



# Enterprise Operations & Intelligence Platform

Engineering Case Study

A confidential, multi-module business application built with React, JavaScript, Supabase, PostgreSQL, serverless functions, image workflows, AI-assisted recognition, dashboards, and operational automation.

**Shannon Curtis**

Frontend Software Developer

[shannondcurtis.com](https://shannondcurtis.com)



# 1. Project Overview

The objective was to design and build a production-minded business application capable of supporting complex operational workflows without exposing users to unnecessary technical complexity.

The platform combines secure authentication, cloud-connected records, image capture, search, filtering, reporting, status-driven workflows, automation, and AI-assisted data extraction inside a consistent responsive interface.

## Confidentiality note

Product names, customer data, proprietary business rules, and unreleased implementation details are intentionally omitted. This case study focuses on the engineering problems solved, the architecture used, and the frontend product decisions made.

## Primary responsibilities

- Translate evolving business requirements into usable product workflows.
- Design reusable React component patterns and shared application structure.
- Connect frontend interfaces to authentication, storage, relational data, and serverless services.
- Build responsive dashboards, data tables, workspaces, scanning tools, and administrative flows.
- Debug integration issues across browser, cloud, API, database, and deployment layers.

| Multi-module          | Cloud-connected     | AI-assisted                |
|-----------------------|---------------------|----------------------------|
| Operational workflows | Auth, data, storage | Recognition and automation |

## 2. Engineering Challenge

The application needed to support many distinct workflows while still feeling like one coherent product. Each module required different controls, data structures, and user decisions, yet navigation, status language, feedback, and visual hierarchy needed to remain predictable.

### Key constraints

- Complex records with images, identifiers, history, pricing, status, and metadata.
- Multi-step intake and review processes.
- Secure browser access across desktop and mobile form factors.
- Integration with external services and serverless functions.
- Clear recovery from incomplete recognition, missing data, API errors, and user corrections.
- Confidential business logic that should not be exposed in the browser.

### Product principle

**Complexity should live in the system, not in the user experience. The interface was designed to reveal the right information at the right stage and to provide review paths whenever automation was uncertain.**

### Why this mattered

Operational software is often judged less by visual novelty than by whether users can trust it under real working conditions. That required consistent states, visible feedback, recoverable actions, and predictable transitions from one stage to the next.

### 3. Frontend and Cloud Architecture

The application uses a modular frontend architecture supported by cloud services. React components manage interaction and presentation, while service modules isolate database operations, external APIs, storage, recognition, pricing, and business rules.

|                    |                                                                                    |
|--------------------|------------------------------------------------------------------------------------|
| Presentation Layer | React pages, components, responsive workspaces, dashboards, forms, tables          |
| Application Layer  | State, workflow orchestration, validation, navigation, review states               |
| Service Layer      | Inventory, pricing, reporting, search, scanning, recognition, marketplace services |
| Cloud Layer        | Supabase authentication, PostgreSQL, storage, Edge Functions                       |
| External Services  | AI recognition, market data, third-party APIs, email and deployment services       |

#### Architecture decisions

- Keep API credentials and privileged logic in serverless functions rather than frontend code.
- Use service modules to prevent components from becoming tightly coupled to database implementation.
- Normalize API responses before they reach UI components.
- Preserve clear fallback behavior when external providers return partial or unavailable data.
- Maintain reusable UI patterns for tabs, status badges, metrics, detail panels, and actions.

## 4. AI-Assisted Recognition Workflow

One of the most technically demanding areas was turning image-based recognition into a usable business workflow. The goal was not simply to display AI output, but to convert uncertain recognition into structured, reviewable data.

### Workflow

|   |           |                                                        |
|---|-----------|--------------------------------------------------------|
| 1 | Capture   | Front and back images are collected                    |
| 2 | Recognize | AI extracts identifiers and descriptive fields         |
| 3 | Normalize | Results are cleaned into consistent application fields |
| 4 | Match     | Catalog and internal records are compared              |
| 5 | Review    | Low-confidence or conflicting results are surfaced     |
| 6 | Continue  | Approved data enters pricing and inventory workflows   |

### Important design decision

**AI output was treated as a recommendation, not unquestionable truth. Confidence, missing fields, conflicts, and provider availability are translated into clear review states so users remain in control.**

### Engineering outcomes

- Structured parsing of recognition results.
- Front/back image coordination.
- Automatic continuation for high-confidence results.
- Manual review and correction for uncertain fields.
- Fallback handling when recognition or market providers are unavailable.

## 5. Data-Rich Workspaces and Operational UX

The platform contains dense operational data, but the interface separates it into focused workspaces. Users can move between details, images, identification, labels, pricing, and history without losing the context of the selected record.

### Key frontend patterns

- Searchable and sortable inventory tables.
- Side-panel record workspaces.
- Tabbed detail views.
- Status badges and operational color cues.
- Metric cards and trend visualizations.
- Upload and preview states for images.
- Print-ready label and QR workflows.
- Toast notifications and visible progress states.

### Responsive strategy

Layouts were designed for large desktop workspaces while preserving usable behavior on smaller screens. Dense tables, multi-column forms, panels, and controls were adjusted through responsive CSS rather than creating unrelated desktop and mobile products.

#### Usability objective

## 6. Integration and Debugging Work

The project required repeated integration across browser code, cloud services, database policies, serverless functions, and third-party APIs. The most valuable engineering work often occurred at the boundaries between systems.

### Examples of problems solved

- Authentication and environment-variable configuration.
- Row-level security and database insert requirements.
- Image upload and record synchronization.
- Camera permission and MediaDevices behavior.
- CORS and Edge Function deployment.
- OAuth client authentication and secure secret storage.
- Third-party search query construction and response normalization.
- Progressive fallback searches when exact queries returned no results.

### Debugging approach

**I used browser DevTools, network traces, targeted console instrumentation, server logs, provider-level isolation, and incremental testing to identify the exact layer where a workflow stopped. Temporary diagnostics were removed after confirming the fix.**

### Result

The final integrations moved from frontend event to cloud function to external provider and back into a normalized business result that could be stored and presented to the user.

## 7. Outcomes and Lessons

This build demonstrates the ability to move beyond isolated webpage development and engineer complete frontend product workflows that coordinate state, data, cloud services, external APIs, automation, and operational decisions.

### Demonstrated capabilities

- React application architecture and reusable UI systems.
- Supabase authentication, PostgreSQL data, storage, and Edge Functions.
- AI-assisted image workflows and confidence-based review states.
- Data tables, dashboards, reporting, pricing, labels, and operational status management.
- Secure API integration and production troubleshooting.
- Responsive product design for desktop and mobile use.

### What I learned

**Reliable business software depends on clear states, graceful failure, reviewable automation, strong data boundaries, and interfaces that make complex workflows feel straightforward.**

### Next engineering focus

Continue improving confidence scoring, provider quality, automated validation, test coverage, performance, accessibility, and documentation while preserving a clean operator experience.

### Contact

**Shannon Curtis**

Frontend Software Developer

admin@shannondcurtis.com

shannondcurtis.com