Requirements:

Frontend:

TWO PAGES TOTAL:
BEFORE: LOGIN/SIGNUP

AFTER: REDIRECT TO POST-LOGIN HOMEPAGE

“I decided we would go with react after further research”

Login Component:

- A form with:
  - Email
  - Password
- Validations:
  - Check for required fields.
  - Validate the email format.
- On successful login:
  - Store the received token in localStorage.
  - Display a welcome message or redirect to a "Dashboard" page.
- **Signup Component**:
- A form with:
  - Name
  - Email
  - Password
  - Confirm Password
- Validations:
  - Check for required fields.
  - Ensure passwords match.
- On successful signup:
  - Display a success message.
  - Redirect to the Login page.

- **Design**:
  - Use React with TypeScript or JavaScript.

    - Implement a responsive layout using CSS frameworks like Tailwind, Material-UI, or plain CSS.
- **API Integration**:
    - Use fetch or axios to make requests to the Express backend.

Backend:

I will be testing you on a express.js server that takes POST requests to /login and /signup.

/Signup will be routed to a broker called alpaca before returning any response.

/Login will validate user information either on a cloud db or localhost db.

Express.js Server:

- **POST** /login
  POST /signup

Future Renditions of this component would:

- Check if the email exists for login.
  Ensure the email is unique for signup.

Database (Optional):

Use an in-memory array to store user data for simplicity.
For bonus points, integrate MongoDB or PostgreSQL.

Endpoints:

Login Endpoint:
URL: /login

- Request Body:

json
CopyEdit

```
{
  "email": "string",
  "password": "string"
}
```

        - Responses:
            - **Success**: Return a JSON Web Token (JWT).
            - **Failure**: Return an error message (e.g., "Invalid credentials").
    - **Signup Endpoint**:
        - URL: /signup
        - Request Body:

```
json
CopyEdit
{
  "username": "string",
  "email": "string",
  "password": "string"
}
```

        - Responses:
            - **Success**: Return a success message.
            - **Failure**: Return an error message (e.g., "Email already exists", I will test this on a database in SQL through our backend).

Evaluation Criteria:

Frontend:

Clean and modular React code.
API integration and proper handling of responses (success and errors, feel free to build your own tiny backend api to test responses, but it is not required).
Responsive and user-friendly UI.

Backend:

Dont worry about storing the user info in the backend, I (Phillip) will take care of testing to see if you're api endpoints work properly.