# 🚀 **CYPHER Dashboard Development Guide**

## **Table of Contents**

1. [Architecture Overview](#architecture-overview)
2. [Tech Stack](#tech-stack)
3. [Design Patterns](#design-patterns)
4. [RBAC Security Implementation](#rbac-security-implementation)
5. [Database Design Standards](#database-design-standards)
6. [API Development Guidelines](#api-development-guidelines)
7. [Frontend Development Standards](#frontend-development-standards)
8. [Code Organization](#code-organization)
9. [Testing Standards](#testing-standards)
10. [Deployment Guidelines](#deployment-guidelines)

## 🏗️ **Architecture Overview**

### **System Architecture Pattern: Layered Architecture with MVC**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Frontend │ │ Backend API │ │ Database │  
│ (React SPA) │◄──►│ (Express.js) │◄──►│ (PostgreSQL) │  
└─────────────────┘ └─────────────────┘ └─────────────────┘

### **Request Flow:**

Client → Nginx → API Gateway → Auth Middleware → RBAC → Controller → Service → Database

### **Core Principles:**

* **Separation of Concerns**: Clear boundaries between layers
* **Single Responsibility**: Each component has one job
* **Dependency Injection**: Services injected into controllers
* **Interface Segregation**: Small, focused interfaces
* **Open/Closed Principle**: Open for extension, closed for modification

## 🛠️ **Tech Stack**

### **Backend Stack**

| Component | Technology | Version | Purpose |
| --- | --- | --- | --- |
| **Runtime** | Node.js | 20.x | JavaScript runtime |
| **Framework** | Express.js | 4.x | Web application framework |
| **Database** | PostgreSQL | 13+ | Primary data store |
| **ORM** | Drizzle ORM | Latest | Database abstraction |
| **Authentication** | JWT | Latest | Token-based auth |
| **Validation** | Joi | Latest | Request validation |
| **Documentation** | Swagger | 3.x | API documentation |
| **Testing** | Jest | Latest | Unit/Integration testing |

### **Frontend Stack**

| Component | Technology | Version | Purpose |
| --- | --- | --- | --- |
| **Framework** | React | 18.x | UI framework |
| **Routing** | React Router | 6.x | Client-side routing |
| **UI Library** | Reactstrap | Latest | Bootstrap components |
| **State Management** | React Hooks | Built-in | Local state management |
| **HTTP Client** | Fetch API | Native | API communication |
| **Styling** | SCSS | Latest | Styling preprocessor |
| **Build Tool** | Vite | Latest | Build and dev server |

### **Infrastructure**

| Component | Technology | Purpose |
| --- | --- | --- |
| **Container** | Docker | Application containerization |
| **Orchestration** | Docker Compose | Multi-container management |
| **Web Server** | Nginx | Reverse proxy & static files |
| **Database** | AWS RDS PostgreSQL | Managed database service |
| **DNS** | AWS Route53 | Domain management |

## 🎨 **Design Patterns**

### **1. Repository Pattern**

// Service Layer (Repository Pattern)  
class AssetManagementService {  
 async createCostRecord(data, userId) {  
 const costRecord = {  
 ...data,  
 createdBy: userId,  
 lastModifiedBy: userId  
 };  
   
 const [result] = await db.insert(assetCostManagement)  
 .values(costRecord)  
 .returning();  
   
 return result;  
 }  
}

### **2. Controller Pattern**

// Controller Layer  
class AssetManagementController {  
 async createCostRecord(req, res) {  
 try {  
 const { error, value } = assetCostManagementSchema.create.validate(req.body);  
 if (error) {  
 return res.status(400).json({   
 error: 'Validation failed',   
 details: error.details   
 });  
 }  
  
 const result = await assetManagementService.createCostRecord(value, req.user.id);  
   
 res.status(201).json({  
 message: 'Cost record created successfully',  
 data: result  
 });  
 } catch (error) {  
 console.error('Error creating cost record:', error);  
 res.status(500).json({ error: 'Internal server error' });  
 }  
 }  
}

### **3. Middleware Pattern**

// Authentication Middleware  
const authenticateToken = async (req, res, next) => {  
 try {  
 const authHeader = req.headers.authorization;  
 const token = authHeader && authHeader.split(' ')[1];  
  
 if (!token) {  
 return res.status(401).json({  
 success: false,  
 message: 'Access token is required',  
 });  
 }  
  
 const decoded = jwt.verify(token, config.JWT\_SECRET);  
 const [user] = await db.select().from(users).where(eq(users.id, decoded.userId));  
   
 if (!user) {  
 return res.status(401).json({  
 success: false,  
 message: 'User not found',  
 });  
 }  
  
 req.user = user;  
 next();  
 } catch (error) {  
 return res.status(401).json({  
 success: false,  
 message: 'Invalid token',  
 });  
 }  
};

### **4. Factory Pattern (Frontend)**

// API Factory Pattern  
const createApiClient = (baseURL) => {  
 const getAuthToken = () => localStorage.getItem('accessToken');  
   
 const createHeaders = () => ({  
 'Authorization': `Bearer ${getAuthToken()}`,  
 'Content-Type': 'application/json',  
 });  
  
 return {  
 async get(endpoint) {  
 const response = await fetch(`${baseURL}${endpoint}`, {  
 method: 'GET',  
 headers: createHeaders(),  
 });  
 return handleResponse(response);  
 },  
   
 async post(endpoint, data) {  
 const response = await fetch(`${baseURL}${endpoint}`, {  
 method: 'POST',  
 headers: createHeaders(),  
 body: JSON.stringify(data),  
 });  
 return handleResponse(response);  
 }  
 };  
};

## 🔐 **RBAC Security Implementation**

### **Permission Structure**

Domain:Action format (e.g., asset\_management:read)

**Standard Permissions:** - {domain}:create - Create new resources - {domain}:read - View resources - {domain}:update - Update existing resources - {domain}:delete - Delete resources - {domain}:admin - Full administrative access

### **RBAC Middleware Implementation**

#### **1. Basic Permission Check**

const requirePermission = (requiredPermission) => {  
 return async (req, res, next) => {  
 try {  
 if (!req.user) {  
 return res.status(401).json({  
 success: false,  
 message: 'Authentication required',  
 });  
 }  
  
 const userPermissions = await getUserPermissions(req.user.id);  
   
 if (!userPermissions.has(requiredPermission)) {  
 return res.status(403).json({  
 success: false,  
 message: `Permission denied. Required permission: ${requiredPermission}`,  
 });  
 }  
  
 next();  
 } catch (error) {  
 console.error('RBAC middleware error:', error);  
 return res.status(500).json({  
 success: false,  
 message: 'Authorization error',  
 });  
 }  
 };  
};

#### **2. Route Protection Example**

// Apply authentication and permission middleware to all routes  
router.use(authenticateToken);  
router.use(requirePermission('asset\_management:read'));  
  
// GET /api/v1/asset-tags/keys - Get all unique tag keys  
router.get('/keys', assetTagsController.getTagKeys);

#### **3. Frontend Permission Checking**

// Frontend permission checking  
const hasPermission = (requiredPermission) => {  
 const userPermissions = getUserPermissions(); // From auth context  
 return userPermissions.includes(requiredPermission);  
};  
  
// Conditional rendering based on permissions  
{hasPermission('asset\_management:create') && (  
 <Button onClick={handleCreateAsset}>Create Asset</Button>  
)}

## 🗄️ **Database Design Standards**

### **Schema Naming Conventions**

* **Tables**: snake\_case (e.g., asset\_cost\_management)
* **Columns**: snake\_case (e.g., created\_at, asset\_uuid)
* **Primary Keys**: id (serial) or {table}\_uuid (UUID)
* **Foreign Keys**: {referenced\_table}\_id (e.g., user\_id)
* **Indexes**: idx\_{table}\_{column} (e.g., idx\_assets\_hostname)

### **Standard Columns**

Every table should include:

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW() NOT NULL,  
updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW() NOT NULL,  
created\_by INTEGER REFERENCES users(id),  
last\_modified\_by INTEGER REFERENCES users(id)

### **Schema Definition Example**

const assets = pgTable('assets', {  
 id: serial('id').primaryKey(),  
 assetUuid: uuid('asset\_uuid').notNull().unique(),  
 hostname: varchar('hostname', { length: 255 }),  
 netbiosName: varchar('netbios\_name', { length: 100 }),  
 systemId: varchar('system\_id', { length: 50 }).references(() => systems.systemId),  
 hasAgent: boolean('has\_agent').default(false),  
 hasPluginResults: boolean('has\_plugin\_results').default(false),  
 firstSeen: timestamp('first\_seen'),  
 lastSeen: timestamp('last\_seen'),  
 exposureScore: integer('exposure\_score'),  
 acrScore: numeric('acr\_score', { precision: 3, scale: 1 }),  
 criticalityRating: varchar('criticality\_rating', { length: 20 }),  
 createdAt: timestamp('created\_at').defaultNow(),  
 updatedAt: timestamp('updated\_at').defaultNow(),  
 source: varchar('source', { length: 50 }).default('tenable'),  
 batchId: uuid('batch\_id'),  
 rawJson: jsonb('raw\_json'),  
});

### **Relationship Patterns**

* **One-to-Many**: Use foreign keys
* **Many-to-Many**: Use junction tables
* **Polymorphic**: Use type discriminator columns
* **Audit Trail**: Separate audit tables with triggers

## 🔌 **API Development Guidelines**

### **RESTful API Standards**

#### **URL Structure**

/api/v1/{resource}/{id?}/{sub-resource?}

**Examples:** - GET /api/v1/assets - Get all assets - GET /api/v1/assets/{uuid} - Get specific asset - POST /api/v1/assets - Create new asset - PUT /api/v1/assets/{uuid} - Update asset - DELETE /api/v1/assets/{uuid} - Delete asset - GET /api/v1/assets/{uuid}/tags - Get asset tags

#### **HTTP Status Codes**

* 200 - OK (successful GET, PUT)
* 201 - Created (successful POST)
* 204 - No Content (successful DELETE)
* 400 - Bad Request (validation errors)
* 401 - Unauthorized (authentication required)
* 403 - Forbidden (insufficient permissions)
* 404 - Not Found (resource doesn’t exist)
* 500 - Internal Server Error (server errors)

#### **Response Format**

// Success Response  
{  
 "success": true,  
 "data": {...},  
 "message": "Operation completed successfully",  
 "meta": {  
 "page": 1,  
 "limit": 50,  
 "total": 150  
 }  
}  
  
// Error Response  
{  
 "success": false,  
 "message": "Error description",  
 "error": "ERROR\_CODE",  
 "details": {...}  
}

### **Route Structure Template**

const express = require('express');  
const controller = require('../controllers/resourceController');  
const { authenticateToken } = require('../middleware/auth');  
const { requirePermission } = require('../middleware/rbac');  
  
const router = express.Router();  
  
// Apply authentication to all routes  
router.use(authenticateToken);  
  
// GET /api/v1/resource - List resources  
router.get('/',  
 requirePermission('resource:read'),  
 controller.getAll  
);  
  
// GET /api/v1/resource/:id - Get specific resource  
router.get('/:id',  
 requirePermission('resource:read'),  
 controller.getById  
);  
  
// POST /api/v1/resource - Create resource  
router.post('/',  
 requirePermission('resource:create'),  
 controller.create  
);  
  
// PUT /api/v1/resource/:id - Update resource  
router.put('/:id',  
 requirePermission('resource:update'),  
 controller.update  
);  
  
// DELETE /api/v1/resource/:id - Delete resource  
router.delete('/:id',  
 requirePermission('resource:delete'),  
 controller.delete  
);  
  
module.exports = router;

### **Controller Template**

const service = require('../services/resourceService');  
const { validationResult } = require('express-validator');  
  
class ResourceController {  
 async getAll(req, res) {  
 try {  
 const filters = {  
 page: parseInt(req.query.page) || 1,  
 limit: parseInt(req.query.limit) || 50,  
 sortBy: req.query.sortBy || 'createdAt',  
 sortOrder: req.query.sortOrder || 'desc'  
 };  
  
 const result = await service.getAll(filters);  
  
 res.status(200).json({  
 success: true,  
 data: result.data,  
 meta: {  
 page: filters.page,  
 limit: filters.limit,  
 total: result.total  
 },  
 message: 'Resources retrieved successfully'  
 });  
 } catch (error) {  
 console.error('Error in getAll:', error);  
 res.status(500).json({  
 success: false,  
 message: error.message || 'Failed to retrieve resources'  
 });  
 }  
 }  
  
 async create(req, res) {  
 try {  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({  
 success: false,  
 message: 'Validation failed',  
 errors: errors.array()  
 });  
 }  
  
 const result = await service.create(req.body, req.user.id);  
  
 res.status(201).json({  
 success: true,  
 data: result,  
 message: 'Resource created successfully'  
 });  
 } catch (error) {  
 console.error('Error in create:', error);  
 res.status(500).json({  
 success: false,  
 message: error.message || 'Failed to create resource'  
 });  
 }  
 }  
}  
  
module.exports = new ResourceController();

## ⚛️ **Frontend Development Standards**

### **Component Structure**

src/  
├── components/ # Reusable components  
│ ├── common/ # Generic components  
│ ├── forms/ # Form components  
│ └── ui/ # UI components  
├── pages/ # Page components  
│ └── assets/ # Feature-specific pages  
│ ├── components/ # Page-specific components  
│ └── utils/ # Page-specific utilities  
├── utils/ # Utility functions  
│ ├── api/ # API clients  
│ ├── auth/ # Authentication utilities  
│ └── helpers/ # Helper functions  
├── hooks/ # Custom React hooks  
├── context/ # React context providers  
└── assets/ # Static assets  
 ├── scss/ # Stylesheets  
 └── images/ # Images

### **API Client Template**

const API\_BASE\_URL = 'http://localhost:3001/api/v1';  
  
const getAuthToken = () => localStorage.getItem('accessToken');  
  
const createHeaders = () => ({  
 'Authorization': `Bearer ${getAuthToken()}`,  
 'Content-Type': 'application/json',  
});  
  
const handleResponse = async (response) => {  
 if (!response.ok) {  
 const errorData = await response.json().catch(() => ({}));  
 throw new Error(errorData.message || `HTTP error! status: ${response.status}`);  
 }  
 return await response.json();  
};  
  
export const resourceApi = {  
 async getAll(params = {}) {  
 try {  
 const queryString = new URLSearchParams(params).toString();  
 const response = await fetch(`${API\_BASE\_URL}/resources?${queryString}`, {  
 method: 'GET',  
 headers: createHeaders(),  
 });  
 return await handleResponse(response);  
 } catch (error) {  
 throw new Error(error.message || 'Failed to fetch resources');  
 }  
 },  
  
 async getById(id) {  
 try {  
 const response = await fetch(`${API\_BASE\_URL}/resources/${id}`, {  
 method: 'GET',  
 headers: createHeaders(),  
 });  
 return await handleResponse(response);  
 } catch (error) {  
 throw new Error(error.message || 'Failed to fetch resource');  
 }  
 },  
  
 async create(data) {  
 try {  
 const response = await fetch(`${API\_BASE\_URL}/resources`, {  
 method: 'POST',  
 headers: createHeaders(),  
 body: JSON.stringify(data),  
 });  
 return await handleResponse(response);  
 } catch (error) {  
 throw new Error(error.message || 'Failed to create resource');  
 }  
 },  
  
 async update(id, data) {  
 try {  
 const response = await fetch(`${API\_BASE\_URL}/resources/${id}`, {  
 method: 'PUT',  
 headers: createHeaders(),  
 body: JSON.stringify(data),  
 });  
 return await handleResponse(response);  
 } catch (error) {  
 throw new Error(error.message || 'Failed to update resource');  
 }  
 },  
  
 async delete(id) {  
 try {  
 const response = await fetch(`${API\_BASE\_URL}/resources/${id}`, {  
 method: 'DELETE',  
 headers: createHeaders(),  
 });  
 return await handleResponse(response);  
 } catch (error) {  
 throw new Error(error.message || 'Failed to delete resource');  
 }  
 }  
};

## 📁 **Code Organization**

### **Backend Structure**

api/  
├── src/  
│ ├── controllers/ # Request handlers  
│ ├── services/ # Business logic  
│ ├── middleware/ # Express middleware  
│ ├── routes/ # Route definitions  
│ ├── db/ # Database related  
│ │ ├── schema/ # Database schemas  
│ │ ├── migrations/ # Database migrations  
│ │ └── seeds/ # Seed data  
│ ├── utils/ # Utility functions  
│ ├── config/ # Configuration files  
│ └── app.js # Application entry point  
├── tests/ # Test files  
├── docs/ # Documentation  
└── scripts/ # Utility scripts

### **Frontend Structure**

client/  
├── src/  
│ ├── components/ # Reusable components  
│ ├── pages/ # Page components  
│ ├── utils/ # Utility functions  
│ ├── hooks/ # Custom hooks  
│ ├── context/ # React contexts  
│ ├── assets/ # Static assets  
│ ├── route/ # Routing configuration  
│ └── layout/ # Layout components  
├── public/ # Public assets  
└── dist/ # Build output

### **Naming Conventions**

#### **Files and Directories**

* **Components**: PascalCase.jsx (e.g., AssetInventory.jsx)
* **Utilities**: camelCase.js (e.g., assetTagsApi.js)
* **Constants**: UPPER\_SNAKE\_CASE.js (e.g., API\_ENDPOINTS.js)
* **Directories**: kebab-case (e.g., asset-management)

#### **Variables and Functions**

* **Variables**: camelCase (e.g., assetData)
* **Functions**: camelCase (e.g., fetchAssets)
* **Constants**: UPPER\_SNAKE\_CASE (e.g., API\_BASE\_URL)
* **Classes**: PascalCase (e.g., AssetManagementService)

## 🧪 **Testing Standards**

### **Testing Strategy**

* **Unit Tests**: Individual functions and components
* **Integration Tests**: API endpoints and database operations
* **E2E Tests**: Complete user workflows
* **Security Tests**: Authentication and authorization

### **Backend Testing Template**

const request = require('supertest');  
const app = require('../src/app');  
const { db } = require('../src/db');  
  
describe('Asset Management API', () => {  
 let authToken;  
 let testAssetId;  
  
 beforeAll(async () => {  
 // Setup test database  
 await db.migrate.latest();  
  
 // Create test user and get auth token  
 const loginResponse = await request(app)  
 .post('/api/v1/auth/login')  
 .send({  
 email: 'test@example.com',  
 password: 'testpassword'  
 });  
  
 authToken = loginResponse.body.token;  
 });  
  
 afterAll(async () => {  
 // Cleanup test database  
 await db.destroy();  
 });  
  
 describe('GET /api/v1/assets', () => {  
 it('should return assets list for authenticated user', async () => {  
 const response = await request(app)  
 .get('/api/v1/assets')  
 .set('Authorization', `Bearer ${authToken}`)  
 .expect(200);  
  
 expect(response.body.success).toBe(true);  
 expect(Array.isArray(response.body.data)).toBe(true);  
 });  
  
 it('should return 401 for unauthenticated request', async () => {  
 await request(app)  
 .get('/api/v1/assets')  
 .expect(401);  
 });  
 });  
});

## 🚀 **Deployment Guidelines**

### **Docker Configuration**

#### **Dockerfile Template**

# Multi-stage build for production  
FROM node:20-alpine AS builder  
  
WORKDIR /app  
  
# Copy package files  
COPY package\*.json ./  
COPY client/package\*.json ./client/  
COPY api/package\*.json ./api/  
  
# Install dependencies  
RUN npm install  
RUN cd client && npm install  
RUN cd api && npm install  
  
# Copy source code  
COPY . .  
  
# Build client  
RUN cd client && npm run build  
  
# Production stage  
FROM node:20-alpine AS production  
  
# Install system dependencies  
RUN apk add --no-cache \  
 postgresql-client \  
 python3 \  
 make \  
 g++ \  
 curl \  
 && rm -rf /var/cache/apk/\*  
  
WORKDIR /app  
  
# Copy package files and install production dependencies  
COPY api/package\*.json ./  
RUN npm ci --only=production  
  
# Copy built application  
COPY --from=builder /app/api ./  
COPY --from=builder /app/client/dist ./public  
  
# Create necessary directories  
RUN mkdir -p /app/data /app/logs /app/uploads  
  
# Set environment variables  
ENV NODE\_ENV=production  
ENV PORT=3001  
ENV CLIENT\_PORT=3000  
  
# Expose ports  
EXPOSE 3001 3000  
  
# Health check  
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \  
 CMD curl -f http://localhost:3001/health || exit 1  
  
# Start command  
CMD ["npm", "start"]

#### **Environment Configuration**

# Database Configuration  
DATABASE\_URL=postgresql://username:password@host:5432/database  
DB\_HOST=rasdash-dev-public.cexgrlslydeh.us-east-1.rds.amazonaws.com  
DB\_PORT=5432  
DB\_NAME=rasdashdev01  
DB\_USER=rasdashadmin  
DB\_PASSWORD=RasDash2025$  
DB\_SSL=true  
  
# Application Configuration  
NODE\_ENV=production  
PORT=3001  
JWT\_SECRET=your-super-secret-jwt-key  
CORS\_ORIGIN=https://rasdash.dev.com,http://localhost:3000  
DOMAIN=rasdash.dev.com  
  
# AWS Configuration  
AWS\_REGION=us-east-1

### **Deployment Checklist**

#### **Pre-Deployment**

* All tests passing
* Code reviewed and approved
* Environment variables configured
* Database migrations ready
* Security scan completed
* Performance testing done

#### **Deployment Steps**

1. **Build and test Docker images**
2. **Run database migrations**
3. **Deploy to staging environment**
4. **Run smoke tests**
5. **Deploy to production**
6. **Verify health checks**
7. **Monitor application logs**

#### **Post-Deployment**

* Health checks passing
* Application logs normal
* Database connections stable
* API endpoints responding
* Frontend loading correctly
* User authentication working

## 🎯 **Development Workflow**

### **Git Workflow**

1. **Feature Branch**: Create from develop
2. **Development**: Implement feature with tests
3. **Code Review**: Create pull request
4. **Testing**: Automated tests must pass
5. **Merge**: Merge to develop
6. **Release**: Merge develop to main

### **Branch Naming**

* feature/asset-management-api
* bugfix/authentication-issue
* hotfix/security-patch
* chore/update-dependencies

### **Commit Messages**

type(scope): description  
  
feat(assets): add asset cost management API  
fix(auth): resolve JWT token expiration issue  
docs(api): update asset management documentation  
test(assets): add unit tests for asset service

## 📚 **Additional Resources**

### **Documentation Standards**

* **API Documentation**: Use Swagger/OpenAPI 3.0
* **Code Comments**: JSDoc for functions and classes
* **README Files**: Include setup, usage, and examples
* **Architecture Diagrams**: Use Mermaid or similar tools

### **Code Quality Tools**

* **Linting**: ESLint for JavaScript/React
* **Formatting**: Prettier for consistent code style
* **Type Checking**: Consider TypeScript for large projects
* **Security**: Use tools like npm audit, Snyk

### **Monitoring and Logging**

* **Application Logs**: Structured logging with Winston
* **Error Tracking**: Consider Sentry or similar
* **Performance Monitoring**: APM tools for production
* **Health Checks**: Implement comprehensive health endpoints

This development guide provides a comprehensive framework for maintaining consistency across the CYPHER Dashboard project. All developers should follow these standards to ensure code quality, security, and maintainability.

### **Controller Template**

const service = require('../services/resourceService');  
const { validationResult } = require('express-validator');  
  
class ResourceController {  
 async getAll(req, res) {  
 try {  
 const filters = {  
 page: parseInt(req.query.page) || 1,  
 limit: parseInt(req.query.limit) || 50,  
 sortBy: req.query.sortBy || 'createdAt',  
 sortOrder: req.query.sortOrder || 'desc'  
 };  
  
 const result = await service.getAll(filters);  
  
 res.status(200).json({  
 success: true,  
 data: result.data,  
 meta: {  
 page: filters.page,  
 limit: filters.limit,  
 total: result.total  
 },  
 message: 'Resources retrieved successfully'  
 });  
 } catch (error) {  
 console.error('Error in getAll:', error);  
 res.status(500).json({  
 success: false,  
 message: error.message || 'Failed to retrieve resources'  
 });  
 }  
 }  
  
 async create(req, res) {  
 try {  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({  
 success: false,  
 message: 'Validation failed',  
 errors: errors.array()  
 });  
 }  
  
 const result = await service.create(req.body, req.user.id);  
  
 res.status(201).json({  
 success: true,  
 data: result,  
 message: 'Resource created successfully'  
 });  
 } catch (error) {  
 console.error('Error in create:', error);  
 res.status(500).json({  
 success: false,  
 message: error.message || 'Failed to create resource'  
 });  
 }  
 }  
}  
  
module.exports = new ResourceController();

## ⚛️ **Frontend Development Standards**

### **Component Structure**

src/  
├── components/ # Reusable components  
│ ├── common/ # Generic components  
│ ├── forms/ # Form components  
│ └── ui/ # UI components  
├── pages/ # Page components  
│ └── assets/ # Feature-specific pages  
│ ├── components/ # Page-specific components  
│ └── utils/ # Page-specific utilities  
├── utils/ # Utility functions  
│ ├── api/ # API clients  
│ ├── auth/ # Authentication utilities  
│ └── helpers/ # Helper functions  
├── hooks/ # Custom React hooks  
├── context/ # React context providers  
└── assets/ # Static assets  
 ├── scss/ # Stylesheets  
 └── images/ # Images

### **Component Template**

import React, { useState, useEffect } from 'react';  
import { Button, Card, CardBody } from 'reactstrap';  
import { Icon } from '@/components/Component';  
import { resourceApi } from '@/utils/api';  
import { toast } from 'react-toastify';  
  
const ResourceComponent = ({  
 resourceId,  
 onUpdate,  
 className = ''  
}) => {  
 const [data, setData] = useState(null);  
 const [loading, setLoading] = useState(false);  
 const [error, setError] = useState(null);  
  
 useEffect(() => {  
 if (resourceId) {  
 fetchData();  
 }  
 }, [resourceId]);  
  
 const fetchData = async () => {  
 try {  
 setLoading(true);  
 setError(null);  
  
 const response = await resourceApi.getById(resourceId);  
 if (response.success) {  
 setData(response.data);  
 } else {  
 throw new Error(response.message);  
 }  
 } catch (err) {  
 console.error('Error fetching data:', err);  
 setError(err.message);  
 toast.error(`Failed to load data: ${err.message}`);  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const handleAction = async () => {  
 try {  
 setLoading(true);  
  
 const response = await resourceApi.performAction(resourceId);  
 if (response.success) {  
 toast.success('Action completed successfully');  
 onUpdate?.(response.data);  
 } else {  
 throw new Error(response.message);  
 }  
 } catch (err) {  
 console.error('Error performing action:', err);  
 toast.error(`Action failed: ${err.message}`);  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 if (loading) {  
 return <div className="text-center">Loading...</div>;  
 }  
  
 if (error) {  
 return <div className="text-danger">Error: {error}</div>;  
 }  
  
 return (  
 <Card className={className}>  
 <CardBody>  
 {data ? (  
 <>  
 <h5>{data.name}</h5>  
 <p>{data.description}</p>  
 <Button  
 color="primary"  
 onClick={handleAction}  
 disabled={loading}  
 >  
 <Icon name="check" /> Action  
 </Button>  
 </>  
 ) : (  
 <p>No data available</p>  
 )}  
 </CardBody>  
 </Card>  
 );  
};  
  
export default ResourceComponent;