Inventory Management

NailStock

Documentación

Desarrollo de NailStock + Docs.

Cruz Flores Arleth Leilani
Barrera López Alejandro
Estrada Mendoza Esteban Uriel
García Acosta Moises
Corona Beltrán Diego Alessandro
Ángeles Juárez Valentín

NailStock: Index
Inventory and Sales Management System

Tabla de contenido

Descripción del Sistema

Descripción general:
NailStock es un sistema de gestión pensado especialmente para ferreterías y tiendas de materiales de construcción. Su objetivo principal es simplificar y automatizar tareas que tradicionalmente se hacen a mano, consumen mucho tiempo y son propensas a errores en este tipo de negocios.

Objectivo del sistema:
Desarrollar una aplicación que optimice la operación diaria de una ferretería, permitiendo gestionar el catálogo de productos, controlar niveles de inventario en tiempo real, procesar ventas de manera rápida y confiable, registrar clientes y proveedores, y generar reportes accesibles de ventas, stock y desempeño del negocio.

Alcance:

- Gestión de productos: agregar, editar y desactivar productos, incluyendo precios, categorías y unidades de medida.
- Control de inventario: registrar entradas y salidas de stock, alertas de productos con stock bajo.
- Procesamiento de ventas: registrar ventas rápidas con cálculo automático de totales, generación de recibos y actualización del inventario.
- Gestión de clientes y proveedores: mantener información completa para facturación, contacto y seguimiento.
- Búsqueda de productos: por código, nombre o categoría.
- Reportes y análisis: generar reportes de ventas por periodo, productos más vendidos, inventario valorizado y exportación a CSV.
- Seguridad y respaldo: login de usuario, prevención de eliminación de registros críticos y respaldo de base de datos.

Características del Sistema

Módulo de Productos

- Agregar nuevos productos con información completa (nombre, categoría, precio, stock, proveedor y unidad de medida)
- Buscar productos por nombre, categoría, código o proveedor
- Actualizar la información de productos existentes
- Desactivar productos en lugar de eliminarlos permanentemente
- Mostrar alertas visuales cuando un producto tiene stock bajo

Módulo de Inventario

- Mostrar todos los productos con sus niveles de stock en tiempo real
- Generar reportes de productos con stock bajo para facilitar la reposición
- Calcular la variación total del inventario (precio de compra * stock disponible)
- Filtrar inventario por categoría, proveedor o estado del producto

Módulo de Ventas

- Registrar ventas mediante una interfaz intuitiva
- Generar recibos o notas de venta (en texto o PDF)
- Buscar ventas por fecha, cliente o rango de fechas
- Generar reportes de ventas diarias, semanales y mensuales

Módulo de Clientes

- Registrar datos de clientes (nombre, teléfono, RFC y notas)
- Buscar, editar o desactivar clientes
- Consultar historial de compras por cliente

Módulo de Proveedores

- Registrar información de proveedores (nombre, teléfono, dirección y observaciones)
- Buscar, editar o eliminar proveedores
- Asociar proveedores con productos para mejor control de inventario

Reports

- Resumen de ventas diarias, semanales y mensuales
- Alertas de productos con stock bajo
- Historial de ventas por cliente o usuario

Diagramas UML

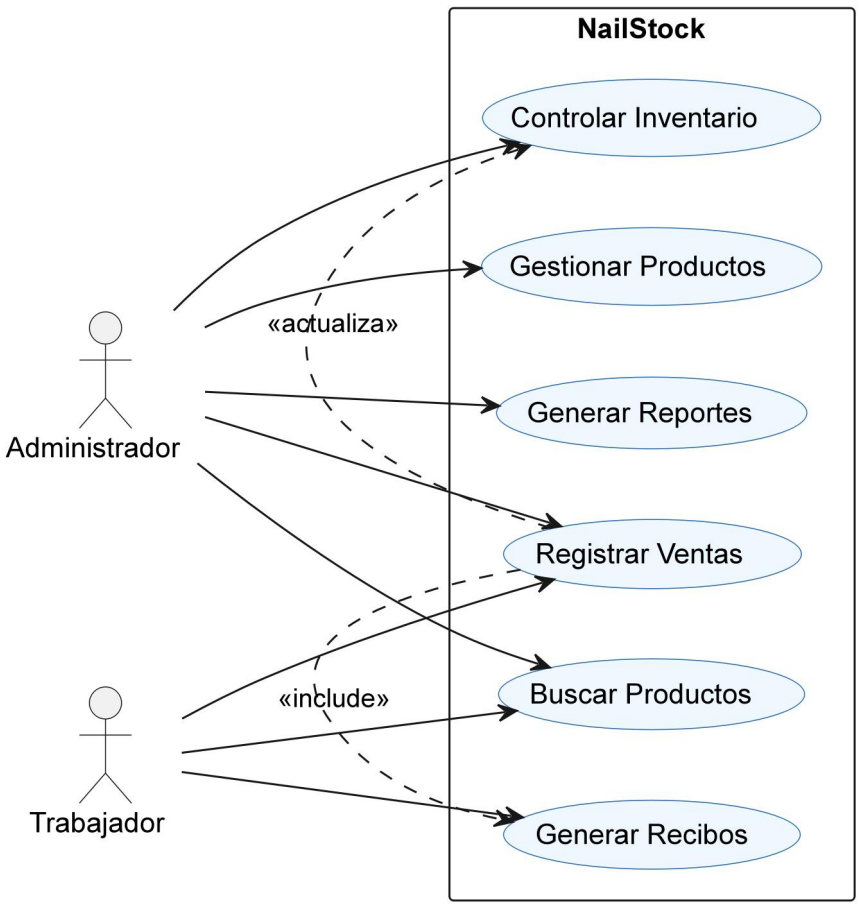
Casos de uso

Actores:

- Trabajador
- Administrador

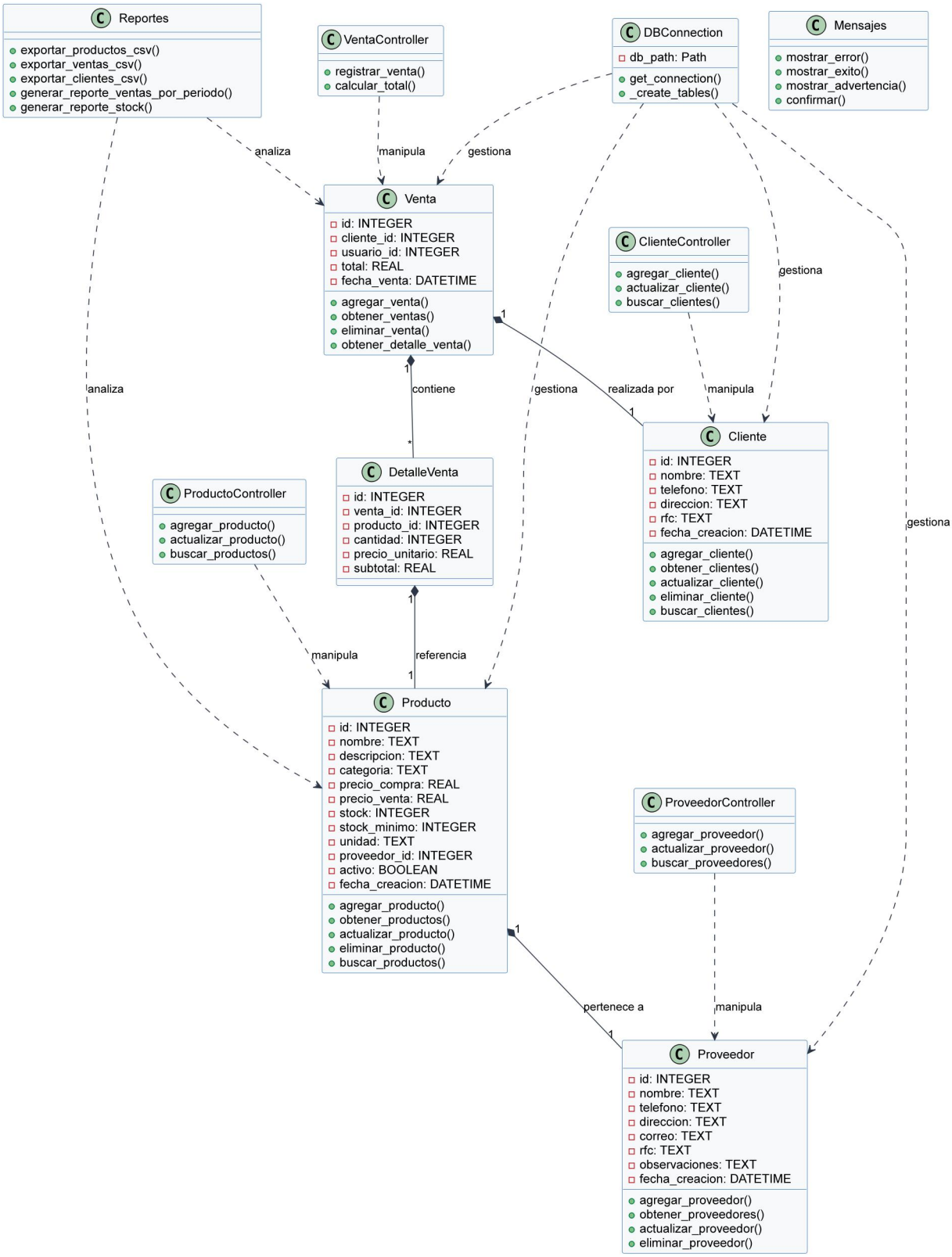
Casos de uso principales:

- Registrar producto
- Generar factura
- Actualizar inventario
- Consultar producto
- Registrar venta
- Ver reporte de ventas



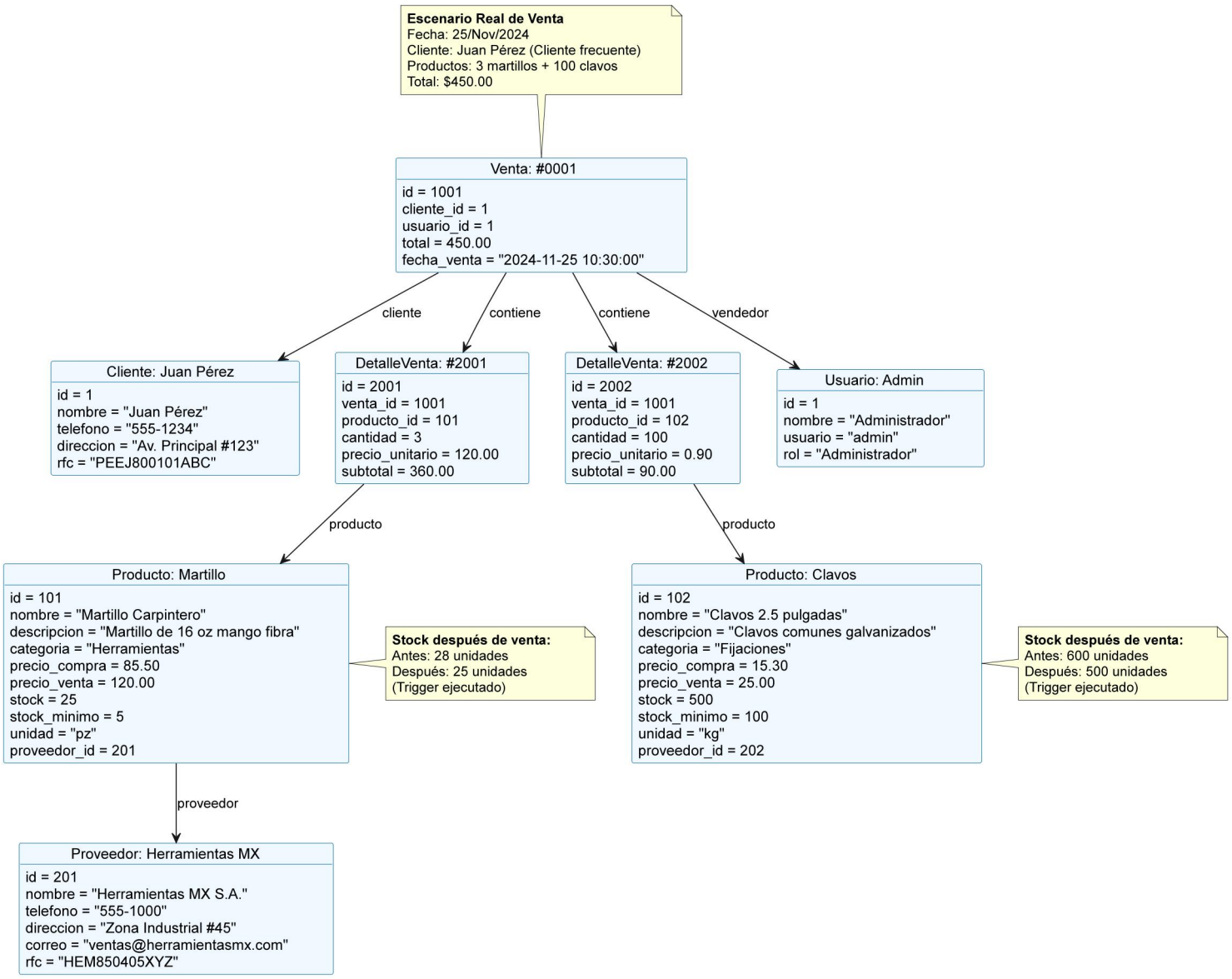
Diagramas UML

Diagrama de clases



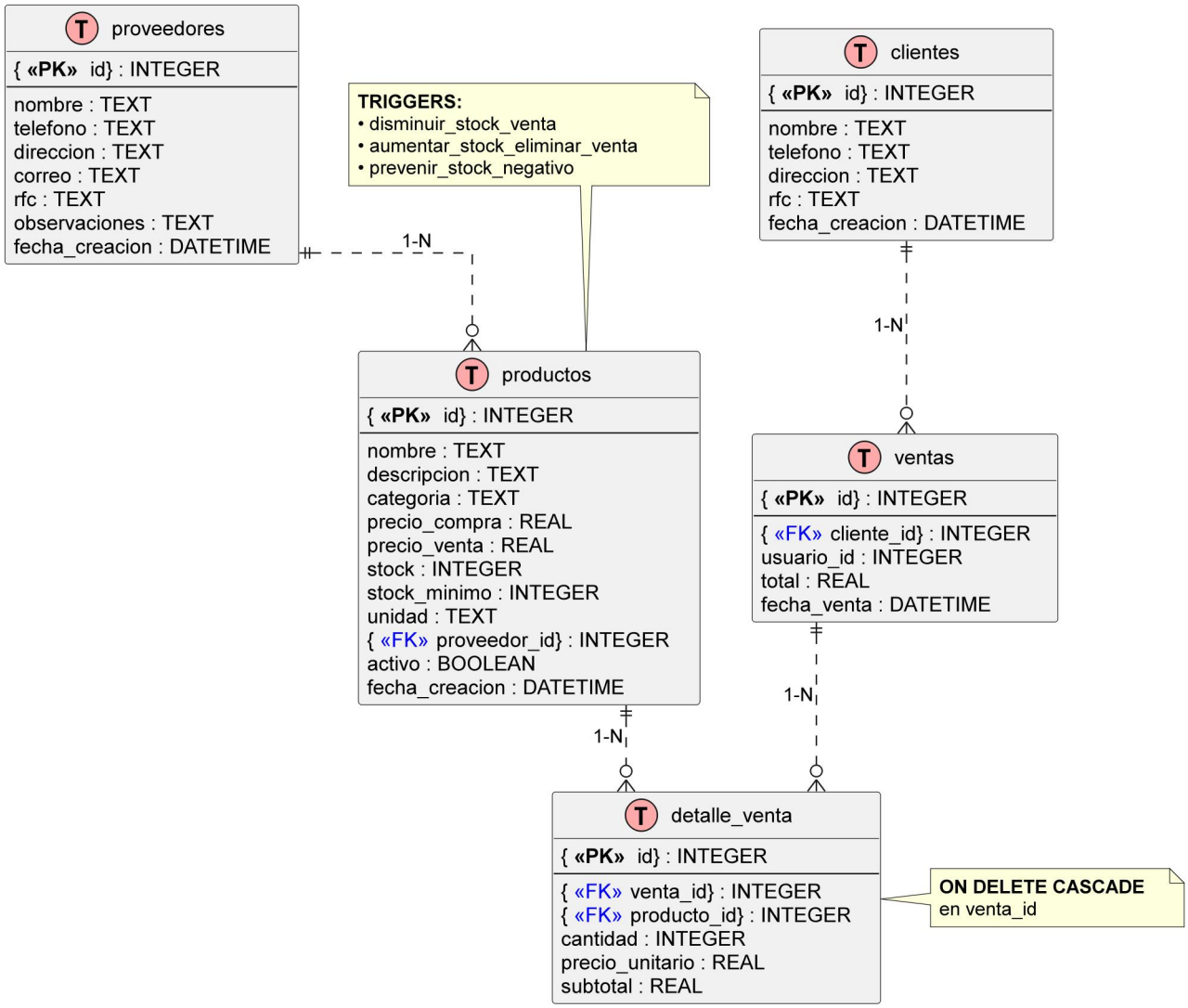
Diagramas UML

Diagrama de Objetos



Diseño de Base de Datos

Diagrama Entidad-Relación (DER)



Diseño de Base de Datos

Diccionario de datos

Tabla: proveedores

Campo	Tipo de Dato	Nulo	Descripción	Referencias
id	INTEGER	NO	Identificador único autoincremental	PK
nombre	TEXT	NO	Nombre o razón social del proveedor	-
telefono	TEXT	SI	Número de contacto del proveedor	-
direccion	TEXT	SI	Dirección física del establecimiento	-
correo	TEXT	SI	Correo electrónico para contacto	-
rfc	TEXT	SI	RFC para facturación electrónica	-
observaciones	TEXT	SI	Notas adicionales sobre el proveedor	-
fecha_creacion	DATETIME	NO	Fecha y hora de registro automático	-

Tabla: clientes

Campo	Tipo de Dato	Nulo	Descripción	Referencias
id	INTEGER	NO	Identificador único autoincremental	PK
nombre	TEXT	NO	Nombre completo del cliente	-
telefono	TEXT	SI	Teléfono de contacto del cliente	-
direccion	TEXT	SI	Dirección para entregas o facturación	-
rfc	TEXT	SI	RFC para facturación electrónica	-
fecha_creacion	DATETIME	NO	Fecha y hora de registro automático	-

Diseño de Base de Datos

Tabla: productos

Campo	Tipo de Dato	Nulo	Descripción	Referencias
id	INTEGER	NO	Identificador único autoincremental	PK
nombre	TEXT	NO	Nombre descriptivo del producto	-
descripcion	TEXT	SI	Características y detalles del producto	-
categoria	TEXT	SI	Categoría para organización y filtrado	-
precio_compra	REAL	NO	Precio de compra al proveedor (costo)	-
precio_venta	REAL	NO	Precio de venta al público	-
stock	INTEGER	NO	Cantidad disponible en inventario	-
stock_minimo	INTEGER	SI	Número mínimo para alertas de reabastecimiento	-
unidad	TEXT	NO	Unidad de medida (pz, kg, m, l, etc.)	-
proveedor_id	INTEGER	SI	Proveedor principal del producto	FK → proveedores(id)
activo	BOOLEAN	NO	Estado activo/inactivo (1=activo, 0=inactivo)	-
fecha_creacion	DATETIME	NO	Fecha y hora de registro automático	-

Tabla: ventas

Campo	Tipo de Dato	Nulo	Descripción	Referencias
id	INTEGER	NO	Identificador único autoincremental	PK
cliente_id	INTEGER	NO	Cliente que realiza la compra	FK → clientes(id)
usuario_id	INTEGER	NO	Usuario del sistema que registra la venta	-
total	REAL	NO	Suma total de la venta en pesos mexicanos	-
fecha_venta	DATETIME	NO	Fecha y hora de la transacción	-

Tabla: detalle_venta

Campo	Tipo de Dato	Nulo	Descripción	Referencias
id	INTEGER	NO	Identificador único autoincremental	PK
venta_id	INTEGER	NO	Venta a la que pertenece el detalle	FK → ventas(id) ON DELETE CASCADE
producto_id	INTEGER	NO	Producto incluido en la venta	FK → productos(id)
cantidad	INTEGER	NO	Cantidad vendida del producto	-
precio_unitario	REAL	NO	Precio unitario al momento de la venta	-
subtotal	REAL	NO	Total del renglón (cantidad × precio_unitario)	-

Diseño de Base de Datos

Script SQL con datos (Algunos ejemplos)

Insertar proveedores

```sql
INSERT INTO proveedores (nombre, telefono, direccion, correo, rfc, observaciones) VALUES
('Ferretería Industrial Mexicana',
'555-123-4567',
'Av. Industria 123, CDMX',
'ventas@ferreteriaindustrial.com',
'FIM890123456',
'Proveedor principal de herramientas industriales');
-- y más proveedores...
```

Insertar clientes

```sql
INSERT INTO clientes (nombre, telefono, direccion, rfc) VALUES
('Constructora Moderna S.A. de C.V.', '555-111-2233', 'Av. Revolución 123, CDMX', 'CMS650701456'),
('Ingeniería y Diseño Integral', '555-222-3344', 'Calzada Ingenieros 456, Monterrey', 'IDI750812567'),
('Taller Mecánico Rápido', '555-333-4455', 'Calle Mecánicos 789, Guadalajara', 'TMR850923678'),
('Proyectos Residenciales', '555-000-1122', 'Av. Residencial 741, Monterrey', 'PRS551690345');
-- y más clientes...
```

Insertar productos

```sql
INSERT INTO productos (nombre, descripcion, categoria, precio_compra, precio_venta, stock, stock_minimo, unidad,
proveedor_id) VALUES
('Martillo de Carpintero 16oz', 'Martillo profesional con mango de fibra de vidrio',
                        'Herramientas Manuales', 85.50, 150.00, 50, 10, 'pz', 1),
('Destornillador Plano 1/4"', 'Destornillador de punta plana profesional',
                        'Herramientas Manuales', 25.00, 45.00, 100, 20, 'pz', 1),
('Taladro Percutor 650W', 'Taladro percutor profesional de 650 watts',
                        'Herramientas Eléctricas', 450.00, 750.00, 25, 5, 'pz', 2),
('Sierra Circular 7-1/4"', 'Sierra circular con láser guía',
                        'Herramientas Eléctricas', 680.00, 1120.00, 15, 3, 'pz', 2);
-- y más...
```

Insertar detalles de ventas

```sql
INSERT INTO detalle_venta (venta_id, producto_id, cantidad, precio_unitario, subtotal) VALUES
(1, 6, 1, 750.00, 750.00),    -- Taladro Percutor
(1, 11, 5, 180.00, 900.00),   -- Cemento
(1, 26, 2, 75.00, 150.00),    -- Tubo PVC
(1, 31, 3, 190.00, 570.00),   -- Pintura Vinílica
(1, 36, 2, 80.00, 160.00),    -- Casco Seguridad
(1, 41, 50, 1.50, 75.00),     -- Tornillo Madera
(1, 16, 1, 680.00, 680.00);   -- Cable THHW
```

Operaciones CRUD Implementadas

Lista de funciones implementadas

- **Módulo Productos**

I. CREATE (Crear)

`ProductoController.agregar_producto()` - Valida y procesa datos
`ProductoModel.agregar_producto()` - Inserta en base de datos
Campos: nombre, descripción, categoría, precios, stock, unidad, proveedor

II. READ (Leer)

`ProductoModel.obtener_productos()` - Lista todos los productos activos
`ProductoModel.obtener_producto_por_id()` - Busca por ID específico
`ProductoModel.buscar_productos()` - Búsqueda por nombre o descripción
`ProductoController.buscar_productos()` - Búsqueda filtrada

III. UPDATE (Actualizar)

`ProductoController.actualizar_producto()` - Valida datos de actualización
`ProductoModel.actualizar_producto()` - Actualiza registro en BD
Actualizable: todos los campos excepto ID y fecha de creacion

IV. DELETE (Eliminar)

`ProductoModel.eliminar_producto()` - Eliminación lógica (activo = 0)
No permite eliminación física para mantener historial

- **Módulo Proveedores**

I. CREATE

`ProveedorController.agregar_proveedor()` - Valida datos del proveedor
`ProveedorModel.agregar_proveedor()` - Inserta nuevo proveedor
Campos: nombre, teléfono, dirección, correo, RFC, observaciones

II. READ

`ProveedorModel.obtener_proveedores()` - Lista todos los proveedores
`ProveedorModel.obtener_proveedor_por_id()` - Busca proveedor específico
`ProveedorController.buscar_proveedores()` - Búsqueda en múltiples campos

Operaciones CRUD Implementadas

III. UPDATE

`ProveedorController.actualizar_proveedor()` - Procesa actualización
`ProveedorModel.actualizar_proveedor()` - Ejecuta UPDATE en BD

IV. DELETE

`ProveedorModel.eliminar_proveedor()` - Eliminación física con validación
Verifica que no tenga productos asociados antes de eliminar

- **Módulo Clientes**

I. CREATE

`ClienteController.agregar_cliente()` - Valida datos del cliente
`ClienteModel.agregar_cliente()` - Inserta nuevo cliente
Campos: nombre, teléfono, dirección, RFC

II. READ

`ClienteModel.obtener_clientes()` - Lista todos los clientes
`ClienteModel.obtener_cliente_por_id()` - Busca cliente específico
`ClienteController.buscar_clientes()` - Búsqueda en nombre, teléfono, RFC

III. UPDATE

`ClienteController.actualizar_cliente()` - Procesa actualización
`ClienteModel.actualizar_cliente()` - Ejecuta UPDATE en BD

IV. DELETE

`ClienteModel.eliminar_cliente()` - Eliminación física con validación
Verifica que no tenga ventas asociadas antes de eliminar

Operaciones CRUD Implementadas

- **Módulo Ventas**

I. CREATE (Crear)

`VentaController.registrar_venta()` - Orquesta el proceso completo
`VentaModel.agregar_venta()` - Transacción completa con:
Inserción en tabla ventas
Inserción múltiple en detalle_venta
Triggers automáticos para control de stock

II. READ (Leer)

`VentaModel.obtener_ventas()` - Lista ventas con filtro de fechas
`VentaModel.obtener_detalle_venta()` - Obtiene detalles específicos
Incluye joins con clientes y productos

III. DELETE (Eliminar)

`VentaModel.eliminar_venta()` - Eliminación en cascada
Triggers automáticos restauran el stock

Operaciones CRUD Implementadas

Capturas de pantalla

ClienteModel (Única captura detallada)

```python
from ..database.db_connection import get_db_connection

class ClienteModel:
    @staticmethod
    def agregar_cliente(nombre, telefono, direccion, rfc):
        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute('''
            INSERT INTO clientes (nombre, telefono, direccion, rfc)
            VALUES (?, ?, ?, ?)
        ''', (nombre, telefono, direccion, rfc))

        cliente_id = cursor.lastrowid
        conn.commit()
        conn.close()

        return cliente_id

    @staticmethod
    def obtener_clientes():
        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute('SELECT * FROM clientes ORDER BY nombre')
        clientes = cursor.fetchall()
        conn.close()

        return clientes

    @staticmethod
    def obtener_cliente_por_id(cliente_id):
        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute('SELECT * FROM clientes WHERE id = ?', (cliente_id,))
        cliente = cursor.fetchone()
        conn.close()

        return cliente

    @staticmethod
    def actualizar_cliente(cliente_id, nombre, telefono, direccion, rfc):
        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute('''
            UPDATE clientes
            SET nombre = ?, telefono = ?, direccion = ?, rfc = ?
            WHERE id = ?
        ''', (nombre, telefono, direccion, rfc, cliente_id))

        conn.commit()
        conn.close()

        return True

    @staticmethod
    def eliminar_cliente(cliente_id):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT COUNT(*) FROM ventas WHERE cliente_id = ?', (cliente_id,))
        count = cursor.fetchone()[0]

        if count > 0:
            raise Exception("No se puede eliminar el cliente porque tiene ventas asociadas")

        cursor.execute('DELETE FROM clientes WHERE id = ?', (cliente_id,))
        conn.commit()
        conn.close()
        return True
```

Operaciones CRUD Implementadas

Capturas de ProductoModel, ProveedorModel, VentaModel simples.
El código completo está en GitHub. (https://github.com/Ismael-Software/proyecto-final-equipoverde/tree/main/nailstock/models)

ProdcutoModel

```python
from ..database.db_connection import get_db_connection

class ProductoModel:
    @staticmethod
    def agregar_producto(nombre, descripcion, categoria, precio_compra, precio_venta, ...

    @staticmethod
    def obtener_productos(activo=True):...

    @staticmethod
    def obtener_producto_por_id(producto_id):...

    @staticmethod
    def actualizar_producto(producto_id, nombre, descripcion, categoria, precio_compra, ...

    @staticmethod
    def eliminar_producto(producto_id):...

    @staticmethod
    def buscar_productos(termino):...
```

ProveedorModel

```python
from ..database.db_connection import get_db_connection

class ProveedorModel:
    @staticmethod
    def agregar_proveedor(nombre, telefono, direccion, correo, rfc, observaciones):...

    @staticmethod
    def obtener_proveedores():...

    @staticmethod
    def obtener_proveedor_por_id(proveedor_id):...

    @staticmethod
    def actualizar_proveedor(proveedor_id, nombre, telefono, direccion, correo, rfc, observaciones):...

    @staticmethod
    def eliminar_proveedor(proveedor_id):...
```

VentaModel

```python
from ..database.db_connection import get_db_connection

class VentaModel:
    @staticmethod
    def agregar_venta(cliente_id, usuario_id, productos):...

    @staticmethod
    def obtener_ventas(fecha_inicio=None, fecha_fin=None):...

    @staticmethod
    def obtener_detalle_venta(venta_id):...

    @staticmethod
    def eliminar_venta(venta_id):...
```

Operaciones CRUD Implementadas

Patrón Modelo-Vista-Controlador (MVC) Implementado

El sistema NailStock sigue el patrón arquitectónico MVC, donde:

- Modelos: Gestionan el acceso directo a la base de datos
- Vistas: Manejan la interfaz de usuario y presentación
- Controladores: Actúan como **intermediarios** entre vistas y modelos

Todas las operaciones CRUD se realizan exclusivamente a través de los controladores, nunca accediendo directamente a los modelos desde las vistas.

Ejemplo con *ProductoController:*

```python
from ..models.producto_model import ProductoModel

class ProductoController:
    @staticmethod
    def agregar_producto(nombre, descripcion, categoria, precio_compra, precio_venta,
                         stock, stock_minimo, unidad, proveedor_id):
        return ProductoModel.agregar_producto(
            nombre, descripcion, categoria, precio_compra, precio_venta,
            stock, stock_minimo, unidad, proveedor_id
        )

    @staticmethod
    def actualizar_producto(producto_id, nombre, descripcion, categoria, precio_compra,
                            precio_venta, stock, stock_minimo, unidad, proveedor_id):
        return ProductoModel.actualizar_producto(
            producto_id, nombre, descripcion, categoria, precio_compra, precio_venta,
            stock, stock_minimo, unidad, proveedor_id
        )

```

Ejemplo con *VentaController:*

```python
from ..models.venta_model import VentaModel

class VentaController:
    @staticmethod
    def registrar_venta(cliente_id, usuario_id, productos):
        return VentaModel.agregar_venta(cliente_id, usuario_id, productos)

    @staticmethod
    def calcular_total(productos):
        return sum(item['cantidad'] * item['precio_unitario'] for item in productos)
```

Backlog del proyecto

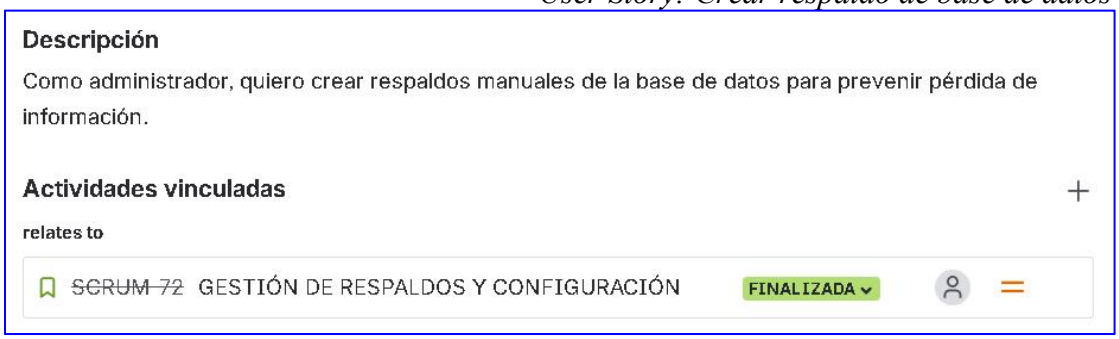
SCRUM

URL:
[https://nailstock.atlassian.net/jira/software/projects/SCRUM/boards/1?atlOrigin=eyJpIjoiNWIyOTUxMzA2NDQ1NDg4Y2EyZjAyNTUxNjVjZmEyOGMiLCJwIjoiaiJ9]

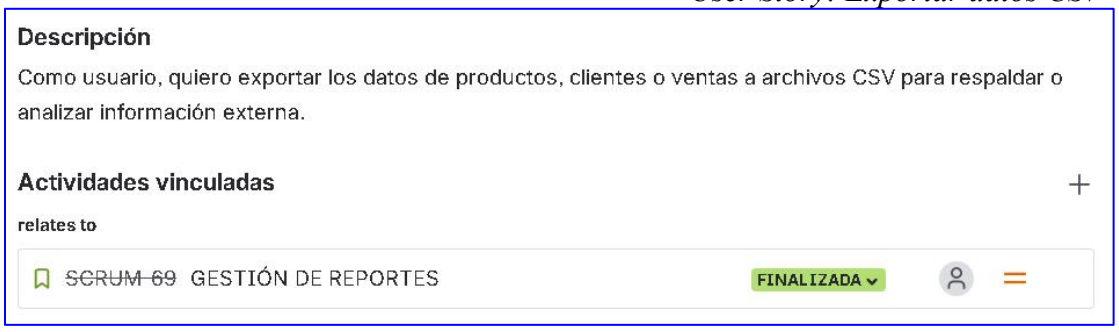
Jira fue seleccionado como herramienta de gestión del proyecto por las siguientes razones:

- User Stories con descripciones detalladas y criterios de aceptación

User Story: Crear respaldo de base de datos



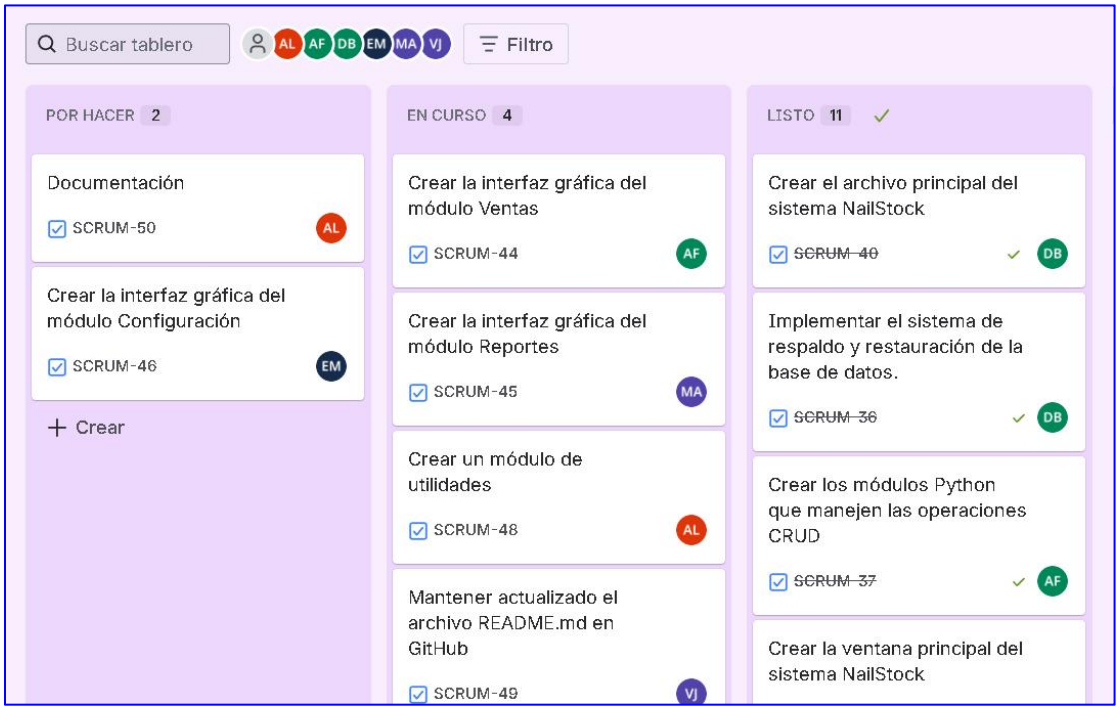
User Story: Exportar datos CSV



Todas las User Stories están vinculadas a una o varias Tareas.

Backlog del proyecto

- Tableros SCRUM con columnas: *To Do*, *In Progress*, Testing, *Done*.



Asignación de Roles

***PO, Scrum Master:** *Barrera López Alejandro [https://github.com/vakdev]*

Desarrollador: *Cruz Flores Arleth Leilani [https://github.com/ArlethC007]*

Desarrollador: *Estrada Mendoza Esteban Uriel [https://github.com/es377161-design]*

Desarrollador: *Corona Beltrán Diego Alessandro [https://github.com/Alecor0]*

Desarrollador: *García Acosta Moises [https://github.com/Moisssssss381]*

Tester: *Ángeles Juárez Valentín [https://github.com/Va1exhh]*

**Debido al tamaño del equipo, el Product Owner también desempeña funciones de Scrum Master y aporta activamente a la codificación.*

Backlog del proyecto

Tareas Principales

Barrera López Alejandro (PO, SM):

- Planificación del proyecto
- Estructura del proyecto
- Documentación
- Creación de Módulo de Utilidades
- Creación de Interfaz Gráfica de Productos
- Creación de Interfaz Gráfica de Clientes
- Creación de Interfaz Gráfica de Proveedores
- Creación de Controladores (para el manejo de BD mediante CRUD models)
- Templates

Cruz Flores Arleth Leilani (Dev):

- Creación de los módulos Python que manejen operaciones CRUD:
  - Proveedor Model
  - Cliente Model
  - Producto Model
  - Venta Model
- Creación de Interfaz Gráfica de Ventas

Estrada Mendoza Esteban Uriel (Dev):

- Creación de la Base de Datos SQLite
- Creación de la Clase Manejadora de Base de Datos
- Creación de la Interfaz Gráfica de Configuración

Corona Beltrán Diego Alessandro (Dev):

- Implementación del Sistema de Respaldo de la Base de Datos
- Creación del archivo principal (iniciador)

García Acosta Moises (Dev):

- Creación de la ventana principal de NailStock
- Creación de la Interfaz Gráfica de Reportes
- Templates

Ángeles Juárez Valentín (Dev / Tester):

- Actualización constante de README.md en el repositorio
- Creación de la Interfaz Gráfica de Inicio de Sesión
- Pruebas con el Software

Repositorio de Código

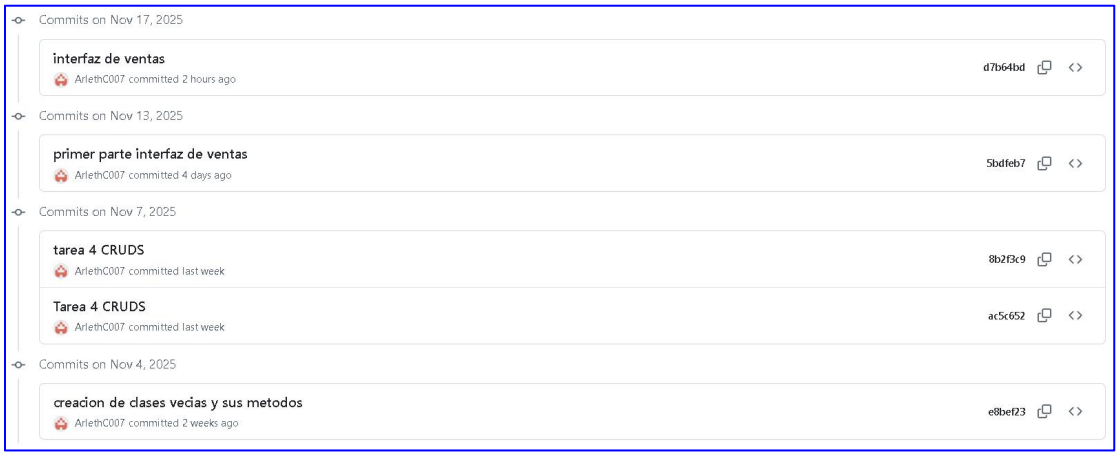
Repositorio Principal: https://github.com/Ismael-Software/proyecto-final-equipoverde
Forks: https://github.com/Ismael-Software/proyecto-final-equipoverde/forks

Nota: Debido a la "migración" que hubo del código a otro repositorio de GitHub-Classroom, no aparecen registrados algunos commits.

Historial General de Commits

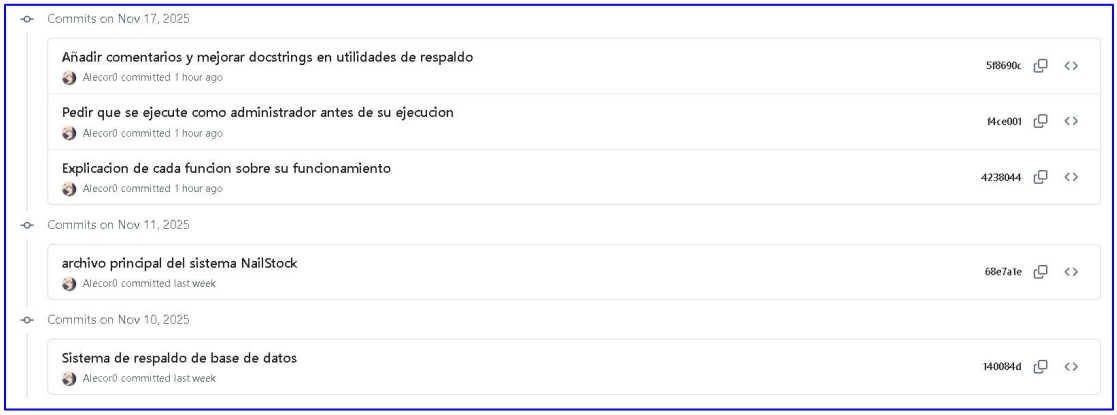
Estrada Mendoza Esteban Uriel:

Cruz Flores Arleth Leilani:



Repositorio de Código

Corona Beltrán Diego Alessandro:



García Acosta Moises:

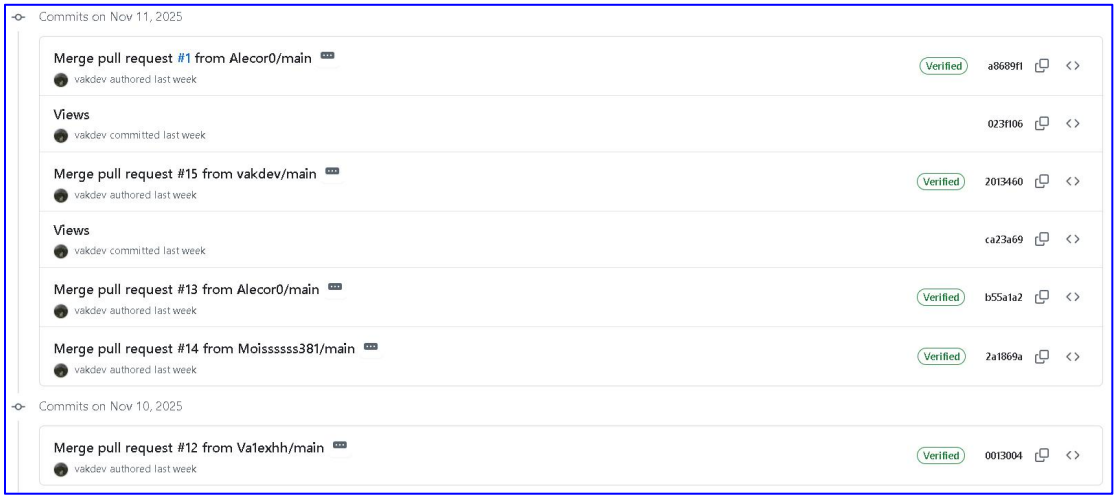
Repositorio de Código

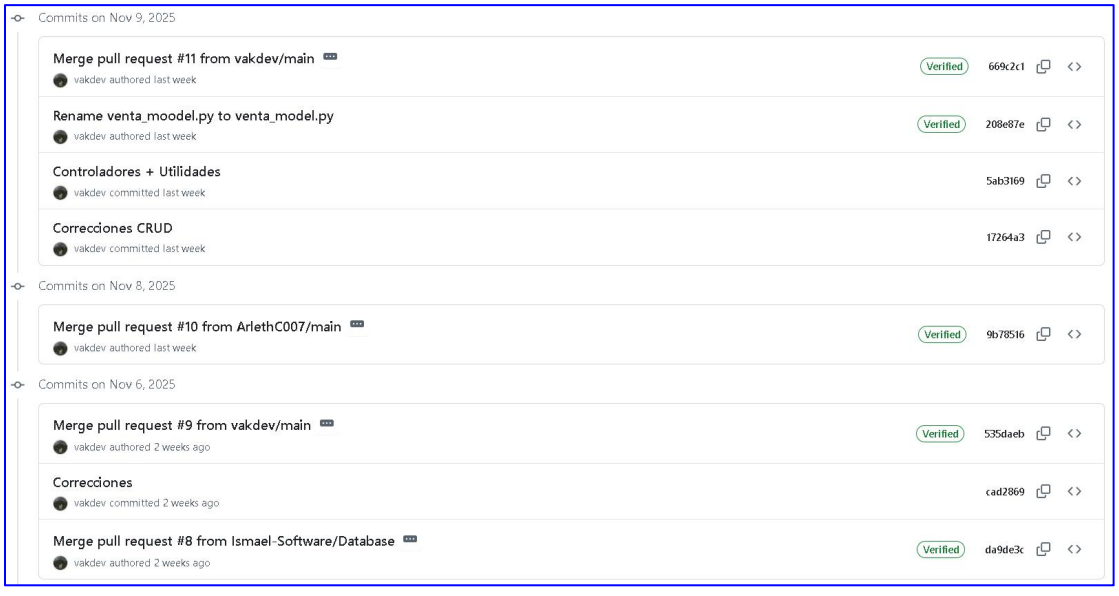
Ángeles Juárez Valentín:

Barrera López Alejandro:

Repositorio de Código

Barrera López Alejandro:





Repositorio de Código

Barrera López Alejandro:

Nailstock: Versión 1.0

Por ahora, el desarrollo de NailStock cubre los requerimentos principales de gestión para una ferretería, sin embargo, hay funcionalidades que se omitieron debido a restricciones de tiempo.

- Sistema completo de Usuarios y Roles
- Manejo de excepciones y errores
- Validaciones avanzadas
- Módulo de facturación electrónica
- Sistema de reportes gráficos

entre otras...

Sin embargo, el sistema fue diseñado con una base escalable gracias a:

1. Patrón MVC Estricto
   a) Separación entre lógica de negocio (Controllers), datos (Models) e interfaz (Views)
   b) Facilita el mantenimiento y la adición de nuevos módulos

2. Modularización del Código
   a) Cada entidad (Productos, Clientes, Proveedores, Ventas) tiene su propia estructura
   b) Los nuevos módulos pueden seguir el mismo patrón establecido

3. Base de Datos Normalizada
   a) Esquema relacional con integridad referencial
   b) Triggers para reglas de negocio críticas (control de stock)
   c) Fácil extensión con nuevas tablas y relaciones

4. Abstracción de Capas
   a) Los controladores actúan como intermediarios, permitiendo cambiar lógica sin afectar vistas
   b) Los modelos encapsulan el acceso a datos, facilitando cambios en el motor de BD