`main.tsx` — App Bootstrapping + Routing Setup

```
// React + ReactDOM are standard for rendering
import React from 'react';
import ReactDOM from 'react-dom/client';

// React Router 7 modern API
import {
  createBrowserRouter,
  RouterProvider,
  redirect,
} from 'react-router-dom';

import './index.css'; // TailwindCSS styling
import App from './App'; // Layout component
import Home from './routes/Home'; // Route page
import About from './routes/About'; // Route page
import Dashboard from './routes/Dashboard'; // Route page
import { requireAuth } from './auth/requireAuth'; // Access
control

//Router config using route objects (Router v7+)
const router = createBrowserRouter([
  {
    path: '/', // Main layout route
    element: <App />,
    children: [
      { index: true, element: <Home /> }, // '/' → Home
page
      { path: 'about', element: <About /> }, // '/about'
      {
        path: 'dashboard',
        // loader runs before route renders
        loader: async () => {
          await requireAuth(); // Redirects if not logged
in
          return { message: 'Welcome to the protected
dashboard!' };
        },
        element: <Dashboard />,
      },
    ],
  },
]);

// Connect the router to your app
ReactDOM.createRoot(document.getElementById('root')!).rende
r(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

`App.tsx` — Layout + Navigation + Login/Logout

```
import { Outlet, Link, useNavigate } from 'react-router-
dom';
import { useAuthStore } from './auth/useAuthStore'; //
Zustand state

export default function App() {
  const { isLoggedIn, login, logout } = useAuthStore();
  const navigate = useNavigate(); // Redirect on logout

  return (
    <div className="p-4 font-sans">
      {/* Navbar */}
      <nav className="flex gap-4 mb-6 text-blue-600 font-
semibold">
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
        {isLoggedIn && <Link to="/dashboard">Dashboard</
Link>}
      </nav>

      {/* Auth buttons */}
      <div className="mb-6">
        {isLoggedIn ? (
          <button
```

```
            className="bg-red-500 text-white px-3 py-1
rounded"
            onClick={() => {
              logout(); // Zustand update + clear
localStorage
              navigate('/');
            }}
          >
            Logout
          </button>
        ) : (
          <button
            className="bg-green-500 text-white px-3 py-1
rounded"
            onClick={login}
          >
            Login
          </button>
        )}
      </div>

      {/* Route outlet (Home/About/Dashboard renders here)
*/}
      <Outlet />
    </div>
  );
}
```

`routes/Home.tsx` — Static Route Page

```
export default function Home() {
  return <h1 className="text-3xl font-bold">Home Page</h1>;
}
```

Simple content shown when you hit `/`.

`routes/About.tsx` — Static Route Page

```
export default function About() {
  return <h1 className="text-3xl font-bold">About Us</h1>;
}
```
Same as above — just showing content for `/about`.

`routes/Dashboard.tsx` — Protected Page with Loader

```
import { useLoaderData } from 'react-router-dom';

export default function Dashboard() {
  const data = useLoaderData() as { message: string };

  return (
    <div>
      <h1 className="text-3xl font-bold text-
green-700">Dashboard</h1>
      <p className="mt-4">{data.message}</p>
    </div>
  );
}
```

What's special here:

- `useLoaderData()` reads the object returned from the `loader()` defined in `main.tsx`
- If the user is logged in, they see the dashboard message
- If not, they're redirected before this even renders

`auth/useAuthStore.ts` — Zustand Global Store for Auth

```
import { create } from 'zustand';

interface AuthState {
  isLoggedIn: boolean;
  login: () => void;
  logout: () => void;
}

// Zustand store with localStorage persistence
export const useAuthStore = create<AuthState>((set) => ({
  isLoggedIn: localStorage.getItem('isLoggedIn') ===
'true',
  login: () => {
    localStorage.setItem('isLoggedIn', 'true');
    set({ isLoggedIn: true });
  },
  logout: () => {
    localStorage.removeItem('isLoggedIn');
    set({ isLoggedIn: false });
  },
}));
```

This replaces Redux or React Context. You can use `useAuthStore()` from any component — even deeply nested ones.

`auth/requireAuth.ts` — Route Guard Logic

```
import { redirect } from 'react-router-dom';

// This is called in the dashboard loader
export function requireAuth() {
  const loggedIn = localStorage.getItem('isLoggedIn') ===
'true';
  if (!loggedIn) {
    throw redirect('/'); // React Router redirects before
rendering the page
```

```
  }
}
```

This is **true route protection** — not just hiding a button, but stopping rendering of the protected page.

`index.css` — Tailwind Setup

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

This enables Tailwind classes like `text-3xl`, `bg-green-500`, `p-4`, etc.

How It Works Together

1. **User visits `/dashboard`**
2. The loader runs `requireAuth()`
3. If logged in: `Dashboard` renders and shows the welcome message
4. If not logged in: `redirect('/')` sends them back to Home
5. Zustand tracks login state across refresh via localStorage
6. Tailwind gives you design with no custom CSS needed

Next Part: Add a Login. Form:

`src/routes/Login.tsx`

```
import { Form, useActionData } from 'react-router-dom';

export default function Login() {
  const error = useActionData() as string | undefined;

  return (
    <div className="max-w-md mx-auto mt-10 p-6 bg-white
shadow rounded">
```

```
      <h1 className="text-2xl font-bold mb-4">Login</h1>

      <Form method="post" className="space-y-4">
        <div>
          <label className="block">Username</label>
          <input name="username" className="border px-2
py-1 w-full" />
        </div>
        <div>
          <label className="block">Password</label>
          <input type="password" name="password"
className="border px-2 py-1 w-full" />
        </div>

        {error && <p className="text-red-500">{error}</p>}

        <button type="submit" className="bg-blue-600 text-
white px-4 py-2 rounded">
          Log In
        </button>
      </Form>
    </div>
  );
}
```

src/routes/loginAction.ts

```
import { redirect } from 'react-router-dom';
import { useAuthStore } from '../auth/useAuthStore';

export async function loginAction({ request }: { request:
Request }) {
  const formData = await request.formData();
  const username = formData.get('username');
  const password = formData.get('password');

  if (username === 'admin' && password === '123') {
    useAuthStore.getState().login(); // Zustand login
    return redirect('/dashboard');
  }
  return 'Invalid credentials';
}
```

How to Test

1. Visit `/` — see Home Page
2. Click **Login**
3. Use credentials:
    - Username: `admin`
    - Password: `123`
4. You'll be redirected to `/dashboard`
5. Click **Logout** — redirected to `/`

We are using action() from Router 7

Perfect follow-up — let's walk through **exactly where and how `action()` is used in your project**, and how it connects everything together.

What is `action()`?

In React Router v6.4+, an `action()` is a function defined for a route that handles **form submissions** or **mutation logic** (like login, post creation, etc.).

You register the `action()` function on a specific route when defining your router.

Where `action()` is used in your project:

1. In `main.tsx` → You assign it to the `/login` route:

```
{
  path: 'login',
  element: <Login />,                // the login form page
  action: loginAction,              // the function that
handles the form POST
},
```

So when a user submits the form on `/login`, React Router automatically calls the `loginAction` function.

2. `loginAction` is your `action()` function

This lives in `src/routes/loginAction.ts`:

```
import { redirect } from 'react-router-dom';
import { useAuthStore } from '../auth/useAuthStore';

export async function loginAction({ request }: { request:
Request }) {
  const formData = await request.formData();
  const username = formData.get('username');
  const password = formData.get('password');

  if (username === 'admin' && password === '123') {
    useAuthStore.getState().login(); // trigger Zustand
login state
    return redirect('/dashboard');   // redirect on success
  }

  return 'Invalid credentials';      // return error for
display
}
```

It receives the `FormData` from the submission and performs logic (auth in this case).

3. In your form → `<Form method="post">`

In `Login.tsx`, you use the `<Form>` component from React Router:

```
import { Form, useActionData } from 'react-router-dom';

<Form method="post">
  <input name="username" />
  <input name="password" />
  <button type="submit">Log In</button>
</Form>
```

- When you hit submit, React Router automatically triggers the `action()` tied to the current route.
- You **don't need an `onSubmit` handler** — React Router wires it all up.

The Flow

Step	What Happens
User goes to `/login`	Sees `<Form method="post">`
User submits credentials	React Router triggers `loginAction()`
`loginAction()` runs	Checks form data, updates Zustand, redirects or returns error
Zustand updates state	`isLoggedIn = true`, UI reacts
React Router redirects	To `/dashboard` or stays on `/login`