Universal Full-Stack Project Architecture Guide

Production-Ready Development Framework

Version 3.0 | October 2025

Elite Software Architecture Documentation

October 6, 2025

This document provides a comprehensive, vendor-lock-in-free framework for building production-ready full-stack applications across any domain: e-commerce, SaaS, content platforms, or real-time collaboration tools.

Contents

List of Figures

List of Tables

Part I

Strategic Foundation

Chapter 1

Introduction

1.1 Purpose and Scope

This guide provides a systematic methodology for architecting, developing, and deploying production-grade full-stack applications. It eliminates vendor lock-in by using exclusively open-source, self-hosted technologies while maintaining enterprise-grade security, performance, and scalability.

1.1.1 Target Audience

- Full-stack developers building modern web applications
- Technical architects designing scalable systems
- DevOps engineers implementing CI/CD pipelines
- Security professionals ensuring OWASP/ISO compliance
- Project managers planning production deployments

1.1.2 Document Structure

This guide is organized into four parts:

1. **Strategic Foundation**: Principles, decision frameworks, and project classification
2. **Technical Architecture**: Stack selection, system design, and data modeling
3. **Implementation Guide**: Development workflows, security hardening, and testing
4. **Operations & Maintenance**: Deployment, monitoring, and incident response

1.2 Core Principles

1.2.1 Zero Vendor Lock-In

Principle: All dependencies must be replaceable without architectural changes.
Implementation:

- Use open-source databases (PostgreSQL) instead of proprietary services (Supabase, PlanetScale)
- Self-host authentication via JWT + Prisma instead of Auth0/Firebase
- Deploy on any VPS/Kubernetes instead of Vercel/Netlify-only solutions
- Use BTCPay Server (self-hosted) alongside Stripe for payment flexibility

1.2.2 Security by Default

Principle: Security measures integrated from initial commit, not retrofitted.

Framework Compliance:

Framework	Coverage	Implementation Timing
OWASP Top 10 2021	100%	Architecture phase
ISO 27001	Core controls	Development phase
GDPR	Art. 25 (by design)	Data modeling phase
SOC 2 Type II	Key criteria	Pre-deployment
NIST Cybersecurity	Selected controls	Continuous

Table 1.1: Security Framework Integration Timeline

1.2.3 Performance First

Targets:

- Largest Contentful Paint (LCP): < 2.0s
- First Input Delay (FID): < 100ms
- Cumulative Layout Shift (CLS): < 0.1
- Time to Interactive (TTI): < 3.0s
- Bundle size: < 400 KB (gzipped)

1.2.4 Test-Driven Quality

Coverage Requirements:

Test Type	Tool	Minimum Coverage
Unit	Jest/Vitest	90%
Integration	Supertest	85%
E2E	Playwright	Critical paths (12+)
Accessibility	Axe-core	100% WCAG 2.2 AA
Security	OWASP ZAP	0 high/critical issues
Performance	Lighthouse CI	Score 95

Table 1.2: Testing Coverage Matrix

Chapter 2

Project Classification System

2.1 Domain Categorization

Before architecture design, classify your project to auto-configure appropriate stack components and focus areas.

2.1.1 Classification Decision Tree

```python
def classify_project(requirements):
    if has_inventory_management(requirements):
        return ProjectType.ECOMMERCE
    elif has_multi_tenancy(requirements):
        return ProjectType.SAAS
    elif has_media_streaming(requirements):
        return ProjectType.CONTENT_PLATFORM
    elif has_realtime_collaboration(requirements):
        return ProjectType.REALTIME_COLLAB
    else:
        return ProjectType.CUSTOM
```

Listing 2.1: Project Type Detection Algorithm

2.1.2 E-Commerce Applications

Characteristics:

- Product catalog with inventory tracking
- Shopping cart and checkout workflows
- Payment gateway integration
- Order management systems

Tech Stack Configuration:

Component	Technology
Frontend	Next.js 15 (App Router)
Database	PostgreSQL 16 + Prisma ORM
Payments	Stripe + BTCPay Server
Search	MeiliSearch (self-hosted)
Cache	Valkey/Redis
Queue	BullMQ + Redis

Table 2.1: E-Commerce Stack Configuration

Focus Areas:

1. Atomic inventory operations (prevent overselling)
2. PCI DSS compliance for payment handling
3. SEO optimization (product pages, schema markup)
4. Abandoned cart recovery systems
5. Multi-currency support

2.1.3 SaaS Applications

Characteristics:

- Multi-tenant architecture
- Subscription billing and metering
- Usage analytics and quotas
- Role-based access control (RBAC)

Tech Stack Configuration:

Component	Technology
Frontend	Next.js 15 + React Query
Database	PostgreSQL (RLS enabled)
Auth	JWT + Refresh tokens
Billing	Stripe Billing + Webhooks
Analytics	PostHog (self-hosted)
Feature Flags	Unleash (self-hosted)

Table 2.2: SaaS Stack Configuration

Focus Areas:

1. Tenant isolation (database per tenant vs. shared schema)
2. API rate limiting and quota enforcement
3. Subscription lifecycle management
4. Usage-based billing integration
5. Admin impersonation with audit trails

2.1.4 Content Platforms

Characteristics:

- Media streaming (audio/video)
- Content moderation workflows
- User-generated content (UGC)
- Recommendation engines

Tech Stack Configuration:

Component	Technology
Frontend	Next.js 15 + Web Audio API
Database	PostgreSQL + Prisma
Storage	MinIO (S3-compatible)
CDN	BunnyCDN / Cloudflare
Transcoding	FFmpeg (self-hosted)
Search	Typesense (self-hosted)

Table 2.3: Content Platform Stack Configuration

Focus Areas:

1. Adaptive bitrate streaming (HLS/DASH)
2. Copyright detection and DMCA compliance
3. Content delivery optimization
4. AI-powered recommendations
5. Transcript generation and search

2.1.5 Real-Time Collaboration Tools

Characteristics:

- Simultaneous multi-user editing
- Conflict-free data synchronization
- Presence indicators
- Live cursors and selections

Tech Stack Configuration:

Component	Technology
Frontend	Next.js 15 + Yjs (CRDT)
Database	PostgreSQL + TimescaleDB
Real-time	NATS / self-hosted WebSockets
State Sync	Yjs + y-websocket
Presence	Socket.io rooms
Storage	MinIO for file attachments

Table 2.4: Real-Time Collaboration Stack Configuration

Focus Areas:

1. Operational Transform (OT) vs. CRDT selection
2. Conflict resolution strategies
3. Offline-first architecture
4. WebSocket connection management
5. Snapshot and replay mechanisms

2.2 Stack Selection Matrix

Layer	E-Commerce	SaaS	Content	Real-Time
Frontend	Next.js 15	Next.js 15	Next.js 15	Next.js 15
State	Zustand	React Query	Zustand	Yjs + Zustand
Database	PostgreSQL	PostgreSQL + RLS	PostgreSQL	PostgreSQL + TimescaleDB
Auth	JWT	JWT + MFA	JWT	JWT + WebSocket auth
Payments	Stripe + BTCPay	Stripe Billing	Stripe Connect	N/A
Real-time	WebSockets	SSE	WebSockets	NATS + WebSockets
Cache	Redis	Redis + CDN	CDN + Redis	Redis for presence
Search	MeiliSearch	Typesense	Typesense	N/A
Storage	MinIO	MinIO	MinIO + CDN	MinIO
Queue	BullMQ	BullMQ	BullMQ	N/A

Table 2.5: Technology Stack by Project Type

Part II

Technical Architecture

Chapter 3

System Architecture Design

3.1 Chain-of-Thought Architecture Process

Before generating any code or infrastructure configurations, follow this systematic reasoning chain:

3.1.1 Step 1: User Flow Mapping

Objective: Identify all critical user journeys from entry to conversion.

Process:

1. List all user personas (e.g., guest, authenticated user, admin)
2. Map 3-5 core workflows per persona
3. Identify decision points and failure scenarios
4. Calculate expected throughput per flow

Example (E-Commerce):

```
Flow 1: Guest Purchase
 Landing page (SEO entry)
 Product search/browse
 Product detail view
 Add to cart
 Guest checkout
 Payment
 Order confirmation

Expected: 1000 users/day, 3% conversion = 30 orders/day
```

3.1.2 Step 2: Database Relationship Modeling

Objective: Define normalized schema with appropriate relationships.

Decision Criteria:

Relationship	Use Case	Implementation
One-to-Many	User → Orders	Foreign key
Many-to-Many	Products Categories	Junction table
One-to-One	User → Profile	Shared primary key
Polymorphic	Comments on multiple entities	Discriminator column

Table 3.1: Database Relationship Patterns

3.1.3 Step 3: Security vs. Performance Trade-offs

Trade-off Matrix:

Decision	Security Impact	Performance Impact	Recommendation
JWT vs. Sessions	JWT: No server-side revocation	Sessions: DB query per request	JWT + short expiry (15min) + refresh tokens
Row-Level Security	Automatic enforcement	Query overhead (10-15%)	Enable for sensitive tables only
API Rate Limiting	Prevents DoS	Redis lookup per request	Implement with sliding window

Table 3.2: Security-Performance Trade-off Analysis

3.1.4 Step 4: Constraint Validation

Mandatory Constraints:

- **No External CDNs**: All assets self-hosted or from approved CDNs (Cloudflare/BunnyCDN)
- **No Proprietary Services**: Supabase → PostgreSQL, Auth0 → JWT, Vercel → Nginx
- **Budget Limits**: For MVP, $50/month max (VPS + domain)
- **Compliance**: GDPR (EU users), PCI DSS (if handling cards), WCAG 2.2 AA

3.1.5 Step 5: Technology Justification

Example Decision Log:

Component	Option A	Option B	Choice & Rationale
Database	PostgreSQL	MongoDB	PostgreSQL: ACID, complex queries, mature ecosystem for e-commerce
ORM	Prisma	Drizzle	Prisma: Type-safe, migrations, admin UI (Prisma Studio)
State Management	Redux	Zustand	Zustand: Simpler API, no boilerplate, 3KB vs. 45KB
Payment Gateway	Stripe only	Stripe + BTCPay	Both: Stripe for UX, BTCPay for crypto + no vendor lock-in

Table 3.3: Technology Selection Decision Log

3.2 High-Level Architecture Patterns

3.2.1 Monolithic vs. Microservices

Decision Criteria:

- **Choose Monolith if**: MVP, < 5 developers, < 10k daily users, limited DevOps resources
- **Choose Microservices if**: > 50k users, independent team scaling, polyglot persistence needed

Recommended Hybrid: Modular monolith with extraction strategy.

```
apps/
          web/                    # Next.js frontend
          api/                    # Single Node.js backend
              modules/
                    auth/           # Isolated auth logic
                    products/       # Product management
                    orders/         # Order processing
                    payments/       # Payment integrations
              shared/
                    database/       # Prisma client
                    cache/          # Redis utilities
                    queue/          # BullMQ setup
              main.ts
```

Listing 3.1: Modular Monolith Structure

3.2.2 C4 Model Architecture Diagrams

Level 1: System Context

Actors and Systems:

- **Primary Actors**: End users (guests, authenticated), Admins
- **External Systems**: Payment Gateway (Stripe), Email Service (self-hosted SMTP), CDN
- **Core System**: Full-stack application (web + API + database)

Context Diagram Description:

```
[End User] --HTTP--> [Web Application]
[Web Application] --API--> [Payment Gateway]
[Web Application] --SMTP--> [Email Service]
[Admin] --HTTPS--> [Admin Panel]
[Admin Panel] ---> [Web Application]
```

Level 2: Container Diagram

Containers (Deployable Units):

Container	Technology	Purpose	Scaling
Web App	Next.js 15	SSR, API routes	Horizontal (3+ pods)
Database	PostgreSQL 16	Persistent data	Vertical (16GB RAM)
Cache	Valkey/Redis	Sessions, queries	Horizontal (2 replicas)
Queue	BullMQ	Background jobs	Horizontal (workers)
Storage	MinIO	Media files	Horizontal (4+ nodes)

Table 3.4: Container Inventory

Level 3: Component Diagram (Web App)

Internal Components:

```
Next.js Application
 App Router
    (auth)/             # Auth pages
    (dashboard)/        # Protected routes
    api/                # API routes
    _components/        # Shared components
 Middleware
    auth.middleware     # JWT verification
    ratelimit.middleware # Rate limiting
 Services
    user.service
    product.service
    order.service
 Utilities
     db.client (Prisma)
     cache.client (Redis)
     queue.client (BullMQ)
```

Chapter 4

Data Architecture

4.1 Database Schema Design

4.1.1 Core Entities (E-Commerce Example)

```sql
-- Users table with RBAC
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL, -- bcrypt hash
    role VARCHAR(20) DEFAULT 'user' CHECK (role IN ('user', 'creator', 'admin')),
    email_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Products table
CREATE TABLE products (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    slug VARCHAR(255) UNIQUE NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL CHECK (price >= 0),
    stock INTEGER DEFAULT 0 CHECK (stock >= 0),
    images JSONB, -- Array of image URLs
    category_id UUID REFERENCES categories(id),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Orders table
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    status VARCHAR(20) DEFAULT 'pending'
        CHECK (status IN ('pending', 'paid', 'shipped', 'delivered', 'refunded')),
    total_amount DECIMAL(10, 2) NOT NULL,
    stripe_payment_id VARCHAR(255),
    shipping_address JSONB NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Order items (junction table)
CREATE TABLE order_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    product_id UUID REFERENCES products(id),
    quantity INTEGER NOT NULL CHECK (quantity > 0),
```

```
    price_at_purchase DECIMAL(10, 2) NOT NULL
);

-- Indexes for performance
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_products_slug ON products(slug);
CREATE INDEX idx_orders_user ON orders(user_id);
CREATE INDEX idx_orders_status ON orders(status);
CREATE INDEX idx_order_items_order ON order_items(order_id);
```

Listing 4.1: PostgreSQL Schema Definition

4.1.2 Row-Level Security (RLS) Policies

Implementation for Multi-Tenant SaaS:

```
-- Enable RLS on sensitive tables
ALTER TABLE orders ENABLE ROW LEVEL SECURITY;

-- Users can only see their own orders
CREATE POLICY user_orders_select ON orders
    FOR SELECT
    USING (user_id = current_setting('app.user_id')::UUID);

-- Admins can see all orders
CREATE POLICY admin_orders_all ON orders
    FOR ALL
    USING (
        EXISTS (
            SELECT 1 FROM users
            WHERE id = current_setting('app.user_id')::UUID
            AND role = 'admin'
        )
    );
```

Listing 4.2: RLS Policy Examples

Prisma Middleware Integration:

```
// lib/prisma-middleware.ts
import { Prisma } from '@prisma/client';

export function applyRLSMiddleware(prisma: PrismaClient, userId: string,
    userRole: string) {
  prisma.$use(async (params, next) => {
    // Set session variables for RLS
    if (params.action === 'findMany' || params.action === 'findFirst') {
      await prisma.$executeRaw`SET LOCAL app.user_id = ${userId}`;
      await prisma.$executeRaw`SET LOCAL app.user_role = ${userRole}`;
    }
    return next(params);
  });
}
```

Listing 4.3: Prisma RLS Middleware

4.1.3 Data Migration Strategy

Prisma Migrations Workflow:

```
# Create migration
npx prisma migrate dev --name add_products_table

# Apply to production (zero-downtime)
```

```
npx prisma migrate deploy

# Rollback (manual)
# 1. Revert migration file
# 2. Run: npx prisma migrate resolve --rolled-back <migration_id>
```

Listing 4.4: Migration Commands

Zero-Downtime Migration Pattern:

1. Add new column as nullable
2. Deploy application code using new column
3. Backfill data for existing rows
4. Make column NOT NULL via another migration

4.2 Caching Strategy

4.2.1 Cache Layer Architecture

Layer	Technology	TTL	Use Case
L1: Client	React Query	5 min	API responses
L2: CDN	Cloudflare	1 hour	Static assets
L3: Application	Redis	15 min	DB queries
L4: Database	PostgreSQL	Permanent	Query results cache

Table 4.1: Multi-Layer Caching Strategy

4.2.2 Redis Caching Implementation

```
// lib/cache.ts
import Redis from 'ioredis';

const redis = new Redis({
  host: process.env.REDIS_HOST,
  port: 6379,
  password: process.env.REDIS_PASSWORD,
});

export async function getCachedOrFetch<T>(
  key: string,
  fetcher: () => Promise<T>,
  ttlSeconds: number = 900
): Promise<T> {
  // Try cache first
  const cached = await redis.get(key);
  if (cached) {
    return JSON.parse(cached) as T;
  }

  // Cache miss: fetch and store
  const data = await fetcher();
  await redis.setex(key, ttlSeconds, JSON.stringify(data));
  return data;
}

```

```
// Usage in API route
export async function GET(request: Request) {
  const products = await getCachedOrFetch(
    'products:all',
    () => prisma.product.findMany(),
    600 // 10 minutes
  );
  return Response.json(products);
}
```

Listing 4.5: Redis Cache Utility

Part III

Implementation Guide

Chapter 5

Security Implementation

5.1 OWASP Top 10 Compliance Checklist

#	Vulnerability	Mitigation	Implementation
A01	Broken Access Control	Enforce RLS + middleware checks	Prisma RLS policies
A02	Cryptographic Failures	HTTPS + encrypt PII	Caddy auto-HTTPS + pg_crypto
A03	Injection	Parameterized queries	Prisma (prevents SQL injection)
A04	Insecure Design	Threat modeling (STRIDE)	See Section 5.5
A05	Security Misconfiguration	Secure headers + defaults	Helmet.js middleware
A06	Vulnerable Components	Dependency scanning	npm audit + Snyk
A07	Auth Failures	JWT + MFA + rate limiting	jose lib + TOTP
A08	Data Integrity Failures	Input validation	Zod schemas
A09	Logging Failures	Audit logs + monitoring	Sentry + database logs
A10	SSRF	URL validation + allowlists	Zod URL schema

Table 5.1: OWASP Top 10 2021 Mitigation Checklist

5.2 Authentication System

5.2.1 JWT Implementation

Token Structure:

```
// lib/auth/jwt.ts
import * as jose from 'jose';

const JWT_SECRET = new TextEncoder().encode(process.env.JWT_SECRET!);

export async function generateTokens(userId: string, role: string) {
  // Access token (short-lived)
  const accessToken = await new jose.SignJWT({ userId, role })
    .setProtectedHeader({ alg: 'HS256' })
    .setIssuedAt()
    .setExpirationTime('15m')
    .sign(JWT_SECRET);

  // Refresh token (long-lived)
  const refreshToken = await new jose.SignJWT({ userId })
    .setProtectedHeader({ alg: 'HS256' })
    .setIssuedAt()
```

```
    .setExpirationTime('7d')
    .sign(JWT_SECRET);

  return { accessToken, refreshToken };
}

export async function verifyToken(token: string) {
  try {
    const { payload } = await jose.jwtVerify(token, JWT_SECRET);
    return payload as { userId: string; role: string };
  } catch (error) {
    throw new Error('Invalid token');
  }
}
```

Listing 5.1: JWT Token Generation

5.2.2 Multi-Factor Authentication (MFA)

TOTP Implementation:

```
// lib/auth/mfa.ts
import * as OTPAuth from 'otpauth';

export function generateMFASecret(email: string) {
  const secret = new OTPAuth.Secret();
  const totp = new OTPAuth.TOTP({
    issuer: 'YourApp',
    label: email,
    algorithm: 'SHA1',
    digits: 6,
    period: 30,
    secret: secret,
  });

  return {
    secret: secret.base32,
    qrCode: totp.toString(), // otpauth:// URL for QR code
  };
}

export function verifyMFAToken(secret: string, token: string): boolean {
  const totp = new OTPAuth.TOTP({
    secret: OTPAuth.Secret.fromBase32(secret),
  });

  // Allow 1 period drift (30s window on each side)
  const delta = totp.validate({ token, window: 1 });
  return delta !== null;
}
```

Listing 5.2: MFA Setup and Verification

5.3 Input Validation and Sanitization

5.3.1 Zod Schema Validation

```
// lib/validators.ts
import { z } from 'zod';

export const RegisterSchema = z.object({
```

```
  email: z.string().email().max(255),
  password: z.string()
    .min(12, 'Password must be at least 12 characters')
    .regex(/[A-Z]/, 'Must contain uppercase')
    .regex(/[a-z]/, 'Must contain lowercase')
    .regex(/[0-9]/, 'Must contain number')
    .regex(/[^A-Za-z0-9]/, 'Must contain special character'),
  name: z.string().min(2).max(100),
});

export const ProductSchema = z.object({
  name: z.string().min(3).max(255),
  slug: z.string().regex(/^[a-z0-9-]+$/),
  description: z.string().max(5000),
  price: z.number().positive().max(999999.99),
  stock: z.number().int().nonnegative(),
  categoryId: z.string().uuid(),
});

// Usage in API route
export async function POST(request: Request) {
  const body = await request.json();

  // Validate and sanitize
  const validated = RegisterSchema.parse(body);

  // Now safe to use validated data
  const user = await createUser(validated);
  return Response.json(user);
}
```

Listing 5.3: Validation Schemas

5.3.2 XSS Prevention

```
// lib/sanitize.ts
import DOMPurify from 'isomorphic-dompurify';

export function sanitizeHTML(dirty: string): string {
  return DOMPurify.sanitize(dirty, {
    ALLOWED_TAGS: ['b', 'i', 'em', 'strong', 'a', 'p', 'br'],
    ALLOWED_ATTR: ['href'],
  });
}

// Usage for user-generated content
const sanitizedDescription = sanitizeHTML(product.description);
```

Listing 5.4: Content Sanitization

5.4 Rate Limiting

5.4.1 Sliding Window Rate Limiter

```
// lib/rate-limiter.ts
import Redis from 'ioredis';

const redis = new Redis(process.env.REDIS_URL);

export async function rateLimit(
```

```
  identifier: string, // IP or user ID
  maxRequests: number = 100,
  windowSeconds: number = 3600
): Promise<{ allowed: boolean; remaining: number }> {
  const key = `ratelimit:${identifier}`;
  const now = Date.now();
  const windowStart = now - windowSeconds * 1000;

  // Remove old entries
  await redis.zremrangebyscore(key, 0, windowStart);

  // Count requests in current window
  const count = await redis.zcard(key);

  if (count >= maxRequests) {
    return { allowed: false, remaining: 0 };
  }

  // Add current request
  await redis.zadd(key, now, `${now}`);
  await redis.expire(key, windowSeconds);

  return { allowed: true, remaining: maxRequests - count - 1 };
}

// Middleware usage
export async function rateLimitMiddleware(req: Request) {
  const ip = req.headers.get('x-forwarded-for') || 'unknown';
  const { allowed, remaining } = await rateLimit(ip, 100, 3600);

  if (!allowed) {
    return new Response('Too Many Requests', {
      status: 429,
      headers: { 'X-RateLimit-Remaining': '0' },
    });
  }

  return { remaining };
}
```

Listing 5.5: Redis-Based Rate Limiter

5.5 Threat Modeling

5.5.1 STRIDE Analysis

Threat Type	Risk Level	Attack Scenario	Mitigation Strategy
Spoofing	High	Attacker steals JWT tokens via XSS	Short expiry (15min) + httpOnly cookies + CSP
Tampering	Medium	Modified API payloads bypass validation	Zod validation + request signing
Repudiation	Low	User denies placing order	Audit logs with immutable timestamps
Information Disclosure	High	Leaked database credentials	Env vars + secret rotation + no .env in repo
Denial of Service	Medium	API flooding overwhelms server	Rate limiting + CDN DDoS protection

Threat Type	Risk Level	Attack Scenario	Mitigation Strategy
Elevation of Privilege	Critical	User escalates to admin role	RLS policies + middleware RBAC checks

Table 5.2: STRIDE Threat Model for E-Commerce Platform

Chapter 6

Testing Strategy

6.1 Test Pyramid

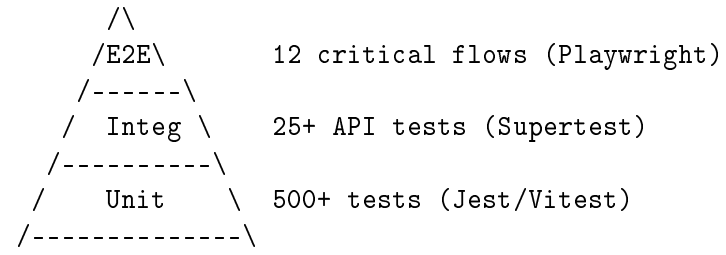

Figure 6.1: Test Distribution Pyramid

6.2 Unit Testing

6.2.1 Component Testing

```
// components/ProductCard.test.tsx
import { render, screen, fireEvent } from '@testing-library/react';
import { ProductCard } from './ProductCard';

describe('ProductCard', () => {
  const mockProduct = {
    id: '123',
    name: 'Test Product',
    price: 29.99,
    image: '/test.jpg',
  };

  it('displays product information correctly', () => {
    render(<ProductCard product={mockProduct} />);

    expect(screen.getByText('Test Product')).toBeInTheDocument();
    expect(screen.getByText('$29.99')).toBeInTheDocument();
  });

  it('calls onAddToCart when button clicked', () => {
    const onAddToCart = jest.fn();
    render(<ProductCard product={mockProduct} onAddToCart={onAddToCart} />);

    fireEvent.click(screen.getByRole('button', { name: /add to cart/i }));
    expect(onAddToCart).toHaveBeenCalledWith(mockProduct.id);
  });
});
```

Listing 6.1: React Component Test

6.3 Integration Testing

6.3.1 API Route Testing

```
// tests/integration/products.test.ts
import request from 'supertest';
import { app } from '../../src/app';
import { prisma } from '../../lib/db';

describe('Product API', () => {
  beforeEach(async () => {
    await prisma.product.deleteMany();
  });

  describe('POST /api/products', () => {
    it('creates a new product with valid data', async () => {
      const response = await request(app)
        .post('/api/products')
        .set('Authorization', `Bearer ${adminToken}`)
        .send({
          name: 'New Product',
          slug: 'new-product',
          price: 49.99,
          stock: 10,
        })
        .expect(201);

      expect(response.body).toMatchObject({
        name: 'New Product',
        slug: 'new-product',
      });

      const dbProduct = await prisma.product.findUnique({
        where: { id: response.body.id },
      });
      expect(dbProduct).not.toBeNull();
    });

    it('returns 400 for invalid price', async () => {
      await request(app)
        .post('/api/products')
        .set('Authorization', `Bearer ${adminToken}`)
        .send({
          name: 'Invalid Product',
          slug: 'invalid',
          price: -10, // Invalid negative price
        })
        .expect(400);
    });
  });
});
```

Listing 6.2: API Integration Test

6.4 End-to-End Testing

6.4.1 Critical User Flows

```
// tests/e2e/checkout.spec.ts
import { test, expect } from '@playwright/test';
```

```

test.describe('Complete Purchase Flow', () => {
  test('guest user can complete checkout', async ({ page }) => {
    // 1. Browse products
    await page.goto('/shop');
    await expect(page.locator('h1')).toContainText('Shop');

    // 2. Add product to cart
    await page.locator('.product-card').first().click();
    await page.locator('button:has-text("Add to Cart")').click();
    await expect(page.locator('.cart-badge')).toContainText('1');

    // 3. Proceed to checkout
    await page.goto('/cart');
    await page.locator('button:has-text("Checkout")').click();

    // 4. Fill shipping information
    await page.fill('[name="email"]', 'test@example.com');
    await page.fill('[name="firstName"]', 'John');
    await page.fill('[name="lastName"]', 'Doe');
    await page.fill('[name="address"]', '123 Test St');
    await page.fill('[name="city"]', 'Test City');
    await page.fill('[name="postalCode"]', '12345');

    // 5. Enter payment (Stripe test mode)
    const stripeFrame = page.frameLocator('iframe[name*="stripe"]');
    await stripeFrame.fill('[name="cardnumber"]', '4242424242424242');
    await stripeFrame.fill('[name="exp-date"]', '12/30');
    await stripeFrame.fill('[name="cvc"]', '123');
    await stripeFrame.fill('[name="postal"]', '12345');

    // 6. Submit order
    await page.locator('button:has-text("Complete Order")').click();

    // 7. Verify confirmation
    await expect(page).toHaveURL(/\/order\/[a-f0-9-]+/);
    await expect(page.locator('h1')).toContainText('Order Confirmed');

    // Verify order exists in database
    const orderId = page.url().split('/').pop();
    // Additional API check can be added here
  });
});
```

Listing 6.3: E2E Checkout Flow

6.5 Accessibility Testing

6.5.1 Automated a11y Audits

```
// tests/accessibility/a11y.test.ts
import { test, expect } from '@playwright/test';
import AxeBuilder from '@axe-core/playwright';

test.describe('Accessibility Compliance', () => {
  test('homepage meets WCAG 2.2 AA', async ({ page }) => {
    await page.goto('/');

    const accessibilityScanResults = await new AxeBuilder({ page })
      .withTags(['wcag2a', 'wcag2aa', 'wcag22aa'])
      .analyze();
```

```
12
13    expect(accessibilityScanResults.violations).toEqual([]);
14  });
15
16  test('product page has proper ARIA labels', async ({ page }) => {
17    await page.goto('/product/test-product');
18
19    const addToCartButton = page.locator('button[aria-label="Add to cart"]');
20    await expect(addToCartButton).toBeVisible();
21
22    const productImage = page.locator('img[alt]').first();
23    await expect(productImage).toHaveAttribute('alt');
24  });
25
26  test('forms have associated labels', async ({ page }) => {
27    await page.goto('/login');
28
29    const emailInput = page.locator('input[type="email"]');
30    const emailLabel = page.locator('label[for]');
31
32    const labelFor = await emailLabel.getAttribute('for');
33    const inputId = await emailInput.getAttribute('id');
34
35    expect(labelFor).toBe(inputId);
36  });
37});
```

Listing 6.4: Axe-core Accessibility Tests

6.6 Performance Testing

6.6.1 Lighthouse CI Integration

```
1 // lighthouserc.js
2 module.exports = {
3   ci: {
4     collect: {
5       url: [
6         'http://localhost:3000/',
7         'http://localhost:3000/shop',
8         'http://localhost:3000/product/test',
9       ],
10      numberOfRuns: 3,
11    },
12    assert: {
13      preset: 'lighthouse:recommended',
14      assertions: {
15        'categories:performance': ['error', { minScore: 0.9 }],
16        'categories:accessibility': ['error', { minScore: 0.95 }],
17        'categories:best-practices': ['error', { minScore: 0.9 }],
18        'categories:seo': ['error', { minScore: 0.9 }],
19        'first-contentful-paint': ['error', { maxNumericValue: 2000 }],
20        'largest-contentful-paint': ['error', { maxNumericValue: 2500 }],
21        'cumulative-layout-shift': ['error', { maxNumericValue: 0.1 }],
22      },
23    },
24    upload: {
25      target: 'temporary-public-storage',
26    },
27  },
```

```
};
```

Listing 6.5: Lighthouse Configuration

6.7 Security Testing

6.7.1 OWASP ZAP Baseline Scan

```
#!/bin/bash
# tests/security/run-zap-scan.sh

docker run -v $(pwd):/zap/wrk/:rw \
  -t owasp/zap2docker-stable \
  zap-baseline.py \
  -t http://host.docker.internal:3000 \
  -r zap-report.html \
  -J zap-report.json \
  -c zap-config.conf \
  -I # Ignore false positives

# Check exit code
if [ $? -ne 0 ]; then
  echo "Security vulnerabilities detected!"
  exit 1
fi
```

Listing 6.6: ZAP Security Scan Script

Part IV

Operations & Maintenance

Chapter 7

Deployment Architecture

7.1 Self-Hosted Deployment

7.1.1 Docker Compose Production Stack

```
# docker/docker-compose.prod.yml
version: '3.8'

services:
  postgres:
    image: postgres:16-alpine
    restart: unless-stopped
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      PGDATA: /var/lib/postgresql/data/pgdata
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backups:/backups
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - backend

  redis:
    image: valkey/valkey:7-alpine
    restart: unless-stopped
    command: valkey-server --requirepass ${REDIS_PASSWORD}
    volumes:
      - redis_data:/data
    healthcheck:
      test: ["CMD", "valkey-cli", "ping"]
      interval: 10s
    networks:
      - backend

  minio:
    image: minio/minio:latest
    restart: unless-stopped
    command: server /data --console-address ":9001"
    environment:
      MINIO_ROOT_USER: ${MINIO_ROOT_USER}
      MINIO_ROOT_PASSWORD: ${MINIO_ROOT_PASSWORD}
    volumes:
      - minio_data:/data
    networks:
      - backend
```

```
47
48  web:
49    build:
50      context: ../apps/web
51      dockerfile: ../../docker/Dockerfile.web
52      args:
53        NODE_ENV: production
54    restart: unless-stopped
55    environment:
56      DATABASE_URL: postgresql://${DB_USER}:${DB_PASSWORD}@postgres:5432/${
    DB_NAME}
57      REDIS_URL: redis://:${REDIS_PASSWORD}@redis:6379
58      JWT_SECRET: ${JWT_SECRET}
59      STRIPE_SECRET_KEY: ${STRIPE_SECRET_KEY}
60      NEXT_PUBLIC_APP_URL: ${APP_URL}
61    depends_on:
62      postgres:
63        condition: service_healthy
64      redis:
65        condition: service_healthy
66    networks:
67      - backend
68      - frontend
69
70  caddy:
71    image: caddy:2-alpine
72    restart: unless-stopped
73    ports:
74      - "80:80"
75      - "443:443"
76    volumes:
77      - ./Caddyfile:/etc/caddy/Caddyfile
78      - caddy_data:/data
79      - caddy_config:/config
80    depends_on:
81      - web
82    networks:
83      - frontend
84
85 volumes:
86   postgres_data:
87   redis_data:
88   minio_data:
89   caddy_data:
90   caddy_config:
91
92 networks:
93   frontend:
94   backend:
```

Listing 7.1: Complete Production Stack

7.1.2 Caddyfile Configuration

```
1 # docker/Caddyfile
2 {
3     email admin@yourdomain.com
4 }
5
6 yourdomain.com {
7     reverse_proxy web:3000
8
```

```
    encode zstd gzip

    header {
        # Security headers
        Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
        X-Content-Type-Options "nosniff"
        X-Frame-Options "DENY"
        X-XSS-Protection "1; mode=block"
        Referrer-Policy "strict-origin-when-cross-origin"
        Permissions-Policy "geolocation=(), microphone=(), camera=()"

        # CSP
        Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-
    inline' https://js.stripe.com; style-src 'self' 'unsafe-inline'; img-src '
    self' data: https:; font-src 'self' data:; connect-src 'self' https://api.
    stripe.com; frame-src https://js.stripe.com"

        # Remove server identification
        -Server
    }

    # Rate limiting
    rate_limit {
        zone dynamic {
            key {remote_host}
            events 100
            window 1m
        }
    }

    # Logging
    log {
        output file /var/log/caddy/access.log
        format json
    }
}

# MinIO console (internal only)
minio.yourdomain.com {
    reverse_proxy minio:9001

    @internal {
        remote_ip 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16
    }
    handle @internal {
        reverse_proxy minio:9001
    }
    handle {
        abort
    }
}
```

Listing 7.2: Caddy Reverse Proxy with Auto-HTTPS

7.2 Kubernetes Deployment

7.2.1 Production Manifests

```
# k8s/web-deployment.yaml
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: web-app
  namespace: production
  labels:
    app: web
    version: v1.0.0
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
        version: v1.0.0
    spec:
      containers:
      - name: web
        image: your-registry.com/web-app:1.0.0
        ports:
        - containerPort: 3000
          name: http
        env:
        - name: NODE_ENV
          value: "production"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: redis-url
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /api/health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3
        readinessProbe:
          httpGet:
            path: /api/health
            port: 3000
          initialDelaySeconds: 10
          periodSeconds: 5
          timeoutSeconds: 3
```

```
          failureThreshold: 2
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web
              topologyKey: kubernetes.io/hostname
---
apiVersion: v1
kind: Service
metadata:
  name: web-service
  namespace: production
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 3000
    protocol: TCP
    name: http
  selector:
    app: web
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: web-hpa
  namespace: production
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web-app
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

Listing 7.3: Kubernetes Deployment Configuration

7.3 CI/CD Pipeline

7.3.1 GitHub Actions Workflow

```yaml
# .github/workflows/deploy.yml
name: CI/CD Pipeline

on:
  push:
    branches: [main, staging]
  pull_request:
    branches: [main]

env:
  NODE_VERSION: '20'
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:16
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: test_db
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 5432:5432

      redis:
        image: redis:7
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 6379:6379

    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Lint code
        run: npm run lint

      - name: Type check
        run: npm run type-check

      - name: Run Prisma migrations
        run: npx prisma migrate deploy
        env:
```

```
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_db

      - name: Unit tests
        run: npm run test:unit -- --coverage
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_db
          REDIS_URL: redis://localhost:6379

      - name: Integration tests
        run: npm run test:integration
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_db
          REDIS_URL: redis://localhost:6379

      - name: Install Playwright
        run: npx playwright install --with-deps

      - name: E2E tests
        run: npm run test:e2e
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_db
          REDIS_URL: redis://localhost:6379

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          files: ./coverage/coverage-final.json

      - name: Security audit
        run: npm audit --production --audit-level=high

      - name: OWASP Dependency Check
        uses: dependency-check/Dependency-Check_Action@main
        with:
          project: 'web-app'
          path: '.'
          format: 'JSON'

  lighthouse:
    runs-on: ubuntu-latest
    needs: test

    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}

      - name: Install dependencies
        run: npm ci

      - name: Build application
        run: npm run build

      - name: Run Lighthouse CI
        run: |
          npm install -g @lhci/cli
          lhci autorun
        env:
          LHCI_GITHUB_APP_TOKEN: ${{ secrets.LHCI_GITHUB_APP_TOKEN }}

```

```yaml
  build-and-push:
    runs-on: ubuntu-latest
    needs: [test, lighthouse]
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'

    permissions:
      contents: read
      packages: write

    steps:
      - uses: actions/checkout@v4

      - name: Log in to Container Registry
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
          tags: |
            type=sha,prefix={{branch}}-
            type=semver,pattern={{version}}
            type=raw,value=latest,enable={{is_default_branch}}

      - name: Build and push Docker image
        uses: docker/build-push-action@v5
        with:
          context: ./apps/web
          file: ./docker/Dockerfile.web
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max

  deploy:
    runs-on: ubuntu-latest
    needs: build-and-push
    if: github.ref == 'refs/heads/main'

    steps:
      - uses: actions/checkout@v4

      - name: Deploy to production
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: ${{ secrets.PRODUCTION_HOST }}
          username: ${{ secrets.PRODUCTION_USER }}
          key: ${{ secrets.PRODUCTION_SSH_KEY }}
          script: |
            cd /opt/web-app
            docker-compose pull
            docker-compose up -d --no-deps web
            docker system prune -af
```

Listing 7.4: Complete CI/CD Pipeline

Chapter 8

Monitoring and Observability

8.1 Health Check Endpoints

```
// app/api/health/route.ts
import { NextResponse } from 'next/server';
import { prisma } from '@/lib/db';
import { redis } from '@/lib/cache';

export async function GET() {
  const checks = {
    timestamp: new Date().toISOString(),
    status: 'healthy',
    checks: {},
  };

  try {
    // Database check
    await prisma.$queryRaw`SELECT 1`;
    checks.checks.database = { status: 'up', responseTime: '< 50ms' };
  } catch (error) {
    checks.checks.database = { status: 'down', error: error.message };
    checks.status = 'unhealthy';
  }

  try {
    // Redis check
    await redis.ping();
    checks.checks.redis = { status: 'up' };
  } catch (error) {
    checks.checks.redis = { status: 'down', error: error.message };
    checks.status = 'unhealthy';
  }

  // External services check
  try {
    const stripeCheck = await fetch('https://status.stripe.com/api/v2/status.
    json');
    const stripeData = await stripeCheck.json();
    checks.checks.stripe = { status: stripeData.status.indicator };
  } catch (error) {
    checks.checks.stripe = { status: 'unknown' };
  }

  const statusCode = checks.status === 'healthy' ? 200 : 503;
  return NextResponse.json(checks, { status: statusCode });
}
```

Listing 8.1: Health Check API

8.2 Error Tracking with Sentry

```
// lib/monitoring/sentry.ts
import * as Sentry from '@sentry/nextjs';

Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV,
  tracesSampleRate: 0.1, // 10% of transactions

  beforeSend(event, hint) {
    // Filter sensitive data
    if (event.request) {
      delete event.request.cookies;
      delete event.request.headers?.authorization;
    }
    return event;
  },

  integrations: [
    new Sentry.Integrations.Http({ tracing: true }),
    new Sentry.Integrations.Prisma({ client: prisma }),
  ],
});

// Usage in error boundaries
export function captureException(error: Error, context?: Record<string, any>) {
  Sentry.captureException(error, {
    contexts: { custom: context },
    level: 'error',
  });
}
```

Listing 8.2: Sentry Integration

8.3 Analytics with PostHog

```
// lib/analytics/posthog.ts
import posthog from 'posthog-js';

export function initAnalytics() {
  if (typeof window !== 'undefined') {
    posthog.init(process.env.NEXT_PUBLIC_POSTHOG_KEY!, {
      api_host: process.env.NEXT_PUBLIC_POSTHOG_HOST || 'https://app.posthog.com
    ',
      loaded: (posthog) => {
        if (process.env.NODE_ENV === 'development') posthog.debug();
      },
      capture_pageview: false, // Manual page view tracking
      autocapture: false, // Manual event tracking for privacy
    });
  }
}

export function trackEvent(event: string, properties?: Record<string, any>) {
  posthog.capture(event, properties);
}

// Track conversions
export function trackPurchase(orderId: string, amount: number) {
  trackEvent('purchase_completed', {
```

```
    order_id: orderId,
    value: amount,
    currency: 'USD',
  });
}
```

Listing 8.3: Self-Hosted Analytics

Chapter 9

Operations Documentation

9.1 Service Inventory

Service	Port	Status	Manager	Purpose
Web Application	3000	Running	Docker/PM2	Next.js frontend + API
PostgreSQL	5432	Running	Docker	Primary database
Redis/Valkey	6379	Running	Docker	Caching + sessions
MinIO	9000	Running	Docker	S3-compatible storage
Caddy	80/443	Running	Docker	Reverse proxy + HTTPS
BullMQ Workers	N/A	Running	PM2	Background jobs

Table 9.1: Production Service Inventory

9.2 Recovery Procedures

9.2.1 Database Recovery

```
#!/bin/bash
# scripts/db-backup.sh

BACKUP_DIR="/backups"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="production_db"

# Create backup
docker exec postgres pg_dump -U postgres $DB_NAME | \
  gzip > "$BACKUP_DIR/${DB_NAME}_${DATE}.sql.gz"

# Keep only last 30 days
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete

# Restore from backup (if needed)
# gunzip < backup.sql.gz | docker exec -i postgres psql -U postgres $DB_NAME
```

Listing 9.1: PostgreSQL Backup and Restore

9.2.2 Service Recovery Commands

```
#!/bin/bash
# scripts/recover-services.sh

# Check all service status
check_status() {
  echo "=== Service Status ==="
```

```
  docker-compose ps
  pm2 status
}

# Restart individual services
restart_web() {
  docker-compose restart web
  # OR for PM2:
  pm2 restart web-app
}

restart_database() {
  docker-compose restart postgres
  # Wait for health check
  until docker exec postgres pg_isready; do
    echo "Waiting for database..."
    sleep 2
  done
}

restart_redis() {
  docker-compose restart redis
}

# Full system restart (last resort)
full_restart() {
  docker-compose down
  docker-compose up -d
  pm2 restart all
}

# Execute based on argument
case "$1" in
  status)
    check_status
    ;;
  web)
    restart_web
    ;;
  database)
    restart_database
    ;;
  redis)
    restart_redis
    ;;
  full)
    full_restart
    ;;
  *)
    echo "Usage: $0 {status|web|database|redis|full}"
    exit 1
esac
```

Listing 9.2: Emergency Recovery Procedures

9.3 Incident Response Log

Date	Issue	Root Cause	Resolution
2025-10-06	Database timeout	Connection pool exhausted	Increased pool size to 20
2025-10-05	High memory usage	Memory leak in worker	Updated BullMQ to v5.2.1
2025-10-03	SSL cert expired	Forgot renewal	Enabled Caddy auto-renewal
2025-09-28	Slow queries	Missing indexes	Added indexes on foreign keys

Table 9.2: Recent Incident Log

9.4 Maintenance Checklist

Task	Frequency	Command/Action
Database backup	Daily	Automated via cron
Security updates	Weekly	`npm audit && npm update`
Log rotation	Weekly	`docker logs -tail 1000`
SSL certificate check	Monthly	Automatic via Caddy
Dependency audit	Monthly	`npm audit && snyk test`
Load testing	Quarterly	`artillery run load-test.yml`
Disaster recovery drill	Quarterly	Restore from backup to staging
Security scan	Quarterly	`owasp-zap baseline scan`

Table 9.3: Operational Maintenance Schedule

Appendix A

Quick Reference

A.1 Environment Variables Template

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/dbname"
POSTGRES_USER="postgres"
POSTGRES_PASSWORD="secure_password_here"
POSTGRES_DB="production_db"

# Redis
REDIS_URL="redis://:password@localhost:6379"
REDIS_PASSWORD="secure_redis_password"

# Authentication
JWT_SECRET="generate_with_openssl_rand_base64_32"
JWT_REFRESH_SECRET="another_random_secret"

# Payments
STRIPE_SECRET_KEY="sk_test_..."
STRIPE_WEBHOOK_SECRET="whsec_..."
STRIPE_PUBLISHABLE_KEY="pk_test_..."

# Storage
MINIO_ROOT_USER="admin"
MINIO_ROOT_PASSWORD="secure_minio_password"
MINIO_ENDPOINT="http://localhost:9000"

# Application
NEXT_PUBLIC_APP_URL="https://yourdomain.com"
NODE_ENV="production"

# Monitoring
SENTRY_DSN="https://...@sentry.io/..."
POSTHOG_KEY="phc_..."
POSTHOG_HOST="https://app.posthog.com"

# Email (optional)
SMTP_HOST="smtp.example.com"
SMTP_PORT="587"
SMTP_USER="noreply@yourdomain.com"
SMTP_PASSWORD="smtp_password"
```

Listing A.1: .env.example

A.2 Common Commands Cheat Sheet

Task	Command
Start development	`npm run dev`
Build for production	`npm run build`
Start production	`npm start`
Run all tests	`npm run test:all`
Database migration	`npx prisma migrate deploy`
Generate Prisma client	`npx prisma generate`
Seed database	`npx prisma db seed`
Open Prisma Studio	`npx prisma studio`
Format code	`npm run format`
Lint code	`npm run lint`
Type check	`npm run type-check`
Docker Compose up	`docker-compose up -d`
Docker Compose logs	`docker-compose logs -f web`
PM2 status	`pm2 status`
PM2 logs	`pm2 logs web-app`
PM2 restart	`pm2 restart web-app`

Table A.1: Development and Operations Commands

Appendix B

Glossary

ACID Atomicity, Consistency, Isolation, Durability - database transaction properties

API Gateway Reverse proxy that routes requests to appropriate services

CDN Content Delivery Network - distributed servers for faster content delivery

CORS Cross-Origin Resource Sharing - security mechanism for browser requests

CRDT Conflict-free Replicated Data Type - for distributed data synchronization

CSP Content Security Policy - HTTP header preventing XSS attacks

JWT JSON Web Token - stateless authentication token format

MFA Multi-Factor Authentication - security using multiple verification methods

ORM Object-Relational Mapping - database abstraction layer (e.g., Prisma)

PWA Progressive Web App - web application with offline capabilities

RBAC Role-Based Access Control - permission system based on user roles

RLS Row-Level Security - PostgreSQL feature for data access control

SSR Server-Side Rendering - generating HTML on the server

STRIDE Threat modeling framework (Spoofing, Tampering, Repudiation, etc.)

TLS Transport Layer Security - encryption protocol for HTTPS

WCAG Web Content Accessibility Guidelines - standards for accessible web design

XSS Cross-Site Scripting - vulnerability allowing script injection

Appendix C

Additional Resources

C.1 Official Documentation

- Next.js: https://nextjs.org/docs
- Prisma: https://www.prisma.io/docs
- PostgreSQL: https://www.postgresql.org/docs/
- Stripe: https://stripe.com/docs
- Docker: https://docs.docker.com
- Kubernetes: https://kubernetes.io/docs

C.2 Security Resources

- OWASP Top 10: https://owasp.org/Top10/
- OWASP Cheat Sheet Series: https://cheatsheetseries.owasp.org
- CWE Top 25: https://cwe.mitre.org/top25/
- NIST Cybersecurity Framework: https://www.nist.gov/cyberframework

C.3 Performance Optimization

- Web.dev (Core Web Vitals): https://web.dev/vitals/
- Lighthouse: https://developers.google.com/web/tools/lighthouse
- WebPageTest: https://www.webpagetest.org