🏗️ FASE 1: Estructura Base - Arquitectura Separada

📐 Arquitectura del Sistema

```
proyecto-optica/
│
├── frontend/          # React + TypeScript + Vite
│   ├── src/
│   ├── public/
│   └── package.json
│
├── backend/           # Node.js + Express + TypeScript
│   ├── src/
│   ├── dist/
│   └── package.json
│
└── database/          # PostgreSQL + Prisma
    ├── prisma/
    └── package.json
```

🗂️ PASO 0: Preparación del Entorno

Requisitos Previos

bash

```
# Instalar Node.js 18+ y npm
node --version  # v18.x.x o superior
npm --version   # 9.x.x o superior

# Instalar PostgreSQL 15+
psql --version  # 15.x o superior

# (Opcional) Instalar Docker para PostgreSQL
docker --version
```

Estructura Inicial

bash

```
# Crear carpeta raíz del proyecto
mkdir proyecto-optica-ecommerce
cd proyecto-optica-ecommerce

# Crear subcarpetas
mkdir frontend backend database

# Inicializar git (opcional)
git init
echo "node_modules/
.env
dist/
build/" > .gitignore
```

🗄️ PARTE 1: BASE DE DATOS

📝 Prompt para IA - Base de Datos

Actúa como arquitecto de bases de datos especializado en e-commerce de productos ópticos.

CONTEXTO:
Diseñar la base de datos PostgreSQL para un e-commerce de óptica con productos (monturas, lentes, accesorios), clientes, órdenes e inventario.

ARQUITECTURA:
- PostgreSQL 15+
- Prisma ORM como capa de abstracción
- Separación total de frontend/backend

REQUERIMIENTOS TÉCNICOS:

1. TABLAS PRINCIPALES:
   - users (clientes y administradores)
   - products (productos del catálogo)
   - categories (monturas sol, oftálmicas, lentes, accesorios)
   - orders (órdenes de compra)
   - order_items (productos en cada orden)
   - cart (carritos activos)
   - addresses (direcciones de envío)
   - inventory (control de stock)

2. RELACIONES:
   - User 1:N Orders (un usuario puede tener muchas órdenes)
   - Order 1:N OrderItems (una orden tiene muchos productos)
   - Product 1:N OrderItems
   - Product N:1 Category
   - User 1:N Addresses

- Product 1:1 Inventory

3. CARACTERÍSTICAS ESPECÍFICAS ÓPTICA:
- Productos con atributos variables (material, color, forma, género)
- Sistema de graduación/prescripción
- Marcas de prestigio (Ray-Ban, Oakley, etc.)
- Control de stock por SKU único

ESQUEMA PRISMA REQUERIDO:
```prisma
// Incluir modelos para:
// - User (con roles: CLIENTE, ADMIN)
// - Category (enum: MONTURAS_SOL, MONTURAS_OFTALMICA, LENTES_CONTACTO, ACCESORIOS)
// - Product (con campos JSON para características variables)
// - Order (con estados: PENDIENTE, PAGADO, ENVIADO, ENTREGADO, CANCELADO)
// - OrderItem
// - Cart
// - Address
// - Inventory

// Índices para optimización:
// - email en User (unique)
// - SKU en Product (unique)
// - userId en Orders
// - productId en Inventory
```

4. SEEDS INICIALES (datos de prueba):
- 1 usuario admin: admin@optivision.com / Admin123!
- 2 usuarios cliente
- 4 categorías
- 20 productos ejemplo con stock
- 5 órdenes de ejemplo

DATOS REALISTAS DE PRODUCTOS:

```typescript
// Ejemplos de productos
{
  nombre: "Ray-Ban Aviator Clásico RB3025",
  marca: "Ray-Ban",
  precio: 89990,
  categoria: "MONTURAS_SOL",
  caracteristicas: {
    material: "metal",
    color: "dorado",
    forma: "aviador",
    genero: "unisex",
    proteccionUV: true
  },
  stock: 15
}

{
  nombre: "Oakley Flak 2.0 XL Polarizado",
  marca: "Oakley",
  precio: 124990,
  categoria: "MONTURAS_SOL",
  caracteristicas: {
    material: "plastico_resistente",
    color: "negro_mate",
    forma: "deportivo",
    genero: "unisex",
    polarizado: true
  },
  stock: 8
}
```

```

ENTREGABLES:
1. Schema completo de Prisma (prisma/schema.prisma)
2. Archivo de migración inicial
3. Seed con datos de prueba (prisma/seed.ts)
4. Scripts de npm para:
   - npm run db:migrate
   - npm run db:seed
   - npm run db:studio (Prisma Studio)
5. Documentación del modelo de datos
6. Archivo .env.example con variables necesarias

PASOS DE IMPLEMENTACIÓN:
1. Inicializar Prisma en carpeta database/
2. Definir schema completo
3. Crear migración inicial
4. Implementar seed con datos realistas
5. Validar con Prisma Studio

## 🚀 Comandos para Ejecutar (Base de Datos)

bash
```

```
# 1. Navegar a carpeta database
cd database

# 2. Inicializar proyecto Node
npm init -y

# 3. Instalar dependencias
npm install prisma @prisma/client typescript ts-node @types/node --save-dev

# 4. Inicializar Prisma
npx prisma init

# 5. Copiar el schema generado por IA en: prisma/schema.prisma

# 6. Configurar variables de entorno
# Editar .env con:
DATABASE_URL="postgresql://usuario:password@localhost:5432/optica_db?schema=public"

# 7. Crear base de datos
createdb optica_db
# O con Docker:
# docker run --name postgres-optica -e POSTGRES_PASSWORD=mipassword -p 5432:5432 -d postgres:15

# 8. Ejecutar migración
npx prisma migrate dev --name init

# 9. Ejecutar seed
npx prisma db seed

# 10. Abrir Prisma Studio para validar
npx prisma studio
```

🔧 PARTE 2: BACKEND (API REST)

📝 Prompt para IA - Backend

> Actúa como arquitecto backend senior especializado en APIs REST para e-commerce.
>
> CONTEXTO:
> Desarrollar API RESTful con Node.js + Express + TypeScript para e-commerce de óptica.
> La base de datos ya está configurada con Prisma.
>
> ARQUITECTURA BACKEND:

```
backend/
├── src/
│   ├── config/          # Configuraciones (DB, env, cors)
│   ├── controllers/     # Lógica de negocio
│   ├── routes/          # Definición de endpoints
│   ├── middlewares/     # Auth, validación, error handling
│   ├── services/        # Servicios de negocio
│   ├── types/           # Tipos TypeScript
│   ├── utils/           # Utilidades
│   └── server.ts        # Punto de entrada
├── prisma/              # Cliente Prisma (desde database/)
├── .env
├── .env.example
├── tsconfig.json
└── package.json
```

STACK TÉCNICO:
- Node.js 18+ con TypeScript 5+
- Express.js 4.18+
- Prisma Client (ya configurado)
- JWT para autenticación
- bcrypt para hashing de passwords
- express-validator para validaciones
- cors para frontend separado
- dotenv para variables de entorno

ENDPOINTS REQUERIDOS (FASE 1):

1. AUTENTICACIÓN (público):
```
POST   /api/auth/register      # Registro de usuario
POST   /api/auth/login         # Login
POST   /api/auth/logout        # Logout
GET    /api/auth/me            # Usuario actual (requiere token)
```

2. PRODUCTOS (público lectura, admin escritura):
```
GET    /api/products           # Listar productos (con filtros, paginación)
GET    /api/products/:id       # Detalle de producto
POST   /api/products           # Crear producto [ADMIN]
PUT    /api/products/:id       # Actualizar producto [ADMIN]
DELETE /api/products/:id       # Eliminar producto [ADMIN]
```

3. CATEGORÍAS (público):
```
GET    /api/categories         # Listar categorías
```

4. CARRITO (requiere auth):
```
GET    /api/cart               # Ver carrito del usuario
POST   /api/cart/items         # Agregar producto al carrito
PUT    /api/cart/items/:id     # Actualizar cantidad
```

DELETE /api/cart/items/:id # Eliminar del carrito
DELETE /api/cart # Vaciar carrito

5. USUARIOS (requiere auth):
GET /api/users/profile # Perfil del usuario
PUT /api/users/profile # Actualizar perfil

MIDDLEWARES REQUERIDOS:

1. authMiddleware.ts:
```typescript
// Verificar JWT token
// Agregar user a req.user
// Si no hay token o es inválido, retornar 401
```

2. adminMiddleware.ts:
```typescript
// Verificar que user.role === 'ADMIN'
// Si no es admin, retornar 403
```

3. validationMiddleware.ts:
```typescript
// Validar body con express-validator
// Schemas para: registro, login, crear producto, etc.
```

4. errorHandler.ts:
```typescript
// Centralizar manejo de errores
// Formato JSON consistente
// Log de errores en servidor
```

```

FORMATO DE RESPUESTAS:
```typescript
// Éxito
{
  success: true,
  data: { ... },
  message: "Operación exitosa"
}

// Error
{
  success: false,
  error: "Mensaje de error",
  details: [ ... ] // Detalles de validación
}

// Lista con paginación
{
  success: true,
  data: [ ... ],
  pagination: {
    page: 1,
    limit: 20,
    total: 150,
    totalPages: 8
  }
}
```

SEGURIDAD:
- Helmet.js para headers de seguridad
```

- Rate limiting (express-rate-limit)
- CORS configurado solo para frontend (http://localhost:5173)
- Sanitización de inputs
- Passwords hasheados con bcrypt (salt rounds: 10)
- JWT con expiración de 7 días

VARIABLES DE ENTORNO (.env):

PORT=4000 DATABASE_URL="postgresql://..." JWT_SECRET="tu_secreto_super_seguro_aqui" JWT_EXPIRES_IN="7d" NODE_ENV="development" FRONTEND_URL="http://localhost:5173"

EJEMPLOS DE CONTROLADORES:

```typescript
// authController.ts
export const register = async (req: Request, res: Response) => {
  // 1. Validar datos
  // 2. Verificar si email ya existe
  // 3. Hashear password
  // 4. Crear usuario en BD
  // 5. Generar JWT
  // 6. Retornar usuario y token
}

// productsController.ts
export const getProducts = async (req: Request, res: Response) => {
  // 1. Extraer query params (category, marca, minPrice, maxPrice, page, limit)
  // 2. Construir filtros para Prisma
  // 3. Consultar productos con paginación
  // 4. Retornar productos y metadata de paginación
}
```

ENTREGABLES:

1. Servidor Express completamente configurado
2. Todos los endpoints funcionando
3. Middlewares implementados
4. Documentación de API (puede ser README.md con ejemplos cURL)
5. Scripts de npm:
   - npm run dev (con nodemon)
   - npm run build
   - npm run start
6. Colección de Postman/Thunder Client para testing
7. Manejo de errores robusto

TESTING:
Proporciona ejemplos cURL para cada endpoint, ejemplo:

```bash
# Registro
curl -X POST http://localhost:4000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "cliente@example.com",
    "password": "Password123!",
    "name": "Juan Pérez",
    "phone": "3001234567"
  }'

# Login
curl -X POST http://localhost:4000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "cliente@example.com",
    "password": "Password123!"
  }'

# Listar productos
curl http://localhost:4000/api/products?category=MONTURAS_SOL&page=1&limit=10
```

🚀 Comandos para Ejecutar (Backend)

bash

```
# 1. Navegar a carpeta backend
cd ../backend

# 2. Inicializar proyecto
npm init -y

# 3. Instalar dependencias
npm install express cors dotenv bcryptjs jsonwebtoken express-validator helmet express-rate-limit

# 4. Instalar dependencias de desarrollo
npm install -D typescript @types/node @types/express @types/cors @types/bcryptjs @types/jsonwebtoken ts-node no

# 5. Copiar Prisma Client desde database
npm install @prisma/client
# Luego copiar carpeta prisma/ desde ../database/

# 6. Inicializar TypeScript
npx tsc --init

# 7. Configurar tsconfig.json (modificar):
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true
  }
}

# 8. Configurar package.json scripts:
```

```
"scripts": {
  "dev": "nodemon src/server.ts",
  "build": "tsc",
  "start": "node dist/server.js"
}

# 9. Crear archivo .env con las variables
cp .env.example .env
# Editar con tus valores

# 10. Copiar código generado por IA

# 11. Ejecutar en modo desarrollo
npm run dev

# 12. Probar endpoints con cURL o Postman
curl http://localhost:4000/api/products
```

🎨 PARTE 3: FRONTEND

📝 Prompt para IA - Frontend

Actúa como desarrollador frontend senior especializado en React + TypeScript para e-commerce.

CONTEXTO:
Desarrollar la interfaz de usuario del e-commerce de óptica conectado a la API REST ya creada.
Diseño moderno, limpio, con paleta neutra y acento azul/verde, inspirado en las referencias proporcionadas.

ARQUITECTURA FRONTEND:

```
frontend/
├── src/
│   ├── api/              # Servicios para llamar al backend
│   ├── components/       # Componentes reutilizables
│   │   ├── common/       # Botones, inputs, modales
│   │   ├── layout/       # Header, Footer, Sidebar
│   │   └── product/      # ProductCard, ProductGrid
│   ├── pages/            # Páginas principales
│   │   ├── Home.tsx
│   │   ├── Products.tsx
│   │   ├── ProductDetail.tsx
│   │   ├── Login.tsx
│   │   ├── Register.tsx
│   │   └── Cart.tsx
│   ├── hooks/            # Custom hooks
│   ├── context/          # Context API (Auth, Cart)
│   ├── types/            # Tipos TypeScript
│   ├── utils/            # Utilidades (formatters, validators)
│   ├── assets/           # Imágenes, iconos
│   ├── styles/           # CSS global
│   ├── App.tsx
│   └── main.tsx
├── public/
├── index.html
├── vite.config.ts
├── tailwind.config.js
└── package.json
```

STACK TÉCNICO:
- React 18+ con TypeScript
- Vite (bundler rápido)
- React Router v6 (navegación)
- Tailwind CSS (estilos)
- Axios (peticiones HTTP)
- React Context API (estado global)
- React Hook Form (formularios)
- React Hot Toast (notificaciones)
- Lucide React (iconos)

PALETA DE COLORES:
```javascript
// tailwind.config.js
colors: {
  primary: {
    50: '#eff6ff',
    100: '#dbeafe',
    500: '#3b82f6', // Azul principal
    600: '#2563eb',
    700: '#1d4ed8',
  },
  neutral: {
    50: '#fafafa',
    100: '#f5f5f5',
    200: '#e5e5e5',
    800: '#262626',
    900: '#171717',
  },
  accent: '#10b981' // Verde menta
}
```

COMPONENTES REQUERIDOS (FASE 1):

1. LAYOUT:
  - Header:
    * Logo "OptiVision"
    * Navegación: Inicio | Monturas Sol | Monturas Oftálmicas | Lentes | Accesorios
    * Buscador
    * Iconos: Carrito (con badge de cantidad), Usuario/Login
    * Sticky al hacer scroll

  - Footer:
    * Información de contacto
    * Links rápidos
    * Redes sociales
    * Copyright

2. HOME PAGE:
  - Hero Section:
    * Imagen de fondo (gafas de sol)
    * Título: "La visión perfecta está a un clic de distancia"
    * CTA: "Ver catálogo"

  - Categorías Grid (4 cols desktop, 2 mobile):
    * Cards con imagen, título, "Ver más"

  - Productos Destacados:
    * Grid de 8 productos
    * ProductCard: imagen, nombre, marca, precio, botón "Ver"

  - Sección "Sobre Nosotros":
    * Breve descripción
    * Valores (Calidad, Servicio, Garantía)

3. PRODUCTS PAGE (Catálogo):
- Sidebar con filtros:
  * Categoría (checkboxes)
  * Marca (checkboxes)
  * Rango de precio (slider)
  * Características (color, material, forma, género)
  * Botón "Aplicar filtros"

- Grid de productos:
  * Toolbar: ordenamiento, vista (grid/lista)
  * ProductCard con hover effect
  * Paginación

- Responsive: Sidebar se convierte en drawer en mobile

4. PRODUCT DETAIL PAGE:
- Galería de imágenes (imagen principal + thumbnails)
- Información del producto:
  * Nombre, marca
  * Precio (descuento si aplica)
  * Descripción
  * Características (tabla)
  * Stock disponible
- Selector de cantidad
- Botón "Agregar al carrito" (grande, llamativo)
- Productos relacionados

5. AUTH PAGES:
- Login:
  * Email, password
  * Checkbox "Recordarme"
  * Link "¿Olvidaste tu contraseña?"

    * Botón "Iniciar sesión"
    * Link "¿No tienes cuenta? Regístrate"

  - Register:
    * Nombre, email, teléfono, password, confirmar password
    * Checkbox "Acepto términos y condiciones"
    * Botón "Crear cuenta"
    * Link "¿Ya tienes cuenta? Inicia sesión"

6. CART PAGE:
  - Lista de productos en el carrito:
    * Imagen, nombre, precio unitario
    * Selector de cantidad (+ / -)
    * Subtotal
    * Botón eliminar

  - Resumen:
    * Subtotal
    * Envío (calculado o "Se calcula en checkout")
    * Total
    * Botón "Proceder al pago"

  - Si está vacío: mensaje + botón "Explorar productos"

SERVICIOS API (api/):
```typescript
// api/axiosConfig.ts
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:4000/api',
  headers: {
    'Content-Type': 'application/json',
```

```
  },
});

// Interceptor para agregar token
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export default api;

// api/products.ts
export const getProducts = (params: FilterParams) => {
  return api.get('/products', { params });
};

export const getProductById = (id: string) => {
  return api.get(`/products/${id}`);
};

// api/auth.ts
export const login = (credentials: LoginCredentials) => {
  return api.post('/auth/login', credentials);
};

export const register = (userData: RegisterData) => {
  return api.post('/auth/register', userData);
};

// api/cart.ts
```

```
export const getCart = () => {
  return api.get('/cart');
};

export const addToCart = (productId: string, quantity: number) => {
  return api.post('/cart/items', { productId, quantity });
};
```

CONTEXT PROVIDERS:
```typescript
// context/AuthContext.tsx
interface AuthContextType {
  user: User | null;
  isAuthenticated: boolean;
  login: (email: string, password: string) => Promise<void>;
  register: (userData: RegisterData) => Promise<void>;
  logout: () => void;
  loading: boolean;
}

// context/CartContext.tsx
interface CartContextType {
  cart: CartItem[];
  totalItems: number;
  totalPrice: number;
  addItem: (product: Product, quantity: number) => void;
  removeItem: (productId: string) => void;
  updateQuantity: (productId: string, quantity: number) => void;
  clearCart: () => void;
}
```

CARACTERÍSTICAS UX:
- Loading spinners en todas las operaciones async
- Mensajes de error/éxito con React Hot Toast
- Validación en tiempo real en formularios
- Animaciones sutiles (hover, transitions)
- Skeleton loaders mientras cargan productos
- Scroll suave entre secciones
- Imágenes con lazy loading
- Botones deshabilitados durante operaciones

RESPONSIVE BREAKPOINTS:
- Mobile: < 640px
- Tablet: 640px - 1024px
- Desktop: > 1024px

DATOS PLACEHOLDER:
- Logo: Texto "OptiVision" con icono de gafas
- Hero background: https://images.unsplash.com/photo-1574158622682-e40e69881006 (gafas de sol)
- Productos sin imagen: placeholder con nombre del producto

ENTREGABLES:
1. Aplicación React completa y funcional
2. Todos los componentes implementados
3. Integración completa con backend
4. Responsive en mobile, tablet y desktop
5. Context providers para auth y cart
6. Manejo de errores y loading states
7. README.md con:
   - Instalación
   - Variables de entorno
   - Comandos disponibles
   - Estructura del proyecto

VARIABLES DE ENTORNO (.env):

VITE_API_URL=http://localhost:4000/api

🚀 Comandos para Ejecutar (Frontend)

bash

```
# 1. Navegar a carpeta frontend
cd ../frontend

# 2. Crear proyecto con Vite
npm create vite@latest . -- --template react-ts

# 3. Instalar dependencias base
npm install

# 4. Instalar dependencias adicionales
npm install react-router-dom axios react-hook-form lucide-react react-hot-toast

# 5. Instalar Tailwind CSS
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p

# 6. Configurar Tailwind (tailwind.config.js):
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#eff6ff',
          500: '#3b82f6',
          600: '#2563eb',
          700: '#1d4ed8',
        },
        accent: '#10b981',
```

```
      },
    },
  },
  plugins: [],
}

# 7. Configurar CSS principal (src/index.css):
@tailwind base;
@tailwind components;
@tailwind utilities;

# 8. Crear archivo .env
echo "VITE_API_URL=http://localhost:4000/api" > .env

# 9. Copiar código generado por IA

# 10. Ejecutar en modo desarrollo
npm run dev

# 11. Abrir navegador en http://localhost:5173
```

✅ CHECKLIST DE FASE 1 COMPLETA

Base de Datos ✓

- [ ] PostgreSQL instalado y corriendo
- [ ] Prisma configurado
- [ ] Schema definido con todas las tablas
- [ ] Migración inicial ejecutada
- [ ] Seed con datos de prueba

- [ ] Validado en Prisma Studio

Backend ✓

- [ ] Servidor Express corriendo en puerto 4000
- [ ] Todos los endpoints implementados
- [ ] Autenticación JWT funcionando
- [ ] Middlewares de validación y auth
- [ ] CORS configurado para frontend
- [ ] Probados todos los endpoints con cURL/Postman

Frontend ✓

- [ ] Aplicación React corriendo en puerto 5173
- [ ] Layout (Header + Footer) implementado
- [ ] Home page completa
- [ ] Página de productos con filtros
- [ ] Página de detalle de producto
- [ ] Páginas de login y registro
- [ ] Página de carrito
- [ ] Context de autenticación funcionando
- [ ] Context de carrito funcionando
- [ ] Responsive en mobile, tablet y desktop

Integración ✓

- [ ] Frontend conectado correctamente al backend
- [ ] Login y registro funcionando end-to-end
- [ ] Listar productos desde base de datos
- [ ] Ver detalle de producto
- [ ] Agregar productos al carrito

- [ ] Ver y modificar carrito

🧪 PRUEBAS DE INTEGRACIÓN

Flujo Completo a Validar:

```bash
# 1. Backend corriendo
cd backend
npm run dev
# Debe mostrar: Server running on http://localhost:4000

# 2. Frontend corriendo (nueva terminal)
cd frontend
npm run dev
# Debe mostrar: Local: http://localhost:5173

# 3. Abrir navegador en http://localhost:5173

# 4. Probar flujo:
# a) Ver página de inicio
# b) Navegar a "Monturas Sol"
# c) Hacer clic en un producto
# d) Ver detalle del producto
# e) Agregar al carrito
# f) Ver carrito (debe mostrar 1 item)
# g) Hacer clic en "Iniciar sesión"
# h) Registrar un nuevo usuario
# i) Login con ese usuario
# j) Carrito debe persistir
```

📁 ARCHIVOS IMPORTANTES

.env.example en cada carpeta

database/.env.example

```
DATABASE_URL="postgresql://usuario:password@localhost:5432/optica_db?schema=public"
```

backend/.env.example

```
PORT=4000
DATABASE_URL="postgresql://usuario:password@localhost:5432/optica_db?schema=public"
JWT_SECRET="tu_secreto_super_seguro_cambiar_en_produccion"
JWT_EXPIRES_IN="7d"
NODE_ENV="development"
FRONTEND_URL="http://localhost:5173"
```

frontend/.env.example

```
VITE_API_URL=http://localhost:4000/api
```

🐛 TROUBLESHOOTING COMÚN

Error: "Cannot connect to database"

```
bash
```

```
# Verificar que PostgreSQL esté corriendo
pg_isready
# O con Docker:
docker ps | grep postgres

# Verificar conexión
psql -U usuario -d optica_db
```

Error: "CORS policy" en frontend

typescript

```
// Verificar en backend/src/server.ts
app.use(cors({
  origin: 'http://localhost:5173',
  credentials: true
}));
```

Error: "Module not found" en backend

bash

```
# Regenerar Prisma Client
cd database
npx prisma generate

# Copiar a backend
cp -r node_modules/.prisma ../backend/node_modules/
cp -r node_modules/@prisma ../backend/node_modules/
```

📊 MÉTRICAS DE ÉXITO FASE 1

Al completar esta fase debes tener:

✅ **Base de datos:**

- 8 tablas relacionadas
- 20+ productos de prueba
- 3 usuarios (1 admin, 2 clientes)

✅ **Backend:**

- 15+ endpoints funcionando
- Autenticación completa
- Validaciones en todos los inputs

✅ **Frontend:**

- 7 páginas completamente funcionales
- 100% responsive
- Conectado a backend real

⏱️ TIEMPO ESTIMADO

- **Base de Datos:** 4-6 horas
- **Backend:** 12-16 horas
- **Frontend:** 20-24 horas
- **Integración y pruebas:** 4-6 horas

TOTAL: 40-52 horas (1-1.5 semanas full-time)

🎯 PRÓXIMOS PASOS

Una vez completada la Fase 1, estarás listo para:

- **Fase 2:** Sistema completo de órdenes y checkout
- **Fase 3:** Panel de administración
- **Fase 4:** Panel de cliente
- **Fase 5:** Pasarela de pagos
- **Fase