Postss Project Ideation

Clone of a social media site, such as Facebook, X, Myspace, or Threads.

# Goals

The project’s primary goal is to:

* Put together the core features of a social media platform like users, profiles, posts, following, and “liking”. (You don’t have to worry about some of the more flashy front-end stuff unless you want to, but you don’t need it to get a nice user experience.)
* Implement some form of authentication. Ideally, you’ll want to use passport.js to support authenticating via the social media site you’re cloning. (Some sites (such as Facebook), have recently made this process impossible. If this is the case for your site, you can use passport.js to support authenticating via username and password with passport-local or via Github with passport-github2.)

# 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. An important part of planning is scope. You obviously can’t build the entire website (which presumably took a full team of engineers years to produce), so you’ll need to identify the site’s core functionality and the “nice-to-have” stuff.
2. Think through the data architecture required to make this work. There are lots of models and the relationship between them is more complicated than anything you’ve done before. Take some time to plan your approach before diving in. (You can populate data like users and posts with fake data using the [Faker](https://github.com/faker-js/faker) module from npm. To accomplish this create a new JavaScript file named `seeds.js` which imports your Prisma models and uses the faker module to generate and save a bunch of new users.)
3. Start building your app by implementing the core functionality in your backend and front-end. Make sure you finish the core functionality BEFORE working on the rest. If you try to do everything at once, you’ll get lost and frustrated. Trust us. Everything takes longer than you expect.

* Users must sign in to see anything except the sign-in page.
* Users can send follow requests to other users.
* Users can create posts (begin with text only).
* Users can like posts.
* Users can comment on posts.
* Posts should always display the post content, author, comments, and likes.
* There should be an index page for posts, which shows all the recent posts from the current user and users they are following.
* Users can create a profile with a profile picture. Depending on how you handle authentication, for example via `passport-github2`, you may be able to use their account’s existing profile picture. If this isn’t the case you can use [Gravatar](https://www.gravatar.com/) to generate them.
* A user’s profile page should contain their profile information, profile photo, and posts.
* There should be an index page for users, which shows all users and buttons for sending follow requests to users the user is not already following or have a pending request.

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

* Make posts also allow images (either just via a URL or by uploading one). If you did the extra credit from the File Uploader project, then you may recall [Cloudinary](https://cloudinary.com/documentation/node_integration) or [Supabase storage](https://supabase.com/docs/guides/storage) being good options for hosting user-uploaded images. The URLs they provide you can then be stored in your database instead of the raw image binary data.
* Allow users to update their profile photo.
* Create a guest sign-in functionality that allows visitors to bypass the login screen without creating an account or supplying credentials. This is especially useful if you are planning on putting this project on your résumé - most recruiters, hiring managers, etc. will not take the time to create an account. This feature will allow them to look at your hard work without going through a tedious sign-up process.
* Make it pretty!

# Structure

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

# User privileges

* **Subscriber:** Follower of an account
* **Owner:** Creator of an account
* **Admin:** App administrator

|  |  |  |  |
| --- | --- | --- | --- |
| **Privilege** | **Subscriber** | **Owner** | **Admin** |
| Create an account | Yes | Yes | Yes |
| Create post | Yes | Yes | Yes |
| Manage post | No | Yes | No |
| Delete post | No | No | Yes |
| Send follow requests | Yes | Yes | Yes |
| Comment | Yes | Yes | Yes |
| Manage profile | No | Yes | No |
| Delete personal account | Yes | Yes | No |
| Delete non-personal account | No | No | Yes |

# Applications objectives

## Pages

* **Homepage:**
* Latest posts from all users
* New post UI
* Profile UI
* Log out UI
* **Sign-up:** Registration page for users to create a new account.
* Username (unique)
* Email (unique)
* Password
* Admin passcode
* **Edit profile:** Allow users to update their profile
* First name (Optional)
* Last name (Optional)
* Photo (Optional)
* Username (unique)
* Email (unique)
* Admin passcode
* **Log-in page:** Allow users to log in to their account.
* username
* Password

## 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 | Subscriber |
| 2 | Owner |
| 3 | Admin |

## user\_status

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

# Tech stack

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

# Usage

## Backend

1. Clone the project

```bash

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

```

2. Navigate into the project repo

```bash

cd postss-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/postss

POSTSS\_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 dev

```

## Frontend

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

1. Clone the project

```bash

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

```

2. Navigate into the project repo

```bash

cd postss-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

```