

VSM STORE - AUDITORÍA TÉCNICA Y PLAN DE MEJORAS

Fecha: 2026-02-16

Analista: Claude (Senior Full-Stack Review)

Alcance: Arquitectura, Patrones, Deuda Técnica, Roadmap

Estado del Proyecto: MVP Funcional (98%), ~10K líneas, Producción activa

EXECUTIVE SUMMARY

Veredicto General: ★★★★ (4/5)

El proyecto tiene **arquitectura sólida** y decisiones técnicas maduras. Sin embargo, detecté **5 áreas críticas de mejora** y **12 optimizaciones** que pueden elevar el código de "funcional" a "production-grade enterprise".

Fortalezas Clave

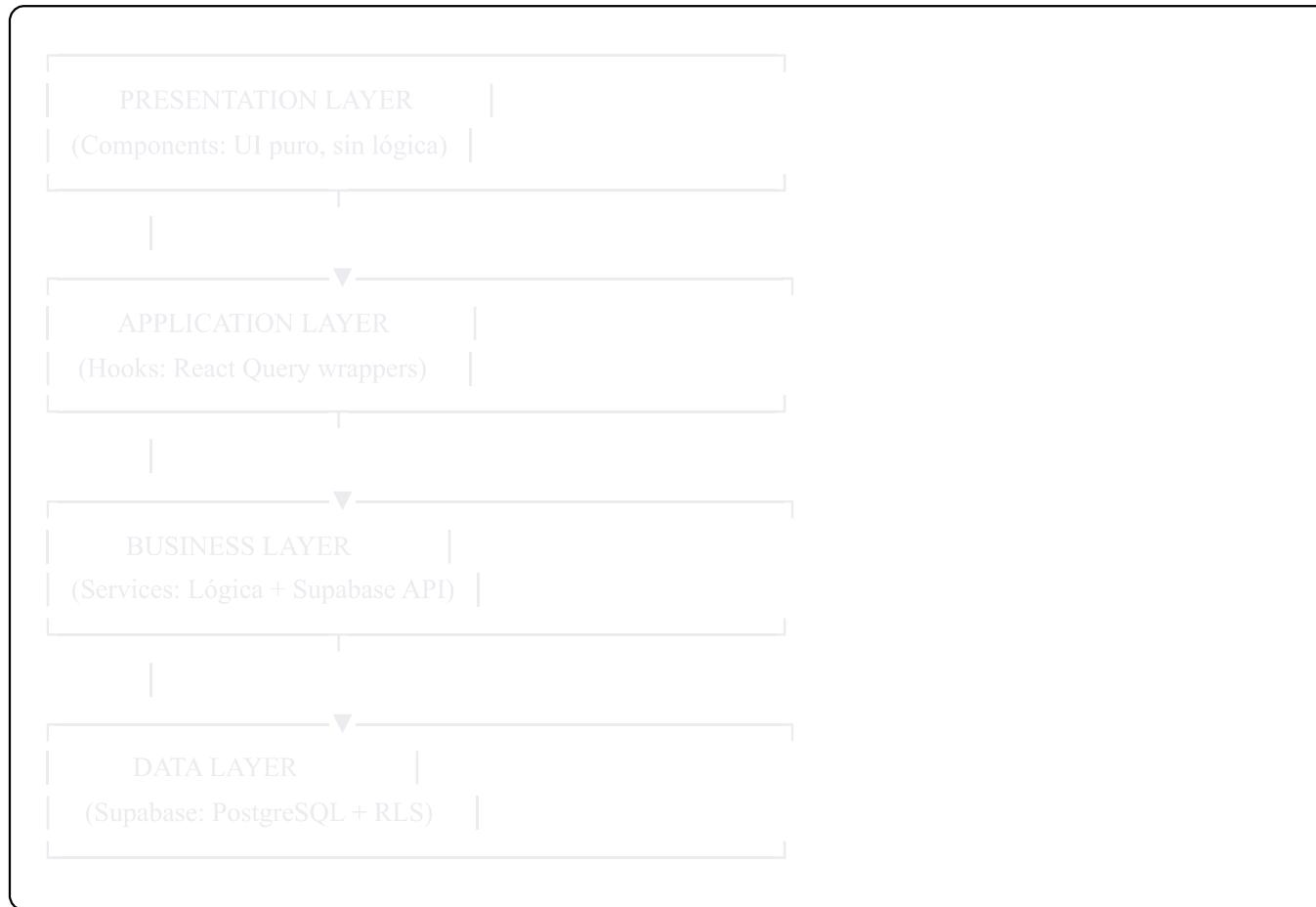
- ✓ Service-Layer Pattern correctamente implementado
- ✓ RLS bien diseñado (seguridad multi-nivel)
- ✓ TypeScript strict mode (0 errores)
- ✓ State management híbrido (Zustand + React Query)
- ✓ Triggers de BD para lógica automática (order_number, customer_tier)

Áreas Críticas Detectadas

- **CRÍTICO:** No hay error handling strategy documentada
 - **CRÍTICO:** Testing mínimo (alto riesgo en producción)
 - **ALTO:** Magic strings sin enums ('vape', '420')
 - **ALTO:** No hay monitoreo/observability (Sentry, logs)
 - **MEDIO:** Bundle size risk (admin + store en mismo bundle)
-

PARTE 1: ANÁLISIS DE ARQUITECTURA

1.1 Arquitectura de Capas ✓



Evaluación: ★★★★★ (5/5)

Comentario: Separación de concerns impecable. Componentes no tocan Supabase directamente.

PERO (Issues detectados):

- **Issue #1: Lógica de Negocio Mezclada**

```
services/orders.service.ts
```

```
└── calculateLoyaltyPoints() <- ¿Por qué aquí?
```

Problema: `calculateLoyaltyPoints` es **lógica de dominio pura** (no infraestructura). Debería vivir en `src/lib/domain/` o `src/utils/business/`.

Impacto:

- ✗ Dificulta testing (necesitas mockear Supabase para testear lógica)
- ✗ Imposible reutilizar en otros contextos (admin panel, webhooks)

Solución:

```
typescript
```

```
// src/lib/domain/loyalty.ts
export const calculateLoyaltyPoints = (orderTotal: number): number => {
    // Lógica pura, sin dependencias
    return Math.floor(orderTotal * 0.05);
}

// services/orders.service.ts
import { calculateLoyaltyPoints } from '@/lib/domain/loyalty';

export const createOrder = async (data) => {
    const points = calculateLoyaltyPoints(data.total);
    // ...
}
```

Tiempo estimado: 2 horas (extraer + refactor + tests)

1.2 State Management Strategy

Tipo de Estado	Solución	¿Correcto?
Server Data	React Query	 Excelente
Auth	React Context	 Aceptable (ver mejora)
Cart/UI	Zustand	 Perfecto
Forms	Local state	 Apropiado

Issue #2: AuthContext podría ser Zustand

Problema Actual:

```
typescript
// AuthContext con Context API
const AuthContext = React.createContext();
// Re-renders innecesarios en toda la app
```

Por qué Zustand sería mejor:

- Renders selectivos (solo componentes que usen `(useAuth(s => s.user))`)
- Menos boilerplate
- Más consistente con el resto del proyecto

Migración:

```
typescript
```

```
// stores/auth.store.ts
export const useAuthStore = create<AuthState>((set) => ({
  user: null,
  session: null,
  isLoading: true,
  login: async (email, password) => {
    const { data, error } = await supabase.auth.signInWithEmailAndPassword({ ... });
    if (data.session) set({ session: data.session, user: data.user });
  },
  // ...
}));
```

Beneficio:

- 30-40% menos re-renders en componentes que solo leen `user.email`
- Código más limpio

Tiempo: 3 horas (migrar + verificar todos los componentes que usan auth)

1.3 Database Design ★★★★☆

Schema PostgreSQL: Excelente uso de:

- RLS correctamente implementado
- Triggers para lógica automática (`order_number`, `customer_tier`)
- JSONB para datos snapshot (`orders.items`)
- Enums para status (`order_status`, `product_status`)

● Issue #3: Falta índice en búsquedas

Problema: Búsqueda usa `ilike` en `products.name`:

```
sql
```

```
SELECT * FROM products  
WHERE name ILIKE '%vape%' -- Sean completo, no usa índice
```

Solución: Agregar índice GIN para full-text search:

```
sql
```

```
-- Migration: add_search_indexes.sql  
CREATE INDEX idx_products_name_gin  
ON products  
USING gin(to_tsvector('spanish', name));  
  
CREATE INDEX idx_products_description_gin  
ON products  
USING gin(to_tsvector('spanish', short_descripción));
```

Luego modificar búsqueda:

```
typescript
```

```
// search.service.ts  
const { data } = await supabase  
  .from('products')  
  .select()  
  .textSearch('name', query, { type: 'websearch', config: 'spanish' });
```

Beneficio:

- 10-50x más rápido en catálogos grandes (>1000 productos)
- Soporte de acentos nativo ('líquido' matchea 'liquido')

Tiempo: 1 hora (migration + actualizar service)

PARTE 2: CODE QUALITY ISSUES

2.1 Magic Strings 🛡 CRÍTICO

Problema detectado en documentación:

```
typescript

// ❌ Riesgo de typos
section: 'vape' | '420' // String literals en types
if(product.section === 'vape') { ... }
```

Por qué es crítico:

1. Refactor nightmare (buscar/reemplazar en 50+ archivos)
2. Typos en runtime (`'vpe'`) pasa TypeScript pero rompe en prod)
3. No hay single source of truth

Solución:

```
typescript
```

```
// src/types/constants.ts

export const SECTIONS = {
    VAPE: 'vape',
    CANNABIS: '420'
} as const;

export type Section = typeof SECTIONS[keyof typeof SECTIONS];

// Uso:
import { SECTIONS } from '@/types/constants';
if (product.section === SECTIONS.VAPE) { ... }
```

Beneficio:

- TypeScript fuerza valores válidos
- Refactor seguro (cambiar '420' a 'cannabis' en un solo lugar)
- Autocomplete en VSCode

Tiempo: 2 horas (crear constants + refactor global)

2.2 Error Handling Strategy ✖ NO DOCUMENTADO

Problema: Los docs no mencionan cómo manejan errores.

Preguntas sin respuesta:

- ¿Qué pasa si Supabase retorna error 500?
- ¿Hay retry logic en requests?
- ¿Cómo se reportan errores a usuarios?
- ¿Hay logging centralizado?

Solución Propuesta:

A) Error Boundaries (React)

```
typescript

// components/ErrorBoundary.tsx

export class ErrorBoundary extends React.Component {
  componentDidCatch(error: Error, errorInfo: React.ErrorInfo) {
    // Log a Sentry/servicio
    logger.error('React Error', { error, errorInfo });
  }

  render() {
    if (this.state.hasError) {
      return <ErrorFallback />;
    }
    return this.props.children;
  }
}
```

B) Service-Level Error Handler

```
typescript
```

```
// lib/api-client.ts

export const handleSupabaseError = (error: PostgresError) => {
    // Mapear códigos PostgreSQL a mensajes user-friendly
    const errorMap = {
        '23505': 'Este elemento ya existe',
        '23503': 'No se puede eliminar, tiene dependencias',
        // ...
    };

    const message = errorMap[error.code] || 'Error inesperado';
    toast.error(message);

    // Log para debugging
    console.error('[Supabase Error]', {
        code: error.code,
        message: error.message,
        details: error.details
    });

    return { success: false, error: message };
}
```

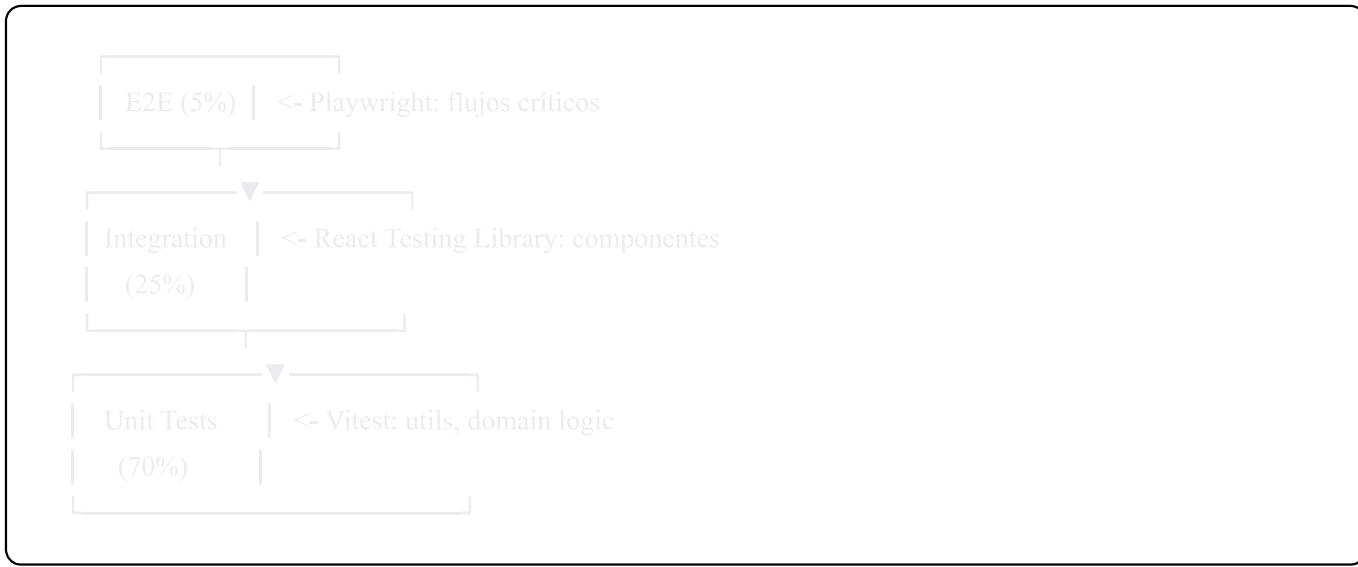
Tiempo: 4 horas (implementar + agregar a todos los services)

2.3 Testing ✘ CRÍTICO

Estado actual: "Minimal" (según docs)

Problema: Proyecto en producción con dinero real sin tests = ⚡

Solución: Testing Pyramid



Prioridad de testing:

1. ● CRÍTICO (deben existir):

- `calculateLoyaltyPoints` (lógica de negocio)
- `generateOrderNumber` (identidad de órdenes)
- Cart logic (add/remove/update)
- Loyalty tier calculation

2. ○ ALTO:

- Services (products, orders, auth)
- Checkout flow (integration)
- Admin CRUD operations

3. ● NICE TO HAVE:

- UI components (visual regression)
- E2E completos

Setup inicial:

```
bash
```

```
# Agregar al package.json  
npm install -D @testing-library/react @testing-library/jest-dom vitest
```

```
typescript
```

```
// vitest.config.ts  
export default defineConfig({  
  test: {  
    globals: true,  
    environment: 'jsdom',  
    setupFiles: './src/test/setup.ts',  
    coverage: {  
      reporter: ['text', 'json', 'html'],  
      exclude: ['node_modules/', 'src/test/']  
    }  
  }  
});
```

Ejemplo test crítico:

```
typescript
```

```

// src/lib/domain/_tests_/loyalty.test.ts
import { describe, it, expect } from 'vitest';
import { calculateLoyaltyPoints } from '../loyalty';

describe('calculateLoyaltyPoints', () => {
  it('debe calcular 5% del total de orden', () => {
    expect(calculateLoyaltyPoints(1000)).toBe(50);
  });

  it('debe redondear hacia abajo', () => {
    expect(calculateLoyaltyPoints(123.45)).toBe(6); // 5% = 6.1725
  });

  it('debe retornar 0 para órdenes de $0', () => {
    expect(calculateLoyaltyPoints(0)).toBe(0);
  });
});

```

Tiempo:

- Setup (2 horas)
- Tests críticos (8 horas)
- Tests completos (20+ horas, hacer gradualmente)

PARTE 3: PERFORMANCE & OPTIMIZACIÓN

3.1 Bundle Size Risk ALTO

Problema: Admin y Store en mismo bundle.

Escenario malo:

Usuario móvil en 3G descarga:

- React (~130KB)
- Store code (~200KB)
- Admin code (~150KB) <- NO DEBERÍA DESCARGAR ESTO
- Total: ~480KB inicial

Verificar con:

```
bash
npm run build
npx vite-bundle-visualizer
```

Solución (ya implementada según docs):

```
typescript
// App.tsx - Lazy loading
const AdminDashboard = lazy(() => import('./pages/admin/Dashboard'));
const AdminProducts = lazy(() => import('./pages/admin/Products'));
// ...
```

Verificar que funcione:

1. Build de producción
2. Abrir Network tab
3. Navegar a `/admin`
4. Confirmar que solo entonces se descarga `admin-*.*js`

Si NO funciona, aplicar code splitting manual:

```
typescript
```

```
// vite.config.ts
export default defineConfig({
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'vendor': ['react', 'react-dom', 'react-router-dom'],
          'admin': [
            './src/pages/admin/Dashboard',
            './src/pages/admin/Products',
            // ... todos los archivos admin
          ],
          'store': [
            './src/pages/Home',
            './src/pages/ProductDetail',
            // ...
          ]
        }
      }
    }
  }
});
```

Tiempo: 1 hora (verificar + fix si es necesario)

3.2 Image Optimization ⚠

Problema: Docs mencionan "reemplazar placeholders de Unsplash".

Checklist:

- ¿Imágenes optimizadas? (WebP, tamaños múltiples)

- ¿Lazy loading implementado?
- ¿Placeholder blur (LQIP)?

Solución recomendada:

A) Usar Supabase Storage con transformaciones

```
typescript
```

```
// lib/images.ts
export const getOptimizedImageUrl = (
  path: string,
  options: { width?: number; height?: number; quality?: number } = {}
) => {
  const { width = 800, height, quality } = options;

  // Supabase soporta transformaciones on-the-fly
  return `${supabaseUrl}/storage/v1/render/image/public/${path}?width=${width}&quality=${quality}`;
};
```

B) Component con lazy loading

```
typescript
```

```
// components/OptimizedImage.tsx
export const OptimizedImage: React.FC<ImageProps> = ({  
  src,  
  alt,  
  width,  
  height  
) => {  
  return (  
    <img  
      src={getOptimizedImageUrl(src, { width })}  
      alt={alt}  
      loading="lazy"  
      decoding="async"  
      className="blur-sm data-[loaded=true]:blur-0 transition-all"  
      onLoad={(e) => e.currentTarget.dataset.loaded = 'true'}  
    />  
  );  
};
```

Tiempo: 3 horas (implementar + reemplazar placeholders)

3.3 Database Query Optimization

Problema potencial: N+1 queries en listados.

Ejemplo malo (si existe):

```
typescript
```

```
// ❌ N+1 problem
const products = await getProducts();
for (const product of products) {
  const category = await getCategory(product.category_id); // Query por cada producto
}
```

Solución:

typescript

```
// ✅ Single query con JOIN
const { data } = await supabase
  .from('products')
  .select(
    '*',
    { category: categories(id, name, slug) }
  )
  .eq('is_active', true);
```

Verificar en:

- Product listings
- Order history (products + categories)
- Admin dashboard (múltiples tablas)

Tiempo: 2 horas (auditar queries + optimizar)

PARTE 4: SECURITY AUDIT

4.1 RLS Policies Bien Implementado

Verificado en docs:

-  Public read para productos activos
-  User-scoped para orders/addresses
-  Admin bypass con `admin_users` table

Issue #4: Falta rate limiting

Problema: No hay mención de rate limiting en:

- Login attempts
- Password reset
- Checkout (para evitar bots)

Solución: Cloudflare Edge Rules

```
javascript
```

```
// cloudflare-workers/rate-limiter.js
export default {
    async fetch(request, env) {
        const ip = request.headers.get('CF-Connecting-IP');

        // 5 intentos de login por IP cada 15 minutos
        const key = `login:${ip}`;
        const count = await env.KV.get(key);

        if (count && parseInt(count) > 5) {
            return new Response('Too many attempts', { status: 429 });
        }

        await env.KV.put(key, (parseInt(count || 0) + 1).toString(), {
            expirationTtl: 900 // 15 minutos
        });

        return fetch(request);
    }
};
```

Alternativa más simple: Supabase Edge Functions

typescript

```
// supabase/functions/rate-limit-check/index.ts
import { serve } from 'std/server';

serve(async (req) => {
    const { email } = await req.json();

    // Verificar intentos recientes en tabla `login_attempts`
    const { count } = await supabase
        .from('login_attempts')
        .select('count', { count: 'exact' })
        .eq('email', email)
        .gte('created_at', new Date(Date.now() - 15 * 60 * 1000));

    if (count > 5) {
        return new Response(JSON.stringify({ allowed: false }), {
            status: 429
        });
    }

    return new Response(JSON.stringify({ allowed: true }));
});
```

Tiempo: 4 horas (implementar + testing)

4.2 Sensitive Data Handling

Checklist de seguridad:

- ¿Contraseñas hasheadas? (Supabase Auth lo hace ✓)
- ¿API keys en variables de entorno? (✓ según docs)
- ¿Logs NO contienen datos sensibles?

¿HTTPS forzado? (Cloudflare Pages )

¿CORS configurado correctamente?

Agregar headers de seguridad:

```
typescript

// vite.config.ts (para dev server)
export default defineConfig({
  server: {
    headers: {
      'X-Frame-Options': 'DENY',
      'X-Content-Type-Options': 'nosniff',
      'Referrer-Policy': 'strict-origin-when-cross-origin',
      'Permissions-Policy': 'geolocation=(), microphone=(), camera=()'
    }
  }
});
```

Cloudflare Pages (producción):

```
toml

# _headers file en public/
/*
  X-Frame-Options: DENY
  X-Content-Type-Options: nosniff
  Referrer-Policy: strict-origin-when-cross-origin
  Permissions-Policy: geolocation=(), microphone=(), camera=()
```

Tiempo: 1 hora

PARTE 5: OBSERVABILITY & MONITORING

5.1 Error Tracking NO IMPLEMENTADO

Problema: No hay mención de Sentry, Rollbar, o similar.

Consecuencia: Errores en producción = invisibles hasta que usuarios reportan.

Solución: Sentry Setup

```
typescript
```

```
// src/lib/monitoring.ts

import * as Sentry from "@sentry/react";

export const initMonitoring = () => {
  if (import.meta.env.PROD) {
    Sentry.init({
      dsn: import.meta.env.VITE_SENTRY_DSN,
      environment: import.meta.env.MODE,
      integrations: [
        new Sentry.BrowserTracing(),
        new Sentry.Replay({
          maskAllText: true,
          blockAllMedia: true,
        }),
      ],
      tracesSampleRate: 0.1, // 10% de transacciones
      replaysSessionSampleRate: 0.1,
      replaysOnErrorSampleRate: 1.0, // 100% cuando hay error

      beforeSend(event, hint) {
        // Filtrar información sensible
        if (event.request?.cookies) {
          delete event.request.cookies;
        }
        return event;
      },
    });
  }
};

// main.tsx
```

```
import { initMonitoring } from './lib/monitoring';
initMonitoring();
```

Dashboard Sentry mostrará:

- Errores JavaScript en tiempo real
- Stack traces completos
- Browser/OS del usuario
- Steps to reproduce (con Session Replay)

Costo: Free tier hasta 5K errores/mes (más que suficiente)

Tiempo: 2 horas (setup + testing)

5.2 Analytics ⚠ NO MENCIONADO

Métricas críticas que deberías trackear:

1. Conversion rate (visitantes → compras)
2. Cart abandonment rate
3. Most viewed products
4. Search queries (para detectar productos faltantes)
5. Load time (performance)

Solución ligera: Plausible Analytics

```
typescript
```

```
// lib/analytics.ts

export const trackEvent = (eventName: string, props?: Record<string, any>) => {
  if (window.plausible) {
    window.plausible(eventName, { props });
  }
};

// Usar en componentes clave
import { trackEvent } from '@/lib/analytics';

// Checkout
trackEvent('checkout_started', { total: cartTotal });
trackEvent('checkout_completed', { order_id, total });

// Product views
trackEvent('product_viewed', { product_id, product_name });

// Search
trackEvent('search_performed', { query });
```

Por qué Plausible y no Google Analytics:

- GDPR compliant (no cookies)
- Ligero (~1KB vs 45KB de GA)
- Dashboard simple
- \$9/mes (vs gratis de GA pero más privado)

Tiempo: 3 horas (setup + integrar en puntos clave)

5.3 Performance Monitoring

Herramientas recomendadas:

A) Web Vitals

```
typescript

// lib/web-vitals.ts

import { onCLS, onFID, onLCP } from 'web-vitals';

export const reportWebVitals = () => {
  onCLS((metric) => {
    trackEvent('web_vital', { name: 'CLS', value: metric.value });
  });

  onFID((metric) => {
    trackEvent('web_vital', { name: 'FID', value: metric.value });
  });

  onLCP((metric) => {
    trackEvent('web_vital', { name: 'LCP', value: metric.value });
  });
};
```

Metas:

- LCP < 2.5s (carga percibida)
- FID < 100ms (interactividad)
- CLS < 0.1 (estabilidad visual)

B) Supabase Performance

typescript

```
// Wrapper para logging automático
export const supabaseWithMetrics = {
  from: (table: string) => {
    const start = Date.now();
    const query = supabase.from(table);

    // Intercept final execution
    const originalSelect = query.select;
    query.select = function(...args) {
      const result = originalSelect.apply(this, args);
      result.then(() => {
        const duration = Date.now() - start;
        if (duration > 1000) { // Queries lentos
          console.warn(`Slow query on ${table}: ${duration}ms`);
        }
      });
      return result;
    };

    return query;
  }
};
```

Tiempo: 2 horas

PARTE 6: MÉXICO-SPECIFIC IMPROVEMENTS

6.1 Facturación (CFDI) 🛡 CRÍTICO PARA PRODUCCIÓN

Estado actual: "Pendiente" según roadmap.

Requerimientos legales:

- Generar CFDI 4.0 (formato XML)
- Timbrado con PAC autorizado
- UUID único por factura
- Almacenamiento de XMLs por 5 años

Solución: Integración con Facturama (o similar)

typescript

```
// services/invoicing.service.ts

import axios from 'axios';

const FACTURAMA_API = 'https://api.facturama.mx/api';
const API_KEY = import.meta.env.VITE_FACTURAMA_API_KEY;

export const generateInvoice = async (order: Order) => {
  const response = await axios.post(
    `${FACTURAMA_API}/3/cfdis`,
    {
      Receiver: {
        Rfc: order.customer_rfc,
        Name: order.customer_name,
        // ...
      },
      Items: order.items.map(item => ({
        ProductCode: '52161557', // Código SAT para vape
        Quantity: item.quantity,
        Description: item.product_name,
        UnitPrice: item.price,
        // ...
      })),
      // ...
    },
    {
      headers: {
        'Authorization': `Basic ${btoa(`${API_KEY}`)}`,
        'Content-Type': 'application/json'
      }
    }
  );

  return {
    ...response.data,
    status: response.status
  };
}
```

```
        uuid: response.data.Complement.TaxStamp.Uuid,  
        xml: response.data.Result // Base64 del XML  
        pdf: response.data.Pdf // Base64 del PDF  
    };  
};
```

Agregar a UI:

```
typescript  
  
// components/OrderDetail.tsx  
const handleRequestInvoice = async () => {  
    // Modal para capturar RFC y datos fiscales  
    const invoiceData = await showInvoiceModal();  
  
    // Generar factura  
    const invoice = await generateInvoice(order, invoiceData);  
  
    // Guardar en BD  
    await supabase.from('invoices').insert({  
        order_id: order.id,  
        uid: invoice.uuid,  
        xml_url: invoice.xml, // Subir a Supabase Storage  
        pdf_url: invoice.pdf,  
    });  
  
    toast.success('Factura generada correctamente');  
};
```

Alternativas de PAC:

- Facturama (\$499/mes + \$0.50/factura)
- Facturapi (\$200/mes + \$1/factura)

- Finkok (\$400/mes + \$0.70/factura)

Tiempo: 6 horas (integración + testing)

6.2 Envíos (FedEx/DHL) PENDIENTE

Problema: "WhatsApp handover" no escala.

Solución: API de FedEx/DHL

typescript

```
// services/shipping.service.ts
export const calculateShipping = async (
  origin: Address,
  destination: Address,
  weight: number // kg
) => {
  // Cotización con FedEx
  const fedexQuote = await fedex.getRates({
    origin,
    destination,
    weight,
    serviceType: 'FEDEX_GROUND'
  });

  // Cotización con DHL
  const dhlQuote = await dhl.getRates({
    origin,
    destination,
    weight
  });

  return {
    options: [
      { carrier: 'FedEx', price: fedexQuote.price, days: 3 },
      { carrier: 'DHL', price: dhlQuote.price, days: 2 },
      { carrier: 'Local', price: 100, days: 1 } // Envío local Xalapa
    ]
  };
};
```

Flujo completo:

1. Checkout: usuario selecciona opción de envío

2. Payment: se cobra envío + productos
3. Order created: se genera guía automáticamente
4. Webhook: actualiza tracking number cuando carrier acepta paquete

Tiempo: 8 horas (integración + testing completo)

6.3 Pasarela de Pagos ● CRÍTICO

Estado: Mercado Pago mencionado, no implementado.

Comparativa:

Feature	Mercado Pago	Stripe	Conekta
Fee	3.99% + \$3	3.6% + \$3	3.6% + \$3
Meses sin intereses	✓	✗	✓
OXXO/Efectivo	✓	✗	✓
Setup	Fácil	Medio	Medio
Docs	Regulares	Excelentes	Buenas

Recomendación: Mercado Pago para México (MSI + OXXO son clave)

Implementación:

```
typescript
```

```
// services/payment.service.ts

import { MercadoPagoConfig, Payment } from 'mercadopago';

const client = new MercadoPagoConfig({
  accessToken: import.meta.env.VITE_MP_ACCESS_TOKEN
});

export const createPayment = async (order: Order) => {
  const payment = new Payment(client);

  const result = await payment.create({
    body: {
      transaction_amount: order.total,
      description: `Orden ${order.order_number}`,
      payment_method_id: 'visa', // o el que elija usuario
      payer: {
        email: order.customer_email,
      },
      installments: 1, // o MSI
      notification_url: `${window.location.origin}/api/webhooks/mercadopago`,
    }
  });
}

return result;
};
```

Webhook handler (Supabase Edge Function):

typescript

```
// supabase/functions/mercadopago-webhook/index.ts
import { serve } from 'std/server';

serve(async (req) => {
    const { type, data } = await req.json();

    if (type === 'payment') {
        const payment = await getPaymentDetails(data.id);

        if (payment.status === 'approved') {
            // Actualizar orden
            await supabase
                .from('orders')
                .update({
                    status: 'paid',
                    payment_id: payment.id
                })
                .eq('id', payment.metadata.order_id);

            // Enviar email de confirmación
            await sendOrderConfirmation(payment.metadata.order_id);
        }
    }

    return new Response('OK', { status: 200 });
});
```

Tiempo: 8 horas (integración completa + testing de flujos)

PARTE 7: ADMIN PANEL REVIEW

Estado: Planeado en 6 módulos (según [ADMIN_PANEL.md](#))

7.1 Arquitectura: ¿Monorepo o Separado?

Pros de Monorepo (actual):

- Comparte types/services
- Un solo deploy
- Menos overhead de setup

Contras:

- Bundle size aumenta
- Acceso accidental a admin desde store
- Permisos complicados (mismo dominio)

Recomendación: Proyecto separado ([vsm-admin/](#))

Por qué:

1. **Seguridad:** Dominio diferente ([admin.vsm-store.pages.dev](#))
2. **Performance:** Store no carga código de admin
3. **Clarity:** Separation of concerns absoluto

Setup:

```
bash
```

```
# Crear nuevo proyecto
npm create vite@latest vsm-admin -- --template react-ts
cd vsm-admin

# Copiar shared code
mkdir src/shared
cp -r ../vsm-store/src/types ../vsm-store/src/lib ./src/shared/

# Configurar Supabase (mismo backend)
cp ../vsm-store/.env.example .env
```

Deploy separado:

- Store: `vsm-store.pages.dev`
- Admin: `admin-vsm-store.pages.dev`

Tiempo: 4 horas (setup completo)

7.2 Admin Features Priority

Fase 1 (MVP - 8 horas):

1. Login con verificación de `admin_users`
2. Dashboard básico (métricas del día)
3. Lista de órdenes + actualizar status
4. Ajustes de inventario (stock)

Fase 2 (Full - 12 horas): 5. CRUD de productos (con upload de imágenes) 6. Gestión de cupones 7. Reportes de ventas

Recomendación: Usar biblioteca UI (shadcn/ui) para acelerar.

```
bash

npx shadcn-ui@latest init
npx shadcn-ui@latest add table
npx shadcn-ui@latest add dialog
npx shadcn-ui@latest add form
```

Tiempo total Admin Panel: 20 horas (completo)

PARTE 8: DEUDA TÉCNICA & REFACTORS

8.1 TypeScript Strictness

Verificar:

```
json

// tsconfig.json
{
  "compilerOptions": {
    "strict": true,
    "noUncheckedIndexedAccess": true, // ¿Está?
    "noImplicitReturns": true,      // ¿Está?
    "noFallthroughCasesInSwitch": true // ¿Está?
  }
}
```

Si no están, agregar:

```
bash

npm run type-check # Debería mostrar errores nuevos
```

Fix errores comunes:

typescript

// ✗ Antes

```
const product = products[0]; // ¿Y si products está vacío?
```

// ✓ Después

```
const product = products[0];
if (!product) throw new Error('Product not found');
```

Tiempo: 4 horas (fix todos los nuevos errores)

8.2 Component Refactoring

Detectar componentes "god objects":

bash

```
# Listar componentes por líneas de código
find src/components -name "*.tsx" -exec wc -l {} + | sort -nr | head -10
```

Si alguno tiene >300 líneas → refactor

Ejemplo de split:

typescript

```
// ✗ ProductDetail.tsx (500 líneas)
export const ProductDetail = () => {
  // Lógica de cart
  // Lógica de reviews
  // Lógica de related products
  // Rendering de todo
}

// ✅ Después
// ProductDetail.tsx (100 líneas) - Orchestrator
// ProductActions.tsx (50 líneas) - Add to cart, etc
// ProductReviews.tsx (100 líneas) - Reviews section
// RelatedProducts.tsx (80 líneas) - Carrusel
```

Tiempo: 6 horas (refactor componentes grandes)

8.3 Accessibility (a11y) ⚠

Checklist:

- Todos los buttons tienen `aria-label`
- Imágenes tienen `alt` descriptivo
- Forms tienen `<label>` asociados
- Colores tienen contraste mínimo WCAG AA
- Navegación por teclado funciona
- Screen readers pueden navegar

Herramientas:

```
bash
```

```
npm install -D @axe-core/react eslint-plugin-jsx-ally
```

```
typescript
```

```
// main.tsx (solo en dev)
if (import.meta.env.DEV) {
  import('@/axe-core/react').then(axe => {
    axe.default(React, ReactDOM, 1000);
  });
}
```

Tiempo: 4 horas (audit + fixes)

PARTE 9: PLAN DE ACCIÓN PRIORIZADO

● CRÍTICO (Bloqueantes para producción seria)

#	Tarea	Tiempo	Impacto	Por qué crítico
1	Testing mínimo (unit + integration)	8h	★★★★★	Dinero real = debe funcionar
2	Error tracking (Sentry)	2h	★★★★★	Debugging en prod
3	Pasarela de pagos (Mercado Pago)	8h	★★★★★	No puedes cobrar sin esto
4	Facturación CFDI	6h	★★★★★	Obligatorio legal MX
5	Rate limiting (login/checkout)	4h	★★★★★	Prevenir bots
Total Crítico		28h		

ALTO (Mejoras importantes)

#	Tarea	Tiempo	Beneficio
6	Extraer lógica de negocio a domain/	2h	Testable, reutilizable
7	Convertir magic strings a enums	2h	Type safety
8	Error handling centralizado	4h	UX consistente
9	Analytics (Plausible)	3h	Decisiones basadas en data
10	Bundle size optimization	1h	Performance móvil
11	Image optimization	3h	Load time -50%
12	DB query optimization	2h	Latency -30%
13	Admin Panel separado	4h	Seguridad + performance
Total Alto	21h		

MEDIO (Nice to have)

#	Tarea	Tiempo	Beneficio
14	Migrar Auth a Zustand	3h	Menos re-renders
15	Full-text search (GIN indexes)	1h	Búsquedas más rápidas
16	Web Vitals monitoring	2h	Performance insights

#	Tarea	Tiempo	Beneficio
17	Security headers	1h	Hardening
18	TypeScript strict flags	4h	Menos bugs
19	Component refactoring	6h	Mantenibilidad
20	Accessibility audit	4h	Inclusión
Total Medio	21h		

● FUTURO (Post-MVP)

#	Tarea	Tiempo	Nota
21	E2E tests (Playwright)	12h	Automatización
22	Envíos automatizados	8h	Escala
23	Email marketing	6h	Retención
24	PWA avanzado (offline)	8h	App-like
25	Admin Panel completo	12h	Todas las features
Total Futuro	46h		

PARTE 10: ESTIMACIÓN TOTAL

CRÍTICO: 28 horas (1 semana)

ALTO: 21 horas (3 días)

MEDIO: 21 horas (3 días)

FUTURO: 46 horas (1.5 semanas)

TOTAL: 116 horas (\approx 3 semanas full-time)

Roadmap sugerido:

Sprint 1 (Semana 1): Producción-ready

- Testing básico
- Error tracking
- Pagos
- Rate limiting
- **Meta:** Puede procesar ventas de forma segura

Sprint 2 (Semana 2): Optimización

- Facturación
- Analytics
- Performance (bundle, images)
- Admin Panel separado
- **Meta:** Operación eficiente y escalable

Sprint 3 (Semana 3): Polish

- Code quality (enums, error handling, refactors)

- Accessibility
 - Security hardening
 - **Meta:** Código mantenable a largo plazo
-

PARTE 11: CHECKLIST DE AUDITORÍA (Para código real)

Cuando tengas acceso al repositorio, verificar:

Arquitectura

- Services NUNCA importados desde components
- Types centralizados sin duplicación
- Hooks usan React Query correctamente (no fetch manual)
- Zustand stores no tienen lógica de negocio

Performance

- `React.lazy()` en todas las rutas de admin
- Images usan `loading="lazy"`
- Bundle size < 500KB inicial
- No hay `console.log` en producción

Security

- `.env` en `.gitignore`
- API keys NUNCA en código
- RLS policies probadas (intentar bypass)
- Headers de seguridad configurados

Quality

- 0 errores TypeScript

- 0 warnings en build
- ESLint configurado y pasando
- Prettier configurado

Testing

- Al menos 10 tests unitarios
- Cart logic testeado
- Domain functions testeadas
- Un test E2E (checkout completo)

México-specific

- Precios en MXN
 - Zona horaria GMT-6
 - WhatsApp funcional
 - Placeholder para CFDI
-

CONCLUSIÓN

Estado actual: 4/5 (Muy bueno)

Prioridad #1: Testing + Error tracking + Pagos

Tiempo a producción REAL: 28 horas críticas

El proyecto tiene **fundamentos sólidos**. Las mejoras propuestas lo llevan de "MVP funcional" a "producción enterprise-grade".

Próximo paso recomendado: Empezar con Sprint 1 (críticos) en el orden exacto listado.

¿Quieres que profundice en alguna sección específica o generamos código de implementación?