Simple Login and Registration System Documentation

Overview

This project is a simple Register-Login application that allows users to register and login. It uses React for the frontend and Express with MongoDB for the backend.

Table of Contents

Setup Instructions

Prerequisites

- Node.js
- MongoDB (It should be running)

Frontend Setup

1. **Navigate to the Backend Directory**:

   ```
   cd client
   ```

1. **Install Dependencies**:

   ```
   npm install bootstrap axios react-router-dom
   ```

2. **Start the React Application**:

   ```
   npm start
   npm run dev
   ```

   The application should be running at `http://localhost:5173`.

Backend Setup

2. **Navigate to the Backend Directory**:

```
cd server
npm init -y
```

3. **Install Dependencies**:

   ```
   npm install express mongoose cors nodemon
   ```

4. **Start the Express Server**:

   ```
   npm start
   ```

   The server should be running.

Technology Stack

Frontend

- **React**: A JavaScript library for building user interfaces.
- **Axios**: A promise-based HTTP client for making API requests.
- **React Router**: A library for routing in React applications.

Backend

- **Express**: A flexible Node.js web application framework.
- **MongoDB**: A NoSQL database for storing user information.
- **Mongoose**: An Object Data Modeling (ODM) library for MongoDB and Node.js.

Code Structure and Design Decisions

Frontend

- **src**
  - **components**: Contains React components.
    - `Login.jsx`: Handles user login functionality.
    - `Signup.jsx`: Handles user registration functionality.
    - `Home.jsx`: Empty home page.
  - **App.js**: Main application component with routing.
  - **index.js**: Entry point for the React application.

Backend

- **models**
  - `Employee.js`: Mongoose schema and model for employee data.
- **index.js**: Main server file that sets up Express, connects to MongoDB, and defines API endpoints.

Design Decisions

- **Separation of Concerns**: The project is divided into a client and server directory to separate frontend and backend concerns.
- **State Management**: Used React's `useState` and `useEffect` hooks for managing component state and side effects.
- **Routing**: Used React Router for navigating between different components and pages.
- **API Structure**: Defined RESTful API endpoints for registration, login, and fetching user data.
- **Error Handling**: Basic error handling in API calls and responses to handle common scenarios like invalid credentials and missing data.

Assumptions and Limitations

Assumptions

- The user email is unique and used as the primary identifier for login.
- Passwords are stored as plain text for simplicity but should be hashed in a real-world application.
- The application runs locally, and MongoDB is accessible at `mongodb://127.0.0.1:27017/register-login`.

Limitations

- **Security**: No encryption for passwords or secure storage mechanisms are implemented.
- **Validation**: Basic form validation is included, but more comprehensive validation should be implemented for better security and user experience.