🗂️ Folder Structure

```
src/
├── assets/                    # Static assets (images, fonts)
├── components/                # Reusable UI components
│   └── ui/                    # ShadCN UI components
├── features/                  # Feature-based modules
│   └── auth/                  # Authentication module
│       ├── components/        # Auth-related components
│       ├── pages/             # Auth pages (Login, Register)
│       └── validation/        # Zod validation schemas
├── hooks/                     # Custom React hooks
├── layouts/                   # Layout components
│   ├── MainLayout.tsx         # Main layout with sidebar, header, footer
│   └── AuthLayout.tsx         # Layout for authentication pages
├── routes/                    # Route definitions
│   └── AppRoutes.tsx          # Main routing setup
├── store/                     # Redux store setup
├── styles/                    # Global styles
│   └── globals.css            # Tailwind CSS base styles
└── App.tsx                    # Main application entry
```

⚙️ Setup Instructions

1. Initialize Vite Project

```
npm create vite@latest my-app --template react-ts
cd my-app
npm install
```

2. Install Dependencies

```
npm install react-router-dom @hookform/resolvers zod lucide-react @radix-
```

3. Configure Tailwind CSS

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init
```

In `tailwind.config.js`:

```
module.exports = {
  content: ['./index.html', './src/**/*.{js,ts,jsx,tsx}'],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

In `src/styles/globals.css`:

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

4. Initialize ShadCN UI

```
npx shadcn-ui@latest init
```

During initialization, choose the `new-york` style and enable `cssVariables`.

🎨 Theme Switching with ShadCN UI

1. Create Theme Context

In `src/hooks/useTheme.ts`:

```
import { createContext, useContext, useEffect, useState } from 'react';

type Theme = 'light' | 'dark' | 'system';

const ThemeContext = createContext<{
  theme: Theme;
```

```
  setTheme: (theme: Theme) => void;
}>({
  theme: 'system',
  setTheme: () => {},
});

export const useTheme = () => useContext(ThemeContext);

export const ThemeProvider: React.FC = ({ children }) => {
  const [theme, setTheme] = useState<Theme>('system');

  useEffect(() => {
    const savedTheme = localStorage.getItem('theme') as Theme;
    if (savedTheme) setTheme(savedTheme);
  }, []);

  useEffect(() => {
    if (theme === 'system') {
      document.documentElement.removeAttribute('data-theme');
    } else {
      document.documentElement.setAttribute('data-theme', theme);
    }
    localStorage.setItem('theme', theme);
  }, [theme]);

  return (
    <ThemeContext.Provider value=(Undefined, Undefined)>
      {children}
    </ThemeContext.Provider>
  );
};
```

2. Implement Theme Switcher Component

In `src/components/ui/ThemeSwitcher.tsx`:

```
import { Sun, Moon } from 'lucide-react';
import { useTheme } from '../../hooks/useTheme';

const ThemeSwitcher = () => {
  const { theme, setTheme } = useTheme();

  return (
```

```
      <div className="flex items-center space-x-2">
        <button onClick={() => setTheme('light')} aria-label="Light Mode">
          <Sun className={theme === 'light' ? 'text-yellow-500' : 'text-gra
        </button>
        <button onClick={() => setTheme('dark')} aria-label="Dark Mode">
          <Moon className={theme === 'dark' ? 'text-blue-500' : 'text-gray-
        </button>
      </div>
    );
  };

  export default ThemeSwitcher;
```

3. Wrap Application with ThemeProvider

In `src/App.tsx`:

```
  import { ThemeProvider } from './hooks/useTheme';
  import AppRoutes from './routes/AppRoutes';

  const App = () => (
    <ThemeProvider>
      <AppRoutes />
    </ThemeProvider>
  );

  export default App;
```

🛠️ Feature: Authentication with Zod Validation

1. Define Zod Schema

In `src/features/auth/validation/loginSchema.ts`:

```
  import { z } from 'zod';

  export const loginSchema = z.object({
    email: z.string().email(),
    password: z.string().min(6),
  });
```

```
export type LoginFormValues = z.infer<typeof loginSchema>;
```

2. Create Login Form Component

In `src/features/auth/components/LoginForm.tsx`:

```
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { loginSchema, LoginFormValues } from '../validation/loginSchema';

const LoginForm = () => {
  const { register, handleSubmit, formState: { errors } } = useForm<Logir
    resolver: zodResolver(loginSchema),
  });

  const onSubmit = (data: LoginFormValues) => {
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
      <div>
        <label htmlFor="email" className="block">Email</label>
        <input id="email" type="email" {...register('email')} className="
        {errors.email && <span>{errors.email.message}</span>}
      </div>
      <div>
        <label htmlFor="password" className="block">Password</label>
        <input id="password" type="password" {...register('password')} cl
        {errors.password && <span>{errors.password.message}</span>}
      </div>
      <button type="submit" className="btn">Login</button>
    </form>
  );
};

export default LoginForm;
```

🧭 Routing with Layouts

1. Define Routes

In `src/routes/AppRoutes.tsx`:

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'
import MainLayout from '../layouts/MainLayout';
import AuthLayout from '../layouts/AuthLayout';
import LoginPage from '../features/auth/pages/LoginPage';

const AppRoutes = () => (
  <Router>
    <Routes>
      <Route path="/login" element={<AuthLayout><LoginPage /></AuthLayout
      <Route path="/" element={<MainLayout>Home Page</MainLayout>} />
    </Routes>
  </Router>
);

export default AppRoutes;
```

2. Create Layout Components

In `src/layouts/MainLayout.tsx`:

```
import { ReactNode } from 'react';
import ThemeSwitcher from '../components/ui/ThemeSwitcher';

const MainLayout = ({ children }: { children: ReactNode }) => (
  <div className="flex flex-col min-h-screen">
    <header className="flex justify-between p-4 bg-gray-800 text-white">
      <h1>My App</h1>
      <ThemeSwitcher />
    </header>
    <main className="flex-1 p-4">{children}</main>
    <footer className="p-4 bg-gray-800 text-white">Footer</footer>
  </div>
);

export default MainLayout;
```

In `src/layouts/AuthLayout.tsx`:

```
import { ReactNode } from 'react';

const AuthLayout = ({ children }: { children: ReactNode }) => (
  <div className="flex justify-center items-center min-h-screen bg-gray-1
    <div className="p-8 bg-white shadow-lg rounded">{children}</div>
  </div>
);

export default AuthLayout;
```

This setup provides:

- **Feature-based architecture** for scalability.
- **ShadCN UI** for modern, customizable components.
- **Zod** and **React Hook Form** for type-safe form validation.
- **Theme switching** with light, dark, and system modes.
- **React Router v6** for declarative routing with layouts.

---

1. Folder Structure (Updated with Redux + Features)

```
src/
├── assets/
├── components/
│   └── ui/                    # ShadCN UI components
├── features/
│   └── auth/
│       ├── components/        # Auth components (LoginForm, Modals)
│       ├── pages/             # Pages (LoginPage, RegisterPage)
│       ├── validation/        # Zod schemas
│       ├── api/               # RTK Query API slice
│       └── store/             # Redux slice for auth
├── hooks/                     # Custom hooks
├── layouts/                   # Layouts
│   ├── MainLayout.tsx
│   └── AuthLayout.tsx
├── routes/                    # React Router routes
│   └── AppRoutes.tsx
├── store/                     # Redux store setup
│   └── index.ts
├── styles/
```

```
|   └── globals.css
└── App.tsx
```

---

2. Redux Toolkit + RTK Query Setup

a) Auth API Slice (`features/auth/api/authApi.ts`)

```ts
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';
import { LoginFormValues } from '../validation/loginSchema';

export interface AuthResponse {
  user: { id: string; email: string };
  token: string;
}

export const authApi = createApi({
  reducerPath: 'authApi',
  baseQuery: fetchBaseQuery({ baseUrl: 'https://api.example.com/' }),
  endpoints: (builder) => ({
    login: builder.mutation<AuthResponse, LoginFormValues>({
      query: (credentials) => ({
        url: 'auth/login',
        method: 'POST',
        body: credentials,
      }),
    }),
  }),
});

export const { useLoginMutation } = authApi;
```

---

b) Auth Slice (`features/auth/store/authSlice.ts`)

```ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { AuthResponse } from '../api/authApi';

interface AuthState {
  user: AuthResponse['user'] | null;
  token: string | null;
```

```
}

const initialState: AuthState = {
  user: null,
  token: null,
};

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    setCredentials: (state, action: PayloadAction<AuthResponse>) => {
      state.user = action.payload.user;
      state.token = action.payload.token;
    },
    logout: (state) => {
      state.user = null;
      state.token = null;
    },
  },
});

export const { setCredentials, logout } = authSlice.actions;
export default authSlice.reducer;
```

---

c) Root Store (`store/index.ts`)

```
import { configureStore } from '@reduxjs/toolkit';
import authReducer from '../features/auth/store/authSlice';
import { authApi } from '../features/auth/api/authApi';

export const store = configureStore({
  reducer: {
    auth: authReducer,
    [authApi.reducerPath]: authApi.reducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(authApi.middleware),
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Wrap your app with the store in `main.tsx` or `App.tsx`:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './store';

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

---

3. Using Redux + RTK Query in Login Form

```
import { useLoginMutation } from '../api/authApi';
import { useDispatch } from 'react-redux';
import { setCredentials } from '../store/authSlice';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { loginSchema, LoginFormValues } from '../validation/loginSchema';

const LoginForm = () => {
  const dispatch = useDispatch();
  const [login, { isLoading }] = useLoginMutation();
  const { register, handleSubmit, formState: { errors } } = useForm<Logir
    resolver: zodResolver(loginSchema),
  });

  const onSubmit = async (data: LoginFormValues) => {
    try {
      const userData = await login(data).unwrap();
      dispatch(setCredentials(userData));
    } catch (err) {
      console.error(err);
    }
  };
```

```
  return (
    <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
      <input {...register('email')} placeholder="Email" />
      {errors.email && <span>{errors.email.message}</span>}
      <input {...register('password')} placeholder="Password" type="passw
      {errors.password && <span>{errors.password.message}</span>}
      <button type="submit" disabled={isLoading}>Login</button>
    </form>
  );
};
```

---

4. Integrating with Layouts & Theme

- `MainLayout` → used for authenticated routes.
- `AuthLayout` → used for login/signup pages.
- Theme switcher (`useTheme`) works on any page.
- Redux store available globally for authentication state.

✅ Benefits

1. **Feature-based structure** → scalable for large apps.
2. **Redux Toolkit + RTK Query** → handles global state and API efficiently.
3. **Zod + React Hook Form** → type-safe, reusable forms.
4. **ShadCN UI** → beautiful, customizable components.
5. **Theme switching** → light/dark/system modes.
6. **Layouts + Modals** → clean separation for public vs authenticated pages.