React Component Development Instructions

Comprehensive Guide for Creating New Components & Pages

📋 Table of Contents

🎯 Component Classification & Placement

When to Create in `/src/components/`

- **Feature-specific components** (e.g., PaymentForm, UserProfile, EmailVerification)
- **Complex UI elements** with specific business logic
- **Reusable functional components** that serve particular features
- **Form components** for specific operations

Examples:

- `ForgotPassword.jsx` - Password recovery functionality
- `VerifyEmail.jsx` - Email verification process
- `SubscriptionPayment.jsx` - Payment processing
- `PaymentConfirmation.jsx` - Payment result display

When to Create in `/src/pages/`

- **Route-level components** that represent full pages
- **Main application views** users navigate to
- **Dashboard pages** or primary app sections
- **Landing pages** or standalone views

Examples:

- `Home.jsx` - Main landing page
- `Profile.jsx` - User profile page
- `Dashboard.jsx` - Main app dashboard
- `Settings.jsx` - Application settings

When to Create in `/src/commons/`

- **Reusable UI components** used across multiple pages
- **Basic building blocks** (buttons, cards, inputs)
- **Navigation components** (navbar, footer, sidebar)
- **Generic form elements** without specific business logic

Examples:

- `Button.jsx` - Reusable button component
- `Card.jsx` - Generic card wrapper
- `Navbar.jsx` - Global navigation
- `Footer.jsx` - Page footer

📂 File Organization Rules

1. Folder Structure Compliance

```
src/
├── components/     # Feature-specific components
├── pages/         # Route-level page components
├── commons/       # Reusable UI building blocks
├── layouts/       # Layout wrappers and containers
├── utils/         # Utility functions and helpers
├── store/         # Redux state management
└── session/       # Session management utilities
```

2. File Naming Conventions

- **PascalCase** for component files: `UserProfile.jsx`
- **Descriptive names** that clearly indicate purpose
- **Consistent extensions**: Use `.jsx` for React components
- **Index files**: Avoid using `index.js` unless for barrel exports

3. Component Structure

```javascript
// Standard component structure
import React from 'react'
// Other imports...

function ComponentName() {
  // Component Logic
  return (
    // JSX
  )
}

export default ComponentName
```

🔧 Required Imports & Dependencies

Essential Imports for Every Component

1. React Core

```javascript
import React, { useState, useEffect } from 'react'
```

2. Routing (for page components)

```javascript
import { useNavigate, useLocation, useParams } from 'react-router-dom'
```

3. State Management (when needed)

```javascript
import { useDispatch, useSelector } from 'react-redux'
import { login, logout, updateUserData } from '../store/authSlice'
```

4. HTTP Requests

```javascript
import axios from 'axios'
```

5. MANDATORY: Error Handling

```javascript
import { safeApiCall, handleApiError } from '../utils/errorHandler'
```

6. Notifications

```javascript
import toast from 'react-hot-toast'
```

7. Layout Components

```javascript
import Container from '../layouts/Container'
```

🛡️ Error Handling Implementation

ALWAYS Use safeApiCall for API Requests

Basic Usage:

```javascript
// ✅ CORRECT - Using safeApiCall
const fetchData = async () => {
  try {
    const response = await safeApiCall(
      () => axios.get(`${import.meta.env.VITE_BACKEND_DOMAIN}/api/endpoint`),
      // Optional custom error handler
      async (error) => {
        if (error.response?.status === 404) {
          toast.error('Data not found');
          return true; // Mark as handled
        }
        return false; // Let default handler manage other errors
      }
    );

    // Handle successful response
    setData(response.data);
  } catch (error) {
    // Error already handled by safeApiCall
    console.error('Operation failed:', error);
  }
};
```

For Authentication Requests:

```javascript
// ✅ CORRECT - Auth-specific error handling
const handleLogin = async (credentials) => {
  try {
    const response = await safeApiCall(
      () => axios.post(
        `${import.meta.env.VITE_BACKEND_DOMAIN}/api/v1/auth/login`,
        credentials,
        { withCredentials: true }
      ),
      async (error) => {
        if (error.response?.status === 401) {
          toast.error('Invalid credentials');
          return true;
        }
        return false;
      }
    );

    // Handle successful login
    dispatch(login(response.data.user));
    navigate('/dashboard');
  } catch (error) {
    // Already handled by safeApiCall
  }
};
```

NEVER Do Direct Axios Calls

```javascript
// ❌ WRONG - Direct axios without error handling
const badExample = async () => {
  const response = await axios.get('/api/data'); // Don't do this!
};
```

🎨 Layout & Styling Guidelines

1. ALWAYS Use Container for Layout

```javascript
import Container from '../layouts/Container'

function MyComponent() {
  return (
    <Container className="py-8">
      <div className="max-w-md mx-auto">
        {/* Your content */}
      </div>
    </Container>
  )
}
```

2. Responsive Design with Tailwind

```javascript
// ✅ Mobile-first responsive design
<div className="w-full max-w-md mx-auto p-4 sm:p-6 lg:p-8">
  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
    {/* Content */}
  </div>
</div>
```

3. Consistent Spacing & Colors

```javascript
// Use consistent Tailwind classes
<div className="space-y-6"> {/* Consistent vertical spacing */}
  <h1 className="text-2xl font-bold text-gray-900">Title</h1>
  <div className="bg-white rounded-lg shadow-sm border p-6">
    {/* Content */}
  </div>
</div>
```

🚪 Routing Configuration

1. Check Existing Routes in main.jsx

Before adding new routes, always review current routing structure:

```javascript
// Current routes in main.jsx
const routes = [
  { path: "/", element: <Home /> },
  { path: "/login", element: <LogIn /> },
  { path: "/signup", element: <SignUp /> },
  { path: "/profile", element: <Profile /> },
  { path: "/forgot-password", element: <ForgotPassword /> },
  { path: "/verify-email", element: <VerifyEmail /> },
  { path: "/subscription-payment", element: <SubscriptionPayment /> },
  { path: "/payment-confirmation", element: <PaymentConfirmation /> }
];
```

2. Adding New Page Routes

When creating a new page component, add corresponding route:

```javascript
// 1. Import the new component
import NewPage from './pages/NewPage.jsx'

// 2. Add route to the router configuration
{
  path: "/new-page",
  element: (
    <AuthLayout authentication={true}> {/* or false for public pages */}
      <NewPage />
    </AuthLayout>
  ),
}
```

3. Route Protection Patterns

```javascript
// Protected route (requires authentication)
{
  path: "/dashboard",
  element: (
    <AuthLayout authentication={true}>
      <Dashboard />
    </AuthLayout>
  ),
}

// Public route (no authentication required)
{
  path: "/about",
  element: (
    <AuthLayout authentication={false}>
      <About />
    </AuthLayout>
  ),
}

// Special routes (bypass normal auth flow)
{
  path: "/verify-email",
  element: (
    <AuthLayout authentication={false}>
      <VerifyEmail />
    </AuthLayout>
  ),
}
```

🔐 Authentication & Protection

1. ALWAYS Use AuthLayout for Pages

```javascript
// ✅ CORRECT - Wrapped with AuthLayout
function MyPage() {
  return (
    <Container>
      <h1>My Page Content</h1>
    </Container>
  )
}

// In main.jsx route configuration:
{
  path: "/my-page",
  element: (
    <AuthLayout authentication={true}>
      <MyPage />
    </AuthLayout>
  ),
}
```

2. Access Authentication State

```javascript
import { useSelector } from 'react-redux'

function MyComponent() {
  const authStatus = useSelector(state => state.auth.status)
  const userData = useSelector(state => state.auth.userData)

  if (!authStatus) {
    return <div>Please log in</div>
  }

  return <div>Welcome, {userData?.name}</div>
}
```

3. Handle Authentication Actions

```javascript
import { useDispatch } from 'react-redux'
import { logout } from '../store/authSlice'

function MyComponent() {
  const dispatch = useDispatch()

  const handleLogout = () => {
    dispatch(logout())
    navigate('/login')
  }

  return (
    <button onClick={handleLogout}>
      Logout
    </button>
  )
}
```

📝 Code Examples & Templates

Page Component Template

javascript

```
import React, { useState, useEffect } from 'react'
import { useNavigate, useParams } from 'react-router-dom'
import { useDispatch, useSelector } from 'react-redux'
import axios from 'axios'
import { safeApiCall } from '../utils/errorHandler'
import { updateUserData } from '../store/authSlice'
import Container from '../layouts/Container'
import toast from 'react-hot-toast'

function NewPage() {
  const [loading, setLoading] = useState(false)
  const [data, setData] = useState(null)
  const navigate = useNavigate()
  const dispatch = useDispatch()
  const userData = useSelector(state => state.auth.userData)

  useEffect(() => {
    fetchData()
  }, [])

  const fetchData = async () => {
    setLoading(true)
    try {
      const response = await safeApiCall(
        () => axios.get(`${import.meta.env.VITE_BACKEND_DOMAIN}/api/endpoint`)
      )
      setData(response.data)
    } catch (error) {
      // Error handled by safeApiCall
    } finally {
      setLoading(false)
    }
  }

  const handleSubmit = async (formData) => {
    try {
      const response = await safeApiCall(
        () => axios.post(`${import.meta.env.VITE_BACKEND_DOMAIN}/api/submit`, formData)
      )

      toast.success('Success!')
      navigate('/success-page')
    } catch (error) {
      // Error handled by safeApiCall
    }
  }
```

```
  if (loading) {
    return (
      <Container className="py-8">
        <div className="flex justify-center">
          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-primary-500"></
        </div>
      </Container>
    )
  }

  return (
    <Container className="py-8">
      <div className="max-w-2xl mx-auto">
        <h1 className="text-3xl font-bold text-gray-900 mb-6">Page Title</h1>

        <div className="bg-white rounded-lg shadow-sm border p-6">
          {/* Page content */}
        </div>
      </div>
    </Container>
  )
}

export default NewPage
```

Feature Component Template

javascript

```
import React, { useState } from 'react'
import { safeApiCall } from '../utils/errorHandler'
import axios from 'axios'
import toast from 'react-hot-toast'

function NewFeatureComponent({ onSuccess, onCancel }) {
  const [loading, setLoading] = useState(false)
  const [formData, setFormData] = useState({})

  const handleSubmit = async (e) => {
    e.preventDefault()
    setLoading(true)

    try {
      const response = await safeApiCall(
        () => axios.post(`${import.meta.env.VITE_BACKEND_DOMAIN}/api/feature`, formData)
      )

      toast.success('Operation completed successfully!')
      onSuccess?.(response.data)
    } catch (error) {
      // Error handled by safeApiCall
    } finally {
      setLoading(false)
    }
  }

  return (
    <div className="space-y-6">
      <h2 className="text-xl font-semibold text-gray-900">Feature Title</h2>

      <form onSubmit={handleSubmit} className="space-y-4">
        {/* Form fields */}

        <div className="flex justify-end space-x-3">
          <button
            type="button"
            onClick={onCancel}
            className="px-4 py-2 text-gray-700 bg-gray-200 rounded-lg hover:bg-gray-300"
          >
            Cancel
          </button>
          <button
            type="submit"
            disabled={loading}
            className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 disabled:c
```

```
          >
            {loading ? 'Processing...' : 'Submit'}
          </button>
        </div>
      </form>
    </div>
  )
}

export default NewFeatureComponent
```

✅ Development Checklist

Before Creating a New Component:

📁 File Organization

- [ ] Determined correct folder placement (`/components/`, `/pages/`, or `/commons/`)
- [ ] Used PascalCase naming convention
- [ ] Created `.jsx` file extension

📥 Imports & Dependencies

- [ ] Imported React core functionality
- [ ] Added routing hooks if needed (`useNavigate`, `useLocation`)
- [ ] Included Redux hooks if state management needed
- [ ] **MANDATORY:** Imported `safeApiCall` from `../utils/errorHandler`
- [ ] Imported `Container` from `../layouts/Container` for layout
- [ ] Added `toast` for user notifications

🛡️ Error Handling

- [ ] Used `safeApiCall` for ALL API requests
- [ ] Added custom error handlers where needed
- [ ] NO direct axios calls without error wrapper
- [ ] Proper try-catch blocks around async operations

🎨 Layout & Styling

- [ ] Wrapped content with `Container` component
- [ ] Used responsive Tailwind classes
- [ ] Followed mobile-first design approach
- [ ] Consistent spacing and color usage

🚪 Routing (for Pages)

- [ ] Checked existing routes in `main.jsx`
- [ ] Added new route configuration
- [ ] Wrapped with `AuthLayout` with correct authentication setting
- [ ] Updated navigation links if needed

🔐 Authentication

- [ ] Used `AuthLayout` wrapper for pages
- [ ] Accessed auth state via Redux selectors
- [ ] Handled authentication actions properly
- [ ] Considered protected vs public route requirements

🧪 Testing & Validation

- [ ] Component renders without errors
- [ ] API calls work with proper error handling
- [ ] Responsive design tested on different screen sizes
- [ ] Navigation flows work correctly
- [ ] Authentication protection works as expected

📚 Documentation

- [ ] Added component to appropriate documentation section
- [ ] Updated routing documentation if new routes added
- [ ] Noted any special requirements or dependencies

Following these instructions ensures consistent, maintainable, and robust component development across the React application.