FashionStore

Análisis Técnico Completo del Proyecto

E-commerce de Moda Premium construido con Astro 5.0,
Supabase, Stripe y Tailwind CSS

Proyecto: TiendaOnlineAstro (FashionStore)

Versión: 1.0.0

Fecha del Análisis: 23 de Febrero de 2026

Tipo: Aplicación Web Full-Stack (SSR/SSG Híbrido)

Astro 5.0 | Supabase | Stripe | React 19 | Tailwind CSS 3 | TypeScript | Docker | Node.js 20

Índice de Contenidos

1. Resumen Ejecutivo

FashionStore es una aplicación web de comercio electrónico especializada en moda premium, tanto masculina como femenina. El proyecto implementa una tienda online completa con catálogo de productos, carrito de compras, pasarela de pago integrada con Stripe, gestión de pedidos, sistema de devoluciones, facturación, códigos de descuento, newsletter y un panel de administración completo.

Características Principales

🛒 Tienda Pública

- Catálogo con filtros por categoría
- Búsqueda de productos
- Detalle de producto con galería
- Carrito persistente (localStorage)
- Checkout con Stripe (tarjeta + PayPal)
- Registro/Login de clientes
- Historial de pedidos del cliente
- Solicitud de devoluciones
- Newsletter con código de descuento
- Recomendador de tallas

⚙️ Panel de Administración

- Dashboard con KPIs y estadísticas
- CRUD completo de productos
- Gestión de categorías jerárquicas
- Gestión de pedidos y estados
- Procesamiento de devoluciones
- Facturas y notas de abono
- Códigos de descuento
- Configuración de ofertas flash
- Registro de actividad (audit log)

Métricas del Proyecto

Métrica	Valor
Páginas/Rutas públicas	~20 rutas
Páginas Admin	~15 rutas
Endpoints API	~21 endpoints REST
Componentes Astro	12 componentes
Islas React (interactivas)	4 componentes
Layouts	3 (Base, Public, Admin)
Tablas en Base de Datos	13 tablas
Migraciones SQL	9 archivos
Funciones de email	5 templates transaccionales

Variables de entorno	11 variables

2. Stack Tecnológico y Justificación

2.1 Framework Frontend: Astro 5.0

Astro fue seleccionado como framework principal por su enfoque **"Zero JavaScript by Default"**, ideal para e-commerce donde el SEO y la velocidad de carga son críticos.

Característica	Astro 5.0	Next.js	React SPA
SEO	✅ Excelente (SSG nativo)	⚠️ Requiere config ISR/SSG	❌ Problemas indexación
Performance	✅ <5KB por página	⚠️ ~80KB bundle	❌ ~200KB+ bundle
JS enviado al cliente	✅ Solo donde se necesita	⚠️ Hidratación completa	❌ Todo el bundle
Modo Híbrido SSG+SSR	✅ Nativo	✅ Con configuración	❌ No soportado
Island Architecture	✅ Nativo	❌ No	❌ No

> **Arquitectura de Islas:** Astro permite insertar componentes React interactivos (islas) dentro de páginas estáticas. Solo esos componentes se hidratan en el cliente, manteniendo el resto como HTML puro.

2.2 Base de Datos: Supabase (PostgreSQL)

Supabase proporciona una base de datos PostgreSQL gestionada con autenticación integrada, almacenamiento de archivos y API REST automática.

- **PostgreSQL**: Motor relacional robusto con soporte para JSON, arrays y transacciones ACID
- **Auth integrado**: Registro, login, recuperación de contraseña, JWT
- **Storage**: Bucket `products-images` para imágenes de productos
- **RLS (Row Level Security)**: Políticas de seguridad a nivel de fila
- **Funciones RPC**: `validate_discount_code`, `apply_discount_code`

2.3 Pasarela de Pago: Stripe

Aspecto	Stripe	PayPal	Redsys
Comisión	2,4% + €0,30	2,9% + €0,30	0,85%-2,5%
Documentación	✅ Excelente	Buena	❌ Pobre
Modo Test	✅ Fácil	Fácil	❌ Difícil
Webhooks	✅ Nativos	⚠️ Complejos	❌ Limitados
SDK Node.js	✅ Oficial	✅ Disponible	❌ XML/SOAP

Métodos de pago habilitados: `Tarjeta` `PayPal` `Google Pay` `Apple Pay`

2.4 Gestión de Estado: Nano Stores

Se utiliza **Nano Stores** para el estado global del carrito. Es una librería ultra-ligera (~300 bytes) que funciona tanto con Astro como con React, ideal para la arquitectura de islas.

2.5 Estilos: Tailwind CSS 3.4

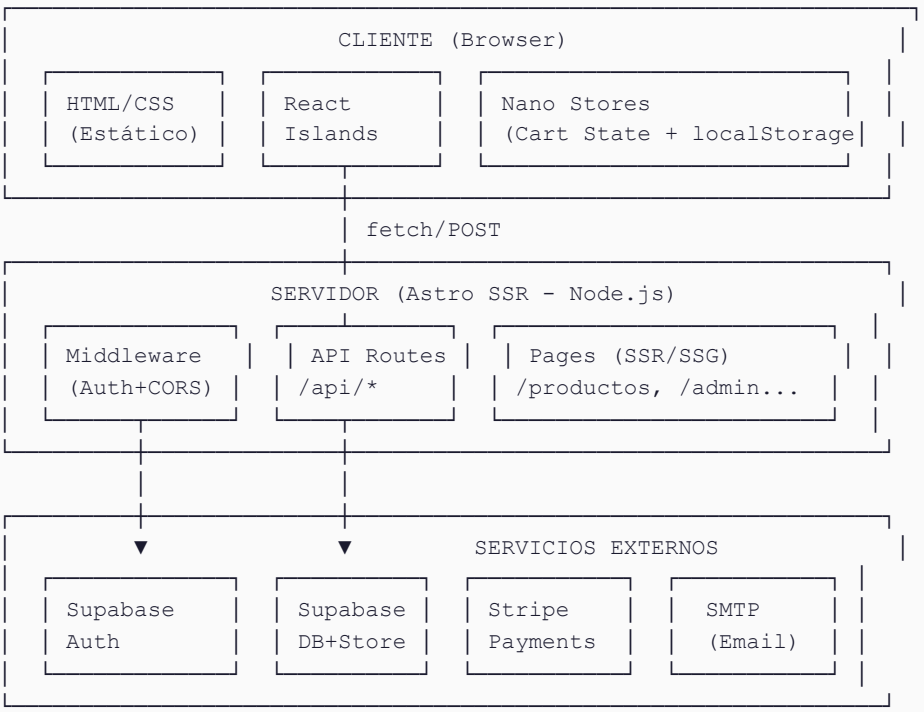
Tailwind CSS con un sistema de diseño personalizado inspirado en Pull & Bear (minimalista, monocromático). Se definen paletas de colores personalizadas (pb, primary, accent, charcoal, navy, gold), tipografía Helvetica Neue y animaciones CSS personalizadas.

2.6 TypeScript (Strict Mode)

Todo el proyecto usa TypeScript en modo estricto (`strictNullChecks: true`) con alias de rutas (`@/*` → `src/*`) para imports limpios.

3. Arquitectura del Sistema

3.1 Patrón Arquitectónico: MPA Híbrido con Island Architecture



3.2 Renderizado Híbrido

Tipo	Rutas	Descripción
SSR	/admin/*, /carrito, /checkout, /cuenta/*	Contenido dinámico, protegido por middleware
SSR	/productos, /categoria/*, /index	Contenido dinámico desde Supabase
API	/api/*	Endpoints REST (prerender=false)

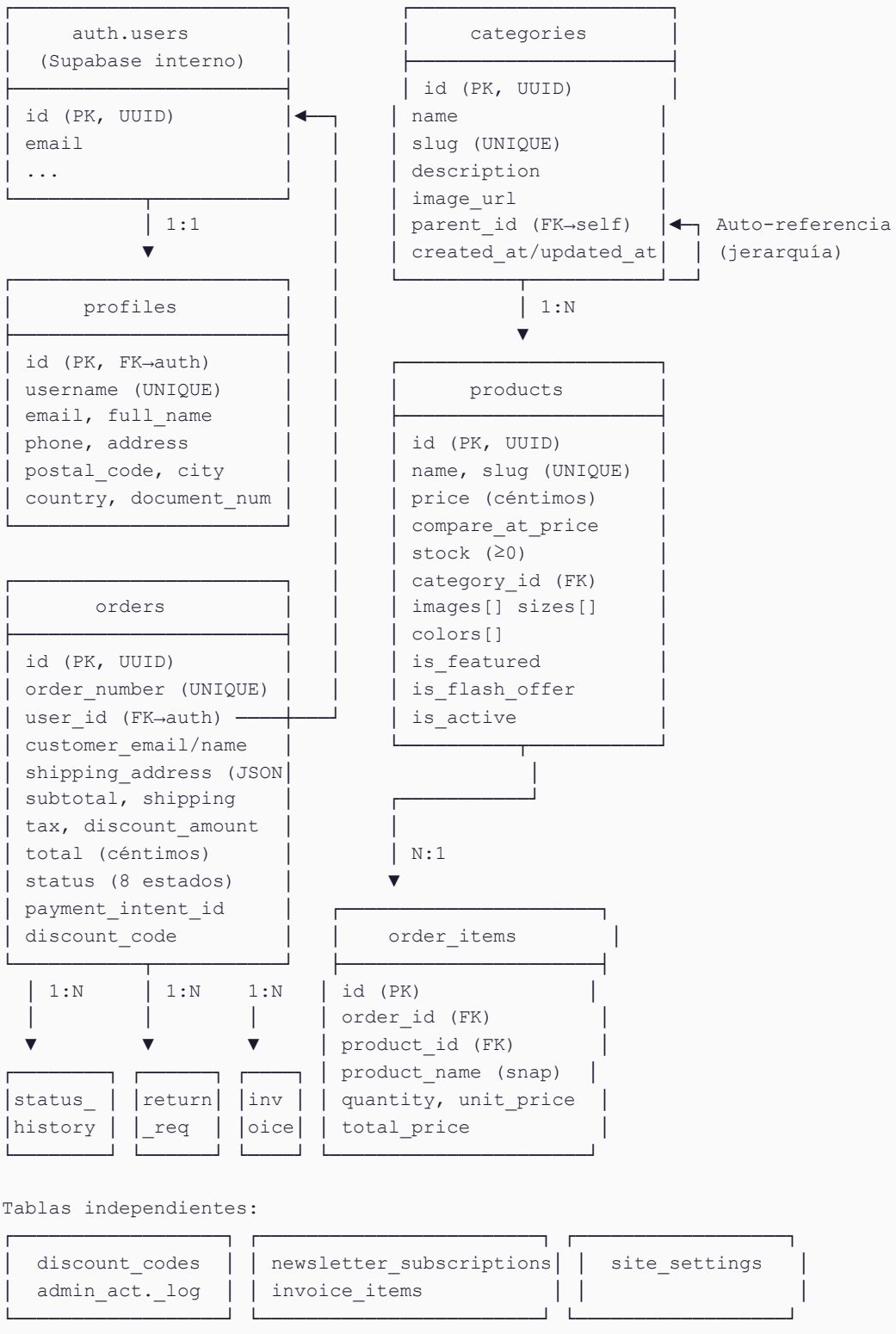
4. Estructura del Proyecto

```
TiendaOnlineAstro/
├── src/
│   ├── components/
│   │   ├── category/
│   │   │   ├── CategoryControls.tsx      # Filtros de categoría (React)
│   │   │   └── CategoryPage.astro        # Template de página de categoría
│   │   ├── islands/                       # 🏝️ Islas React (interactivas)
│   │   │   ├── AddToCartButton.tsx        # Botón añadir al carrito
│   │   │   ├── CartDrawer.tsx             # Panel lateral del carrito
│   │   │   ├── CartIcon.tsx               # Icono carrito con badge
│   │   │   └── SizeRecommender.tsx        # Recomendador de tallas
│   │   ├── product/
│   │   │   ├── ProductCard.astro          # Tarjeta de producto
│   │   │   └── ProductGallery.astro       # Galería de imágenes
│   │   └── ui/
│   │       ├── Button.astro               # Botón reutilizable
│   │       ├── NewsletterPopup.astro       # Popup newsletter
│   │       ├── SearchBar.astro            # Barra búsqueda (Astro)
│   │       └── SearchBar.tsx              # Barra búsqueda (React)
│   ├── layouts/
│   │   ├── AdminLayout.astro              # Layout panel admin (sidebar)
│   │   ├── BaseLayout.astro               # Layout base (head, meta, SEO)
│   │   └── PublicLayout.astro             # Layout público (header, footer)
│   ├── lib/
│   │   ├── email.ts                       # 5 funciones de email (nodemailer)
│   │   ├── supabase.ts                    # Cliente, tipos, queries, demo data
│   │   └── utils.ts                       # Utilidades (formatPrice, slugify...)
│   ├── pages/
│   │   ├── admin/                         # 🔒 Panel de administración
│   │   │   ├── index.astro                # Dashboard con KPIs
│   │   │   ├── configuracion.astro        # Toggle flash offers
│   │   │   ├── productos/                 # CRUD productos
│   │   │   ├── categorias/                # CRUD categorías
│   │   │   ├── pedidos/                   # Gestión pedidos
│   │   │   ├── descuentos/                # CRUD descuentos
│   │   │   └── facturas/                  # Facturas y abonos
│   │   ├── api/                           # 🔌 Endpoints REST
│   │   │   ├── auth/                      # Login, logout, change-password
│   │   │   ├── checkout.ts                # Crear sesión Stripe
│   │   │   ├── webhook.ts                 # Webhook de Stripe
│   │   │   ├── orders/                    # Cancelar, devolver, reembolsar
│   │   │   ├── products/                  # Búsqueda, detalle
│   │   │   ├── discount/                  # CRUD y validación descuentos
│   │   │   ├── newsletter/                # Suscripción newsletter
│   │   │   └── settings/                  # Configuración del sitio
│   │   ├── cuenta/                        # 👤 Área del cliente
│   │   │   ├── pedidos.astro              # Historial de pedidos
│   │   │   ├── direcciones.astro          # Direcciones de envío
│   │   │   └── seguridad.astro            # Cambio contraseña
│   │   ├── index.astro                    # Homepage
│   │   ├── carrito.astro                  # Página del carrito
```

```
│   │   ├── login.astro / registro.astro   # Autenticación
│   │   └── productos/ / categoria/        # Catálogo
│   ├── stores/
│   │   └── cart.ts                        # Estado global carrito (NanoStores)
│   └── middleware.ts                      # Auth + Security Headers
├── supabase/
│   ├── esquema_completo_ER.sql            # Schema consolidado (13 tablas)
│   └── migrations/                        # 9 migraciones incrementales
├── Dockerfile                             # Multi-stage build
├── docker-compose.yml                     # Despliegue Docker
├── astro.config.mjs                       # Config Astro (SSR, sitemap)
├── tailwind.config.mjs                    # Sistema de diseño
├── tsconfig.json                          # TypeScript strict
└── package.json                           # Dependencias
```

5. Modelo de Base de Datos (Diagrama Entidad-Relación)

La base de datos cuenta con **13 tablas** en PostgreSQL, gestionadas a través de Supabase con RLS habilitado.



5.1 Tablas Principales y Campos Clave

Tabla	Propósito	Campos Clave	Relaciones
categories	Categorías de productos	name, slug, parent_id	Auto-ref jerárquica
products	Catálogo de productos	price (cents), stock, images[], sizes[], colors[]	N:1 → categories
profiles	Perfil del cliente	username, email, address, document_number	1:1 → auth.users
orders	Pedidos	order_number, status (8 vals), payment_intent_id	N:1 → auth.users
order_items	Líneas del pedido	product_name (snapshot), quantity, unit_price	N:1 → orders, products
order_status_history	Historial de estados	old_status, new_status, changed_by	N:1 → orders
return_requests	Devoluciones	reason, status (5 vals), refund_amount	N:1 → orders
invoices	Facturas y abonos	type (invoice/credit_note), tax_rate, total	N:1 → orders
invoice_items	Líneas de factura	product_name, quantity, unit_price	N:1 → invoices
discount_codes	Códigos descuento	code, discount_type, discount_value, max_uses	Independiente
newsletter_subscriptions	Newsletter	email, source, is_active	Independiente
site_settings	Configuración global	key, value (JSONB)	Independiente
admin_activity_log	Auditoría	action, entity_type, details (JSONB)	Independiente

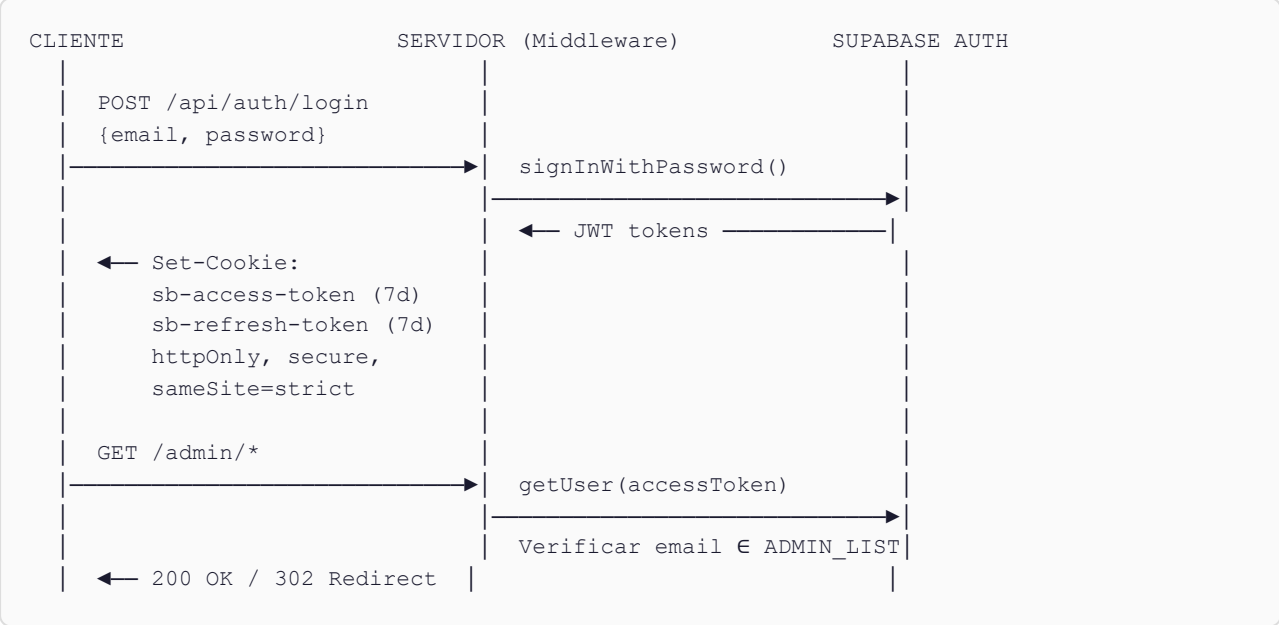
5.2 Índices de Rendimiento

Se definen **17 índices** optimizados, incluyendo índices parciales (WHERE clause) para filtros frecuentes:

- `idx_products_featured` — WHERE is_featured = TRUE
- `idx_products_flash_offer` — WHERE is_flash_offer = TRUE
- `idx_products_active` — WHERE is_active = TRUE
- Índices en slug, category_id, order_id, email, etc.

6. Sistema de Autenticación y Seguridad

6.1 Flujo de Autenticación



6.2 Control de Acceso por Roles

Rol	Verificación	Acceso
Admin	Email en whitelist (`ADMIN_EMAILS`)	/admin/*, APIs administrativas
Cliente autenticado	JWT válido en cookies	/cuenta/*, historial, devoluciones
Visitante	Sin autenticación	Tienda pública, carrito, checkout

6.3 Cookies de Sesión

Cookie	Flags	Expiración
sb-access-token	httpOnly, secure, sameSite=strict	7 días
sb-refresh-token	httpOnly, secure, sameSite=strict	7 días

El middleware implementa **auto-refresh**: si el access token expira, usa el refresh token para obtener nuevos tokens automáticamente.

7. API REST - Endpoints

Método	Ruta	Descripción	Auth

POST	/api/auth/logout	Cerrar sesión admin	Admin
POST	/api/auth/customer-logout	Cerrar sesión cliente	Cliente
POST	/api/auth/change-password	Cambiar contraseña	Cliente
POST	/api/checkout	Crear sesión de pago Stripe	Público
POST	/api/webhook	Webhook Stripe (checkout.session.completed)	Stripe
GET	/api/products/search	Buscar productos	Público
GET	/api/products/[id]	Detalle producto	Público
GET	/api/categories	Listar categorías	Público
GET/POST	/api/discount	CRUD códigos descuento	Admin
PUT/DEL	/api/discount/[id]	Editar/Eliminar descuento	Admin
POST	/api/discount/validate	Validar código descuento	Público
POST	/api/orders/cancel	Cancelar pedido	Admin
POST	/api/orders/return-request	Solicitar devolución	Cliente
POST	/api/orders/process-refund	Procesar reembolso	Admin
POST	/api/newsletter/subscribe	Suscribir newsletter	Público
PUT	/api/settings	Actualizar configuración	Admin

8. Módulos Funcionales

8.1 Carrito de Compras

Implementado con **Nano Stores** y persistencia en `localStorage`. El carrito funciona como estado global compartido entre componentes Astro y React.

Función	Descripción
`addToCart(product, qty, size, color)`	Añade producto. Valida stock. Si existe variante, incrementa cantidad
`removeFromCart(itemId)`	Elimina línea del carrito
`updateCartItemQuantity(itemId, qty)`	Actualiza cantidad (mín 1)
`clearCart()`	Vacía el carrito completo
`getCartCheckoutPayload()`	Formatea datos para enviar al checkout

Stores computados: `cartCount`, `cartSubtotal`, `cartTotal` — se recalculan automáticamente.

8.2 Flujo de Checkout y Pago

```
1. Cliente pulsa "Pagar" en /carrito
2. Frontend POST /api/checkout con items + email + discountCode
3. API calcula envío: <50€ → 4,99€ / ≥50€ → GRATIS
4. Si hay código descuento → valida via RPC validate_discount_code
5. Crea cupón en Stripe dinámicamente (porcentaje o fijo)
6. Crea Stripe Checkout Session con:
   - line_items, shipping_options, discounts
   - payment_methods: [card, paypal]
   - success_url → /pedido/exito?session_id={ID}
7. Redirige al cliente a Stripe Checkout
8. Cliente paga → Stripe envía webhook a /api/webhook
9. Webhook:
   a. Verifica firma del webhook
   b. Crea orden en Supabase (orders + order_items)
   c. Reduce stock de productos (UPDATE ... SET stock = stock - qty)
   d. Envía email de confirmación al cliente
   e. Genera factura automática
```

8.3 Sistema de Devoluciones

Flujo completo de devolución con 5 estados: `pending` → `approved` → `received` → `refunded` (o `rejected`).

- Cliente solicita devolución desde su cuenta
- Admin revisa, aprueba/rechaza desde el panel
- Al aprobar: genera nota de abono y reembolso en Stripe

- Emails transaccionales en cada paso

8.4 Sistema de Descuentos

Campo	Descripción
discount_type	`percentage` (ej: 10% OFF) o `fixed` (ej: 5€ OFF)
min_purchase_amount	Compra mínima para aplicar (en céntimos)
max_uses	Límite de usos (NULL = ilimitado)
valid_from / valid_until	Periodo de validez
is_newsletter_code	Generado por suscripción newsletter

8.5 Ofertas Flash (Toggle)

Mecanismo de activación/desactivación controlado desde `site_settings` (key: `show_flash_offers`). El admin puede activar/desactivar la sección de ofertas desde `/admin/configuracion`. El frontend lee este valor y renderiza/oculta la sección en la homepage.

9. Sistema de Emails Transaccionales

Implementado con **Nodemailer** via SMTP. Cada email incluye un template HTML completo responsive.

Función	Trigger	Contenido
`sendOrderConfirmationEmail`	Pago completado (webhook)	Resumen pedido, items, totales, dirección
`sendPasswordResetEmail`	Solicitud de recuperación	Link de reseteo
`sendRefundTicketEmail`	Solicitud de devolución	Ticket devolución, items, motivo
`sendRefundConfirmationEmail`	Reembolso procesado	Confirmación con importes
`sendNewsletterWelcomeEmail`	Suscripción newsletter	Bienvenida + código descuento

10. Despliegue e Infraestructura

10.1 Docker Multi-Stage Build

```
Stage 1: deps     → node:20-alpine, npm ci (instalar dependencias)
Stage 2: builder  → Copiar código + npm run build (generar /dist)
Stage 3: runner   → Solo /dist + node_modules + package.json
                    Usuario no-root (astro:nodejs)
                    Healthcheck cada 30s
                    Puerto: 4321
```

10.2 Variables de Entorno

Variable	Tipo	Descripción
PUBLIC_SUPABASE_URL	Pública	URL del proyecto Supabase
PUBLIC_SUPABASE_ANON_KEY	Pública	Clave anónima de Supabase
SUPABASE_SERVICE_ROLE_KEY	🔒 Secreta	Clave de servicio (operaciones admin)
PUBLIC_STRIPE_PUBLISHABLE_KEY	Pública	Clave publicable de Stripe
STRIPE_SECRET_KEY	🔒 Secreta	Clave secreta de Stripe
STRIPE_WEBHOOK_SECRET	🔒 Secreta	Secreto para validar webhooks
PUBLIC_SITE_URL	Pública	URL pública del sitio
SMTP_HOST/PORT/USER/PASS	🔒 Secreta	Configuración del servidor SMTP

10.3 Plataforma de Despliegue

La aplicación está configurada para desplegarse en **Coolify** (plataforma PaaS self-hosted) con soporte para Traefik como reverse proxy, certificados SSL automáticos y healthchecks.

11. Configuración de Seguridad (Headers HTTP)

El middleware inyecta los siguientes headers de seguridad en **todas las respuestas**:

Header	Valor	Propósito
Content-Security-Policy	default-src 'self'; script-src ... stripe, maps; img-src https: data: blob:	Previene XSS, inyección de scripts

Strict-Transport-Security	max-age=31536000; includeSubDomains; preload	Fuerza HTTPS durante 1 año
X-Content-Type-Options	nosniff	Previene MIME sniffing
X-Frame-Options	SAMEORIGIN	Previene clickjacking
X-XSS-Protection	1; mode=block	Protección XSS legacy
Referrer-Policy	strict-origin-when-cross-origin	Controla info de referrer
Permissions-Policy	camera=(), microphone=(), geolocation=(), payment=(self)	Restringe APIs del navegador

12. Sistema de Diseño

12.1 Paleta de Colores

Inspirada en **Pull & Bear** — minimalista, monocromática con acentos rojos.

Token	Color Principal	Uso
pb-black / pb-white	#000000 / #FFFFFF	Base monocromática
pb-red / accent-500	#E50014	CTAs, ofertas, badges
charcoal (50-950)	#FAFAFA → #121212	Fondos, textos
gold (50-950)	#FFFAEB → #181204	Elementos premium
success/danger	Verde/Rojo semántico	Estados, alertas

12.2 Animaciones Personalizadas

7 animaciones CSS: fadeIn, fadeOut, slideUp, slideDown, slideInLeft, slideInRight, scaleIn, menuOpen.

13. Análisis de Calidad y Buenas Prácticas

13.1 Prácticas Implementadas ✅

Práctica	Implementación
TypeScript Strict	strictNullChecks habilitado, interfaces para todos los modelos
Separación de responsabilidades	Layouts, Components, Pages, Lib, Stores, API separados
Precios en céntimos	Evita errores de punto flotante (1000 = 10,00€)
Snapshots en pedidos	product_name/price se guardan en order_items (no FK)
Índices parciales	WHERE clauses para consultas frecuentes
Security Headers	CSP, HSTS, X-Frame, etc. en middleware
Cookies seguras	httpOnly, secure, sameSite=strict
Validación backend	Todas las APIs validan inputs antes de procesar
Webhook verification	Stripe firma verificada antes de procesar pagos
User no-root Docker	Contenedor corre como usuario astro:nodejs
Caché de categorías	In-memory cache con TTL de 5 minutos
SEO	Sitemap automático (excluye /admin, /api)
Demo data fallback	Datos demo si Supabase no está configurado

13.2 Patrones de Diseño Utilizados

- **Island Architecture**: Hidratación selectiva de componentes React
- **Observer Pattern**: Nano Stores con stores computados reactivos
- **Middleware Chain**: Pipeline de auth + security headers
- **Repository Pattern**: Funciones en supabase.ts encapsulan queries
- **Strategy Pattern**: Descuentos porcentuales vs fijos
- **Snapshot Pattern**: Datos de producto se copian al pedido

13.3 Dependencias del Proyecto

Dependencia	Versión	Propósito
astro	^5.17.1	Framework principal
@astrojs/node	^9.5.2	Adaptador SSR para Node.js

@astrojs/react	^4.4.2	Integración React (islas)
@astrojs/tailwind	^6.0.2	Integración Tailwind CSS
@astrojs/sitemap	^3.7.0	Generación automática de sitemap
@supabase/supabase-js	^2.90.0	Cliente Supabase (DB + Auth)
react / react-dom	^19.2.3	Runtime para islas interactivas
nanostores / @nanostores/react	^1.1.0 / ^1.0.0	Estado global del carrito
stripe	^20.1.2	SDK de Stripe para pagos
nodemailer	^6.9.4	Envío de emails transaccionales
tailwindcss	^3.4.0	Framework CSS utility-first
typescript	^5.9.3	Tipado estático

13.4 Resumen de Tecnologías

```
┌──────────────────────────────────────────────────────────┐
│                    STACK TECNOLÓGICO                     │
├──────────────────┬───────────────────────────────────────┤
│ Frontend         │ Astro 5.0 + React 19 (Islands) + TW CSS │
│ Backend          │ Astro SSR (Node.js Adapter)           │
│ Base de Datos    │ Supabase (PostgreSQL 15+)             │
│ Autenticación    │ Supabase Auth (JWT + Cookies)         │
│ Pagos            │ Stripe Checkout (Card + PayPal)       │
│ Estado Global    │ Nano Stores + localStorage            │
│ Emails           │ Nodemailer (SMTP)                     │
│ Estilos          │ Tailwind CSS 3.4 (Custom Design System) │
│ Tipado           │ TypeScript 5.9 (Strict Mode)          │
│ Despliegue       │ Docker Multi-Stage + Coolify          │
│ CDN/Proxy        │ Cloudflare (HTTPS, WAF)               │
│ Control Versión  │ Git + GitHub                          │
└──────────────────┴───────────────────────────────────────┘
```