# Mega Blog Description

This project is a modern web application that combines Appwrite, React, Redux Toolkit, React Hook Form, and Tailwind CSS with DaisyUI for a seamless user experience. It features secure authentication, user management, and post management. TinyMCE is used for rich text editing, while html-react-parser renders HTML content in the UI. Articles are styled with @tailwindcss/typography for readability. The app offers efficient backend integration, real-time validation, and a clean, responsive design.

Table of Contents

[Mega Blog 2](#_Toc184224081)

[1 - Folder Structure Overview 2](#_Toc184224082)

[2 - App Routing 2](#_Toc184224083)

[3 - Appwrite Integration (Using AuthService, DatabaseService, and StorageService) 2](#_Toc184224084)

[4 - Component Structure 3](#_Toc184224085)

[5 - Custom Hooks 3](#_Toc184224086)

[6 - html-react-parser ("html-react-parser") 3](#_Toc184224087)

[7 - React Hook Form ("react-hook-form") 4](#_Toc184224088)

[8 - Redux Toolkit for State Management 4](#_Toc184224089)

[9 - AuthLayout.tsx - Authentication and Authorization Layout 4](#_Toc184224090)

[10 - Types 5](#_Toc184224091)

[11 - Validations Using React Hook Form 5](#_Toc184224092)

[12 - Config.ts for Environment Variables 6](#_Toc184224093)

[13 - Routing and Path Management 6](#_Toc184224094)

[14 - Environment Configuration 6](#_Toc184224095)

# Mega Blog

## 1 - Folder Structure Overview

The project follows a modular and organized structure, with each folder serving a distinct purpose to maintain scalability and clarity.

**/src -** The main source folder containing all project files.

* **/app -** Contains application-specific logic and routes.
* **/appwrite -** Contains classes and services for Appwrite backend interaction (e.g., authentication, database, storage).
* **/components -** Contains reusable components used across the application.
* **/components/elements -** Contains smaller, reusable UI elements (e.g., TextInput, Button, FileInput).
* **/fonts -** Stores font files used in the application.
* **/hooks -** Custom hooks for interacting with Redux and other logic (e.g., useAuthState, usePostsState).
* **/icons -** Contains icons used in various components.
* **/store -** Contains Redux-related files, including slices and the Redux store setup.
* **/types -** Contains TypeScript types and interfaces for better type safety.
* **/validations -** Contains validation logic for forms (e.g., login, signup, and post-related validations).
* **config.ts -** Contains configuration settings, including environment variables.
* **pathNames.ts -** Centralized file for managing and defining all route paths.

## 2 - App Routing

* App routing is implemented using Next.js with dynamic routing for efficient navigation.
* Routes are defined in a separate file, **pathNames.ts**, for cleaner management and to avoid hardcoded strings in components.
* **Authentication Routing:** Pages like Login and Signup have conditional routing logic to prevent access when a user is already logged in.
* **Protected Routes:** Certain pages require authentication, and if a user is not logged in, they are redirected to the login page.

## 3 - Appwrite Integration (Using AuthService, DatabaseService, and StorageService)

**Appwrite** is used for backend services, and the integration is organized into separate service files for better maintainability and clarity.

* **AuthService (auth.ts) -** Contains asynchronous methods for user authentication, including:
  + **createAccount -** Registers a new user.
  + **Login -** Logs in an existing user.
  + **getCurrentUser -** Retrieves the currently authenticated user.
  + **logout -** Logs out the user.
* **DatabaseService (database.ts) -** Handles CRUD operations for posts with Appwrite's database, with methods for handling posts and content:
  + **createPost -** Creates a new post in the database.
  + **updatePost -** Updates an existing post.
  + **deletePost -** Deletes a post.
  + **getPost -** Fetches a specific post by its ID.
  + **getPosts -** Retrieves multiple posts, typically for listing.
* **StorageService (storage.ts) -** Handles file uploads and retrieval, including:
  + **uploadFile -** Uploads a file to Appwrite's storage.
  + **deleteFile -** Deletes a file from Appwrite's storage.
  + **getFilePreview -** Retrieves a preview of a file from storage.

By structuring your Appwrite integration into distinct service classes, you ensure that each service has a clear responsibility, making it easier to manage and extend

## 4 - Component Structure

**UI Components** are built using DaisyUI and Tailwind CSS, focusing on reusable and modular components.

**Reusable Components -**

* **TextInput, Button, FileInput, Select,** and **Loading:** Basic UI components.
* **PostCard** and **PostCardSkeleton:** Display posts and loading states.
* **RTE.tsx**: Rich text editor component for **TinyMCE**.
* **ThemeSwitcher.tsx:** Allows toggling between dark and light themes.

**AuthLayout.tsx -** Protects certain pages that require authentication, checking if the user is logged in before rendering.

**Footer.tsx -** Contains the footer element for all pages.

## 5 - Custom Hooks

Custom hooks are defined for handling Redux state in a modular manner.

* **useAuthState.ts -** Provides hooks like **useUserData,** **useAuthStatus,** and **useSyncAuthStatus** to access and manage user authentication state.
* **usePostsState.ts -** Provides hooks like **usePost** and **usePosts** for accessing and managing post data in the state.

## 6 - html-react-parser ("html-react-parser")

* A library that converts HTML strings into React components.
* Useful for rendering dynamic HTML content within React components, preserving its structure and allowing React to manage the DOM.
* Enables you to safely inject and render raw HTML inside your app

## 7 - React Hook Form ("react-hook-form")

* A library for handling form validation and state management in React applications.
* Simplifies form handling with minimal re-renders and allows easy integration with third-party libraries.
* Provides easy-to-use APIs for managing complex form validation and submission.

## 8 - Redux Toolkit for State Management

* Redux Toolkit is used for managing global application state, with slices defined for authentication and posts.
* **authSlice.ts:** Manages user authentication state, including login status and user data.
* **postsSlice.ts:** Manages post-related data, including adding, updating, deleting, and retrieving posts.
* The global store is configured using **configureStore** from **Redux Toolkit**, combining both the **auth** and **posts** reducers

## 9 - AuthLayout.tsx - Authentication and Authorization Layout

The **AuthLayout.tsx** component is responsible for managing and protecting routes that require authentication in your application. It is a higher-order component (HOC) designed to wrap around pages or components that need to ensure the user is logged in to access them. Here’s a detailed breakdown of how it works.

1. **Authentication Check:**
   * The primary function of **AuthLayout.tsx** is to check whether the user is authenticated or not. This is done by comparing the user's login status (from the Redux store) to the requirements of the page.
   * If a page requires authentication (authentication = true) and the user is not logged in **(authStatus = false),** the component will prevent access to the page. In this case, it will **redirect** the user to the **Login page** to prompt them to log in before accessing the protected route.
   * Conversely, if a page does not require authentication **(authentication = false)** but the user is already logged in **(authStatus = true),** the component will prevent the user from accessing login-related pages (like **Login** or **Signup**) and redirect them to the **Home or dashboard page** instead.
2. **Conditional Rendering:**
   * If the authentication status matches the required conditions (i.e., the user is either logged in or logged out as needed), the wrapped component is rendered, allowing the user to proceed with the operation (such as viewing posts, creating new posts, etc.).
   * This ensures that only authorized users can access specific pages, preventing unauthorized access to sensitive features like post creation, post editing, or user account management.
3. **Redirection:**
   * **AuthLayout.tsx** uses routing logic (via **useRouter** mechanism) to handle redirection. If the user is not authorized to access the current route, they are redirected to the appropriate page (either **Login or Home**).
   * This improves the user experience by ensuring that users are always directed to the correct page based on their authentication state.
4. **Flexibility:**
   * By centralizing authentication logic within **AuthLayout.tsx,** you make it easier to manage protected routes. Rather than adding authentication checks to each individual page, you wrap pages that require authentication with this layout, simplifying the overall architecture.
   * This also allows for the dynamic display of content based on user roles or access levels (if implemented), adding another layer of flexibility to your application.
5. **Usage:**
   * The **AuthLayout.tsx** component is used to wrap any page or component that requires authentication, such as user profile pages, dashboard pages, or any pages involving user-specific operations like creating or editing posts.

By implementing **AuthLayout.tsx,** you enforce proper user authentication flows, reduce redundant authentication checks across the app, and streamline the user experience when accessing protected resources.

## 10 - Types

All project types are stored in a types folder, which includes the following files.

* **AuthState.ts:** Defines types such as UserData and AuthState for managing authentication state.
* **AuthUser.ts:** Contains types like LoginData and SignupData for handling user login and signup data.
* **PostsState.ts:** Contains types like Post and PostsState for managing post data in the application.

## 11 - Validations Using React Hook Form

**React Hook Form** is used to manage the **authentication** (login/signup) and **post management** (create/update posts) forms

The library provides an efficient way to manage form state, track validation errors, and handle form submissions. This leads to a better user experience by giving immediate feedback and reducing form handling complexity.

**Structure of Validation Logic -** is organized in a separate folder called /validations, which houses two key files:

1. **authValidations.ts –** contains validation rules are defined for login and signup forms. These validations ensure that user inputs like **name**, **email**, and **password** meet specific requirements.
2. **postValidations.ts –** contains validation rules are set for creating and updating posts. The focus is on the title and slug fields, which need to meet specific requirements for consistency and SEO-friendly formatting.

## 12 - Config.ts for Environment Variables

* A **config.ts** file is used to manage and expose the environment variables to the rest of the application.
* This configuration file centralizes all the environment-specific settings (e.g., **Appwrite** **endpoint**, **TinyMCE** API key) in one place, making it easier to manage and maintain.

## 13 - Routing and Path Management

Routing in the application is handled by Next.js, with routes defined in a centralized file:

**pathNames.ts:** Contains all route paths for the application, making it easier to manage and reference them throughout the project.

## 14 - Environment Configuration

The .env file holds important environment variables for various services like Appwrite and TinyMCE etc. These environment variables are essential for connecting your frontend with backend services and ensuring that keys, endpoints, and other configurations remain secure and easy to manage. The config.ts file centralizes and maintains the environment variables for ease of use throughout the project. There are following variables:

* **NEXT\_PUBLIC\_APPWRITE\_URL**=[your-appwrite-endpoint]
* **NEXT\_PUBLIC\_APPWRITE\_PROJECT\_ID**=[your-appwrite-project-id]
* **NEXT\_PUBLIC\_APPWRITE\_DATABASE\_ID**=[your-appwrite-database-id]
* **NEXT\_PUBLIC\_APPWRITE\_COLLECTION\_ID**=[your-appwrite-collection-id]
* **NEXT\_PUBLIC\_APPWRITE\_BUCKET\_ID**=[your-appwrite-bucket-id]
* **NEXT\_PUBLIC\_APPWRITE\_API\_KEY**=[your-appwrite-api-key]
* **NEXT\_PUBLIC\_TINY\_MCE\_API\_KEY**=[your-tiny-mce-api-key]