Messaging App Project Ideation

A web app that allows users to send messages to each other.

# Goals

The project’s primary goal is to build a web app that allows users to send messages to each other, chatting about specific topics. Good precedents are Discord, Telegram, WhatsApp, Facebook Messenger, and AOL Instant Messenger. The app should support:

* Authorization
* Sending messages to another user
* Customizing a user profile

# Task

1. Think about what you’ll need to do to get all this working together. This is where it’s super helpful to think it completely through on paper or a whiteboard ahead of time! A few minutes of thought can save you from wasting an hour on coding.

* What will the user interface look like?
* What will the data model look like?
* What libraries will you need to use?

1. Start building your app by implementing the core functionality in your backend and front-end.

* You may have realized that a REST API backend cannot handle real time updates. It is “request-response”, so the server can only respond to a request. If a user sends a message to another user, a REST API cannot automatically notify the recipient as the recipient will not have requested that data. Methods for real time server-client updates (particularly when dealing with a split front/backend) have not been taught, so you are not expected to implement any in this app.

1. Deploy your app to the web.
2. **Extra credit:**

* Allow sending images in chat.
* Add a friends list that users can add other users to and see when someone is online (alternative: “Add a users list to show which users are currently online”; same thing except maybe a step or two less since it doesn’t require adding a friend).
* Allow users to create and send messages in group chats.

# Structure

* Front-end app for the user interfaces.
* Back-end app for data management.

# User privileges

* **Basic:** Authenticated user
* **Admin:** An administrator

|  |  |  |
| --- | --- | --- |
| **Privilege** | **Basic** | **Admin** |
| Create an account | Yes | Yes |
| Send messages | Yes | Yes |
| Receive messages | Yes | Yes |
| Update personal profile | Yes | Yes |
| Update non-personal profile | No | No |
| Delete personal account | Yes | No |
| Delete non-personal account | No | Yes |
| Create chat rooms | No | Yes |
| Manage chat rooms | No | Yes |

# Applications objectives

## Pages

* **Homepage:**
* Welcome user
* User login/signup
* **Sign-up:** Registration page for users to create a new account.
* First name
* Last name
* Username (unique)
* Email (unique)
* Password
* Admin passcode
* **Log-in page:** Allow users to log in to their account.
* username
* Password
* **Chat page:**
* Prompt the user to join a chat room
* List of available chat rooms
* Allow the user to send and receive messages to and from other users in the chat room.
* Allow the user to update their profile.
* **Dashboard Homepage (admin-only)**
* List all users
* Delete user UI

## Backend

* Provide RESTful APIs for the app.
* Create users with statuses that determines what they can do on the app.
* Allow admins to manage users.
* Allow administrators to manage users’ access to the restricted privileges.
* Sanitize and validate forms’ data.

# Models

## users

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | first\_name | last\_name | username | email | password |
| 1 | Oluwatobi | Sofela | sofela | contact@codesweetly.com | test |
| 2 | Sarah | Precious | precious | sp@example.com | example |
| 3 | Dav | Emma | emma | de@sample.com | sample |

## statuses

|  |  |
| --- | --- |
| id | name |
| 1 | basic |
| 2 | Admin |

## user\_status

|  |  |  |
| --- | --- | --- |
| id | user\_id | status\_id |
| 1 | 3 | 1 |
| 2 | 2 | 2 |

## channel

|  |  |  |
| --- | --- | --- |
| id | name | image |
| 1 | JavaScript | https://www.researchgate.net/profile/Simon-Frimpong-Yeboah/publication/385907322/figure/fig2/AS:11431281291221821@1731945210103/JavaScript-Source-JavaScript-Icon-nd.jpg |
| 2 | React | https://cdn.worldvectorlogo.com/logos/react-1.svg |
| 3 | Node | www.node-image.com |

# Tech stack

* **Frontend:** Next.js, React, TypeScript
* **Backend:** Node.js, Express.js
* **Database:** PostgreSQL, Prisma ORM
* **Data Fetching:** SWR
* **Chat:** Stream
* **Authentication:** Passport.js (local strategy), jsonwebtoken
* **Validation:** express-validator
* **Styling:** Tailwind CSS
* **Deployment:** Koyeb, Netlify

# Usage

## Backend

1. Clone the project

```bash

git clone https://github.com/oluwatobiss/gist-backend.git

```

2. Navigate into the project repo

```bash

cd gist-backend

```

3. Install dependencies

```bash

npm install

```

4. Create an environment variable file

```bash

touch .env

```

5. Define the project's environment variables

```

ADMIN\_CODE=example-pass

DATABASE\_URL=postgresql://username:password@localhost:5432/gist

GIST\_APP\_URI=http://localhost:3000

JWT\_SECRET=example\_jwt\_secret

PORT=3001

STREAM\_API\_KEY=x0xxxxxx0x0x

STREAM\_API\_SECRET=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

6. Migrate the project's schema to your database

```bash

npx prisma migrate dev

```

7. Start the server

```bash

npm run start

```

## Frontend

**Note:** The backend must be running for the frontend to function appropriately.

1. Clone the project

```bash

git clone https://github.com/oluwatobiss/gist-frontend.git

```

2. Navigate into the project repo

```bash

cd gist-frontend

```

3. Install dependencies

```bash

npm install

```

4. Create an environment variable file

```bash

touch .env

```

5. Define the project's environment variables

```

PUBLIC\_BACKEND\_URI="http://localhost:3001"

NEXT\_PUBLIC\_STREAM\_API\_KEY=x0xxxxxx0x0x

```

6. Start the server

```bash

npm run dev

```