Web Application Architectures

Lab 3 - Create a Next.js app to work with an API

**Create a Next.js 15 frontend using the app/ directory to work with the JWT API. This will offer a clean folder structure and working pages for login and protected routes.**

**For an introduction to Next.js please follow:** [**https://nextjs.org/learn**](https://nextjs.org/learn)

This lab is using the API created in Lab 3. Make sure this is up and running on port 8000 before completing this example.

# **Step 1 - Create the project**

**Create a new project folder:** client

Accept all the default options. Ie. Just hit enter for each option

Open the folder in VS Code.

**Open a terminal window and run:**

npx create-next-app@15 my-next-app --ts

This command creates a new Next.js 15 project named my-next-app, preconfigured with TypeScript.

cd my-next-app

npm install axios

**axios is a popular JavaScript HTTP client library.**

For more details please see: <https://axios-http.com/docs/intro>

**Setup .env.local to include:**

|  |  |
| --- | --- |
| In the project directory (my-next-app) **create a file named .env.local**  Add this to the file:  NEXT\_PUBLIC\_API\_URL=http://localhost:8000/api |  |

A **.env.local** file is a special file used in Next.js (and many Node.js apps) to store environment variables that are specific to your local development setup. Keeps sensitive values (like API keys, database URLs, secrets) out of your codebase. We are using this file to store the location of our API

**Run the app:**

In terminal (in the my-next-app directory), run:

npm run dev

A computer screen with white text

AI-generated content may be incorrect.

In a web browser go to: <http://localhost:3000>

A screenshot of a computer

AI-generated content may be incorrect.

# **Step 2 – Setup the axios helper**

|  |  |
| --- | --- |
| In the **src** directory create a new directory named **lib**  In the lib directory create a new file named **api.ts**  **Add the following code:**  import axios from "axios";  const api = axios.create({  baseURL: process.env.NEXT\_PUBLIC\_API\_URL,  });  export default api; |  |

**Run the app:**

Stop the server in terminal if it is running (Click into terminal and press **ctrl + c**)

In terminal (in the my-next-app directory), run:

npm run dev

Make sure there are no errors in the terminal window

# **Step 3 – Define the root layout of the application**

Go to app -> layout.tsx

These files are part of next.js project structure.

|  |  |
| --- | --- |
| **Update layout.tsx to look as follows:**  import type { Metadata } from "next";  import "./globals.css";  export const metadata: Metadata = {    title: "Next.js JWT Demo",    description: "Consuming data from an API",  };  export default function RootLayout({ children }: { children: React.ReactNode }) {    return (      <html lang="en">        <body style={{ padding: "2rem", fontFamily: "sans-serif" }}>          <h1>Next.js JWT Demo</h1>          {children}        </body>      </html>    );  } |  |

In a Next.js App Router project, the file app/layout.tsx has a very specific role: it defines the root layout for your entire application.

What it does:

* Wraps every page in your app with a shared UI structure.
* Typically includes elements like:
* <html> and <body> tags
* Global providers (e.g. theme, authentication, state)
* Reusable UI like headers, footers, navigation, sidebars
* Global styles or fonts

**Run the app:**

Stop the server in terminal if it is running (Click into terminal and press **ctrl + c**)

In terminal (in the my-next-app directory), run:

npm run dev

Make sure there are no errors in the terminal window

Go to: <http://localhost:3000/>

A screenshot of a computer

AI-generated content may be incorrect.

# **Step 4 – Update the root page content**

**Go to app -> page.tsx**

|  |  |
| --- | --- |
| **Update page.tsx to look as follows:**  import Link from "next/link";  export default function HomePage() {  return (  <div>  <p>Welcome! Navigate to:</p>  <ul>  <li><Link href="/login">Login</Link></li>  <li><Link href="/protected">Protected Page</Link></li>  </ul>  </div>  );  } |  |

In a Next.js App Router project, the file app/page.tsx is the default route’s page component — it represents what users see when they visit the root URL / of your app.

* Every folder in app/ can have a page.tsx file → it defines the UI for that route.
  + app/page.tsx → /
  + app/about/page.tsx → /about
  + app/blog/page.tsx → /blog
* A page.tsx must export a React component (usually a function component).
* Pages are server components by default (better performance, no client bundle overhead), but can include client components if needed.

**Run the app:**

Stop the server in terminal if it is running (Click into terminal and press **ctrl + c**)

In terminal (in the my-next-app directory), run:

npm run dev

Make sure there are no errors in the terminal window

Go to: <http://localhost:3000/>

A screenshot of a computer

AI-generated content may be incorrect.

# **Step 5 – Create a login page**

In the app directory **create a new directory named login**.

In the new login directory create a new file named page.tsx.

|  |  |
| --- | --- |
| "use client";  import { useState } from "react";  import api from "../../lib/api";  import { useRouter } from "next/navigation";  export default function LoginPage() {  const [email, setEmail] = useState("");  const [password, setPassword] = useState("");  const router = useRouter();  const handleLogin = async () => {  try {  const res = await api.post("/login", { email, password });  localStorage.setItem("token", res.data.token);  alert("Logged in successfully!");  router.push("/protected");  } catch {  alert("Invalid credentials");  }  };  return (  <div>  <h2>Login</h2>  <input  type="email"  placeholder="Email"  value={email}  onChange={(e) => setEmail(e.target.value)}  />  <input  type="password"  placeholder="Password"  value={password}  onChange={(e) => setPassword(e.target.value)}  />  <button onClick={handleLogin}>Login</button>  </div>  );  } |  |

**Overview:**

"use client"; → Marks this as a client component (needed because it uses React hooks and browser APIs).

Imports:

* useState → to manage form inputs (email, password).
* api → a custom axios instance for API calls.
* useRouter → to programmatically navigate between pages.

LoginPage component:

* Renders a simple login form with email & password fields.
* On clicking Login, it calls an API (/login).
* If successful: stores the returned token in localStorage, shows an alert, and redirects to /protected.
* If failed: shows an “Invalid credentials” alert.

This is a login page component in Next.js that handles user authentication with an API and redirects after success.

**Run the app:**

Stop the server in terminal if it is running (Click into terminal and press **ctrl + c**)

In terminal (in the my-next-app directory), run:

npm run dev

Make sure there are no errors in the terminal window

Go to: <http://localhost:3000/login>

A screen shot of a computer

AI-generated content may be incorrect.

# **Step 6 – Create a protected page**

In the app directory **create a new directory named protected**.

In the new protected directory create a new file named page.tsx.

|  |  |
| --- | --- |
| "use client";  import { useEffect, useState } from "react";  import api from "../../lib/api";  export interface User {  id: number;  email: string;  username: string;  role: string;  }  export default function ProtectedPage() {  const [message, setMessage] = useState("");  const [user, setUser] = useState<User | null>(null);  useEffect(() => {  const fetchProtected = async () => {  const token = localStorage.getItem("token");  if (!token) {  setMessage("No token found. Please log in.");  return;  }  try {  const res = await api.get("/protected", {  headers: { Authorization: `Bearer ${token}` },  });  setMessage(res.data.message);  const me = await api.get("/users", {  headers: { Authorization: `Bearer ${token}` },  });  setUser(me.data);  } catch {  setMessage("Unauthorized. Please log in.");  }  };  fetchProtected();  }, []);  return (  <div>  <h2>Protected Page</h2>  <p>{message}</p>  {user && (  <div>  <h3>User Info</h3>  <pre>{JSON.stringify(user, null, 2)}</pre>  </div>  )}  </div>  );  } |  |

**Overview**

"use client"; → Marks this as a client component.

Defines a User interface (id, email, username, role).

State:

* message → feedback about access/authentication.
* user → holds the logged-in user’s data.

useEffect hook runs on mount:

1. Reads a token from localStorage.
2. If no token → shows “No token found. Please log in.”
3. If token → calls /protected API to check access.
4. Then fetches /users to get user info.
5. On error → shows “Unauthorized. Please log in.”

UI:

* Always shows a heading + message.
* If a user is loaded, displays their details as JSON.

This is a protected page component that verifies a stored token, fetches secure data, and shows user info if authenticated.

# **Step 7 – Test the App**

**Start the API:**

npx ts-node src/index.ts

This should be running on: localhost:8000

**Start the Next App:**

npm run dev

This should be running on: localhost:3000

**In a browser navigate to:**

/login → login with admin@mail.com / password

A screenshot of a computer

AI-generated content may be incorrect.

Click login.

/protected → view protected data and user info

A screenshot of a computer

AI-generated content may be incorrect.

To view the JWT token on the browser.

Open Developer tools (Right click the page and select Inspect)

Go to the Storage tab:

A screen shot of a computer

AI-generated content may be incorrect.