CLG Technical Specification

Executive Summary

CLG (Claude Landing Generator) is a multi-brand landing page system built for Access Group. It enables marketing teams to create and manage landing pages across multiple brands (Access Hire Australia, Access Express, etc.) using a visual editor, while maintaining consistent brand theming and equipment enquiry functionality.

The system is built on Next.js 16 with Builder.io as the headless CMS, deployed on Vercel with automatic CI/CD from GitHub.

System Architecture

Frontend Application

The application uses Next.js 16 with the App Router architecture. All pages are server-side rendered for optimal SEO and performance. The application follows a component-based architecture where Builder.io serves as the content management layer and Next.js handles rendering and business logic.

The routing structure uses catch-all routes to handle dynamic URLs. The main route at `app/[[...slug]]/page.tsx` handles all general pages, while `app/equipment/[[...slug]]/page.tsx` handles equipment-specific routes. Both routes fetch content from Builder.io based on the URL path and render it using registered components.

Multi-Brand Theming System

The theming system is the core architectural feature that enables multiple brands to share the same codebase. It works through a layered approach:

Brand Definition Layer: Brands are defined either in Builder.io's Brand data model or locally in `src/lib/themes/brands.ts`. Each brand contains a complete set of design tokens including colour scales, semantic colours, typography, spacing, and asset URLs.

Theme Resolution Layer: When a page loads, the system resolves which brand to use. It first checks if the Builder.io page content has a brand reference field. If found, it fetches the brand data from Builder.io. As a fallback, it checks for a `brandId` URL parameter. If neither exists, it uses the default brand (Access Hire Australia).

CSS Variable Injection Layer: The ThemeProvider component converts the resolved brand theme into CSS custom properties. These variables are injected as inline styles on a wrapper div, making them available to all child components. This approach ensures that brand styles are scoped to the page content and don't leak between different brand pages.

Component Consumption Layer: All brand-aware components read colours and styles using CSS variable references like `var(--color-primary)`. Components can also access the full brand object via the `useTheme()` React hook for assets like logos.

Content Management

Builder.io serves as the headless CMS with two key data models:

Page Model (cc-equipment-category): Stores page content including the visual layout, component configurations, and metadata. Each page can reference a brand, which determines its visual theming.

Brand Model: Stores brand configurations including all colour tokens, typography settings, and asset URLs. Brands are referenced by pages and fetched at render time.

The Builder.io SDK is used for content fetching with ISR (Incremental Static Regeneration) caching. Content is revalidated every 60 seconds to balance freshness with performance.

API Connections

Builder.io Content API

The application connects to Builder.io's Content API for fetching pages and brand data. The public API key is used for read operations. Content is fetched using the `fetchOneEntry` function from the Builder.io SDK, which handles visual editor integration and preview mode.

Endpoint pattern: `https://cdn.builder.io/api/v3/content/{model}?apiKey={key}`

The SDK automatically handles query parameters for filtering by URL path and managing preview/editing states.

Builder.io Write API

For programmatic content updates (such as creating or updating brand entries), the Write API is used with a private API key. This is primarily used by maintenance scripts rather than the application runtime.

Endpoint pattern: `https://builder.io/api/v1/write/{model}`

Contact Request API

The application includes a contact form that submits enquiries to an internal API endpoint at `/api/contact`. This endpoint processes form submissions and forwards them to the Access Group's contact management system.

CLG Visitor Tracking

A custom visitor tracking SDK (`/clg-visitor.js`) is loaded on all pages to track visitor behaviour and personalisation data. This integrates with Firebase for user identification and campaign attribution.

Functionality

Equipment Enquiry System

The equipment enquiry system allows visitors to build a list of equipment they're interested in and submit an enquiry. It consists of several interconnected components:

Enquiry Cart Context: A React context (`EnquiryCartProvider`) manages the cart state across the application. Items are stored in localStorage for persistence across page views.

Equipment Cards: Display individual equipment items with an "Add to Enquiry" button. When clicked, the item is added to the cart context. Cards show visual feedback when an item is already in the cart.

Enquiry Cart Bubble: A floating button that appears when items are in the cart. It shows the item count and animates when new items are added. Clicking it opens the enquiry panel.

Enquiry Cart Panel: A slide-out panel that lists all selected equipment and provides a form for submitting the enquiry. The form collects contact details, industry, branch preference, and project location.

Quick View Modal: A detailed view of equipment that can be opened from equipment cards. It shows specifications, pricing, and allows adding/removing from the enquiry cart.

Personalisation

The Hero component supports URL-based personalisation. Campaign parameters (utm_campaign, utm_source, etc.) can trigger different content variants. The personalisation context is built on page load and can modify headlines, images, and calls-to-action.

Visual Editing

Builder.io's visual editor is fully integrated. Content editors can drag and drop registered components, modify properties, and see live previews. The application detects when it's running in the Builder.io editor and adjusts its behaviour accordingly.

Component Library

The following components are registered with Builder.io and available in the visual editor:

Layout Components: Header (brand-aware with logo switching), Footer (brand-aware with location tabs and contact info)

Hero Components: FigmaHero (full-width hero with personalisation support, configurable CTAs, overlay intensity options)

Content Components: TextBlock (WYSIWYG rich text), ContactForm (quote request form with API integration)

Equipment Components: EquipmentGrid (displays equipment from API with filtering), EquipmentCard (individual equipment display), EquipmentSearch (search and filter interface), QuickViewModal (equipment detail popup)

Landing Page Components: LPHeader, LPFooter, LPHero, LPTrustBadges, LPBenefits, LPTestimonials, LPFaq, LPCtaBanner, LPProductsGrid, LPStickyForm

All components use CSS variables for theming and adapt automatically to the active brand.

CI/CD Process

Repository

The codebase is hosted on GitHub at `github.com/cbertozz-access/clg`. The main branch is protected and serves as the production branch.

Deployment Pipeline

Vercel is connected to the GitHub repository and automatically deploys on every push to the main branch. The deployment process follows these steps:

1. Push to main branch triggers Vercel webhook
2. Vercel pulls the latest code

3. Dependencies are installed via npm
4. Next.js build runs (`next build`)
5. TypeScript compilation and type checking
6. Static pages are pre-rendered
7. Serverless functions are created for dynamic routes
8. Assets are deployed to Vercel's edge network
9. DNS is updated to point to the new deployment

Build Configuration

The build uses Turbopack for faster compilation. Build caching is enabled by default but can be bypassed with `--force` flag when needed to clear stale data.

Environment Variables

Environment variables are managed in Vercel's dashboard and locally via `.env.local` :

- `NEXT_PUBLIC_BUILDER_API_KEY` : Public key for Builder.io content fetching
- `BUILDER_PRIVATE_KEY` : Private key for Builder.io Write API (not exposed to client)

Manual Deployment

For immediate deployments or cache-busting, the Vercel CLI can be used:

- Standard deploy: `vercel --prod`
- Force deploy (clear cache): `vercel --prod --force`

Preview Deployments

Every pull request automatically receives a preview deployment on a unique URL. This allows testing changes before merging to main.

Brand Configuration

Access Hire Australia (Default Brand)

The primary brand uses a red colour scheme with white header and red footer. Typography uses Lato for headings and Roboto for body text. This brand serves as the default when no brand is specified.

Access Express

A secondary brand using a navy/black colour scheme for the primary colour, header, and footer. Orange is used as the secondary colour for special call-to-action buttons. Blue serves as the accent colour for links and highlights. Typography uses Montserrat for headings and Open Sans for body text.

Adding New Brands

New brands can be added either through Builder.io's Brand data model (recommended for marketing team access) or by adding entries to the local `brands.ts` file (for developer-managed brands). The system automatically makes new brands available for selection on pages.

File Structure

```
src/
├── app/                          # Next.js App Router pages
│   ├── [[...slug]]/             # Catch-all route for general pages
│   ├── equipment/[[...slug]]/   # Equipment-specific routes
│   ├── api/contact/             # Contact form API endpoint
│   └── layout.tsx               # Root layout with providers
├── components/
│   ├── builder/                 # Builder.io registered components
│   │   ├── equipment/          # Equipment-related components
│   │   ├── figma/              # Design system components
│   │   └── lp/                 # Landing page components
│   ├── layout/                  # Header, Footer
│   ├── ThemeProvider.tsx        # Brand theming context
│   └── EnquiryCartBubble.tsx   # Floating cart button
├── lib/
│   ├── themes/                  # Theming system
│   │   ├── types.ts            # TypeScript interfaces
│   │   ├── brands.ts           # Brand definitions
│   │   └── css-variables.ts    # CSS variable generation
│   ├── builder/                 # Builder.io integration
│   │   └── brand-model.ts      # Brand fetching and transformation
│   ├── enquiry-cart.tsx        # Cart context and hooks
│   └── builder-register.ts     # Component registration
└── styles/
    └── globals.css              # Global styles and CSS variable defaults
```

Performance Considerations

The application is optimised for performance through several mechanisms:

- Server-side rendering ensures fast initial page loads and good SEO
- ISR caching reduces Builder.io API calls while maintaining content freshness
- CSS variables enable theming without runtime JavaScript overhead
- Images are optimised through Next.js Image component
- Fonts are loaded via Next.js font optimisation with display swap
- Components are code-split automatically by Next.js

Security

- Builder.io private key is server-side only, never exposed to clients
- Contact form submissions are validated server-side
- No sensitive data is stored in localStorage (only equipment IDs for cart)
- All external resources are loaded over HTTPS
- Content Security Policy headers are configured via Vercel

Monitoring and Debugging

A debug panel (`VisitorDebugPanel`) is available in development and can be toggled to show:

- Current visitor identification status

- Active personalisation context
- Builder.io SDK status
- Current brand information

This panel is hidden in production but can be enabled for troubleshooting.