# **Logo Builder Frontend - Comprehensive Documentation**

## **Project Overview**

**Project Name:** Logo Builder Frontend

**Type:** React Single Page Application (SPA)

**Primary Framework:** React 19.0.0

**State Management:** Redux Toolkit, React Query

**UI Framework:** Material-UI (MUI) v6.4.7

**Build Tool:** Webpack 5.98.0

**Language:** JavaScript/JSX with some TypeScript support

### **Business Domain**

The Logo Builder Frontend is a sophisticated web application designed for creating, editing, and managing custom logos. It serves as a comprehensive logo design tool with features including:

Interactive logo creation and editing

Template-based logo generation

Real-time preview and variations

Logo download and sharing capabilities

User account management and logo storage

Micro-frontend architecture support

## **Architecture Overview**

### **Current State Architecture**

graph TD  
 A[React SPA] --> B[Redux Store]  
 A --> C[React Query]  
 A --> D[Material-UI Components]  
   
 B --> E[Logo State Management]  
 B --> F[User Authentication]  
 B --> G[Application Config]  
   
 C --> H[API Clients]  
 H --> I[Logo API]  
 H --> J[Authentication API]  
 H --> K[Storage API]  
   
 A --> L[Micro-Frontend Modules]  
 L --> M[Logo Editor MFE]  
 L --> N[Onboarding MFE]  
 L --> O[Logo Parcel MFE]  
   
 A --> P[Core Components]  
 P --> Q[Logo Maker Engine]  
 P --> R[Canvas Rendering]  
 P --> S[Template System]

### **Micro-Frontend Architecture**

The application follows a micro-frontend pattern with multiple independent modules:

**app-logo** - Main logo application

**app-logo-mfe** - Logo editor micro-frontend

**app-logo-parcel** - Parcel-based logo module

**app-logo-builder-onboarding** - Onboarding flow

**app-mfe-loader** - Micro-frontend loader

## **File Structure and Sitemap**

### **Root Directory Structure**

├── src/  
│ ├── api-config/ # API configuration and endpoints  
│ ├── clients/ # HTTP clients for external APIs  
│ ├── components/ # Reusable UI components  
│ ├── core/ # Core business logic and utilities  
│ ├── helpers/ # Utility functions and helpers  
│ ├── hooks/ # Custom React hooks  
│ ├── images/ # Static image assets  
│ ├── logomaker/ # Main logo creation engine  
│ ├── modules/ # Feature-based modules  
│ ├── services/ # Micro-frontend services  
│ └── tools/ # Development and build tools  
├── config/ # Build and environment configuration  
├── scripts/ # Build and deployment scripts  
├── test/ # Test configurations and utilities  
├── Documentation/ # Project documentation  
└── .storybook/ # Storybook configuration

### **Detailed File-by-File Structure**

#### **API Configuration (src/api-config/)**

**index.js** - Central API configuration with environment-specific endpoints

#### **HTTP Clients (src/clients/)**

**logo.client.js** - Logo CRUD operations API client

**temporary-logo.client.js** - Temporary logo handling

**limitations.client.js** - Account limitations and restrictions

**ids4.client.js** - Identity server integration

**onboarding-info.client.js** - Onboarding data management

#### **Core Components (src/components/)**

**controls/** - Form controls and input components

**items/** - Individual control components

**colors-modal/** - Color picker functionality

**merch-config/** - Merchandise configuration

**icon-settings-menu/** - Icon customization interface

**logo-area/** - Main logo display area

**logo-share-dialog/** - Logo sharing functionality

**logo-tile/** - Logo thumbnail display

**merch-preview/** - Merchandise preview components

**variations-area/** - Logo variations display

#### **Core Engine (src/core/)**

**components/** - Core UI components

**grid.js** - Logo grid display system

**checkbox-icon.js** - Selection interface

**free-flow-authentication.js** - Authentication flow

**helpers/** - Core utility functions

**api-middleware.js** - API request/response handling

**post-logo.js** - Logo submission logic

**logo-maker/** - Logo creation engine

**images/placeholder/** - Placeholder image system

**layers/shapes/** - Shape rendering components

**layouts/** - Logo layout templates

**resources/** - Fonts, symbols, and assets

#### **Logo Maker Engine (src/logomaker/)**

**index.js** - Main logo editor component (387 lines)

**container.js** - Logo container wrapper

**components/** - Logo editor specific components

**config/** - Editor configuration

**context/** - React context providers

**helpers/** - Editor utility functions

**lang/** - Internationalization

**mock/** - Mock data for testing

**store/** - Logo state management

**styles/** - SCSS styling

#### **Modules (src/modules/)**

**application-config/** - App configuration management

**editor/** - Logo editor module

**routes.js** - Editor routing configuration

**containers/** - Editor page containers

**my-logos/** - User logo management

**onboarding/** - User onboarding flow

**shared-store/** - Shared state management

**transfer/** - Logo transfer functionality

#### **Services (Micro-frontends) (src/services/)**

**app-logo/** - Main logo service

**app-logo-mfe/** - Logo editor micro-frontend

**app-logo-parcel/** - Parcel-based implementation

**app-logo-builder-onboarding/** - Onboarding micro-frontend

**app-mfe-loader/** - Micro-frontend loader

## **Key Dependencies and Libraries**

### **Core Framework Dependencies**

**React 19.0.0** - Main UI framework

**React DOM 19.0.0** - DOM rendering

**React Router 7.3.0** - Client-side routing

**Redux 5.0.1** - State management

**@reduxjs/toolkit 2.6.0** - Modern Redux utilities

### **UI and Styling**

**@mui/material 6.4.7** - Material Design components

**@mui/icons-material 6.4.7** - Material Design icons

**@emotion/react 11.14.0** - CSS-in-JS styling

**@emotion/styled 11.14.0** - Styled components

### **Data Fetching and State**

**@tanstack/react-query 5.67.2** - Server state management

**axios 1.8.2** - HTTP client

**redux-saga 1.3.0** - Side effect management

### **Build and Development**

**webpack 5.98.0** - Module bundler

**babel** - JavaScript transpilation

**sass 1.85.1** - CSS preprocessing

**eslint** - Code linting

**jest 29.7.0** - Testing framework

### **Specialized Libraries**

**single-spa-react 6.0.2** - Micro-frontend framework

**i18next 24.2.2** - Internationalization

**lodash 4.17.21** - Utility functions

**moment 2.30.1** - Date manipulation

**draft-js 0.11.7** - Rich text editing

## **API Integration Documentation**

### **Primary API Endpoints**

#### **Logo Management API**

**Base URL:** <https://api-gw.builderservices.io/logos/>

**Authentication:** Bearer token via Authorization header

**Endpoints:**

GET /v1.0/logos/list?include\_logo=true - Retrieve user logos

GET /v1.0/logos/{logoId}/load - Load specific logo

POST /v1.0/logos/save - Save new logo

POST /v1.0/logos/{logoId}/save - Update existing logo

DELETE /v1.0/logos/{logoId}/delete - Delete logo

POST /v1.0/logos/{logoId}/duplicate - Duplicate logo

POST /v1.0/logos/{logoId}/downloaded - Mark logo as downloaded

#### **Temporary Logo API**

GET /v1.0/temporary-logos/{logoId}/load - Load temporary logo

POST /v1.0/temporary-logos/save - Save temporary logo

PUT /v1.0/temporary-logos/{logoId}/save - Update temporary logo

POST /v1.0/temporary-logos/transfer/{logoId} - Transfer to permanent

#### **Account Limitations API**

GET /v1.0/limitation?resourceId={logoId} - Get logo limitations

GET /v2.0/limitation - Get account limitations

GET /v1.0/accounts/info - Get account information

### **HTTP Client Implementation**

The application uses a custom performFetch wrapper around the native fetch API:

const performFetch = async (url, options = {}) => {  
 const defaultHeaders = {  
 'Content-Type': 'application/json'  
 }  
   
 const mergedOptions = {  
 method: options.method || 'GET',  
 headers: {  
 ...defaultHeaders,  
 ...options.headers  
 },  
 ...(options.body && {   
 body: typeof options.body === 'string'   
 ? options.body   
 : JSON.stringify(options.body)   
 })  
 }  
   
 const response = await window.fetch(url, mergedOptions)  
 return await handleResponse(response)  
}

## **Component Architecture**

### **Core Components**

#### **LogoMakerEditor (src/logomaker/index.js)**

**Purpose:** Main logo editing interface

**Type:** Functional component with hooks

**Key Features:**

Logo template management

Real-time editing capabilities

Undo/redo functionality

Save and export operations

Segment-based editing (name, slogan, symbol, etc.)

**Props:**

selectedLogo - Current logo data

onChange - Logo change handler

saveLogo - Save function

historyLogo - Logo history for undo/redo

#### **Grid Component (src/core/components/grid.js)**

**Purpose:** Logo selection and display grid

**Type:** Functional component with memo optimization

**Key Features:**

Logo template grid display

Selection handling

Animation support

Voting mode for A/B testing

#### **LogoArea (src/components/logo-area/logo-area.js)**

**Purpose:** Main logo display and interaction area

**Type:** React component

**Key Features:**

Logo rendering

Interactive editing

Preview modes

Responsive design

### **Control Components**

The application includes a comprehensive set of control components for logo customization:

**TextControl** - Text input and formatting

**SliderControl** - Numeric value adjustment

**ColorControl** - Color picker interface

**FontControl** - Font selection and styling

**SymbolControl** - Icon and symbol management

## **State Management**

### **Redux Store Structure**

{  
 logos: {  
 list: [],  
 selected: null,  
 loading: false,  
 error: null  
 },  
 editor: {  
 template: {},  
 history: [],  
 activeSegment: 'name',  
 unsavedChanges: false  
 },  
 user: {  
 authenticated: false,  
 limitations: {},  
 account: {}  
 },  
 application: {  
 config: {},  
 modals: {},  
 notifications: []  
 }  
}

### **React Query Integration**

React Query is used for server state management with the following key queries:

useLogos() - Fetch user logos

useLogoById(id) - Fetch specific logo

useLogoUpdate() - Update logo mutation

useLogoDelete() - Delete logo mutation

useAccountLimitations() - Fetch account limits

## **Routing Structure**

### **Main Routes**

const routes = [  
 {  
 path: '/',  
 component: MyLogosPage,  
 pageType: 'MANAGE',  
 flowType: 'LOGO'  
 },  
 {  
 path: '/editor/:logoId/:segment?',  
 component: EditorPage,  
 pageType: 'CREATION',  
 flowType: 'LOGO'  
 },  
 {  
 path: '/editor/:logoId/preview',  
 component: LogoPreview,  
 pageType: 'CREATION',  
 flowType: 'LOGO'  
 },  
 {  
 path: '/editor/:logoId/download',  
 component: LogoDownloadPage,  
 pageType: 'CREATION',  
 flowType: 'LOGO'  
 },  
 {  
 path: '/onboarding/:currentStep/:logoId',  
 component: OnboardingPage,  
 pageType: 'CREATION',  
 flowType: 'LOGO'  
 },  
 {  
 path: '/transfer/:logoId',  
 component: LogoTransfer,  
 pageType: 'CREATION',  
 flowType: 'LOGO'  
 }  
]

## **Security Implementation**

### **Authentication**

JWT-based authentication with Bearer tokens

Session management through AuthUjwtHelper

XSRF token protection for sensitive operations

### **Authorization**

Account-based limitations and restrictions

Feature gating based on user subscription level

Resource-level permissions for logo operations

### **Data Protection**

Input sanitization for user-generated content

Secure API communication over HTTPS

Client-side validation with server-side verification

## **Performance Optimizations**

### **Code Splitting**

React.lazy() for route-based code splitting

Dynamic imports for micro-frontend modules

Webpack chunk optimization

### **Caching Strategies**

React Query caching for API responses

Browser caching for static assets

Service worker implementation for offline support

### **Rendering Optimizations**

React.memo for component memoization

useMemo and useCallback for expensive computations

Virtual scrolling for large logo lists

## **Testing Strategy**

### **Test Configuration**

**Jest** - Unit and integration testing

**React Testing Library** - Component testing

**Cypress** - End-to-end testing (legacy)

**Storybook** - Component documentation and testing

### **Test Coverage Areas**

Component rendering and interaction

API client functionality

State management logic

User workflow scenarios

## **Build and Deployment**

### **Build Configuration**

**Webpack 5** with custom configuration

**Babel** for JavaScript transpilation

**SASS** preprocessing

**ESLint** and **Prettier** for code quality

### **Environment Configuration**

Development, UAT, and Production environments

Environment-specific API endpoints

Feature flags for gradual rollouts

### **Deployment Pipeline**

**Bitbucket Pipelines** for CI/CD

Automated testing and linting

Multi-environment deployment

Asset optimization and minification

## **Internationalization**

### **i18next Implementation**

Multi-language support

Dynamic language switching

Namespace-based translation organization

Browser language detection

### **Supported Languages**

English (default)

Additional languages configurable via translation files

## **Accessibility**

### **WCAG Compliance**

Semantic HTML structure

ARIA labels and roles

Keyboard navigation support

Screen reader compatibility

### **Material-UI Accessibility**

Built-in accessibility features

Focus management

Color contrast compliance

Responsive design principles

## **Error Handling and Monitoring**

### **Error Boundaries**

React error boundaries for component error catching

Graceful degradation for failed components

User-friendly error messages

### **Monitoring Integration**

Application Insights integration

Error tracking and reporting

Performance monitoring

User analytics

## **Future State Architecture (Angular Migration)**

### **Migration Strategy Overview**

The migration from React to Angular would involve a phased approach:

**Component Migration** - Convert React components to Angular components

**State Management** - Replace Redux with NgRx

**Routing** - Migrate to Angular Router

**API Integration** - Update to Angular HttpClient

**Build System** - Transition to Angular CLI

### **Angular Equivalent Mapping**

|  |  |
| --- | --- |
| **React Technology** | **Angular Equivalent** |
| React Components | Angular Components |
| Redux/Redux Toolkit | NgRx Store |
| React Router | Angular Router |
| React Query | Angular HttpClient + RxJS |
| Material-UI | Angular Material |
| Emotion/Styled | Angular Material Theming |
| Webpack | Angular CLI |
| Jest | Jasmine/Karma |

### **Future State Architecture Diagram**

graph TD  
 A[Angular SPA] --> B[NgRx Store]  
 A --> C[Angular Services]  
 A --> D[Angular Material]  
   
 B --> E[Logo State]  
 B --> F[User State]  
 B --> G[App State]  
   
 C --> H[HTTP Interceptors]  
 C --> I[Logo Service]  
 C --> J[Auth Service]  
   
 A --> K[Feature Modules]  
 K --> L[Logo Editor Module]  
 K --> M[Onboarding Module]  
 K --> N[My Logos Module]  
   
 A --> O[Shared Module]  
 O --> P[Components]  
 O --> Q[Directives]  
 O --> R[Pipes]

## **Conclusion**

The Logo Builder Frontend represents a sophisticated React-based application with a well-structured architecture supporting micro-frontends, comprehensive state management, and robust API integration. The codebase demonstrates modern React patterns and best practices, making it well-positioned for future enhancements and potential migration to Angular.

The application's modular design, comprehensive testing strategy, and performance optimizations ensure scalability and maintainability for continued development and feature expansion.