System Design Document

Team Citrus:

Group Members	Student Number
Ishaan Upadhyay	1008331636
Daniel Grishanov	1008158958
Miraj Ismail	1008464065
Matthew Mao	1008405673
Aaliyah Jaleel	1008060385
Sana Alia	1007529890
Shinza Ahmed	1008263567

Updated 2023-07-21

Table of Contents:

CRC Cards

We preface this design document by stating that we are not using an object-oriented development model. The following CRC cards are meant to illustrate our data model and routing system, but do not correspond with actual objects that will be instantiated and used. There are a few actual objects, which are indicated with an asterisk on the class name.

Template:

Class name:	
Parent class (if any**):** **Subclasses:**	
Responsibilities: ●	**Collaborators:** ●

Class name: Pool*	
Parent class (if any**): N/A** **Subclasses (**if any**): N/A**	
Responsibilities: ● Connecting to our PostgreSQL database, which is hosted on CockroachDB ● Executing SQL queries for the purpose of returning or updating data	**Collaborators:** ●

Class name: Experience	
Parent class: N/A **Subclasses:**	
Responsibilities: ● Knows internal ID?, name of experience, location, description, organizer, start time, end time, event category, tags?, max capacity, current attendance, list of users attending, list of users interested?,	**Collaborators:** ● Users ● Organizer

Class name: User

Parent class (if any**):**
Subclasses:

Responsibilities:	Collaborators:
● Knows username, hashed password, email, phone number, social media links, subscription status, lists of attending, attended, and interest experiences, and a list of participating chats ids.	● Experiences ● Clubs ● University ● Chats

Class name: Club (Organizer)

Parent class (if any**):**
Subclasses:

Responsibilities:	Collaborators:
● Knows club name, id, hashed password, club's social media links, website, a list of users in the organization and a list of participating chats ids	● Experiences ● Users ● University ● Message

Class name: University (Enterprise)

Parent class (if any**):**
Subclasses:

Responsibilities:	Collaborators:
● Knows a list of participating chats ids	● Clubs ● Users ● Experiences ● Chats ● Message

Class name: Message

Parent class (if any**):**
Subclasses:

Responsibilities:	Collaborators:

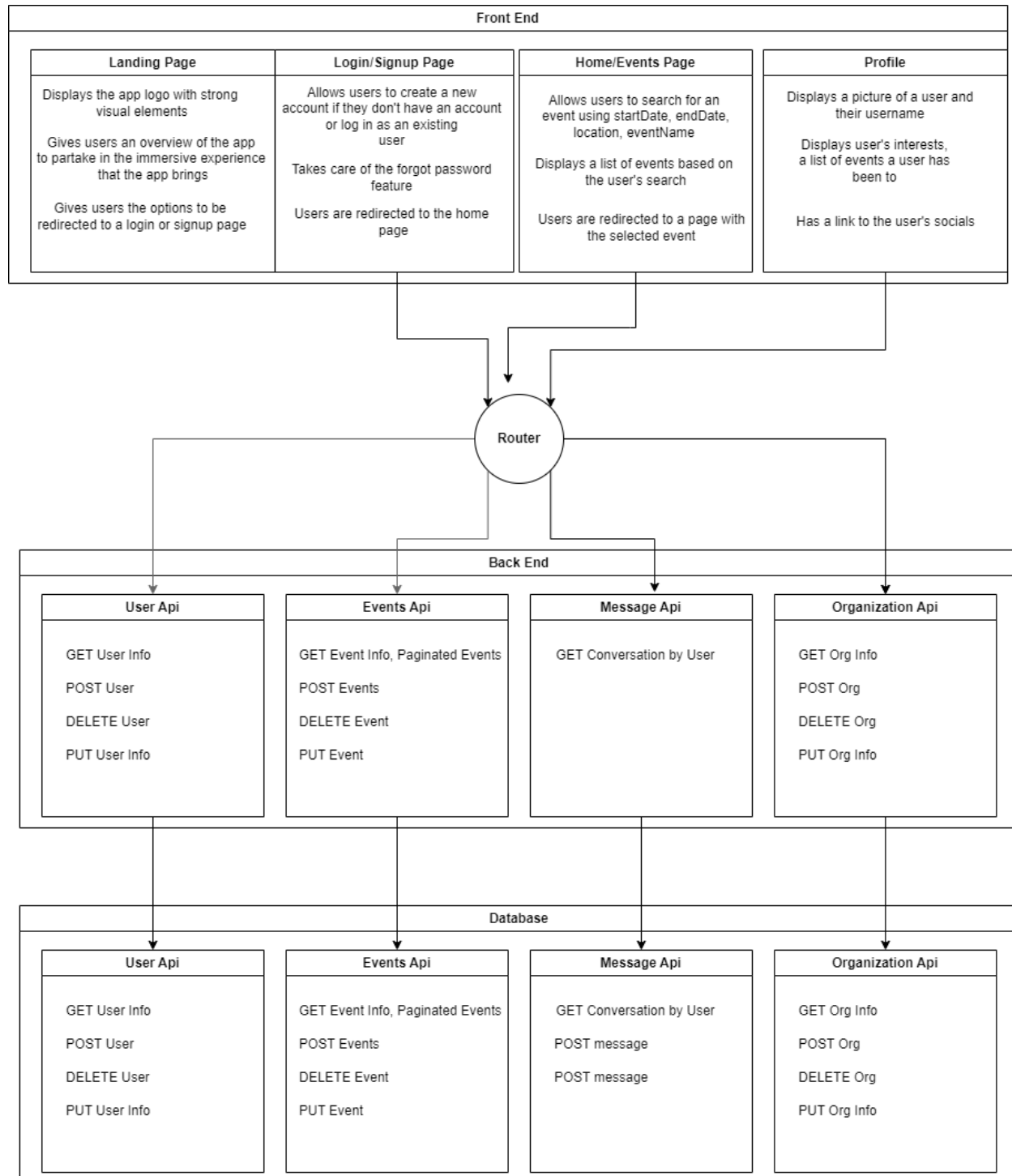|	
● Knows internal id, chat ID, message text, message time, message sender,	● Users ● Clubs ● University

Class name: Api Route
Parent class (if any**):** **Subclasses:** user api route, club api route, university api route, event api route, message api route

Responsibilities:	Collaborators:
● Get data from database ● Post data to database ● Delete data from database ● Put (update) data in database	● Pool

System interaction with the environment:

- **Dependencies and assumptions:**
  - *Programming language compilers*: If the user is self-hosting the app, they must have Node.js installed. They must also install the dependencies for the app using npm install.
  - *Databases:* The app assumes that the user has a PostgreSQL database set up, and that they have run our setup.sql file to create all the relevant tables. The connection string for this database must be stored in .env
  - *Network configuration:* The app assumes that port 3000 is unblocked and available, as it will try to host itself at localhost:3000 for local runs.
  - *Operating system:* The development work for the app was done in Linux. It should work on Windows or MacOS as well.

System Architecture Diagram

Front End

Landing Page	Login/Signup Page	Home/Events Page	Profile
Displays the app logo with strong visual elements	Allows users to create a new account if they don't have an account or log in as an existing user	Allows users to search for an event using startDate, endDate, location, eventName	Displays a picture of a user and their username
Gives users an overview of the app to partake in the immersive experience that the app brings	Takes care of the forgot password feature	Displays a list of events based on the user's search	Displays user's interests, a list of events a user has been to
Gives users the options to be redirected to a login or signup page	Users are redirected to the home page	Users are redirected to a page with the selected event	Has a link to the user's socials

Router

Back End

User Api	Events Api	Message Api	Organization Api
GET User Info	GET Event Info, Paginated Events	GET Conversation by User	GET Org Info
POST User	POST Events		POST Org
DELETE User	DELETE Event		DELETE Org
PUT User Info	PUT Event		PUT Org Info

Database

User Api	Events Api	Message Api	Organization Api
GET User Info	GET Event Info, Paginated Events	GET Conversation by User	GET Org Info
POST User	POST Events	POST message	POST Org
DELETE User	DELETE Event	POST message	DELETE Org
PUT User Info	PUT Event		PUT Org Info

System Decomposition

- **Router:** The router is the core entity of any web app created using Next.js. The router picks up all files under the app/ directory in the source code with special file names, such as page.tsx or route.tsx, and creates HTTPS routes to each of them.
    - **route.tsx**: Designates the URL *baseurl/parentfolder* as an API route, enabling the handling of HTTP API requests
    - **page.tsx:** Designates the URL *baseurl/parentfolder* as a normal route, displaying front-end elements generated dynamically and returned by page.tsx
- **Specific routes:**
    - **Backend / API**
        - User REST API
            - **GET**: retrieve single user info, paginated list of users, search for users
            - **POST:** create user, add event to interested, going, or attended list
            - **PUT**: update user information
            - **DELETE**: remove user, remove event from interested, going, or attended
        - Events REST API
            - **GET:** get event information, get paginated list of events, search for events
            - **POST:** create events
            - **PUT:** update event information
            - **DELETE**: remove event
        - Message API
            - **GET:** get paginated list of messages from message id, get info about chat, get specific message
            - **POST**: create message, create chat
            - **PUT:** edit message, edit chat
            - **DELETE:** delete chat, delete message
        - Organization API
            - **GET:** get org info, get list of org, search org
            - **POST:** create org, add events
            - **PUT:** edit org info
            - **DELETE:** delete org, remove event
    - **Frontend**
        - Landing page
        - Login/signup page
        - Events page
        - Profile page
    - **Database**
        - **Tables**
            - **Explained in detail in the CRC cards above**
            - Users

- Events
- Organizations
- Messages

Anticipated Errors and Exceptions

Usually, our endpoints will return a 200-level return code to indicate that the request was made correctly and the backend handled it correctly as well.

Anticipated errors include invalid input, malformed requests, database errors, users attempting to access resources that don't exist or authentication errors. Other types of errors aren't typically found in web applications as the manipulation of underlying data takes place only through an authenticated API.

Our application will handle these errors by having the API return an appropriate status code for the call, which enables the frontend to dynamically handle the returned error and display the correct message to the user.