FashionVoid E-Commerce Platform

Complete Technical Documentation

Version: 1.0.1
Author: Petar Vukovic
Live Site: https://www.fashionvoid.net
Date: January 2025

Table of Contents

Executive Summary

FashionVoid is a production-ready, full-stack e-commerce platform built from scratch. It features a complete shopping experience with product management, secure authentication, multiple payment gateways, order processing, and a comprehensive admin dashboard. The platform is deployed live and has been tested with 15+ users successfully signing up and making purchases.

Key Achievements:

- Built complete e-commerce platform from scratch
- Integrated Stripe and PayPal payment gateways
- Implemented secure authentication with role-based access control
- Created intuitive admin dashboard with full CRUD operations
- Deployed to production (Railway) with HTTPS/SSL
- Mobile-responsive design with smooth animations
- Real-time inventory management
- Review system with admin approval workflow

Project Overview

Purpose

A modern e-commerce webshop for an Instagram fashion business, allowing customers to browse products, add items to cart, complete purchases, and manage their accounts. Admins can manage products, orders, users, and view analytics.

Target Users

- **Customers:** Browse products, make purchases, manage profiles
- **Admins:** Manage inventory, process orders, view analytics

Core Value Proposition

- Complete e-commerce solution with modern UI/UX
- Secure payment processing
- Comprehensive admin tools
- Mobile-first responsive design
- Production-ready deployment

Technology Stack

Programming Languages

1. **TypeScript** (Primary Language)
    - Type-safe JavaScript
    - Used throughout entire codebase
    - Prevents runtime errors with compile-time checking
2. **JavaScript** (Runtime)
    - Node.js runtime environment
    - Server-side execution
    - Client-side React components
3. **SQL** (Database Queries)
    - MySQL database queries
    - Prisma ORM generates type-safe queries
    - Raw SQL for migrations
4. **CSS** (Styling)
    - Tailwind CSS utility classes
    - Custom CSS for animations
    - Responsive design utilities

Frontend Technologies

Technology	Version	Purpose
Next.js	14.2.35	React framework with App Router, SSR, API routes
React	18.2.0	UI component library
TypeScript	5.3.3	Type safety and developer experience

Tailwind CSS	3.4.0	Utility-first CSS framework
Framer Motion	10.16.16	Animation library for smooth transitions
Zustand	4.4.7	Lightweight state management
React Hot Toast	2.4.1	Toast notification system
Lucide React	0.303.0	Icon library

Backend Technologies

Technology	Version	Purpose
Next.js API Routes	14.2.35	Serverless API endpoints
Prisma	6.19.1	Type-safe ORM for database operations
NextAuth.js	4.24.13	Authentication and session management
bcryptjs	3.0.3	Password hashing and security
MySQL2	3.15.3	MySQL database driver

Services & Integrations

Service	Purpose
Cloudinary	Image hosting, optimization, and CDN
Stripe	Payment processing and checkout
PayPal	Alternative payment gateway
Nodemailer	Email sending for order confirmations
Railway	Cloud hosting and deployment platform

Development Tools

Tool	Purpose
ESLint	Code linting and quality
PostCSS	CSS processing
Autoprefixer	CSS vendor prefixing
Sharp	Image optimization
tsx	TypeScript execution

Complete Feature List

E-Commerce Core Features

Product Management

- ✅ Product catalog with image galleries
- ✅ Product variants (sizes, colors) with individual inventory
- ✅ Featured products system (up to 4 featured items)
- ✅ Product categories (tops, bottoms, footwear, accessories)
- ✅ Product search and filtering
- ✅ Out-of-stock management
- ✅ Product detail pages with scrollable image galleries
- ✅ Archive collection display

Shopping Cart

- ✅ Persistent shopping cart (saved to database)
- ✅ Add/remove items
- ✅ Quantity management
- ✅ Size and color selection
- ✅ Cart synchronization across devices
- ✅ Real-time cart updates
- ✅ Cart persistence after logout/login

Checkout Process

- ✅ Multi-step checkout flow (3 steps)
- ✅ Address management (add, edit, delete)
- ✅ Shipping cost calculation
- ✅ VAT/tax calculation
- ✅ Order summary with breakdown
- ✅ Discount code application
- ✅ Payment method selection (Stripe/PayPal)

Order Management

- ✅ Order creation and tracking
- ✅ Order status updates (pending, processing, shipped, delivered, completed, cancelled, refunded)
- ✅ Order history for customers
- ✅ Order details with items
- ✅ Order number generation
- ✅ Email confirmations
- ✅ Admin order management dashboard

Wishlist

- ✅ Add/remove products to wishlist
- ✅ View wishlist page
- ✅ Wishlist persistence
- ✅ Quick add to cart from wishlist

User Features

Authentication & Authorization

- ✅ User registration with email/password
- ✅ Secure login with NextAuth.js
- ✅ Password hashing with bcryptjs
- ✅ Session management
- ✅ Role-based access control (Admin/Customer)

- ✅ Protected routes
- ✅ Password reset functionality (admin)

User Profiles

- ✅ Profile page with user information
- ✅ Profile picture upload (Cloudinary)
- ✅ Display name customization
- ✅ Bio and location fields
- ✅ Public profile pages (`/u/[id]`)
- ✅ User verification badges (for trusted users)
- ✅ Profile picture removal

User Management (Admin)

- ✅ View all users
- ✅ Edit user profiles
- ✅ Remove user profile pictures
- ✅ Mark users as verified
- ✅ User search functionality
- ✅ View user creation dates

Admin Dashboard Features

Product Management

- ✅ Add new products with multiple images
- ✅ Edit existing products
- ✅ Delete products
- ✅ Upload images via Cloudinary
- ✅ Manage product variants (sizes, colors)
- ✅ Set inventory quantities
- ✅ Toggle featured status (max 4)
- ✅ Set product categories
- ✅ Product search and filtering
- ✅ Bulk operations

Order Management

- ✅ View all orders
- ✅ Filter orders by status
- ✅ Update order status
- ✅ View order details
- ✅ Track order progress
- ✅ Order analytics

Inventory Management

- ✅ Low stock alerts
- ✅ Quantity updates
- ✅ Stock status toggling
- ✅ Inventory tracking per variant

Discount Code Management

- ✅ Create discount codes
- ✅ Set discount type (percentage/fixed)
- ✅ Set minimum purchase requirements

- ✅ Set maximum discount limits
- ✅ Set usage limits
- ✅ Set validity dates
- ✅ Activate/deactivate codes
- ✅ View usage statistics

Review Management

- ✅ View all reviews
- ✅ Filter by status (pending/approved/all)
- ✅ Approve/reject reviews
- ✅ Edit review content
- ✅ Delete reviews
- ✅ View review images
- ✅ Link reviews to orders

Analytics Dashboard

- ✅ Sales statistics
- ✅ Order metrics
- ✅ Product performance
- ✅ User statistics

Database Migrations

- ✅ Run migrations from admin dashboard
- ✅ Safe migration execution
- ✅ Column existence checking
- ✅ Error handling

Payment Features

Stripe Integration

- ✅ Stripe Checkout Sessions
- ✅ Payment Intent creation
- ✅ Webhook handling for payment events
- ✅ Order creation from Stripe events
- ✅ Payment status tracking

PayPal Integration

- ✅ PayPal Orders API
- ✅ Order creation
- ✅ Payment capture
- ✅ Return URL handling
- ✅ Order confirmation

Payment Security

- ✅ Secure payment processing
- ✅ Webhook signature verification
- ✅ Payment status validation
- ✅ Error handling

Review System

Review Submission

- ✅ Submit reviews with payment method selection

- ✅ Payment methods: PayPal (FNF/GS), Revolut, Website
- ✅ Website reviews require completed order
- ✅ External reviews (PayPal/Revolut) allow anyone
- ✅ Image upload for reviews
- ✅ Rating system (1-5 stars)
- ✅ Title and content fields

Review Display

- ✅ Public reviews page
- ✅ Show approved reviews only
- ✅ Display review images
- ✅ Show order number if linked
- ✅ Display payment method
- ✅ Show user verification badges
- ✅ Review filtering

Additional Features

Image Management

- ✅ Cloudinary integration
- ✅ Image upload and optimization
- ✅ Multiple image support per product
- ✅ Profile picture uploads
- ✅ Review image uploads
- ✅ Image compression (client-side)
- ✅ Automatic image optimization

Email System

- ✅ Order confirmation emails
- ✅ Email templates
- ✅ Nodemailer integration

Shipping

- ✅ Shipping cost calculation
- ✅ Multiple shipping methods
- ✅ Address validation
- ✅ Country-based shipping

Security Features

- ✅ HTTPS/SSL enforcement
- ✅ Password hashing
- ✅ CSRF protection
- ✅ SQL injection prevention (Prisma)
- ✅ XSS protection
- ✅ Secure session management
- ✅ Role-based access control
- ✅ API authentication

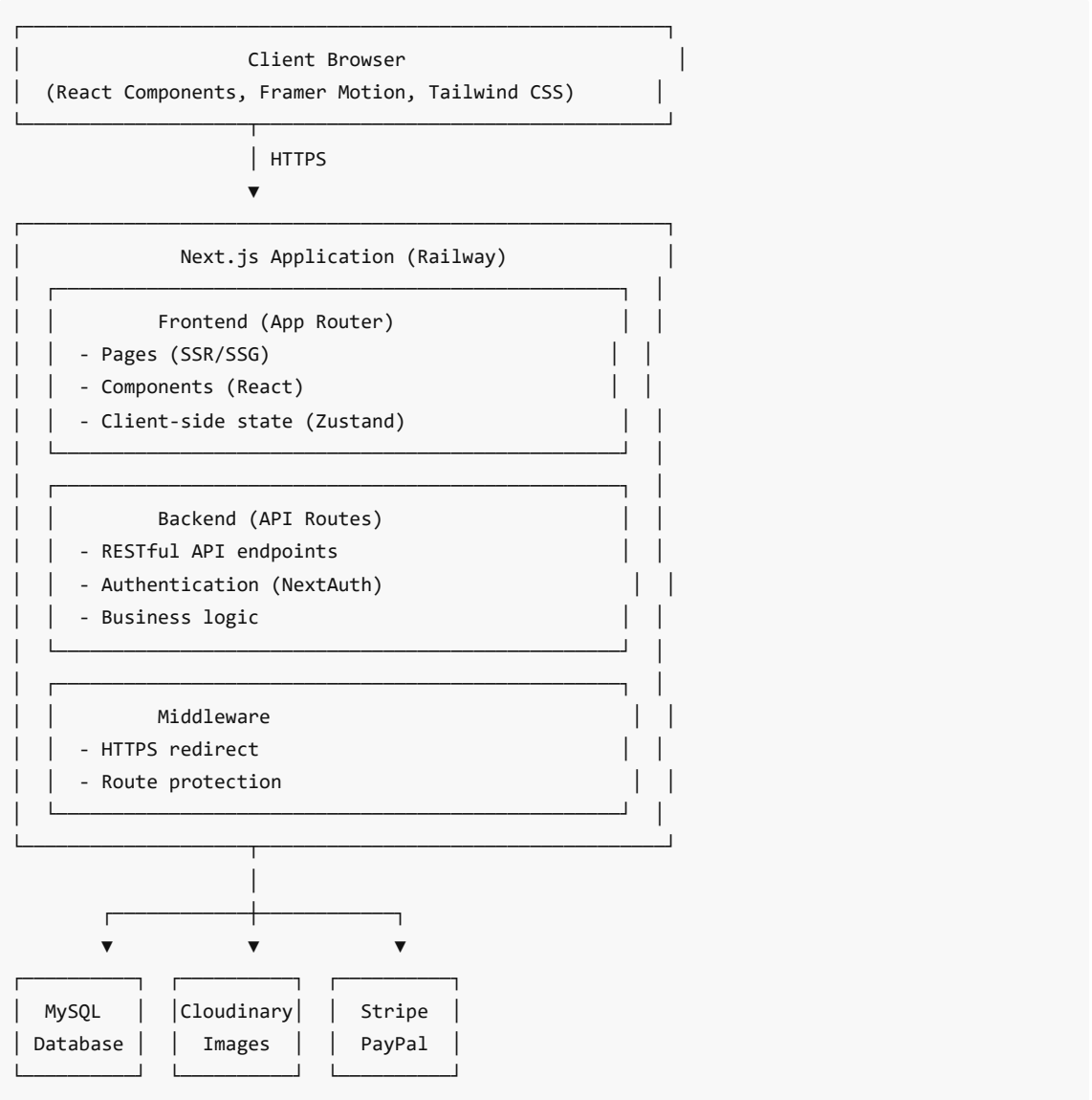
UI/UX Features

- ✅ Dark theme design
- ✅ Smooth animations (Framer Motion)
- ✅ Responsive design (mobile-first)

- ✅ Loading states
- ✅ Error handling
- ✅ Toast notifications
- ✅ Modal dialogs
- ✅ Form validation
- ✅ Scroll animations
- ✅ Galaxy-themed start screen

Architecture & Design

Application Architecture



Design Patterns

1. **Server-Side Rendering (SSR)**
    - Next.js App Router for SEO and performance

- Server components for data fetching

2. **API Routes Pattern**
   - RESTful endpoints
   - Type-safe request/response handling
   - Error handling middleware
3. **Component-Based Architecture**
   - Reusable React components
   - Separation of concerns
   - Props-based data flow
4. **State Management**
   - Zustand for client-side state
   - Server state via API calls
   - Session state via NextAuth
5. **Database ORM Pattern**
   - Prisma for type-safe database access
   - Migration-based schema management
   - Relationship handling

Database Schema

Entity Relationship Diagram

Tables

Users

```
- id (String, Primary Key, CUID)
- email (String, Unique)
- password (String, Hashed)
- name (String, Optional)
- displayName (String, Optional)
- avatarUrl (String, Optional)
- bio (String, Optional)
- location (String, Optional)
- website (String, Optional)
```

```
- role (String, Default: "customer") // "admin" or "customer"
- verified (Boolean, Default: false)
- createdAt (DateTime)
- updatedAt (DateTime)
```

Products

```
- id (String, Primary Key, CUID)
- name (String)
- description (Text)
- price (Decimal 10,2)
- category (String) // tops, bottoms, footwear, accessories
- images (JSON) // Array of image URLs
- inStock (Boolean, Default: true)
- featured (Boolean, Default: false)
- brand (String, Optional)
- material (String, Optional)
- weight (Decimal 8,2, Optional)
- measurements (Text, Optional)
- quantity (Int, Default: 0)
- createdAt (DateTime)
- updatedAt (DateTime)
```

ProductVariants

```
- id (String, Primary Key, CUID)
- productId (String, Foreign Key → Products)
- size (String, Optional)
- color (String, Optional)
- sku (String, Unique, Optional)
- price (Decimal 10,2, Optional)
- quantity (Int, Default: 0)
- inStock (Boolean, Default: true)
- createdAt (DateTime)
- updatedAt (DateTime)
```

Orders

```
- id (String, Primary Key, CUID)
- userId (String, Foreign Key → Users, Optional)
- orderNumber (String, Unique)
- status (String, Default: "pending")
  // pending, processing, shipped, delivered, completed, cancelled, refunded
- total (Decimal 10,2)
- subtotal (Decimal 10,2)
- shipping (Decimal 10,2)
- tax (Decimal 10,2)
- currency (String, Default: "EUR")
- shippingAddressId (String, Foreign Key → Addresses, Optional)
```

- shippingMethod (String, Optional)
- trackingNumber (String, Optional)
- paymentMethod (String, Optional)
- paymentIntentId (String, Optional)
- paidAt (DateTime, Optional)
- customerEmail (String, Optional)
- customerName (String, Optional)
- createdAt (DateTime)
- updatedAt (DateTime)

OrderItems

- id (String, Primary Key, CUID)
- orderId (String, Foreign Key → Orders)
- productId (String, Foreign Key → Products, Optional)
- variantId (String, Foreign Key → ProductVariants, Optional)
- quantity (Int)
- price (Decimal 10,2)
- name (String)
- size (String, Optional)
- color (String, Optional)
- createdAt (DateTime)

Addresses

- id (String, Primary Key, CUID)
- userId (String, Foreign Key → Users)
- fullName (String)
- street (String)
- apartment (String, Optional)
- city (String)
- province (String, Optional)
- postalCode (String)
- country (String, Default: "Netherlands")
- phone (String, Optional)
- isDefault (Boolean, Default: false)
- createdAt (DateTime)
- updatedAt (DateTime)

Wishlist

- id (String, Primary Key, CUID)
- userId (String, Foreign Key → Users)
- productId (String, Foreign Key → Products)
- createdAt (DateTime)
- Unique constraint: (userId, productId)

Cart

```
- id (String, Primary Key, CUID)
- userId (String, Foreign Key → Users, Unique)
- items (JSON) // Array of cart items
- updatedAt (DateTime)
- createdAt (DateTime)
```

Reviews

```
- id (String, Primary Key, CUID)
- userId (String, Foreign Key → Users)
- orderId (String, Foreign Key → Orders, Optional)
- title (String, Optional)
- content (String)
- rating (Int, Optional) // 1-5
- paymentMethod (String) // paypal_fnf, paypal_gs, revolut, website
- imageUrl (String, Optional)
- approved (Boolean, Default: false)
- createdAt (DateTime)
- updatedAt (DateTime)
```

DiscountCodes

```
- id (String, Primary Key, CUID)
- code (String, Unique)
- type (String) // "percentage" or "fixed"
- value (Decimal 10,2)
- minPurchase (Decimal 10,2, Optional)
- maxDiscount (Decimal 10,2, Optional)
- usageLimit (Int, Optional)
- usedCount (Int, Default: 0)
- validFrom (DateTime, Optional)
- validUntil (DateTime, Optional)
- isActive (Boolean, Default: true)
- createdAt (DateTime)
- updatedAt (DateTime)
```

Relationships

- **User** → **Orders:** One-to-Many
- **User** → **Addresses:** One-to-Many
- **User** → **Wishlist:** One-to-Many
- **User** → **Cart:** One-to-One
- **User** → **Reviews:** One-to-Many
- **Order** → **OrderItems:** One-to-Many
- **Order** → **Address:** Many-to-One
- **Order** → **Reviews:** One-to-Many
- **Product** → **ProductVariants:** One-to-Many
- **Product** → **OrderItems:** One-to-Many
- **Product** → **Wishlist:** One-to-Many

- **ProductVariant → OrderItems:** One-to-Many

API Endpoints

Authentication Endpoints

POST `/api/auth/register`

- **Purpose:** User registration
- **Body:** `{ email, password, name }`
- **Response:** `{ user, message }`
- **Security:** Password hashing with bcryptjs

POST `/api/auth/login`

- **Purpose:** User login
- **Body:** `{ email, password }`
- **Response:** `{ user, message }`
- **Security:** NextAuth.js session creation

GET `/api/auth/session`

- **Purpose:** Get current session
- **Response:** `{ user, expires }`
- **Security:** Session validation

GET `/api/auth/user`

- **Purpose:** Get authenticated user
- **Response:** `{ user }`
- **Security:** Requires authentication

Product Endpoints

GET `/api/products`

- **Purpose:** Get all products (public)
- **Query Params:** `?featuredCount=true`, `?allFeatured=true`
- **Response:** `Product[]`
- **Filters:** Excludes products without images

GET `/api/products/[id]`

- **Purpose:** Get single product
- **Response:** `Product`
- **Includes:** Variants, images

POST `/api/products` **(Admin)**

- **Purpose:** Create product
- **Body:** Product data with images
- **Response:** `{ product }`
- **Security:** Requires admin role

PUT `/api/products/[id]` **(Admin)**

- **Purpose:** Update product
- **Body:** Product data
- **Response:** `{ product }`

- **Security:** Requires admin role

DELETE `/api/products/[id]` **(Admin)**

- **Purpose:** Delete product
- **Response:** `{ message }`
- **Security:** Requires admin role

Order Endpoints

GET `/api/orders`

- **Purpose:** Get user's orders
- **Response:** `Order[]`
- **Security:** Requires authentication

GET `/api/orders/[id]`

- **Purpose:** Get single order
- **Response:** `Order`
- **Security:** Requires authentication (owner or admin)

POST `/api/orders/create`

- **Purpose:** Create order
- **Body:** Order data
- **Response:** `{ order }`
- **Security:** Requires authentication

PATCH `/api/orders/[id]` **(Admin)**

- **Purpose:** Update order status
- **Body:** `{ status }`
- **Response:** `{ order }`
- **Security:** Requires admin role

Cart Endpoints

GET `/api/cart`

- **Purpose:** Get user's cart
- **Response:** `{ items }`
- **Security:** Requires authentication

POST `/api/cart`

- **Purpose:** Update cart
- **Body:** `{ items }`
- **Response:** `{ cart }`
- **Security:** Requires authentication

Wishlist Endpoints

GET `/api/wishlist`

- **Purpose:** Get user's wishlist
- **Response:** `Product[]`
- **Security:** Requires authentication

POST `/api/wishlist/[productId]`

- **Purpose:** Add to wishlist

- **Response:** `{ message }`
- **Security:** Requires authentication

DELETE `/api/wishlist/[productId]`

- **Purpose:** Remove from wishlist
- **Response:** `{ message }`
- **Security:** Requires authentication

Checkout Endpoints

POST `/api/checkout/session`

- **Purpose:** Create Stripe checkout session
- **Body:** `{ items, shippingAddress }`
- **Response:** `{ sessionId, url }`
- **Security:** Requires authentication

Payment Endpoints

POST `/api/payments/stripe/webhook`

- **Purpose:** Handle Stripe webhooks
- **Body:** Stripe event
- **Response:** `{ received: true }`
- **Security:** Webhook signature verification

POST `/api/payments/paypal/create-order`

- **Purpose:** Create PayPal order
- **Body:** Order data
- **Response:** `{ orderId }`
- **Security:** Requires authentication

POST `/api/payments/paypal/capture`

- **Purpose:** Capture PayPal payment
- **Body:** `{ orderId }`
- **Response:** `{ order }`
- **Security:** Requires authentication

Discount Endpoints

GET `/api/discounts`

- **Purpose:** Get all discount codes (Admin)
- **Response:** `DiscountCode[]`
- **Security:** Requires admin role

POST `/api/discounts` **(Admin)**

- **Purpose:** Create discount code
- **Body:** Discount code data
- **Response:** `{ discount }`
- **Security:** Requires admin role

GET `/api/discounts/validate`

- **Purpose:** Validate discount code
- **Query:** `?code=DISCOUNT_CODE`

- **Response:** `{ valid, discount }`

PATCH `/api/discounts/[id]` **(Admin)**

- **Purpose:** Update discount code
- **Body:** Discount code data
- **Response:** `{ discount }`
- **Security:** Requires admin role

DELETE `/api/discounts/[id]` **(Admin)**

- **Purpose:** Delete discount code
- **Response:** `{ message }`
- **Security:** Requires admin role

Review Endpoints

GET `/api/reviews`

- **Purpose:** Get approved reviews (public)
- **Response:** `Review[]`
- **Includes:** User data, order number

POST `/api/reviews`

- **Purpose:** Submit review
- **Body:** `{ title, content, rating, paymentMethod, orderId?, imageUrl? }`
- **Response:** `{ review }`
- **Security:** Requires authentication
- **Logic:** Auto-approves website reviews, requires approval for external

POST `/api/reviews/upload`

- **Purpose:** Upload review image
- **Body:** FormData with image
- **Response:** `{ imageUrl }`
- **Security:** Requires authentication

Admin Endpoints

GET `/api/users` **(Admin)**

- **Purpose:** Get all users
- **Response:** `User[]`
- **Security:** Requires admin role

GET `/api/users/[id]`

- **Purpose:** Get public user profile
- **Response:** `{ user, reviewsCount }`

GET `/api/admin/users/[id]` **(Admin)**

- **Purpose:** Get user details (admin)
- **Response:** `User`
- **Security:** Requires admin role

PATCH `/api/admin/users/[id]` **(Admin)**

- **Purpose:** Update user
- **Body:** `{ name, displayName, bio, location, verified, removeAvatar }`

- **Response:** `{ user }`
- **Security:** Requires admin role

GET `/api/admin/reviews` (Admin)

- **Purpose:** Get reviews (all/pending/approved)
- **Query:** `?filter=pending|approved|all`
- **Response:** `Review[]`
- **Security:** Requires admin role

PATCH `/api/admin/reviews/[id]` (Admin)

- **Purpose:** Update review
- **Body:** `{ approved, title, content, rating, paymentMethod, imageUrl }`
- **Response:** `{ review }`
- **Security:** Requires admin role

DELETE `/api/admin/reviews/[id]` (Admin)

- **Purpose:** Delete review
- **Response:** `{ message }`
- **Security:** Requires admin role

POST `/api/admin/migrate` (Admin)

- **Purpose:** Run database migrations
- **Response:** `{ success, message }`
- **Security:** Requires admin role
- **Logic:** Checks for existing columns, adds missing ones

Profile Endpoints

GET `/api/profile`

- **Purpose:** Get user profile
- **Response:** `User`
- **Security:** Requires authentication

PATCH `/api/profile`

- **Purpose:** Update profile
- **Body:** `{ displayName, avatarUrl, bio, location }`
- **Response:** `{ user }`
- **Security:** Requires authentication

POST `/api/profile/upload`

- **Purpose:** Upload profile picture
- **Body:** FormData with image
- **Response:** `{ imageUrl }`
- **Security:** Requires authentication

Shipping Endpoints

POST `/api/shipping/calculate`

- **Purpose:** Calculate shipping costs
- **Body:** `{ country, weight }`
- **Response:** `{ cost, method }`

Authentication System

NextAuth.js Configuration

Provider: Credentials (Email/Password)

Session Strategy: JWT

Configuration:

```
- Credentials provider with email/password
- Prisma adapter for user management
- JWT token generation
- Session expiration handling
- Password comparison with bcryptjs
```

Authentication Flow

1. **Registration:**
    - User submits email, password, name
    - Password hashed with bcryptjs (10 rounds)
    - User created in database
    - Session created automatically
2. **Login:**
    - User submits email and password
    - Password verified against hash
    - Session created via NextAuth
    - JWT token generated
    - User redirected to dashboard
3. **Session Management:**
    - JWT tokens stored in HTTP-only cookies
    - Session data includes user ID and role
    - Automatic session refresh
    - Session expiration handling
4. **Protected Routes:**
    - Middleware checks authentication
    - Redirects to login if not authenticated
    - Role-based access control for admin routes

Password Security

- **Hashing:** bcryptjs with 10 salt rounds
- **Storage:** Hashed passwords only (never plain text)
- **Validation:** Password comparison on login
- **Reset:** Admin can reset passwords (generates new hash)

Role-Based Access Control

Roles:

- `admin` : Full access to admin dashboard
- `customer` : Standard user access

Access Control:

- Admin routes protected by `requireAdminApi()` helper
- Frontend checks role via session
- API routes validate role before processing

Payment Integration

Stripe Integration

Setup

- Stripe account with API keys
- Webhook endpoint configured
- Environment variables set

Checkout Flow

1. User clicks checkout
2. Create Stripe Checkout Session via API
3. Redirect to Stripe hosted checkout
4. User completes payment
5. Stripe webhook fires on success
6. Order created in database
7. Email confirmation sent

Webhook Handling

- Signature verification
- Event type handling:
  - `checkout.session.completed` : Create order
  - `payment_intent.succeeded` : Update order status
- Idempotency handling
- Error logging

PayPal Integration

Setup

- PayPal Developer account
- Client ID and Secret
- Sandbox/Production environment

Checkout Flow

1. User selects PayPal payment
2. Create PayPal Order via API
3. Redirect to PayPal
4. User approves payment
5. Capture payment on return
6. Order created in database
7. Email confirmation sent

Order Management

- Order creation
- Payment capture
- Return URL handling
- Error handling

Payment Security

- **Webhook Verification:** Signature validation
- **Idempotency:** Prevent duplicate orders
- **Error Handling:** Comprehensive error catching
- **Logging:** Payment event logging
- **Testing:** Sandbox environments for testing

Admin Dashboard

Dashboard Structure

Tabs:

1. Products
2. Orders
3. Analytics
4. Discounts
5. Customers
6. Inventory
7. Users
8. Reviews

Product Management

Features:

- Add/Edit/Delete products
- Image upload via Cloudinary
- Variant management (sizes, colors)
- Inventory tracking
- Featured product toggle (max 4)
- Category assignment
- Search and filter

UI Components:

- Product list with cards
- Modal forms for add/edit
- Image gallery
- Variant table
- Stock status indicators

Order Management

Features:

- View all orders
- Filter by status
- Update order status

- View order details
- Track shipping
- Order analytics

Status Options:

- Pending
- Processing
- Shipped
- Delivered
- Completed
- Cancelled
- Refunded

User Management

Features:

- View all users
- Edit user profiles
- Remove profile pictures
- Mark users as verified
- Search users
- View user details

Review Management

Features:

- View all reviews
- Filter by status (pending/approved/all)
- Approve/reject reviews
- Edit review content
- Delete reviews
- View review images

Analytics Dashboard

Metrics:

- Total sales
- Order count
- Product performance
- User statistics
- Revenue trends

User Features

User Profile

Features:

- Profile page (`/profile`)
- Edit profile information
- Upload profile picture

- View purchase history
- Manage addresses
- View wishlist

Public Profile

Features:

- Public profile pages (`/u/[id]`)
- Display name and bio
- Profile picture
- Verification badge
- Review count
- Join date

Shopping Experience

Features:

- Browse products
- View product details
- Add to cart
- Add to wishlist
- Apply discount codes
- Complete checkout
- Track orders

Code Structure

Directory Structure

```
fashionvoid_webshop/
├── app/                          # Next.js App Router
│   ├── admin/                   # Admin dashboard page
│   ├── api/                     # API routes
│   │   ├── admin/              # Admin endpoints
│   │   ├── auth/               # Authentication
│   │   ├── cart/               # Cart management
│   │   ├── checkout/           # Checkout flow
│   │   ├── discounts/          # Discount codes
│   │   ├── orders/             # Order management
│   │   ├── payments/          # Payment processing
│   │   ├── products/           # Product CRUD
│   │   ├── profile/            # User profile
│   │   ├── reviews/            # Review system
│   │   ├── shipping/           # Shipping calculation
│   │   ├── stripe/             # Stripe webhooks
│   │   ├── users/              # User management
│   │   └── wishlist/           # Wishlist
│   ├── cart/                    # Cart page
│   ├── checkout/                # Checkout page
│   ├── login/                  # Login page
│   ├── order-confirmation/      # Order confirmation
```

```
│   ├── orders/                 # User orders
│   ├── product/[id]/           # Product detail
│   ├── profile/                # User profile
│   ├── purchases/              # Purchase history
│   ├── register/               # Registration
│   ├── reviews/                # Reviews page
│   ├── u/[id]/                 # Public profiles
│   ├── wishlist/               # Wishlist page
│   ├── layout.tsx              # Root layout
│   ├── page.tsx                # Homepage
│   └── globals.css             # Global styles
├── components/                  # React components
│   ├── AdminDashboard.tsx      # Admin dashboard
│   ├── Cart.tsx                # Shopping cart
│   ├── CartSync.tsx            # Cart synchronization
│   ├── CheckoutProgress.tsx    # Checkout steps
│   ├── CheckoutStep1.tsx       # Address step
│   ├── CheckoutStep2.tsx       # Payment step
│   ├── CheckoutStep3.tsx       # Confirmation step
│   ├── GalaxyStartScreen.tsx   # Welcome screen
│   ├── Navbar.tsx              # Navigation
│   ├── ProductCard.tsx         # Product card
│   ├── Providers.tsx           # Context providers
│   └── ScrollReveal.tsx        # Scroll animations
├── lib/                         # Utility libraries
│   ├── auth.ts                 # NextAuth config
│   ├── auth-helpers.ts         # Auth utilities
│   ├── countries.ts            # Country data
│   ├── email.ts                # Email sending
│   ├── payments.ts             # Payment utilities
│   ├── paypal.ts               # PayPal client
│   ├── prisma.ts               # Prisma client
│   ├── products.ts             # Product utilities
│   ├── shipping.ts             # Shipping calculation
│   ├── store.ts                # Zustand store
│   ├── stripe.ts               # Stripe client
│   └── vat.ts                  # VAT calculation
├── prisma/                      # Database
│   ├── schema.prisma           # Database schema
│   ├── migrations/             # Migration files
│   └── seed.ts                 # Database seeding
├── scripts/                     # Utility scripts
│   ├── seed-products.ts        # Product seeding
│   ├── check-products.ts       # Product checking
│   └── ...                     # Other utilities
├── public/                      # Static assets
├── types/                       # TypeScript types
├── middleware.ts                # Next.js middleware
├── next.config.js              # Next.js config
├── package.json                # Dependencies
```

```
├── tailwind.config.js           # Tailwind config
└── tsconfig.json                # TypeScript config
```

Key Files

Configuration:

- `next.config.js` : Next.js configuration, image domains, headers
- `tailwind.config.js` : Tailwind CSS configuration
- `tsconfig.json` : TypeScript configuration
- `middleware.ts` : HTTPS redirects, route protection

Core Libraries:

- `lib/auth.ts` : NextAuth configuration
- `lib/prisma.ts` : Database client
- `lib/store.ts` : Zustand state management

Components:

- `components/AdminDashboard.tsx` : Main admin interface (3000+ lines)
- `components/Cart.tsx` : Shopping cart component
- `components/Navbar.tsx` : Navigation bar

Key Implementations

1. Shopping Cart Persistence

Implementation:

- Cart stored in database (`Cart` model)
- Synchronized across devices
- Persists after logout/login
- Real-time updates

Code Location:

- `app/api/cart/route.ts`
- `components/Cart.tsx`
- `components/CartSync.tsx`

2. Image Upload System

Implementation:

- Cloudinary integration
- Client-side compression
- Multiple image support
- Automatic optimization

Code Location:

- `app/api/upload/route.ts`
- `app/api/profile/upload/route.ts`
- `app/api/reviews/upload/route.ts`

3. Order Processing

Implementation:

- Order creation from cart
- Payment processing
- Email confirmations
- Status tracking

Code Location:

- `app/api/orders/create/route.ts`
- `app/api/stripe/webhook/route.ts`
- `lib/email.ts`

4. Review System

Implementation:

- Multiple payment method support
- Image uploads
- Admin approval workflow
- Order linking

Code Location:

- `app/api/reviews/route.ts`
- `app/api/admin/reviews/route.ts`
- `app/reviews/page.tsx`

5. Admin Migration System

Implementation:

- Safe migration execution
- Column existence checking
- Error handling
- Admin-only access

Code Location:

- `app/api/admin/migrate/route.ts`

6. HTTPS Enforcement

Implementation:

- Middleware redirects
- Multiple header checking
- Production-only enforcement

Code Location:

- `middleware.ts`

7. Payment Gateway Integration

Implementation:

- Stripe Checkout Sessions
- PayPal Orders API
- Webhook handling
- Error recovery

Code Location:

- `app/api/payments/stripe/webhook/route.ts`
- `app/api/payments/paypal/create-order/route.ts`
- `lib/stripe.ts`
- `lib/paypal.ts`

Deployment

Railway Deployment

Platform: Railway.app

Configuration:

- Node.js 22.22.0
- Standalone build output
- Environment variables configured
- Custom domains: `www.fashionvoid.net`, `fashionvoid.net`
- SSL certificates (automatic)

Build Process:

1. Install dependencies (`npm ci`)
2. Generate Prisma Client
3. Build Next.js app (`next build`)
4. Copy static assets
5. Start server (`node .next/standalone/server.js`)

Environment Variables:

- `DATABASE_URL` : MySQL connection string
- `NEXTAUTH_URL` : https://www.fashionvoid.net
- `NEXTAUTH_SECRET` : Secret key for sessions
- `CLOUDINARY_*` : Image hosting credentials
- `STRIPE_*` : Payment processing keys
- `PAYPAL_*` : PayPal API credentials

Domain Setup:

- CNAME record: `www` → Railway domain
- A record or redirect: `@` → `www.fashionvoid.net`
- SSL certificates provisioned automatically

Deployment Features:

- Automatic deployments on git push
- Health checks
- Logging and monitoring
- Zero-downtime deployments

Security Features

Authentication Security

- Password hashing (bcryptjs, 10 rounds)
- JWT token-based sessions
- HTTP-only cookies
- CSRF protection
- Session expiration

API Security

- Role-based access control
- Authentication middleware
- Input validation
- SQL injection prevention (Prisma)
- XSS protection

Payment Security

- Webhook signature verification
- Secure payment processing
- PCI compliance (via Stripe/PayPal)
- Idempotency handling

Infrastructure Security

- HTTPS/SSL enforcement
- Secure headers (HSTS, CSP)
- Environment variable protection
- Secure session storage

Performance Optimizations

Frontend Optimizations

- Server-side rendering (SSR)
- Image optimization (Cloudinary, Sharp)
- Code splitting
- Lazy loading
- Animation optimization (Framer Motion)

Backend Optimizations

- Database indexing
- Query optimization (Prisma)
- Caching strategies
- Efficient API responses
- Standalone build output

Database Optimizations

- Proper indexing
- Relationship optimization
- Query batching

- Connection pooling

Interview Summary

Quick Overview (30 seconds)

"I built FashionVoid, a full-stack e-commerce platform using Next.js 14, TypeScript, and MySQL. It includes product management, secure authentication, Stripe/PayPal payments, an admin dashboard, and order processing. The site is live at fashionvoid.net and has processed real orders."

Technical Stack (1 minute)

Frontend: Next.js 14 with React, TypeScript, Tailwind CSS, and Framer Motion for animations.

Backend: Next.js API Routes with Prisma ORM for type-safe database operations.

Database: MySQL with a relational schema including users, products, orders, and reviews.

Authentication: NextAuth.js with JWT sessions and role-based access control.

Payments: Integrated both Stripe and PayPal with webhook handling.

Deployment: Railway with HTTPS/SSL, custom domains, and automatic deployments.

Key Features (2 minutes)

1. **Complete E-Commerce:** Product catalog, shopping cart, checkout, order management
2. **Admin Dashboard:** Full CRUD for products, orders, users, discounts, reviews
3. **Payment Integration:** Stripe and PayPal with secure webhook processing
4. **User System:** Registration, profiles, wishlist, order history, verification badges
5. **Review System:** Multi-payment-method reviews with admin approval workflow
6. **Image Management:** Cloudinary integration with optimization
7. **Security:** HTTPS enforcement, password hashing, role-based access
8. **Responsive Design:** Mobile-first with smooth animations

Technical Challenges Solved (2 minutes)

1. **Cart Persistence:** Implemented database-backed cart that syncs across devices
2. **Payment Webhooks:** Built robust webhook handling with signature verification and idempotency
3. **Image Optimization:** Client-side compression before Cloudinary upload
4. **Database Migrations:** Created admin-accessible migration system for Railway deployment
5. **HTTPS Redirects:** Implemented middleware to force HTTPS in production
6. **Review Approval:** Built workflow for different review types with conditional approval

Code Quality Highlights

- **Type Safety:** Full TypeScript implementation
- **Error Handling:** Comprehensive error handling throughout
- **Security:** Password hashing, CSRF protection, SQL injection prevention
- **Performance:** SSR, image optimization, efficient queries
- **User Experience:** Smooth animations, loading states, error messages
- **Maintainability:** Clean code structure, reusable components, proper separation of concerns

Metrics & Results

- **Live Site:** https://www.fashionvoid.net

- **Users Tested:** 15+ successful signups
- **Payment Methods:** Stripe and PayPal integrated
- **Admin Features:** 8 comprehensive dashboard tabs
- **Database Tables:** 10 tables with proper relationships
- **API Endpoints:** 40+ RESTful endpoints
- **Codebase:** 3000+ lines in admin dashboard alone

What I Learned

- Full-stack development with Next.js
- Payment gateway integration
- Database design and optimization
- Authentication and authorization
- Production deployment and DevOps
- API design and security
- State management patterns
- Image handling and optimization

Future Enhancements (If Asked)

- Product search with filters
- Email notifications
- Order tracking integration
- Analytics dashboard expansion
- Multi-language support
- Mobile app (React Native)

Conclusion

FashionVoid represents a complete, production-ready e-commerce platform built with modern web technologies. It demonstrates proficiency in full-stack development, payment integration, database design, authentication, and deployment. The platform is live, tested, and ready for real-world use.

Key Takeaways:

- Built from scratch with no templates
- Production-ready with real deployments
- Comprehensive feature set
- Secure and performant
- Well-documented and maintainable

Document Version: 1.0
Last Updated: January 2025
Author: Petar Vukovic