




# React Frontend Documentation

## Comprehensive Guide to Folder Structure & Application Flow

---



### Table of Contents

1.  [Folder Structure & Purpose](#)
  2.  [Application Flow & File Relationships](#)
  3.  [Key Features & Integrations](#)
- 



## Folder Structure & Purpose

### `/Frontend` - Main Application Directory

Root directory containing all frontend application files and configurations.

### `/src` - Source Code Directory

Contains all the React application source code, components, and assets.

#### `/src/assets`

- **Purpose:** Static assets like images, icons, and SVGs
- **Contents:** React logo, Vite logo, and other visual assets used throughout the application

#### `/src/commons`

- **Purpose:** Reusable UI components that are used across multiple pages
- **Contents:** Common components like Navbar, Footer, Login, SignUp, Button, and Card components
- **Usage:** These components serve as building blocks for pages and provide consistent UI elements

#### `/src/components`

- **Purpose:** Specific feature components that implement particular functionalities
- **Contents:** ForgotPassword, VerifyEmail, SubscriptionPayment, PaymentConfirmation components
- **Usage:** These components are imported and used within pages or layouts for specific features

#### `/src/layouts`

- **Purpose:** Layout components that provide structure and routing logic
- **Contents:**
  - `AuthLayout.jsx` - Handles authentication-based routing and access control

- `Container.jsx` - Provides responsive container wrapper for content
- **Usage:** Wraps pages to provide authentication protection and consistent layout structure

#### `/src/pages`

- **Purpose:** Main page components representing different routes/views
- **Contents:** Home, Profile pages and other route-level components
- **Usage:** These are the main views that users navigate to, composed of various components and commons

#### `/src/session`

- **Purpose:** Session management and authentication-related utilities
- **Contents:** `SessionExpireAlert.jsx` - Handles token expiration warnings and session refresh
- **Usage:** Manages user session state and provides session expiration alerts

#### `/src/store`

- **Purpose:** Redux store configuration and state management
- **Contents:**
  - `store.js` - Main Redux store configuration
  - `authSlice.js` - Authentication state management with actions and reducers
- **Usage:** Centralized state management for authentication, user data, and token information

#### `/src/utils`

- **Purpose:** Utility functions and helper modules
- **Contents:** `errorHandler.js` - Centralized error handling and API call wrapper functions
- **Usage:** Provides reusable utility functions used across the application

#### `/public`

- **Purpose:** Static files served directly by the web server
- **Contents:** Vite SVG icons and other public assets

---

## Application Flow & File Relationships

### Entry Point Flow

1. `main.jsx` → Application entry point
2. `App.jsx` → Main application wrapper with routing setup

3. `index.css` → Global styles with Tailwind CSS imports

## Routing & Navigation Flow

`main.jsx`

- └ Defines router configuration with all routes
- └ Wraps app with Redux Provider
- └ Routes include:
  - └ `/` (Home)
  - └ `/login` (LogIn)
  - └ `/signup` (SignUp)
  - └ `/profile` (Profile - Protected)
  - └ `/forgot-password` (ForgotPassword)
  - └ `/verify-email` (VerifyEmail)
  - └ `/subscription-payment` (SubscriptionPayment)
  - └ `/payment-confirmation` (PaymentConfirmation - Protected)

## Authentication Flow

`App.jsx`

- └ Checks authentication status on app load
- └ Uses `safeApiCall` from `utils/errorHandler.js`
- └ Dispatches login/logout actions to `store/authSlice.js`
- └ Renders `SessionExpireAlert` for authenticated users

`AuthLayout.jsx`

- └ Protects routes based on authentication requirements
- └ Handles route redirections for authenticated/unauthenticated users
- └ Provides loading states during authentication checks
- └ Allows special routes (verification, payment) to bypass normal auth flow

## State Management Flow

`store/store.js`

- └ Configures Redux store with `authSlice`

`store/authSlice.js`

- └ Manages authentication state
- └ Handles user data, tokens, and session information
- └ Provides actions for login, logout, token refresh
- └ Used throughout the app for authentication state

## Component Usage Flow

**Commons → Pages**

- `Navbar.jsx` → Used in `App.jsx` for global navigation
- `LogIn.jsx`, `SignUp.jsx` → Used as route components in `main.jsx`
- `Button.jsx`, `Card.jsx` → Reusable UI components for any page/component

## Components → Pages/Routes

- `ForgotPassword.jsx` → Standalone route for password recovery
- `VerifyEmail.jsx` → Email verification route (bypasses auth)
- `SubscriptionPayment.jsx` → Payment processing route
- `PaymentConfirmation.jsx` → Post-payment confirmation route

## Layouts → Route Protection

- `AuthLayout.jsx` → Wraps routes requiring authentication protection
- `Container.jsx` → Provides consistent responsive layout wrapper

## Session Management

- `SessionExpireAlert.jsx` → Automatically rendered in `App.jsx` for authenticated users
- Monitors token expiration and handles session refresh

## Error Handling Flow

`utils/errorHandler.js`

- └ `safeApiCall()` → Wraps all API calls with consistent error handling
- └ `handleApiError()` → Centralized error processing and user notifications
- └ Used in `App.jsx` for authentication checks
- └ Used in `SessionExpireAlert.jsx` for token refresh
- └ Provides toast notifications for various error scenarios

## Styling & Configuration

- **Tailwind CSS:** Configured via `tailwind.config.js` and imported in `index.css`
- **PostCSS:** Configured in `postcss.config.js` for Tailwind processing
- **Vite:** Build tool configured in `vite.config.js`
- **ESLint:** Code linting configured in `eslint.config.js`

---

## Key Features & Integrations

### Authentication System

- JWT token-based authentication with automatic refresh

- Session expiration warnings with countdown timer
- Protected routes with automatic redirection
- Persistent authentication state via Redux

## **Error Handling**

- Centralized error handling for all API calls
- User-friendly error messages via toast notifications
- Automatic logout on authentication failures
- Network error detection and handling

## **State Management**

- Redux Toolkit for authentication state
- Persistent user data and token management
- Session warning states and payment status tracking

## **Routing**

- React Router DOM for navigation
- Protected routes with authentication middleware
- Special route handling for verification and payments
- Automatic redirections based on authentication status

---

**This structure provides a scalable foundation for a React application with robust authentication, error handling, and state management capabilities.**