# 1. Executive Summary

## 1.1 Purpose

This document describes the software architecture of an enterprise-grade React TypeScript application built with scalability, maintainability, and type safety as core principles. The architecture supports multilingual applications, comprehensive error handling, and follows industry best practices.

## 1.2 Scope

The architecture covers:

Frontend application structure and organization

State management patterns

API integration layer

Authentication and authorization

UI component library with Material-UI wrappers

Internationalization support

Error handling and notifications

Theme management (light/dark mode)

## 1.3 Target Audience

Frontend Developers

Technical Architects

QA Engineers

DevOps Engineers

Project Managers

## 1.4 Key Features

**Type-Safe Architecture -** Full TypeScript support with strict typing  
**Scalable Structure -** Feature-based organization  
**Material-UI Wrappers -** Custom component library  
**Multilingual Support -** i18next integration  
**React Query Integration -** Efficient data fetching and caching  
**Protected Routing -** Authentication-based access control  
**Theme Management -** Light/Dark mode with persistence  
**Error Boundaries -** Graceful error handling  
**Form Validation -** Formik + Yup integration  
**Code Standards -** ESLint + Prettier configuration

# 2. Architecture Overview

## 2.1 High-Level Architecture Diagram

text

┌─────────────────────────────────────────────────────────────────┐

│ PRESENTATION LAYER │

│ ┌────────────────────────────────────────────────────────────┐ │

│ │ React Components │ │

│ │ ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐ │ │

│ │ │ Pages │ │ Layouts │ │ Wrappers │ │ Common │ │ │

│ │ └──────────┘ └──────────┘ └──────────┘ └──────────┘ │ │

│ └────────────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────────┘

↕

┌─────────────────────────────────────────────────────────────────┐

│ STATE MANAGEMENT LAYER │

│ ┌────────────────────────────────────────────────────────────┐ │

│ │ React Query │ Zustand Store │ Context API │ Hooks │ │

│ └────────────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────────┘

↕

┌─────────────────────────────────────────────────────────────────┐

│ BUSINESS LOGIC LAYER │

│ ┌────────────────────────────────────────────────────────────┐ │

│ │ Custom Hooks │ Services │ Utilities │ Validators │ │

│ └────────────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────────┘

↕

┌─────────────────────────────────────────────────────────────────┐

│ DATA ACCESS LAYER │

│ ┌────────────────────────────────────────────────────────────┐ │

│ │ Axios Instance → API Interceptors → Backend │ │

│ └────────────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────────┘

↕

┌─────────────────────────────────────────────────────────────────┐

│ BACKEND API │

│ (External System) │

└─────────────────────────────────────────────────────────────────┘

## 2.2 Architecture Principles

**Separation of Concerns -** Each layer has a distinct responsibility

**Single Responsibility -** Components and functions have one clear purpose

**DRY (Don't Repeat Yourself) -** Reusable components and utilities

**Type Safety -** Strict TypeScript enforcement

**Feature-Based Organization -** Code organized by business features

**Dependency Injection -** Services injected rather than instantiated

**Immutability -** State updates follow immutable patterns

# 3. System Architecture

# 3.1 Technology Stack

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Technology | Version | Purpose |
| Build Tool | Vite | 5.4.8 | Fast development server and build tool |
| Framework | React | 18.3.1 | UI library for building component-based interfaces |
| Language | TypeScript | 5.6.2 | Type-safe JavaScript superset |
| UI Library | Material-UI | 6.1.3 | Component library |
| State Management | Zustand | 5.0.0 | Lightweight state management |
| Data Fetching | React Query | 5.59.8 | Server state management |
| HTTP Client | Axios | 1.7.7 | Promise-based HTTP client |
| Routing | React Router | 6.27.0 | Client-side routing |
| Forms | Formik | 2.4.6 | Form management |
| Validation | Yup | 1.4.0 | Schema validation |
| i18n | react-i18next | 15.0.2 | Internationalization |
| Notifications | Notistack | 3.0.1 | Toast notifications |
| Linting | ESLint | 9.12.0 | Code quality |
| Formatting | Prettier | 3.3.3 | Code formatting |

## 3.2 System Components Diagram

text

┌───────────────────────────────────────────────────────────────┐

│ Application │

│ ┌──────────────────────────────────────────────────────────┐ │

│ │ App.tsx (Root Component) │ │

│ │ - QueryClientProvider │ │

│ │ - ThemeContextProvider │ │

│ │ - NotificationProvider │ │

│ │ - BrowserRouter │ │

│ └──────────────────────────────────────────────────────────┘ │

└───────────────────────────────────────────────────────────────┘

↓

┌───────────────────────────────────────────────────────────────┐

│ Route Handler │

│ ┌──────────────────────────────────────────────────────────┐ │

│ │ AppRoutes.tsx │ │

│ │ ├─ Public Routes (AuthLayout) │ │

│ │ │ ├─ Login │ │

│ │ │ ├─ Register │ │

│ │ │ └─ Unauthorized │ │

│ │ │ │ │

│ │ └─ Protected Routes (MainLayout) │ │

│ │ ├─ Dashboard │ │

│ │ ├─ Profile │ │

│ │ └─ Settings │ │

│ └──────────────────────────────────────────────────────────┘ │

└───────────────────────────────────────────────────────────────┘

↓

┌───────────────────────────────────────────────────────────────┐

│ Feature Modules │

│ ┌──────────────────────────────────────────────────────────┐ │

│ │ Feature Structure (per feature) │ │

│ │ ├─ components/ │ │

│ │ ├─ hooks/ │ │

│ │ ├─ services/ │ │

│ │ ├─ types/ │ │

│ │ └─ pages/ │ │

│ └──────────────────────────────────────────────────────────┘ │

└───────────────────────────────────────────────────────────────┘

# 4. Folder Structure

## 4.1 Complete Directory Tree

text

project-root/

├── public/ # Static assets

├── src/

│ ├── core/ # Core application configuration

│ │ ├── config/ # Configuration files

│ │ │ ├── axios.config.ts # Axios instance setup

│ │ │ ├── react-query.config.ts # React Query configuration

│ │ │ ├── theme.config.ts # Material-UI theme

│ │ │ └── index.ts # Barrel export

│ │ │

│ │ ├── contexts/ # React contexts

│ │ │ ├── ThemeContext.tsx # Theme mode management

│ │ │ └── index.ts

│ │ │

│ │ ├── i18n/ # Internationalization

│ │ │ ├── locales/ # Translation files

│ │ │ │ ├── en/

│ │ │ │ │ ├── common.json

│ │ │ │ │ ├── validation.json

│ │ │ │ │ └── messages.json

│ │ │ │ ├── es/

│ │ │ │ └── fr/

│ │ │ └── i18n.config.ts # i18next configuration

│ │ │

│ │ ├── types/ # Global TypeScript types

│ │ │ ├── api.types.ts

│ │ │ ├── common.types.ts

│ │ │ └── index.ts

│ │ │

│ │ └── constants/ # Application constants

│ │ ├── routes.constants.ts

│ │ ├── app.constants.ts

│ │ └── index.ts

│ │

│ ├── components/ # Reusable components

│ │ ├── wrappers/ # Material-UI wrapper components

│ │ │ ├── Button/

│ │ │ │ ├── Button.tsx

│ │ │ │ ├── Button.types.ts

│ │ │ │ └── index.ts

│ │ │ ├── TextField/

│ │ │ └── index.ts

│ │ │

│ │ ├── common/ # Common components

│ │ │ ├── ErrorBoundary/

│ │ │ ├── LoadingFallback/

│ │ │ ├── NotificationProvider/

│ │ │ └── ProtectedRoute/

│ │ │

│ │ └── layout/ # Layout components

│ │ ├── MainLayout/

│ │ ├── AuthLayout/

│ │ └── index.ts

│ │

│ ├── features/ # Feature modules

│ │ ├── auth/

│ │ │ ├── components/ # Feature-specific components

│ │ │ ├── hooks/ # Custom hooks

│ │ │ │ └── useAuth.ts

│ │ │ ├── services/ # API services

│ │ │ │ └── auth.service.ts

│ │ │ ├── types/ # Feature types

│ │ │ │ └── auth.types.ts

│ │ │ └── pages/ # Feature pages

│ │ │ ├── Login.tsx

│ │ │ └── Register.tsx

│ │ │

│ │ ├── dashboard/

│ │ ├── profile/

│ │ └── settings/

│ │

│ ├── hooks/ # Global custom hooks

│ │ ├── useNotification.ts

│ │ ├── useLocalStorage.ts

│ │ └── index.ts

│ │

│ ├── services/ # Global services

│ │ ├── api/

│ │ │ ├── axios.instance.ts

│ │ │ └── interceptors.ts

│ │ ├── base.service.ts # Base service class

│ │ └── index.ts

│ │

│ ├── routes/ # Routing configuration

│ │ ├── AppRoutes.tsx # Main route configuration

│ │ ├── PublicRoutes.tsx # Public route wrapper

│ │ └── PrivateRoutes.tsx # Private route wrapper

│ │

│ ├── store/ # Global state management

│ │ └── auth.store.ts # Zustand auth store

│ │

│ ├── utils/ # Utility functions

│ │ ├── helpers.ts # Helper functions

│ │ ├── validators.ts # Validation utilities

│ │ └── index.ts

│ │

│ ├── App.tsx # Root component

│ ├── main.tsx # Application entry point

│ └── vite-env.d.ts # Vite environment types

│

├── .env # Environment variables

├── .env.example # Example environment file

├── .eslintrc.cjs # ESLint configuration

├── .prettierrc # Prettier configuration

├── tsconfig.json # TypeScript configuration

├── tsconfig.node.json # TypeScript Node configuration

├── vite.config.ts # Vite configuration

└── package.json # Project dependencies

## 4.2 Folder Organization Principles

### 4.2.1 Core (src/core/)

Contains application-wide configurations, constants, types, and contexts. Nothing in this folder should depend on features.

### 4.2.2 Components (src/components/)

**wrappers/**: Material-UI component wrappers for consistent behavior

**common/**: Shared components used across multiple features

**layout/**: Page layout components (headers, footers, sidebars)

### 4.2.3 Features (src/features/)

Feature-based organization where each feature is self-contained:

Components specific to the feature

Custom hooks for business logic

Service layer for API calls

Types specific to the feature

Page components

### 4.2.4 Services (src/services/)

Base service class with CRUD operations

Axios configuration and interceptors

Feature-specific services extend base service

# 5. Core Components

## 5.1 Application Entry Point Flow

text

main.tsx (Entry Point)

↓

App.tsx (Root Component)

↓

Providers Setup:

├─ QueryClientProvider (React Query)

├─ ThemeContextProvider (Theme Management)

├─ NotificationProvider (Toasts)

└─ BrowserRouter (Routing)

↓

AppRoutes.tsx

↓

Routes Resolution

├─ Public Routes (with AuthLayout)

└─ Protected Routes (with MainLayout)

↓

Feature Pages

## 5.2 Component Architecture

### 5.2.1 Component Hierarchy

text

Layout Components

├─ MainLayout (Protected pages)

│ ├─ AppBar

│ │ ├─ Theme Toggle

│ │ ├─ Language Switcher

│ │ └─ User Menu

│ └─ Content Area (Outlet)

│

└─ AuthLayout (Public pages)

├─ AppBar (Simple)

├─ Content Area (Outlet)

└─ Footer

### 5.2.2 Wrapper Components Pattern

All Material-UI components are wrapped to provide:

Consistent styling

Extended functionality

Type safety

Loading states

Error handling

**Example: Button Wrapper**

typescript

// src/components/wrappers/Button/Button.tsx

interface CustomButtonProps extends Omit<MuiButtonProps, 'variant'> {

variant?: 'contained' | 'outlined' | 'text' | 'primary' | 'secondary';

loading?: boolean;

}

export const Button: React.FC<CustomButtonProps> = ({

children,

loading = false,

disabled = false,

...rest

}) => {

return (

<MuiButton

disabled={disabled || loading}

startIcon={loading ? <CircularProgress size={20} /> : rest.startIcon}

{...rest}

>

{children}

</MuiButton>

);

};

**5.3 Layout Components**

**5.3.1 MainLayout (Protected Pages)**

**Purpose**: Provides layout for authenticated users

**Features**:

Top navigation bar with app title

Theme toggle button

Language switcher

User menu (profile, settings, logout)

Content area with Outlet for nested routes

**Location**: src/components/layout/MainLayout/MainLayout.tsx

**5.3.2 AuthLayout (Public Pages)**

**Purpose**: Provides layout for authentication pages

**Features**:

Simple header with app title

Language switcher

Centered content area

Footer with copyright

**Location**: src/components/layout/AuthLayout/AuthLayout.tsx

**6. Data Flow & State Management**

**6.1 State Management Architecture**

text

┌──────────────────────────────────────────────────────────┐

│ STATE MANAGEMENT │

│ │

│ ┌─────────────────┐ ┌─────────────────┐ │

│ │ Server State │ │ Client State │ │

│ │ (React Query) │ │ (Zustand) │ │

│ └─────────────────┘ └─────────────────┘ │

│ │ │ │

│ ↓ ↓ │

│ ┌─────────────────┐ ┌─────────────────┐ │

│ │ - User Data │ │ - Auth State │ │

│ │ - Dashboard │ │ - User Info │ │

│ │ - Profile │ │ - Tokens │ │

│ │ - Settings │ │ │ │

│ └─────────────────┘ └─────────────────┘ │

│ │

│ ┌──────────────────────────────────────┐ │

│ │ Context API │ │

│ │ - Theme Mode (Light/Dark) │ │

│ └──────────────────────────────────────┘ │

│ │

│ ┌──────────────────────────────────────┐ │

│ │ Local Storage │ │

│ │ - Theme Preference │ │

│ │ - Language Preference │ │

│ │ - User Settings │ │

│ │ - Auth Tokens │ │

│ └──────────────────────────────────────┘ │

└──────────────────────────────────────────────────────────┘

**6.2 React Query (Server State)**

**Purpose**: Manage server-side data with caching, synchronization, and background updates

**Usage Pattern**:

typescript

*// In custom hook*

**export** **const** useAuth = (): UseAuthReturn => {

**const** loginMutation = useMutation({

mutationFn: authService.login,

onSuccess: (response) => {

*// Handle success*

},

onError: (error) => {

*// Handle error*

},

});

**return** {

login: loginMutation.mutateAsync,

isLoading: loginMutation.isPending,

};

};

**Configuration**: src/core/config/react-query.config.ts

**6.3 Zustand (Client State)**

**Purpose**: Lightweight state management for auth and global client state

**Usage Pattern**:

typescript

*// Store definition*

**export** **const** useAuthStore = create<AuthState>()(

persist(

(set) => ({

user: **null**,

accessToken: **null**,

isAuthenticated: false,

setAuth: (user, accessToken, refreshToken) => {

set({ user, accessToken, refreshToken, isAuthenticated: true });

},

clearAuth: () => {

set({ user: **null**, accessToken: **null**, isAuthenticated: false });

},

}),

{ name: 'auth-storage' }

)

);

*// Usage in components*

**const** { user, isAuthenticated, setAuth } = useAuthStore();

**Location**: src/store/auth.store.ts

**6.4 Context API (UI State)**

**Purpose**: Manage UI-related state like theme mode

**Implementation**:

typescript

*// Theme Context*

**export** **const** ThemeContextProvider: React.FC = ({ children }) => {

**const** [mode, setMode] = useLocalStorage<ThemeMode>('theme-mode', 'light');

**const** toggleTheme = () => {

setMode((prev) => (prev === 'light' ? 'dark' : 'light'));

};

**const** theme = useMemo(() => createAppTheme(mode), [mode]);

**return** (

<ThemeContext.Provider value={{ mode, toggleTheme, setTheme }}>

<ThemeProvider theme={theme}>{children}</ThemeProvider>

</ThemeContext.Provider>

);

};

**Location**: src/core/contexts/ThemeContext.tsx

**6.5 Data Flow Diagram**

text

User Action

↓

Component Event Handler

↓

Custom Hook (Business Logic)

↓

Service Layer (API Call)

↓

Axios Instance → Interceptors → Backend API

↓

Response

↓

React Query Cache Update

↓

Component Re-render

↓

UI Update

**7. Routing Architecture**

**7.1 Route Structure**

text

Application Routes

│

├─ Public Routes (AuthLayout)

│ ├─ /login → Login Page

│ ├─ /register → Register Page

│ └─ /unauthorized → Unauthorized Page

│

├─ Protected Routes (MainLayout + ProtectedRoute)

│ ├─ / → Redirect to /dashboard

│ ├─ /dashboard → Dashboard Page

│ ├─ /profile → Profile Page

│ └─ /settings → Settings Page

│

└─ Error Routes

└─ /\* → 404 Not Found Page

**7.2 Route Protection Flow**

text

User navigates to /dashboard

↓

ProtectedRoute component checks authentication

↓

Is user authenticated?

├─ YES → Render MainLayout + Dashboard

│

└─ NO → Redirect to /login

**7.3 Route Configuration**

**Location**: src/routes/AppRoutes.tsx

typescript

**export** **const** AppRoutes: React.FC = () => {

**return** (

<ErrorBoundary>

<Suspense fallback={<LoadingFallback fullScreen />}>

<Routes>

{*/\* Public routes with AuthLayout \*/*}

<Route element={<AuthLayout />}>

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

</Route>

{*/\* Protected routes with MainLayout \*/*}

<Route element={<ProtectedRoute />}>

<Route element={<MainLayout />}>

<Route path="/dashboard" element={<Dashboard />} />

<Route path="/profile" element={<Profile />} />

<Route path="/settings" element={<Settings />} />

</Route>

</Route>

{*/\* 404 route \*/*}

<Route path="\*" element={<NotFound />} />

</Routes>

</Suspense>

</ErrorBoundary>

);

};

**7.4 Protected Route Implementation**

typescript

**export** **const** ProtectedRoute: React.FC<ProtectedRouteProps> = ({

requiredRoles = [],

redirectPath = '/login',

}) => {

**const** { isAuthenticated, user } = useAuthStore();

*// Check authentication*

**if** (!isAuthenticated) {

**return** <Navigate to={redirectPath} replace />;

}

*// Check roles*

**if** (requiredRoles.length > 0 && user) {

**const** hasRequiredRole = requiredRoles.some((role) =>

user.roles.includes(role)

);

**if** (!hasRequiredRole) {

**return** <Navigate to="/unauthorized" replace />;

}

}

**return** <Outlet />;

};

**8. API Integration Layer**

**8.1 API Architecture**

text

Component/Hook

↓

Feature Service (e.g., authService.login())

↓

Base Service (CRUD operations)

↓

Axios Instance (with interceptors)

↓

Backend API

**8.2 Service Layer Pattern**

**8.2.1 Base Service**

**Location**: src/services/base.service.ts

Provides generic CRUD operations:

getAll(params) - Fetch list with pagination

getById(id) - Fetch single item

create(data) - Create new item

update(id, data) - Update existing item

delete(id) - Delete item

typescript

**export** **class** BaseService<T> {

**protected** endpoint: string;

constructor(endpoint: string) {

**this**.endpoint = endpoint;

}

**async** getAll(params?: QueryParams): Promise<PaginatedResponse<T>> {

**const** response = **await** axiosInstance.get(**this**.endpoint, { params });

**return** response.data.data;

}

**async** getById(id: string | number): Promise<T> {

**const** response = **await** axiosInstance.get(`${**this**.endpoint}/${id}`);

**return** response.data.data;

}

**async** create(data: Partial<T>): Promise<T> {

**const** response = **await** axiosInstance.post(**this**.endpoint, data);

**return** response.data.data;

}

**async** update(id: string | number, data: Partial<T>): Promise<T> {

**const** response = **await** axiosInstance.put(`${**this**.endpoint}/${id}`, data);

**return** response.data.data;

}

**async** **delete**(id: string | number): Promise<**void**> {

**await** axiosInstance.delete(`${**this**.endpoint}/${id}`);

}

}

**8.2.2 Feature Service**

**Location**: src/features/{feature}/services/{feature}.service.ts

Extends BaseService with feature-specific methods:

typescript

**class** AuthService **extends** BaseService<AuthResponse> {

constructor() {

**super**('/auth');

}

**async** login(credentials: LoginRequest): Promise<AuthResponse> {

**const** response = **await** axiosInstance.post(

`${**this**.endpoint}/login`,

credentials

);

**return** response.data.data;

}

**async** register(data: RegisterRequest): Promise<AuthResponse> {

**const** response = **await** axiosInstance.post(

`${**this**.endpoint}/register`,

data

);

**return** response.data.data;

}

}

**export** **const** authService = **new** AuthService();

**8.3 Axios Configuration**

**8.3.1 Instance Setup**

**Location**: src/core/config/axios.config.ts

typescript

**export** **const** createAxiosInstance = (): AxiosInstance => {

**const** instance = axios.create({

baseURL: **import**.meta.env.VITE\_API\_BASE\_URL,

timeout: 30000,

headers: {

'Content-Type': 'application/json',

},

});

*// Request interceptor - Add auth token*

instance.interceptors.request.use(

(config) => {

**const** token = localStorage.getItem('accessToken');

**if** (token && config.headers) {

config.headers.Authorization = `Bearer ${token}`;

}

**return** config;

}

);

*// Response interceptor - Handle errors and token refresh*

instance.interceptors.response.use(

(response) => response,

**async** (error) => {

*// Handle 401 and token refresh*

**if** (error.response?.status === 401) {

*// Token refresh logic*

}

**return** Promise.reject(error);

}

);

**return** instance;

};

**8.3.2 API Response Format**

All API responses follow this structure:

typescript

**interface** ApiResponse<T> {

data: T;

message: string;

success: boolean;

statusCode: number;

}

*// Example response*

{

"data": {

"user": { ... },

"accessToken": "...",

"refreshToken": "..."

},

"message": "Login successful",

"success": true,

"statusCode": 200

}

**8.4 React Query Integration**

typescript

*// In custom hook*

**export** **const** useAuth = () => {

**const** { showSuccess, showError } = useNotification();

**const** loginMutation = useMutation({

mutationFn: authService.login,

onSuccess: (response) => {

setAuth(response.user, response.accessToken, response.refreshToken);

showSuccess('Login successful');

navigate('/dashboard');

},

onError: (error: Error) => {

showError(error.message);

},

});

**return** {

login: loginMutation.mutateAsync,

isLoading: loginMutation.isPending,

};

};

**9. Internationalization (i18n)**

**9.1 i18n Architecture**

text

i18next Configuration

↓

Language Detector (Browser/LocalStorage)

↓

Load Translation Resources

├─ English (en)

├─ Spanish (es)

└─ French (fr)

↓

React Components

↓

useTranslation() Hook

↓

Translated UI

**9.2 Configuration**

**Location**: src/core/i18n/i18n.config.ts

typescript

**import** i18n **from** 'i18next';

**import** { initReactI18next } **from** 'react-i18next';

**import** LanguageDetector **from** 'i18next-browser-languagedetector';

*// Import translations*

**import** enCommon **from** './locales/en/common.json';

**import** enValidation **from** './locales/en/validation.json';

**import** enMessages **from** './locales/en/messages.json';

**const** resources = {

en: {

common: enCommon,

validation: enValidation,

messages: enMessages,

},

es: { */\* ... \*/* },

fr: { */\* ... \*/* },

};

i18n

.use(LanguageDetector)

.use(initReactI18next)

.init({

resources,

defaultNS: 'common',

fallbackLng: 'en',

supportedLngs: ['en', 'es', 'fr'],

});

**9.3 Translation File Structure**

text

src/core/i18n/locales/

├─ en/

│ ├─ common.json # Common UI labels

│ ├─ validation.json # Validation messages

│ └─ messages.json # Success/error messages

├─ es/

│ ├─ common.json

│ ├─ validation.json

│ └─ messages.json

└─ fr/

├─ common.json

├─ validation.json

└─ messages.json

**9.4 Usage in Components**

typescript

*// Import hook*

**import** { useTranslation } **from** 'react-i18next';

*// In component*

**const** MyComponent: React.FC = () => {

**const** { t, i18n } = useTranslation('common');

*// Use translations*

**return** (

<Typography>{t('navigation.dashboard')}</Typography>

);

*// Change language*

**const** changeLanguage = (lang: string) => {

i18n.changeLanguage(lang);

};

};

**9.5 Adding New Language**

Create new folder: src/core/i18n/locales/{language\_code}/

Add translation files: common.json, validation.json, messages.json

Import translations in i18n.config.ts

Add to resources object

Add to supportedLngs array

Update language switcher UI

**10. Error Handling & Logging**

**10.1 Error Handling Strategy**

text

Error Occurs

↓

Error Boundary (React errors)

├─ Logs error

├─ Shows fallback UI

└─ Optional: Send to monitoring service

OR

API Error (Axios)

├─ Interceptor catches error

├─ Shows notification

├─ Logs to console

└─ Optional: Send to monitoring service

OR

Form Validation Error (Yup)

├─ Formik handles error

└─ Shows inline error message

**10.2 Error Boundary**

**Location**: src/components/common/ErrorBoundary/ErrorBoundary.tsx

typescript

**export** **class** ErrorBoundary **extends** Component<Props, State> {

**static** getDerivedStateFromError(error: Error): State {

**return** { hasError: true, error };

}

componentDidCatch(error: Error, errorInfo: ErrorInfo): **void** {

console.error('ErrorBoundary caught:', error, errorInfo);

*// Optional: Send to error monitoring service*

*// logErrorToService(error, errorInfo);*

}

render(): ReactNode {

**if** (**this**.state.hasError) {

**return** (

<Container>

<Typography variant="h4">Oops! Something went wrong</Typography>

<Button onClick={**this**.handleReset}>Try Again</Button>

</Container>

);

}

**return** **this**.props.children;

}

}

**10.3 API Error Handling**

Errors from API calls are handled in multiple layers:

**Axios Interceptor**: Catches all HTTP errors

**React Query**: Handles query/mutation errors

**Custom Hooks**: Shows notifications

**Components**: Displays error UI states

**10.4 Notification System**

**Location**: src/hooks/useNotification.ts

typescript

**export** **const** useNotification = () => {

**const** { enqueueSnackbar } = useSnackbar();

**return** {

showSuccess: (message: string) =>

enqueueSnackbar(message, { variant: 'success' }),

showError: (message: string) =>

enqueueSnackbar(message, { variant: 'error' }),

showWarning: (message: string) =>

enqueueSnackbar(message, { variant: 'warning' }),

showInfo: (message: string) =>

enqueueSnackbar(message, { variant: 'info' }),

};

};

**11. Theme & Styling**

**11.1 Theme Architecture**

text

Theme Context Provider

↓

useLocalStorage (persist theme mode)

↓

createAppTheme(mode)

↓

Material-UI ThemeProvider

↓

All Components (styled with theme)

**11.2 Theme Configuration**

**Location**: src/core/config/theme.config.ts

typescript

**export** **const** createAppTheme = (mode: PaletteMode): Theme => {

**return** createTheme({

palette: {

mode,

primary: {

main: mode === 'light' ? '#1976d2' : '#90caf9',

},

background: {

**default**: mode === 'light' ? '#f5f5f5' : '#121212',

paper: mode === 'light' ? '#ffffff' : '#1e1e1e',

},

},

components: {

MuiButton: {

styleOverrides: {

root: {

textTransform: 'none',

fontWeight: 600,

},

},

},

},

});

};

**11.3 Theme Context**

**Location**: src/core/contexts/ThemeContext.tsx

Manages theme mode (light/dark) and provides:

mode - Current theme mode

toggleTheme() - Toggle between light and dark

setTheme(mode) - Set specific theme mode

**11.4 Using Theme in Components**

typescript

*// Using theme values*

**import** { useTheme } **from** '@mui/material/styles';

**const** MyComponent = () => {

**const** theme = useTheme();

**return** (

<Box sx={{ bgcolor: theme.palette.background.paper }}>

<Typography color={theme.palette.text.primary}>

Hello World

</Typography>

</Box>

);

};

*// Using theme context*

**import** { useThemeContext } **from** '@core/contexts';

**const** ThemeToggle = () => {

**const** { mode, toggleTheme } = useThemeContext();

**return** (

<IconButton onClick={toggleTheme}>

{mode === 'dark' ? <LightModeIcon /> : <DarkModeIcon />}

</IconButton>

);

};

**12. Type Safety & Validation**

**12.1 TypeScript Configuration**

**Location**: tsconfig.json

Key settings:

json

{

"compilerOptions": {

"strict": true,

"noUnusedLocals": true,

"noUnusedParameters": true,

"noFallthroughCasesInSwitch": true,

"paths": {

"@/\*": ["./src/\*"],

"@components/\*": ["./src/components/\*"],

"@features/\*": ["./src/features/\*"],

*// ... other path aliases*

}

}

}

**12.2 Type Organization**

text

Global Types (src/core/types/)

├─ common.types.ts # ApiResponse, PaginatedResponse, etc.

├─ api.types.ts # HTTP methods, request configs

└─ index.ts # Barrel export

Feature Types (src/features/{feature}/types/)

└─ {feature}.types.ts # Feature-specific interfaces

Component Types

└─ {Component}.types.ts # Component prop interfaces

**12.3 Form Validation with Yup**

typescript

**import** \* **as** Yup **from** 'yup';

**const** validationSchema = Yup.object({

name: Yup.string()

.min(2, 'Name must be at least 2 characters')

.max(50, 'Name must be less than 50 characters')

.required('Name is required'),

email: Yup.string()

.email('Invalid email address')

.required('Email is required'),

password: Yup.string()

.min(8, 'Password must be at least 8 characters')

.matches(

/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])/,

'Must contain uppercase, lowercase, number, and special character'

)

.required('Password is required'),

});

**12.4 Runtime Validation Utilities**

**Location**: src/utils/validators.ts

Provides validation functions:

validateEmail(email) - Email format validation

validatePassword(password) - Password strength check

validatePhone(phone) - Phone number format

isNumeric(value) - Check if numeric

isRequired(value) - Check if not empty

**13. Development Guidelines**

**13.1 Code Style**

**13.1.1 ESLint Configuration**

**Location**: .eslintrc.cjs

Rules enforced:

TypeScript recommended rules

React hooks rules

No unused variables

Consistent import order

**13.1.2 Prettier Configuration**

**Location**: .prettierrc

json

{

"semi": true,

"trailingComma": "es5",

"singleQuote": true,

"printWidth": 100,

"tabWidth": 2

}

**13.2 Naming Conventions**

| **Element** | **Convention** | **Example** |
| --- | --- | --- |
| **Components** | PascalCase | UserProfile.tsx |
| **Hooks** | camelCase with 'use' prefix | useAuth.ts |
| **Services** | camelCase with 'Service' suffix | authService.ts |
| **Types/Interfaces** | PascalCase | UserProfile, ApiResponse<T> |
| **Constants** | UPPER\_SNAKE\_CASE | API\_BASE\_URL |
| **Functions** | camelCase | handleSubmit() |
| **Variables** | camelCase | userName |
| **Files** | kebab-case or PascalCase | user-profile.tsx, UserProfile.tsx |

**13.3 Component Structure**

typescript

*/\*\**

*\* Component description*

*\* Brief explanation of component purpose*

*\*/*

**import** React **from** 'react';

**import** { */\* imports \*/* } **from** 'library';

*/\*\**

*\* Component props interface*

*\*/*

**interface** ComponentProps {

prop1: string;

prop2?: number; *// Optional prop*

}

*/\*\**

*\* Component implementation*

*\* @param props - Component properties*

*\* @returns JSX Element*

*\*/*

**export** **const** Component: React.FC<ComponentProps> = ({ prop1, prop2 }) => {

*// 1. Hooks*

**const** [state, setState] = useState<Type>(initialValue);

*// 2. Event handlers*

**const** handleClick = (): **void** => {

*// Handler logic*

};

*// 3. Effects*

useEffect(() => {

*// Effect logic*

}, [dependencies]);

*// 4. Render*

**return** (

<div>

{*/\* JSX \*/*}

</div>

);

};

**export** **default** Component;

**13.4 Custom Hook Structure**

typescript

*/\*\**

*\* Hook description*

*\* Explanation of hook purpose and usage*

*\*/*

**import** { useState, useEffect } **from** 'react';

*/\*\**

*\* Hook return type interface*

*\*/*

**interface** UseHookReturn {

data: Type | **null**;

loading: boolean;

error: Error | **null**;

refetch: () => **void**;

}

*/\*\**

*\* Custom hook implementation*

*\* @param param - Hook parameter*

*\* @returns Hook return values*

*\*/*

**export** **const** useHook = (param: string): UseHookReturn => {

**const** [data, setData] = useState<Type | **null**>(**null**);

**const** [loading, setLoading] = useState(false);

**const** [error, setError] = useState<Error | **null**>(**null**);

**const** fetchData = **async** (): Promise<**void**> => {

*// Fetch logic*

};

useEffect(() => {

fetchData();

}, [param]);

**return** { data, loading, error, refetch: fetchData };

};

**13.5 Service Structure**

typescript

*/\*\**

*\* Service description*

*\* Handles API operations for {feature}*

*\*/*

**import** { BaseService } **from** '@services/base.service';

**import** { axiosInstance } **from** '@core/config/axios.config';

**import** { Type1, Type2 } **from** '../types';

*/\*\**

*\* Feature service class*

*\* Extends BaseService with feature-specific methods*

*\*/*

**class** FeatureService **extends** BaseService<Type1> {

constructor() {

**super**('/api/endpoint');

}

*/\*\**

*\* Custom service method*

*\* @param param - Method parameter*

*\* @returns Promise with result*

*\*/*

**async** customMethod(param: Type2): Promise<Type1> {

**const** response = **await** axiosInstance.post(

`${**this**.endpoint}/custom`,

param

);

**return** response.data.data;

}

}

**export** **const** featureService = **new** FeatureService();

**13.6 Comments and Documentation**

All files should include:

**File header comment**: Describes file purpose

**Function/Method JSDoc**: Describes parameters and return values

**Interface documentation**: Explains each property

**Inline comments**: Explains complex logic (when necessary)

**14. Adding New Features**

**14.1 Feature Development Checklist**

When adding a new feature, follow this comprehensive checklist:

**Phase 1: Planning**

 Define feature requirements

 Design data models and types

 Plan API endpoints needed

 Design component hierarchy

 Create wireframes/mockups

**Phase 2: Setup**

 Create feature folder structure

 Define TypeScript interfaces

 Add translation keys

 Create route constants

**Phase 3: Implementation**

 Create service layer

 Implement custom hooks

 Build UI components

 Add form validation

 Implement error handling

 Add loading states

**Phase 4: Testing**

 Test happy path

 Test error scenarios

 Test edge cases

 Test responsive design

 Test accessibility

**Phase 5: Documentation**

 Add code comments

 Update architecture docs

 Document API usage

 Add user documentation

**14.2 Step-by-Step Guide to Add New Feature**

Let's walk through adding a "Products" feature as an example:

**Step 1: Create Feature Folder Structure**

bash

src/features/products/

├── components/

│ ├── ProductCard/

│ │ ├── ProductCard.tsx

│ │ ├── ProductCard.types.ts

│ │ └── index.ts

│ └── ProductList/

├── hooks/

│ └── useProducts.ts

├── services/

│ └── product.service.ts

├── types/

│ └── product.types.ts

└── pages/

├── Products.tsx

├── ProductDetail.tsx

└── CreateProduct.tsx

**Step 2: Define Types**

**src/features/products/types/product.types.ts**

typescript

*/\*\**

*\* Product type definitions*

*\*/*

*/\*\**

*\* Product interface*

*\*/*

**export** **interface** Product {

id: string;

name: string;

description: string;

price: number;

category: string;

imageUrl: string;

createdAt: string;

updatedAt: string;

}

*/\*\**

*\* Create product request*

*\*/*

**export** **interface** CreateProductRequest {

name: string;

description: string;

price: number;

category: string;

imageUrl?: string;

}

*/\*\**

*\* Update product request*

*\*/*

**export** **interface** UpdateProductRequest **extends** Partial<CreateProductRequest> {}

**Step 3: Create Service**

**src/features/products/services/product.service.ts**

typescript

*/\*\**

*\* Product service*

*\* Handles all product-related API operations*

*\*/*

**import** { BaseService } **from** '@services/base.service';

**import** { axiosInstance } **from** '@core/config/axios.config';

**import** { Product, CreateProductRequest, UpdateProductRequest } **from** '../types/product.types';

**import** { ApiResponse } **from** '@core/types';

*/\*\**

*\* Product service class*

*\*/*

**class** ProductService **extends** BaseService<Product> {

constructor() {

**super**('/products');

}

*/\*\**

*\* Fetches products by category*

*\* @param category - Product category*

*\* @returns Promise with products*

*\*/*

**async** getByCategory(category: string): Promise<Product[]> {

**const** response = **await** axiosInstance.get<ApiResponse<Product[]>>(

`${**this**.endpoint}/category/${category}`

);

**return** response.data.data;

}

*/\*\**

*\* Searches products by query*

*\* @param query - Search query*

*\* @returns Promise with products*

*\*/*

**async** search(query: string): Promise<Product[]> {

**const** response = **await** axiosInstance.get<ApiResponse<Product[]>>(

`${**this**.endpoint}/search`,

{ params: { q: query } }

);

**return** response.data.data;

}

}

**export** **const** productService = **new** ProductService();

**Step 4: Create Custom Hook**

**src/features/products/hooks/useProducts.ts**

typescript

*/\*\**

*\* Products hook*

*\* Provides product-related operations and state*

*\*/*

**import** { useQuery, useMutation, useQueryClient } **from** '@tanstack/react-query';

**import** { productService } **from** '../services/product.service';

**import** { useNotification } **from** '@hooks/useNotification';

**import** { useTranslation } **from** 'react-i18next';

**import** { Product, CreateProductRequest } **from** '../types/product.types';

*/\*\**

*\* Hook for fetching all products*

*\*/*

**export** **const** useProducts = () => {

**return** useQuery({

queryKey: ['products'],

queryFn: () => productService.getAll(),

});

};

*/\*\**

*\* Hook for fetching single product*

*\* @param id - Product ID*

*\*/*

**export** **const** useProduct = (id: string) => {

**return** useQuery({

queryKey: ['products', id],

queryFn: () => productService.getById(id),

enabled: !!id,

});

};

*/\*\**

*\* Hook for creating product*

*\*/*

**export** **const** useCreateProduct = () => {

**const** queryClient = useQueryClient();

**const** { showSuccess, showError } = useNotification();

**const** { t } = useTranslation('messages');

**return** useMutation({

mutationFn: (data: CreateProductRequest) => productService.create(data),

onSuccess: () => {

queryClient.invalidateQueries({ queryKey: ['products'] });

showSuccess(t('product.createSuccess'));

},

onError: (error: Error) => {

showError(error.message || t('product.createError'));

},

});

};

*/\*\**

*\* Hook for deleting product*

*\*/*

**export** **const** useDeleteProduct = () => {

**const** queryClient = useQueryClient();

**const** { showSuccess, showError } = useNotification();

**const** { t } = useTranslation('messages');

**return** useMutation({

mutationFn: (id: string) => productService.delete(id),

onSuccess: () => {

queryClient.invalidateQueries({ queryKey: ['products'] });

showSuccess(t('product.deleteSuccess'));

},

onError: (error: Error) => {

showError(error.message || t('product.deleteError'));

},

});

};

**Step 5: Create Components**

**src/features/products/components/ProductCard/ProductCard.types.ts**

typescript

*/\*\**

*\* ProductCard component types*

*\*/*

**import** { Product } **from** '../../types/product.types';

**export** **interface** ProductCardProps {

product: Product;

onEdit?: (product: Product) => **void**;

onDelete?: (productId: string) => **void**;

}

**src/features/products/components/ProductCard/ProductCard.tsx**

typescript

*/\*\**

*\* ProductCard component*

*\* Displays product information in card format*

*\*/*

**import** React **from** 'react';

**import** {

Card,

CardContent,

CardMedia,

Typography,

CardActions,

IconButton,

} **from** '@mui/material';

**import** { Edit **as** EditIcon, Delete **as** DeleteIcon } **from** '@mui/icons-material';

**import** { ProductCardProps } **from** './ProductCard.types';

**import** { formatCurrency } **from** '@utils/helpers';

*/\*\**

*\* ProductCard component*

*\* @param props - Component props*

*\* @returns JSX Element*

*\*/*

**export** **const** ProductCard: React.FC<ProductCardProps> = ({

product,

onEdit,

onDelete,

}) => {

**return** (

<Card>

<CardMedia

component="img"

height="200"

image={product.imageUrl}

alt={product.name}

/>

<CardContent>

<Typography variant="h6" gutterBottom>

{product.name}

</Typography>

<Typography variant="body2" color="text.secondary" gutterBottom>

{product.description}

</Typography>

<Typography variant="h6" color="primary">

{formatCurrency(product.price)}

</Typography>

</CardContent>

<CardActions>

{onEdit && (

<IconButton onClick={() => onEdit(product)} size="small">

<EditIcon />

</IconButton>

)}

{onDelete && (

<IconButton onClick={() => onDelete(product.id)} size="small" color="error">

<DeleteIcon />

</IconButton>

)}

</CardActions>

</Card>

);

};

**export** **default** ProductCard;

**Step 6: Create Page Component**

**src/features/products/pages/Products.tsx**

typescript

*/\*\**

*\* Products page*

*\* Lists all products with CRUD operations*

*\*/*

**import** React, { useState } **from** 'react';

**import** {

Container,

Grid,

Typography,

Box,

CircularProgress,

Alert,

} **from** '@mui/material';

**import** { Button } **from** '@components/wrappers';

**import** { ProductCard } **from** '../components/ProductCard';

**import** { useProducts, useDeleteProduct } **from** '../hooks/useProducts';

**import** { useTranslation } **from** 'react-i18next';

**import** { Add **as** AddIcon } **from** '@mui/icons-material';

*/\*\**

*\* Products page component*

*\* @returns JSX Element*

*\*/*

**const** Products: React.FC = () => {

**const** { t } = useTranslation('common');

**const** { data, isLoading, error } = useProducts();

**const** deleteProduct = useDeleteProduct();

**const** handleDelete = (productId: string): **void** => {

**if** (window.confirm(t('product.confirmDelete'))) {

deleteProduct.mutate(productId);

}

};

**if** (isLoading) {

**return** (

<Container>

<Box display="flex" justifyContent="center" alignItems="center" minHeight="50vh">

<CircularProgress />

</Box>

</Container>

);

}

**if** (error) {

**return** (

<Container>

<Alert severity="error">{error.message}</Alert>

</Container>

);

}

**return** (

<Container maxWidth="lg" sx={{ mt: 4, mb: 4 }}>

<Box sx={{ display: 'flex', justifyContent: 'space-between', mb: 3 }}>

<Typography variant="h4">Products</Typography>

<Button variant="primary" startIcon={<AddIcon />}>

Add Product

</Button>

</Box>

<Grid container spacing={3}>

{data?.items.map((product) => (

<Grid item xs={12} sm={6} md={4} key={product.id}>

<ProductCard

product={product}

onDelete={handleDelete}

/>

</Grid>

))}

</Grid>

</Container>

);

};

**export** **default** Products;

**Step 7: Add Routes**

**Update src/routes/AppRoutes.tsx**

typescript

*// Add import*

**const** Products = lazy(() => **import**('@features/products/pages/Products'));

**const** ProductDetail = lazy(() => **import**('@features/products/pages/ProductDetail'));

*// Add routes inside ProtectedRoute*

<Route path="/products" element={<Products />} />

<Route path="/products/:id" element={<ProductDetail />} />

**Step 8: Add Route Constants**

**Update src/core/constants/routes.constants.ts**

typescript

**export** **const** ROUTES = {

*// ... existing routes*

PRODUCTS: '/products',

PRODUCT\_DETAIL: '/products/:id',

} **as** **const**;

**Step 9: Add Translations**

**Update src/core/i18n/locales/en/common.json**

json

{

"navigation": {

"products": "Products"

}

}

**Update src/core/i18n/locales/en/messages.json**

json

{

"product": {

"createSuccess": "Product created successfully",

"createError": "Failed to create product",

"deleteSuccess": "Product deleted successfully",

"deleteError": "Failed to delete product",

"confirmDelete": "Are you sure you want to delete this product?"

}

}

**Step 10: Add Navigation Link**

**Update MainLayout to add Products link**

typescript

<MenuItem onClick={() => { handleCloseUserMenu(); navigate('/products'); }}>

<ListItemIcon>

<ShoppingCartIcon fontSize="small" />

</ListItemIcon>

{t('navigation.products')}

</MenuItem>

**14.3 Feature Development Best Practices**

**Start with Types** - Define interfaces before implementation

**Service First** - Create service layer before hooks

**Test API Calls** - Test service methods with mock data first

**Build Components Bottom-Up** - Start with small components, compose larger ones

**Handle Loading States** - Always show loading indicators

**Handle Errors** - Display user-friendly error messages

**Add Validation** - Validate user input on forms

**Make it Responsive** - Test on different screen sizes

**Add Translations** - Support all configured languages

**Document Code** - Add comments and JSDoc

**15. Testing Strategy**

**15.1 Testing Pyramid**

text

/\

/ \

/ E2E \ End-to-End Tests (Few)

/\_\_\_\_\_\_\

/ \

/Integration\ Integration Tests (Some)

/\_\_\_\_\_\_\_\_\_\_\_\_\

/ \

/ Unit Tests \ Unit Tests (Many)

/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\

**15.2 Testing Tools (To Be Implemented)**

| **Type** | **Tool** | **Purpose** |
| --- | --- | --- |
| **Unit Testing** | Vitest | Test individual functions and components |
| **Component Testing** | React Testing Library | Test component behavior |
| **E2E Testing** | Playwright | Test user workflows |
| **Visual Testing** | Storybook | Component documentation and visual testing |
| **Code Coverage** | Vitest Coverage | Track test coverage |

**15.3 What to Test**

**15.3.1 Unit Tests**

Utility functions (helpers.ts, validators.ts)

Custom hooks

Service methods

State management (Zustand stores)

**15.3.2 Component Tests**

Component rendering

User interactions (clicks, inputs)

Props handling

Conditional rendering

Error states

Loading states

**15.3.3 Integration Tests**

Form submissions

API calls with mocked responses

Route navigation

Authentication flows

State updates

**15.3.4 E2E Tests**

Complete user workflows

Login → Dashboard → Feature interaction

Critical business paths

**15.4 Test File Organization**

text

src/

├── features/

│ └── auth/

│ ├── hooks/

│ │ ├── useAuth.ts

│ │ └── useAuth.test.ts # Hook tests

│ ├── services/

│ │ ├── auth.service.ts

│ │ └── auth.service.test.ts # Service tests

│ └── pages/

│ ├── Login.tsx

│ └── Login.test.tsx # Component tests

└── utils/

├── helpers.ts

└── helpers.test.ts # Utility tests

**15.5 Example Test Cases**

**Unit Test Example**

typescript

*// helpers.test.ts*

**import** { describe, it, expect } **from** 'vitest';

**import** { formatCurrency, getInitials } **from** './helpers';

describe('formatCurrency', () => {

it('should format USD currency correctly', () => {

expect(formatCurrency(1234.56)).toBe('$1,234.56');

});

it('should handle zero', () => {

expect(formatCurrency(0)).toBe('$0.00');

});

});

describe('getInitials', () => {

it('should return initials from full name', () => {

expect(getInitials('John Doe')).toBe('JD');

});

it('should handle single name', () => {

expect(getInitials('John')).toBe('J');

});

});

**Component Test Example**

typescript

*// Login.test.tsx*

**import** { describe, it, expect, vi } **from** 'vitest';

**import** { render, screen, fireEvent, waitFor } **from** '@testing-library/react';

**import** { Login } **from** './Login';

describe('Login Component', () => {

it('should render login form', () => {

render(<Login />);

expect(screen.getByLabelText(/email/i)).toBeInTheDocument();

expect(screen.getByLabelText(/password/i)).toBeInTheDocument();

expect(screen.getByRole('button', { name: /submit/i })).toBeInTheDocument();

});

it('should show validation errors for invalid input', **async** () => {

render(<Login />);

**const** submitButton = screen.getByRole('button', { name: /submit/i });

fireEvent.click(submitButton);

**await** waitFor(() => {

expect(screen.getByText(/email is required/i)).toBeInTheDocument();

expect(screen.getByText(/password is required/i)).toBeInTheDocument();

});

});

it('should call login function on valid submission', **async** () => {

**const** mockLogin = vi.fn();

render(<Login onLogin={mockLogin} />);

fireEvent.change(screen.getByLabelText(/email/i), {

target: { value: 'test@example.com' },

});

fireEvent.change(screen.getByLabelText(/password/i), {

target: { value: 'Password123!' },

});

**const** submitButton = screen.getByRole('button', { name: /submit/i });

fireEvent.click(submitButton);

**await** waitFor(() => {

expect(mockLogin).toHaveBeenCalledWith({

email: 'test@example.com',

password: 'Password123!',

});

});

});

});

**16. Performance Optimization**

**16.1 Code Splitting Strategy**

The application uses React lazy loading for route-based code splitting:

typescript

*// Lazy load pages*

**const** Dashboard = lazy(() => **import**('@features/dashboard/pages/Dashboard'));

**const** Profile = lazy(() => **import**('@features/profile/pages/Profile'));

*// Wrapped in Suspense*

<Suspense fallback={<LoadingFallback />}>

<Routes>

<Route path="/dashboard" element={<Dashboard />} />

<Route path="/profile" element={<Profile />} />

</Routes>

</Suspense>

**Benefits**:

Smaller initial bundle size

Faster initial load time

Load components only when needed

**16.2 Build Optimization**

**Vite Configuration** (vite.config.ts):

typescript

**export** **default** defineConfig({

build: {

rollupOptions: {

output: {

manualChunks: {

vendor: ['react', 'react-dom', 'react-router-dom'],

mui: ['@mui/material', '@mui/icons-material'],

query: ['@tanstack/react-query'],

},

},

},

},

});

**Result**:

Separate chunks for vendor libraries

Better caching (vendor code rarely changes)

Parallel loading of chunks

**16.3 React Query Caching**

React Query automatically caches server data:

typescript

*// Configuration*

**const** queryConfig = {

queries: {

staleTime: 5 \* 60 \* 1000, *// 5 minutes*

gcTime: 10 \* 60 \* 1000, *// 10 minutes*

refetchOnWindowFocus: false,

},

};

**Benefits**:

Reduced API calls

Instant data display from cache

Background refetching

Automatic cache invalidation

**16.4 Memoization**

Use React memoization hooks appropriately:

typescript

*// useMemo for expensive calculations*

**const** expensiveValue = useMemo(() => {

**return** computeExpensiveValue(data);

}, [data]);

*// useCallback for event handlers passed to children*

**const** handleClick = useCallback(() => {

*// Handler logic*

}, [dependencies]);

*// React.memo for pure components*

**export** **const** PureComponent = React.memo<Props>(({ data }) => {

**return** <div>{data}</div>;

});

**16.5 Image Optimization**

**Use appropriate formats**: WebP for modern browsers

**Lazy load images**: Use loading="lazy" attribute

**Provide multiple sizes**: Use srcset for responsive images

**Compress images**: Use tools like ImageOptim

**Use CDN**: Serve images from CDN for faster delivery

**16.6 Performance Monitoring**

Monitor performance metrics:

**First Contentful Paint (FCP)**

**Largest Contentful Paint (LCP)**

**Time to Interactive (TTI)**

**Cumulative Layout Shift (CLS)**

Use tools:

Chrome DevTools Performance tab

Lighthouse

Web Vitals extension

**17. Security Considerations**

**17.1 Authentication & Authorization**

**17.1.1 Token Management**

Store tokens in localStorage (acceptable for web apps)

Include tokens in Authorization header

Implement token refresh mechanism

Clear tokens on logout

**17.1.2 Protected Routes**

Check authentication before rendering protected pages

Redirect unauthorized users to login

Implement role-based access control (RBAC)

**17.2 API Security**

**17.2.1 HTTPS Only**

Always use HTTPS in production

Set VITE\_API\_BASE\_URL to HTTPS endpoint

**17.2.2 Request Validation**

Validate all user input

Sanitize data before sending to API

Use Yup schemas for form validation

**17.2.3 Error Handling**

Don't expose sensitive error details

Log errors securely on server

Show user-friendly messages only

**17.3 XSS Prevention**

**React's Built-in Protection**: React escapes values by default

**Avoid dangerouslySetInnerHTML**: Only use when absolutely necessary

**Sanitize HTML**: If you must render HTML, sanitize it first

**Content Security Policy**: Implement CSP headers

**17.4 CSRF Protection**

**Use CORS**: Configure proper CORS policy

**SameSite Cookies**: Use SameSite attribute for cookies

**CSRF Tokens**: Include CSRF tokens in requests

**17.5 Dependency Security**

**Regular Updates**: Keep dependencies up to date

**Audit**: Run npm audit regularly

**Avoid Suspicious Packages**: Verify package authenticity

**Lock Dependencies**: Use package-lock.json

**17.6 Environment Variables**

**Never Commit Secrets**: Use .env files (added to .gitignore)

**Use Environment-Specific Files**: .env.development, .env.production

**Vite Prefix**: Only variables prefixed with VITE\_ are exposed to client

text

# .env.example

VITE\_API\_BASE\_URL=https://api.example.com

VITE\_APP\_NAME=My Application

# Never commit actual .env with real values!

**17.7 Security Checklist**

 Use HTTPS in production

 Implement proper authentication

 Validate and sanitize all inputs

 Implement CSRF protection

 Use Content Security Policy

 Keep dependencies updated

 Don't expose sensitive data in client

 Implement rate limiting on API

 Use secure headers

 Regular security audits

**18. Deployment Guide**

**18.1 Build Process**

**18.1.1 Development Build**

bash

*# Install dependencies*

npm install

*# Start development server*

npm run dev

*# Server runs at http://localhost:3000*

**18.1.2 Production Build**

bash

*# Type check*

npm run type-check

*# Lint code*

npm run lint

*# Build for production*

npm run build

*# Output: dist/ folder*

**18.2 Environment Configuration**

**Development**

text

# .env.development

VITE\_API\_BASE\_URL=http://localhost:5000/api

VITE\_APP\_NAME=Enterprise App (Dev)

**Production**

text

# .env.production

VITE\_API\_BASE\_URL=https://api.production.com/api

VITE\_APP\_NAME=Enterprise Application

**18.3 Deployment Platforms**

**18.3.1 Vercel**

bash

*# Install Vercel CLI*

npm install -g vercel

*# Deploy*

vercel

*# Production deployment*

vercel --prod

**Configuration** (vercel.json):

json

{

"buildCommand": "npm run build",

"outputDirectory": "dist",

"framework": "vite",

"rewrites": [

{ "source": "/(.\*)", "destination": "/index.html" }

]

}

**18.3.2 Netlify**

bash

*# Install Netlify CLI*

npm install -g netlify-cli

*# Deploy*

netlify deploy

*# Production deployment*

netlify deploy --prod

**Configuration** (netlify.toml):

text

[build]

command = "npm run build"

publish = "dist"

[[redirects]]

from = "/\*"

to = "/index.html"

status = 200

**18.3.3 AWS S3 + CloudFront**

Build application: npm run build

Upload dist/ to S3 bucket

Enable static website hosting

Configure CloudFront distribution

Set up Route 53 for domain

**18.3.4 Docker**

**Dockerfile**:

text

# Build stage

FROM node:20-alpine AS build

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

RUN npm run build

# Production stage

FROM nginx:alpine

COPY --from=build /app/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

**nginx.conf**:

text

server {

listen 80;

server\_name localhost;

root /usr/share/nginx/html;

index index.html;

location / {

try\_files $uri $uri/ /index.html;

}

# Cache static assets

location ~\* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {

expires 1y;

add\_header Cache-Control "public, immutable";

}

}

**Build and Run**:

bash

*# Build image*

docker build -t enterprise-app .

*# Run container*

docker run -p 3000:80 enterprise-app

**18.4 CI/CD Pipeline**

**GitHub Actions Example**

**.github/workflows/deploy.yml**:

text

name: Deploy to Production

on:

push:

branches: [main]

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: '20'

cache: 'npm'

- name: Install dependencies

run: npm ci

- name: Type check

run: npm run type-check

- name: Lint

run: npm run lint

- name: Build

run: npm run build

env:

VITE\_API\_BASE\_URL: ${{ secrets.API\_BASE\_URL }}

- name: Deploy to Vercel

uses: amondnet/vercel-action@v20

with:

vercel-token: ${{ secrets.VERCEL\_TOKEN }}

vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}

vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID }}

vercel-args: '--prod'

**18.5 Post-Deployment Checklist**

 Verify environment variables are set correctly

 Test all critical user flows

 Check API endpoints are accessible

 Verify authentication works

 Test on multiple browsers

 Check mobile responsiveness

 Monitor error logs

 Test language switching

 Verify theme switching

 Check performance metrics

 Enable monitoring/analytics

**19. Maintenance & Monitoring**

**19.1 Regular Maintenance Tasks**

**Weekly**

 Check and update dependencies

 Review error logs

 Monitor performance metrics

 Check security advisories

**Monthly**

 Run npm audit and fix vulnerabilities

 Review and update documentation

 Analyze bundle size

 Review and optimize queries

**Quarterly**

 Major dependency updates

 Code quality review

 Performance audit

 Security audit

**19.2 Monitoring Tools (Recommended)**

| **Tool** | **Purpose** |
| --- | --- |
| **Sentry** | Error tracking and monitoring |
| **LogRocket** | Session replay and debugging |
| **Google Analytics** | User analytics |
| **Lighthouse CI** | Performance monitoring |
| **Datadog** | Application performance monitoring |

**19.3 Key Metrics to Monitor**

**Performance**: Page load time, FCP, LCP, TTI

**Errors**: JavaScript errors, API failures

**Usage**: Active users, page views, user flows

**Business**: Conversion rates, feature adoption

**20. Troubleshooting Guide**

**20.1 Common Issues**

**Build Errors**

**Problem**: TypeScript compilation errors

text

Solution:

1. Run: npm run type-check

2. Fix type errors in reported files

3. Clear cache: rm -rf node\_modules/.vite

4. Rebuild: npm run build

**Problem**: Module not found errors

text

Solution:

1. Check path aliases in tsconfig.json

2. Verify import paths

3. Clear node\_modules: rm -rf node\_modules

4. Reinstall: npm install

**Runtime Errors**

**Problem**: "Cannot read property of undefined"

text

Solution:

1. Add optional chaining: user?.name

2. Add null checks

3. Use default values

4. Check ErrorBoundary logs

**Problem**: API calls failing

text

Solution:

1. Check network tab in DevTools

2. Verify API endpoint URL

3. Check CORS configuration

4. Verify authentication token

5. Check axios interceptors

**Styling Issues**

**Problem**: Theme not applying

text

Solution:

1. Check ThemeProvider is wrapping app

2. Verify theme.config.ts

3. Clear localStorage

4. Check browser cache

**Problem**: Dark mode not working

text

Solution:

1. Check ThemeContext implementation

2. Verify useLocalStorage hook

3. Check theme mode persistence

4. Clear localStorage: localStorage.clear()

**20.2 Debug Mode**

Enable detailed logging:

typescript

*// In development, enable query devtools*

**if** (**import**.meta.env.DEV) {

console.log('Development mode enabled');

}

*// Enable React Query devtools*

<ReactQueryDevtools initialIsOpen={false} />

**20.3 Performance Issues**

**Problem**: Slow page load

text

Solution:

1. Check Network tab for slow requests

2. Analyze bundle size

3. Implement code splitting

4. Optimize images

5. Enable caching

**Problem**: Unnecessary re-renders

text

Solution:

1. Use React DevTools Profiler

2. Add React.memo where appropriate

3. Use useMemo for expensive calculations

4. Use useCallback for event handlers

**21. Glossary**

| **Term** | **Definition** |
| --- | --- |
| **API** | Application Programming Interface - Backend service endpoints |
| **Axios** | Promise-based HTTP client for making API requests |
| **Barrel Export** | index.ts file that re-exports multiple modules |
| **Code Splitting** | Breaking code into smaller bundles loaded on demand |
| **Component** | Reusable piece of UI in React |
| **Context** | React's way to share data across component tree |
| **CRUD** | Create, Read, Update, Delete operations |
| **Hook** | React function that lets you use state and lifecycle features |
| **HOC** | Higher-Order Component - function that takes a component and returns a new component |
| **i18n** | Internationalization - supporting multiple languages |
| **Lazy Loading** | Loading components/data only when needed |
| **Material-UI** | React component library implementing Material Design |
| **Memoization** | Caching technique to optimize performance |
| **Mutation** | React Query term for creating/updating/deleting data |
| **Query** | React Query term for fetching data |
| **React Query** | Library for managing server state |
| **Route** | URL path that maps to a component |
| **Service Layer** | Layer that handles API communication |
| **State** | Data that changes over time in a component |
| **Store** | Centralized place to manage application state |
| **TypeScript** | Typed superset of JavaScript |
| **Vite** | Modern build tool for web development |
| **Yup** | Schema validation library |
| **Zustand** | Lightweight state management library |

**22. Appendix**

**22.1 Useful Commands**

bash

*# Development*

npm run dev *# Start development server*

npm run type-check *# Check TypeScript types*

npm run lint *# Run ESLint*

npm run lint --fix *# Auto-fix linting issues*

npm run format *# Format code with Prettier*

*# Building*

npm run build *# Build for production*

npm run preview *# Preview production build*

*# Maintenance*

npm audit *# Check for vulnerabilities*

npm audit fix *# Fix vulnerabilities*

npm outdated *# Check for outdated packages*

npm update *# Update packages*

**22.2 Project Structure Diagram**

text

┌─────────────────────────────────────────────────────────────┐

│ Enterprise React App │

└─────────────────────────────────────────────────────────────┘

│

┌─────────────────────┼─────────────────────┐

│ │ │

┌───▼────┐ ┌────▼────┐ ┌────▼────┐

│ Core │ │Features │ │Services │

└───┬────┘ └────┬────┘ └────┬────┘

│ │ │

┌─────┼──────┐ ┌─────┼──────┐ ┌─────┼──────┐

│ │ │ │ │ │ │ │ │

Config Types i18n Auth Dashboard │ Base API │

Profile Service Instance │

Settings Interceptors

**22.3 Key Files Reference**

| **File** | **Purpose** | **Location** |
| --- | --- | --- |
| **App.tsx** | Root component | src/App.tsx |
| **main.tsx** | Entry point | src/main.tsx |
| **AppRoutes.tsx** | Route configuration | src/routes/AppRoutes.tsx |
| **axios.config.ts** | Axios setup | src/core/config/axios.config.ts |
| **theme.config.ts** | Theme configuration | src/core/config/theme.config.ts |
| **i18n.config.ts** | i18n setup | src/core/i18n/i18n.config.ts |
| **auth.store.ts** | Auth state | src/store/auth.store.ts |
| **base.service.ts** | Base API service | src/services/base.service.ts |

**22.4 External Resources**

React Documentation

TypeScript Handbook

Material-UI Documentation

React Query Documentation

Vite Guide

Formik Documentation

i18next Documentation

**Document Revision History**

| **Version** | **Date** | **Author** | **Changes** |
| --- | --- | --- | --- |
| 1.0.0 | Oct 23, 2025 | Architecture Team | Initial document creation |