Software Development Life Cycle (SDLC)

Leather Shoes E-Commerce Platform

Project Name	Leather Shoes E-Commerce Platform
Tech Stack	MERN (MongoDB, Express.js, React/Next.js, Node.js)
Frontend Framework	Next.js 14 (App Router)
Backend	Node.js + Express.js
Database	MongoDB
Object Storage	MinIO (self-hosted on VPS)
Shipping Integration	Shiprocket
Payment Gateway	Razorpay
Deployment	Hostinger VPS KVM2 + Dokploy

Requirements Analysis & Planning

1. Business Requirements

Core Objectives:

- Build a premium leather shoes e-commerce platform
- Optimize for desktop and mobile web experiences
- Ensure high performance and SEO optimization via Next.js
- Implement secure payment processing with Razorpay
- Automate shipping and logistics with Shiprocket
- Maintain full data sovereignty with self-hosted MinIO storage
- Provide intuitive admin dashboard for inventory management

2. Functional Requirements

(a) User Authentication

- JWT-based auth
- Google OAuth
- Password reset

- Email verification

(b) Product Catalog

- Category browsing
- Advanced filters (size, colour, material, price)
- Search with autocomplete

(c) Product Details

- High-res image gallery (served via CDN)
- Size guide
- Material info
- Reviews
- Related products

(d) Shopping Cart

- Persistent cart (guest + logged-in)
- Save for later
- Cart abandonment recovery

(e) Checkout

- Multi-step checkout
- Razorpay payment integration (Cards, UPI, Netbanking, Wallets)
- Shiprocket shipping rate calculator
- Address validation

(f) Order Management

- Order tracking via Shiprocket
- Invoice generation
- Email notifications
- Return requests

(g) User Dashboard

- Order history with tracking links
- Wishlist
- Address book
- Profile management

(h) Admin Panel

- Product CRUD with MinIO image uploads
- Inventory management
- Order fulfillment with Shiprocket label generation
- Analytics dashboard

- User management

(i) Content Management

- Blog/lookbook
- Promotional banners
- SEO metadata management

3. Non-Functional Requirements

(j) Performance

- Initial page load under 2 seconds
- Lighthouse score 90+
- ISR for product pages
- CDN delivery for static assets and images

(k) Security

- HTTPS only
- PCI DSS compliance via Razorpay (tokenization, no card data storage)
- Input sanitization
- MinIO encrypted storage at rest

(l) Scalability

- Horizontal scaling ready
- CDN for images and static assets
- Database indexing
- MinIO distributed mode ready

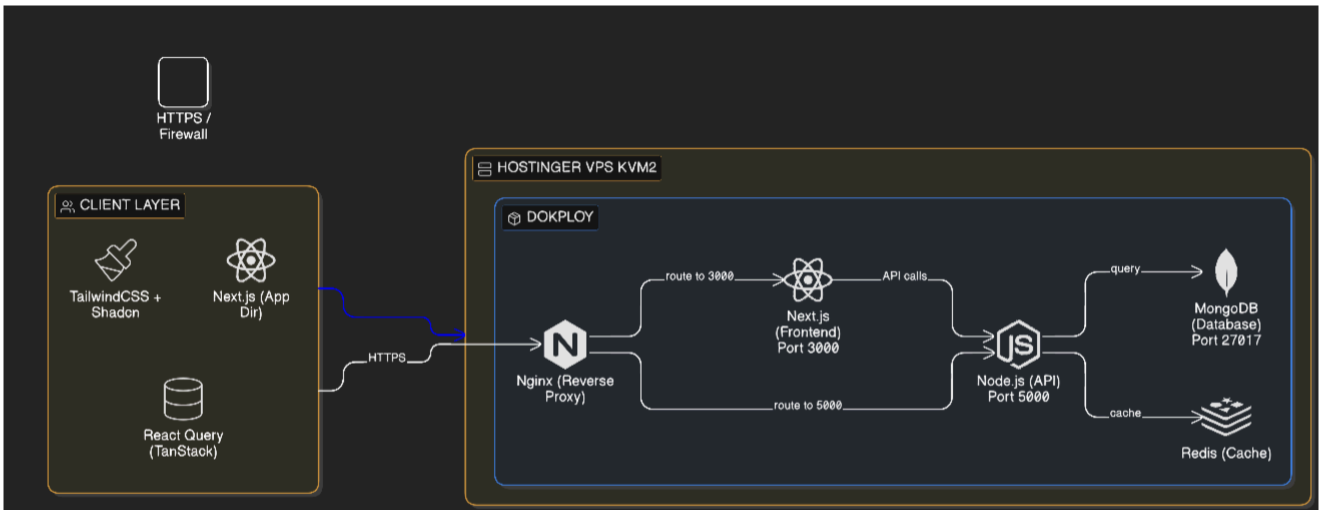
(i) SEO

- Server-side rendering (SSR)
- Structured data
- Sitemap.xml
- Meta tags

(m) Availability

- 99.9% uptime
- Automated backups
- Monitoring

4. Technical Architecture



(n) Authentication Endpoints

- POST /api/auth/register - User registration
- POST /api/auth/login - User login
- GET /api/auth/me - Get current user

(o) Product Endpoints

- GET /api/products - List products (with filters/pagination)
- GET /api/products/:slug - Single product details

(p) Cart Endpoints

- GET /api/cart - Get cart
- POST /api/cart - Add to cart
- PUT /api/cart - Update cart item
- DELETE /api/cart - Remove from cart

(q) Shipping Endpoints

- POST /api/shipping/calculate - Calculate shipping rates via Shiprocket
- POST /api/shipping/create - Create shipment
- GET /api/shipping/track/:awb - Track shipment

(r) Payment Endpoints

- POST /api/payments/razorpay/order - Create Razorpay order

- POST /api/payments/razorpay/verify - Verify Razorpay payment signature
- POST /api/payments/razorpay/webhook - Razorpay webhook handler

(s) Order Endpoints

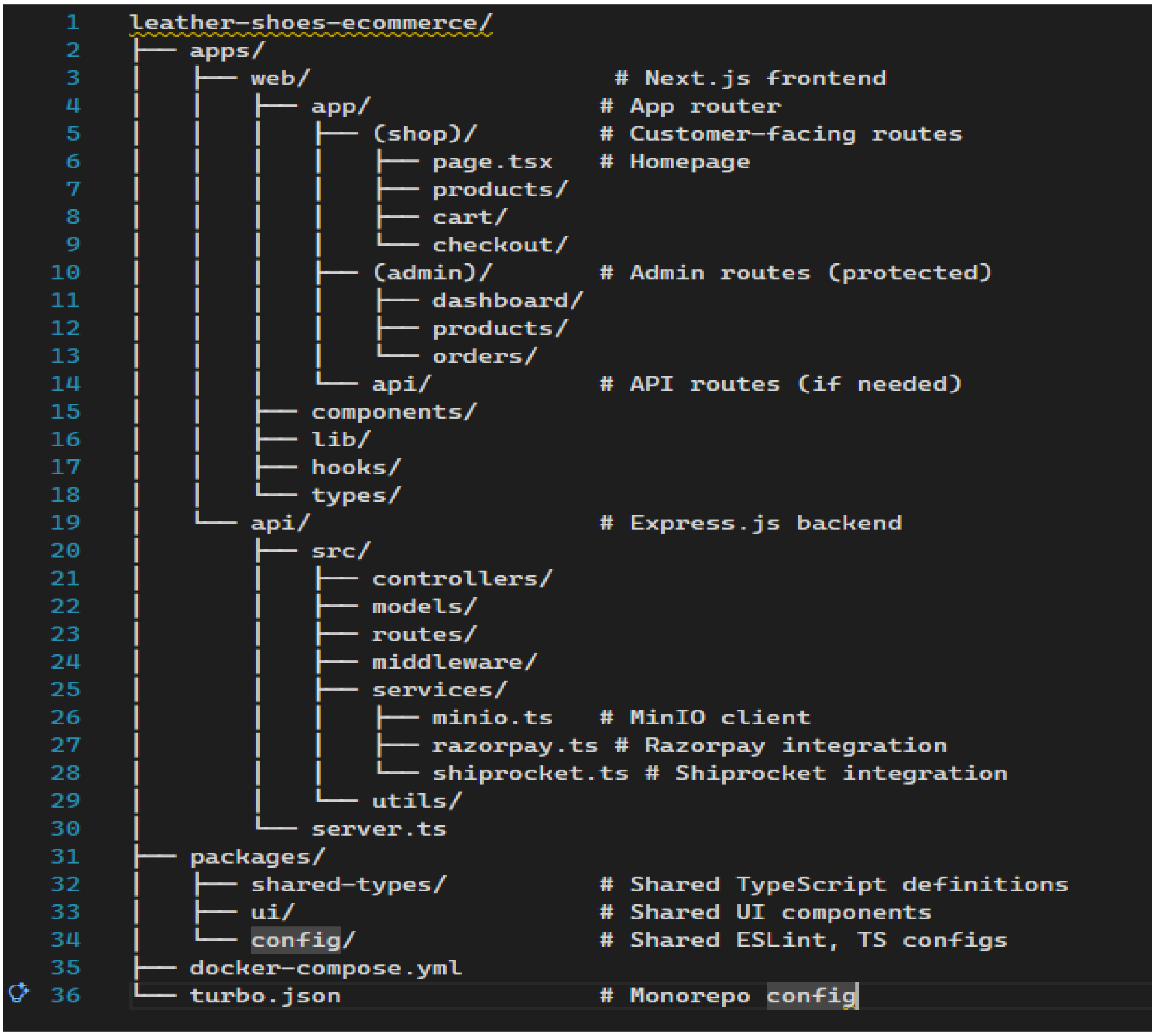
- POST /api/orders - Create order (post-payment)
- GET /api/orders/:id - Order details
- GET /api/orders - List user orders

(t) Admin Endpoints

- POST /api/admin/products - Create product
- PUT /api/admin/products/:id - Update product
- DELETE /api/admin/products/:id - Delete product
- POST /api/admin/upload - Upload image to MinIO
- GET /api/admin/orders - List all orders
- PUT /api/admin/orders/:id - Update order status
- POST /api/admin/orders/:id/ship - Generate Shiprocket label

(u) Development

1. Project Structure



```
leather-shoes-ecommerce/
├── apps/
│   ├── web/                    # Next.js frontend
│   │   ├── app/               # App router
│   │   │   ├── (shop)/        # Customer-facing routes
│   │   │   │   ├── page.tsx   # Homepage
│   │   │   │   ├── products/
│   │   │   │   ├── cart/
│   │   │   │   └── checkout/
│   │   │   ├── (admin)/       # Admin routes (protected)
│   │   │   │   ├── dashboard/
│   │   │   │   ├── products/
│   │   │   │   └── orders/
│   │   │   └── api/           # API routes (if needed)
│   │   ├── components/
│   │   ├── lib/
│   │   ├── hooks/
│   │   └── types/
│   └── api/                   # Express.js backend
│       ├── src/
│       │   ├── controllers/
│       │   ├── models/
│       │   ├── routes/
│       │   ├── middleware/
│       │   ├── services/
│       │   │   ├── minio.ts   # MinIO client
│       │   │   ├── razorpay.ts # Razorpay integration
│       │   │   └── shiprocket.ts # Shiprocket integration
│       │   └── utils/
│       └── server.ts
├── packages/
│   ├── shared-types/          # Shared TypeScript definitions
│   ├── ui/                    # Shared UI components
│   └── config/                # Shared ESLint, TS configs
├── docker-compose.yml
└── turbo.json                 # Monorepo config
```

2. Technology Stack Details

(v) Frontend Layer

- Next.js 14 (App Router) - SSR, SSG, ISR, API routes
- TailwindCSS - Utility-first CSS, accessible components
- React Query - Client state, server state caching
- Razorpay Checkout.js - Payment UI integration

(w) Backend Layer

- Express.js - REST API, business logic
- MongoDB (Mongoose) - Document storage
- NextAuth.js + JWT - Secure session management

(x) Storage & CDN

- MinIO - Self-hosted S3-compatible object storage on VPS
- CloudFlare - Global CDN for MinIO assets (origin pull)
- MinIO Console - Web UI for bucket management

(y) Payments

- Razorpay - Payment gateway (PCI DSS Level 1 compliant)
- Razorpay Webhooks - Payment status updates
- Razorpay Orders API - Server-side order creation

(z) Shipping

- Shiprocket API - Courier aggregation, rate calculation, label generation
- Shiprocket Webhooks - Shipment status updates

Search & Cache

- MongoDB Atlas Search - Full-text product search
- Redis - Session cache, rate limiting, API response caching

Monitoring

- Dokploy built-in monitoring - Container health
- Sentry - Error tracking
- UptimeRobot - External uptime monitoring

3. Third-Party Services & Compliance

Payment Processing

- **Service:** Razorpay
- **Compliance:** PCI DSS Level 1 (Razorpay handles all card data)
- **Features:** Cards, UPI, Netbanking, Wallets, EMI
- **Integration:** Server-side order creation, client-side checkout, webhook verification

Shipping & Logistics

- **Service:** Shiprocket
- **Features:** Multi-courier aggregation, automated label generation, tracking
- **Integration:** Rate API, Shipment API, Tracking API, Webhooks

Content Delivery

- **Service:** CloudFlare or BunnyCDN
- **Purpose:** Global edge caching for MinIO-hosted images and static assets
- **Configuration:** Origin pull from MinIO, custom domain, SSL/TLS

Object Storage

- **Service:** MinIO (Self-hosted)
- **Compliance:** Data sovereignty, encrypted at rest
- **Features:** S3-compatible API, presigned URLs, bucket policies
- **Backup:** MinIO bucket replication to external S3 (optional)

Email Delivery

- **Service:** Resend / AWS SES / SendGrid
- **Purpose:** Transactional emails (orders, shipping, password reset)

Security & Performance

- **WAF:** CloudFlare (if using their CDN)
- **DDoS Protection:** CloudFlare / Hostinger native
- **SSL:** Let's Encrypt (auto-renewed via Dokploy)

Testing

1. Testing Strategy

Unit Testing

- Tools: Jest, React Testing Library
- Coverage: Components, utilities, API handlers, MinIO service, Razorpay service

Integration Testing

- Tools: Jest, Supertest
- Coverage: API endpoints, database operations, Razorpay webhook handling, Shiprocket API integration

E2E Testing

- Tools: Playwright
- Coverage: Critical user flows (purchase journey, payment completion, tracking)

Visual Testing

- Tools: Chromatic
- Coverage: UI component regression

Performance Testing

- Tools: Lighthouse CI
- Coverage: Core Web Vitals, CDN cache hit rates, image loading times

Security Testing

- Tools: OWASP ZAP

- Coverage: Payment flow security, webhook signature verification, file upload validation

2. Critical Test Scenarios

   1. **User Authentication:** Registration, login, password reset, session persistence
   2. **Product Discovery:** Search, filtering, sorting, pagination, image loading from CDN
   3. **Cart Operations:** Add/remove items, quantity updates, persistence across sessions
   4. **Payment Flow:** Razorpay order creation, checkout modal, payment verification, webhook handling
   5. **Shipping Integration:** Rate calculation, address validation, label generation, tracking updates
   6. **File Upload:** MinIO presigned URL generation, image upload, CDN accessibility, deletion
   7. **Inventory:** Stock deduction on payment, out-of-stock handling, variant selection
   8. **Admin:** Product CRUD with images, order status updates, Shiprocket label printing

Deployment & DevOps

1. Infrastructure Setup (Hostinger VPS KVM2)

Server Specifications:

- Plan: KVM 2 (2 vCPU, 8GB RAM, 100GB NVMe)
- OS: Ubuntu 22.04 LTS
- Control Panel: Dokploy (self-hosted PaaS)

Domain & DNS:

- Primary Domain: yourdomain.com
- CDN Domain: cdn.yourdomain.com (CNAME to CDN)
- MinIO Console: storage.yourdomain.com

3. MinIO Configuration

Bucket Setup:

- Bucket Name: leather-shoes
- Public Read Policy (for CDN access)
- CORS configuration for web access
- Lifecycle policy for temporary upload cleanup

CDN Integration:

- Origin: MinIO endpoint (minio:9000)
- Custom domain: cdn.yourdomain.com
- SSL/TLS: Full strict
- Cache rules: 30 days for images, 1 year for static assets
- Image optimization: WebP conversion, responsive images

Backup Strategy:

- Daily MinIO bucket sync to external S3 (AWS/GCP) using MinIO Client (mc)
- Versioning enabled for critical buckets

4. Razorpay Configuration

Test Mode:

- Key ID: rzp_test_xxx
- Key Secret: [Stored in Dokploy secrets]
- Webhook Secret: [Stored in Dokploy secrets]
- Webhook URL: https://api.yourdomain.com/api/payments/razorpay/webhook

Production Mode:

- Key ID: rzp_live_xxx
- Key Secret: [Stored in Dokploy secrets]
- Webhook Secret: [Stored in Dokploy secrets]
- Payment methods: All enabled (Cards, UPI, Netbanking, Wallets)

PCI DSS Compliance Notes:

- No card data touches our servers (tokenized by Razorpay)
- Webhook signature verification mandatory
- HTTPS only for all payment flows
- Audit logs maintained for 1 year

5. Shiprocket Configuration

API Integration:

- Base URL: https://apiv2.shiprocket.in/v1/external
  - Authentication: Email/Password → Token-based
  - Token refresh handled automatically

Database Backups:

- Daily automated backups (MongoDB dumps)
- Point-in-time recovery capability
- Retention: 30 days

MinIO Backups:

- Daily sync to external S3 using mc mirror
- Bucket versioning enabled
- Cross-region replication (optional)

Monitoring Tools:

- Dokploy dashboard for container health
- UptimeRobot for external uptime monitoring (API, CDN, Web)
- Sentry for error tracking (Razorpay failures, Shiprocket errors)
- MinIO Console for storage metrics
- Razorpay Dashboard for payment analytics
- Shiprocket Dashboard for shipping analytics

Log Management:

- Dokploy log aggregation for all containers
- Separate log streams for payment and shipping events (compliance)

SSL Management:

- Let's Encrypt auto-renewal via Dokploy for all domains
- CDN SSL managed by CloudFlare

Overview:

Phase		**Description**	**Duration**	**Key Deliverables**
Phase 1		Planning & Design	3 Weeks	PRD, System Architecture, Third-party account setup (Razorpay, Shiprocket)
Phase 2		Development	10 Weeks	Fully functional application with MinIO storage, Razorpay payment integration, Shiprocket shipping integration
Phase 3		Testing	2 Weeks	Test reports (including payment & shipping flows), bug fixes
Phase 4		Deployment	1 Week	Live production environment, CDN configuration, production API keys
—		**Total Project Duration**	**16 Weeks**	—

- **Access Control & Authorization**
- The platform implements role-based access control to ensure administrative access is restricted based on defined responsibilities. Authorization is enforced at the application and API layers using backend authentication mechanisms and database-level role definitions, minimizing operational risk, preventing unauthorized actions, and improving overall system security.
- **Supporting Services:** Node.js, Express.js, JWT authentication, MongoDB

- **Regulatory & Legal Compliance (India)**
- The system aligns with the Information Technology Act, 2000 and the Consumer Protection (E-Commerce) Rules, 2020 by enforcing secure data handling practices, transparent pricing, clearly defined return and refund policies, and audit-ready transaction logs. Secure payment processing and controlled data storage reduce regulatory exposure and compliance risk.
- **Supporting Services:** Razorpay, MongoDB, HTTPS (SSL/TLS), server-side logging

- **Cash on Delivery (COD) Handling**
- Cash on Delivery (COD) orders are supported through logistics system integration, enabling delivery confirmation, payment collection status, and settlement tracking. This approach supports Indian market preferences while maintaining accurate order and financial reconciliation.
- **Supporting Services:** Shiprocket APIs, MongoDB, webhook-based status updates
- ---
- **Return to Origin (RTO) Management**
- Failed deliveries are managed through automated Return-to-Origin (RTO) workflows that update order status, restore inventory upon product return, and record associated logistics costs. This ensures accurate stock levels and visibility into operational losses.
- **Supporting Services:** Shiprocket webhooks, MongoDB inventory management

---

- **GST Invoicing & Tax Compliance**
- GST-compliant invoices are generated for completed orders, and corresponding credit notes are issued for returned or refunded items. Invoice documents are securely stored and shared with customers, ensuring accurate tax records and regulatory compliance.
- **Supporting Services:** Backend invoice generation services, MongoDB, MinIO object storage, email delivery service
- ---
- **Returns & Refund Management**
- Structured workflows handle customer return and refund requests with defined approval, processing, and closure stages. Refund execution varies by payment method to ensure accuracy and compliance with Indian consumer protection guidelines.

- **Supporting Services:** Razorpay refund APIs (prepaid orders), manual COD reconciliation records, MongoDB

- **User Data Management & Privacy**
- Users are provided with secure controls to manage personal information, including profile and address details. Data access is restricted through authentication mechanisms, and sensitive information is handled using secure storage and transmission practices.
- **Supporting Services:** JWT authentication, MongoDB, HTTPS, access control middleware

---

- **Project Delivery Efficiency**
- The project delivery timeline reflects optimized execution practices, parallel development and validation activities, and reuse of proven architectural patterns. While timelines may vary due to external dependencies such as third-party approvals or content availability, the development approach and infrastructure are designed to support completion at or before the planned schedule without compromising quality or security.
- **Supporting Services:** Modular backend architecture, containerized deployment, automated testing tools
-

- **Final Deliverables**
- Upon completion, the client receives a secure, scalable, and India-compliant e-commerce platform with integrated payments, logistics, inventory management, GST invoicing, administrative controls, and operational workflows. The system is deployed on a cost-optimized infrastructure with monitoring and backup mechanisms to support long-term stability and growth.
- **Supporting Services:** Next.js, Node.js, MongoDB, Razorpay, Shiprocket, Redis, MinIO, CDN, VPS infrastructure

---

- ## Additional Notes

  Actual implementation details, timelines, and deliverables may vary based on requirements and external dependencies.