

National Buying & Selling Marketplace - Final Tech Stack & Project Plan

Comprehensive architecture, tech stack, and backend/frontend blueprint Generated for: Parin Date: 2025-11-06

1. High-Level Architecture A split architecture with a Next.js frontend and an Express backend using REST APIs. Frontend (Next.js + GSAP + Tailwind) -> REST API (Express) -> PostgreSQL (Prisma) Backend includes Redis for caching, Cloudinary for storage, and Stripe/Razorpay for payments. Both layers are independently deployable for scalability and reliability.

2. Frontend Stack - Framework: Next.js 15 (App Router) with TypeScript - Styling: TailwindCSS + Custom CSS Modules (for GSAP elements) - UI: Optional shadcn/ui components - Animations: GSAP + ScrollTrigger (plus optional Framer Motion for micro-animations) - State: Zustand (UI state) + React Query (data fetching/caching) - Forms & Validation: React Hook Form + Zod - Auth Handling: JWT tokens with secure storage via refresh tokens - Deployment: Vercel (global CDN + SSR support)

3. Backend Stack - Runtime: Node.js (v20+) with TypeScript - Framework: Express.js (modular architecture) - ORM: Prisma (PostgreSQL) - Database: PostgreSQL (Neon.tech or Supabase) - Cache: Redis (Upstash or managed Redis) - File Storage: Cloudinary (image hosting + CDN) - Auth: JWT (access & refresh tokens) - Payments: Stripe and optional Razorpay for India; webhooks handled by backend - Real-time: Socket.IO + Redis adapter for scalable chat - Validation: Zod or Joi - Security: Helmet, express-rate-limit, CORS, HTTPS enforcement - Logging: Winston + Morgan; Monitoring: Sentry / Logtail - Testing: Jest + Supertest for unit and integration tests - Deployment: Render / Railway / AWS ECS for backend containers

4. Database Layer (PostgreSQL + Prisma) Entities/Models: - User (id, name, email, role, profile, createdAt) - Product (id, title, description, price, images, category, sellerId, status, createdAt) - Category - Chat / Conversation - Message - Order (payment tracking) - Wishlist - Review - Notification Indexes and considerations: - Unique index on users.email - Full-text search on products.title & description - Index on messages.chatId for performance

5. API Design (REST) Base URL: <https://api.yourmarketplace.com/api/v1> - Auth: POST /auth/register, /auth/login, /auth/refresh - Users: GET /users/:id, PUT /users/:id - Products: GET /products (filters, pagination), POST /products, GET /products/:id, PUT/DELETE /products/:id - Chat: GET /chats, POST /chats/:id/message - Payments: POST /payments/create-session, POST /payments/webhook - Admin: GET /admin/dashboard, POST /admin/products/:id/approve

Responses follow a consistent format: { "status": "success" | "error", "data": {...}, "message": "..." }

6. Infrastructure & Deployment - Frontend: Vercel (Next.js) for CDN and SSR - Backend: Render / Railway / AWS ECS for scalable services - DB: Neon.tech (serverless Postgres) or Supabase - Redis: Upstash (serverless Redis) for cache & pub/sub - Storage: Cloudinary for images & CDN - CI/CD: GitHub Actions for test and deploy pipelines - Monitoring: Sentry for errors, Logtail for logging, Prometheus/Grafana optional for metrics

7. Security & Optimization - Enforce HTTPS and CORS origin whitelists - Use Helmet for secure headers - Implement rate limiting, especially for auth & payment routes - Hash passwords with bcrypt and rotate secrets - Use Redis caching for read-heavy endpoints and DB connection pooling for Prisma - Image optimization with Cloudinary; lazy load images; enable gzip/brotli compression

8. Scalability & Future Proofing - Backend autoscaling and load balancing - DB read replicas, partitioning, and connection pooling for high traffic - Redis for caching and queueing with BullMQ for background jobs -

Option to split services into microservices (chat, payments, listings) - Add GraphQL gateway later if multi-client querying becomes complex - Mobile app integration using same REST APIs (React Native / Flutter)

9. Developer Workflow & Repo Structure Monorepo suggestion: /frontend (Next.js app) /backend (Express API) /infra (IaC, scripts) /docs (API docs, onboarding) CI/CD with GitHub Actions: - Lint & tests on PRs - Deploy frontend to Vercel, backend to Render on main branch merge Environment variables (.env):
DATABASE_URL= CLOUDINARY_URL= REDIS_URL= STRIPE_SECRET_KEY= JWT_SECRET=
FRONTEND_URL=

10. Final Recommendations & Next Steps - Use PostgreSQL (Neon.tech) + Prisma as primary DB - Implement REST API on Express with clear versioning (/api/v1) - Use Socket.IO with Redis adapter for scalable real-time chat - Host frontend on Vercel and backend on Render or Railway - Start with an MVP (auth, product CRUD, search, wishlist, basic chat, admin moderation) - Iterate: payment integration, advanced search (Algolia), analytics, and scaling measures