Zolar Drop 02 - Complete Development Documentation

Project: E-Commerce Platform for Exclusive Product Drops
Version: 1.0.0
Date: October 21, 2025
Developer Documentation

Table of Contents

---

1. Executive Summary

1.1 Project Overview

Zolar Drop 02 is a modern e-commerce platform built with Next.js 14, TypeScript, and Prisma ORM. The platform specializes in exclusive product drops with a complete admin management system, real-time order tracking, and seamless customer shopping experience.

1.2 Key Features

Customer Features:

- Browse product catalog with variant support
- Interactive product cards with image carousels
- Color/variant selection with real-time price updates
- Shopping cart with localStorage persistence
- Cash on Delivery (COD) checkout
- Responsive design for mobile and desktop

Admin Features:

- Secure dashboard with token authentication
- Real-time order notifications via Pusher
- Complete product CRUD operations

- Order status management workflow
- Multi-variant product support
- Image upload and management
- Live statistics dashboard

1.3 Technical Highlights

- **Framework:** Next.js 14 (Hybrid App Router + Pages Router)
- **Language:** TypeScript for type safety
- **Database:** SQLite with Prisma ORM
- **Real-Time:** Pusher WebSocket integration
- **Styling:** Tailwind CSS with custom utilities
- **State Management:** React Context API
- **Authentication:** Token-based (MVP)

---

2. Technology Stack

2.1 Frontend Stack

Technology	Version	Purpose
Next.js	14.0.4	React framework with SSR
React	18.x	UI library
TypeScript	5.x	Type safety
Tailwind CSS	3.3.0	Utility-first styling
Pusher JS	8.0.0	Client-side WebSocket

2.2 Backend Stack

Technology	Version	Purpose
Next.js API Routes	14.0.4	Backend endpoints
Prisma	5.7.1	ORM and migrations
SQLite	-	Database
Pusher	14.9.0	Server-side WebSocket
Node.js	20.x	Runtime environment

2.3 Development Tools

Tool	Version	Purpose
ESLint	8.x	Code linting
Prettier	3.0.0	Code formatting
TypeScript	5.x	Type checking
Jest	29.7.0	Unit testing
Playwright	1.40.0	E2E testing

2.4 Package Dependencies

```json
{
  "dependencies": {
    "next": "14.0.4",
    "react": "^18",
    "react-dom": "^18",
    "@prisma/client": "^5.7.1",
    "axios": "^1.6.2",
    "react-query": "^3.39.3",
    "pusher": "^14.9.0",
    "pusher-js": "^8.0.0",
    "node-fetch": "^3.3.0"
  },
  "devDependencies": {
    "typescript": "^5",
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "prisma": "^5.7.1",
    "tailwindcss": "^3.3.0",
    "postcss": "^8",
    "autoprefixer": "^10.0.1",
    "eslint": "^8",
    "eslint-config-next": "14.0.4",
    "prettier": "^3.0.0",
    "jest": "^29.7.0",
    "@types/jest": "^29.5.8",
    "playwright": "^1.40.0",
    "tsx": "^4.0.0"
  }
}
```

---

3. Architecture Overview

3.1 System Architecture

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                          PRESENTATION LAYER                                 │
│                                                                             │
│  ┌──────────────────┐  ┌──────────────────┐  ┌──────────────────┐           │
│  │   Storefront     │  │     Admin        │  │   Components     │           │
│  │   (Customer)     │  │   Dashboard      │  │    (Shared)      │           │
│  │                  │  │                  │  │                  │           │
│  │ /products        │  │ /admin           │  │ ProductCard      │           │
│  │ /product/[id]    │  │ /admin/          │  │ CartDrawer       │           │
│  │ /                │  │   products       │  │ CartIcon         │           │
│  └──────────────────┘  └──────────────────┘  └──────────────────┘           │
└─────────────────────────────────────────────────────────────────────────────┘
```

```
───────────────────────────────────────┘
                  │
                  ▼
┌──────────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────────────┐
│              API LAYER                     │
│                                       │
│  ┌─────────────────────────────┐    ┌─────────────────────────────┐
┌──────────────────────────────┐    │
│  │ Public APIs  │    │ Admin APIs  │    │ Utility APIs  │    │
│  │             │    │             │    │             │    │
│  │ /api/        │    │ /api/admin/  │    │ /api/upload  │    │
│  │ products     │    │ products     │    │             │    │
│  │ /api/        │    │ /api/admin/  │    │             │    │
│  │ checkout     │    │ orders       │    │             │    │
│  └─────────────────────────────┘    └─────────────────────────────┘
└──────────────────────────────┘    │
└──────────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────────────┘
                  │
                  ▼
┌──────────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────────────┐
│            BUSINESS LOGIC LAYER                 │
│                                       │
│  ┌─────────────────────────────┐    ┌─────────────────────────────┐
┌──────────────────────────────┐    │
│  │  Prisma     │    │  Pusher     │    │ Validation   │    │
│  │   ORM       │    │ WebSocket   │    │   Logic      │    │
│  │             │    │             │    │             │    │
│  │ - CRUD Ops  │    │ - Events     │    │ - Input      │    │
│  │ - Relations  │    │ - Channels   │    │ - Auth       │    │
│  │ - Migrations │    │ - Broadcasting │    │ - Business    │    │
│  └─────────────────────────────┘    └─────────────────────────────┘
└──────────────────────────────┘    │
└──────────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────────────┘
                  │
                  ▼
┌──────────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────────────┐
│              DATA LAYER                      │
│                                       │
│           ┌─────────────────────────────────────┐                │
│           │  SQLite DB    │              │
│           │              │              │
│           │ Tables:       │              │
│           │ - Product      │              │
│           │ - Variant      │              │
│           │ - Order        │              │
│           │ - User         │              │
│           └─────────────────────────────────────┘                │
```

```
└──────────────────────────────────────────────────────────────
──────────────────────────┘
```

3.2 Application Flow

Customer Journey

```
Home Page (/)
   ↓
Browse Products (/products)
   ↓
Select Product → View Details (/product/[slug])
   ↓
Choose Variant (Color)
   ↓
Add to Cart (Context State)
   ↓
Open Cart Drawer
   ↓
Fill Customer Info
   ↓
Submit COD Order (API Call)
   ↓
Order Confirmation
```

Admin Workflow

```
Login (/admin)
   ↓
Dashboard
   ├──> View Orders
   │    ├──> Real-time Notifications
   │    └──> Update Status (pending→confirmed→shipped→delivered)
   │
   └──> Manage Products (/admin/products)
      ├──> List Products
      ├──> Create New Product
      │    ├──> Basic Info
      │    ├──> Upload Images
      │    └──> Add Variants
      ├──> Edit Product
      └──> Delete Product
```

3.3 Design Patterns

1. Component-Based Architecture

- Reusable UI components
- Props-based data flow
- Separation of concerns

2. Context API Pattern

- CartContext for global cart state
- Provider wraps entire app
- Consumers access cart methods

3. Repository Pattern

- Prisma client as data access layer
- Centralized database operations
- Type-safe queries

4. API Route Handler Pattern

- Middleware for authentication
- Consistent error handling
- Structured JSON responses

5. Server-Side Rendering (SSR)

- Next.js App Router
- Static generation for performance
- Dynamic data fetching

---

4. Database Design

4.1 Schema Overview

The database uses Prisma ORM with SQLite and consists of 4 main models:

1. **Product** - Main product information
2. **Variant** - Product variants (colors/styles)
3. **Order** - Customer orders
4. **User** - User accounts

4.2 Complete Prisma Schema

```
// File: /prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

// ============================================
// PRODUCT MODEL
// ============================================
model Product {
  id          String    @id @default(uuid())
  sku          String    @unique
  title        String
  description    String?
  images         String?    // JSON: ["url1", "url2"]
  priceCents     Int        // Price in cents (2999 = $29.99)
  salePriceCents Int?       // Optional sale price
  currency       String     @default("USD")
  stock          Int        // Total stock across variants
  variants       Variant[]  // One-to-many relationship
  metadata       String?    // JSON: {"key": "value"}
  createdAt      DateTime   @default(now())
  updatedAt      DateTime   @updatedAt
}

// ============================================
// VARIANT MODEL (Colors/Styles)
// ============================================
model Variant {
  id         String   @id @default(uuid())
  product    Product  @relation(fields: [productId], references: [id])
  productId  String   // Foreign key
  color      String   // e.g., "Black", "White", "Navy"
  sku        String   @unique
  priceCents Int      // Variant-specific price
  stock      Int      // Variant-specific stock
  images     String?  // JSON: ["url1", "url2"]
  metadata   String?  // JSON: {"key": "value"}
  createdAt  DateTime @default(now())
  updatedAt  DateTime @updatedAt
}

// ============================================
// ORDER MODEL
// ============================================
model Order {
  id            String   @id @default(uuid())
  userId        String?  // Optional user reference
  items         String   // JSON: [{"productId":"...", "qty":2}]
  subtotalCents Int      // Sum of items
  taxCents      Int      // Tax amount
  shippingCents Int      // Shipping cost
  totalCents    Int      // Final total
  status        String   @default("pending")
  paymentMethod String   @default("COD")
  paymentId     String?  // Optional payment reference
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}

// ============================================
// USER MODEL
// ============================================
model User {
  id            String   @id @default(uuid())
  email        String   @unique
  name         String?
  role         String   @default("customer")
  passwordHash String?
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
}
```

4.3 Entity Relationships

```
┌─────────────────┐
│    Product      │
│ id (PK)         │ ───┐
│ sku (UNIQUE)    │    │
│ title           │    │
│ priceCents      │    │ One-to-Many
│ stock           │    │
└─────────────────┘    │
         │
         ▼
    ┌─────────────────┐
    │    Variant      │
    │ id (PK)         │
    │ productId (FK)  │
    │ color           │
    │ sku (UNIQUE)    │
    │ priceCents      │
    │ stock           │
    └─────────────────┘

┌─────────────────┐
│     Order       │
│ id (PK)         │
│ items (JSON)    │
│ totalCents      │
│ status          │
│ paymentMethod   │
└─────────────────┘

┌─────────────────┐
│     User        │
│ id (PK)         │
│ email (UNIQUE)  │
│ role            │
└─────────────────┘
```

4.4 JSON Field Structures

Product.images:

```
[
  "https://images.unsplash.com/photo-1.jpg",
  "https://images.unsplash.com/photo-2.jpg"
]
```

Variant.images:

```
[
  "https://cdn.example.com/black-variant.jpg"
]
```

Order.items:

```
[
  {
    "productId": "uuid-123",
    "variantId": "uuid-456",
    "qty": 2,
    "priceCents": 2999
  },
  {
    "productId": "uuid-789",
    "variantId": "uuid-012",
    "qty": 1,
    "priceCents": 5999
  }
]
```

4.5 Database Migrations

Migration History:

1. **Initial Migration** (20251021154716_init)
   - Created Product, Variant, Order, User tables
   - Set up relationships and constraints
2. **Nullable Images** (20251021155739_make_images_nullable)
   - Changed images from Json to String?
   - Made compatible with SQLite
3. **Payment Method** (20251021161308_add_payment_method)
   - Added paymentMethod field to Order
   - Default value: "COD"

Commands:

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

# Apply migrations
npx prisma migrate deploy

# Generate Prisma Client
npx prisma generate

# Seed database
npm run prisma:seed
```

4.6 Seed Data

Location: /prisma/seed.ts

Products Seeded:

1. Classic Cotton T-Shirt (4 variants)
2. Zip-Up Hoodie (4 variants)
3. Crew Neck Sweatshirt (4 variants)
4. Slim Fit Chinos (4 variants)

Total: 4 products, 16 variants

---

5. API Documentation

5.1 Public APIs

5.1.1 GET /api/products

Purpose: Fetch paginated product list with variants

Method: GET

Query Parameters:

- page (optional, default: 1) - Page number
- limit (optional, default: 10) - Items per page

Response Format:

```json
{
  "products": [
    {
      "id": "uuid",
      "sku": "TS-BASIC-001",
      "title": "Classic Cotton T-Shirt",
      "description": "Premium 100% cotton...",
      "images": "[\"url1\", \"url2\"]",
      "priceCents": 2999,
      "salePriceCents": null,
      "currency": "USD",
      "stock": 100,
      "metadata": null,
      "createdAt": "2025-10-21T...",
      "updatedAt": "2025-10-21T...",
      "variants": [
        {
          "id": "uuid",
          "productId": "uuid",
          "color": "Black",
          "sku": "TS-BASIC-001-BLK",
          "priceCents": 2999,
          "stock": 25,
          "images": "[\"url\"]",
          "metadata": null,
          "createdAt": "2025-10-21T...",
          "updatedAt": "2025-10-21T..."
        }
      ]
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 10,
    "total": 4,
    "pages": 1
  }
}
```

Status Codes:

- 200: Success
- 500: Internal server error

5.1.2 GET /api/products/[id]

Purpose: Fetch single product with all variants

Method: GET

URL Parameters:

- id (required) - Product UUID or SKU

Response Format:

```
{
  "product": {
    "id": "uuid",
    "sku": "TS-BASIC-001",
    "title": "Classic Cotton T-Shirt",
    "description": "Premium 100% cotton...",
    "images": "[...]",
    "priceCents": 2999,
    "stock": 100,
    "variants": [...]
  }
}
```

Status Codes:

- 200: Success
- 404: Product not found
- 500: Internal server error

5.1.3 POST /api/checkout/cod

Purpose: Place Cash on Delivery order

Method: POST

Headers:

```
Content-Type: application/json
```

Request Body:

```
{
  "items": [
    {
      "productId": "uuid",
      "variantId": "uuid",
      "qty": 2
    }
  ],
  "customer": {
    "name": "John Doe",
    "email": "john@example.com",
    "address": "123 Main Street, City, State 12345",
    "phone": "555-123-4567"
  },
  "shippingCents": 0
}
```

Validation Rules:

- All customer fields required
- Email must be valid format
- Quantity must be positive
- Stock must be available

Response Format:

```
{
  "success": true,
  "orderId": "uuid",
  "message": "Order placed successfully",
  "order": {
    "id": "uuid",
    "subtotalCents": 5998,
    "taxCents": 0,
    "shippingCents": 0,
    "totalCents": 5998,
    "status": "pending",
    "paymentMethod": "COD"
  }
}
```

Error Response:

```
{
  "message": "Insufficient stock for Classic T-Shirt (Black). Available: 10, requested: 15"
}
```

Status Codes:

- 201: Order created successfully
- 400: Validation error or insufficient stock
- 404: Product/variant not found
- 500: Internal server error

Side Effects:

1. Order created in database
2. Stock decremented for all variants
3. Pusher event broadcast to admin dashboard
4. Email notification (future enhancement)

5.2 Admin APIs

Authentication Required: All admin endpoints require Bearer token

Header Format:

```
Authorization: Bearer admin-token-123
```

5.2.1 GET /api/admin/orders

Purpose: Fetch all orders (admin only)

Method: GET

Headers:

```
Authorization: Bearer admin-token-123
```

Response Format:

```
{
  "orders": [
    {
      "id": "uuid",
      "userId": null,
      "items": "[{\"productId\":\"...\",\"qty\":2}]",
      "subtotalCents": 5998,
      "taxCents": 0,
      "shippingCents": 0,
      "totalCents": 5998,
      "status": "pending",
      "paymentMethod": "COD",
      "paymentId": null,
      "createdAt": "2025-10-21T...",
      "updatedAt": "2025-10-21T..."
    }
  ]
}
```

Status Codes:

- 200: Success
- 401: Unauthorized (invalid/missing token)
- 500: Internal server error

5.2.2 PATCH /api/admin/orders/[id]

Purpose: Update order status

Method: PATCH

Headers:

```
Authorization: Bearer admin-token-123
Content-Type: application/json
```

URL Parameters:

- id (required) - Order UUID

Request Body:

```
{
  "status": "confirmed"
}
```

Valid Status Values:

- pending - Initial state
- confirmed - Admin confirmed
- shipped - Order shipped
- delivered - Order delivered
- cancelled - Order cancelled

Response Format:

```
{
  "success": true,
  "order": {
    "id": "uuid",
    "status": "confirmed",
    "updatedAt": "2025-10-21T..."
  },
  "message": "Order status updated to confirmed"
}
```

Status Codes:

- 200: Success
- 400: Invalid status value
- 401: Unauthorized
- 404: Order not found
- 500: Internal server error

5.2.3 POST /api/admin/products

Purpose: Create new product with variants

Method: POST

Headers:

```
Authorization: Bearer admin-token-123
Content-Type: application/json
```

Request Body:

```json
{
  "title": "New Product",
  "description": "Product description",
  "sku": "PROD-001",
  "priceCents": 4999,
  "stock": 100,
  "images": [
    "https://cdn.example.com/image1.jpg",
    "https://cdn.example.com/image2.jpg"
  ],
  "variants": [
    {
      "color": "Black",
      "sku": "PROD-001-BLK",
      "priceCents": 4999,
      "stock": 50,
      "images": ["https://cdn.example.com/black.jpg"]
    },
    {
      "color": "White",
      "sku": "PROD-001-WHT",
      "priceCents": 4999,
      "stock": 50,
      "images": ["https://cdn.example.com/white.jpg"]
    }
  ]
}
```

Validation Rules:

- Title, SKU, price, stock required
- SKU must be unique
- Images array required (min 1)
- Price/stock must be positive
- Variant SKUs must be unique

Response Format:

```json
{
  "product": {
    "id": "uuid",
    "sku": "PROD-001",
    "title": "New Product",
    "priceCents": 4999,
    "stock": 100,
    "variants": [...]
  }
}
```

Status Codes:

- 201: Product created
- 400: Validation error

- 401: Unauthorized
- 409: SKU already exists
- 500: Internal server error

5.2.4 PUT /api/admin/products/[id]

Purpose: Update existing product

Method: PUT

Headers:

```
Authorization: Bearer admin-token-123
Content-Type: application/json
```

URL Parameters:

- id (required) - Product UUID

Request Body: (same as POST, all fields optional)

Behavior:

- Updates product fields
- Deletes old variants
- Creates new variants
- Cascading update

Status Codes:

- 200: Product updated
- 400: Validation error
- 401: Unauthorized
- 404: Product not found
- 409: SKU conflict
- 500: Internal server error

5.2.5 DELETE /api/admin/products/[id]

Purpose: Delete product and all variants

Method: DELETE

Headers:

```
Authorization: Bearer admin-token-123
```

URL Parameters:

- id (required) - Product UUID

Response Format:

```
{
  "success": true,
  "message": "Product and all variants deleted successfully"
}
```

Status Codes:

- 200: Product deleted
- 401: Unauthorized
- 404: Product not found
- 500: Internal server error

Side Effects:

1. All variants deleted (cascading)
2. Product deleted from database
3. Removed from storefront immediately

5.2.6 POST /api/upload

Purpose: Upload image (simulated for MVP)

Method: POST

Headers:

```
Content-Type: application/json
```

Request Body:

```
{
  "filename": "product-image.jpg"
}
```

Response Format:

```
{
  "success": true,
  "url": "https://cdn.example.com/products/1234567890-abc123.jpg",
  "message": "Image uploaded successfully (simulated)"
}
```

Notes:

- Currently returns fake CDN URL
- Simulates 500ms delay
- In production, replace with real S3/Cloudinary upload

Status Codes:

- 200: Success
- 405: Method not allowed
- 500: Upload failed

---

6. Frontend Implementation

6.1 Page Routes

Public Pages (App Router)

1. Home Page: /app/page.tsx

Purpose: Landing page with hero and CTA

Features:

- Hero section with "Drop 02" branding
- "Shop Collection" CTA button
- Feature highlights grid
- Navigation with cart icon

Layout:

```
┌─────────────────────────────────────────┐
│  [Logo]              [Cart Icon]     │
├─────────────────────────────────────────┤
│                                 │
│            Drop 02                    │
│   Exclusive limited edition          │
│   [Shop Collection →]                │
│                                 │
├─────────────────────────────────────────┤
│  ⚡                                    │
│  Limited    Premium     Fast         │
│  Drops       Quality     Shipping   │
└─────────────────────────────────────────┘
```

---

2. Products Page: /app/products/page.tsx

Purpose: Product listing grid

Data Fetching:

```
async function getProducts(): Promise<Product[]> {
  const baseUrl = process.env.VERCEL_URL
    ? `https://${process.env.VERCEL_URL}`
    : 'http://localhost:3000'

  const res = await fetch(`${baseUrl}/api/products`, {
    cache: 'no-store'
  })

  const data = await res.json()
  return data.products
}
```

Features:

- Server-side product fetching
- Grid layout (responsive)
- ProductCard components
- Cart icon in header

Layout:

```
┌─────────────────────────────────────────────┐
│  Products          [Cart Icon]   │
├─────────────────────────────────────────────┤
│   ┌────────┐  ┌────────┐  ┌────────┐  ┌───────┐  │
│   │  Img  │   │  Img  │   │  Img  │   │ Img  │  │
│   │ Title │   │ Title │   │ Title │   │ Ttle │  │
│   │ $$ ●● │   │ $$ ●● │   │ $$ ●● │   │ $$●● │  │
│   │ [Add] │   │ [Add] │   │ [Add] │   │ [Ad] │  │
│   └────────┘  └────────┘  └────────┘  └───────┘  │
└─────────────────────────────────────────────┘
```

3. Product Detail: /app/product/[slug]/page.tsx

Purpose: Single product view with variants

Dynamic Route: Uses product SKU as slug

Features:

- Image gallery with thumbnails
- Variant selector (color swatches)
- Stock indicator
- Price display
- Add to cart button
- Breadcrumb navigation

Client-Side State:

```
const [selectedVariant, setSelectedVariant] = useState(variants[0])
const [currentImageIndex, setCurrentImageIndex] = useState(0)
```

Layout:

```
┌─────────────────────────────────────────────────┐
│  Home / Products / T-Shirt  [Cart Icon] │
├──────────────────────┬──────────────────────────┤
│               │  Classic T-Shirt     │
│               │  $29.99              │
│  [Main Image]   │                      │
│               │  Color:              │
│               │  ● ○ ○ ○             │
│ [●][○][○][○]   │                      │
│  Thumbnails    │  25 in stock         │
│               │                      │
│               │  [Add to Cart]       │
└──────────────────────┴──────────────────────────┘
```

Admin Pages (App Router)

4. Admin Dashboard: /app/admin/page.tsx

Purpose: Order management and real-time monitoring

Authentication:

```
useEffect(() => {
  const authToken = sessionStorage.getItem('admin_token')
  if (authToken === 'admin-token-123') {
    setIsAuthenticated(true)
    fetchOrders()
  }
}, [])
```

Features:

- Login screen (password protection)
- Statistics cards
- Real-time order table
- Status update buttons
- Pusher WebSocket connection
- Toast notifications
- Logout button
- "Manage Products" link

Status Workflow:

```
const updateOrderStatus = async (orderId, newStatus) => {
  await fetch(`/api/admin/orders/${orderId}`, {
    method: 'PATCH',
    headers: {
      'Authorization': `Bearer ${sessionStorage.getItem('admin_token')}`
    },
    body: JSON.stringify({ status: newStatus })
  })
}
```

---

5. Product Management: /app/admin/products/page.tsx

Purpose: Product list with edit/delete

Features:

- Product table with thumbnails
- Edit/Delete buttons
- "+ Add New Product" button
- Loading states
- Empty state
- Confirmation dialogs

Delete Handler:

```
const handleDelete = async (productId, productTitle) => {
  if (!confirm(`Delete "${productTitle}"?`)) return

  await fetch(`/api/admin/products/${productId}`, {
    method: 'DELETE',
    headers: { 'Authorization': `Bearer ${token}` }
  })

  // Remove from local state
  setProducts(products.filter(p => p.id !== productId))
}
```

---

6. Create Product: /app/admin/products/new/page.tsx

Purpose: Product creation form

Form Sections:

1. Basic Information
   - Title, Description, SKU
   - Price, Stock
2. Product Images
   - Multi-file upload
   - Preview grid
   - Remove buttons
3. Variants
   - Dynamic form
   - Color, SKU, Price, Stock
   - Variant images
   - Add/Remove variants

Image Upload:

```
const handleImageUpload = async (files) => {
  const uploadedUrls = []

  for (let i = 0; i < files.length; i++) {
    const res = await fetch('/api/upload', {
      method: 'POST',
      body: JSON.stringify({ filename: files[i].name })
    })
    const data = await res.json()
    uploadedUrls.push(data.url)
  }

  setImages([...images, ...uploadedUrls])
}
```

Submit Handler:

```
const handleSubmit = async (e) => {
  e.preventDefault()

  await fetch('/api/admin/products', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`
    },
    body: JSON.stringify({
      title,
      sku,
      priceCents: Math.round(parseFloat(price) * 100),
      stock: parseInt(stock),
      images,
      variants
    })
  })

  router.push('/admin/products')
}
```

---

7. Edit Product: /app/admin/products/[id]/page.tsx

Purpose: Update existing product

Same as Create, but:

- Pre-populated fields
- Fetches existing data on mount
- PUT instead of POST
- Cascade updates to variants

6.2 Components

6.2.1 ProductCard Component

Location: /components/ProductCard.tsx

Purpose: Reusable product display card

Props:

```
interface ProductCardProps {
  product: Product
}
```

Features:

- Image carousel
- Color swatches
- Price display

- Stock indicator
- Add to cart button
- Hover effects

State Management:

```
const [selectedVariant, setSelectedVariant] = useState(variants[0])
const [currentImageIndex, setCurrentImageIndex] = useState(0)
```

Add to Cart Integration:

```
const { addItem } = useCart()

const handleAddToCart = () => {
  addItem({
    productId: product.id,
    variantId: selectedVariant.id,
    qty: 1,
    priceCents: selectedVariant.priceCents,
    title: product.title,
    image: variantImages[0],
    variantName: selectedVariant.color
  })
}
```

6.2.2 CartContext Component

Location: /components/CartContext.tsx

Purpose: Global cart state management

Context Structure:

```
interface CartItem {
  productId: string
  variantId: string
  qty: number
  priceCents: number
  title: string
  image: string
  variantName: string
}

interface CartState {
  items: CartItem[]
  isOpen: boolean
}

interface CartContextType {
  state: CartState
  addItem: (item: CartItem) => void
  removeItem: (variantId: string) => void
  updateQuantity: (variantId: string, qty: number) => void
  clearCart: () => void
  toggleCart: () => void
  getSubtotal: () => number
  getItemCount: () => number
}
```

localStorage Integration:

```
useEffect(() => {
  const saved = localStorage.getItem('cart')
  if (saved) {
    setState(JSON.parse(saved))
  }
}, [])

useEffect(() => {
  localStorage.setItem('cart', JSON.stringify(state))
}, [state])
```

6.2.3 CartDrawer Component

Location: /components/CartDrawer.tsx

Purpose: Slide-out cart UI with checkout

Features:

- Slide animation from right
- Item list with thumbnails
- Quantity controls (+/-)
- Remove item buttons
- Subtotal display

- Checkout form
- Empty state

Checkout Flow:

```
const handleCheckout = async (e) => {
  e.preventDefault()

  const response = await fetch('/api/checkout/cod', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      items: state.items.map(item => ({
        productId: item.productId,
        variantId: item.variantId,
        qty: item.qty
      })),
      customer: customerInfo
    })
  })

  if (response.ok) {
    setOrderSuccess(true)
    clearCart()
  }
}
```

6.2.4 CartIcon Component

Location: /components/CartIcon.tsx

Purpose: Header cart icon with badge

Features:

- Shopping bag SVG icon
- Item count badge
- Click to toggle drawer
- Responsive positioning

```
const { state, toggleCart, getItemCount } = useCart()

return (
  <button onClick={toggleCart}>

    {getItemCount() > 0 && (
      <span className="badge">{getItemCount()}</span>
    )}
  </button>
)
```

6.3 Styling System

Tailwind Configuration

File: /tailwind.config.js

```
module.exports = {
  content: [
    './pages/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
    './app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {
      colors: {
        // Custom colors
      },
      animation: {
        'slide-in': 'slideIn 0.3s ease-out',
      },
    },
  },
  plugins: [],
}
```

Global Styles

File: /app/globals.css

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Custom utilities */
@layer utilities {
  .line-clamp-2 {
    overflow: hidden;
    display: -webkit-box;
    -webkit-box-orient: vertical;
    -webkit-line-clamp: 2;
  }

  .border-3 {
    border-width: 3px;
  }

  .bg-gradient-radial {
    background: radial-gradient(ellipse at center, var(--tw-gradient-stops));
  }

  .animate-slide-in {
    animation: slideIn 0.3s ease-out;
  }

  @keyframes slideIn {
    from {
      transform: translateX(100%);
      opacity: 0;
    }
    to {
      transform: translateX(0);
      opacity: 1;
    }
  }
}
```

Color System

Status Colors:

- Pending: Yellow (bg-yellow-100 text-yellow-800)
- Confirmed: Blue (bg-blue-100 text-blue-800)
- Shipped: Purple (bg-purple-100 text-purple-800)
- Delivered: Green (bg-green-100 text-green-800)

Variant Colors:

```
const colorMap = {
  'Black': 'bg-gray-900',
  'White': 'bg-gray-100 border border-gray-300',
  'Red': 'bg-red-600',
  'Blue': 'bg-blue-600',
  'Navy': 'bg-blue-800',
  'Gray': 'bg-gray-500',
  'Heather Gray': 'bg-gray-400',
  'Burgundy': 'bg-red-800',
  'Cream': 'bg-yellow-200',
  'Forest Green': 'bg-green-800',
  'Khaki': 'bg-yellow-600',
  'Olive': 'bg-green-700',
}
```

---

7. Admin Panel Features

7.1 Authentication System

Type: Token-based (MVP)

Flow:

```
1. Admin visits /admin
2. Enters password "admin-token-123"
3. Token stored in sessionStorage
4. All API calls include Authorization header
5. Server validates token
```

Implementation:

```
// Login
const handleLogin = (password) => {
  if (password === 'admin-token-123') {
    sessionStorage.setItem('admin_token', password)
    setIsAuthenticated(true)
  }
}

// API Request
fetch(url, {
  headers: {
    'Authorization': `Bearer ${sessionStorage.getItem('admin_token')}`
  }
})

// Server Validation
const token = req.headers.authorization?.replace('Bearer ', '')
if (token !== 'admin-token-123') {
  return res.status(401).json({ message: 'Unauthorized' })
}
```

7.2 Order Management

Features:

1. **Order List**
    - Fetches from /api/admin/orders
    - Sorted by newest first
    - Shows all order details
2. **Status Updates**
    - One-click buttons
    - Contextual actions based on current status
    - Immediate UI feedback
3. **Statistics Dashboard**
    - Total Orders
    - Pending Orders
    - COD Orders
    - Auto-updating counters
4. **Real-Time Notifications**
    - Pusher WebSocket
    - Toast alerts
    - Auto-refresh list

7.3 Product Management

Complete CRUD Operations:

1. **Create Product**
    - Form with validation
    - Multi-image upload
    - Variant management
    - Instant storefront visibility
2. **Read/List Products**
    - Table view with thumbnails
    - Sortable columns
    - Search (future)
3. **Update Product**
    - Pre-populated form
    - Edit all fields
    - Cascading variant updates
4. **Delete Product**

    - Confirmation dialog
    - Cascading delete
    - Immediate removal

---

8. Real-Time System

8.1 Pusher Integration

Purpose: Real-time notifications for new orders

Architecture:

```
Customer Order
   ↓
API creates order
   ↓
Pusher.trigger('admin-orders', 'new-order', data)
   ↓
Admin Dashboard listens
   ↓
Toast notification
   ↓
Auto-refresh
```

Server Setup:

```
// /lib/pusher.ts
import Pusher from 'pusher'

export const pusherServer = new Pusher({
  appId: process.env.PUSHER_APP_ID,
  key: process.env.PUSHER_KEY,
  secret: process.env.PUSHER_SECRET,
  cluster: process.env.PUSHER_CLUSTER,
  useTLS: true
})
```

Client Setup:

```
// /lib/pusher-client.ts
import PusherClient from 'pusher-js'

export const pusherClient = new PusherClient(
  process.env.NEXT_PUBLIC_PUSHER_KEY,
  {
    cluster: process.env.NEXT_PUBLIC_PUSHER_CLUSTER
  }
)
```

Broadcasting:

```
// In COD endpoint
await pusherServer.trigger('admin-orders', 'new-order', {
  id: order.id,
  totalCents: order.totalCents,
  paymentMethod: 'COD',
  customer: customer.name,
  createdAt: order.createdAt,
  itemCount: items.length
})
```

Listening:

```
// In admin dashboard
useEffect(() => {
  const channel = pusherClient.subscribe('admin-orders')

  channel.bind('new-order', (data) => {
    setToast('    New COD order received!')
    fetchOrders()
  })

  return () => {
    channel.unbind('new-order')
    pusherClient.unsubscribe('admin-orders')
  }
}, [])
```

9. Security Implementation

9.1 Authentication

Current (MVP):

- Simple token comparison
- sessionStorage persistence
- Authorization header

Future Recommendations:

- NextAuth.js integration
- JWT tokens with expiration
- Refresh token rotation
- Role-based access control (RBAC)

9.2 Authorization

Admin Routes:

- All /api/admin/* endpoints protected
- Token validation on every request
- 401 response for invalid tokens

9.3 Input Validation

Client-Side:

- Required field checking
- Email format validation
- Number type validation
- SKU uniqueness checking

Server-Side:

- Type validation
- Range validation (price > 0)
- SKU uniqueness (database constraint)
- Stock availability checks

9.4 Data Sanitization

JSON Fields:

- Images stored as JSON strings
- Parsed safely with try/catch
- Type checking after parse

SQL Injection Prevention:

- Prisma parameterized queries
- No raw SQL
- Type-safe database access

---

10. Testing & Validation

10.1 Manual Testing Results

Product CRUD:

- Create product with variants
- Update product details
- Delete product and variants
- Product appears on storefront

Order Flow:

- Place COD order
- Stock decremented
- Order in database
- Real-time notification
- Status updates

Cart System:

- Add to cart
- Update quantity
- Remove items

- Persist across reloads
- Subtotal calculation

10.2 API Testing

Commands Used:

```
# Fetch products
curl http://localhost:3000/api/products

# Place order
curl -X POST http://localhost:3000/api/checkout/cod \
  -H "Content-Type: application/json" \
  -d '{"items":[...], "customer":{...}}'

# Create product (admin)
curl -X POST http://localhost:3000/api/admin/products \
  -H "Authorization: Bearer admin-token-123" \
  -d '{"title":"...", "sku":"..."}'

# Update order status
curl -X PATCH http://localhost:3000/api/admin/orders/[id] \
  -H "Authorization: Bearer admin-token-123" \
  -d '{"status":"confirmed"}'
```

10.3 Test Data

Seeded Products: 4 products, 16 variants
Test Orders: Multiple COD orders placed
Test Users: Admin user functional

---

11. Deployment Guide

11.1 Environment Variables

Create .env file:

```
# Database
DATABASE_URL="file:./dev.db"

# Admin
ADMIN_TOKEN="admin-token-123"

# Pusher (optional for MVP)
PUSHER_APP_ID="your-app-id"
PUSHER_KEY="your-key"
PUSHER_SECRET="your-secret"
PUSHER_CLUSTER="us2"
NEXT_PUBLIC_PUSHER_KEY="your-key"
NEXT_PUBLIC_PUSHER_CLUSTER="us2"
```

11.2 Build & Start

```
# Install dependencies
npm install

# Generate Prisma Client
npx prisma generate

# Run migrations
npx prisma migrate deploy

# Seed database
npm run prisma:seed

# Build for production
npm run build

# Start production server
npm start
```

11.3 Deployment Platforms

Vercel (Recommended):

1. Connect GitHub repository
2. Add environment variables
3. Deploy automatically

Other Platforms:

- Railway
- Netlify
- AWS Amplify
- DigitalOcean App Platform

---

12. File Structure

```
zolar2.0/
├── app/                    # Next.js App Router
│   ├── admin/              # Admin pages
│   │   ├── page.tsx        # Dashboard
│   │   └── products/       # Product management
│   │       ├── page.tsx    # Product list
│   │       ├── new/        # Create product
│   │       │   └── page.tsx
│   │       └── [id]/       # Edit product
│   │           └── page.tsx
│   ├── product/            # Product detail
│   │   └── [slug]/
│   │       └── page.tsx
│   ├── products/           # Product listing
```

```
|     |    └─── page.tsx
|     ├─── layout.tsx          # Root layout
|     ├─── page.tsx            # Homepage
|     └─── globals.css         # Global styles
|
├─── components/               # Shared components
|     ├─── CartContext.tsx     # Cart state
|     ├─── CartDrawer.tsx      # Cart UI
|     ├─── CartIcon.tsx        # Cart icon
|     └─── ProductCard.tsx     # Product card
|
├─── lib/                      # Utilities
|     ├─── prisma.ts           # Prisma client
|     ├─── pusher.ts           # Pusher server
|     └─── pusher-client.ts    # Pusher client
|
├─── pages/                    # Next.js Pages Router
|     └─── api/                # API routes
|          ├─── admin/         # Admin APIs
|          |     ├─── orders/
|          |     |     ├─── index.ts   # List orders
|          |     |     └─── [id].ts    # Update order
|          |     └─── products/
|          |          ├─── index.ts    # Create product
|          |          └─── [id].ts     # Update/Delete product
|          ├─── checkout/
|          |     └─── cod.ts           # COD checkout
|          ├─── products/
|          |     ├─── index.ts         # List products
|          |     └─── [id].ts          # Get product
|          └─── upload.ts              # Image upload
|
├─── prisma/                   # Database
|     ├─── schema.prisma       # Database schema
|     ├─── seed.ts             # Seed script
|     ├─── dev.db              # SQLite database
|     └─── migrations/         # Migration files
|
├─── public/                   # Static assets
|
├─── .env                      # Environment variables
├─── .gitignore                # Git ignore
├─── next.config.js            # Next.js config
├─── package.json              # Dependencies
├─── postcss.config.js         # PostCSS config
├─── tailwind.config.js        # Tailwind config
├─── tsconfig.json             # TypeScript config
├─── README.md                 # Project readme
|
└─── Documentation/            # Project docs
   ├─── ADMIN_PRODUCTS_GUIDE.md
   ├─── ADMIN_WORKFLOW_TEST.md
```

```
├─── IMPLEMENTATION_SUMMARY.md
├─── PRODUCT_MANAGEMENT_SUMMARY.md
└─── PUSHER_SETUP.md
```

Appendix A: Key Code Snippets

A.1 Prisma Client Usage

```
// Create product with variants
const product = await prisma.product.create({
  data: {
    sku: 'TS-001',
    title: 'T-Shirt',
    priceCents: 2999,
    stock: 100,
    images: JSON.stringify(['url1', 'url2']),
    variants: {
      create: [
        {
          color: 'Black',
          sku: 'TS-001-BLK',
          priceCents: 2999,
          stock: 25,
          images: JSON.stringify(['url'])
        }
      ]
    }
  },
  include: { variants: true }
})
```

A.2 Cart Context Implementation

```
const CartContext = createContext<CartContextType | undefined>(undefined)

export function CartProvider({ children }) {
  const [state, setState] = useState<CartState>({
    items: [],
    isOpen: false
  })

  const addItem = (item: CartItem) => {
    setState(prev => ({
      ...prev,
      items: [...prev.items, item]
    }))
  }

  return (
    <CartContext.Provider value={{ state, addItem, ... }}>
      {children}
    </CartContext.Provider>
  )
}
```

A.3 API Route Handler

```
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' })
  }

  // Authentication
  const token = req.headers.authorization?.replace('Bearer ', '')
  if (token !== 'admin-token-123') {
    return res.status(401).json({ message: 'Unauthorized' })
  }

  // Business logic
  try {
    const result = await prisma.product.create({...})
    res.status(201).json({ product: result })
  } catch (error) {
    res.status(500).json({ message: 'Internal server error' })
  }
}
```

Appendix B: Environment Setup

B.1 Development Setup

```
# Clone repository
git clone <repo-url>
cd zolar2.0

# Install dependencies
npm install

# Setup database
npx prisma generate
npx prisma migrate dev
npm run prisma:seed

# Start development server
npm run dev
```

B.2 Production Setup

```
# Build application
npm run build

# Start production server
npm start
```

Appendix C: Common Commands

```
# Development
npm run dev                  # Start dev server
npm run build                # Build for production
npm start                    # Start production server
npm run lint                 # Run ESLint

# Database
npx prisma generate          # Generate Prisma client
npx prisma migrate dev       # Run migrations
npx prisma migrate reset     # Reset database
npm run prisma:seed          # Seed database
npx prisma studio            # Open Prisma Studio

# Testing
npm test                     # Run tests (future)
curl http://localhost:3000/api/products  # Test API
```

Conclusion

This documentation covers the complete implementation of the Zolar Drop 02 e-commerce platform. The system includes:

Complete Product Catalog with variants
Shopping Cart System with persistence
COD Checkout Flow with stock management
Admin Dashboard with real-time notifications
Product Management with full CRUD operations
Real-Time Updates via Pusher WebSocket
Responsive Design for all devices
Type-Safe implementation with TypeScript
Production-Ready architecture

Access Points:

- Storefront: http://localhost:3000
- Admin: http://localhost:3000/admin
- API Docs: This document

For Support:

- Review documentation files in /Documentation
- Check code comments in source files
- Test with provided curl commands

End of Documentation