Full Stack Web Development with Python, Django & React

LMS Project - Frontend

Working with Login, Registration, Profile Management | Forget Password, Reset Password

Md. Tausif Hossain
Founding Member & Technical Leader, DevTechGuru
Founder & CEO, TechnicalBind
Lead Instructor, Ostad

📧 Email: tausif1337@gmail.com
📱 WhatsApp: +8801748181448
📅 Date: July 07, 2025

What We’ll Learn Today

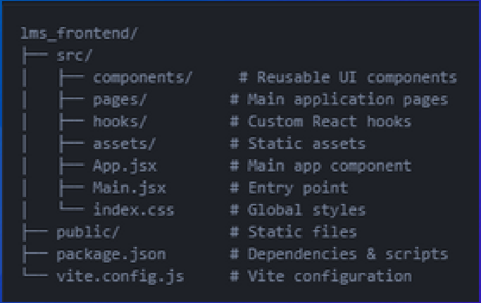
- Build a lms_frontend using React
  - Login
  - Registration
  - Profile Management
  - Dashboard
  - Users
  - Forgot Password
- Fetch data from lms_backend API with Axios
- Style with Tailwind CSS

Project Structure & Technology Stack

Core Technologies:

- **React 18** - Main UI framework
- Vite - Build tool and development server
- **Tailwind CSS** - Utility-first CSS framework
- **Axios** - HTTP client for API calls
- React Router DOM - Client-side routing
- **JWT Decode** - Token handling

Project Organization



```
lms_frontend/
├── src/
│   ├── components/      # Reusable UI components
│   ├── pages/          # Main application pages
│   ├── hooks/          # Custom React hooks
│   ├── assets/         # Static assets
│   ├── App.jsx         # Main app component
│   ├── Main.jsx        # Entry point
│   └── index.css       # Global styles
├── public/             # Static files
├── package.json        # Dependencies & scripts
└── vite.config.js      # Vite configuration
```

Key Features

Role-Based Access Control

- Admin Users: Can access Users page and manage all users
- Regular Users: Limited to dashboard and profile management
- Dynamic Navigation: Sidebar adapts based on user role

Modern UI/UX

- Tailwind CSS: Consistent, responsive design
- Smooth Animations: Transitions and hover effects
- Mobile-Friendly: Responsive design across devices
- Accessibility: Proper ARIA labels and keyboard navigation

Key Features

State Management

- Context API: Global auth state management
- Local Storage: Persistent login sessions
- Real-time Updates: Immediate UI updates after API calls

API Integration

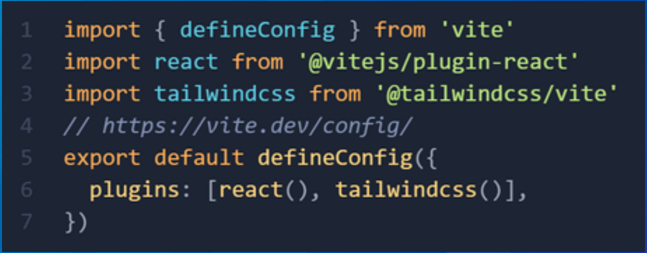
- RESTful API: Communicates with Django backend
- JWT Authentication: Secure token-based auth
- Error Handling: Graceful error messages and fallbacks

Project Setup

- npm create vite@latest lms_frontend -- --template react
- cd lms_frontend
- npm install / npm i
- npm install axios
- Install Tailwind CSS
  - npm install tailwindcss @tailwindcss/vite
- **Configure the Vite plugin**
  - Add the **@tailwindcss/vite** plugin to your Vite configuration.
- Import Tailwind CSS
  - Add an @import to your CSS file that imports Tailwind CSS.

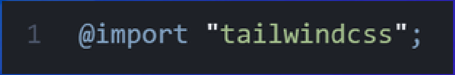
Project Setup

- Configure the Vite plugin
  - Add the @tailwindcss/vite plugin to your Vite configuration.



```js
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'
// https://vite.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss()],
})
```

- import in src/index.css



```css
@import "tailwindcss";
```

Authentication System

- **AuthContext (components/AuthContext.jsx**
  - State Management: Manages authentication token and current page
  - Token Storage: Uses localStorage for persistent login
  - API Integration: Connects to backend at http://localhost:8000/api/
  - Key Functions:
    - login() - Authenticates users and stores JWT token
    - logout() - Clears token and redirects to login
    - register() - Creates new user accounts
    - goTo() - Navigation between auth pages
- **useAuth Hook (hooks/useAuth.js)**
    - Provides easy access to authentication context throughout the app
    - Used by all components that need auth state

UI Components

Core Components:

- Sidebar (components/Sidebar.jsx)
  - Collapsible navigation sidebar
  - Dynamic navigation items based on user role
  - Responsive design with smooth animations
- Topbar (components/Topbar.jsx)
  - User profile dropdown
  - Avatar display with user info
  - Logout functionality
- Avatar (components/Avatar.jsx)
  - Generates initials-based avatars
  - Consistent styling with blue theme

Application Pages

Authentication Pages:

- LoginPage (pages/LoginPage.jsx)
    - Username/password authentication
    - Password visibility toggle
    - Error handling and validation
    - Links to registration and forgot password
- RegisterPage (pages/RegisterPage.jsx)
    - User registration with role selection
    - Password confirmation
    - Form validation
    - Role options: Student, Teacher, Admin

Main Application Pages

DashboardPage (pages/DashboardPage.jsx)

- Main Layout: Combines Sidebar, Topbar, and content area
- Role-Based Access: Shows different navigation for admins vs regular users
- Content Routing: Manages which page content to display
- User Profile Fetching: Loads user data on mount
- Responsive Design: Adapts to different screen sizes

ProfilePage (pages/ProfilePage.jsx)

- **User Profile Display**: Shows current user information
- **Profile Editing**: Inline form for updating user details
- **Password Management**: Optional password updates
- **Real-time Updates**: Refreshes data after successful updates

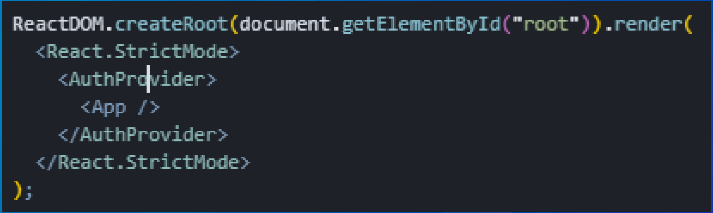
Main Application Pages

UsersPage (pages/UsersPage.jsx)

- Admin-Only Access: Restricted to admin users
- User Management: Displays all users in a table format
- Role Display: Shows user roles and information
- Access Control: Redirects non-admin users

Application Flow

Entry Point (Main.jsx)



```
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <AuthProvider>
      <App />
    </AuthProvider>
  </React.StrictMode>
);
```

Main App Logic (App.jsx)

- Conditional Rendering: Shows dashboard if authenticated, auth pages if not
- Page Navigation: Manages transitions between login, register, and forgot password
- Responsive Layout: Beautiful gradient background with centered auth forms

Auth Context

```
import { createContext } from "react";
export const AuthContext = createContext();
```

- The Context API in React is a built-in feature that provides a way to share data across the component tree without manually passing props down through every level (known as "prop drilling"). It is particularly useful for managing global state or data that many components need to access, such as themes, user authentication status, or localization settings.

- In React, AuthContext refers to a specific implementation of the React Context API designed to manage and share authentication-related state and functionality throughout an application.