



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	3
Características del Sistema	4
<i>Diagramas UML</i>	5
<i>Diagramas UML</i>	6
<i>Diagramas UML</i>	7
Diseño de Base de Datos	8
<i>Diseño de Base de Datos</i>	9
<i>Diseño de Base de Datos</i>	10
<i>Diseño de Base de Datos</i>	11
Operaciones CRUD Implementadas	12
I. CREATE	12
II. READ	12
<i>Operaciones CRUD Implementadas</i>	13
III. UPDATE	13
IV. DELETE	13
I. CREATE	13
II. READ	13
III. UPDATE	13
IV. DELETE	13
<i>Operaciones CRUD Implementadas</i>	14
<i>Operaciones CRUD Implementadas</i>	15
<i>Operaciones CRUD Implementadas</i>	16
<i>Operaciones CRUD Implementadas</i>	17
Backlog del proyecto	18
<i>Backlog del proyecto</i>	19
<i>Backlog del proyecto</i>	20
Repositorio de Código	21
<i>Repositorio de Código</i>	22
<i>Repositorio de Código</i>	23
<i>Repositorio de Código</i>	24
<i>Repositorio de Código</i>	25
Nailstock: Versión 1.0	26
1. Patrón MVC Estricto	26
2. Modularización del Código	26
3. Base de Datos Normalizada	26
4. Abstracción de Capas	26

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.

Objetivo 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

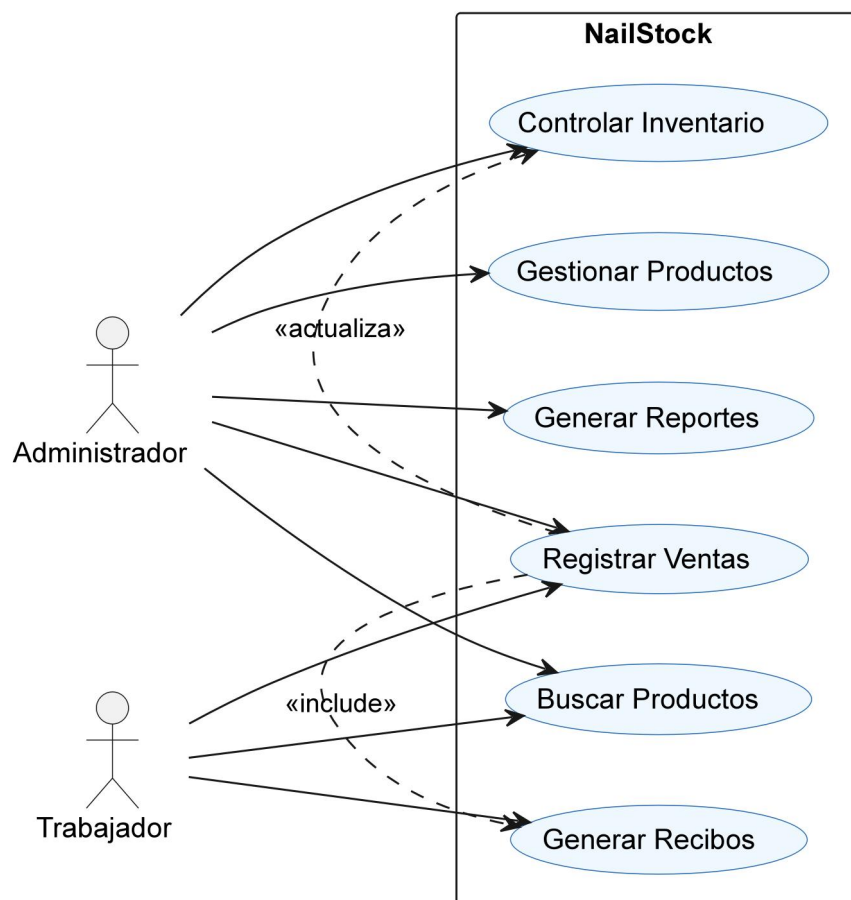
Casos de uso

Actores:

- Trabajador
- Administrador

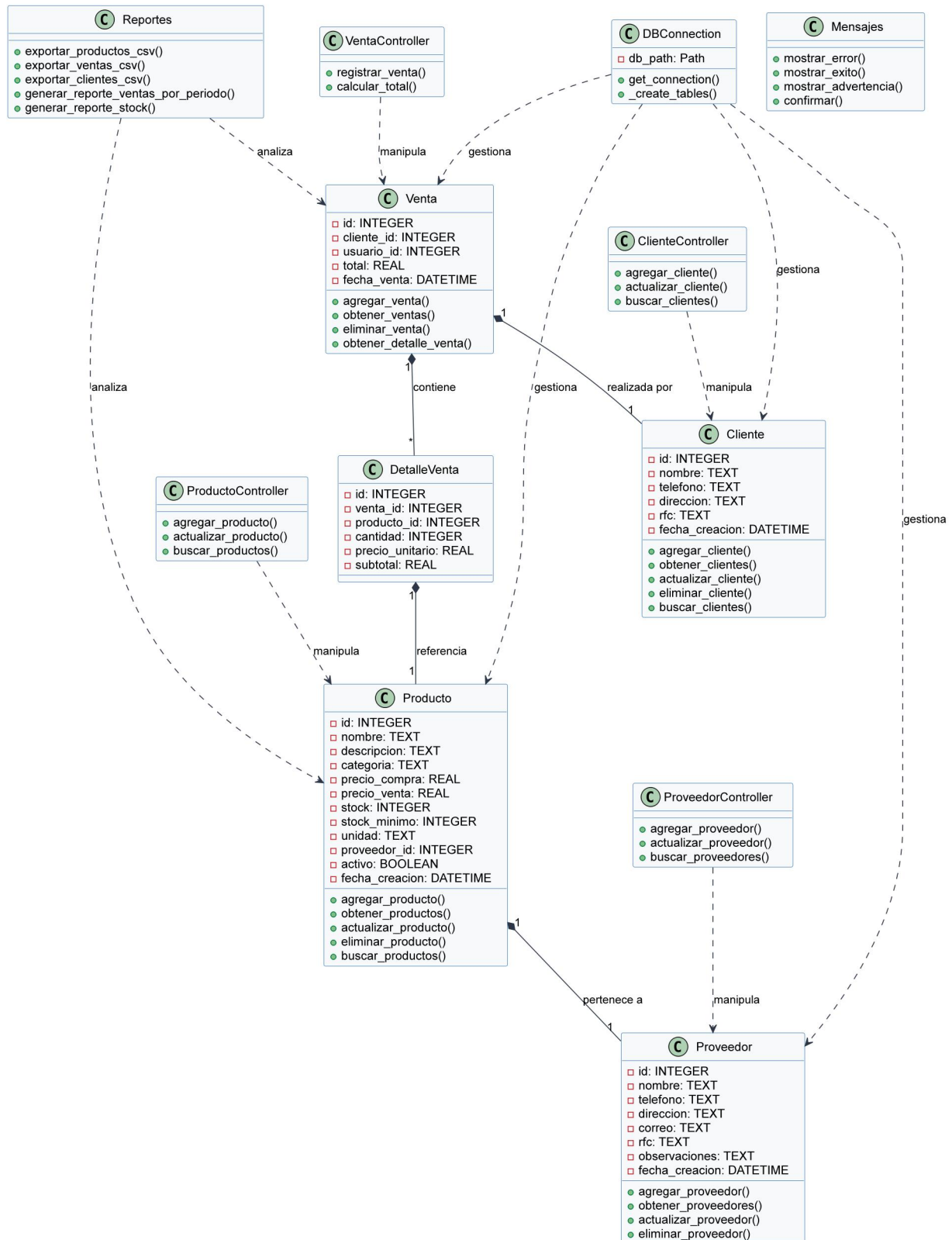
Casos de uso principales:

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



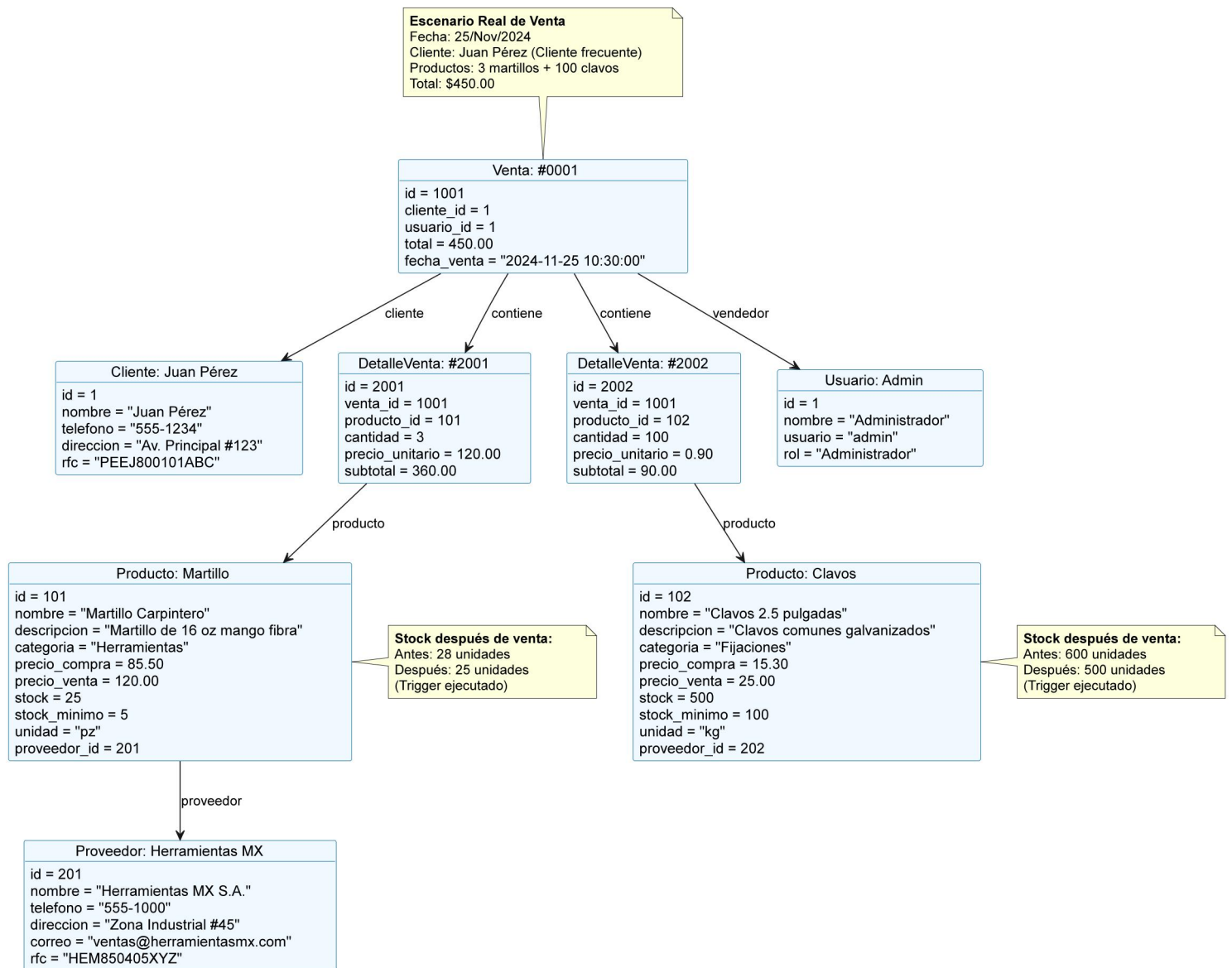
Diagramas UML

Diagrama de clases



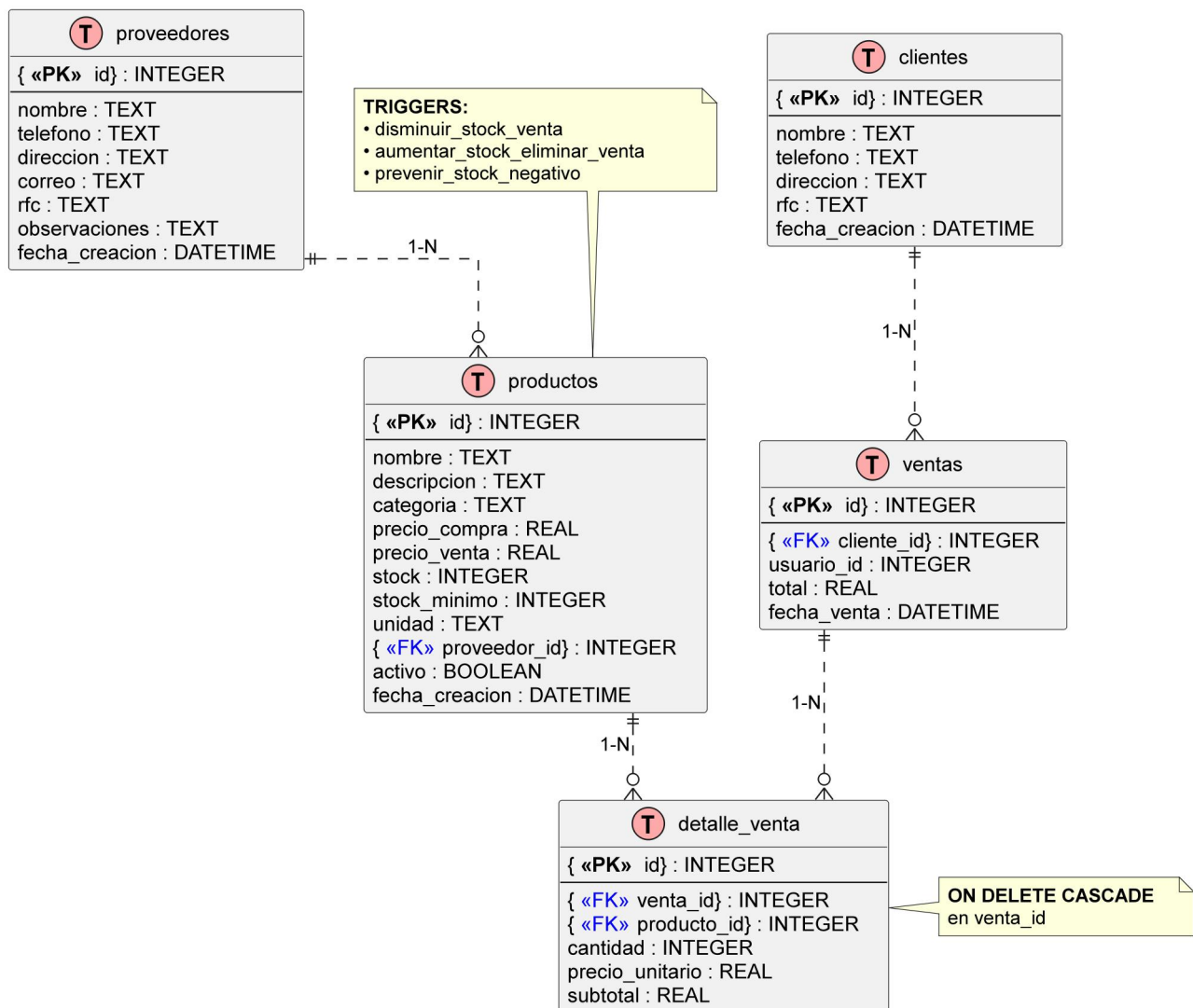
Diagramas UML

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

```
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

```
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

```
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

```
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)

```

1  from ..database.db_connection import get_db_connection
2
3  class ClienteModel:
4      @staticmethod
5      def agregar_cliente(nombre, telefono, direccion, rfc):
6          conn = get_db_connection()
7          cursor = conn.cursor()
8
9          cursor.execute('''
10             INSERT INTO clientes (nombre, telefono, direccion, rfc)
11             VALUES (?, ?, ?, ?)
12             ''', (nombre, telefono, direccion, rfc))
13
14             cliente_id = cursor.lastrowid
15             conn.commit()
16             conn.close()
17
18             return cliente_id
19
20     @staticmethod
21     def obtener_clientes():
22         conn = get_db_connection()
23         cursor = conn.cursor()
24
25         cursor.execute('SELECT * FROM clientes ORDER BY nombre')
26         clientes = cursor.fetchall()
27         conn.close()
28
29         return clientes
30
31     @staticmethod
32     def obtener_cliente_por_id(cliente_id):
33         conn = get_db_connection()
34         cursor = conn.cursor()
35
36         cursor.execute('SELECT * FROM clientes WHERE id = ?', (cliente_id,))
37         cliente = cursor.fetchone()
38         conn.close()
39
40         return cliente
41
42     @staticmethod
43     def actualizar_cliente(cliente_id, nombre, telefono, direccion, rfc):
44         conn = get_db_connection()
45         cursor = conn.cursor()
46
47         cursor.execute('''
48             UPDATE clientes
49             SET nombre = ?, telefono = ?, direccion = ?, rfc = ?
50             WHERE id = ?
51             ''', (nombre, telefono, direccion, rfc, cliente_id))
52
53         conn.commit()
54         conn.close()
55
56         return True
57
58     @staticmethod
59     def eliminar_cliente(cliente_id):
60         conn = get_db_connection()
61         cursor = conn.cursor()
62         cursor.execute('SELECT COUNT(*) FROM ventas WHERE cliente_id = ?', (cliente_id,))
63         count = cursor.fetchone()[0]
64
65         if count > 0:
66             raise Exception("No se puede eliminar el cliente porque tiene ventas asociadas")
67
68         cursor.execute('DELETE FROM clientes WHERE id = ?', (cliente_id,))
69         conn.commit()
70         conn.close()
71         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>)

ProductoModel

```

1  from ..database.db_connection import get_db_connection
2
3  class ProductoModel:
4      @staticmethod
5  >  def agregar_producto(nombre, descripcion, categoria, precio_compra, precio_venta, ...
23
24      @staticmethod
25  >  def obtener_productos(activo=True):...
42
43      @staticmethod
44  >  def obtener_producto_por_id(producto_id):...
60
61      @staticmethod
62  >  def actualizar_producto(producto_id, nombre, descripcion, categoria, precio_compra, ...
80
81      @staticmethod
82  >  def eliminar_producto(producto_id):...
93
94      @staticmethod
95  >  def buscar_productos(termino):...
112

```

ProveedorModel

```

1  from ..database.db_connection import get_db_connection
2
3  class ProveedorModel:
4      @staticmethod
5  >  def agregar_proveedor(nombre, telefono, direccion, correo, rfc, observaciones):...
19
20      @staticmethod
21  >  def obtener_proveedores():...
30
31      @staticmethod
32  >  def obtener_proveedor_por_id(proveedor_id):...
41
42      @staticmethod
43  >  def actualizar_proveedor(proveedor_id, nombre, telefono, direccion, correo, rfc, observaciones):...
57
58      @staticmethod
59  >  def eliminar_proveedor(proveedor_id):...

```

VentaModel

```

1  from ..database.db_connection import get_db_connection
2
3  class VentaModel:
4      @staticmethod
5  >  def agregar_venta(cliente_id, usuario_id, productos):...
38
39      @staticmethod
40  >  def obtener_ventas(fecha_inicio=None, fecha_fin=None):...
64
65      @staticmethod
66  >  def obtener_detalle_venta(venta_id):...
81
82      @staticmethod
83  >  def eliminar_venta(venta_id):...
98

```


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*:

```
1 from ..models.producto_model import ProductoModel
2
3 class ProductoController:
4     @staticmethod
5     def agregar_producto(nombre, descripcion, categoria, precio_compra, precio_venta,
6                           stock, stock_minimo, unidad, proveedor_id):
7         return ProductoModel.agregar_producto(
8             nombre, descripcion, categoria, precio_compra, precio_venta,
9             stock, stock_minimo, unidad, proveedor_id
10        )
11
12    @staticmethod
13    def actualizar_producto(producto_id, nombre, descripcion, categoria, precio_compra,
14                            precio_venta, stock, stock_minimo, unidad, proveedor_id):
15        return ProductoModel.actualizar_producto(
16            producto_id, nombre, descripcion, categoria, precio_compra, precio_venta,
17            stock, stock_minimo, unidad, proveedor_id
18        )
19
```

Ejemplo con *VentaController*:

```
1 from ..models.venta_model import VentaModel
2
3 class VentaController:
4     @staticmethod
5     def registrar_venta(cliente_id, usuario_id, productos):
6         return VentaModel.agregar_venta(cliente_id, usuario_id, productos)
7
8     @staticmethod
9     def calcular_total(productos):
10        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=eyJpIjoiNWlyOTUxMzA2NDQ1NDg4Y2EyZjAyNTUxNjVjZmEyOGMiLCJwIjoiajJ9>]

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

Descripción
Como administrador, quiero crear respaldos manuales de la base de datos para prevenir pérdida de información.

Actividades vinculadas +

relates to



 SCRUM-72 GESTIÓN DE RESPALDOS Y CONFIGURACIÓN FINALIZADA  =

User Story: Exportar datos CSV

Descripción
Como usuario, quiero exportar los datos de productos, clientes o ventas a archivos CSV para respaldar o analizar información externa.

Actividades vinculadas +

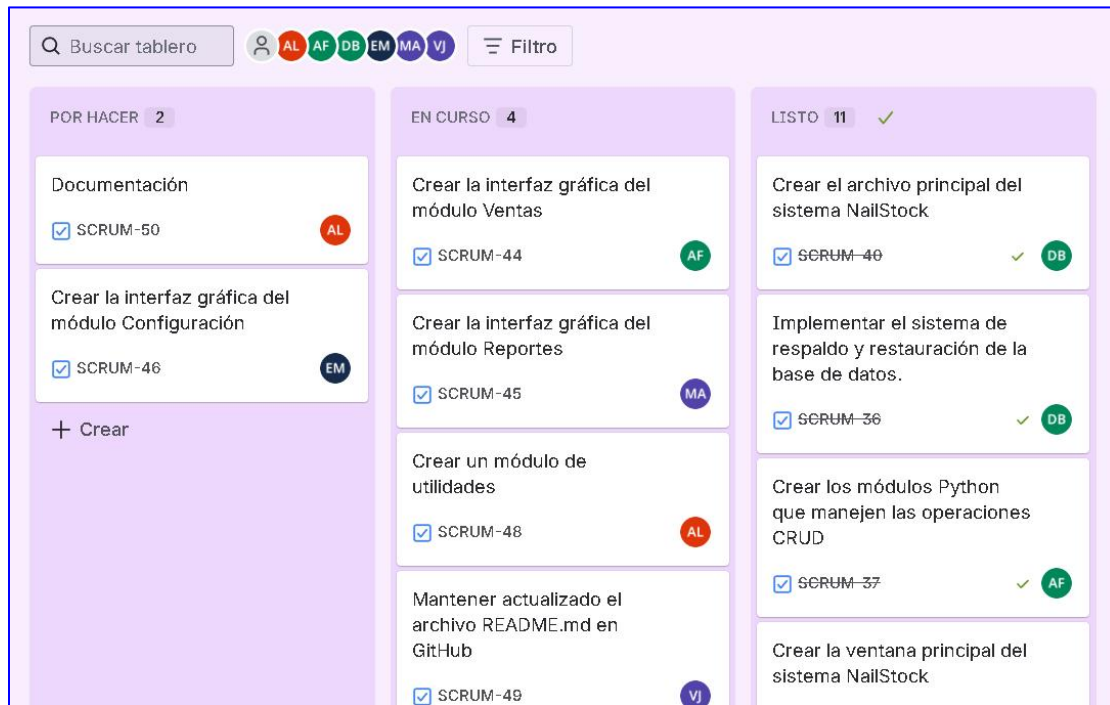
relates to

 SCRUM-69 GESTIÓN DE REPORTES FINALIZADA  =

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/Moissssss381>]

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:






Commits on Nov 17, 2025		
Explicación del código	es377161-design committed 5 hours ago	52b878a  
Commits on Nov 16, 2025		
interfaz grafica de configuracion	es377161-design committed yesterday	19185a9  
Commits on Nov 5, 2025		
db_conection	es377161-design committed 2 weeks ago	4f9516f  
naja	es377161-design committed 2 weeks ago	c802737  
nada	es377161-design committed 2 weeks ago	36c9531  

Cruz Flores Arleth Leilani:






Commits on Nov 17, 2025		
interfaz de ventas	ArlethC007 committed 2 hours ago	d7b64bd  
Commits on Nov 13, 2025		
primer parte interfaz de ventas	ArlethC007 committed 4 days ago	5bd1eb7  
Commits on Nov 7, 2025		
tarea 4 CRUDS	ArlethC007 committed last week	8b2f3c9  
Tarea 4 CRUDS	ArlethC007 committed last week	ac5c652  
Commits on Nov 4, 2025		
creacion de clases viejas y sus metodos	ArlethC007 committed 2 weeks ago	e8bef23  

Repositorio de Código

Corona Beltrán Diego Alessandro:

Commits on Nov 17, 2025		
Añadir comentarios y mejorar docstrings en utilidades de respaldo	Alecor0 committed 1 hour ago	5f8690c  <>
Pedir que se ejecute como administrador antes de su ejecucion	Alecor0 committed 1 hour ago	f4ce001  <>
Explicacion de cada funcion sobre su funcionamiento	Alecor0 committed 1 hour ago	4238044  <>
Commits on Nov 11, 2025		
archivo principal del sistema NailStock	Alecor0 committed last week	68e7a1e  <>
Commits on Nov 10, 2025		
Sistema de respaldo de base de datos	Alecor0 committed last week	140084d  <>

García Acosta Moises:

Commits on Nov 17, 2025		
Proposisto de las funciones de reportes	Moissssss381 committed 4 hours ago	ee155e6  <>
Proposisto de las funciones	Moissssss381 committed 4 hours ago	3010361  <>
Commits on Nov 12, 2025		
Actualize el titulo, tenia otro jeje	Moissssss381 committed 5 days ago	6fd051a  <>
Creacion modulo de reportes	Moissssss381 committed 5 days ago	56e195a  <>
Commits on Nov 10, 2025		
Ventana principal	Moissssss381 committed last week	522d101  <>

Repositorio de Código

Ángeles Juárez Valentín:

Commits on Nov 15, 2025		
Implementación Banner animado	0b0e0eb	<>
● Va1exhh committed 2 days ago		
Corrección	6da6db1	<>
● Va1exhh committed 2 days ago		
Modificación del README	b18cadf	<>
● Va1exhh committed 2 days ago		
Commits on Nov 9, 2025		
Agrego interfaz de login	e799a7c	<>
● Va1exhh committed last week		
Commits on Nov 4, 2025		
Modificación	4c85bc2	<>
● Va1exhh committed 2 weeks ago		
Modifico el formato visual del README	549871e	<>
● Va1exhh committed 2 weeks ago		
Agrego README con logotipo de NailStock	cda357c	<>
● Va1exhh committed 2 weeks ago		
Commits on Nov 2, 2025		
primer commit	a7cb61b	<>
● Va1exhh committed 2 weeks ago		

Barrera López Alejandro:

Commits on Nov 17, 2025		
Merge pull request #6 from Alecor0/main	Verified 34d6c69	<>
● vakdev authored 9 minutes ago		
Merge pull request #5 from Moissssss381/main	Verified 32d3f7a	<>
● vakdev authored 2 hours ago		
Corrección	Verified a19b398	<>
● vakdev authored 2 hours ago		

Commits on Nov 16, 2025		
Corrección de path a .png	9468101	<>
● vakdev committed 18 hours ago		
Corrección Modularización + Compilador	fb55ce3	<>
● vakdev committed 18 hours ago		
Commits on Nov 15, 2025		
Merge pull request #4 from Va1exhh/main	Verified 79bbd4c	<>
● vakdev authored 2 days ago		
Commits on Nov 14, 2025		
Merge pull request #2 from Moissssss381/main	Verified d7f8712	<>
● vakdev authored 3 days ago		
Merge pull request #3 from ArlethC007/main	Verified 76e260a	<>
● vakdev authored 3 days ago		

Repositorio de Código

Barrera López Alejandro:

Commits on Nov 11, 2025		
Merge pull request #1 from Alecor0/main	Verified	a8689f1
vakdev authored last week		
Views		023f106
vakdev committed last week		
Merge pull request #15 from vakdev/main	Verified	2013460
vakdev authored last week		
Views		ca23a69
vakdev committed last week		
Merge pull request #13 from Alecor0/main	Verified	b55a1a2
vakdev authored last week		
Merge pull request #14 from Moissssss381/main	Verified	2a1869a
vakdev authored last week		
Commits on Nov 10, 2025		
Merge pull request #12 from Va1exhh/main	Verified	0013004
vakdev authored last week		

Commits on Nov 9, 2025		
Merge pull request #11 from vakdev/main	Verified	669c2c1
vakdev authored last week		
Rename venta_moodel.py to venta_model.py	Verified	208e87e
vakdev authored last week		
Controladores + Utilidades		5ab3169
vakdev committed last week		
Correcciones CRUD		17264a3
vakdev committed last week		
Commits on Nov 8, 2025		
Merge pull request #10 from ArlethC007/main	Verified	9b78516
vakdev authored last week		
Commits on Nov 6, 2025		
Merge pull request #9 from vakdev/main	Verified	535daeb
vakdev authored 2 weeks ago		
Correcciones		cad2869
vakdev committed 2 weeks ago		
Merge pull request #8 from Ismael-Software/Database	Verified	da9de3c
vakdev authored 2 weeks ago		

Commits on Nov 5, 2025		
Merge pull request #7 from vakdev/main	Verified	ccb0ee
vakdev authored 2 weeks ago		
Update image source path in README.md	Verified	bc0531e
vakdev authored 2 weeks ago		
Muevo assets y models dentro de nailstock		3008759
vakdev committed 2 weeks ago		
Merge branch 'Ismael-Software:main' into main	Verified	b089f11
vakdev authored 2 weeks ago		
Delete proveedor_model.py	Verified	8116e4f
vakdev authored 2 weeks ago		
Rename project to NailStock in README	Verified	8edbd2d8
vakdev authored 2 weeks ago		
Merge pull request #4 from Va1exhh/main	Verified	1e10843
vakdev authored 2 weeks ago		

Repositorio de Código

Barrera López Alejandro:

Commits on Nov 4, 2025		
Merge pull request #6 from ArlethC007/main	Verified	206cc0a
vakdev authored 2 weeks ago		
Add static methods to VentaModel class	Verified	c3ec229
vakdev authored 2 weeks ago		
Refactor ProductModel to use static methods	Verified	470b2d0
vakdev authored 2 weeks ago		
Convert ClienteModel methods to static methods	Verified	08c6ce0
vakdev authored 2 weeks ago		
Refactor proveedorModel to ProveedorModel with static methods	Verified	903953e
vakdev authored 2 weeks ago		
Creadón de clases vacías		7c10972
vakdev committed 2 weeks ago		
Commits on Nov 2, 2025		
Delete test.py	Verified	fb4d085
vakdev authored 2 weeks ago		
Merge pull request #1 from zetalevk/main	Verified	032cf09
vakdev authored 2 weeks ago		

Nailstock: Versión 1.0

Por ahora, el desarrollo de NailStock cubre los requerimientos 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