React TS Auth Cookie

create a react typescript skeleton with cookie based authentication and authorization

Here's a **React + TypeScript skeleton project** that includes:

- Cookie-based authentication (using `httpOnly` cookies for secure auth)
- Authorization logic for protected routes
- React Router v6
- Axios setup with credentials
- Simple role-based access control

🔧 Tech Stack

- React 18+
- TypeScript
- React Router v6
- Axios
- Express.js (for a mock backend)

🔁 Folder Structure

```
pgsql

my-auth-app/
├── client/    (React App)
│   ├── public/
│   ├── src/
│   │   ├── api/
│   │   ├── components/
│   │   ├── contexts/
│   │   ├── hooks/
│   │   ├── pages/
│   │   ├── routes/
│   │   ├── App.tsx
│   │   ├── main.tsx
│   ├── tsconfig.json
├── server/    (Node.js API)
│   ├── index.ts
│   ├── auth.ts
├── package.json (root)
```

1️⃣ Client: React + TypeScript

client/src/api/axios.ts

```
ts

import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5000',
  withCredentials: true, // for sending cookies
});

export default api;
```

---

client/src/contexts/AuthContext.tsx

```
tsx

import { createContext, useContext, useEffect, useState } from 'react';
import api from '../api/axios';

interface AuthContextType {
  user: null | { username: string; role: string };
  loading: boolean;
  login: (username: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider = ({ children }: { children: React.ReactNode }) => {
  const [user, setUser] = useState<AuthContextType['user']>(null);
  const [loading, setLoading] = useState(true);

  const fetchUser = async () => {
    try {
      const res = await api.get('/auth/me');
      setUser(res.data);
    } catch {
      setUser(null);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchUser();
  }, []);

  const login = async (username: string, password: string) => {
    await api.post('/auth/login', { username, password });
    await fetchUser();
  };

  const logout = async () => {
    await api.post('/auth/logout');
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout, loading }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => useContext(AuthContext)!;
```

---

client/src/routes/ProtectedRoute.tsx

```tsx
import { Navigate, Outlet } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const ProtectedRoute = ({ role }: { role?: string }) => {
  const { user, loading } = useAuth();

  if (loading) return <div>Loading...</div>;

  if (!user) return <Navigate to="/login" replace />;

  if (role && user.role !== role) return <Navigate to="/unauthorized" replace />;

  return <Outlet />;
};

export default ProtectedRoute;
```

---

client/src/pages/Login.tsx

```tsx
import { useState } from 'react';
import { useAuth } from '../contexts/AuthContext';
import { useNavigate } from 'react-router-dom';

const Login = () => {
  const { login } = useAuth();
  const navigate = useNavigate();
  const [form, setForm] = useState({ username: '', password: '' });

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      await login(form.username, form.password);
      navigate('/');
    } catch (err) {
      alert('Login failed');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input name="username" onChange={handleChange} placeholder="Username" />
      <input name="password" type="password" onChange={handleChange} placeholder="Password" />
      <button type="submit">Login</button>
    </form>
  );
};

export default Login;
```

---

client/src/App.tsx

```tsx
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { AuthProvider } from './contexts/AuthContext';
import ProtectedRoute from './routes/ProtectedRoute';
import Login from './pages/Login';
import Home from './pages/Home';
import Admin from './pages/Admin';

const App = () => (
  <AuthProvider>
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route element={<ProtectedRoute />}>
          <Route path="/" element={<Home />} />
        </Route>
        <Route element={<ProtectedRoute role="admin" />}>
          <Route path="/admin" element={<Admin />} />
        </Route>
        <Route path="/unauthorized" element={<div>Unauthorized</div>} />
      </Routes>
    </BrowserRouter>
  </AuthProvider>
);

export default App;
```

2️⃣ Server: Express (Mock Backend)

> Install: `npm install express cors cookie-parser`

`server/index.ts`

```ts
import express from 'express';
import cors from 'cors';
import cookieParser from 'cookie-parser';

const app = express();
const PORT = 5000;

const users = {
  admin: { password: 'admin123', role: 'admin' },
  user: { password: 'user123', role: 'user' },
};

app.use(
  cors({
    origin: 'http://localhost:5173',
    credentials: true,
  })
);
app.use(express.json());
app.use(cookieParser());

app.post('/auth/login', (req, res) => {
  const { username, password } = req.body;
  const user = users[username as keyof typeof users];

  if (!user || user.password !== password) {
    return res.status(401).send('Invalid credentials');
  }

  res.cookie('token', username, { httpOnly: true });
  res.sendStatus(200);
});

app.post('/auth/logout', (_, res) => {
  res.clearCookie('token');
  res.sendStatus(200);
});

app.get('/auth/me', (req, res) => {
  const token = req.cookies.token;
  const user = users[token as keyof typeof users];

  if (!user) return res.status(401).send('Unauthorized');

  res.json({ username: token, role: user.role });
});

app.listen(PORT, () => console.log(`Auth API running on http://localhost:${PORT}`));
```

🏁 To Run the App

```bash
# Terminal 1
cd server
npm install
npx tsx index.ts

# Terminal 2
cd client
npm create vite@latest my-auth-app --template react-ts
npm install axios react-router-dom
npm run dev
```

Would you like a GitHub repo or ZIP version of this setup?

create a front end modular structure with generics

To create a **modular and generic frontend structure in React + TypeScript**, we'll apply best practices such as:

- Generic service layer
- Reusable form/input components
- Modularized contexts, hooks, and pages
- Scalable folder structure

This design helps keep your codebase **clean**, **reusable**, and **testable** as it grows.

🧱 Recommended Folder Structure

graphql

```
src/
├── api/                  # Generic HTTP services
│   ├── httpClient.ts     # Axios wrapper
│   └── userService.ts    # Example: User-specific API
│
├── components/           # Reusable components
│   ├── forms/
│   │   └── Input.tsx
│   └── layout/
│       └── Navbar.tsx
│
├── contexts/             # React contexts
│   └── AuthContext.tsx
│
├── hooks/                # Custom hooks
│   ├── useFetch.ts       # Generic data fetcher
│   └── useAuth.ts        # Auth state
│
├── models/               # TypeScript interfaces and types
│   ├── User.ts
│   └── ApiResponse.ts
│
├── pages/                # Page components
│   ├── Home.tsx
│   └── Login.tsx
│
├── routes/               # Routing and route guards
│   └── AppRoutes.tsx
│
├── utils/                # Utilities and helpers
│   └── validators.ts
│
├── App.tsx
└── main.tsx
```

🔁 1. `api/httpClient.ts` — Generic Axios Wrapper

ts

```
import axios from 'axios';

const httpClient = axios.create({
  baseURL: 'http://localhost:5000',
  withCredentials: true,
});

export default httpClient;
```

🔁 2. `api/genericService.ts` — Generic Service Layer

ts

```
import httpClient from './httpClient';

export const createCRUDService = <T>(baseURL: string) => ({
  getAll: async (): Promise<T[]> => {
    const res = await httpClient.get<T[]>(baseURL);
    return res.data;
  },
  getOne: async (id: string): Promise<T> => {
    const res = await httpClient.get<T>(`${baseURL}/${id}`);
    return res.data;
  },
  create: async (data: T): Promise<T> => {
    const res = await httpClient.post<T>(baseURL, data);
    return res.data;
  },
  update: async (id: string, data: Partial<T>): Promise<T> => {
    const res = await httpClient.put<T>(`${baseURL}/${id}`, data);
    return res.data;
  },
  remove: async (id: string): Promise<void> => {
    await httpClient.delete(`${baseURL}/${id}`);
  },
});
```

👤 3. `models/User.ts`

```ts
export interface User {
  id: string;
  username: string;
  role: 'user' | 'admin';
}
```

🧠 4. `hooks/useFetch.ts` — Generic Hook

```ts
import { useEffect, useState } from 'react';

export const useFetch = <T>(fetchFn: () => Promise<T>) => {
  const [data, setData] = useState<T | null>(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<Error | null>(null);

  useEffect(() => {
    fetchFn()
      .then(setData)
      .catch(setError)
      .finally(() => setLoading(false));
  }, []);

  return { data, loading, error };
};
```

✏️ 5. `components/forms/Input.tsx` — Generic Input

```tsx
import React from 'react';

interface InputProps<T> {
  name: keyof T;
  value: T[keyof T];
  onChange: (name: keyof T, value: T[keyof T]) => void;
  type?: string;
  label?: string;
}
```

```
export function Input<T>({ name, value, onChange, type = 'text', label }: InputProps<T>) {
  return (
    <div>
      {label && <label>{label}</label>}
      <input
        type={type}
        value={String(value)}
        onChange={(e) => onChange(name, e.target.value as T[keyof T])}
      />
    </div>
  );
}
```

👥 6. `contexts/AuthContext.tsx` — Auth Context

```tsx
import { createContext, useContext, useState, useEffect } from 'react';
import httpClient from '../api/httpClient';
import { User } from '../models/User';

interface AuthContextType {
  user: User | null;
  login: (username: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider = ({ children }: { children: React.ReactNode }) => {
  const [user, setUser] = useState<User | null>(null);

  const login = async (username: string, password: string) => {
    await httpClient.post('/auth/login', { username, password });
    const res = await httpClient.get<User>('/auth/me');
    setUser(res.data);
  };

  const logout = async () => {
    await httpClient.post('/auth/logout');
    setUser(null);
  };

  useEffect(() => {
    httpClient.get<User>('/auth/me')
      .then((res) => setUser(res.data))
      .catch(() => setUser(null));
  }, []);

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => useContext(AuthContext)!;
```

🛣️ 7. `routes/AppRoutes.tsx`

```
tsx

import { Routes, Route, Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';
import Home from '../pages/Home';
import Login from '../pages/Login';

const ProtectedRoute = ({ children }: { children: JSX.Element }) => {
  const { user } = useAuth();
  return user ? children : <Navigate to="/login" />;
};

const AppRoutes = () => (
  <Routes>
    <Route path="/login" element={<Login />} />
    <Route path="/" element={<ProtectedRoute><Home /></ProtectedRoute>} />
  </Routes>
);

export default AppRoutes;
```

✅ Example Usage: `Login.tsx`

```
tsx

import { useState } from 'react';
import { Input } from '../components/forms/Input';
import { useAuth } from '../contexts/AuthContext';
import { useNavigate } from 'react-router-dom';

interface LoginForm {
  username: string;
  password: string;
}

const Login = () => {
  const { login } = useAuth();
  const [form, setForm] = useState<LoginForm>({ username: '', password: '' });
  const navigate = useNavigate();

  const handleChange = <K extends keyof LoginForm>(key: K, value: LoginForm[K]) => {
    setForm((prev) => ({ ...prev, [key]: value }));
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    await login(form.username, form.password);
    navigate('/');
  };

  return (
    <form onSubmit={handleSubmit}>
      <Input<LoginForm> name="username" value={form.username} onChange={handleChange}
label="Username" />
      <Input<LoginForm> name="password" type="password" value={form.password} onChange=
{handleChange} label="Password" />
      <button type="submit">Login</button>
    </form>
  );
};

export default Login;
```

⚡ Ready to Bootstrap

Want this as a starter template on GitHub or zipped to download?

I can generate a downloadable ZIP or scaffold the repo directly for you. Let me know your preference.