# Component Structure and Organization

## Overview

This document provides a detailed breakdown of the CareIQ Builder component structure, file organization, and code architecture.

## File Organization

### Project Structure

CareIQ Builder/  
├── src/  
│ └── cadal-careiq-builder/  
│ ├── index.js # Main component (21,166 lines)  
│ ├── effects.js # HTTP effects (50+ endpoints)  
│ ├── config-actions.js # Configuration actions  
│ ├── core-actions.js # Core UI actions  
│ ├── utils.js # Utility functions  
│ └── styles.scss # Component styles  
├── package.json # Dependencies and metadata  
└── now-ui.json # Component configuration

## index.js - Main Component (21,166 lines)

**File Location**: src/cadal-careiq-builder/index.js

This is the primary component file containing all view logic, state management, and action handlers.

### File Structure Breakdown

index.js Structure:  
├── Lines 1-16: Imports and dependencies  
├── Lines 17-12000+: View Layer (JSX rendering)  
│ ├── Reusable components (icons, overlays)  
│ ├── Assessment list view  
│ ├── Builder interface  
│ ├── Modals and dialogs  
│ └── System messages  
├── Lines 12000+: State Management  
│ ├── Initial state definition (8500+ lines)  
│ └── Action handlers (100+ actions)  
└── Lines (end): Component registration

## Import Structure (Lines 1-16)

import {createCustomElement, actionTypes} from '@servicenow/ui-core';  
import {createHttpEffect} from '@servicenow/ui-effect-http';  
import snabbdom from '@servicenow/ui-renderer-snabbdom';  
import styles from './styles.scss';  
import packageJson from '../../package.json';  
import {  
 groupAssessmentsByMasterId,  
 paginateAssessments,  
 loadCareIQConfig,  
 hasRelationships,  
 calculateVisibleQuestions  
} from './utils.js';  
import \* as effects from './effects.js';  
import {coreActions} from './core-actions.js';  
import {configActions} from './config-actions.js';  
  
const {COMPONENT\_BOOTSTRAPPED} = actionTypes;

### Key Dependencies

**ServiceNow Framework**: - createCustomElement - Component factory - actionTypes - Standard action type constants - createHttpEffect - HTTP effect creator - snabbdom - Virtual DOM renderer

**Local Modules**: - utils.js - Helper functions for data transformation - effects.js - All HTTP effect definitions - config-actions.js - Configuration/initialization actions - core-actions.js - UI state management actions

**Metadata**: - styles.scss - Component styling - package.json - Version and dependency info

## View Layer Architecture

### View Function Signature

const view = (state, {updateState, dispatch}) => {  
 // Component rendering logic  
};

**Parameters**: - state - Current application state (read-only) - updateState - Function to update state - dispatch - Function to dispatch actions/effects

**Returns**: Virtual DOM tree (JSX)

### Reusable Components

The view layer defines several reusable component functions:

#### CheckIcon Component (Lines 22-26)

const CheckIcon = () => (  
 <svg attrs={{width: "14", height: "14", viewBox: "0 0 16 16", fill: "currentColor"}}>  
 <path attrs={{d: "M13.485 3.429a1 1 0 0 1 0 1.414L6.707 11.62a1 1 0 0 1-1.414 0L2.515 8.843a1 1 0 1 1 1.414-1.414L6 9.5a1 1 0 0 1 0 0l6.071-6.071a1 1 0 0 1 1.414 0z"}} />  
 </svg>  
);

**Usage**: Confirm buttons, success indicators

#### XIcon Component (Lines 28-32)

const XIcon = () => (  
 <svg attrs={{width: "14", height: "14", viewBox: "0 0 16 16", fill: "currentColor"}}>  
 <path attrs={{d: "M3.646 3.646a1 1 0 0 1 1.414 0L8 6.586l2.94-2.94a1 1 0 1 1 1.414 1.414L9.414 8l2.94 2.94a1 1 0 0 1-1.414 1.414L8 9.414l-2.94 2.94a1 1 0 0 1-1.414-1.414L6.586 8 3.646 5.06a1 1 0 0 1 0-1.414z"}} />  
 </svg>  
);

**Usage**: Close buttons, delete actions, cancel operations

#### SpinnerIcon Component (Lines 35-54)

const SpinnerIcon = ({size = "24"}) => (  
 <svg  
 attrs={{  
 width: size,  
 height: size,  
 viewBox: "0 0 24 24",  
 fill: "none",  
 stroke: "currentColor",  
 "stroke-width": "2",  
 "stroke-linecap": "round",  
 "stroke-linejoin": "round"  
 }}  
 style={{  
 animation: "spin 1s linear infinite"  
 }}  
 >  
 <circle attrs={{cx: "12", cy: "12", r: "10", opacity: "0.25"}} />  
 <path attrs={{d: "M12 2 A10 10 0 0 1 22 12", opacity: "0.75"}} />  
 </svg>  
);

**Features**: - Configurable size - CSS animation for spinning - Semi-transparent circle with progress arc

**Usage**: Loading indicators throughout the application

#### LoadingOverlay Component (Lines 56-105)

const LoadingOverlay = ({message = "Loading...", isModal = false}) => (  
 <div  
 style={{  
 position: isModal ? "fixed" : "absolute",  
 top: 0,  
 left: 0,  
 width: "100%",  
 height: "100%",  
 backgroundColor: isModal ? "rgba(0, 0, 0, 0.6)" : "rgba(255, 255, 255, 0.9)",  
 display: "flex",  
 flexDirection: "column",  
 alignItems: "center",  
 justifyContent: "center",  
 zIndex: isModal ? 999999 : 1000,  
 borderRadius: isModal ? "0" : "4px",  
 pointerEvents: "auto",  
 cursor: isModal ? "wait" : "default"  
 }}  
 onclick={(e) => {  
 if (isModal) {  
 e.stopPropagation();  
 e.preventDefault();  
 }  
 }}  
 >  
 <div style={{  
 backgroundColor: "#fff",  
 padding: "32px 48px",  
 borderRadius: "12px",  
 boxShadow: "0 20px 25px -5px rgba(0, 0 0, 0.1), 0 10px 10px -5px rgba(0, 0, 0, 0.04)",  
 display: "flex",  
 flexDirection: "column",  
 alignItems: "center",  
 gap: "16px",  
 minWidth: "200px"  
 }}>  
 <SpinnerIcon size="48" />  
 <div style={{  
 fontSize: "16px",  
 color: "#111827",  
 fontWeight: "600",  
 textAlign: "center",  
 whiteSpace: "nowrap"  
 }}>  
 {message}  
 </div>  
 </div>  
 </div>  
);

**Features**: - Two modes: isModal (full-screen overlay) and inline (section overlay) - Customizable message - Prevents interaction during loading - Centered spinner and message - Smooth styling with shadows and transparency

**Usage**: - Saving questions, answers, sections - Loading PGI data - API operations in progress

## Major View Sections

The view layer renders different UI based on application state:

### 1. System Messages Ticker (Top of View)

**Purpose**: Display success, error, and informational messages

**Key State**: - state.systemMessages - Array of message objects - state.systemMessageHistoryExpanded - Whether history is expanded

**Components**: - Message ticker (scrolling messages) - Connection status indicator - Message history panel (expandable)

**Example Structure**:

<div className="system-messages-ticker">  
 {state.systemMessages.map(msg => (  
 <div className={`message ${msg.type}`}>  
 {msg.message}  
 </div>  
 ))}  
</div>

### 2. Assessment List View

**Purpose**: Display and manage assessments

**Rendered When**: state.builderView === false (not in builder)

**Key State**: - state.assessments - Array of assessment objects - state.searchTerm - Current search filter - state.pageSize - Items per page - state.expandedAssessments - Which assessments show version history

**Components**: - Search box - Page size selector - Assessment cards (with expand/collapse for versions) - New Assessment button - Pagination controls

**Typical Structure**:

if (!state.builderView) {  
 return (  
 <div className="assessment-list">  
 <div className="search-controls">  
 <input  
 type="text"  
 value={state.searchTerm}  
 oninput={(e) => dispatch('FILTER\_ASSESSMENTS', {term: e.target.value})}  
 />  
 </div>  
  
 <div className="assessments">  
 {paginatedAssessments.map(assessment => (  
 <AssessmentCard assessment={assessment} />  
 ))}  
 </div>  
 </div>  
 );  
}

### 3. Builder Interface

**Purpose**: Edit assessment content

**Rendered When**: state.builderView === true

**Layout**: Three-panel design

┌─────────────────────────────────────────────┐  
│ Top Bar (Mode, Actions) │  
├──────────────┬──────────────────────────────┤  
│ Sections │ Questions & Answers │  
│ Panel │ Panel │  
│ (Left) │ (Main) │  
│ │ │  
│ │ │  
└──────────────┴──────────────────────────────┘

**Key State**: - state.currentAssessment - Current assessment data - state.selectedSection - Currently selected section ID - state.currentQuestions - Questions for selected section - state.builderMode - Edit vs Preview mode

#### Builder Top Bar

**Components**: - Assessment title and version - Mode toggle (Edit/Preview/Relationships) - Publish button (Draft status only) - Back to list button

**Example**:

<div className="builder-top-bar">  
 <h2>{state.currentAssessment.name} v{state.currentAssessment.version}</h2>  
  
 <div className="mode-toggles">  
 <button  
 className={state.builderMode ? 'active' : ''}  
 onclick={() => dispatch('TOGGLE\_BUILDER\_MODE')}  
 >  
 Edit Mode  
 </button>  
 <button  
 className={!state.builderMode ? 'active' : ''}  
 onclick={() => dispatch('TOGGLE\_BUILDER\_MODE')}  
 >  
 Preview Mode  
 </button>  
 </div>  
  
 {state.currentAssessment.status === 'Draft' && (  
 <button onclick={() => dispatch('PUBLISH\_ASSESSMENT')}>  
 Publish  
 </button>  
 )}  
</div>

#### Sections Panel (Left)

**Purpose**: Section hierarchy navigation and management

**Components**: - Section tree (collapsible parent/child structure) - Add section button (Edit mode) - Section edit controls (Edit mode) - Section selection highlighting

**Key Features**: - Click section to select and load questions - Expand/collapse parent sections - Add/edit/delete sections (Edit mode only) - Drag-and-drop reordering

**Example Structure**:

<div className="sections-panel">  
 <div className="panel-header">  
 <h3>Sections</h3>  
 {state.builderMode && (  
 <button onclick={() => dispatch('ADD\_SECTION\_DIALOG')}>  
 + Add Section  
 </button>  
 )}  
 </div>  
  
 <div className="sections-list">  
 {state.currentAssessment.sections.map(section => (  
 <SectionItem  
 section={section}  
 selected={section.id === state.selectedSection}  
 onSelect={() => dispatch('SELECT\_SECTION', {sectionId: section.id})}  
 />  
 ))}  
 </div>  
</div>

#### Questions Panel (Main)

**Purpose**: Display and edit questions and answers for selected section

**Components**: - Section label - Question list (numbered) - Each question includes: - Question label - Type selector - Voice selector - Answer list - Save/Cancel buttons (if unsaved) - Delete button - Relationship indicators - Add question button (Edit mode)

**Conditional Rendering**: - **Edit Mode**: All questions visible, full controls - **Preview Mode**: Questions filtered by conditional logic, read-only

**Example Structure**:

<div className="questions-panel">  
 <h3>{state.selectedSectionLabel}</h3>  
  
 <div className="questions-list">  
 {visibleQuestions.map((question, index) => (  
 <div className="question-card">  
 <div className="question-number">{index + 1}.</div>  
  
 <div className="question-content">  
 <input  
 type="text"  
 value={question.label}  
 disabled={!state.builderMode}  
 oninput={(e) => dispatch('UPDATE\_QUESTION\_LABEL', {  
 questionId: question.ids.id,  
 label: e.target.value  
 })}  
 />  
  
 {state.builderMode && (  
 <select  
 value={question.type}  
 onchange={(e) => dispatch('UPDATE\_QUESTION\_TYPE', {  
 questionId: question.ids.id,  
 type: e.target.value  
 })}  
 >  
 <option value="Single Select">Single Select</option>  
 <option value="Multiselect">Multiselect</option>  
 <option value="Free Text">Free Text</option>  
 <option value="Numeric">Numeric</option>  
 <option value="Date">Date</option>  
 </select>  
 )}  
  
 {/\* Answers section \*/}  
 {(question.type === 'Single Select' || question.type === 'Multiselect') && (  
 <div className="answers-list">  
 {question.answers.map(answer => (  
 <AnswerItem answer={answer} questionId={question.ids.id} />  
 ))}  
 </div>  
 )}  
  
 {/\* Save/Cancel buttons if unsaved \*/}  
 {question.isUnsaved && state.builderMode && (  
 <div className="action-buttons">  
 <button onclick={() => dispatch('SAVE\_QUESTION\_IMMEDIATELY', {  
 questionId: question.ids.id  
 })}>  
 💾 Save  
 </button>  
 <button onclick={() => dispatch('CANCEL\_QUESTION\_CHANGES', {  
 questionId: question.ids.id  
 })}>  
 ↶ Cancel  
 </button>  
 </div>  
 )}  
 </div>  
 </div>  
 ))}  
 </div>  
  
 {state.builderMode && (  
 <button onclick={() => dispatch('ADD\_QUESTION\_TO\_SECTION')}>  
 + Add Question  
 </button>  
 )}  
</div>

### 4. Modals and Dialogs

**Purpose**: Display overlay UI for specific operations

#### Relationship Modal

**Rendered When**: state.relationshipPanelOpen === true

**Purpose**: Manage relationships for selected answer

**Structure**: - Modal backdrop - Tabbed interface: - Guidelines tab - Questions tab (triggered questions) - Problems tab (PGI hierarchy) - Barriers tab - Evidence tab - Each tab contains: - Existing relationships list - Typeahead search - Add/delete controls

**Example**:

{state.relationshipPanelOpen && (  
 <div className="modal-backdrop" onclick={() => dispatch('CLOSE\_RELATIONSHIP\_MODAL')}>  
 <div className="modal-content" onclick={(e) => e.stopPropagation()}>  
 <div className="modal-header">  
 <h2>Relationships for Answer</h2>  
 <button onclick={() => dispatch('CLOSE\_RELATIONSHIP\_MODAL')}>  
 <XIcon />  
 </button>  
 </div>  
  
 <div className="tabs">  
 <button  
 className={state.relationshipTab === 'guidelines' ? 'active' : ''}  
 onclick={() => dispatch('SWITCH\_RELATIONSHIP\_TAB', {tab: 'guidelines'})}  
 >  
 Guidelines ({state.guidelineRelationships.length})  
 </button>  
 <button  
 className={state.relationshipTab === 'questions' ? 'active' : ''}  
 onclick={() => dispatch('SWITCH\_RELATIONSHIP\_TAB', {tab: 'questions'})}  
 >  
 Questions ({state.triggeredQuestions.length})  
 </button>  
 {/\* More tabs... \*/}  
 </div>  
  
 <div className="tab-content">  
 {state.relationshipTab === 'guidelines' && (  
 <GuidelinesTab />  
 )}  
 {state.relationshipTab === 'questions' && (  
 <QuestionsTab />  
 )}  
 {/\* More tab content... \*/}  
 </div>  
 </div>  
 </div>  
)}

#### Confirmation Dialog

**Rendered When**: state.confirmationDialogOpen === true

**Purpose**: Confirm destructive operations (delete, publish, etc.)

**Structure**:

{state.confirmationDialogOpen && (  
 <div className="confirmation-dialog-backdrop">  
 <div className="confirmation-dialog">  
 <h3>{state.confirmationDialogTitle}</h3>  
 <p>{state.confirmationDialogMessage}</p>  
  
 <div className="button-group">  
 <button  
 className="btn-danger"  
 onclick={() => {  
 dispatch(state.confirmationDialogAction);  
 dispatch('CLOSE\_CONFIRMATION\_DIALOG');  
 }}  
 >  
 {state.confirmationDialogConfirmText || 'Confirm'}  
 </button>  
 <button  
 className="btn-secondary"  
 onclick={() => dispatch('CLOSE\_CONFIRMATION\_DIALOG')}  
 >  
 Cancel  
 </button>  
 </div>  
 </div>  
 </div>  
)}

#### Text Editor Modal

**Purpose**: Edit long text content (question labels, answer text)

**Rendered When**: state.textEditorModalOpen === true

**Structure**: - Large textarea for editing - Save/Cancel buttons - Character count (optional)

## effects.js - HTTP Effects

**File Location**: src/cadal-careiq-builder/effects.js

This file defines all HTTP effects using ServiceNow’s createHttpEffect utility.

### Effect Definition Pattern

export const MAKE\_[OPERATION]\_REQUEST = createHttpEffect(  
 '[ENDPOINT\_URL]',  
 {  
 method: 'POST',  
 dataParam: 'requestBody',  
 headers: {'Content-Type': 'application/json'},  
 startActionType: '[OPERATION]\_START',  
 successActionType: '[OPERATION]\_SUCCESS',  
 errorActionType: '[OPERATION]\_ERROR'  
 }  
);

### Key Effects

**Assessment Operations**: - MAKE\_ASSESSMENTS\_REQUEST - Fetch all assessments - MAKE\_CREATE\_ASSESSMENT\_REQUEST - Create new assessment - MAKE\_ASSESSMENT\_DETAILS\_REQUEST - Fetch assessment sections - MAKE\_PUBLISH\_ASSESSMENT\_REQUEST - Publish assessment - MAKE\_CREATE\_VERSION\_REQUEST - Create new version

**Section Operations**: - MAKE\_ADD\_SECTION\_REQUEST - Add section - MAKE\_SECTION\_UPDATE\_REQUEST - Update section - MAKE\_DELETE\_SECTION\_REQUEST - Delete section - MAKE\_REORDER\_SECTIONS\_REQUEST - Reorder sections

**Question Operations**: - ADD\_QUESTION\_TO\_SECTION\_API - Add question to section - MAKE\_UPDATE\_QUESTION\_REQUEST - Update question - MAKE\_DELETE\_QUESTION\_REQUEST - Delete question - MAKE\_MOVE\_QUESTION\_REQUEST - Move question to different section

**Answer Operations**: - MAKE\_ADD\_ANSWERS\_TO\_QUESTION\_REQUEST - Add answers to question - MAKE\_UPDATE\_ANSWER\_REQUEST - Update answer - MAKE\_DELETE\_ANSWER\_REQUEST - Delete answer

**Relationship Operations**: - MAKE\_LOAD\_ANSWER\_RELATIONSHIPS\_REQUEST - Load relationships for answer - MAKE\_ADD\_BRANCH\_QUESTION\_REQUEST - Add triggered question - MAKE\_DELETE\_BRANCH\_QUESTION\_REQUEST - Delete triggered question - MAKE\_ADD\_GUIDELINE\_RELATIONSHIP\_REQUEST - Add guideline - MAKE\_DELETE\_GUIDELINE\_RELATIONSHIP\_REQUEST - Delete guideline

**PGI Operations**: - MAKE\_LOAD\_PROBLEM\_GOALS\_REQUEST - Load goals for problem - MAKE\_ADD\_GOAL\_REQUEST - Add goal to problem - MAKE\_LOAD\_GOAL\_INTERVENTIONS\_REQUEST - Load interventions for goal - MAKE\_ADD\_INTERVENTION\_REQUEST - Add intervention to goal

**Typeahead Operations**: - MAKE\_GENERIC\_TYPEAHEAD\_REQUEST - Generic typeahead search - Used for questions, answers, sections, problems, goals, interventions

**Configuration**: - MAKE\_CAREIQ\_CONFIG\_REQUEST - Load CareIQ configuration - MAKE\_USE\_CASE\_CATEGORIES\_REQUEST - Load use case categories

## config-actions.js

**File Location**: src/cadal-careiq-builder/config-actions.js

Exported as configActions object.

### Actions

**LOAD\_CAREIQ\_CONFIG**: - Loads CareIQ platform configuration - Dispatches effect to fetch config - Sets loading state

**CAREIQ\_CONFIG\_FETCH\_SUCCESS**: - Handles successful config load - Stores config in state - Clears loading state

**FETCH\_USE\_CASE\_CATEGORIES**: - Loads use case categories for assessments - Used in new assessment form

## core-actions.js

**File Location**: src/cadal-careiq-builder/core-actions.js

Exported as coreActions object.

### Actions

**CHECK\_MOBILE\_VIEW**: - Detects screen size - Updates state.isMobileView - Triggers responsive layout changes

**TOGGLE\_SECTIONS\_PANEL**: - Shows/hides sections panel - Updates state.sectionsPanelExpanded

**TOGGLE\_QUESTIONS\_PANEL**: - Shows/hides questions panel - Updates state.questionsPanelExpanded

**ADD\_SYSTEM\_MESSAGE**: - Adds message to system messages array - Sets message type (success, error, warning, info) - Auto-dismisses after timeout (optional)

**DISMISS\_SYSTEM\_MESSAGE**: - Removes message from system messages - By message ID or index

**EXPAND\_MESSAGE\_HISTORY**: - Expands message history panel - Shows all past messages

**COLLAPSE\_MESSAGE\_HISTORY**: - Collapses message history panel

## utils.js

**File Location**: src/cadal-careiq-builder/utils.js

Utility functions for data transformation and calculations.

### Key Functions

**groupAssessmentsByMasterId(assessments)**: - Groups assessment versions by master ID - Returns object: {masterId: [version1, version2, ...]}

**paginateAssessments(assessments, page, pageSize)**: - Returns paginated subset of assessments - Calculates total pages

**loadCareIQConfig()**: - Helper for loading CareIQ configuration - May include URL, authentication details

**hasRelationships(answer)**: - Checks if answer has any relationships - Returns boolean

**calculateVisibleQuestions(questions, selectedAnswers, relationshipMap)**: - Determines which questions should be visible - Based on conditional logic and selected answers - Returns filtered question list

**Example Implementation**:

export function calculateVisibleQuestions(questions, selectedAnswers, relationshipMap) {  
 const visibleQuestionIds = new Set();  
  
 // Always include base questions (no triggers)  
 questions.forEach(q => {  
 if (!q.triggeredBy || q.triggeredBy.length === 0) {  
 visibleQuestionIds.add(q.ids.id);  
 }  
 });  
  
 // Check triggered questions  
 questions.forEach(q => {  
 if (q.triggeredBy && q.triggeredBy.length > 0) {  
 // Check if any triggering answer is selected  
 const isTriggered = q.triggeredBy.some(answerId =>  
 selectedAnswers[q.sourceQuestionId]?.includes(answerId)  
 );  
 if (isTriggered) {  
 visibleQuestionIds.add(q.ids.id);  
 }  
 }  
 });  
  
 return questions.filter(q => visibleQuestionIds.has(q.ids.id));  
}

## styles.scss

**File Location**: src/cadal-careiq-builder/styles.scss

Component-scoped SCSS styles.

### Key Style Sections

**Layout**: - .builder-container - Main container - .three-panel-layout - Three-column layout - .sections-panel, .questions-panel - Panel layouts

**Components**: - .assessment-card - Assessment card styling - .question-card - Question card styling - .answer-item - Answer item styling - .modal-backdrop, .modal-content - Modal styles

**States**: - .loading - Loading states - .disabled - Disabled elements - .selected - Selected items - .unsaved - Unsaved changes indicator

**Responsive**: - Media queries for mobile/tablet layouts - Breakpoint: 1400px for panel stacking

## package.json

**File Location**: package.json

### Key Fields

{  
 "name": "cadal-careiq-builder",  
 "version": "0.1.092",  
 "scopeName": "x\_cadal\_careiq\_b\_0",  
 "dependencies": {  
 "@servicenow/ui-core": "^24.1.1",  
 "@servicenow/ui-effect-http": "^24.1.1",  
 "@servicenow/ui-renderer-snabbdom": "^24.1.1",  
 "@servicenow/sass-kit": "^0.3.13",  
 "@babel/runtime": "^7.25.7",  
 "sass": "^1.53.0"  
 }  
}

### Version Management

**Pattern**: 0.1.XXX (increment last digit) - Increment after each change - Track changes in CLAUDE.md or changelog

## now-ui.json

**File Location**: now-ui.json

Component configuration and metadata for ServiceNow.

### Typical Structure

{  
 "components": {  
 "cadal-careiq-builder": {  
 "innerComponents": [],  
 "uiBuilder": {  
 "associatedTypes": ["Global"],  
 "label": "CareIQ Builder",  
 "icon": "form-outline",  
 "description": "Assessment builder for CareIQ platform",  
 "category": "custom"  
 }  
 }  
 }  
}

## Component Registration

At the end of index.js:

createCustomElement('cadal-careiq-builder', {  
 renderer: {type: snabbdom},  
 view,  
 initialState: {  
 // Initial state object  
 },  
 actionHandlers: {  
 [COMPONENT\_BOOTSTRAPPED]: (coeffects) => {  
 const {dispatch} = coeffects;  
 dispatch('LOAD\_CAREIQ\_CONFIG');  
 dispatch('CHECK\_MOBILE\_VIEW');  
 },  
 ...configActions,  
 ...coreActions,  
 // All other action handlers  
 },  
 effects: {  
 ...effects  
 },  
 styles  
});

**Key Parts**: - Component name: cadal-careiq-builder - Renderer: Snabbdom virtual DOM - View function - Initial state - Action handlers (merged from multiple sources) - Effects (imported from effects.js) - Styles (imported from styles.scss)

## Code Organization Best Practices

### What Works Well

1. **Separation of Effects**: effects.js keeps HTTP logic separate
2. **Reusable Components**: Icon and overlay components defined once
3. **Utility Functions**: utils.js provides helper functions
4. **Style Isolation**: SCSS keeps styles scoped to component

### Areas for Improvement

1. **File Size**: 21,166-line file is large; could be split further
2. **Component Extraction**: More view components could be extracted
3. **Action Handler Organization**: Could be split by feature area
4. **State Structure Documentation**: Initial state could use more comments

## Summary

CareIQ Builder’s component structure is organized into:

* **index.js**: Monolithic component with view, state, and actions
* **effects.js**: HTTP effect definitions (50+ endpoints)
* **config-actions.js**: Configuration and initialization
* **core-actions.js**: UI state management
* **utils.js**: Helper functions
* **styles.scss**: Component styling

The architecture prioritizes: - Clear separation of HTTP effects - Reusable UI components - Centralized state management - Scoped styling

Next sections will detail state management, action handlers, and API communication patterns.