



MANUAL TÉCNICO

Sistema de Inventario y Ventas - Ferretería Nacional

Versión: 1.0

Fecha: Diciembre 2024

Desarrollado por: Equipo de Desarrollo

Tecnologías: Angular 15+ | ASP.NET Core 6+ | SQL Server | Entity Framework



TABLA DE CONTENIDOS

- [1. Introducción](#)
 - [2. Arquitectura del Sistema](#)
 - [3. Diagrama de Entidades](#)
 - [4. Componentes del Sistema](#)
 - [5. Tecnologías Utilizadas](#)
 - [6. Instalación y Configuración](#)
 - [7. Guía de Usuario](#)
 - [8. API Documentation](#)
 - [9. Troubleshooting](#)
 - [10. Mantenimiento](#)
-



INTRODUCCIÓN

Propósito del Sistema

El **Sistema de Inventario y Ventas** es una aplicación web integral diseñada específicamente para ferreterías nacionales como Construplaza. El sistema automatiza y optimiza los procesos de:

- ✓ **Gestión de inventario** con control de stock en tiempo real
- ✓ **Procesamiento de ventas** con punto de venta intuitivo
- ✓ **Administración de productos** con categorización avanzada
- ✓ **Control de proveedores** y cadena de suministro
- ✓ **Generación de reportes** ejecutivos y operativos
- ✓ **Dashboard analítico** con KPIs en tiempo real

Objetivos del Sistema

1. **Eficiencia Operativa:** Reducir tiempos de procesamiento de ventas en 60%
 2. **Control de Inventario:** Eliminar quiebres de stock mediante alertas automáticas
 3. **Trazabilidad:** Seguimiento completo de productos desde compra hasta venta
 4. **Reportería:** Información ejecutiva para toma de decisiones estratégicas
 5. **Experiencia de Usuario:** Interfaz intuitiva y responsiva para todos los dispositivos
-



ARQUITECTURA DEL SISTEMA

Arquitectura General

El sistema implementa una **arquitectura de 3 capas** con separación clara de responsabilidades:

CAPA DE PRESENTACIÓN
(Angular 15+)

Dashboard | Productos | Ventas & POS

Categorías | Proveedores | Reportes

HTTP/HTTPS
REST API

CAPA DE NEGOCIO
(ASP.NET Core 6+)

Controllers | Services | Middleware

DTOs | Validators | Authentication

Entity Framework
Core ORM

CAPA DE DATOS
(SQL Server)

Tablas | Índices | Triggers

Vistas | SPs | Funciones

Patrones de Diseño Implementados

1. Repository Pattern

- Abstrae el acceso a datos
- Facilita testing con mocks
- Centraliza lógica de consultas

2. Unit of Work Pattern

- Gestiona transacciones complejas
- Garantiza consistencia de datos
- Optimiza rendimiento

3. Dependency Injection

- Inversión de control
- Facilita testing unitario
- Código más mantenible

4. DTO (Data Transfer Objects)

- Serialización optimizada
- Validación de entrada
- Mapeo de objetos



DIAGRAMA DE ENTIDADES

Modelo de Base de Datos

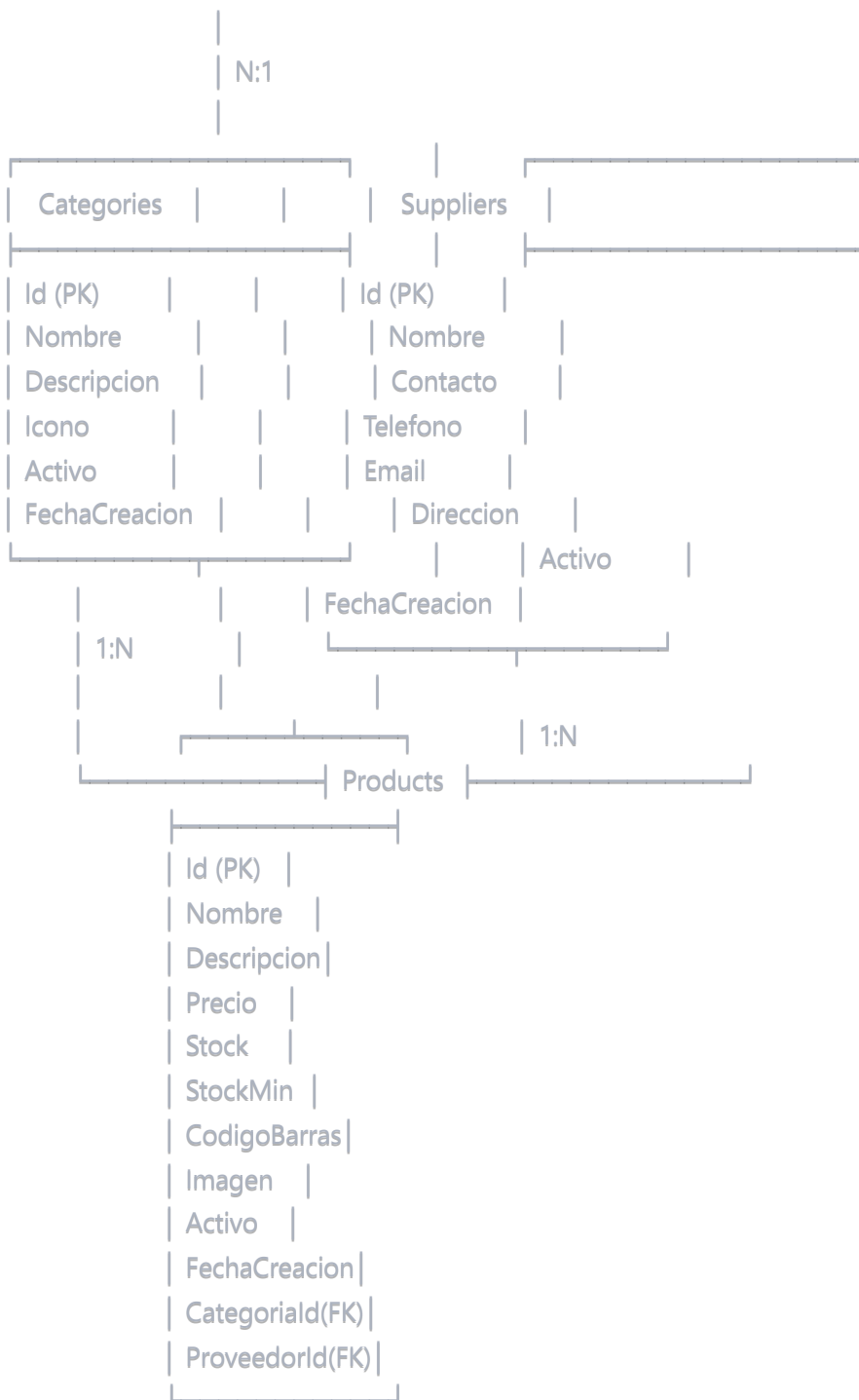
Users	
Id (PK)	
Nombre	
Email (UQ)	
PasswordHash	
Rol	
Activo	
FechaCreacion	

1:N

Sales	
Id (PK)	
NumeroVenta(UQ)	
ClienteNombre	
Clienteld	
ClienteTelefono	
Total	
Impuesto	
Descuento	
FechaVenta	
Estado	
Usuariold (FK)	

1:N

SaleDetails	
Id (PK)	
Cantidad	
PrecioUnitario	
Descuento	
Subtotal	
Ventald (FK)	
Productold (FK)	



Relaciones del Modelo

Relaciones Uno a Muchos (1:N)

1. **Users** → **Sales**: Un usuario puede realizar múltiples ventas
2. **Sales** → **SaleDetails**: Una venta puede tener múltiples detalles
3. **Categories** → **Products**: Una categoría puede tener múltiples productos
4. **Suppliers** → **Products**: Un proveedor puede suministrar múltiples productos

Relaciones Muchos a Muchos (N:M)

1. **Products** ↔ **Sales**: A través de la tabla SaleDetails

Restricciones e Integridad

Claves Primarias

- Todas las tablas tienen clave primaria auto-incremental
- Garantizan unicidad de registros

Claves Foráneas

- Mantienen integridad referencial
- Evitan registros huérfanos
- Configuradas con DELETE RESTRICT para proteger datos

Restricciones de Dominio

```
sql

-- Precios no negativos
CHECK (Precio >= 0)

-- Stock no negativo
CHECK (Stock >= 0)

-- Stock mínimo positivo
CHECK (StockMinimo > 0)

-- Estados válidos para ventas
CHECK (Estado IN ('Completada', 'Cancelada', 'Pendiente'))

-- Roles válidos para usuarios
CHECK (Rol IN ('Admin', 'Vendedor', 'Cliente'))
```

COMPONENTES DEL SISTEMA

Frontend (Angular 15+)

Estructura de Componentes

```

src/app/
├── components/
│   ├── auth/
│   │   └── login/          # Autenticación
│   ├── layout/            # Layout principal
│   ├── dashboard/         # Dashboard ejecutivo
│   ├── products/
│   │   ├── products.component    # Lista de productos
│   │   └── product-dialog/      # Formulario de producto
│   ├── sales/
│   │   ├── sales.component       # Historial de ventas
│   │   └── sale-dialog/         # Punto de venta
│   ├── categories/            # Gestión de categorías
│   └── suppliers/             # Gestión de proveedores
├── services/
│   ├── auth.service           # Servicio de autenticación
│   ├── product.service        # Servicio de productos
│   ├── sale.service           # Servicio de ventas
│   ├── category.service       # Servicio de categorías
│   ├── supplier.service       # Servicio de proveedores
│   └── dashboard.service      # Servicio de dashboard
├── models/                   # Interfaces TypeScript
├── guards/                   # Guards de routing
├── interceptors/             # HTTP Interceptors
└── shared/                   # Componentes compartidos

```

Servicios Principales

1. AuthService

```

typescript

export class AuthService {
  private currentUserSubject = new BehaviorSubject<User | null>(null);

  login(credentials: LoginRequest): Observable<LoginResponse>
  logout(): void
  isLoggedIn(): boolean
  getCurrentUser(): User | null
  generateJwtToken(user: User): string
}

```

2. ProductService

typescript

```
export class ProductService {  
  getProducts(): Observable<Product[]>  
  getProduct(id: number): Observable<Product>  
  createProduct(product: Product): Observable<Product>  
  updateProduct(id: number, product: Product): Observable<any>  
  deleteProduct(id: number): Observable<any>  
  getLowStockProducts(): Observable<Product[]>  
}
```

3. SaleService

typescript

```
export class SaleService {  
  getSales(): Observable<Sale[]>  
  createSale(sale: CreateSale): Observable<Sale>  
  getDailyReport(fecha?: Date): Observable<any>  
}
```

Guards y Security

AuthGuard

typescript

```
@Injectable()  
export class AuthGuard implements CanActivate {  
  canActivate(): boolean {  
    if (this.authService.isLoggedIn()) {  
      return true;  
    }  
    this.router.navigate(['/login']);  
    return false;  
  }  
}
```

AuthInterceptor

typescript

```

@Inject()
export class AuthInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const token = this.authService.getToken();
    if (token) {
      req = req.clone({
        setHeaders: { Authorization: `Bearer ${token}` }
      });
    }
    return next.handle(req);
  }
}

```

Backend (ASP.NET Core 6+)

Estructura de Controllers

1. AuthController

```

csharp

[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] LoginDto loginDto)

    [HttpPost("register")]
    public async Task<IActionResult> Register([FromBody] RegisterDto registerDto)
}

```

2. ProductsController

```

csharp

```

```

[ApiController]
[Route("api/[controller]")]
[Authorize]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public async Task<ActionResult<IEnumerable<ProductDto>>> GetProducts()

    [HttpGet("{id}")]
    public async Task<ActionResult<ProductDto>> GetProduct(int id)

    [HttpPost]
    [Authorize(Roles = "Admin,Vendedor")]
    public async Task<ActionResult<Product>> CreateProduct(Product product)

    [HttpPut("{id}")]
    [Authorize(Roles = "Admin,Vendedor")]
    public async Task<ActionResult> UpdateProduct(int id, Product product)

    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult> DeleteProduct(int id)

    [HttpGet("low-stock")]
    public async Task<ActionResult<IEnumerable<ProductDto>>> GetLowStockProducts()
}

```

3. SalesController

csharp

```

[ApiController]
[Route("api/[controller]")]
[Authorize]
public class SalesController : ControllerBase
{
    [HttpGet]
    public async Task<ActionResult<IEnumerable<SaleDto>>> GetSales()

    [HttpPost]
    [Authorize(Roles = "Admin,Vendedor")]
    public async Task<ActionResult<Sale>> CreateSale(CreateSaleDto createSaleDto)

    [HttpGet("reports/daily")]
    public async Task<ActionResult> GetDailyReport(DateTime? fecha = null)
}

```

Data Layer

DbContext Configuration

```

csharp

public class FerreriaContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Sale> Sales { get; set; }
    public DbSet<SaleDetail> SaleDetails { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Configuración de relaciones
        // Configuración de índices
        // Datos semilla
    }
}

```

Repository Pattern

```

csharp

```

```
public class GenericRepository<T> : IGenericRepository<T> where T : class
{
    private readonly FerreriaContext _context;
    private readonly DbSet<T> _db;

    public async Task<IEnumerable<T>> GetAllAsync()
    public async Task<T> GetByIdAsync(int id)
    public async Task<T> AddAsync(T entity)
    public async Task UpdateAsync(T entity)
    public async Task DeleteAsync(T entity)
}
```

TECNOLOGÍAS UTILIZADAS

Frontend Stack

Angular 15+

- **Framework principal:** Single Page Application
- **TypeScript:** Tipado estático para JavaScript
- **RxJS:** Programación reactiva y observables
- **Angular CLI:** Herramientas de desarrollo y build

Angular Material

- **UI Components:** Componentes pre-diseñados
- **Theming:** Personalización de colores y estilos
- **Responsive Design:** Adaptable a todos los dispositivos
- **Accessibility:** Cumple estándares WCAG

Additional Libraries

```
json
{
  "chart.js": "^4.2.1",    // Gráficos interactivos
  "ng2-charts": "^4.1.1",  // Wrapper Angular para Chart.js
  "angular-material": "^15.0.0" // UI Framework
}
```

Backend Stack

ASP.NET Core 6+

- **Web API:** RESTful services
- **Dependency Injection:** IoC container nativo
- **Middleware Pipeline:** Procesamiento de requests
- **Configuration:** Sistema de configuración flexible

Entity Framework Core

- **ORM:** Object-Relational Mapping
- **Code First:** Migrations automáticas
- **LINQ:** Consultas integradas en C#
- **Change Tracking:** Seguimiento automático de cambios

Authentication & Security

```
csharp

// JWT Authentication
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options => {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true
        };
    });

// Password Hashing
BCrypt.Net.BCrypt.HashPassword(password);
```

Database Stack

SQL Server

- **Relational Database:** ACID compliance
- **Stored Procedures:** Lógica de negocio optimizada

- **Triggers:** Automatización de procesos
- **Indexing:** Optimización de consultas

Database Features

- **Transactional Integrity:** Garantía ACID
 - **Backup & Recovery:** Estrategias de respaldo
 - **Security:** Encriptación y acceso controlado
 - **Scalability:** Soporte para crecimiento
-

INSTALACIÓN Y CONFIGURACIÓN

Requisitos del Sistema

Hardware Mínimo

- **Procesador:** Intel i3 / AMD Ryzen 3 o superior
- **Memoria RAM:** 8GB DDR4
- **Almacenamiento:** 50GB SSD disponible
- **Red:** Conexión a internet estable

Software Requerido

- ✓ Node.js 16.x o superior
- ✓ .NET 6 SDK
- ✓ SQL Server 2019 Express o superior
- ✓ Git 2.x
- ✓ Visual Studio Code (recomendado)
- ✓ Angular CLI 15.x

Instalación Paso a Paso

Fase 1: Preparación del Entorno

1. Verificar Instalaciones

```
bash
```

```
# Verificar Node.js
node --version # v16.x.x o superior
npm --version # 8.x.x o superior

# Verificar .NET
dotnet --version # 6.0.x o superior

# Verificar Git
git --version # 2.x.x o superior
```

2. Instalar Angular CLI

```
bash

npm install -g @angular/cli@15
ng version
```

3. Configurar SQL Server

- Descargar e instalar SQL Server Express
- Configurar instancia con autenticación mixta
- Habilitar TCP/IP en SQL Server Configuration Manager
- Abrir puerto 1433 en firewall

Fase 2: Configuración del Backend

1. Clonar y Configurar Proyecto

```
bash

git clone https://github.com/tu-usuario/ferreteria-system.git
cd ferreteria-system/backend
```

2. Configurar cadena de conexión

Editar `appsettings.json`:

```
json
```



```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost\\SQLEXPRESS;Database=FerreriaDB;Trusted_Connection=true;TrustServerCertificate=true;"
  },
  "Jwt": {
    "Key": "tu-clave-secreta-super-segura-de-al-menos-32-caracteres",
    "Issuer": "FerreriaAPI",
    "Audience": "FerreriaClient",
    "ExpireDays": 7
  }
}
```

3. Restaurar Paquetes y Crear Base de Datos

```
bash

# Restaurar paquetes NuGet
dotnet restore

# Instalar herramientas EF
dotnet tool install --global dotnet-ef

# Crear migración inicial
dotnet ef migrations add InitialCreate

# Actualizar base de datos
dotnet ef database update

# Ejecutar API
dotnet run
```

✅ **Backend corriendo en:** <https://localhost:5001>

Fase 3: Configuración del Frontend

1. Navegar e Instalar Dependencias

```
bash

cd ../frontend
npm install
```

2. Verificar Configuración de Ambiente

Revisar `src/environments/environment.ts`:

```
typescript

export const environment = {
  production: false,
  apiUrl: 'https://localhost:5001/api'
};
```

3. Ejecutar Aplicación

```
bash

ng serve --open
```

✅ **Frontend corriendo en:** `http://localhost:4200`

Fase 4: Verificación de la Instalación

1. Acceder a la Aplicación

- URL: `http://localhost:4200`
- Credenciales de prueba:
 - **Admin:** `admin@ferreteria.com` / `admin123`
 - **Vendedor:** `vendedor@ferreteria.com` / `vendedor123`

2. Verificar API

- Swagger UI: `https://localhost:5001/swagger`
- Health Check: `https://localhost:5001/api/health`

3. Verificar Base de Datos

```
sql

USE FerreriaDB;
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES;
SELECT COUNT(*) FROM Users;
SELECT COUNT(*) FROM Products;
```

Configuración de Producción

Configuraciones de Seguridad

1. Variables de Entorno

```
bash

# Configurar variables del sistema
ASPNETCORE_ENVIRONMENT=Production
JWT_SECRET_KEY=tu-clave-super-secreta-de-produccion
CONNECTION_STRING=Server=prod-server;Database=FerreriaDB;...
```

2. Configurar HTTPS

```
csharp

// Startup.cs o Program.cs
app.UseHttpsRedirection();
app.UseHsts();
```

3. Configurar CORS para Producción

```
csharp

services.AddCors(options =>
{
    options.AddPolicy("Production", policy =>
    {
        policy.WithOrigins("https://tu-dominio-produccion.com")
            .AllowAnyHeader()
            .AllowAnyMethod();
    });
});
```

Build para Producción

1. Backend

```
bash

dotnet publish -c Release -o ./publish
```







2. Frontend

```
bash
```







GUÍA DE USUARIO

Roles y Permisos




Administrador

-  **Acceso completo** a todas las funcionalidades
-  **Gestión de usuarios** (crear, editar, desactivar)
-  **Configuración del sistema** y parámetros
-  **Reportes ejecutivos** y análisis avanzados
-  **Gestión de categorías y proveedores**
-  **Eliminación de registros**

Vendedor

-  **Punto de venta** completo
-  **Gestión de productos** (crear, editar)
-  **Consulta de inventario** y stock
-  **Historial de ventas** propias
-  **Dashboard básico** con KPIs
-  **No puede eliminar** productos o ventas

Cliente (Opcional)

-  **Consulta de productos** y precios
-  **Historial de compras** personal
-  **No acceso** a funciones administrativas

Manual de Usuario por Módulo

1. Dashboard Ejecutivo

Acceso al Dashboard

1. Iniciar sesión con credenciales válidas
2. El dashboard se carga automáticamente

3. Vista panorámica de KPIs principales

KPIs Disponibles

- **Ventas del Mes:** Monto total facturado
- **Productos en Stock:** Cantidad de productos disponibles
- **Ventas Hoy:** Número de transacciones del día
- **Stock Bajo:** Productos que requieren reabastecimiento

Gráficos y Análisis

- **Ventas por Mes:** Tendencias de facturación
- **Top 10 Productos:** Artículos más vendidos
- **Alertas de Stock:** Productos críticos

2. Gestión de Productos


Listar Productos

1. Menu lateral → **Productos**
2. Vista tabular con información completa
3. **Filtros disponibles:**
 - Búsqueda por nombre
 - Filtro por categoría
 - Filtro por proveedor
 - Estado de stock


Crear Nuevo Producto

1. Clic en botón "**Nuevo Producto**"
2. Completar formulario:
 - **Información básica:** Nombre, descripción
 - **Precio y stock:** Precio, stock actual, stock mínimo
 - **Categorización:** Categoría y proveedor
 - **Opciones:** Código de barras, imagen
3. Clic en "**Crear**"

Editar Producto

1. Clic en icono de edición ()
2. Modificar campos necesarios
3. Clic en "**Actualizar**"

Eliminar Producto (Solo Admin)

1. Clic en icono de eliminación ()
2. Confirmar acción en diálogo
3. Se realiza **eliminación lógica** (no física)

3. Sistema de Ventas

Crear Nueva Venta

1. Menu lateral → **Ventas**
2. Clic en botón "**Nueva Venta**"
3. **Información del Cliente:**
 - Nombre (obligatorio)
 - Identificación (opcional)
 - Teléfono (opcional)

Agregar Productos a la Venta

1. Seleccionar producto del dropdown
2. Especificar cantidad (máximo = stock disponible)
3. Aplicar descuento individual (opcional)
4. Clic en "+" para agregar más productos

Finalizar Venta

1. Revisar detalles en resumen
2. Aplicar descuento general (opcional)
3. Verificar totales:
 - Subtotal
 - IVA (13%)
 - Total final
4. Clic en "**Registrar Venta**"

Consultar Historial de Ventas






- Lista completa de transacciones
- Filtros por fecha y cliente
- Detalles de cada venta
- Estado de la transacción

4. Gestión de Categorías

Crear Categoría

1. Menu lateral → **Categorías**
2. Clic en "**Nueva Categoría**"
3. Completar:
 - Nombre de la categoría
 - Descripción
 - Icono representativo
4. Guardar cambios

Organización de Categorías

- **Herramientas:**  Herramientas manuales y eléctricas
- **Materiales:**  Materiales de construcción
- **Plomería:**  Accesorios de plomería
- **Electricidad:**  Materiales eléctricos
- **Jardinería:**  Herramientas de jardín

5. Gestión de Proveedores

Registrar Proveedor

1. Menu lateral → **Proveedores**
2. Clic en "**Nuevo Proveedor**"
3. Información de contacto:
 - Nombre de la empresa
 - Persona de contacto
 - Teléfono y email
 - Dirección física

4. Guardar información

Gestión de Relaciones

- Asignación de productos por proveedor
- Historial de suministros
- Información de contacto actualizada

Mejores Prácticas de Uso

Gestión de Inventario

1. **Revisar alertas diariamente** en el dashboard
2. **Actualizar stock mínimo** según rotación de productos
3. **Registrar ingresos** de mercadería inmediatamente
4. **Auditorías periódicas** de inventario físico

Proceso de Ventas

1. **Verificar stock** antes de confirmar venta
2. **Capturar datos completos** del cliente
3. **Revisar totales** antes de finalizar
4. **Imprimir comprobante** de venta

Seguridad del Sistema

1. **Cambiar contraseñas** periódicamente
2. **Cerrar sesión** al terminar el turno
3. **No compartir credenciales** entre usuarios
4. **Reportar problemas** inmediatamente



API DOCUMENTATION

Base URL

`https://localhost:5001/api`

Authentication

POST /api/auth/login

Autenticar usuario y obtener token JWT.

Request Body:

```
json
{
  "email": "admin@ferreteria.com",
  "password": "admin123"
}
```

Response:

```
json
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "message": "Login exitoso"
}
```

POST /api/auth/register

Registrar nuevo usuario.

Request Body:

```
json
{
  "nombre": "Nuevo Usuario",
  "email": "usuario@email.com",
  "password": "password123",
  "rol": "Vendedor"
}
```

Products API

GET /api/products

Obtener lista de productos activos.

Headers:

Authorization: Bearer {jwt-token}

Response:

```
json
[
  {
    "id": 1,
    "nombre": "Martillo de Carpintero 16oz",
    "descripcion": "Martillo con mango de madera",
    "precio": 15.50,
    "stock": 25,
    "stockMinimo": 5,
    "categoriaNombre": "Herramientas",
    "proveedorNombre": "Distribuidora Central"
  }
]
```

POST /api/products

Crear nuevo producto.

Headers:

Authorization: Bearer {jwt-token}
Content-Type: application/json

Request Body:

```
json
{
  "nombre": "Nuevo Producto",
  "descripcion": "Descripción del producto",
  "precio": 25.99,
  "stock": 100,
  "stockMinimo": 10,
  "categoriald": 1,
  "proveedorId": 1,
  "activo": true
}
```

Sales API

GET /api/sales

Obtener historial de ventas.

Response:

```
json
[
  {
    "id": 1,
    "numeroVenta": "V20241201143022",
    "clienteNombre": "Juan Pérez",
    "total": 156.75,
    "fechaVenta": "2024-12-01T14:30:22",
    "estado": "Completada",
    "usuarioNombre": "Vendedor Demo",
    "detallesVenta": [
      {
        "cantidad": 2,
        "precioUnitario": 15.50,
        "subtotal": 31.00,
        "productoNombre": "Martillo de Carpintero 16oz"
      }
    ]
  }
]
```

POST /api/sales

Registrar nueva venta.

Request Body:

```
json
```

```
{
  "clienteNombre": "Cliente Ejemplo",
  "clienteIdentificacion": "123456789",
  "clienteTelefono": "2222-3333",
  "descuento": 0,
  "detallesVenta": [
    {
      "productoid": 1,
      "cantidad": 2,
      "descuento": 0
    }
  ]
}
```

Dashboard API

GET /api/dashboard/kpis

Obtener KPIs del dashboard.

Response:

```
json
{
  "ventasDelMes": 45680.50,
  "ventasHoy": 12,
  "productosEnStock": 1247,
  "productosStockBajo": 23
}
```

Error Handling

Códigos de Estado HTTP

- `200 OK`: Operación exitosa
- `201 Created`: Recurso creado exitosamente
- `400 Bad Request`: Datos de entrada inválidos
- `401 Unauthorized`: Token inválido o expirado
- `403 Forbidden`: Permisos insuficientes
- `404 Not Found`: Recurso no encontrado

- `500 Internal Server Error`: Error del servidor

Formato de Errores

```
json
{
  "message": "Descripción del error",
  "errors": {
    "campo": ["Error específico del campo"]
  },
  "timestamp": "2024-12-01T14:30:22Z"
}
```

TROUBLESHOOTING

Problemas Comunes

Error de Conexión a Base de Datos

Síntomas:

- Error: "A network-related or instance-specific error occurred"
- No se puede conectar a SQL Server

Soluciones:

1. Verificar servicio SQL Server:

```
cmd
services.msc
# Buscar SQL Server y verificar que esté ejecutándose
```

2. Verificar cadena de conexión:

```
json
"Server=localhost\\SQLEXPRESS;Database=FerreriaDB;Trusted_Connection=true;TrustServerCertificate=true;"
```

3. Verificar protocolo TCP/IP:

- Abrir SQL Server Configuration Manager
- Protocols for SQLEXPRESS → TCP/IP → Enable

Error CORS en Frontend

Síntomas:

- Error: "Access to fetch at '<https://localhost:5001>' blocked by CORS policy"

Solución: Verificar configuración CORS en `Program.cs`:

```
csharp
app.UseCors("AllowAngularApp");
```

Token JWT Expirado

Síntomas:

- Error 401 Unauthorized
- Usuario redirigido al login constantemente

Solución:

1. **Verificar tiempo de expiración** en `appsettings.json`
2. **Limpiar localStorage:**

```
javascript
localStorage.clear();
```

Error de Migración EF

Síntomas:

- Error al ejecutar `dotnet ef database update`

Solución:

1. **Eliminar migraciones existentes:**

```
bash
rm -rf Migrations/
```

2. ****Crear nueva migración**