**Frontend for User**

**components**

**pages**

**routes**

**services**

**Components :** In components we have register form, login form, profile form , navigation bar.

**registerform.js**

| **import React, { useState } from 'react';**  **import { registerUser } from '../services/authService';**  **import DatePicker from 'react-datepicker';**  **import 'react-datepicker/dist/react-datepicker.css';**  **import './RegisterForm.css'; // Import CSS file for styling**  **const RegisterForm = () => {**  **const [formData, setFormData] = useState({**  **name: '',**  **email: '',**  **password: '',**  **dateOfBirth: null,**  **gender: '',**  **location: '',**  **});**  **const handleChange = (e) => {**  **setFormData({ ...formData, [e.target.name]: e.target.value });**  **};**  **const handleDateChange = (date) => {**  **setFormData({ ...formData, dateOfBirth: date });**  **};**  **const handleSubmit = async (e) => {**  **e.preventDefault();**  **try {**    **await registerUser(formData);**  **// Handle successful registration, e.g., redirect to login page**  **} catch (error) {**  **// Handle registration error**  **}**  **};**  **return (**  **<form onSubmit={handleSubmit} className="register-form">**  **<div>**  **<h1>**  **<label htmlFor="name">Name</label>**  **<input type="text" id="name" name="name" value={formData.name} onChange={handleChange} />**  **</h1>**  **</div>**  **<div>**  **<h1>**  **<label htmlFor="email">Email</label>**  **<input type="email" id="email" name="email" value={formData.email} onChange={handleChange} />**  **</h1>**  **</div>**  **<div>**  **<h1>**  **<label htmlFor="password">Password</label>**  **<input type="password" id="password" name="password" value={formData.password} onChange={handleChange} />**  **</h1>**  **</div>**  **<div>**  **<h1>**  **<label htmlFor="dateOfBirth">Date of Birth</label>**  **<DatePicker**  **id="dateOfBirth"**  **name="dateOfBirth"**  **selected={formData.dateOfBirth}**  **onChange={handleDateChange}**  **placeholderText="Select Date"**  **/>**  **</h1>**  **</div>**  **<div>**  **<h1>**  **<label htmlFor="gender">Gender</label>**  **<select id="gender" name="gender" value={formData.gender} onChange={handleChange}>**  **<option value="">Select Gender</option>**  **<option value="male">Male</option>**  **<option value="female">Female</option>**  **<option value="other">Other</option>**  **</select>**  **</h1>**  **</div>**  **<div>**  **<h1>**  **<label htmlFor="location">Location</label>**  **<input type="text" id="location" name="location" value={formData.location} onChange={handleChange} />**  **</h1>**  **</div>**  **<button type="submit">Register</button>**  **</form>**  **);**  **};**  **export default RegisterForm;** |
| --- |

* The form uses the **useState** hook from React to manage the form state. The state variable formData holds the data for the user registration form, such as name, email, password, dateOfBirth, gender, and location.
* The **handleChange** function is used to update the state whenever the user types in the input fields. It uses the spread operator (...) to create a copy of the current formData and update the specific field based on the input's name attribute and its new value.
* The **handleDateChange** function is specifically used to update the dateOfBirth field in the state. It is called when a date is selected using the react-datepicker component.
* The **handleSubmit** function is called when the user submits the form. It prevents the default form submission behavior (which would cause a page reload) using **e**.**preventDefault().** Then, it calls the registerUser function (presumably defined in the authService module) with the formData as an argument to attempt user registration.
* The form elements (input fields and date picker) are associated with the corresponding formData properties using the value and onChange attributes. This ensures that the form fields are controlled components, and their values are updated according to the state.
* The form elements are labeled using the <**label**> tags with the htmlFor attribute, making them accessible to screen readers.
* The form also includes a <**select**> element for the gender field and options for "Male," "Female," and "Other."
* Once the user fills in the form and clicks the "Register" button, the **handleSubmit** function is called. If the registration is successful (i.e., no errors are thrown), you might handle the redirect to the login page inside the try block.

**profile.js**

| **import React, { useState, useEffect } from 'react';**  **import { getUserProfile, sendConnectionRequest } from'../services/authService';**  **import axios from 'axios';**  **const Profile = () => {**  **const [user, setUser] = useState(null);**  **const [matchedUser, setMatchedUser] = useState([]);**  **const [connectionId, setConnectionId] = useState('');**  **const userToken = localStorage.getItem('user\_token');**  **useEffect(() => {**  **const fetchUserProfile = async () => {**  **try {**  **const userProfile = await getUserProfile(userToken);**  **setUser(userProfile?.data);**  **} catch (error) {**  **error?.response?.data?.message && alert(error.response.data.message);**  **}**  **};**  **fetchUserProfile();**  **}, []);**  **const handleConnectionFormSubmit = async (e) => {**  **e.preventDefault();**  **try {**  **await sendConnectionRequest(user.id, connectionId, userToken);**  **alert('Connection request sent successfully!');**  **} catch (error) {**  **error?.response?.data?.message && alert(error.response.data.message);**  **}**  **setConnectionId('');**  **};**  **if (!user) {**  **return <div>Loading...</div>;**  **}**  **const handleSearch = async (event) => {**  **event.preventDefault();**  **const age = event.target.elements.age.value;**  **const gender = event.target.elements.gender.value;**  **try {**  **const res = await axios.get(`http://localhost:3001/api/users/search?age=${age}&gender=${gender}`, {**    **headers: {**  **'Authorization': userToken,**  **},**  **})**  **setMatchedUser(res.data)**  **} catch (error) {**  **console.log(error);**  **}**  **};**  **return (**  **<div>**  **<h2>Welcome, {user?.name}!</h2>**  **<p>Email: {user?.email}</p>**  **<p><h2>Get matched with the user you want </h2> </p>**  **{/\* Form to search for matches \*/}**  **<form id="searchForm" onSubmit={handleSearch}>**  **<input type="number" name="age" placeholder="Age" />**  **<input type="text" name="gender" placeholder="Gender" />**  **<button type="submit">Search</button>**  **</form>**  **{matchedUser.length === 0 ?**  **<p>User is not macthed!</p> :**  **matchedUser.map((data) => <ul key={data.\_id}>**  **<li>**  **<h2>{data.name}</h2>**  **<p>Location: {data.location}</p>**    **<button onClick={() => sendConnectionRequest(data.\_id)}>Send Connection Request</button>**  **</li>**    **</ul>)**  **}**  **</div>**  **);**  **};**  **export default Profile;** |
| --- |

* The component uses the **useState** and **useEffect** hooks from React to manage the state and perform side effects. The state variables are user, matchedUser, and connectionId. The user state holds the current user's profile data, matchedUser stores the list of users that match the search criteria, and connectionId is used to store the ID of the user to whom a connection request will be sent**.**
* The component retrieves the user token from **localStorage** using **localStorage**.**getItem**(**'user\_token'**). This token is likely used for authentication when making API requests.
* The **useEffect** hook is used to fetch the user profile data once when the component mounts. It calls the **getUserProfile** function from the **../services/authService** module with the user token, and if successful, it sets the user data in the state.
* The **handleConnectionFormSubmit** function is called when the user submits the connection request form. It prevents the default form submission behavior, calls the **sendConnectionRequest** function from the ../services/authService module with the relevant data (user ID and connection ID), and shows an alert indicating a successful connection request. The connectionId state is then reset to an empty string.
* The **handleSearch** function is called when the user submits the search form to find matched users based on age and gender. It retrieves the values from the form inputs and makes a **GET** request to http://localhost:3001/api/users/search with the provided age and gender as query parameters and the user token in the headers. If the request is successful, the matched user data is stored in the matchedUser state.
* The **render()** function displays the user's name and email along with a search form where the user can input age and gender to find matches. If there are no matched users, a message is displayed indicating that there are no matches. Otherwise, the matched users are displayed in an unordered list (**ul**).
* For each matched user, their name, location, and a "**Send** **Connection Request**" button are displayed. Clicking the button will call the sendConnectionRequest function with the matched user's ID. However, there's a small issue with this part of the code: the **sendConnectionRequest** function is not defined within this component, and it should be imported from the ../**services**/**authService** module like the other functions.

**navbar.js**

| **// Navbar.js**  **import React from 'react';**  **import { Link } from 'react-router-dom';**  **import './Navbar.css';**  **const Navbar = () => {**  **return (**  **<nav>**  **<ul>**    **<li>**  **<h1>**  **<Link to="/">Home</Link>**  **</h1>**  **</li>**  **<li>**  **<h1>**  **<Link to="/register">Register</Link>**  **</h1>**  **</li>**  **<li>**  **<h1>**  **<Link to="/login">Login</Link>**  **</h1>**  **</li>**  **<li>**  **<h1>**  **<Link to="/profile">User Profile</Link>**  **</h1>**  **</li>**  **<li><h1>**  **<Link to="/admindashboard">AdminDashboard</Link></h1>**  **</li>**  **</ul>**  **</nav>**  **);**  **};**  **export default Navbar;** |
| --- |

* The component imports **React** and the **Link** component from **react-router-dom**.
* The **Navbar** component is defined as a functional component.
* Inside the Navbar component, a <**nav**> element is used as the container for the navigation bar.
* The navigation bar consists of an unordered list (<**ul**>) that contains multiple list items (<**li**>).
* Each list item represents a link in the navigation bar. The links are created using the **Link** component from react-router-dom. The to prop of each Link component specifies the URL path to navigate to when the link is clicked.
* The text for each link is wrapped inside an <**h1**> element for styling purposes. You can adjust the heading level (<**h1**>, <**h2**>, etc.) based on your styling needs.
* The available links in the navigation bar include "**Home**," "**Register**," "**Login**," "**User** **Profile**," and "**AdminDashboard**." Each link is wrapped inside an <h1> element.
* The Link components are nested inside the list items (<**li**>), forming a clickable navigation menu.
* Finally, the **Navbar** component is exported as the default export of the module.

**loginform.js**

| import React, { useState } from 'react';  import { loginUser } from '../services/authService';  import { useNavigate } from 'react-router-dom';  const LoginForm = () => {  const [formData, setFormData] = useState({  email: '',  password: '',  });  const navigate = useNavigate();  const handleChange = (e) => {  setFormData({ ...formData, [e.target.name]: e.target.value });  };  const handleSubmit = async (e) => {  e.preventDefault();  try {  const result = await loginUser(formData);  if (result?.token) {  localStorage.setItem('user\_token', result.token);  navigate('/profile');  }  else {  alert('Unauthorized User');  }  // Handle successful login, e.g., redirect to profile page  } catch (error) {  alert(error.response.data.message);  }  };  return (  <form onSubmit={handleSubmit}>  <input type="email" name="email" placeholder="Email" value={formData.email} onChange={handleChange} />  <input type="password" name="password" placeholder="Password" value={formData.password} onChange={handleChange} />  <button type="submit">Login</button>  </form>  );  };  export default LoginForm; |
| --- |

* The component imports **React, useState from React, loginUser** function from the **../services/authService** module, and the **useNavigate** hook from react-router-dom.
* The **LoginForm** component is defined as a functional component.
* Inside the **LoginForm** component, the state variables **formData** and **navigate** are created using the **useState** hook. The formData state holds the values of the email and password inputs, and the navigate state is used to navigate to different routes after a successful login.
* The **handleChange** function is called when the user types in the email or password input fields. It updates the **formData** state with the new values entered by the user using the spread operator and the name attribute of the input field.
* The **handleSubmit** function is called when the user submits the login form. It prevents the default form submission behavior, calls the **loginUser** function from the ../**services**/**authService** module with the **formData**, and awaits the result.
* If the result returned from **loginUser** contains a token, it means the login was successful. In that case, the **user\_token** is stored in the **localStorage**, and the user is navigated to the '/**profile'** route using the navigate function.
* If the login was unsuccessful, i.e., result does not contain a token, an alert is shown with the message **'Unauthorized User'**.
* If an error occurs during the login process, an alert is shown with the error message from the server response.
* The **render()** function displays a form with an **email** input, **password** input, and a **login** button. The **onSubmit** event is set to the **handleSubmit** function, and the value and **onChange** props of the input fields are bound to the **formData** state and the **handleChange** function, respectively.
* Finally, the **LoginForm** component is exported as the default export of the module.

**Pages**

**registerpage.js**

| **import React from 'react';**  **import RegisterForm from '../components/RegisterForm';**  **const RegisterPage = () => {**  **return (**  **<div>**  **<h2>Register as a new user</h2>**  **<RegisterForm />**  **</div>**  **);**  **};**  **export default RegisterPage;** |
| --- |

* The component imports **React** and the **RegisterForm** component from **../components/RegisterForm**.
* The **RegisterPage** component is defined as a functional component.
* Inside the RegisterPage component, a <**div**> element is used as a container for the register page content.
* The component renders an <**h2**> element with the text "**Register as a new user"** as the heading for the register page.
* The **RegisterForm** component is rendered, allowing users to fill out the registration form.
* Finally, the **RegisterPage** component is exported as the default export of the module.

**Loginpage.js**

| **import React from 'react';**  **import LoginForm from '../components/LoginForm';**  **const LoginPage = () => {**  **return (**  **<div>**  **<h2>Login</h2>**  **<LoginForm />**  **</div>**  **);**  **};**  **export default LoginPage;** |
| --- |

* The component imports React and the **LoginForm** component from **../components/LoginForm.**
* The **LoginPage** component is defined as a functional component.
* Inside the **LoginPage** component, a <**div**> element is used as a container for the login page content.
* The component renders an <**h2**> element with the text "**Login**" as the heading for the login page.
* Finally, the **LoginPage** component is exported as the default export of the module.

**homepage.js**

| **import React from 'react';**  **const HomePage = () => {**  **return (**  **<div>**  **<h2>Welcome to Connecting Hearts Matrimony</h2>**  **</div>**  **);**  **};**  **export default HomePage;** |
| --- |

* The component imports **React**.
* The **HomePage** component is defined as a functional component.
* Inside the **HomePage** component, a <**div**> element is used as a container for the home page content.
* The component renders an <**h2**> element with the text "**Welcome to Connecting Hearts Matrimony**" as the heading for the home page.
* Finally, the **HomePage** component is exported as the default export of the module.

**Routes**

**Index.js**

| **import React from 'react';**  **import { Routes, Route } from 'react-router-dom';**  **import HomePage from '../pages/HomePage';**  **import RegisterPage from '../pages/RegisterPage';**  **import LoginPage from '../pages/LoginPage';**  **import Profile from '../components/Profile';**  **import AdminDashboard from '../components/AdminDashboard';**  **import RequireAdmin from '../pages/RequireAdmin';**  **const AppRoutes = () => {**  **return (**  **<Routes>**  **<Route path="/" element={<HomePage />} />**  **<Route path="/register" element={<RegisterPage />} />**  **<Route path="/login" element={<LoginPage />} />**  **<Route path="/profile" element={<Profile />} />**  **<Route path="/admindashboard" element={**  **<RequireAdmin>**  **< AdminDashboard />**  **</RequireAdmin>**  **} />**  **</Routes>**  **);**  **};**  **export default AppRoutes;** |
| --- |

* The component imports **React, Routes, Route**, and various page and component components.
* The **AppRoutes** component is defined as a functional component is used as the top-level component to define the route configuration.
* The **Route** components are used to define individual routes. Each Route component has a path prop that specifies the URL path for that route and an element prop that specifies the component to render when the corresponding route is matched.
* The **HomePage** component is rendered when the root URL path ("/") is matched.
* The **RegisterPage** component is rendered when the "/**register**" path is matched.
* The **LoginPage** component is rendered when the "/**login**" path is matched.
* The **Profile** component is rendered when the "/**profile**" path is matched.
* The **AdminDashboard** component is rendered when the "/**admindashboard**" path is matched. However, it is wrapped in the **RequireAdmin** component, which ensures that only authenticated admin users can access the admin dashboard.
* Finally, the **AppRoutes** component is exported as the default export of the module.

**privateRoutes.js**

| **import React from 'react';**  **import { Route, Redirect } from 'react-router-dom';**  **import AdminDashboard from '.components/AdminDashboard';**  **const isAuthenticated = () => {**  **const token = localStorage.getItem('token');**  **return token !== null;**  **};**  **const PrivateRoute = ({ component: Component, ...rest }) => {**  **return (**  **<Route**  **{...rest}**  **render={(props) =>**  **isAuthenticated() ? (**  **<AdminDashboard {...props} />**  **) : (**  **<Redirect to="admin/login" />**  **)**  **}**  **/>**  **);**  **};**  **export default PrivateRoute;** |
| --- |

* The component imports **React, Route, Redirect,** and the **AdminDashboard** component.
* The **isAuthenticated** function is defined, which represents your authentication logic. In this example, it checks if a token is stored in the local storage. You can modify this function to implement your own authentication logic.
* The **PrivateRoute** component is defined as a functional component that takes a component prop and other rest props.
* Inside the **PrivateRoute** component, a **Route** component is used to define the **private route**. It spreads the rest **props** onto the Route component.
* The render prop of the **Route** component is used to conditionally render the **AdminDashboard** component or redirect to the "**admin**/**login**" route if the user is not authenticated.
* If the user is authenticated (as determined by the isAuthenticated function), the **AdminDashboard** component is rendered, and any additional props passed to the **PrivateRoute** component are spread onto the AdminDashboard component.
* If the user is not authenticated, a Redirect component is rendered with the "**admin**/**login**" route as the to prop. This will redirect the user to the login page.
* Finally, the **PrivateRoute** component is exported as the default export of the module.

**Services**

**authservices.js**

| **import axios from 'axios';**  **const API\_BASE\_URL = 'http://localhost:3001/api'; // Replace with your backend API base URL**  **// axios.get('http://localhost:3001/api')**  **// .then(response => {**  **// // Handle the response from the backend**  **// console.log(response.data);**  **// })**  **// .catch(error => {**  **// // Handle any errors that occur during the request**  **// console.error(error);**  **// });**  **export const registerUser = async (userData) => {**  **try {**  **const response = await axios.post(`${API\_BASE\_URL}/auth/register`, userData);**  **return response.data;**  **} catch (error) {**  **throw new Error(error.response.data.error);**  **}**  **};**  **export const loginUser = async (userData) => {**  **const response = await axios.post(`${API\_BASE\_URL}/auth/login`, userData);**  **console.log(response);**  **return response.data;**  **};**  **export const getUserProfile = async (userToken) => {**  **const response = await axios.get(`${API\_BASE\_URL}/profile/single`, {**  **method: 'GET',**  **headers: {**  **'Authorization': userToken**  **}**  **});**  **return response;**  **};**  **// Function to send a connection request**  **export const sendConnectionRequest = async (userId, connectionId, userToken) => {**  **try {**  **const response = await axios.post(**  **`/api/users/${userId}/connections`,**  **{ connectionId },**  **{**  **headers: {**  **Authorization: `Bearer ${userToken}`,**  **},**  **}**  **);**  **return response.data;**    **} catch (error) {**  **throw error;**  **}**  **};** |
| --- |

* The code imports the Axios library, which is used for making HTTP requests.
* The **API\_BASE\_URL** constant is defined with the base URL of your backend API. Make sure to replace it with the actual URL of your backend API.
* The **registerUse**r function is defined. It takes **userData** as a parameter, sends a POST request to the **/auth/register** endpoint of your API with the provided user data, and returns the response data if successful. If an error occurs, it throws an error with the error message from the API response.
* The **loginUser** function is defined. It takes userData as a parameter, sends a POST request to the **/auth/login** endpoint of your API with the provided user data, and returns the response data if successful. The response contains the user token that can be stored in local storage for authentication purposes.
* The **getUserProfile** function is defined. It takes **userToken** as a parameter, sends a GET request to the **/profile/single** endpoint of your API with the user token in the Authorization header, and returns the response data containing the user profile if successful.
* The **sendConnectionRequest** function is defined. It takes **userId, connectionId, and userToken** as parameters. It sends a POST request to the **/api/users/${userId}/connections** endpoint with the connectionId as the request body. It includes the Authorization header with the user token for authentication. If successful, it returns the response data.

**App.js**

| **import React from 'react';**  **import { BrowserRouter as Router } from 'react-router-dom';**  **import Routes from './routes';**  **import Header from './components/Header';**  **import './Footer';**  **import Footer from './Footer';**  **const App = () => {**  **return (**    **<Router>**    **<Header />**    **<Routes />**  **<Footer />**  **</Router>**  **);**  **};**  **export default App;** |
| --- |

* The code imports the necessary components and dependencies, including **React, BrowserRouter from react-router-dom, Routes component, Header component**, and **Footer component.**
* The App component is defined as a functional component.
* Inside the App component, the **Router** component from react-router-dom is used to wrap the application and enable routing functionality.
* The Header component is rendered, which presumably contains the **navigation** menu or header section of your application.
* The **Routes** component is rendered, which handles the routing and rendering of different pages based on the URL.
* The **Footer** component is rendered, which presumably contains the footer section of your application.
* The **App** component is exported as the default export of the module.

**index.js**

| **import React from 'react';**  **import ReactDOM from 'react-dom/client';**  **import './index.css';**  **import App from './App';**  **import reportWebVitals from './reportWebVitals';**  **const root = ReactDOM.createRoot(document.getElementById('root'));**  **root.render(**  **// <React.StrictMode>**  **<App />**  **// </React.StrictMode>**  **);**  **reportWebVitals();** |
| --- |

* The code imports the necessary dependencies: **React** and **ReactDOM** from react and react-dom, respectively. It also imports the index.css file for styling and the reportWebVitals function.
* The **createRoot** function from **ReactDOM** is used to create a root for concurrent mode rendering. It takes the DOM element with the ID **'root'** and returns a root object.
* The render method of the root object is called, and the **App** component is passed as the root component to render. This will render the entire application starting from the App component and its child components.
* The **reportWebVitals** function is called, which reports performance metrics of your application. This function is typically used for monitoring and analyzing performance in development or production environments.