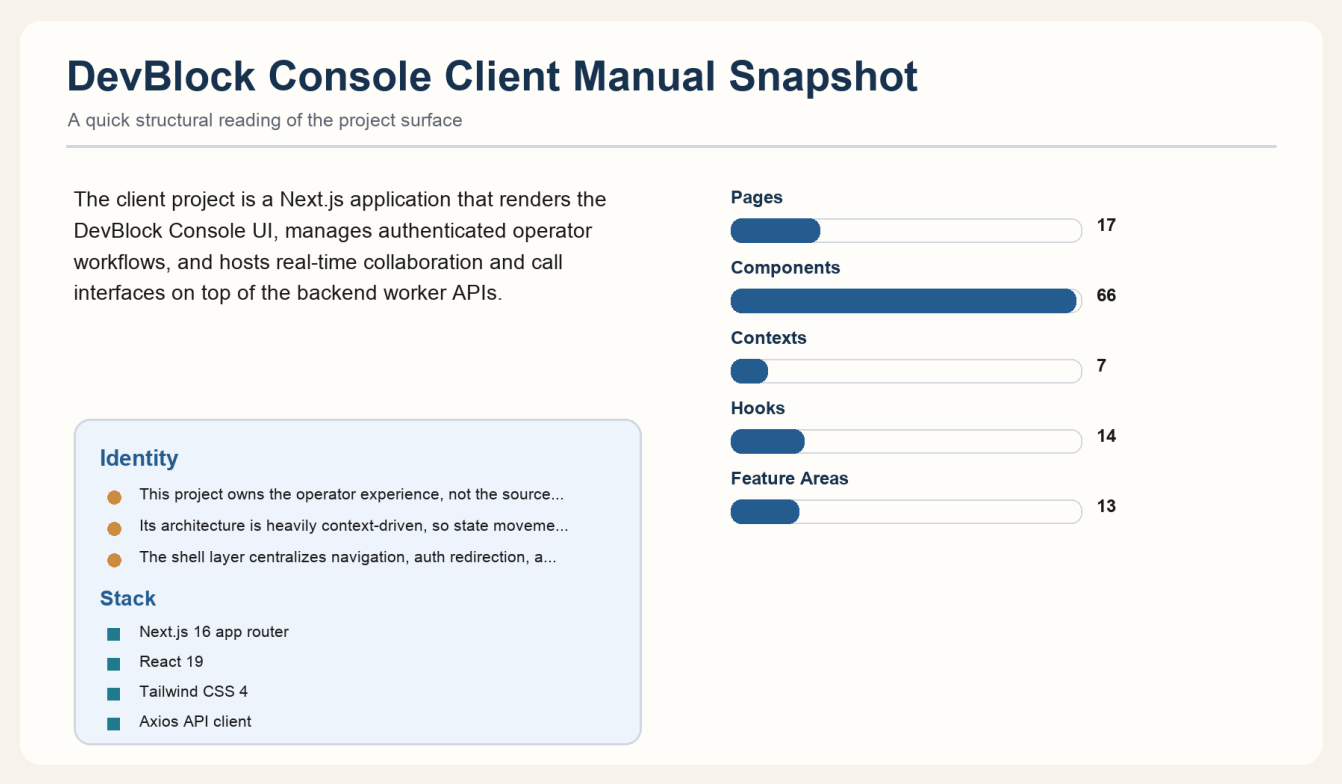
DevBlock Console Client Manual

Frontend operations, layout, contexts, route surfaces, and browser behavior

The client project is a Next.js application that renders the DevBlock Console UI, manages authenticated operator workflows, and hosts real-time collaboration and call interfaces on top of the backend worker APIs.



DevBlock Console Client Manual visual snapshot

What this project is for

The client project is a Next.js application that renders the DevBlock Console UI, manages authenticated operator workflows, and hosts real-time collaboration and call interfaces on top of the backend worker APIs.

> **Primary audience: Use this when changing the UI shell, route behavior, contexts, browser-side API integration, or operator-facing feature pages.**

How to identify it quickly

- This project owns the operator experience, not the source of truth for business state.
- Its architecture is heavily context-driven, so state movement matters as much as JSX structure.
- The shell layer centralizes navigation, auth redirection, and global call overlays.
- Feature pages mirror backend domains closely: dashboard, projects, files, logs, clients, chat, settings.

Technology stack

- Next.js 16 app router
- React 19
- Tailwind CSS 4
- Axios API client
- Context-heavy browser state
- WebRTC call UI

Structural counts

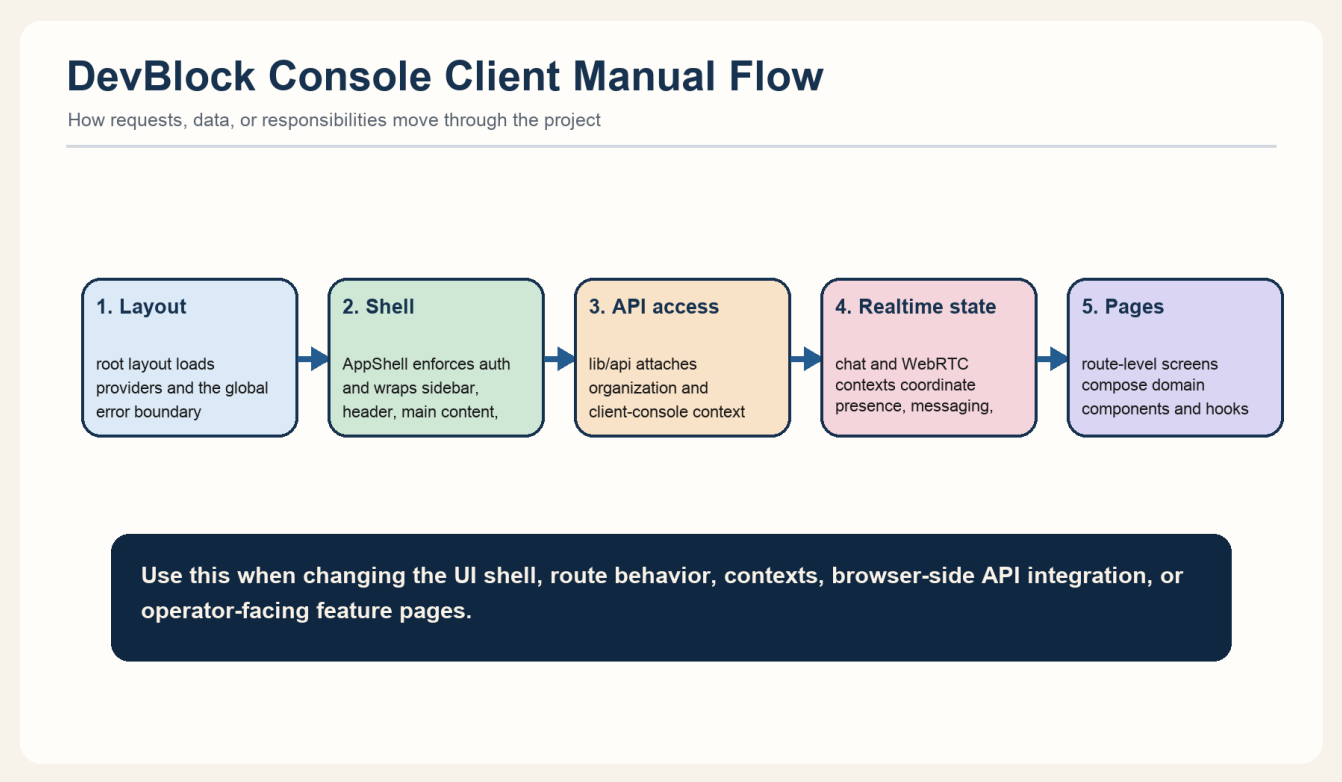
Metric	Count
Pages	17
Components	66
Contexts	7
Hooks	14
Feature Areas	13

Architecture reading

- The client is best understood as a shell-plus-context system rather than a set of isolated pages.
- Global UX features such as auth-expiry handling and active calls are mounted above page level.
- The folder layout under `components/` reveals feature ownership more clearly than the generic README does.
- The UI shares vocabulary with the backend routes, which makes end-to-end reasoning faster.

Core flow

This sequence is the shortest useful mental model for how the project behaves at runtime or during operator use.



DevBlock Console Client Manual responsibility or data flow

Responsibility matrix

DevBlock Console Client Manual Responsibility Matrix

What this project owns versus what it coordinates

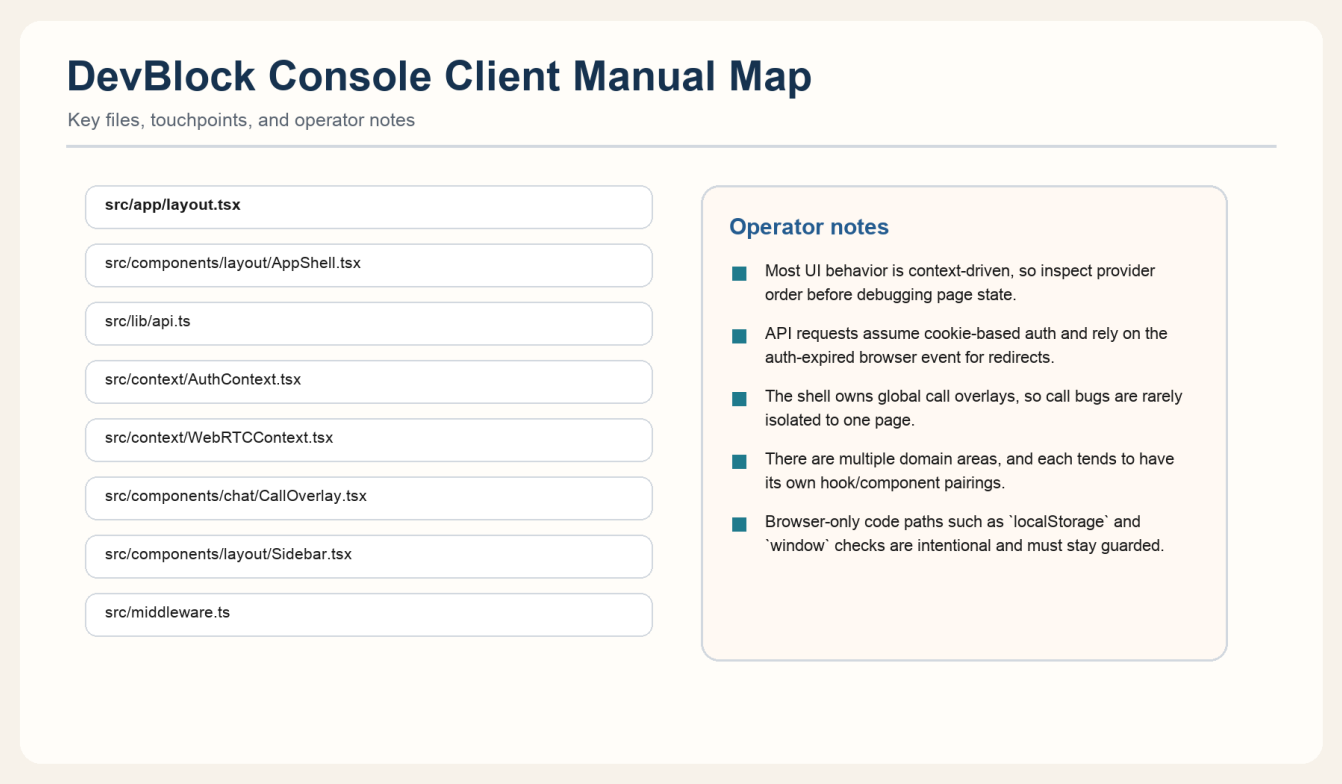
Area	Owns	Coordinates with	Failure mode
Product role	This project owns the operator experience, not the source of truth for business state.	Its architecture is heavily context-driven, so state movement matters as much as JSX structure.	Context coupling can create hidden side effects across screens.
Main inputs	Layout: root layout loads providers and the global error boundary	Shell: AppShell enforces auth and wraps sidebar, header, main content, and call UX	The default client README is generic and understates the actual product complexity.
Change surface	Start from the visible feature area, then trace upward into the owning context and shell if the behavior	Inspect `lib/api.ts` before assuming a frontend bug is local; request shaping and redirect behavior live	Browser-only assumptions in auth and localStorage flows must stay guarded for SSR and hydration
Operator posture	Prefer targeted lint and type checks for touched files because the broader frontend baseline can be	Any auth or redirect change should be tested through the shell, not just with unit-level reasoning.	Most UI behavior is context-driven, so inspect provider order before debugging page state.

DevBlock Console Client Manual ownership matrix

Key files to read first

If you are new to this project, read these files in order before making broad edits.

- src/app/layout.tsx
- src/components/layout/AppShell.tsx
- src/lib/api.ts
- src/context/AuthContext.tsx
- src/context/WebRTCContext.tsx
- src/components/chat/CallOverlay.tsx
- src/components/layout/Sidebar.tsx
- src/middleware.ts



DevBlock Console Client Manual orientation map

Change workflow

- Start from the visible feature area, then trace upward into the owning context and shell if the behavior feels global.
- Inspect `lib/api.ts` before assuming a frontend bug is local; request shaping and redirect behavior live there.
- Use route files to identify screen entrypoints, then inspect hooks and context providers for state movement.
- When changing chat or call behavior, treat the overlay and WebRTC context as part of the same system.

Operator notes

- Most UI behavior is context-driven, so inspect provider order before debugging page state.
- API requests assume cookie-based auth and rely on the auth-expired browser event for redirects.
- The shell owns global call overlays, so call bugs are rarely isolated to one page.
- There are multiple domain areas, and each tends to have its own hook/component pairings.
- Browser-only code paths such as `localStorage` and `window` checks are intentional and must stay guarded.

Validation posture

- Prefer targeted lint and type checks for touched files because the broader frontend baseline can be noisy.
- Any auth or redirect change should be tested through the shell, not just with unit-level reasoning.
- Visual changes in shared layout surfaces should be checked on at least one representative feature page.

Current risks or watchpoints

- Context coupling can create hidden side effects across screens.
- The default client README is generic and understates the actual product complexity.
- Browser-only assumptions in auth and localStorage flows must stay guarded for SSR and hydration safety.

Glossary

Term	Meaning
App shell	The persistent layout layer that wraps navigation, header, and global UI behaviors.
Context-driven	A project style where shared state and actions live in React context providers.
Auth-expired event	A browser event used to redirect operators cleanly after session loss.