**Authentication Procedure:**

Two ways are describe that when user login and then its credentials like name, username and other fields are store in browser section (I think).  
  
**Study it to understand its flow**

**Auth.jsx** (frontend) **(**1st way/method**)**

import { useState, useEffect, useContext, createContext } from "react";

import axios from "axios";

*const* AuthContext = createContext();

*const* AuthProvider = ({ *children* }) *=>* {

*const* [auth, setAuth] = useState({

    user: null,

    token: "",

  });

  //default axios

  axios.defaults.headers.common["Authorization"] = auth?.token;

  useEffect(() *=>* {

*const* data = localStorage.getItem("auth");

    if (data) {

*const* parseData = JSON.parse(data);

      setAuth({

        ...auth,

        user: parseData.user,

        token: parseData.token,

      });

    }

  }, []);

  return (

    <*AuthContext.Provider* value={[auth, setAuth]}>

      {*children*}

    </*AuthContext.Provider*>

  );

};

// custom hook

*const* useAuth = () *=>* useContext(AuthContext);

export { useAuth, AuthProvider };

**main.jsx / index.jsx**

import { StrictMode } from 'react'

import { createRoot } from 'react-dom/client'

import App from './App.jsx'

import './index.css'

import { BrowserRouter } from 'react-router-dom';

import { AuthProvider } from "./context/auth";

createRoot(document.getElementById('root')).render(

    <*AuthProvider*>

        <*BrowserRouter*>

            <*App* />

        </*BrowserRouter*>

    </*AuthProvider*>

)

**Auth.jsx**  **(**2nd way/method**)**

import { createContext, useEffect, useReducer } from "react";

import AuthReducer from "./AuthReducer";

*const* INITIAL\_STATE = {

  user:JSON.parse(localStorage.getItem("user")) || null,

  isFetching: false,

  error: false,

};

export *const* AuthContext = createContext(INITIAL\_STATE);

export *const* AuthContextProvider = ({ *children* }) *=>* {

*const* [state, dispatch] = useReducer(AuthReducer, INITIAL\_STATE);

  useEffect(()*=>*{

    localStorage.setItem("user", JSON.stringify(state.user))

  },[state.user])

  return (

    <*AuthContext.Provider*

      value={{

        user: state.user,

        isFetching: state.isFetching,

        error: state.error,

        dispatch,

      }}

    >

      {*children*}

    </*AuthContext.Provider*>

  );

};

**authReducer.jsx**

*const* AuthReducer = (*state*, *action*) *=>* {

    switch (*action*.type) {

      case "LOGIN\_START":

        return {

          user: null,

          isFetching: true,

          error: false,

        };

      case "LOGIN\_SUCCESS":

        return {

          user: *action*.payload,

          isFetching: false,

          error: false,

        };

      case "LOGIN\_FAILURE":

        return {

          user: null,

          isFetching: false,

          error: true,

        };

      case "FOLLOW":

        return {

          ...*state*,

          user: {

            ...*state*.user,

            followings: [...*state*.user.followings, *action*.payload],

          },

        };

      case "UNFOLLOW":

        return {

          ...*state*,

          user: {

            ...*state*.user,

            followings: *state*.user.followings.filter(

              (*following*) *=>* *following* !== *action*.payload

            ),

          },

        };

      default:

        return *state*;

    }

  };

export default AuthReducer;

**main.jsx / index.jsx**

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

import { AuthContextProvider } from "./context/AuthContext";

ReactDOM.render(

  <*React.StrictMode*>

    <*AuthContextProvider*>

      <*App* />

    </*AuthContextProvider*>

  </*React.StrictMode*>,

  document.getElementById("root")

);

**Explanation of the Two Codes**

Both pieces of code manage **authentication state** in a React application using Context API. However, they use slightly different approaches for managing and updating the state.

**1st Code Explanation**

This code uses **React's useContext and useState hooks** to create an authentication context.

**Key Components**

1. **AuthContext**:
   * Created using createContext.
   * Stores the authentication state and a function (setAuth) to update it.
2. **AuthProvider**:
   * A context provider that wraps the application's components.
   * Uses useState to store auth (with user and token).
   * Retrieves authentication data from localStorage during the initial render (useEffect).
   * Sets the Authorization header in axios for all requests globally.
   * Provides [auth, setAuth] as the context value, allowing components to read and update the auth state.
3. **useAuth (Custom Hook)**:
   * Simplifies the process of consuming the AuthContext.
   * Components can call useAuth() to access and update auth.

**How It Works**

* On the initial render, useEffect checks if auth data exists in localStorage. If so:
  + Parses the data and updates the auth state.
  + Automatically sets the Authorization header for axios requests.
* Provides authentication state (auth) and the update function (setAuth) to the children components.

**Example Usage**

jsx

Copy code

import { useAuth } from "./AuthContext";

const Profile = () => {

const [auth, setAuth] = useAuth();

return <h1>Welcome, {auth?.user?.name || "Guest"}!</h1>;

};

**Use Case**

This approach is simple and suitable for apps where state updates are infrequent or not overly complex.

**2nd Code Explanation**

This code uses **useReducer** to manage the authentication state, making it better for handling complex state transitions.

**Key Components**

1. **INITIAL\_STATE**:
   * Defines the initial authentication state, including:
     + user: Retrieved from localStorage or null if not available.
     + isFetching: Indicates whether authentication requests are in progress.
     + error: Tracks errors during authentication.
2. **AuthContext**:
   * Created using createContext.
   * Stores authentication state and a dispatch function for updating it.
3. **AuthContextProvider**:
   * Uses useReducer to manage the authentication state.
   * Saves user state to localStorage whenever it changes (useEffect).
   * Provides the state (user, isFetching, error) and the dispatch function to the children components.
4. **AuthReducer**:
   * A separate reducer function (imported as AuthReducer) handles state transitions based on dispatched actions.

**How It Works**

* The reducer listens for dispatched actions (e.g., LOGIN\_START, LOGIN\_SUCCESS, LOGIN\_FAILURE) and updates the state accordingly.
* useEffect syncs the user state with localStorage whenever it changes.
* Provides state and dispatch function to children components, allowing fine-grained control over state updates.

**Example Usage**

jsx

Copy code

import { AuthContext } from "./AuthContext";

const Login = () => {

const { user, dispatch } = useContext(AuthContext);

const handleLogin = async () => {

dispatch({ type: "LOGIN\_START" });

try {

const res = await axios.post("/api/login", { email, password });

dispatch({ type: "LOGIN\_SUCCESS", payload: res.data });

} catch (error) {

dispatch({ type: "LOGIN\_FAILURE" });

}

};

return user ? <h1>Welcome back!</h1> : <button onClick={handleLogin}>Login</button>;

};

**Use Case**

This approach is ideal for more complex applications where state updates are triggered by multiple actions and require more structured management (e.g., asynchronous login flows).

**Comparison**

| **Feature** | **1st Code** | **2nd Code** |
| --- | --- | --- |
| **State Management** | useState | useReducer |
| **Complexity** | Simple | Handles complex state transitions |
| **Data Sync** | Syncs auth with localStorage on init | Syncs user with localStorage whenever updated |
| **Global State Update** | Directly via setAuth | Via dispatch and AuthReducer |
| **Use Case** | Small to medium-sized apps | Apps with complex state management |
| **Custom Hook** | useAuth | Not provided, directly uses useContext |

Both are valid patterns for managing authentication in React apps, with the choice depending on the complexity of the application's requirements.

**when user login then its information and token in the form of object is store in network response. tell me how to get network response in any component of application. I use following code to store it.**

**auth.jsx file code is:**

import { useState, useEffect, useContext, createContext } from "react";

import axios from "axios";

*const* AuthContext = createContext();

*const* AuthProvider = ({ *children* }) *=>* {

*const* [auth, setAuth] = useState({

    user: null,

    token: "",

  });

  //default axios

  axios.defaults.headers.common["Authorization"] = auth?.token;

  useEffect(() *=>* {

*const* data = localStorage.getItem("auth");

    if (data) {

*const* parseData = JSON.parse(data);

      setAuth({

        ...auth,

        user: parseData.user,

        token: parseData.token,

      });

    }

    //eslint-disable-next-line

  }, []);

  return (

    <*AuthContext.Provider* value={[auth, setAuth]}>

      {*children*}

    </*AuthContext.Provider*>

  );

};

// custom hook

*const* useAuth = () *=>* useContext(AuthContext);

export { useAuth, AuthProvider };

**and my login code is:**

import React, { useState } from 'react'

import { Link } from "react-router-dom";

import toast from "react-hot-toast";

import axios from "axios";

import { useAuth } from "../context/auth";

import { FaKey } from "react-icons/fa";

import { MdEmail } from "react-icons/md";

*function* Recipient\_Login() {

*const* [email, setEmail] = useState("");

*const* [password, setPassword] = useState("");

*const* [auth, setAuth] = useAuth();

  // const navigate = useNavigate();

  // const location = useLocation();

  async *function* handleSubmit(*e*) {

*e*.preventDefault();

    try {

      // const res = await axios.post(`${process.env.REACT\_APP\_API}/user/login`, {

      //   email: email,

      //   password,

      // });

*const* res = await axios.post(`http://localhost:5000/recipient-login`, {

        email: email,

        password,

      });

      console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

      console.log("\*\*res", res, "\*\*res.data", res.data, "\*\*res.data.success", res.data.success)

      console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

      if (res && res.data.success) {

        toast.success(res.data && res.data.success);//  line:35

        // toast.success(res.data.success && res.data);//  line:36

        setAuth({

          // ...auth,

          user: res.data.RECIPIENT,

          token: res.data.token,

        });

        // navigate("/");

        // navigate(location.state || "/");//is sy agr login ni hai to pehly login krwa kr phir usi page pr byj dy ga jisy access krna ho ga

        localStorage.setItem("auth", JSON.stringify(res.data))

        console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Auth\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        console.log(auth)

        console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Auth\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

      } else {

        toast.error(res.data.message);

      }

    } catch (error) {

      console.log("\*\*\*\*\*\*\*\*error\*\*\*\*\*\*\*", error, "\*\*\*\*\*\*\*\*\*\*\*");

      // toast.error("Something went wrong");

      toast.error(error.response.data.message);

    }

  };

  return (

    <div>

      <div className='m-12'>

        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde, libero odio odit dolores expedita saepe magnam consequatur id nostrum labore a ipsum iste quasi at possimus, accusantium culpa mollitia! Placeat!

      </div>

      <div className="bg-slate-200 hero min-h-screen">

      recipient-login

        <div className='m-12'>AAAAAAsdjkkkkkkkkkkk

          <p>asddddddddd</p>qqqq

        </div>

        <div className=" hero-content flex-col lg:flex-row-reverse">

          <div className="w-[90%] md:w-[80%] lg:w-[35%] text-center lg:text-left">

            <div className='text-center'

            // className='bg-teal-200' style={{ borderRadius: "25% 25% 0% 71% / 0% 0% 25% 68%", border: "5px solid #1a73e8" }}

            >

              {/\* <h1 className="text-5xl font-bold text-center">Register Now!</h1> \*/}

              <img src="/donor register image.png" alt="" className='h-96 lg:h-auto mx-auto' />

            </div>

          </div>

          <div className="bg-slate-300 w-[90%] md:w-[80%] lg:w-[60%] card shrink-0 shadow-2xl flex items-center"

            style={{

              boxShadow: "0 0 36px 2px #074799",

            }}

          >

            <form onSubmit={handleSubmit} encType="multipart/form-data"

              className=" card-body grid grid-cols-1 md:grid-cols-2 gap-4">

              <div className="flex flex-col gap-1">

                <label htmlFor="email" className="text-sm font-medium text-gray-700">Email</label>

                <div className="input input-bordered flex items-center gap-2">

                  <div className="">

                    <*MdEmail* size={20} />

                  </div>

                  <input

                    type="text"

                    value={email}

                    onChange={*function* (*e*) { setEmail(e.target.value) }}

                    required

                    id="email"

                    className="grow"

                    placeholder=""

                  />

                </div>

              </div>

              <div className="flex flex-col gap-1">

                <label htmlFor="password" className="text-sm font-medium text-gray-700">Password</label>

                <div className="input input-bordered flex items-center gap-2">

                  <div className="">

                    <*FaKey* size={20} />

                  </div>

                  <input

                    type="password"

                    value={password}

                    onChange={*function* (*e*) { setPassword(e.target.value) }}

                    required

                    id="password"

                    className="grow"

                    placeholder=""

                  />

                </div>

              </div>

              <div className="mt-6 w-[90%]  text-center">

                <button type="submit" className="bg-[#24599e] border-0 text-white btn w-[50%] font-bold text-2xl hover:bg-[#888b8e] hover:text-black"

                  style={{

                    boxShadow: "0 0 16px 2px #074799",

                  }}

                >Login</button>

              </div>

            </form>

          </div>

        </div>

        ggggggggggg

      </div>

      <hr />

      <div>

        AAAAAAAAAAA

      </div>

    </div>

  );

}

export default Recipient\_Login;

**Explanation:**

To get the user's authentication information and token stored in the network response in any component of your React application, you can leverage the AuthContext you've created. Here's how you can achieve that:

**1. Accessing Auth Context in Any Component**

Since you are storing the authentication information (auth object) in the AuthContext, you can access it in any component by using the useAuth custom hook. Here's an example:

**Example of Accessing User Data**

import React from 'react';

import { useAuth } from "../context/auth";

*const* Dashboard = () *=>* {

*const* [auth] = useAuth(); // Access the auth state

  return (

    <div>

      <h1>Welcome to the Dashboard</h1>

      {auth.user ? (

        <div>

          <p><strong>User Name:</strong> {auth.user.name}</p>

          <p><strong>Email:</strong> {auth.user.email}</p>

          <p><strong>Token:</strong> {auth.token}</p>

        </div>

      ) : (

        <p>No user is logged in.</p>

      )}

    </div>

  );

};

export default Dashboard;

**2. Explanation**

* **useAuth()**:
  + This custom hook provides access to the auth state and the setAuth function defined in the AuthProvider component.
* **auth**:
  + The auth object contains the user details and the token.
* **Default Value**:
  + If no user is logged in, auth.user will be null, and auth.token will be an empty string.

**3. How the Context Works**

* When a user logs in, the Recipient\_Login component stores the authentication data (user and token) in the AuthContext and in localStorage.
* This data is accessible globally in any component that uses the AuthContext.

**4. Accessing Data After Login**

After the login request is successful, you update the auth state:

setAuth({

user: res.data.RECIPIENT,

token: res.data.token,

});

This triggers a re-render in all components that consume the AuthContext, ensuring the updated authentication data is available immediately.

**5. Handling Token Automatically**

The following line in your AuthProvider ensures that the token is automatically included in every Axios request:

axios.defaults.headers.common["Authorization"] = auth?.token;

**6. Storing and Retrieving Data**

The authentication data is persisted using localStorage:

* **Storing Data**:

localStorage.setItem("auth", JSON.stringify(res.data));

* **Retrieving Data** (on page reload):

const data = localStorage.getItem("auth");

if (data) {

const parseData = JSON.parse(data);

setAuth({

...auth,

user: parseData.user,

token: parseData.token,

});

}

**7. Full Workflow**

1. User logs in via Recipient\_Login.
2. Login response updates AuthContext and localStorage.
3. Any component using useAuth gets immediate access to the auth state.

**8. Use Case in Another Component**

For example, if you need the token to make an API call:

import React from 'react';

import axios from 'axios';

import { useAuth } from "../context/auth";

const Profile = () => {

const [auth] = useAuth();

const fetchProfile = async () => {

try {

const res = await axios.get("http://localhost:5000/user/profile", {

headers: {

Authorization: `Bearer ${auth.token}`, // Use the token from context

},

});

console.log("Profile Data:", res.data);

} catch (error) {

console.error("Error fetching profile:", error);

}

};

return (

<div>

<button onClick={fetchProfile}>Get Profile</button>

</div>

);

};

export default Profile;

By following this approach, you ensure that the user's authentication data is available throughout the application in a secure and consistent way.