Production Optimiser Frontend Documentation

Technical Documentation

December 9, 2024

Contents

Chapter 1

Project Overview

1.1 Introduction

The Production Optimiser is a sophisticated React-TypeScript application designed for production optimization management. It provides role-based access control, real-time data visualization, and comprehensive production management capabilities.

1.2 Key Features

- **Role-based Authentication** (Admin/Customer)
- **Dynamic Theme Switching** (Light/Dark)
- **Real-time Production Data Visualization**
- **Admin Dashboard** for user and model management
- **Responsive Design** for all devices
- **Secure API Integration** with JWT
- **Production Optimization Tracking**

Chapter 2

Getting Started

2.1 Prerequisites

```
Required software
node -v  # v22.0.0 or higher
npm -v   # v10.0.0 or higher
git --version  # v2.0.0 or higher
```

2.2 Installation

```
Clone repository
git clone [repository-url]
cd production-optimiser
Install dependencies
npm install
Set up environment
cp .env.example .env
Edit .env with your configuration
Start development server
npm run dev          # Start development server
npm run preview      # Preview production build
```

2.3 Dependency Updates

```
Check outdated packages
npm outdated
Update dependencies
npm update
Security audit
npm audit
npm audit fix
```

Chapter 3

Resource Links

3.1 Official Documentation

- React Documentation
- TypeScript Documentation
- Tailwind CSS
- Vite
- React Router

3.2 Tools and Libraries

- shadcn/ui Components
- Radix UI
- Recharts
- Biome

Chapter 4

Team Contacts

4.1 Development Team

4.2 Support

4.3 Business Logic

- **User Role Hierarchy**
  - Admin: Full system access
  - Customer: Limited access
- **Model Management Workflow**
- **Production Optimization Algorithms**
- **Data Visualization Patterns**

Chapter 5

Technical Architecture

5.1 Core Technologies

Frontend Framework
React 18.3.1

Language
TypeScript 4.9.5

Build Tool
Vite 5.4.10

State Management
React Context + Hooks

Routing
React Router 6.28.0

Styling
Tailwind CSS 3.4.14

Components
shadcn/ui + Radix UI

Charts
Recharts 2.13.3

HTTP Client
Axios 1.7.7

5.2 Dependencies

5.2.1 Core Dependencies

```
{
  "dependencies": {
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "typescript": "^4.9.5",
    "react-router-dom": "^6.28.0",
    "axios": "^1.7.7",
    "recharts": "^2.13.3"
  }
}
```

5.2.2 Development Dependencies

```
{
  "devDependencies": {
    "vite": "^5.4.10",
    "@biomejs/biome": "^1.9.4",
    "tailwindcss": "^3.4.14",
    "@types/react": "^18.3.12",
    "@types/react-dom": "^18.3.1"
  }
}
```

Chapter 6

Project Structure

6.1 Directory Organization

```
frontend/
├── public/ ......................................... Static assets
│   ├── favicon.ico
│   ├── logo192.png
│   └── robots.txt
├── src/
│   ├── Components/ ............................ Reusable components
│   │   ├── ui/ ................................ Base UI components
│   │   ├── UserProfile/
│   │   ├── ThemeProvider.tsx
│   │   └── ProtectedRoute.tsx
│   ├── Contexts/ .................................... React contexts
│   │   ├── authContext.tsx
│   │   └── ThemeContext.tsx
│   ├── Layouts/ ................................ Layout components
│   ├── Pages/ .................................... Page components
│   ├── services/ ...................................... API services
│   ├── types/ ................................... TypeScript types
│   └── utils/ ................................. Utility functions
└── configuration files
```

6.2 Key Files

App.tsx
Application entry point

index.tsx
React DOM rendering

vite.config.js
Build configuration

tailwind.config.js
Styling configuration

tsconfig.json
TypeScript configuration

Chapter 7

Authentication System

7.1 JWT Implementation

```
const authService = {
async login(email: string, password: string): Promise<User> {
const response = await axiosInstance.post<
AuthenticationResponseDTO>(
'/auth/login',
{ email, password }
);
const { token } = response.data;
localStorage.setItem('token', token);
return user;
}
};
```

7.2 Protected Routes

```
interface ProtectedRouteProps {
  children: React.ReactNode;
  requiredRole: Role;
}

export const ProtectedRoute = ({
  children,
  requiredRole,
}: ProtectedRouteProps) => {
  const { isAuthenticated, hasRole } = useAuth();
  if (!isAuthenticated) return <Navigate to="/login" replace />;
  if (!hasRole(requiredRole)) return <Navigate to="/unauthorized"
   replace />;
  return <>{children}</>;
};
```

7.3 Role Management

```
export type Role = 'CUSTOMER' | 'ADMIN';

interface User {
  id: string;
  email: string;
  roles: Role[];
}
```

Chapter 8

Component System

8.1 Base Components

8.1.1 Button Component

```
const Button: React.FC<ButtonProps> = ({
  variant = "default",
  size = "default",
  children,
  className,
  ...props
}) => {
  return (
    <button
      className={cn(
        buttonVariants({ variant, size, className })
      )}
      {...props}
    >
      {children}
    </button>
  );
};
```

8.2 Layout Components

```
const AppLayout: React.FC<AppLayoutProps> = ({ children }) => {
  return (
    <div className="flex h-screen bg-background">
      <div className="flex-1 flex flex-col">
        <header className="border-b">
          <NavBar />
        </header>
        <main className="flex-1 overflow-auto">
          {children}
        </main>
      </div>
    </div>
  );
};
```

Chapter 9

State Management

9.1 Context Implementation

9.1.1 Authentication Context

```
export const AuthProvider: FC<{ children: React.ReactNode }> = ({
  children
}) => {
  const [user, setUser] = useState<User | null>(() => {
    const savedUser = localStorage.getItem('user');
    return savedUser ? JSON.parse(savedUser) : null;
  });

  const login = (userData: User) => {
    localStorage.setItem('user', JSON.stringify(userData));
    setUser(userData);
  };

  const logout = () => {
    localStorage.removeItem('user');
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{
      user,
      isAuthenticated: !!user,
      login,
      logout
    }}>
      {children}
    </AuthContext.Provider>
  );
};
```

9.2 Custom Hooks

9.2.1 Mobile Detection Hook

```
export function useIsMobile() {
  const [isMobile, setIsMobile] = useState<boolean>(false);

  useEffect(() => {
    const mql = window.matchMedia(`(max-width: 768px)`);
    const onChange = () => {
      setIsMobile(window.innerWidth < 768);
    };

    mql.addEventListener('change', onChange);
    setIsMobile(window.innerWidth < 768);

    return () => mql.removeEventListener('change', onChange);
  }, []);

  return isMobile;
}
```

Chapter 10

API Integration

10.1 Axios Configuration

```
const instance = axios.create({
  baseURL: API_URL,
  withCredentials: true,
  timeout: 15000,
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
});

// Request Interceptor
instance.interceptors.request.use(
  (config: InternalAxiosRequestConfig) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  }
);

// Response Interceptor
instance.interceptors.response.use(
  response => response,
  async (error: AxiosError) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

10.2 Error Handling

```
export const handleApiError = (error: AxiosError): ApiError => {
  if (error.response) {
    return {
      status: error.response.status,
      message: error.response.data?.message || 'An error occurred',
      data: error.response.data
    };
  }

  return {
    status: 0,
    message: error.message || 'Request failed',
    data: null
  };
};
```

Chapter 11

Performance Optimization

11.1 Code Splitting

```
// Lazy loading components
const AdminDashboard = React.lazy(() => import('./Pages/
    adminDashboard'));

function App() {
  return (
    <Suspense fallback={<Loading />}>
      <AdminDashboard />
    </Suspense>
  );
}
```

11.2 Memoization

```
const MemoizedComponent = React.memo(({ prop1, prop2 }) => {
  return (
    <div>
      <h1>{prop1}</h1>
      <p>{prop2}</p>
    </div>
  );
});
```

Chapter 12

Testing Strategy

```
import { render, screen } from '@testing-library/react';
import { Button } from './Button';

describe('Button', () => {
  it('renders correctly', () => {
    render(<Button>Click me</Button>);
    expect(screen.getByText('Click me')).toBeInTheDocument();
  });

  it('handles click events', () => {
    const handleClick = jest.fn();
    render(<Button onClick={handleClick}>Click me</Button>);
    screen.getByText('Click me').click();
    expect(handleClick).toHaveBeenCalledTimes(1);
  });
});
```

Chapter 13

Deployment

13.1 Docker Configuration

```
FROM node:22-alpine
WORKDIR /app
CMD ["npm", "run", "build"]
COPY package.json /app
RUN npm i
COPY . /app
```

13.2 Build Process

```
# Production build
npm run build

# Preview build
npm run preview
```

Chapter 14

Security Considerations

14.1 Authentication Security

- JWT token storage in localStorage
- Automatic token refresh mechanism
- Secure route protection
- Role-based access control

14.2 API Security

- CORS configuration
- Token-based authentication
- Request/Response encryption
- Error handling and logging

Chapter 15

Maintenance Guide

15.1 Regular Tasks

Daily
- Code reviews
- Bug triage
- CI/CD monitoring

Weekly
- Dependency updates
- Performance monitoring
- Code quality checks

Monthly
- Security audits
- Documentation updates
- Technical debt review

15.2 Troubleshooting

Authentication Issues
- Check token expiration
- Verify localStorage access
- Confirm API connectivity

Performance Issues
- Monitor component re-renders
- Check network requests
- Analyze bundle size

Chapter 16

Appendix

16.1 Useful Commands

```
# Development
npm run dev           # Start development server
npm run format        # Format code
npm run lint          # Lint code

# Production
npm run build         # Build for production
npm run preview       # Preview production build
```