

Documentación Técnica - StudiMarket ## Web Mini Market Universitario **Versión:** 1.0.0 **Fecha:** Noviembre 2025 **Equipo:** cristian leon | fernanda calderon | maximiliano parra | lorenzo figueroa --- ## Modelo de Datos ### Entidades Principales ##### 1. Usuario (Usuario) Gestiona la autenticación y perfiles de usuarios del sistema.

sql CREATE TABLE usuarios (id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(100) NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, password_hash VARCHAR(255) NOT NULL, rol ENUM('admin', 'cliente') DEFAULT 'cliente', fecha_registro DATETIME DEFAULT CURRENT_TIMESTAMP, activo BOOLEAN DEFAULT TRUE); **Campos:** - id: Identificador único del usuario - nombre: Nombre completo del usuario - email: Correo electrónico (único, usado para login) - password_hash: Contraseña encriptada con bcrypt - rol: Tipo de usuario (admin o cliente) - fecha_registro: Timestamp de creación - activo: Estado de la cuenta ##### 2. Producto (Producto) Almacena el catálogo de productos disponibles. sql CREATE TABLE productos (id INT PRIMARY KEY AUTO_INCREMENT, titulo VARCHAR(200) NOT NULL, descripcion TEXT, precio DECIMAL(10,2) NOT NULL, stock INT NOT NULL DEFAULT 0, categoria VARCHAR(50), imagen_url TEXT, fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP, activo BOOLEAN DEFAULT TRUE); **Campos:** - id: Identificador único del producto - titulo: Nombre del producto - descripcion: Descripción detallada - precio: Precio en pesos chilenos (CLP) - stock: Cantidad disponible - categoria: Categoría del producto (Tecnología, Papelería, Alimentos, etc.) - imagen_url: URL de la imagen del producto - fecha_creacion: Timestamp de creación - activo: Estado del producto (disponible/no disponible) ##### 3. Carrito (Carrito) Representa el carrito de compras de cada usuario. sql CREATE TABLE carritos (id INT PRIMARY KEY AUTO_INCREMENT, usuario_id INT NOT NULL, fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP, activo BOOLEAN DEFAULT TRUE, FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE); **Campos:** - id: Identificador único del carrito - usuario_id: Referencia al usuario propietario - fecha_creacion: Timestamp de creación - activo: Estado del carrito (activo/completado) ##### 4. Item Carrito (ItemCarrito) Relación muchos a muchos entre Carrito y Producto. sql CREATE TABLE items_carrito (id INT PRIMARY KEY AUTO_INCREMENT, carrito_id INT NOT NULL, producto_id INT NOT NULL, cantidad INT NOT NULL DEFAULT 1, precio_unitario DECIMAL(10,2) NOT NULL, FOREIGN KEY (carrito_id) REFERENCES carritos(id) ON DELETE CASCADE, FOREIGN KEY (producto_id) REFERENCES productos(id) ON DELETE CASCADE); **Campos:** - id: Identificador único del item - carrito_id: Referencia al carrito - producto_id: Referencia al producto - cantidad: Cantidad de unidades - precio_unitario: Precio al momento de agregar al carrito ##### Relaciones del Modelo Usuario (1) —— (N) Carrito Carrito (1) —— (N) ItemCarrito Producto (1) —— (N) ItemCarrito --- ##  API Endpoints ### Base URL <http://localhost:8000/api>

Autenticación ##### POST /auth/register Registra un nuevo usuario en el sistema. **Request Body:**

```
json { "nombre": "Juan Pérez", "email": "juan@ejemplo.com", "password": "contraseña123" } Response (201):
json { "message": "Usuario creado exitosamente", "user": { "id": 1, "nombre": "Juan Pérez", "email": "juan@ejemplo.com", "rol": "cliente" } }
```

POST /auth/login Inicia sesión y obtiene un token JWT. Request Body:

```
json { "email": "juan@ejemplo.com", "password": "contraseña123" } Response (200): json { "access_token": "eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9...", "token_type": "bearer", "user": { "id": 1, "nombre": "Juan Pérez", "email": "juan@ejemplo.com", "rol": "cliente" } }
```

GET /auth/me Obtiene información del usuario autenticado. Headers:

Authorization: Bearer **Response (200):** json { "id": 1, "nombre": "Juan Pérez", "email": "juan@ejemplo.com", "rol": "cliente", "fecha_registro": "2025-11-13T10:30:00" }

Productos ##### GET /products Lista todos los productos disponibles. **Query Parameters:** - categoria (opcional): Filtrar por categoría - buscar (opcional): Buscar por nombre o descripción - activo (opcional): Filtrar por estado (true/false) **Response (200):**

```
json { "products": [ { "id": 1, "titulo": "Laptop Dell XPS 15", "descripcion": "Laptop de alto rendimiento", "precio": 1299990.00, "stock": 5, "categoria": "Tecnología", "imagen_url": "https://ejemplo.com/imagen.jpg", "activo": true } ], "total": 1 }
```

GET /products/{id} Obtiene los detalles de un producto específico. Response (200):

```
json { "id": 1, "titulo": "Laptop Dell XPS 15", "descripcion": "Laptop de alto rendimiento con procesador Intel i7", "precio": 1299990.00, "stock": 5, "categoria": "Tecnología", "imagen_url": "https://ejemplo.com/imagen.jpg", "activo": true, "fecha_creacion": "2025-11-01T08:00:00" }
```

POST /products  Crea un nuevo producto (requiere rol admin). Headers:

Authorization: Bearer <admin_token> **Request Body:** json { "titulo": "Mouse Logitech G502", "descripcion": "Mouse gaming de alta precisión", "precio": 49990.00, "stock": 20, "categoria": "Tecnología", "imagen_url": "https://ejemplo.com/mouse.jpg" } **Response (201):** json { "id": 2, "titulo": "Mouse Logitech G502", "precio": 49990.00, "stock": 20, "activo": true }

PUT /products/{id}  Actualiza un producto existente (requiere rol admin). Headers:

Authorization: Bearer <admin_token> **Request Body:** json { "precio": 44990.00, "stock": 25 } **Response (200):** json { "message": "Producto actualizado exitosamente", "product": { "id": 2, "titulo": "Mouse Logitech G502", "precio": 44990.00, "stock": 25 } }

DELETE /products/{id}  Elimina (desactiva) un producto (requiere rol admin). Headers:

Authorization: Bearer <admin_token> **Response (200):** json { "message": "Producto eliminado exitosamente" }

Categorías ##### GET /productos/categorias Lista todas las categorías disponibles. **Response (200):**

```
json { "categorias": [ "Tecnología", "Papelería", "Alimentos", "Accesorios" ] }
```

Health Check ##### GET / Verifica el estado del servidor. **Response (200):**

json { "status": "ok", "message": "StudiMarket API funcionando correctamente", "version": "1.0.0", "endpoints": { "docs": "/docs", "health": "/health" } } --- ## 🔑 Decisiones Técnicas ### Backend ##### 1. **Framework:**

FastAPI Justificación: - Alto rendimiento basado en ASGI (Starlette) - Documentación automática con Swagger/OpenAPI - Validación de datos automática con Pydantic - Soporte nativo para async/await - Type hints que mejoran la calidad del código ##### 2. **Base de Datos: MySQL 8.0 Justificación:** - Amplia adopción en la industria - Excelente rendimiento para operaciones CRUD - Soporte robusto de transacciones ACID - Herramientas de administración maduras - Facilidad de escalabilidad vertical y horizontal ##### 3. **ORM:**

SQLAlchemy Justificación: - ORM maduro y confiable para Python - Soporta relaciones complejas entre modelos - Permite queries optimizadas y raw SQL cuando es necesario - Migración de esquemas con Alembic

- Independencia del motor de base de datos ##### 4. **Autenticación: JWT (JSON Web Tokens) Justificación:** - Stateless - no requiere almacenamiento en servidor - Escalable para arquitecturas distribuidas - Información del usuario en el token (claims) - Seguridad mediante firma HMAC-SHA256 - Expiración automática de tokens ##### 5. **Seguridad de Contraseñas: Bcrypt Justificación:** - Algoritmo de hashing diseñado para contraseñas

- Resistente a ataques de fuerza bruta - Salt automático por cada contraseña - Factor de costo ajustable (work factor) - Estándar de la industria ##### 6. **Migraciones: Alembic Justificación:** - Control de versiones del esquema de BD - Migraciones automáticas y reversibles - Integración nativa con SQLAlchemy - Facilita el trabajo en equipo - Despliegues seguros a producción #### Frontend ##### 1. **Framework: React 18 Justificación:**

- Ecosistema maduro y extenso - Componentes reutilizables - Virtual DOM para rendimiento óptimo - Hooks para manejo de estado - Gran comunidad y recursos ##### 2. **Build Tool: Vite 5 Justificación:**

- Desarrollo ultra rápido con HMR (Hot Module Replacement) - Build optimizado para producción - Soporte nativo de ESM (ES Modules) - Menor tiempo de inicio comparado con Webpack - Configuración mínima out-of-the-box ##### 3. **Manejo de Estado: Context API Justificación:** - Nativo de React, sin dependencias adicionales - Suficiente para la complejidad actual del proyecto - Evita prop drilling - Fácil de entender y mantener - Permite escalabilidad a Redux si es necesario ##### 4. **Estilos: CSS Puro Justificación:**

- Control total sobre los estilos - Sin overhead de librerías CSS-in-JS - Mejor rendimiento en runtime - Facilita la personalización - Mantiene el bundle size reducido ##### 5. **Almacenamiento Local: localStorage Justificación:**

- Persistencia del carrito entre sesiones - No requiere backend para datos temporales - API simple y síncrona - Soporte universal en navegadores modernos - 5-10MB de capacidad suficiente #### DevOps & Infraestructura ##### 1. **Containerización: Docker Justificación:** - Entorno consistente entre desarrollo y producción - Aislamiento de dependencias - Fácil despliegue y escalabilidad - Reproducibilidad del ambiente - Docker Compose para orquestación local ##### 2. **Proxy Inverso: Nginx (implícito en producción) Justificación:** - Balanceo de carga - Compresión gzip - Caché de contenido estático - Terminación SSL/TLS - Alto rendimiento para servir assets ##### 3. **CORS: Configurado en FastAPI Justificación:**

- Permite comunicación frontend-backend en desarrollo - Seguridad configurable por origen - Necesario para arquitectura SPA - Control granular de métodos y headers #### Patrones de Diseño Aplicados ##### 1. **Repository Pattern (Backend)** - Abstracción de la capa de datos - Facilita testing con mocks - Separación de lógica de negocio y persistencia ##### 2. **Dependency Injection (FastAPI)** - Inyección de dependencias nativa - Gestión automática del ciclo de vida - Testing más sencillo ##### 3. **Component Pattern (React)**

- Componentes reutilizables - Separación de responsabilidades - Composición sobre herencia ##### 4. **Context Provider Pattern (React)** 🔐 Implementar paginación en listado de productos - [] Agregar búsqueda avanzada con filtros - [] Sistema de valoraciones y reseñas - [] Notificaciones en tiempo real #### Medio Plazo - [] Panel de administración completo - [] Sistema de inventario avanzado - [] Integración con pasarelas de pago reales (Webpay, Mercado Pago) - [] Historial de pedidos #### Largo Plazo - [] Migrar a microservicios si escala - [] Implementar caché con Redis - [] Sistema de recomendaciones con ML - [] App móvil nativa (React Native) --- ## 🔒 Consideraciones de Seguridad #### Implementadas Contraseñas hasheadas con bcrypt Autenticación JWT con expiración Validación de datos en backend (Pydantic)

CORS configurado SQL Injection prevenido (ORM) Variables de entorno para secrets ### Por Implementar ✅ Rate limiting en API ✅ HTTPS en producción ✅ Refresh tokens ✅ Two-factor authentication (2FA) ✅ Logging de actividad de usuarios ✅ Validación de XSS en frontend --- ## 📄 Notas de Desarrollo ### Cuentas de Prueba Admin:

- Email: admin@admin.com
- Password: admin123

Usuario Test:

- Email: test@user.com
- Password: test123

Puertos Utilizados

Frontend: http://localhost:5175 Backend: http://localhost:8000 MySQL: localhost:3306 Swagger Docs: http://localhost:8000/docs ReDoc: http://localhost:8000/redoc

Variables de Entorno Requeridas

env

Backend

DATABASE_URL=mysql://user:password@localhost:3306/studimarket SECRET_KEY=tu-clave-secreta-super-segura ALGORITHM=HS256 ACCESS_TOKEN_EXPIRE_MINUTES=30

Frontend

VITE_API_URL=http://localhost:8000/api --- ## 👤 Equipo de Desarrollo **Desarrollador Principal:** CrisLeonAvello **Repositorio:** github.com/CrisLeonAvello/web-mini-market-Universitario **Licencia:** MIT --- **Generado:** Noviembre 2025 **Versión del Documento:** 1.0.0