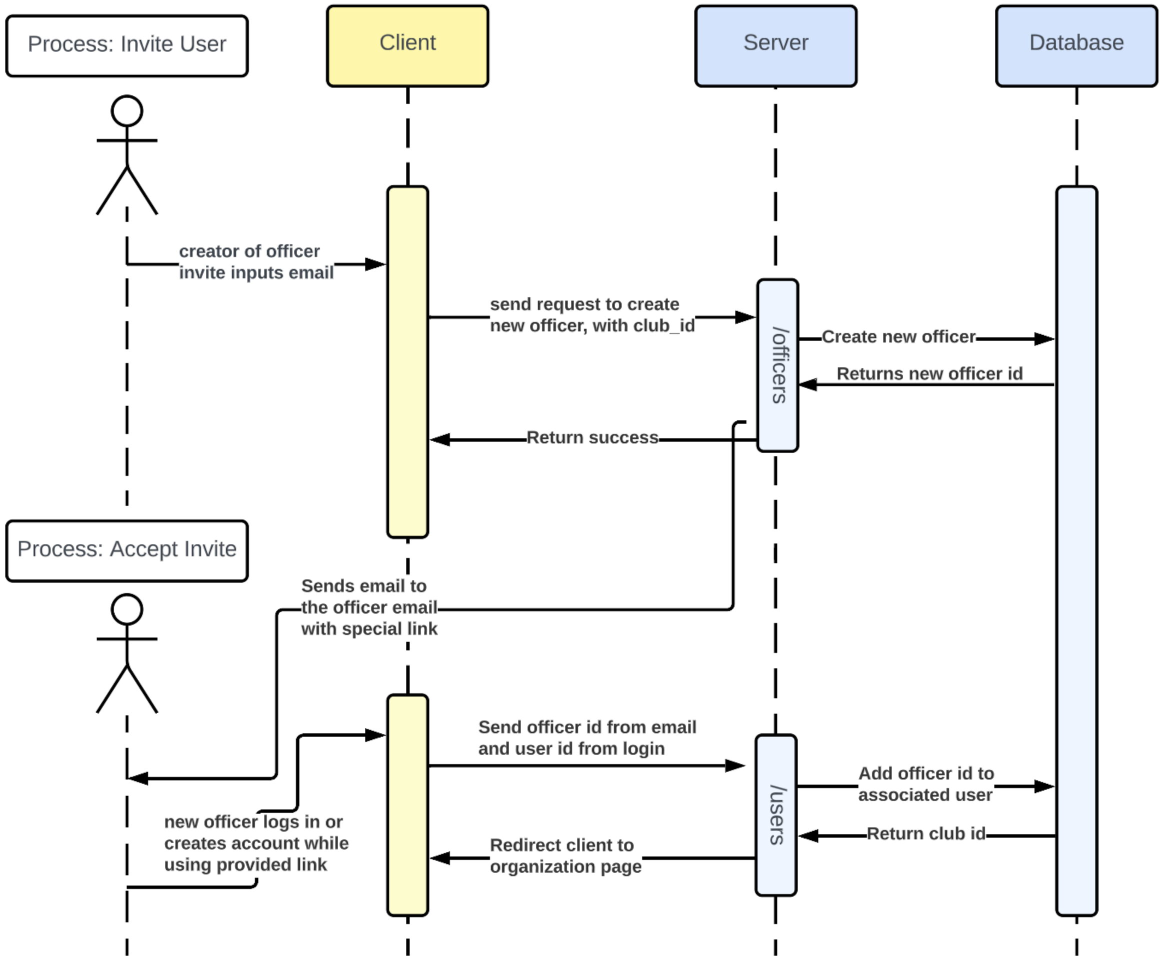
Sequence Diagrams for Common Processes

Below are sequence diagrams demonstrating the component interaction for common processes that might occur.

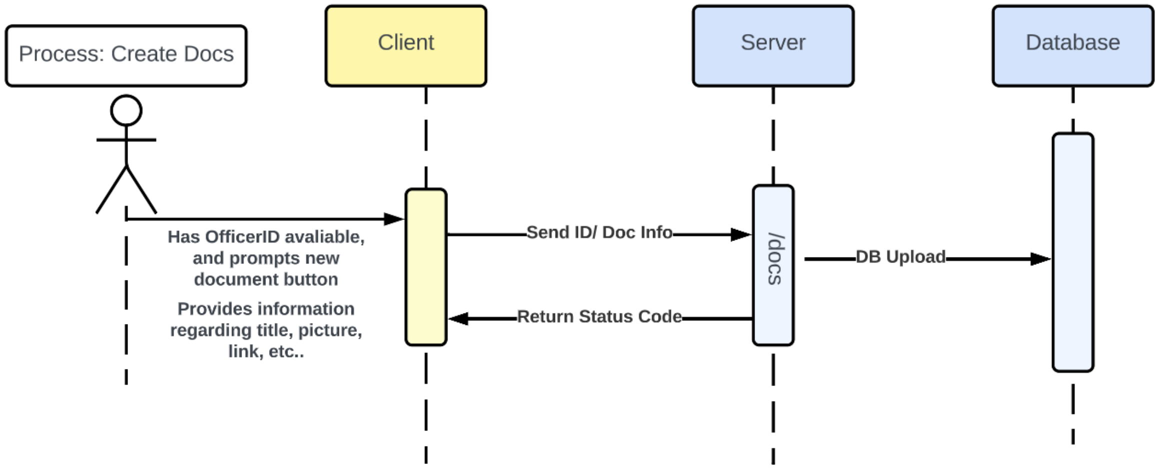
1. Sequence when a user logs in


Process: Login
Client
Server
Database
Send inputted password and email
User enters password and email
/auth
DB Lookup
Return Session Token or Error
If error, ask user for credentials again
Sends Session Token
/users
Requests information for that user
Returns Correct User
Return user_id
Request Clubs associated with user
Send user_id
/clubs
Return list of clubs
Return club_ids and club names

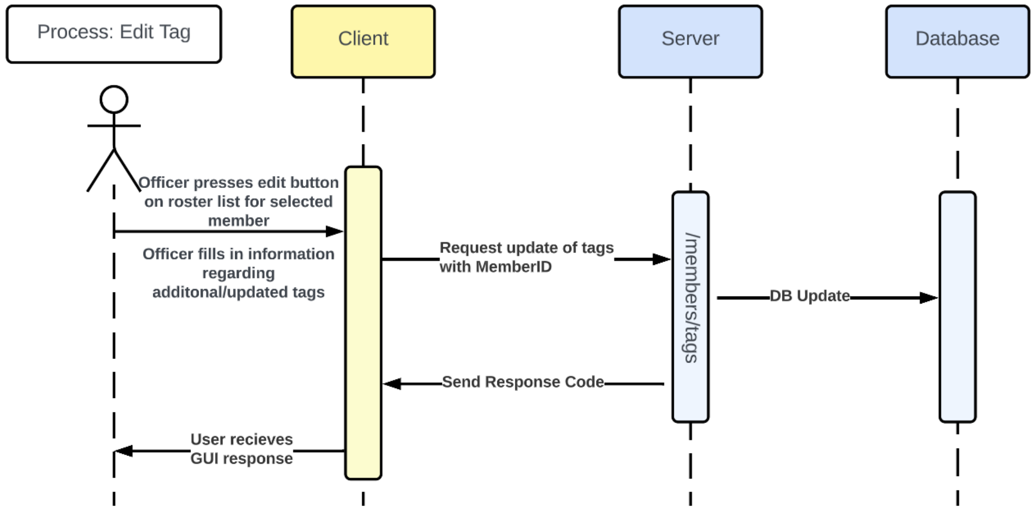
2. Sequence when a user is invited to be an officer for an organization


Process: Invite User
Client
Server
Database
creator of officer invite inputs email
send request to create new officer, with club_id
/officers
Create new officer
Returns new officer id
Return success
Process: Accept Invite
Sends email to the officer email with special link
Send officer id from email and user id from login
/users
Add officer id to associated user
Return club id
new officer logs in or creates account while using provided link
Redirect client to organization page

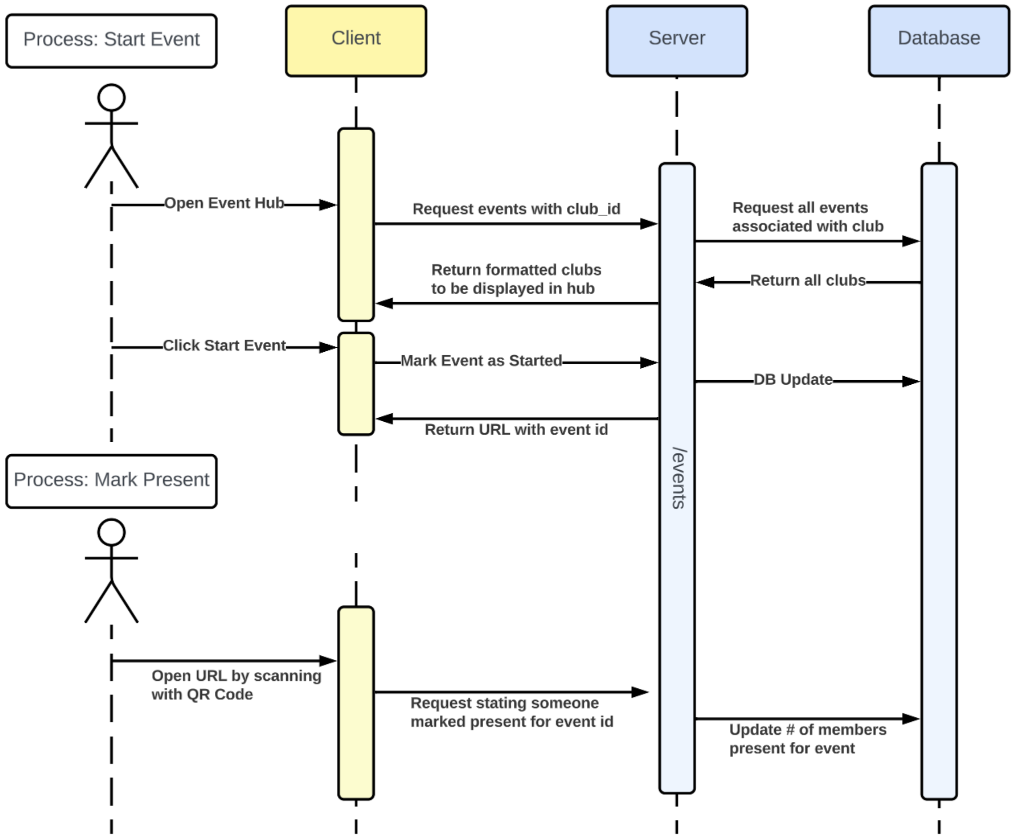
3. Sequence when an officer creates documentation


Process: Create Docs
Client
Server
Database
Has OfficerID avaliable, and prompts new document button
Provides information regarding title, picture, link, etc..
Send ID/ Doc Info
/docs
DB Upload
Return Status Code

4. Sequence when an officer edits a members' tags

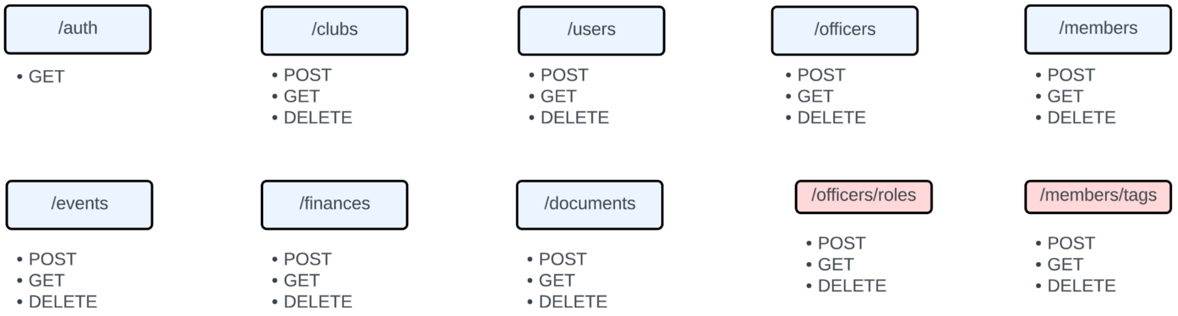

Process: Edit Tag
Client
Server
Database
Officer presses edit button on roster list for selected member
Officer fills in information regarding additonal/updated tags
Request update of tags with MemberID
/members/tags
DB Update
Send Response Code
User recieves GUI response

5. Sequence for an event starting and a member registering as present


Process: Start Event
Client
Server
Database
Open Event Hub
Request events with club_id
Request all events associated with club
Return formatted clubs to be displayed in hub
Return all clubs
Click Start Event
Mark Event as Started
DB Update
Return URL with event id
/events
Process: Mark Present
Open URL by scanning with QR Code
Request stating someone marked present for event id
Update # of members present for event

Server (Backend) Design

API Design



Our API will be designed with the REST philosophy in mind. Every endpoint represents a resource, and the verbs will allow the client to communicate its desire to the server. The supported verbs with corresponding resource is as follows:

Notes:

- "Users" in this context refer to users of the API.
- Authentication is supported using the given api_token.
- Any verb not listed with no parameters can be assumed to return 404 (Not Found) or 403 (Restricted Access).
  - Ex: GET on /users without params will be 403 always.
  - Ex: GET on /auth without params will be 404 always.

Route	**Verb**	**Params**	**Description**
/auth	GET	?email* ?password*	User is able to request the server for an API Session Token given correct params. Server will return either status code 200 with an API Session Token or the correct HTTP error response depending on the context. *Both parameters are in plaintext. We use HTTPS.
/clubs	POST	?club_name ?club_image* ?club_desc* ?api_token	User is able to request the server to create a new club by providing a club_name. Users may also include club_image, club_desc but is not required. Server will return either status code 200 with the newly created club_id or the correct HTTP error response depending on the context.

			*Not necessary for creation.
/clubs	POST	?club_id ?club_name* ?club_image* ?club_desc* ?api_token	Users are able to request the server to update a club’s information by providing a club_id. Users are expected to provide at least one of the editable fields, but is not required. In the event none of the fields are provided, no change will occur. Server will return either status code 200 with the club_id or the correct HTTP error response depending on the context. *Able to update.
/clubs	GET	?user_id ?api_token	Users are able to request the server to give them a list of clubs that the corresponding user_id is associated with. Server will return either status code 200 with a subscriptable list of club_ids or the correct HTTP error response depending on the context.
/clubs	DELETE	?club_id ?api_token	Users are able to request the server to delete a club with a given club_id. Server will return either status code 200 or the correct HTTP error response depending on the context.
/users	POST	?email ?name ?password	Users are able to request the server to create a new user account given the required params. Server will return either status code 200 with the newly created $user_id or the correct HTTP error response depending on the context. Error may include when an email is already taken by another user.
/users	POST	?user_id ?new_password ?api_token	Users are able to request the server to change the password of the user account with a given new_password. Server will return either status code 200 or the correct HTTP error response depending on the context.

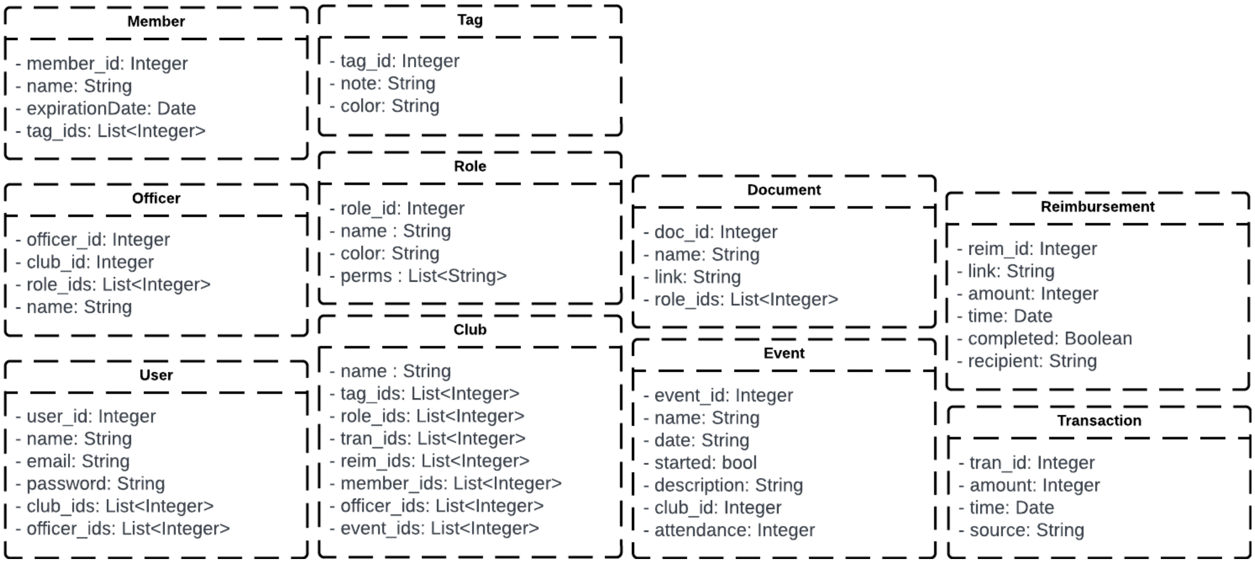
/users	POST	?user_id ?new_name ?api_token	Users are able to request the server to change the name of the user account with a given new_name. Server will return either status code 200 or the correct HTTP error response depending on the context.
/users	GET	?user_id ?api_token	Users are able to get all necessary information about a particular user by requesting the server. Token must be associated with the user_id. Server will return either status code 200 with name, email, clubs_ids, and officer_ids or the correct HTTP error response depending on the context.
/users	DELETE	?user_id ?api_token	Users are able to request the server to delete a user account with a given user_id. Server will return either status code 200 or the correct HTTP error response depending on the context.
/officers	POST	?user_id ?club_id ?role_ids ?name ?api_token	Users are able to request the server to create an officer account with a given user_id, club_id, roles_id, and name. If a member with that email does not already exist, a member is also created, identically to if a POST request was sent to /members. Server will return either the newly created officer_id or the correct HTTP error response depending on the context.
/officers	POST	?officer_id ?new_role_id ?api_token	Users are able to request the server to add the provided role to the given officer_id. Server will return either status code 200 or the correct HTTP error response depending on the context.
/officers	GET	?club_id ?api_token	Users are able to request a list of officers for a club. Server will either respond with a subscriptable object with all officers or the correct HTTP error response depending on the context.

/officers	DELETE	?officer_id ?api_token	Users are able to request the server to delete an officer account with a given officer_id. Server will return either status code 200 or the correct HTTP error response depending on the context.
/members	POST	?club_id ?name ?email ?api_token	Users are able to request the server create a new member with the provided name and email. Server will return either the created member_id or the correct HTTP error response depending on context.
/members	GET	?club_id ?api_token	Users are able to request the server return all available members. Server will return either list of member_ids or the correct HTTP error response depending on context.
/members	DELETE	?member_id ?club_id ?api_token	Users are able to request the server delete a member given an member_id. Server will return status code 200 or the correct HTTP error response depending on context.
/events	POST	?name ?date ?des ?club_id ?api_token	Users are able to create/update an event by providing the required parameters. Server will return either status code 200 and the associated event_id or correct HTTP error responses depending on the context.
/events	POST	?event_id ?started ?api_token	Users are able to update the status of the event started. Server will return either status code 200 or the correct HTTP error response depending on the context.
/events	POST	?event_id ?amount ?api_token	Users are able to update the attendance count to a certain amount. Server will return either status code 200 or the correct HTTP error response depending on the context.
/events	GET	?club_id	Users are able to get all the events associated

		?api_token	with a given club. Server will return either status code 200 and a subscriptable list of all events or the correct HTTP error response depending on the context.
/events	GET	?event_id ?api_token	Users are able to get specific events from the server. Server will return either status code 200 and the specific event or the correct HTTP error responses depending on the context.
/events	DELETE	?event_id ?api_token	Users are able to delete a specific event. Server will return either status code 200 or the correct HTTP error responses depending on the context.
/finances	POST	?amount ?time ?source ?api_token	Users are able to create/update a transaction by providing the required parameters. Server will return either status code 200 and the associated tran_id or the correct HTTP error response depending on the context.
/finances	POST	?amount ?time ?link ?recipient ?api_token	Users are able to create/update a reimbursement by providing the required parameters. Server will return either status code 200 and the associated reim_id or the correct HTTP error response depending on the context.
/finances	POST	?reim_id ?api_token	Users are able to mark a reimbursement as completed. Server will return either status code 200 or the correct HTTP error response depending on the context.
/finances	GET	?club_id ?api_token	Users are able to get a list of all transactions and reimbursements associated with a club. Server will return either status code 200 and a subscriptable list of transactions/reimbursements or the correct HTTP error response depending on the context.

/finances	DELETE	?reim_id ?api_token	Users are able to delete a reimbursement. Server will return either status code 200 or the correct HTTP error response depending on the context.
/finances	DELETE	?tran_id ?api_token	Users are able to delete a transaction. Server will return either status code 200 or the correct HTTP error response depending on the context.
/documents	POST	?club_id ?name ?link ?role_ids ?api_token	Users are able to create a document by providing the required parameters. Server will return either status code 200 and the associated doc_Id or the correct HTTP error response depending on the context.
/documents	POST	?doc_id ?role_ids ?api_token	Users are able to update document permissions. Server will return either status code 200 or the correct HTTP error responses depending on the context.
/documents	GET	?club_id ?api_token	Users are able to get specific documents from the server. Server will return either status code 200 and the list of documents with information in the schema or the correct HTTP error responses depending on the context.
/documents	DELETE	?doc_id ?api_token	Users are able to delete a specific document. Server will return either status code 200 or the correct HTTP error responses depending on the context.

Database Schema



Member

- member_id: Integer
- name: String
- expirationDate: Date
- tag_ids: List<Integer>

Officer

- officer_id: Integer
- club_id: Integer
- role_ids: List<Integer>
- name: String

User

- user_id: Integer
- name: String
- email: String
- password: String
- club_ids: List<Integer>
- officer_ids: List<Integer>

Tag

- tag_id: Integer
- note: String
- color: String

Role

- role_id: Integer
- name : String
- color: String
- perms : List<String>

Club

- name : String
- tag_ids: List<Integer>
- role_ids: List<Integer>
- tran_ids: List<Integer>
- reim_ids: List<Integer>
- member_ids: List<Integer>
- officer_ids: List<Integer>
- event_ids: List<Integer>

Document

- doc_id: Integer
- name: String
- link: String
- role_ids: List<Integer>

Event

- event_id: Integer
- name: String
- date: String
- started: bool
- description: String
- club_id: Integer
- attendance: Integer

Reimbursement

- reim_id: Integer
- link: String
- amount: Integer
- time: Date
- completed: Boolean
- recipient: String

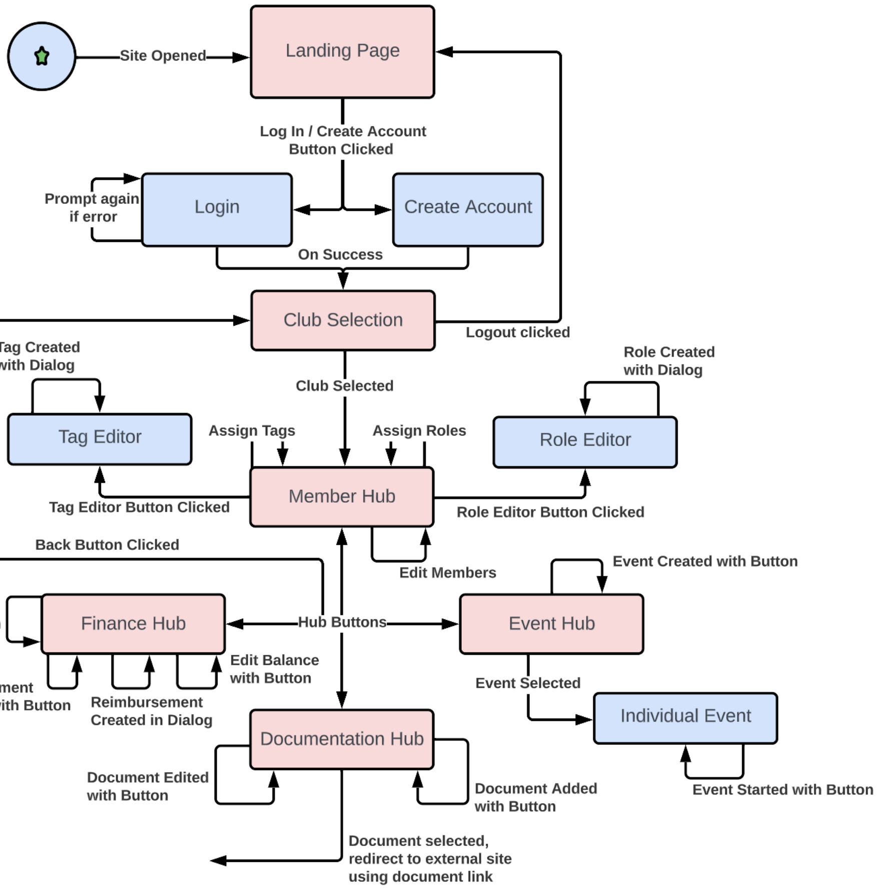
Transaction

- tran_id: Integer
- amount: Integer
- time: Date
- source: String

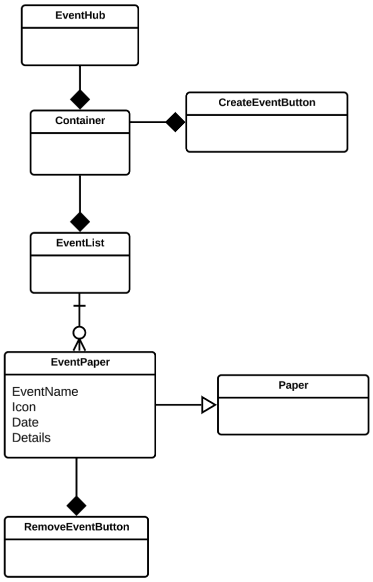
Web (Frontend) Design

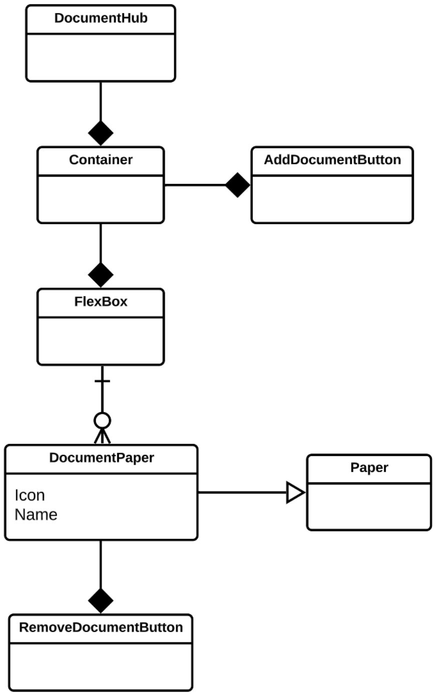
Overview and State Diagram

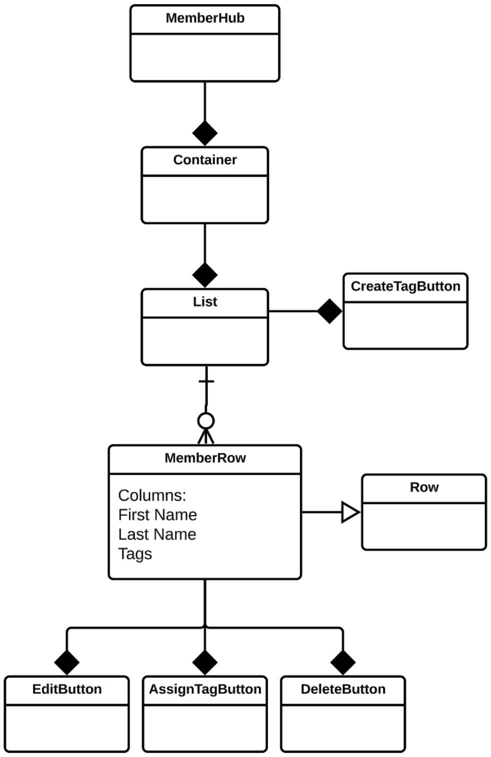
For our user interface, we will use the Material UI library for consistent, easy components. MUI also allows us to apply a theme to our entire project using a color palette and set of fonts. Styling will be applied through this customization, stylesheets, and Tailwind. Below is a high level overview of how the user will navigate between the pages. Below that are UML diagrams detailing the specific pages more in depth.

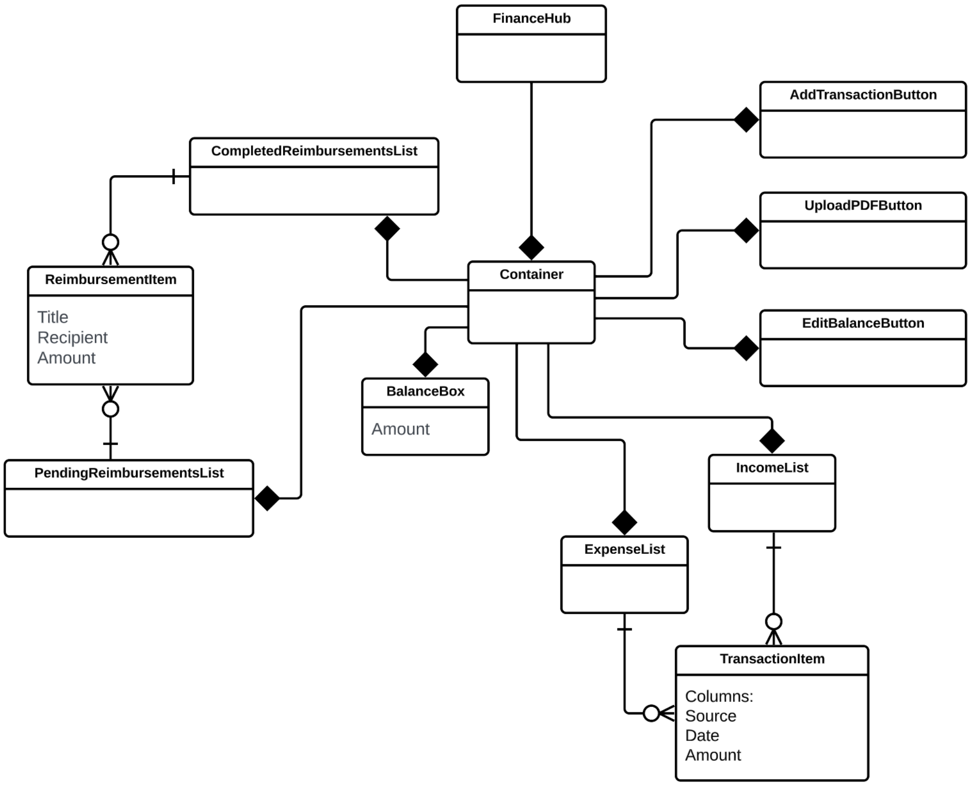

Site Opened
Landing Page
Log In / Create Account Button Clicked
Prompt again if error
Login
Create Account
On Success
Club Selection
Logout clicked
Tag Created with Dialog
Club Selected
Role Created with Dialog
Tag Editor
Assign Tags
Assign Roles
Role Editor
Member Hub
Tag Editor Button Clicked
Role Editor Button Clicked
Back Button Clicked
Edit Members
Event Created with Button
Transaction Created with Button
Finance Hub
Hub Buttons
Event Hub
Edit Balance with Button
Reimbursement Approval with Button
Reimbursement Created in Dialog
Event Selected
Individual Event
Documentation Hub
Document Edited with Button
Document Added with Button
Event Started with Button
Document selected, redirect to external site using document link

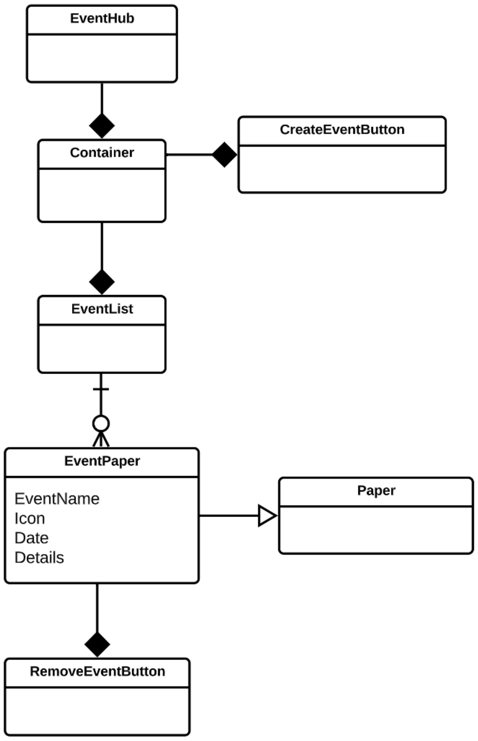
Frontend UML




DocumentHub
Container
AddDocumentButton
FlexBox
DocumentPaper
Icon
Name
Paper
RemoveDocumentButton


MemberHub
Container
List
CreateTagButton
MemberRow
Columns:
First Name
Last Name
Tags
Row
EditButton
AssignTagButton
DeleteButton


FinanceHub
AddTransactionButton
CompletedReimbursementsList
UploadPDFButton
Container
ReimbursementItem
Title
Recipient
Amount
EditBalanceButton
BalanceBox
Amount
PendingReimbursementsList
IncomeList
ExpenseList
TransactionItem
Columns:
Source
Date
Amount

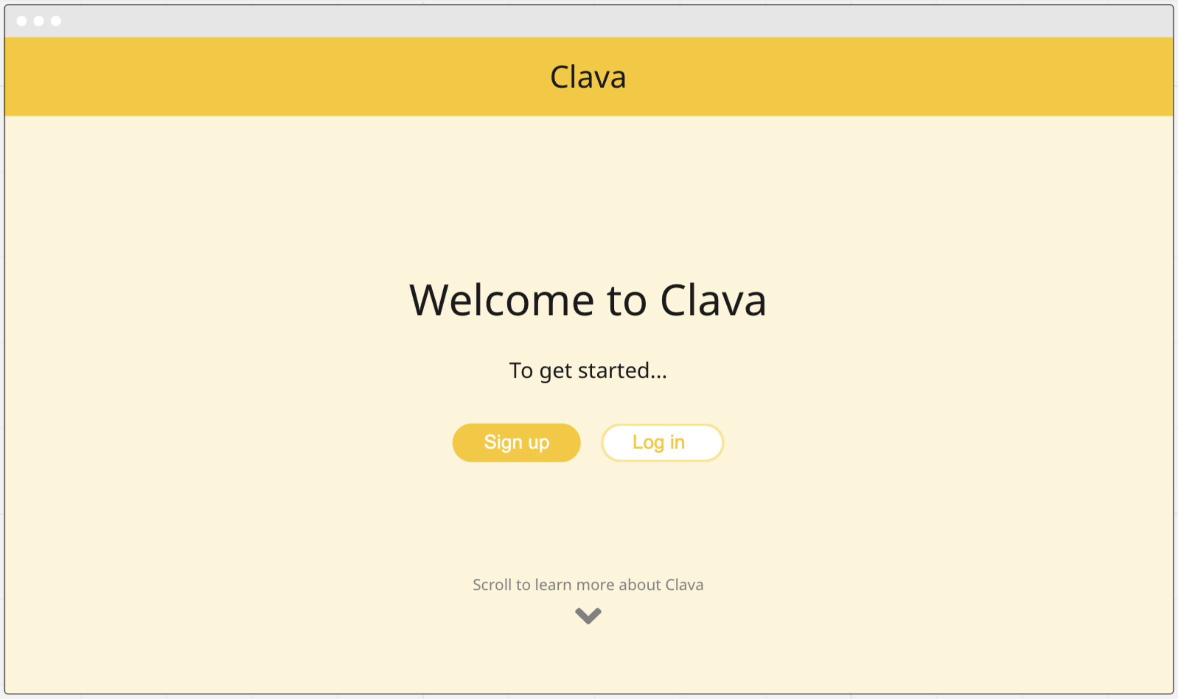


Webpages and UI Mockups

Above is a state diagram that demonstrates the multiple pages Clava will have.

Below are mockups of the majority of those pages.

Pre-Login/Sign up



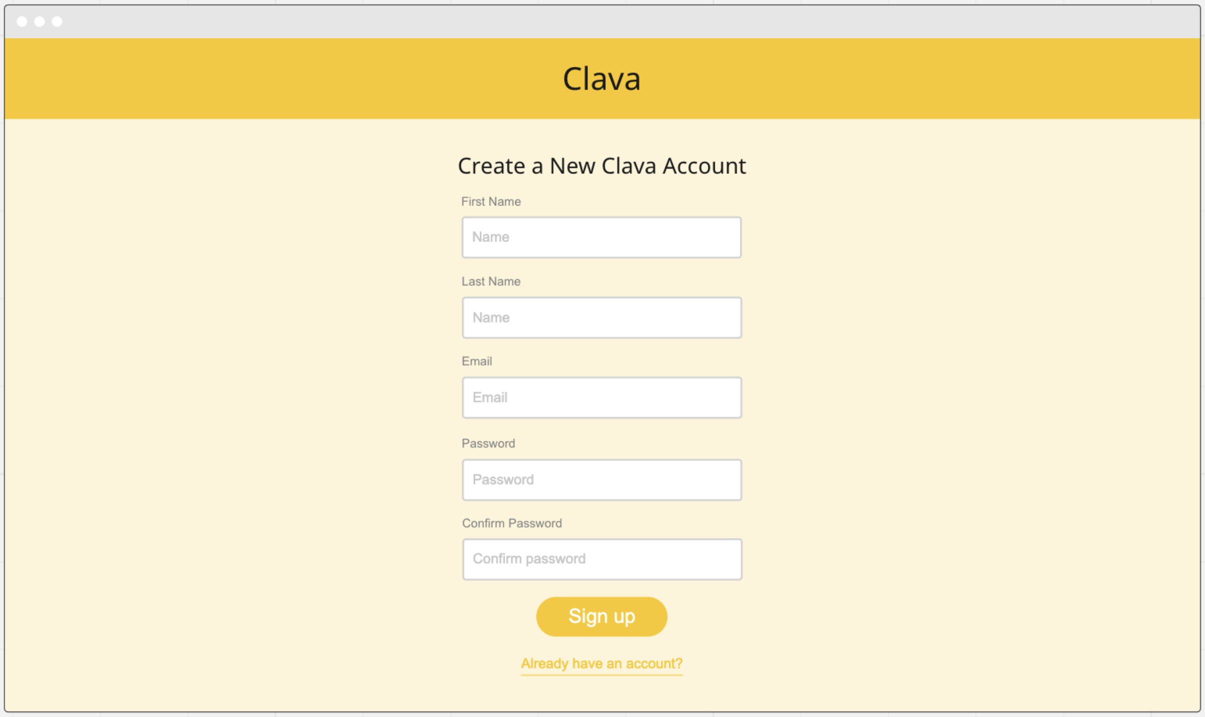
Clava

Welcome to Clava

To get started...

Scroll to learn more about Clava

Sign up



Clava

Create a New Clava Account

First Name

Name

Last Name

Name

Email

Email

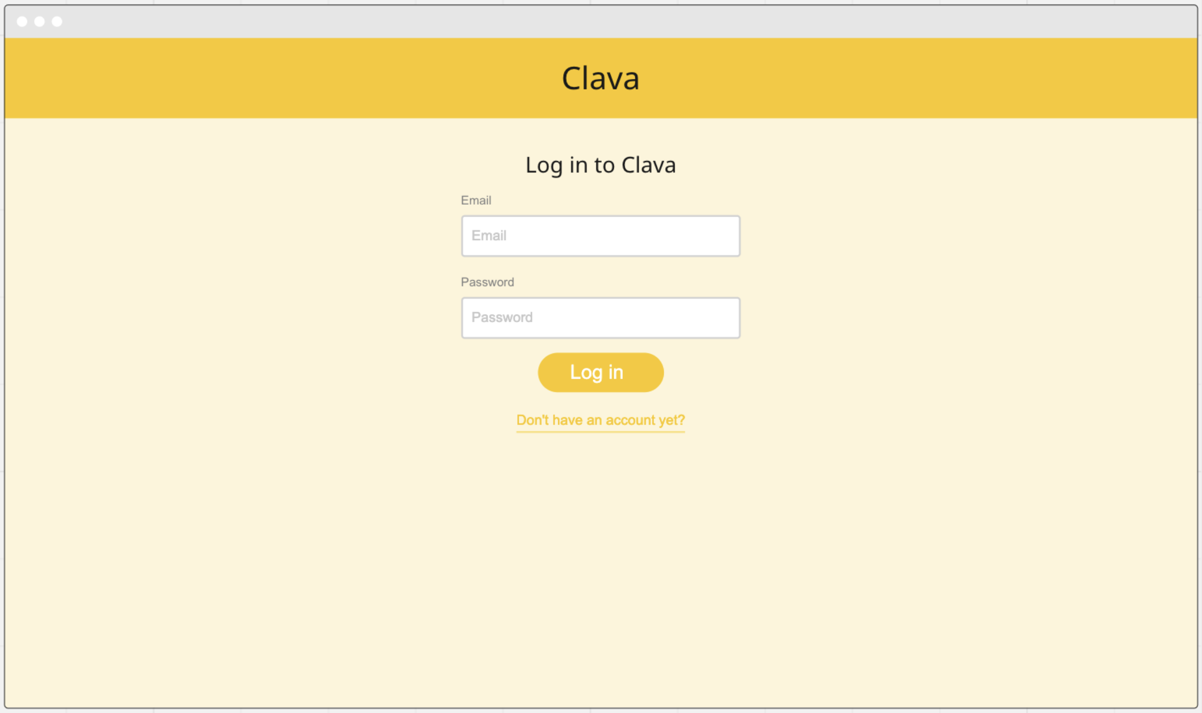
Password

Password

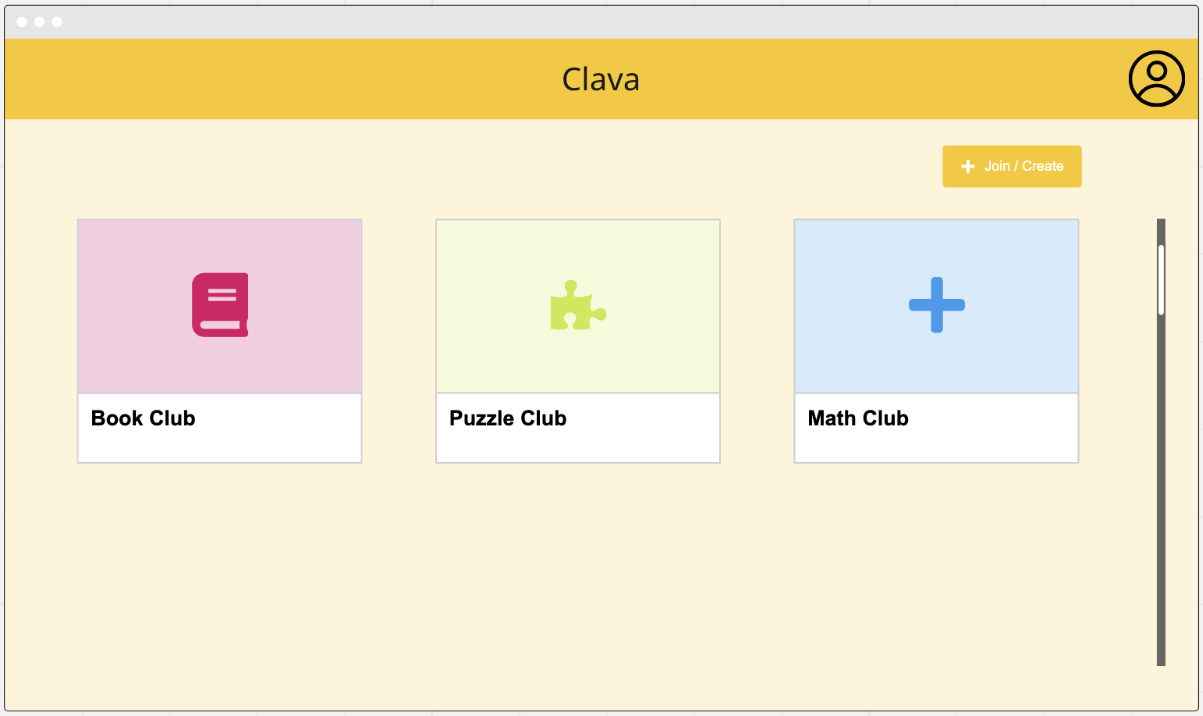
Confirm Password

Confirm password

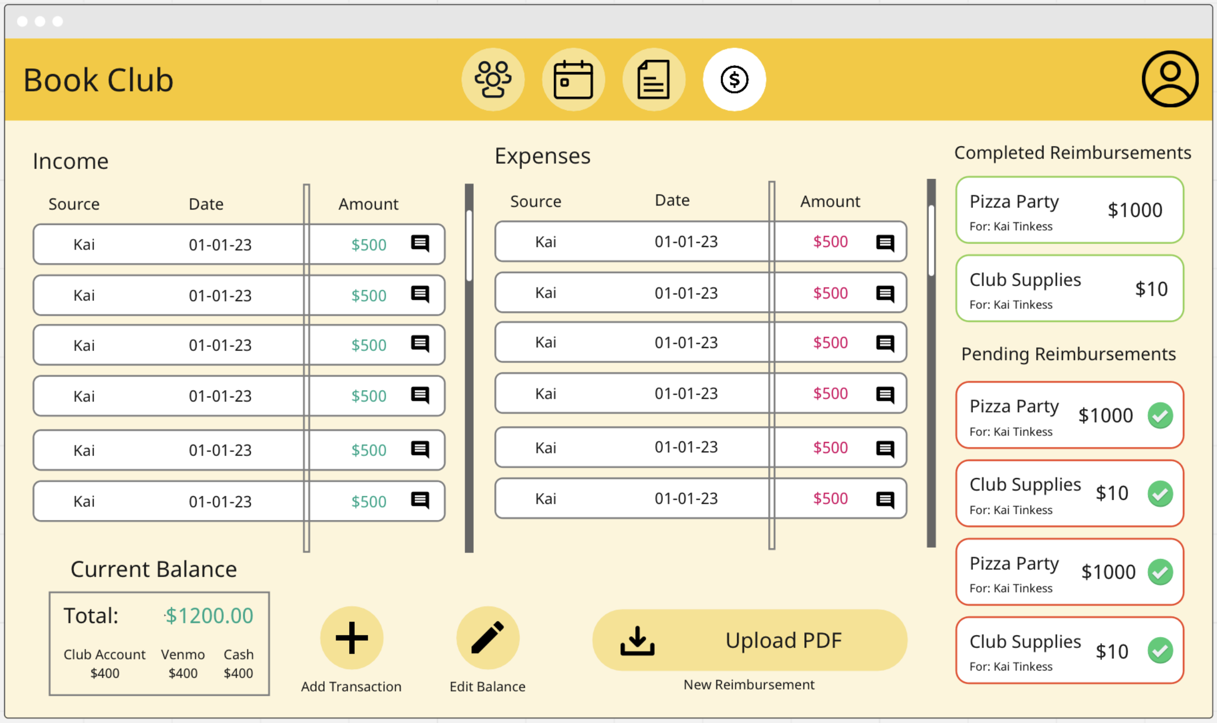
Login



Home page: Clubs (Browser)



Finance Hub (Browser)



Book Club

Income

Source	Date	Amount
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500

Expenses

Source	Date	Amount
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500
Kai	01-01-23	$500

Completed Reimbursements

Pizza Party $1000
For: Kai Tinkess

Club Supplies $10
For: Kai Tinkess

Pending Reimbursements

Pizza Party $1000 ✓
For: Kai Tinkess

Club Supplies $10 ✓
For: Kai Tinkess

Pizza Party $1000 ✓
For: Kai Tinkess

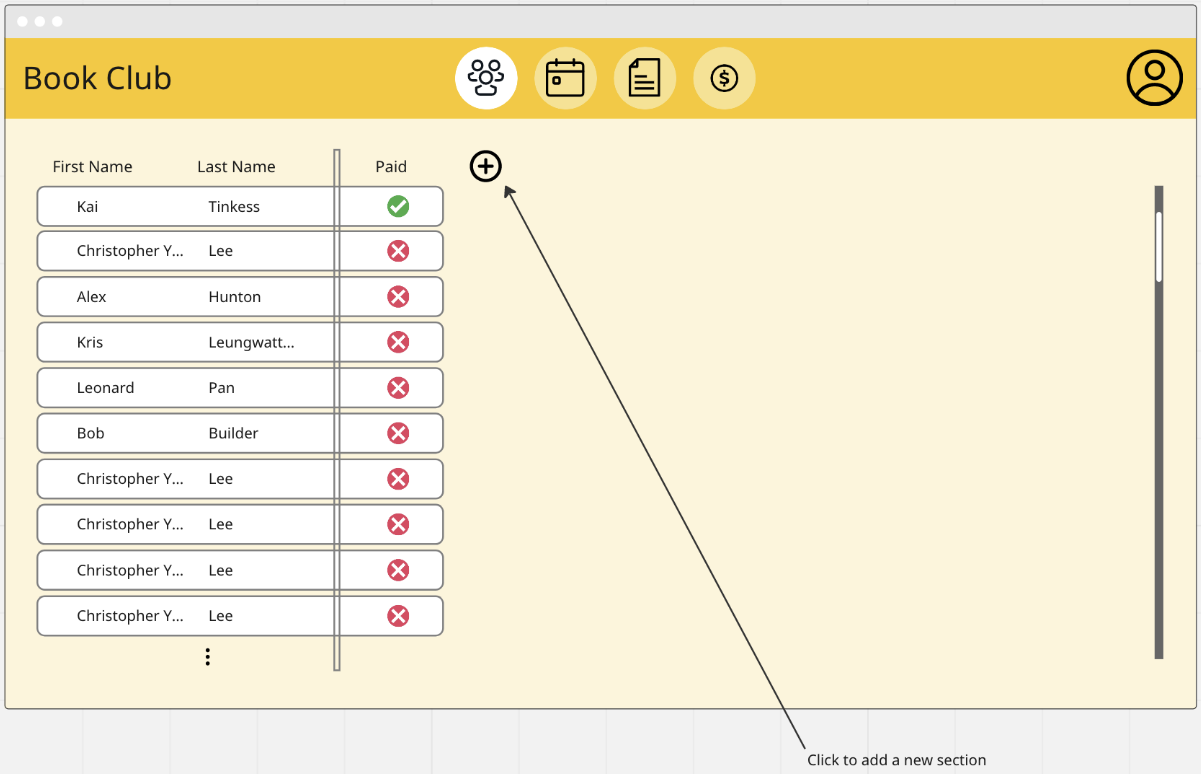
Club Supplies $10 ✓
For: Kai Tinkess

Current Balance

Total: $1200.00

Club Account	Venmo	Cash
$400	$400	$400

Member Management Hub (Browser)



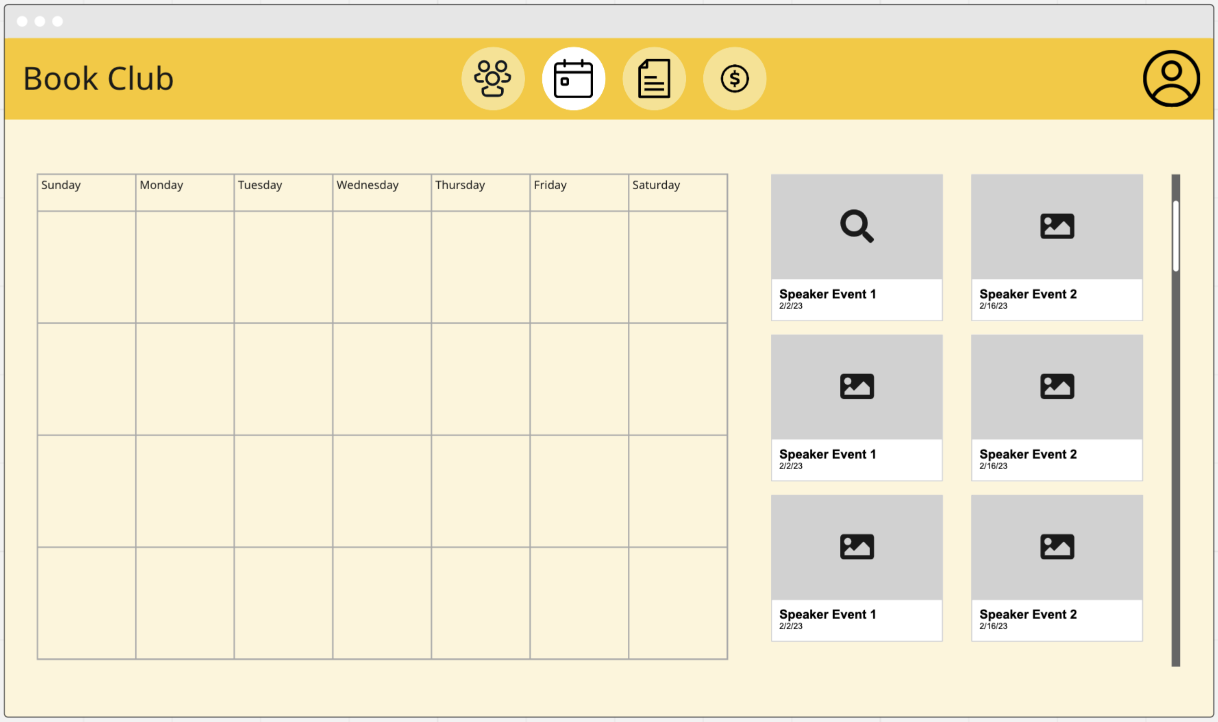
Book Club

First Name	Last Name	Paid
Kai	Tinkess	✓
Christopher Y...	Lee	✗
Alex	Hunton	✗
Kris	Leungwatt...	✗
Leonard	Pan	✗
Bob	Builder	✗
Christopher Y...	Lee	✗
Christopher Y...	Lee	✗
Christopher Y...	Lee	✗
Christopher Y...	Lee	✗

⋮

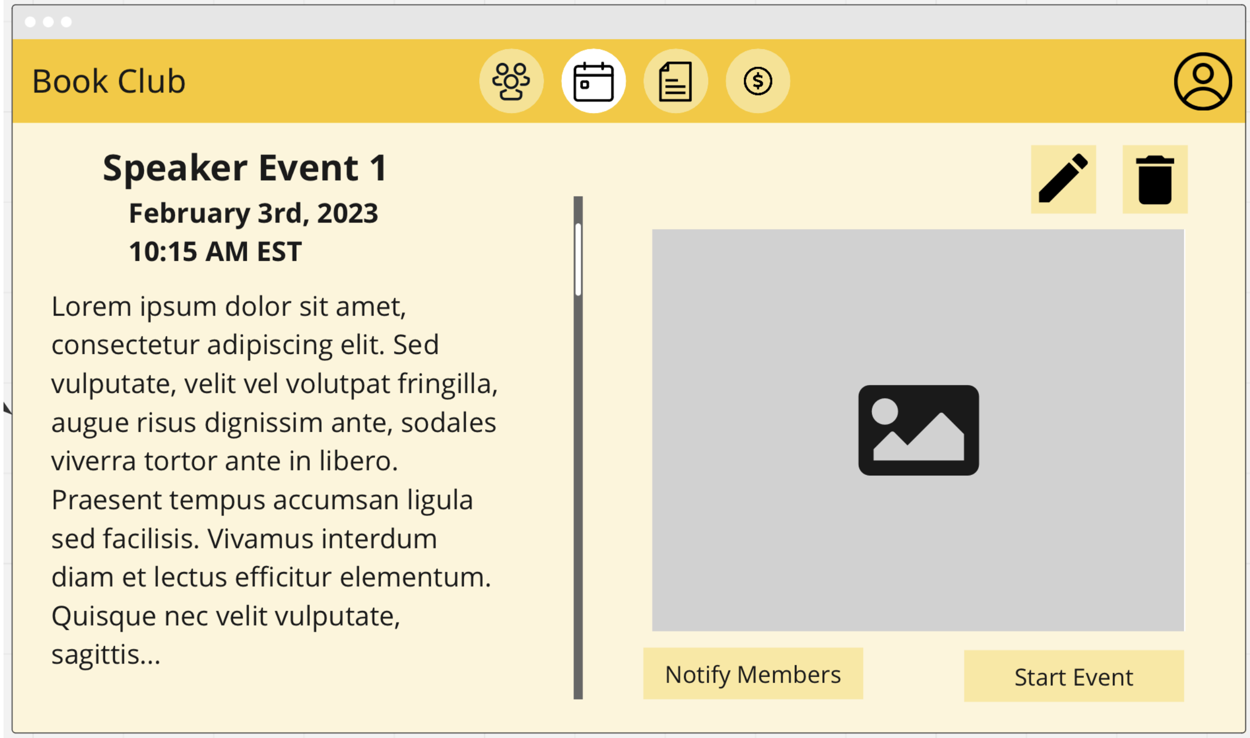
Click to add a new section

Events Hub (Browser)



Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday

Events Hub (Individual Event)

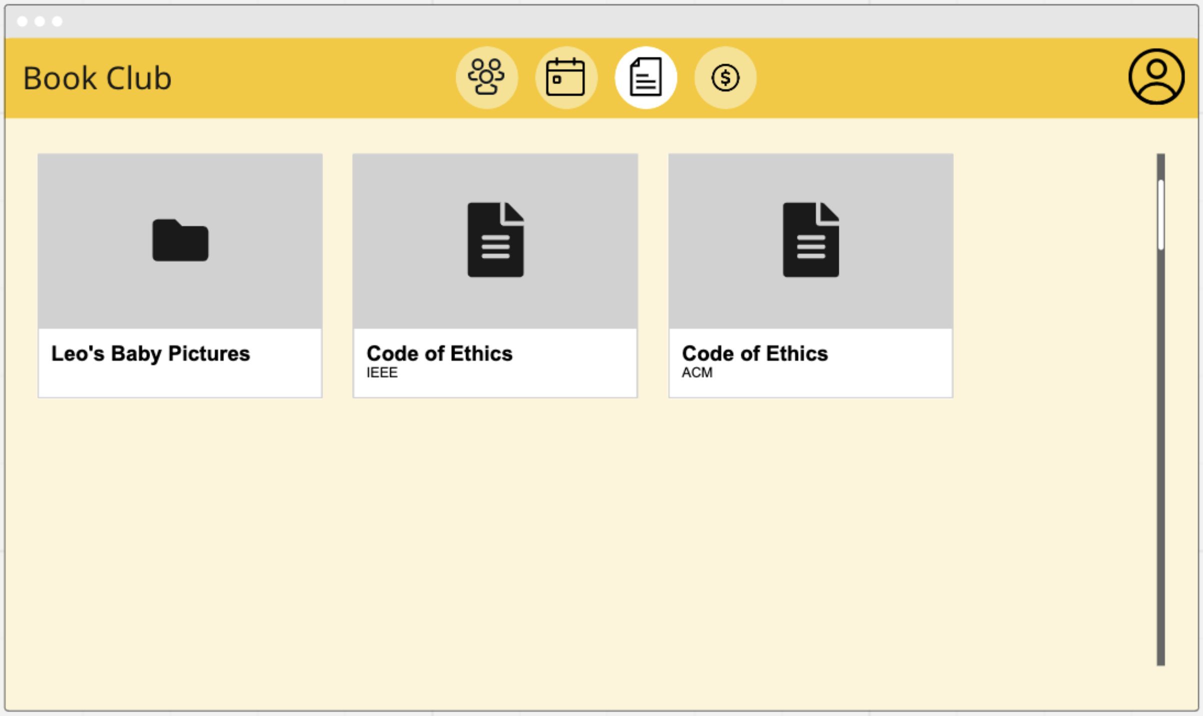


Speaker Event 1

February 3rd, 2023
10:15 AM EST

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed vulputate, velit vel volutpat fringilla, augue risus dignissim ante, sodales viverra tortor ante in libero. Praesent tempus accumsan ligula sed facilisis. Vivamus interdum diam et lectus efficitur elementum. Quisque nec velit vulputate, sagittis...

Documents Hub (Browser)


Book Club
Leo's Baby Pictures
Code of Ethics
IEEE
Code of Ethics
ACM