PROJECT CASE STUDY

PropLead India

A Full-Stack AI-Powered Real Estate Marketplace

A three-sided marketplace connecting property buyers, sellers/agencies, and a platform owner — built end-to-end with a modern web stack and AI woven into both the product and the engineering process itself.

317	~33,600	3
TypeScript / TSX files	lines of application code	distinct user consoles

propleadindia.com

Prepared by **Acetrum** · https://acetrum.com

Executive Summary

PropLead India is a production real estate marketplace serving the Indian property market: houses, apartments, plots, shops and commercial/warehousing listings across multiple cities. The platform is built as a three-sided system — a public buyer-facing site, a seller console for agencies and individual dealers, and an owner/admin dashboard for platform operations — with AI embedded throughout rather than bolted on as an afterthought.

The engineering approach paired a modern, type-safe Next.js stack with an AI-assisted development workflow (Claude Code), enabling a small team to ship, verify, and iterate on production features — from a multi-seller lead marketplace to a full email-based seller authentication system — at a pace that would typically require a much larger team.

What this document covers

- The project brief and the problem the platform solves
- How the UI and design system were established and iterated
- The complete technology stack and system architecture
- AI usage — both inside the product and in how the product was built
- The feature set delivered, organised by user type
- SEO, content, security and the day-to-day development approach

Project Brief

The brief was to build a real estate marketplace that goes beyond a listings directory: a platform that actively helps sellers convert enquiries into sales, gives the platform owner real operational control over lead monetisation, and gives buyers a fast, trustworthy way to find verified properties — all while being genuinely SEO-competitive against established portals.

Core requirements

- **Buyer discovery:** searchable/filterable listings across property types (house, apartment, plot, shop) with map views, EMI/loan-eligibility tools, comparisons, and saved favourites.
- **Seller self-service:** agencies and dealers register, get verified, list and manage their own inventory, and see which of their leads are worth calling first.
- **Lead monetisation for the owner:** the ability to sell a single buyer enquiry to more than one seller — by city, by manual assignment, or by AI suggestion — rather than a rigid one-lead-one-seller model.
- **Trust and verification:** RERA fields, KYC document verification for sellers, and a moderated review/comments system so buyers aren't navigating an unverified free-for-all.
- **Content-led SEO:** a blog engine capable of rich, data-backed articles (not template filler), structured data, canonical URLs, and human-readable slugs.

- **Operational control:** an owner dashboard covering approvals, plans/pricing, revenue, site content, and AI model configuration — without needing a developer for routine changes.

Design & UI Process

The UI was not designed once and frozen — it was finalised through continuous, evidence-based iteration: build a component, load it in a live browser preview, inspect the rendered DOM and console for real issues, and adjust before moving on. That loop (build → preview → verify → refine) is the backbone of how every screen in the product reached its final form.

Design system foundations

- **Token-driven theming:** a small set of semantic CSS custom properties (background, surface, ink, muted, line, and a brand green in default/dark/soft variants) feed directly into Tailwind CSS v4's `@theme` layer, so the entire UI stays consistent by construction rather than by convention.
- **Component-first UI:** shared building blocks (property cards, dealer cards, badges, modals, tabs, pagination) are reused across the buyer site, seller console and owner dashboard, keeping the three very different consoles visually and behaviourally consistent.
- **Mobile-first, responsive by default:** layouts are built for small screens first and progressively enhanced for desktop (e.g. list/grid toggle views, sticky in-page navigation, collapsible mega-menus).
- **Iterative refinement from real usage:** UI decisions were revisited against actual feedback — repositioning a mega-menu's "View all" link after the first placement felt too spread out, enlarging listing imagery for better visual impact, making entire card surfaces clickable rather than relying on small text links, and removing badge placements that left awkward empty space when a listing had nothing to show.
- **Accessible by habit:** semantic roles and keyboard interaction (Enter-to-activate, focusable custom controls) were added to custom-built interactive elements such as clickable card surfaces, not just default HTML links and buttons.

Technology Stack

Every layer of the stack was chosen for the same reason: type-safety and directness, so the team (human and AI-assisted) can move fast without the codebase becoming fragile.

Layer	Technology	Why
Framework	Next.js 16 (App Router) + React 19	Server Components by default, file-based routing, built-in image optimisation and edge middleware
Language	TypeScript, end to end	Compile-time safety across UI, API routes and database queries — the same types flow from schema to screen
Styling	Tailwind CSS v4	Utility-first styling driven by a small set of semantic design tokens (see Design System)
Database	PostgreSQL + Drizzle ORM	Type-safe schema and queries; migrations tracked and applied deliberately rather than auto-synced in production
Auth	jose (JWT) + bcryptjs	Stateless signed sessions for buyers/owner (phone+OTP) and a separate email+password flow with bcrypt hashing for sellers
AI orchestration	LangChain + LangGraph, multi-provider	Anthropic, OpenAI, Google Gemini and Groq all supported behind one router — the owner can choose per feature
Email	Resend (SMTP-compatible fallback)	Transactional email for verification and password flows, with a self-hosted SMTP path also supported
Rich content	TipTap editor + sanitize-html	WYSIWYG editing for the blog CMS, with server-side HTML sanitisation before anything is rendered back to visitors
Maps	Leaflet + Google Maps JS API	Interactive plot/property maps and geocoding
Validation	Zod	Every API route validates its input against a schema before it touches the database
Charts	Chart.js	Analytics dashboards for sellers and the owner

System Architecture

The application is a single Next.js codebase split cleanly along two axes: Server Components for data-heavy, SEO-relevant pages (property/blog detail pages, listing pages) and Client Components only where real interactivity is needed (filters, forms, the AI chat widgets). API routes handle mutations and are independently authenticated and Zod-validated — the middleware layer never assumes a route is safe just because a page around it is.

- **Edge middleware** handles two jobs at the request boundary: canonical-domain redirects (www → apex) and role-based route protection for the seller and owner consoles, before a single line of page code runs.
- **Server-first data fetching:** pages fetch directly from the database via typed server functions rather than round-tripping through an internal API, keeping data fresh and reducing request waterfalls.
- **Database as source of truth:** nothing in the UI is templated from static seed data in production — every listing, lead, seller and blog post is a real row, and the codebase's own working rule throughout development was "real data only": no fabricated ratings, no invented statistics, no placeholder trust signals.
- **Background/administrative scripts** handle one-off operations (migrations, content seeding, data backfills) outside the request path, verified directly against the database before and after each change.

AI Integration

AI is used in two distinct ways on this project: as a product capability buyers and sellers interact with directly, and as the engineering method used to build the platform itself.

AI inside the product

Feature	What it does
AI Lead Router	Scores every incoming buyer lead Hot / Warm / Cold and drafts a ready-to-send reply, shown to both the seller and the owner
WhatsApp draft integration	The AI-drafted reply opens directly in a pre-filled WhatsApp link, closing the gap between "AI wrote a reply" and "it was actually sent"
RAG-powered search	Natural-language property search (e.g. "3BHK under 1 crore near a metro in Noida") using an LLM planning step over embeddings-backed retrieval
AI concierge	A conversational assistant that can look up real listings and answer buyer questions using live catalogue data, not a static script
Owner AI analyst	A chat-driven assistant for the owner dashboard that can query platform data conversationally for operational questions
Content moderation	AI-assisted review of buyer comments/reviews before they go live
Multi-provider model routing	The owner can select which AI provider/model powers each feature independently, from Anthropic, OpenAI, Google Gemini or Groq

AI in how the product was built

The platform was built using Claude Code as an AI-assisted engineering partner throughout the development lifecycle — not just for boilerplate, but for full feature delivery: schema design and migrations, API routes with authentication and validation, UI components, and live browser-based verification of every change before it was considered done. Every non-trivial change followed the same discipline regardless of who (or what) wrote it:

- Research the actual current behaviour and data before changing anything — no assumptions
- Implement the smallest correct change, matching existing code conventions exactly
- Verify with real tooling: TypeScript compiler, linter, and a live browser preview against actual rendered output and network requests — not just "it looks right"
- Confirm against the real database, not seed/mock data, wherever the feature touched data

Feature Set by User Type

Buyers (public site)

- Search and filter houses, apartments, plots and shops by city, locality, type, budget and configuration, with grid and list views
- Interactive plot and property maps (Leaflet), nearby-infrastructure chips (metro, school, hospital, market)
- EMI calculator and home-loan eligibility tools
- Property comparison, favourites, and reservation (refundable token hold) flows
- Visit scheduling and WhatsApp / call enquiry with masked seller phone numbers until contact
- Verified-seller badges, RERA details, and a moderated reviews/comments system
- A full blog/content engine — rich SVG charts, comparison tables, TL;DR summaries and social sharing built into every article, not just a plain text CMS
- AI concierge chat and natural-language property search

Sellers (agency/dealer console)

- Self-service registration with email verification, and email+password authentication (migrated from an earlier phone-OTP-only model)
- Listing management: create, edit, duplicate, and track approval status per property
- Their own AI lead score and draft reply per enquiry, with one-click WhatsApp send
- City-based coverage plans, free-tier limits, and paid promotion requests
- KYC/verification document upload and status tracking
- Seller analytics: response time, conversion, revenue
- Password reset (self-service, and owner-triggered when needed)

Owner (platform admin)

- Full property/plot/seller approval workflows
- **Multi-seller lead distribution:** sell a single buyer lead to more than one seller, by manual pick or AI-suggested match, with per-sale pricing and a full sales history per lead
- Revenue tracking across listing plans and lead sales
- Site content editor (homepage copy, categories, hero content) without a code deploy
- Blog CMS with a rich content editor and publish/approve workflow
- AI settings: choose the model/provider behind each AI feature independently
- Plans, city-coverage, and pricing management
- Reservation, visit-request and review moderation queues

SEO & Content Engine

- Human-readable, auto-generated URL slugs for every listing (e.g. `/properties/3bhk-apartment-sector-105-noida-a1b2c3`) with permanent redirects from old/bare-ID links so nothing that's already indexed ever 404s
- JSON-LD structured data on listings (Product/Residence + BreadcrumbList) and blog posts (BlogPosting), so search engines can read price, location and article metadata directly
- Canonical-domain enforcement (www → apex) at the middleware layer, fixing duplicate-URL indexing before it starts
- A dynamic sitemap generated from live, approved listings and published posts — never stale
- Blog content built for genuine engagement: real sourced statistics, comparison tables, SVG charts, TL;DR summaries and FAQ blocks, not thin filler content

Security & Data Handling

- Role-based access control across three distinct roles (buyer, seller, owner), enforced both at the edge middleware layer and inside each API route — a page being protected is never assumed to protect the API routes it calls
- Passwords hashed with bcrypt; sessions are signed, stateless JWTs (via jose) with httpOnly cookies
- Every API route validates its input against a Zod schema before touching the database
- User-generated HTML (blog content, comments) is sanitised server-side before rendering, with an explicit allow-list of tags and attributes
- Sensitive contact details (phone numbers) are masked in seller/dealer views until a buyer actively engages
- A documented, deliberate database-migration process (generate → review → apply → register) rather than auto-sync, so schema changes in production are always intentional and reviewable

Development Approach

Delivery followed a tight, evidence-based loop on every change, regardless of size:

- **Understand before changing** — read the real current implementation and real current data before writing a line of code
- **Match existing conventions** — new code reads like the surrounding code: same patterns, same naming, same component structure
- **Verify, don't assume** — every change was checked with the TypeScript compiler, the linter, and a live browser preview against actual rendered output, console errors and network requests before being considered complete
- **Real data only** — no fabricated ratings, statistics or trust signals anywhere in the product; where real data wasn't available (e.g. a listing photo), that gap was surfaced explicitly rather than papered over with a plausible-looking fake

- **Ship in small, reviewable increments** — features were built, verified and committed as discrete units rather than large unreviewable batches

Why This Approach Works

PropLead India demonstrates a development model where a modern, type-safe stack and an AI-assisted engineering workflow compound each other: the stack's strong typing and clear conventions make AI-assisted changes safer and easier to verify, and the AI-assisted workflow's discipline of research-first, small-increment, always-verified delivery keeps a codebase this size (300+ files, three distinct user-facing consoles, a multi-provider AI layer) coherent rather than accumulating the inconsistency that usually comes with fast iteration.

The result is a platform that doesn't just list properties — it actively routes and monetises leads, gives sellers AI-assisted tools to close faster, gives the owner real operational control without needing a developer for every change, and is built on an SEO foundation designed to compete with established portals rather than just exist alongside them.

PropLead India was designed and built by **Acetrum** (https://acetrum.com/), covering product strategy, UI/UX, full-stack engineering and the AI systems described in this document.

This case study describes the PropLead India platform (propleadindia.com) as of the engagement covered. Figures for code size and file counts are measured directly from the project repository. This document is intended for portfolio/pitch use by the team that built the platform.