**Frontend for Admin in the Matrimonial App**

**Code for AdminDashboard.js:**

import React, { useState, useEffect } from 'react';

import AdminLogin from './AdminLogin';

import axios from 'axios';

const AdminDashboard = () => {

  const [isOpen, setIsOpen] = useState(false); // State to manage dropdown menu

  const [users, setUsers] = useState([]); // State to store user data

  const [loading, setLoading] = useState(true); // State to store user data

  // Fetch user data from the backend

  useEffect(() => {

    const fetchUsers = async () => {

      try {

        const response = await axios.get('http://localhost:3001/api/profile', {

          method: 'GET',

          headers: {

            'Authorization': localStorage.getItem('user\_token')

          }

        }); // Replace '/api/users' with the appropriate endpoint URL

        setUsers(response.data);

        response && setLoading(false);

      } catch (error) {

        error && setLoading(false);

        console.error(error);

      }

    };

    fetchUsers();

  }, []);

  if (loading) return <p>Loading...</p>

  // Toggle the dropdown menu

  const toggleDropdown = () => {

    setIsOpen(!isOpen);

  };

  return (

    <div>

      <h1>Admin Dashboard</h1>

      <div>

        {

          users.map((data) => <ul key={data.\_id}>

            <li>

              <h2>Name: {data.name}</h2>

              <h4>Email: {data.email}</h4>

              <p>Gender: {data.gender}</p>

              <p>Birth: {data.dateOfBirth}</p>

              <p>Location: {data.location}</p>

            </li>

          </ul>)

        }

      </div>

    </div>

  );

};

export default AdminDashboard;

The code snippet represents a React component called ‘**AdminDashboard’**. It is responsible for rendering the admin dashboard interface, fetching user data from the backend, and displaying the user information.

**AdminDashboard Component:**

* This component is written using functional components and the React Hooks API.
* State Variables:
  + **`isOpen`** (Boolean): Represents the state of the dropdown menu. It is used to toggle the visibility of the dropdown.
  + **`users`** (Array): Stores the user data fetched from the backend.
  + **`loading`** (Boolean): Represents the loading state while fetching user data.
* useEffect Hook:
  + The `**useEffect`** hook is used to fetch user data from the backend when the component mounts.
  + It makes an asynchronous request to the `**/api/profile`** endpoint on the backend server using the Axios library.
  + The request includes an Authorization header containing the user token stored in the local storage.
  + If the request is successful, the user data is stored in the `**users`** state variable, and the `**loading`** state is set to **false**.
  + If there's an error during the request, the error is logged to the console, and the `**loading`** state is set to **false**.
  + The `**useEffect`** hook has an empty dependency array (**[]**), which ensures that the effect runs only once when the component mounts.
* Rendering User Data:
  + The `**users`** state is mapped over to render each user's information as a list item (**<li>**).
  + The user data, such as name, email, gender, date of birth, and location, is displayed within the list item.
  + Each user item has a unique key based on the` **\_id`** property of the user object.
* Dropdown Toggle:
  + The `**toggleDropdown`** function is used to toggle the visibility of the dropdown menu.
  + It updates the `**isOpen`** state by toggling its value.
* Conditional Rendering:
  + While the user data is being fetched (**loading** is **true**), a loading message (**<p>Loading...</p>**) is rendered.
  + Once the user data is fetched, the loading message is replaced with the user information displayed in the list format.

**Usage:**

* The `**AdminDashboard`** component can be used within a React application to render the admin dashboard interface and fetch user data from the backend.
* The component assumes the existence of the `**AdminLogin**` component, which is responsible for handling admin login functionality.
* The `**axios`** library is used for making HTTP requests to the backend server.
* To use this component, include it in the appropriate part of your application, such as within a router configuration or another component's render method.

It's important to note that the code snippet fetches user data from the backend at the `**/api/profile`** endpoint and expects an Authorization header with a user token. Make sure to replace **'http://localhost:3001/api/profile'** with the correct backend URL and endpoint for fetching user data in your application.

**Code for AdminLogin.js:**

import React, { useState } from 'react';

const AdminLogin = () => {

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const [errorMessage, setErrorMessage] = useState('');

  const handleUsernameChange = (event) => {

    setUsername(event.target.value);

  };

  const handlePasswordChange = (event) => {

    setPassword(event.target.value);

  };

  const handleSubmit = (event) => {

    event.preventDefault();

    // Here you can implement the logic to handle the form submission

    // For example, you can send an API request to validate the credentials

    if (username === 'admin' && password === 'password') {

      // Successful login

      console.log('Logged in as admin');

      setErrorMessage('');

    } else {

      // Invalid credentials

      setErrorMessage('Invalid username or password');

    }

  };

  return (

    <div>

      <h2>Admin Login</h2>

      {errorMessage && <div style={{ color: 'red' }}>{errorMessage}</div>}

      <form onSubmit={handleSubmit}>

        <label>

          Username:

          <input type="text" value={username} onChange={handleUsernameChange} />

        </label>

        <br />

        <label>

          Password:

          <input type="password" value={password} onChange={handlePasswordChange} />

        </label>

        <br />

        <button type="submit">Login</button>

      </form>

    </div>

  );

};

export default AdminLogin;

The code snippet represents a React component called **AdminLogin**. It is responsible for rendering an admin login form, capturing the username and password input from the user, and handling the form submission.

**AdminLogin Component:**

* This component is written using functional components and the React Hooks API.
* State Variables:
  + **`username`** (String): Represents the username entered by the user in the login form.
  + **`password`** (String): Represents the password entered by the user in the login form.
  + **`errorMessage`** (String): Stores an error message to display if the login attempt fails.
* Event Handlers:
  + **`handleUsernameChange`**: Updates the `**username`** state with the value entered in the username input field.
  + **`handlePasswordChange`**: Updates the `**password`** state with the value entered in the password input field.
  + **`handleSubmit`**: Handles the form submission event.
    - It prevents the default form submission behavior to prevent a page refresh.
    - It checks if the entered username and password match the expected admin credentials (e.g., 'admin' and 'password').
    - If the credentials are valid, it logs a success message to the console and clears the `**errorMessage`** state.
    - If the credentials are invalid, it sets the `**errorMessage`** state to display an error message to the user.
* Rendering:
  + The component renders an admin login form.
  + If there is an `**errorMessage`** state, it is displayed in a `**<div>`** with a red color.
  + The form consists of two input fields (username and password) and a login button.
  + The `**value`** of the input fields is set to the corresponding state variables (**username** and **password**).
  + The `**onChange`** event handlers are attached to the input fields to update the respective state variables when the user types in the inputs.
  + The `**onSubmit`** event handler is attached to the form to handle the form submission.
  + When the user clicks the login button, the `**handleSubmit`** function is called.

**Usage:**

* The `**AdminLogin`** component can be used within a React application to render an admin login form and handle the login logic.
* To use this component, include it in the appropriate part of your application, such as within a router configuration or another component's render method.
* The form submission logic in the `**handleSubmit`** function is a placeholder and needs to be replaced with actual authentication logic.
* Customize the logic inside the `**handleSubmit`** function to validate the user's credentials, such as sending an API request to the backend for authentication.

Ensure that you have appropriate styling and further functionality (such as authentication API integration) implemented in your application according to your specific requirements.

**Code for AdminDashboardPage.js:**

import React, { useState, useEffect } from 'react';

const AdminDashboard = () => {

  const [users, setUsers] = useState([]);

  useEffect(() => {

    // Fetch data from the API endpoint

    fetch('http://localhost:3001')

      .then((response) => response.json())

      .then((data) => setUsers(data))

      .catch((error) => {

        console.error('Error fetching data:', error);

        // Handle error state if necessary

      });

  }, []);

  return (

    <div>

      <h2>Admin Dashboard</h2>

      <h3>User List</h3>

      {users.length === 0 ? (

        <p>Loading...</p>

      ) : (

        <ul>

          {users.map((user) => (

            <li key={user.id}>{user.name}</li>

          ))}

        </ul>

      )}

    </div>

  );

};

export default AdminDashboard;

The code snippet represents a React component called **AdminDashboard**. It is responsible for rendering the admin dashboard interface, fetching user data from an API endpoint, and displaying the user list.

**AdminDashboard Component:**

* This component is written using functional components and the React Hooks API.
* State Variable:
  + **`users`** (Array): Stores the user data fetched from the API endpoint.
* useEffect Hook:
  + The `**useEffect`** hook is used to fetch user data from the API endpoint when the component mounts.
  + It makes an asynchronous request using the `**fetch`** function to the URL **'http://localhost:3001'**.
  + The response is converted to JSON using the `**.json()`** method, and the resulting data is stored in the **users** state variable.
  + If there is an error during the request, the error is logged to the console, and you can handle the error state if necessary.
  + The `**useEffect`** hook has an empty dependency array (**[]**), which ensures that the effect runs only once when the component mounts.
* Rendering User Data:
  + The user data stored in the `**users`** state is rendered conditionally.
  + If the `**users`** array is empty, a loading message (**<p>Loading...</p>**) is displayed.
  + Once the user data is fetched and stored in the `**users`** array, the user list is rendered.
  + Each user's name is displayed in a list item (**<li>**), and the `**user.id`** property is used as the unique **key** prop.
  + The user list is wrapped in an unordered list (**<ul>**).

**Usage:**

* The `**AdminDashboard`** component can be used within a React application to render the admin dashboard interface and fetch user data from an API endpoint.
* To use this component, include it in the appropriate part of your application, such as within a router configuration or another component's render method.
* Ensure that the API endpoint URL **'http://localhost:3001'** is replaced with the correct URL for your backend server.
* Customize the rendering of the user data to suit your specific requirements. For example, you can display additional user information or include actions for each user in the list.

**Code for AdminLoginPage.js:**

import React, { useState } from 'react';

const AdminLoginPage = () => {

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const handleUsernameChange = (event) => {

    setUsername(event.target.value);

  };

  const handlePasswordChange = (event) => {

    setPassword(event.target.value);

  };

  const handleFormSubmit = (event) => {

    event.preventDefault();

    // Your login logic here

    // Example: Check username and password against a database or API

    if (username === 'admin' && password === 'admin123') {

      // Successful login, redirect to admin dashboard or protected route

      console.log('Logged in as admin');

    } else {

      // Invalid credentials, display an error message

      console.log('Invalid credentials');

    }

  };

  return (

    <div>

      <h2>Admin Login</h2>

      <form onSubmit={handleFormSubmit}>

        <div>

          <label htmlFor="username">Username:</label>

          <input

            type="text"

            id="username"

            value={username}

            onChange={handleUsernameChange}

          />

        </div>

        <div>

          <label htmlFor="password">Password:</label>

          <input

            type="password"

            id="password"

            value={password}

            onChange={handlePasswordChange}

          />

        </div>

        <button type="submit">Login</button>

      </form>

    </div>

  );

};

export default AdminLoginPage;

The code snippet represents a React component called `**AdminLoginPage`**. It is responsible for rendering an admin login page, capturing the username and password input from the user, and handling the form submission.

**AdminLoginPage Component:**

* This component is written using functional components and the React Hooks API.
* State Variables:
  + **`username`** (String): Represents the username entered by the user in the login form.
  + **`password`** (String): Represents the password entered by the user in the login form.
* Event Handlers:
  + **`handleUsernameChange`**: Updates the `**username`** state with the value entered in the username input field.
  + **`handlePasswordChange`**: Updates the `**password`** state with the value entered in the password input field.
  + **`handleFormSubmit`**: Handles the form submission event.
    - It prevents the default form submission behavior to prevent a page refresh.
    - It checks if the entered username and password match the expected admin credentials (e.g., 'admin' and 'admin123').
    - If the credentials are valid, it logs a success message to the console.
    - If the credentials are invalid, it logs an error message to the console.
* Rendering:
  + The component renders an admin login form.
  + The form consists of two input fields (username and password) and a login button.
  + The `**value`** of the input fields is set to the corresponding state variables (**username** and **password**).
  + The `**onChange`** event handlers are attached to the input fields to update the respective state variables when the user types in the inputs.
  + The `**onSubmit`** event handler is attached to the form to handle the form submission.
  + When the user clicks the login button or presses enter, the `**handleFormSubmit`** function is called.

**Usage:**

* The `**AdminLoginPage`** component can be used within a React application to render an admin login page and handle the login logic.
* To use this component, include it in the appropriate part of your application, such as within a router configuration or another component's render method.
* The form submission logic in the `**handleFormSubmit`** function is a placeholder and needs to be replaced with actual authentication logic.
* Customize the logic inside the `**handleFormSubmit`** function to validate the user's credentials, such as checking against a database or making an API request to the backend for authentication.

**Code for AdminPage.js:**

import React from 'react';

import AdminLogin from '../components/AdminLogin';

const AdminPage = () => {

  return (

    <div>

      <h2>Login</h2>

      <AdminLogin />

    </div>

  );

};

export default AdminPage;

The code snippet represents a React component called **AdminPage**. It is responsible for rendering a login page for the admin by including the **AdminLogin** component.

**AdminPage Component:**

* This component is written using functional components.
* Rendering:
  + The component renders a login page for the admin.
  + It includes an `**AdminLogin`** component, which is responsible for rendering the actual login form.
* Usage:
  + The `**AdminPage`** component can be used within a React application to render an admin login page.
  + To use this component, include it in the appropriate part of your application, such as within a router configuration or another component's render method.
  + Customize the rendering and layout of the login page as per your application's design requirements.
  + The `**AdminLogin`** component included in this page handles the login form rendering and form submission logic. You can customize it separately if needed.

**Code for RequireAdmin.js:**

import axios from 'axios';

import React, { useEffect, useState } from 'react';

import { Navigate, useLocation } from 'react-router-dom';

const RequireAdmin = ({ children }) => {

    const [adminData, setAdminData] = useState('');

    const location = useLocation();

    const [loading, setLoading] = useState(true);

    useEffect(() => {

        axios.get('http://localhost:3001/api/profile/single', {

            method: 'GET',

            headers: {

                'Authorization': localStorage.getItem('user\_token')

            }

        }).then((response) => {

            setAdminData(response.data);

            setLoading(false);

        }).catch((error) => {

            console.log(error);

            setLoading(false);

        })

    }, []);

    if (loading) {

        return <p>Loading... </p>

    }

    console.log(adminData.isAdmin);

    if (adminData?.isAdmin) {

        return children

    }

    else {

        localStorage.removeItem('user\_token');

        return <Navigate to="/login" state={{ from: location }} replace />;

    };

};

export default RequireAdmin;

The code snippet represents a React component called `**RequireAdmin`**. It is a higher-order component (HOC) that acts as a middleware to protect routes and allows access only to authenticated admins.

**RequireAdmin Component:**

* This component is written using functional components and the React Hooks API.
* Props:
  + **`children`** (React Node): Represents the child components that will be rendered if the user is an admin.
* State Variables:
  + **`adminData`** (any): Represents the data of the admin retrieved from the server.
  + **`loading`** (boolean): Represents the loading state while fetching admin data.
* Additional Hooks and Functions:
  + **`useLocation`** (from **react-router-dom**): Allows accessing the current location object.
* Effect Hook:
  + **`useEffect`**: Fetches admin data from the server when the component mounts.
    - It sends a GET request to the server's `**/api/profile/single`** endpoint.
    - The request includes an `**Authorization`** header with a token retrieved from the` **localStorage`**.
    - If the request is successful, it sets the `**adminData`** state with the response data and updates the loading state.
    - If an error occurs, it logs the error and updates the loading state.
* Conditional Rendering:
  + If the loading state is **true**, it displays a "Loading..." message.
  + If the `**adminData`** object has an `**isAdmin`** property with a truthy value, it renders the `**children`** components.
  + If the `**adminData`** object does not have an `**isAdmin`** property or its value is falsy, it removes the user token from the `**localStorage`** and redirects the user to the login page.
* Usage:
  + The `**RequireAdmin`** component can be used as a wrapper around protected routes that should only be accessible to admins.
  + To use this component, include it in the appropriate part of your application, such as within a router configuration or as a wrapper around specific components or routes.
  + Customize the redirect behavior or authentication logic inside the component based on your application's requirements.