UNIVERSIDAD NACIONAL DEL ALTIPLANO

Facultad de Ingeniería Estadística e Informática

Escuela Profesional de Ingeniería Estadística e Informática

MANUAL TÉCNICO

Sistema E-commerce AKEMY

Librería y Papelería Online

Curso: Ingeniería de Software II
Docente: Laura Murillo Ramiro Pedro

Estudiante: Junior Beltran Huaraya Chipana
GitHub: https://github.com/BeltranHC

Proyecto: AKEMY - Sistema E-commerce
Tipo: Aplicación Web Full Stack
URL Producción: https://akemy.app

Puno - Perú

Diciembre 2025

Índice

1 Introducción Técnica

1.1 Propósito del Manual Técnico

El presente manual técnico tiene como objetivo documentar de manera exhaustiva la arquitectura, diseño e implementación del sistema de comercio electrónico AKEMY. Este documento está destinado a proporcionar toda la información técnica necesaria para comprender, mantener, extender y desplegar el sistema.

El manual cubre los siguientes aspectos:

- Arquitectura general del sistema y sus componentes
- Tecnologías y frameworks utilizados en el desarrollo
- Diseño de la base de datos y API REST
- Medidas de seguridad implementadas
- Procedimientos de configuración y despliegue
- Integración con servicios externos (Mercado Pago, Resend)
- Consideraciones de escalabilidad y rendimiento

1.2 Alcance del Documento

Este manual abarca la totalidad del sistema AKEMY, incluyendo:

- **Backend API:** Desarrollado con NestJS 10.3 y PostgreSQL
- **Frontend Web:** Desarrollado con Next.js 15 y React 18
- **Base de Datos:** PostgreSQL 16 alojada en Neon (Serverless)
- **Servicios Externos:** Mercado Pago (pagos), Resend (emails)
- **Infraestructura:** Vercel (frontend), Render (backend)

1.3 Público Objetivo

Este documento está dirigido a:

- Desarrolladores que trabajarán en el mantenimiento o extensión del sistema
- Arquitectos de software que necesiten comprender la estructura del sistema
- Administradores de sistemas encargados del despliegue y operación
- Auditores técnicos que evalúen la seguridad y calidad del código

2 Descripción General del Sistema

2.1 Visión Técnica del Sistema

AKEMY es una plataforma de comercio electrónico diseñada específicamente para el sector de papelería y librería en el mercado peruano. El sistema implementa una arquitectura moderna de tres capas con separación clara entre la presentación (Frontend), lógica de negocio (Backend API) y persistencia de datos (PostgreSQL).

La plataforma está diseñada para ser:

- **Escalable:** Capaz de manejar crecimiento en usuarios y transacciones
- **Segura:** Implementando múltiples capas de protección
- **Responsiva:** Funcionando óptimamente en dispositivos móviles y desktop
- **Mantenible:** Con código modular y bien documentado

2.2 Problema Abordado desde el Enfoque Tecnológico

El sistema aborda los siguientes desafíos técnicos:

1. **Gestión de Inventario en Tiempo Real:** Control preciso de stock con actualizaciones inmediatas al procesar pedidos.
2. **Procesamiento de Pagos Seguro:** Integración con Mercado Pago para pagos con tarjeta, cumpliendo estándares PCI DSS.
3. **Comunicación en Tiempo Real:** Sistema de chat cliente-soporte mediante WebSockets (Socket.io).
4. **Autenticación Robusta:** Sistema de doble token (Access + Refresh) con JWT.
5. **Prevención de Ataques:** Sanitización XSS, rate limiting, headers de seguridad HTTP.

2.3 Funcionalidades Técnicas Principales

Módulo	Funcionalidad
Auth	Registro, login, JWT, refresh tokens, verificación email
Products	CRUD productos, variantes, imágenes, stock
Cart	Carrito persistente, cálculo con ofertas
Orders	Creación pedidos, flujo de estados, historial
Payments	Integración Mercado Pago, webhooks
Chat	WebSocket bidireccional, tiempo real
Offers	Promociones con fechas, descuentos automáticos
Coupons	Validación, límites de uso, descuentos
Reviews	Reseñas con moderación, calificaciones

Cuadro 1: Módulos técnicos principales

2.4 Flujo General de Operación del Sistema

El flujo típico de una transacción en el sistema es:

1. El cliente navega el catálogo (Frontend consulta API de productos)
2. Agrega productos al carrito (API calcula totales con ofertas activas)
3. Procede al checkout (validación de datos, stock, cupones)
4. Selecciona método de pago:
   - **Mercado Pago:** Redirección a checkout externo, webhook de confirmación
   - **Contra entrega:** Pedido creado con estado PENDING
5. Sistema actualiza stock, envía emails, asigna puntos de fidelidad
6. Administrador gestiona el pedido hasta su entrega

3 Arquitectura del Sistema

3.1 Arquitectura General del Sistema

El sistema utiliza una arquitectura de microservicios monolítica modular, donde el backend está organizado en módulos independientes pero desplegados como una única unidad.

```
+-----------------------------------------------------------+
|                      CLIENTE (Browser)                    |
|  +-------------------------------------------------+  |
|  |              Frontend (Next.js 15)                  |  |
|  |   React 18 | TailwindCSS | Zustand | TanStack Query |  |
|  +-------------------------------------------------+  |
+-----------------------------------------------------------+
                              |
```

```
             HTTP/HTTPS (REST API) + WebSocket (Socket.io)
                                v
+-----------------------------------------------------------+
|                    BACKEND (NestJS 10.3)                  |
|  +-----------------------------------------------------+  |
|  |  Modules: Auth, Products, Cart, Orders, Payments,   |  |
|  |  Chat, Offers, Coupons, Reviews, Returns, Loyalty   |  |
|  +-----------------------------------------------------+  |
|  | JWT | Rate Limiting | CORS | Helmet | Sanitize XSS  |  |
|  +-----------------------------------------------------+  |
|                            |                              |
|                       Prisma ORM                          |
|                            v                              |
|  +-----------------------------------------------------+  |
|  |           PostgreSQL 16 (Neon Serverless)           |  |
|  +-----------------------------------------------------+  |
+-----------------------------------------------------------+
                            |
         +------------------+------------------+
         v                                     v
+------------------+                  +------------------+
|   Mercado Pago   |                  |      Resend      |
|   (Pagos)        |                  |   (Emails)       |
+------------------+                  +------------------+
```

Listing 1: Diagrama de Arquitectura General

3.1.1 Componentes de la Arquitectura

- **Frontend (Vercel):** Aplicación Next.js 15 con App Router, renderizado del lado del servidor (SSR) y generación estática (SSG).
- **Backend API (Render):** API REST con NestJS 10.3, autenticación JWT, WebSockets para chat en tiempo real.
- **Base de Datos (Neon):** PostgreSQL 16 serverless con connection pooling mediante PgBouncer.
- **Servicios Externos:**
  - Mercado Pago: Procesamiento de pagos con tarjeta
  - Resend: Envío de emails transaccionales

3.1.2 Flujo de Comunicación entre Componentes

Origen	Destino	Protocolo	Propósito
Frontend	Backend	HTTPS (REST)	Operaciones CRUD
Frontend	Backend	WSS (Socket.io)	Chat tiempo real
Backend	Neon DB	PostgreSQL	Persistencia
Backend	Mercado Pago	HTTPS	Crear preferencias
Mercado Pago	Backend	HTTPS (Webhook)	Confirmar pagos
Backend	Resend	HTTPS	Enviar emails

Cuadro 2: Flujo de comunicación entre componentes

3.1.3 Principios Arquitectónicos Aplicados

- **Separación de Responsabilidades (SoC):** Cada módulo tiene una responsabilidad única y bien definida.
- **Inyección de Dependencias:** NestJS proporciona un contenedor IoC nativo para gestionar dependencias.
- **Repository Pattern:** Prisma ORM abstrae el acceso a datos con un cliente tipado.
- **DTO Pattern:** Objetos de transferencia de datos para validación y serialización.
- **Guard Pattern:** Protección de rutas mediante guards de autenticación y roles.

3.2 Arquitectura del Backend

3.2.1 Organización Interna del Backend

```
backend/src/
|-- main.ts                  # Punto de entrada
|-- app.module.ts            # Modulo raiz
|-- prisma/                  # Servicio Prisma
|-- common/                  # Pipes, Guards, Decoradores
|   |-- pipes/
|   |   +-- sanitize-input.pipe.ts
|   +-- decorators/
|-- auth/                    # Autenticacion JWT
|-- users/                   # Gestion usuarios
|-- products/                # Catalogo
|-- categories/              # Categorias jerarquicas
|-- brands/                  # Marcas
|-- cart/                    # Carrito de compras
|-- orders/                  # Pedidos
|-- payments/                # Mercado Pago
|-- chat/                    # WebSocket Gateway
|-- offers/                  # Promociones
|-- coupons/                 # Cupones
|-- reviews/                 # Resenas
|-- returns/                 # Devoluciones
```

```
|-- loyalty/                # Puntos fidelidad
|-- wishlist/               # Lista deseos
|-- comparison/             # Comparador
|-- banners/                # Banners
|-- settings/               # Configuracion
|-- dashboard/              # Estadisticas
|-- upload/                 # Archivos
+-- mail/                   # Emails (Resend)
```

Listing 2: Estructura del Backend

3.2.2 Capa de Rutas y Controladores

Los controladores definen los endpoints REST y están decorados con metadatos de Swagger para documentación automática:

```
@Controller('products')
@ApiTags('Products')
export class ProductsController {
  @Get()
  @Public()
  @ApiOperation({ summary: 'Listar productos' })
  findAll(@Query() filters: FilterProductsDto) {
    return this.productsService.findAll(filters);
  }

  @Post()
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles('ADMIN', 'SUPERADMIN')
  create(@Body() dto: CreateProductDto) {
    return this.productsService.create(dto);
  }
}
```

Listing 3: Ejemplo de Controlador

3.2.3 Capa de Servicios y Lógica de Negocio

La lógica de negocio está encapsulada en servicios inyectables:

```
@Injectable()
export class CartService {
  constructor(
    private prisma: PrismaService,
    private offersService: OffersService,
  ) {}

  async addItem(dto: AddToCartDto, userId?: string) {
    // Validar stock
    // Verificar ofertas activas
    // Calcular totales
    // Persistir cambios
  }
}
```

Listing 4: Ejemplo de Servicio

3.2.4 Persistencia de Datos y Base de Datos

El sistema utiliza Prisma ORM para la persistencia de datos:

- **Schema declarativo:** Modelo de datos definido en `schema.prisma`
- **Migraciones:** Control de versiones del esquema de base de datos
- **Cliente tipado:** Consultas con autocompletado y type-safety
- **Connection Pooling:** PgBouncer integrado en Neon

3.2.5 Seguridad, Autenticación y Control de Acceso

- **JWT:** Tokens firmados con secreto configurable
- **Refresh Tokens:** Renovación automática de sesiones
- **Guards:** JwtAuthGuard, RolesGuard para protección de rutas
- **Rate Limiting:** Throttler con límites por endpoint
- **Sanitización:** Pipe global que limpia inputs de XSS

3.2.6 Comunicación en Tiempo Real

El módulo de chat utiliza WebSockets mediante Socket.io:

```
@WebSocketGateway({
  cors: { origin: FRONTEND_URL, credentials: true },
  namespace: '/chat',
  transports: ['polling', 'websocket'],
})
export class ChatGateway {
  @SubscribeMessage('sendMessage')
  handleMessage(client: Socket, payload: SendMessageDto) {
    // Persistir mensaje
    // Emitir a sala
    // Actualizar contador no leidos
  }
}
```

Listing 5: WebSocket Gateway

3.3 Arquitectura del Frontend

3.3.1 Estructura del Proyecto Frontend

```
frontend/src/
|-- app/                    # App Router (Next.js 15)
|   |-- (shop)/             # Paginas de tienda
|   |-- admin/              # Panel administracion
|   |-- checkout/           # Checkout y pagos
|   |   |-- success/
|   |   |-- failure/
|   |   +-- pending/
|   +-- cuenta/             # Area cliente
|-- components/
|   |-- layout/             # Header, Footer, etc.
|   |-- ui/                 # shadcn/ui components
|   |-- products/           # ProductCard, etc.
|   |-- cart/               # CartDrawer
|   +-- chat/               # ChatWidget
|-- lib/
|   |-- api.ts              # Cliente API (Axios)
|   |-- store.ts            # Estado global (Zustand)
|   |-- socket.tsx          # Provider WebSocket
|   +-- utils.ts            # Funciones helper
```

Listing 6: Estructura del Frontend

3.3.2 Capa de Presentación y Componentes

- **React 18:** Componentes funcionales con hooks
- **TailwindCSS 3.4:** Estilos utilitarios con diseño responsive
- **shadcn/ui:** Componentes accesibles basados en Radix UI
- **Lucide React:** Iconografía consistente

3.3.3 Gestión del Estado y Acceso a Datos

- **Zustand:** Estado global para carrito, autenticación y wishlist
- **TanStack Query:** Data fetching, caching y sincronización
- **React Hook Form:** Manejo de formularios con validación Zod

3.3.4 Comunicación en Tiempo Real

El frontend se conecta al WebSocket del backend mediante un provider Context:

```
export function SocketProvider({ children }) {
  const socket = useMemo(() =>
    io(SOCKET_URL, {
      withCredentials: true,
      transports: ['polling', 'websocket'],
    }), []
  );

```

```
  return (
    <SocketContext.Provider value={socket}>
      {children}
    </SocketContext.Provider>
  );
}
```

Listing 7: Provider Socket.io

3.4 Comunicación e Integración entre Servicios

3.4.1 Comunicación Frontend–Backend

Todas las llamadas HTTP utilizan Axios con interceptores configurados:

```
const api = axios.create({
  baseURL: API_URL,
  withCredentials: true,
});

api.interceptors.request.use((config) => {
  const token = getAccessToken();
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Listing 8: Cliente API

3.4.2 Comunicación Backend–Mercado Pago

La integración con Mercado Pago sigue el flujo de preferencias:

1. Backend crea preferencia con items del pedido
2. Frontend redirige a checkout de Mercado Pago
3. Usuario completa pago en plataforma segura
4. Mercado Pago envía webhook con resultado
5. Backend actualiza estado del pedido

3.4.3 Consideraciones de Seguridad en la Integración

- **HTTPS obligatorio:** Todas las comunicaciones son cifradas
- **Webhooks verificados:** Validación de origen de Mercado Pago
- **Tokens seguros:** Access tokens con expiración corta (15min)
- **CORS estricto:** Solo orígenes permitidos pueden acceder

4 Tecnologías Utilizadas

4.1 Tecnologías del Backend

Tecnología	Versión	Propósito
NestJS	10.3	Framework principal (Node.js)
TypeScript	5.3	Lenguaje de programación
Prisma	5.8	ORM y migraciones
PostgreSQL	16	Base de datos relacional
Socket.io	4.8	WebSockets tiempo real
Mercado Pago SDK	2.11	Pasarela de pagos
Resend	6.6	Emails transaccionales
JWT (@nestjs/jwt)	10.2	Autenticación
Bcrypt	2.4	Hashing contraseñas
Helmet	7.1	Headers seguridad HTTP
Sanitize-html	2.17	Prevención XSS
Swagger	7.2	Documentación API
Class-validator	0.14	Validación DTOs

Cuadro 3: Stack tecnológico del Backend

4.2 Tecnologías del Frontend

Tecnología	Versión	Propósito
Next.js	15	Framework React (App Router)
React	18	Biblioteca UI
TypeScript	5.3	Lenguaje de programación
TailwindCSS	3.4	Framework CSS utilitario
shadcn/ui	Latest	Componentes accesibles
Radix UI	Latest	Primitivos UI
Zustand	5	Gestión estado global
TanStack Query	5	Data fetching/caching
React Hook Form	7.53	Manejo formularios
Zod	3.22	Validación esquemas
Socket.io-client	4.8	Cliente WebSocket
Lucide React	Latest	Iconografía

Cuadro 4: Stack tecnológico del Frontend

4.3 Tecnologías de Infraestructura

Servicio	Plataforma	Propósito
Frontend Hosting	Vercel	Deploy Next.js con SSR
Backend Hosting	Render	Deploy NestJS
Base de Datos	Neon	PostgreSQL Serverless
Pagos	Mercado Pago	Procesamiento tarjetas
Emails	Resend	Transaccionales
Control de Versiones	GitHub	Repositorio código

Cuadro 5: Infraestructura de producción

5 Diseño del Sistema

5.1 Diseño de Componentes del Sistema

El sistema está compuesto por los siguientes componentes principales:

1. **Módulo de Autenticación:** Gestión de identidad, tokens JWT, verificación email.
2. **Módulo de Catálogo:** Productos, categorías, marcas, variantes, imágenes.
3. **Módulo de Comercio:** Carrito, pedidos, ofertas, cupones.
4. **Módulo de Pagos:** Integración Mercado Pago, webhooks.
5. **Módulo de Comunicación:** Chat en tiempo real, notificaciones.
6. **Módulo de Fidelización:** Puntos, wishlist, reseñas.

5.2 Diseño de la Base de Datos

Las entidades principales del sistema son:

Entidad	Descripción
User	Usuarios (clientes y administradores)
Product	Catálogo de productos
ProductImage	Imágenes de productos
ProductVariant	Variantes (color, tamaño)
Category	Categorías jerárquicas
Brand	Marcas de productos
Cart / CartItem	Carrito de compras
Order / OrderItem	Pedidos y líneas de pedido
Coupon	Cupones de descuento
Offer / OfferProduct	Promociones
Review	Reseñas de productos
Conversation / Message	Chat

Cuadro 6: Entidades principales de la base de datos

5.3 Diseño de la API REST

La API sigue principios RESTful con las siguientes convenciones:

- **Prefijo global:** `/api`
- **Versionado:** Implícito en URL base
- **Formato:** JSON para request/response
- **Autenticación:** Bearer token en header Authorization
- **Códigos HTTP:** 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found

Documentación Swagger: Disponible en https://akemy-backend.onrender.com/api/docs

5.4 Diseño de la Comunicación en Tiempo Real

Evento	Dirección	Payload
startConversation	Cliente → Servidor	{ subject?: string }
sendMessage	Cliente → Servidor	{ conversationId, content }
joinConversation	Cliente → Servidor	{ conversationId }
markAsRead	Cliente → Servidor	{ conversationId }
newMessage	Servidor → Cliente	Message object
conversation	Servidor → Cliente	Conversation object
unreadCount	Servidor → Cliente	{ count: number }

Cuadro 7: Eventos WebSocket del sistema de chat

6 Seguridad del Sistema

6.1 Autenticación y Autorización

El sistema implementa un esquema de doble token JWT:

- **Access Token (15 minutos):** Token de corta duración para acceso a recursos.
- **Refresh Token (7 días):** Token de larga duración almacenado hasheado en BD.

```
{
  "sub": "user-uuid",
  "email": "usuario@email.com",
  "role": "CUSTOMER",
  "firstName": "Nombre",
  "iat": 1702656000,
  "exp": 1702656900
}
```

Listing 9: Estructura del JWT

6.2 Gestión de Roles y Permisos

Rol	**Permisos**
CUSTOMER	Compras, perfil, historial, chat
ADMIN	Lo anterior + gestión productos, pedidos
SUPERADMIN	Acceso total, configuración sistema
EDITOR	Gestión de contenido (banners, categorías)
PRODUCT_MANAGER	Gestión de productos e inventario

Cuadro 8: Roles y permisos del sistema

6.3 Protección de Datos y Privacidad

Medida	**Implementación**	**Descripción**
Hashing contraseñas	bcryptjs	Salt rounds: 10
Sanitización XSS	sanitize-html	Pipe global
Rate Limiting	@nestjs/throttler	Límites por endpoint
Headers HTTP	Helmet	CSP, HSTS, X-XSS-Protection
Validación inputs	class-validator	DTOs estrictos
CORS	@nestjs/cors	Orígenes permitidos

Cuadro 9: Medidas de protección de datos

6.4 Rate Limiting por Endpoint

Endpoint	Límite	Período
/auth/login	5 intentos	60 segundos
/auth/admin/login	3 intentos	60 segundos
General	100 requests	60 segundos

Cuadro 10: Configuración de Rate Limiting

6.5 Seguridad en la Comunicación

- **HTTPS obligatorio:** TLS en todas las comunicaciones
- **HSTS:** HTTP Strict Transport Security habilitado
- **CSP:** Content Security Policy para prevenir inyecciones
- **Cookies seguras:** httpOnly, secure, sameSite

7 Configuración y Despliegue

7.1 Requisitos Técnicos del Sistema

Requisito	Especificación
Node.js	20.x o superior
PostgreSQL	16.x (o Neon Serverless)
npm	10.x o superior
Memoria RAM	Mínimo 512MB por servicio

Cuadro 11: Requisitos técnicos

7.2 Variables de Entorno del Backend

```
# Servidor
NODE_ENV=production
PORT=3001

# Base de Datos (Neon con pooling)
DATABASE_URL="postgresql://...@...-pooler...?sslmode=require&pgbouncer
    =true"

# JWT
JWT_SECRET=clave-secreta-muy-larga-y-segura
JWT_REFRESH_SECRET=otra-clave-secreta-diferente
JWT_EXPIRATION=15m
JWT_REFRESH_EXPIRATION=7d
```

```
13
14 # CORS
15 FRONTEND_URL=https://akemy.app
16
17 # Mercado Pago
18 MERCADO_PAGO_ACCESS_TOKEN=APP_USR-xxx
19 MERCADO_PAGO_PUBLIC_KEY=APP_USR-xxx
20 BACKEND_URL=https://akemy-backend.onrender.com
21
22 # Resend (Emails)
23 RESEND_API_KEY=re_xxx
24 EMAIL_FROM="AKEMY <noreply@akemy.app>"
25
26 # Rate Limiting
27 THROTTLE_TTL=60
28 THROTTLE_LIMIT=100
```

Listing 10: Variables de entorno (.env)

7.3 Despliegue Mediante Docker Compose

```
1 # Construir e iniciar servicios
2 docker-compose up -d
3
4 # Ver logs en tiempo real
5 docker-compose logs -f
6
7 # Ejecutar migraciones
8 docker-compose exec backend npx prisma migrate deploy
9
10 # Ejecutar seed de datos iniciales
11 docker-compose exec backend npx prisma db seed
12
13 # Detener servicios
14 docker-compose down
```

Listing 11: Comandos Docker

7.4 Despliegue en Producción

7.4.1 Frontend (Vercel)

1. Conectar repositorio GitHub a Vercel
2. Configurar variable `NEXT_PUBLIC_API_URL`
3. Deploy automático en cada push a main

7.4.2 Backend (Render)

1. Crear Web Service conectado a GitHub

2. Configurar Build Command: `npm install && npm run build`
3. Configurar Start Command: `npm run start:prod`
4. Agregar todas las variables de entorno
5. Deploy automático en cada push

8 Integración de Pagos con Mercado Pago

8.1 Rol del Servicio de Pagos

El módulo de pagos gestiona la integración con Mercado Pago para procesar pagos con tarjeta de crédito y débito. El flujo está diseñado para que los datos sensibles de tarjetas nunca pasen por nuestros servidores.

8.2 Flujo de Pago

1. **Crear Preferencia:** Backend crea preferencia con items del pedido
2. **Redirección:** Frontend redirige al checkout de Mercado Pago
3. **Pago:** Usuario ingresa datos en plataforma segura de MP
4. **Webhook:** MP notifica resultado al backend
5. **Actualización:** Backend actualiza estado del pedido
6. **Redirección:** Usuario vuelve a página de éxito/fallo/pendiente

8.3 Endpoints del Módulo de Pagos

Método	Endpoint	Descripción
POST	/payments/create-preference	Crear preferencia de pago
POST	/payments/webhook	Recibir notificación de MP
GET	/payments/status/:id	Consultar estado del pago
GET	/payments/config	Obtener public key

Cuadro 12: Endpoints del módulo de pagos

8.4 Tarjetas de Prueba

Tipo	Número	CVV	Estado
Visa	4509 9535 6623 3704	123	Aprobada
Mastercard	5031 7557 3453 0604	123	Aprobada

Cuadro 13: Tarjetas de prueba de Mercado Pago

9 Consideraciones de Escalabilidad y Rendimiento

9.1 Connection Pooling (Neon)

La base de datos utiliza PgBouncer para connection pooling, evitando el overhead de crear conexiones nuevas en cada request. Configuración mediante parámetro `&pgbouncer=true` en DATABASE_URL.

9.2 Manejo de Concurrencia

- **WebSockets:** Socket.io maneja múltiples conexiones concurrentes
- **Rate Limiting:** Previene sobrecarga por abuso
- **Transacciones Prisma:** Operaciones atómicas en BD

9.3 Caché y Optimización

- **TanStack Query:** Cache de datos en frontend (staleTime configurable)
- **Zustand:** Estado persistente del carrito
- **Next.js:** SSG para páginas estáticas, ISR para contenido dinámico

9.4 Posibles Mejoras Futuras

- Implementar Redis para caché del backend
- CDN para assets estáticos
- Upgrade a Render Starter para eliminar cold starts
- Implementar queues para tareas pesadas (emails, reportes)

10 Mantenimiento y Extensibilidad del Sistema

10.1 Organización del Código Fuente

El código sigue las convenciones de NestJS y Next.js:

- Módulos independientes con responsabilidad única
- DTOs para validación de entrada
- Servicios para lógica de negocio
- Controllers para definición de endpoints
- Componentes React reutilizables

10.2 Buenas Prácticas de Mantenimiento

- Mantener dependencias actualizadas (npm audit)
- Ejecutar migraciones en staging antes de producción
- Revisar logs de Render/Vercel periódicamente
- Monitorear métricas de Neon (conexiones, storage)
- Realizar backups periódicos de la base de datos

10.3 Extensiones Futuras Sugeridas

1. Sistema de notificaciones push (FCM)
2. Integración con más pasarelas de pago (Stripe, PayPal)
3. App móvil nativa (React Native)
4. Sistema de reportes y analytics
5. Múltiples idiomas (i18n)

11 Conclusiones Técnicas

El sistema AKEMY representa una implementación moderna de comercio electrónico con las siguientes características destacables:

- **Arquitectura Modular:** Facilita el mantenimiento y extensión del sistema.
- **Stack Moderno:** NestJS + Next.js proporcionan una base sólida y escalable.
- **Seguridad Multicapa:** Múltiples medidas de protección implementadas.

- **Integración de Pagos:** Mercado Pago proporciona procesamiento seguro de tarjetas.
- **Comunicación en Tiempo Real:** Chat integrado para soporte al cliente.
- **Despliegue en la Nube:** Infraestructura serverless que escala automáticamente.

12 Licencia del Software

Este proyecto está licenciado bajo la **Licencia MIT**, lo que permite:

- Uso comercial y privado
- Modificación del código fuente
- Distribución del software
- Sublicenciamiento

La única condición es mantener el aviso de copyright original y la licencia en todas las copias.

13 Créditos del Proyecto

Rol	Responsable
Desarrollo Full Stack	Junior Beltran Huaraya Chipana
Diseño UI/UX	Junior Beltran Huaraya Chipana
Arquitectura del Sistema	Junior Beltran Huaraya Chipana

Cuadro 14: Equipo de desarrollo

Repositorio: https://github.com/BeltranHC/ecomerce_akemy

Contacto: huaraya0804@email.com

AKEMY - Librería y Papelería Online
Tu papelería favorita

https://akemy.app

Universidad Nacional del Altiplano - Puno, Perú