

**Asignatura:** Capstone (PTY4614\_002D)

**Título del Proyecto:** Mejoras a sistema de caja FastPOS para pizzerías

**Integrantes:** Fernanda Monsalve Luco – Matías Gutiérrez

**Semana de entrega:** Semana 9

[**Abstract (Español) 3**](#_heading=h.rqytugsadni4)

[**Abstract (English) 3**](#_heading=h.snhv7df0jroz)

[**Ajustes al Proyecto APT 3**](#_heading=h.55wgs9ro5b9z)

[Durante el desarrollo del proyecto, se identificaron las siguientes dificultades: 3](#_heading=h.38w4wcksa9p6)

[Facilitadores: 4](#_heading=h.dsgrk55fbdmt)

[Ajustes realizados: 4](#_heading=h.iru1no3tbg69)

[**Metodología de Trabajo 4**](#_heading=h.xwfya6spbe5w)

[Se mantiene la metodología Scrum, con las siguientes prácticas: 4](#_heading=h.tflf8ax65l4w)

[Esta metodología permite: 5](#_heading=h.yyutkitnyybc)

[**Evidencias de Avance 5**](#_heading=h.kh286uixe737)

[**Herramientas y Métodos 11**](#_heading=h.44qz9vmjtswp)

[**Responsables 12**](#_heading=h.edlfhwfto3hd)

[**Anexos 12**](#_heading=h.52ktvvr1s9a6)

# Abstract (Español)

Este informe presenta el avance del Proyecto APT “Mejoras al sistema de caja FastPOS para pizzerías”, desarrollado durante las semanas 1 a 9. Se describen los ajustes realizados a la propuesta inicial en respuesta a dificultades técnicas, retroalimentación docente y facilitadores del entorno de desarrollo. Se detalla la metodología Scrum aplicada, con sprints semanales, y se presentan evidencias concretas de avance en documentación (prototipos en Figma), programación (módulos de autenticación y caja en React/Node.js) y almacenamiento de datos (modelo relacional implementado en Firebase/Firestore). El proyecto avanza conforme a los estándares de la industria y las competencias del perfil de egreso en Ingeniería en Informática.

# Abstract (English)

This report presents the progress of the APT Project “Enhancements to the FastPOS cash register system for pizzerias,” developed between weeks 1 and 9. It outlines adjustments made to the initial proposal in response to technical challenges, instructor feedback, and enabling factors in the development environment. The Scrum methodology is detailed, with weekly sprints, and concrete evidence of progress is provided in documentation (Figma prototypes), programming (authentication and cashier modules in React/Node.js), and data storage (relational model implemented in Firebase/Firestore). The project advances in alignment with industry standards and the Informatics Engineering graduate profile competencies.

# Ajustes al Proyecto APT

## Durante el desarrollo del proyecto, se identificaron las siguientes dificultades:

* Integración entre frontend y backend: inicialmente se planificó una API REST con Node.js y una base de datos SQL, pero ante la complejidad de despliegue local y pruebas en equipo, se optó por migrar a Firebase (Firestore + Authentication), lo que simplificó la autenticación con roles y la sincronización en tiempo real.
* Retroalimentación docente: se sugirió priorizar la validación de cálculos de impuestos y descuentos antes de avanzar a reportes avanzados, para asegurar la integridad del módulo de caja.
* Disponibilidad de tiempo: uno de los integrantes tuvo compromisos académicos externos en la semana 5, lo que generó un ligero retraso en el módulo de inventario.

## Facilitadores:

* Dominio previo de React y Firebase por parte del equipo.
* Uso de herramientas colaborativas (GitHub, Figma, Trello) que permitieron un trabajo asíncrono eficiente.
* Retroalimentación semanal del docente, que permitió corregir rutas de desarrollo a tiempo.

## Ajustes realizados:

* Cambio de stack tecnológico: de SQL + Express a Firebase (Firestore + Auth).
* Planificación del sprint 5: el módulo de inventario se retrasó una semana, y se prioriza la estabilidad del módulo de caja.
* Ampliación del plan de pruebas: se incluyeron casos específicos para validación de IVA y descuentos, como solicitado en la retroalimentación.

# Metodología de Trabajo

## Se mantiene la metodología Scrum, con las siguientes prácticas:

* Sprints de 1 semana (semanas 3–9).
* Daily Scrums virtuales (3 veces por semana vía Discord).
* Backlog en Trello, con tarjetas asignadas a cada integrante.
* Sprint Review al final de cada semana (demo funcional en Firebase Hosting).
* Sprint Retrospective para ajustar prioridades.

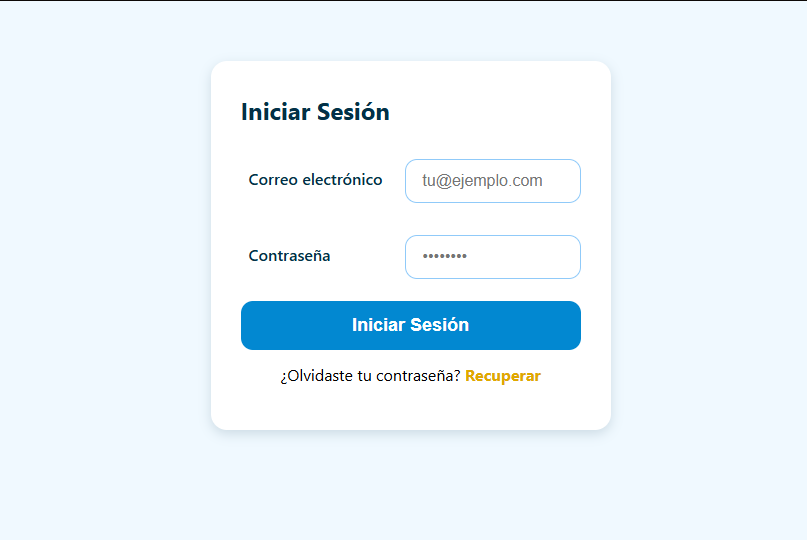
## Esta metodología permite:

* Entregas incrementales y medibles.
* Adaptación rápida a cambios técnicos o de alcance.
* Visibilidad del progreso para el docente y el equipo.
* Está alineada con el estándar industrial para proyectos de software ágiles y con la competencia del perfil de egreso: “Gestionar proyectos informáticos, ofreciendo alternativas para la toma de decisiones” (IL 2.2).

# Evidencias de Avance

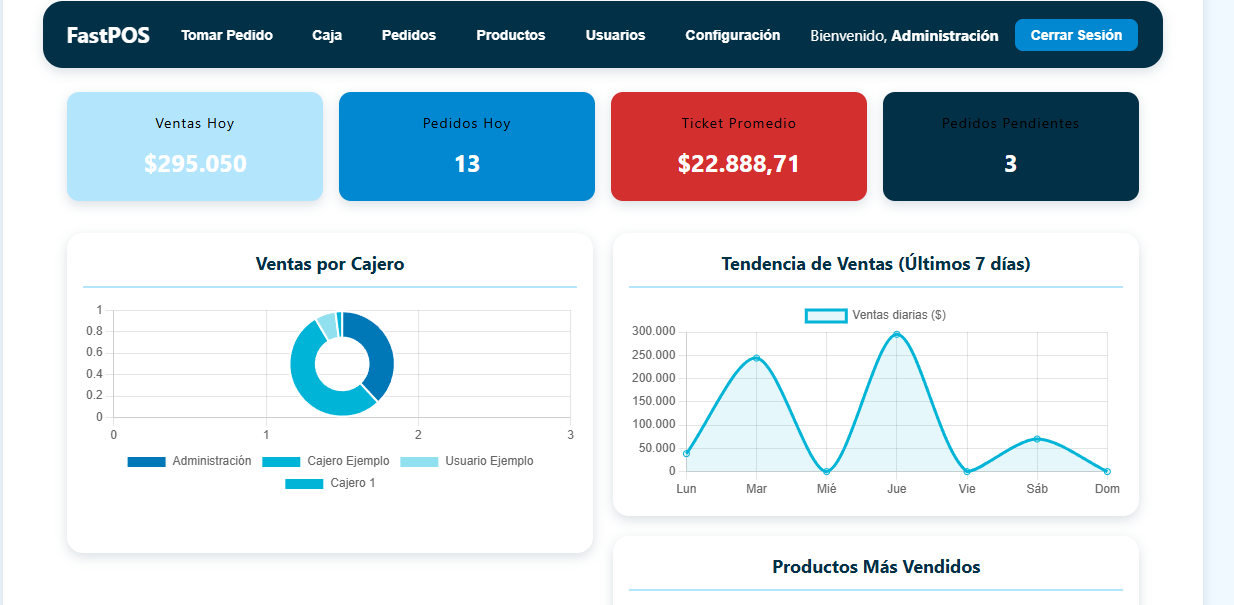
## Documentación

* 1. Prototipos actualizados en Figma: pantallas de login, caja, toma de pedidos y administración.









* 1. Plan de pruebas: incluye casos para cálculo de IVA, descuentos y actualización de inventario.

Este plan cumple con buenas prácticas de la industria (ISO/IEC/IEEE 29119).

### Plan de Pruebas Funcionales – FastPOS

* Versión: 2.0
* Fecha: 09/10/2025
* Elaborado por: Fernanda Monsalve – Matías Gutiérrez
* Proyecto: Mejoras a sistema de caja FastPOS para pizzerías
* Asignatura:Capstone (PTY4614\_002D)

Objetivo: Validar que las funcionalidades críticas del sistema —específicamente el cálculo correcto de IVA (19%), la aplicación de descuentos y la actualización automática del inventario al registrar o cancelar pedidos— operen de acuerdo con los requisitos definidos y sin errores que afecten la integridad del negocio.

1. Alcance

Este plan cubre las siguientes funcionalidades:

* Cálculo automático de IVA sobre el subtotal del pedido.
* Aplicación de descuentos por promoción o rol (ej. “2x1”, “10% de descuento”).
* Actualización en tiempo real del inventario al:
* Agregar un producto a un pedido.
* Cancelar o eliminar un pedido (reversión de inventario).

No incluye: pruebas de rendimiento, seguridad avanzada ni integración con pasarelas de pago reales.

### Casos de Prueba

| **ID** | **Descripción** | **Precondiciones** | **Pasos** | **Resultado esperado** | **Resultado obtenido** | **Estado** |
| --- | --- | --- | --- | --- | --- | --- |
| CP-01 | Validar cálculo de IVA (19%) sobre subtotal | - Sistema iniciado  - Producto “Pizza Margarita” ($12.000) en catálogo | 1. Iniciar sesión como caja  2. Agregar 1 Pizza Margarita al pedido  3. Ver total | Subtotal: $12.000  IVA (19%): $2.280  Total: $14.280 | Subtotal:$12.000  IVA (19%)$2280  Total:$14.280 | Success |
| CP-02 | Validar IVA con múltiples productos | - Sistema iniciado  - 2 productos en catálogo: Pizza ($12.000 y $12.500), Bebida ($2.000 y $5.000) | 1. Agregar 2 Pizzas y 3 Bebidas  2. Ver total | Subtotal: $33.500  IVA: $6365  Total: $39865 | Subtotal:$33.500  IVA (19%)$6365  Total:$39.865 | Success |
| CP-03 | Aplicar descuento del 10% sobre total | - Pedido con total $33.500 (CP-01) | 1. Aplicar descuento “10%” desde interfaz de caja  2. Ver total final | Total antes descuento: $33.500  Descuento: -$3350  Total final: $30.150 | Total antes del descuento:$33.500  Descuento (10%):-$3350  **Total final:$30.150** | Success |
| CP-04 | Validar que el descuento no afecte el cálculo del IVA | - Mismo contexto que CP-03 | Revisar desglose en interfaz | Base imponible:  $28.151  IVA (19%):  $5349  Total bruto:  $33.500  Descuento (10%):  -$3350  Total neto:  $30.150 | Base imponible:$28.151  IVA (19%):$5349  **Total bruto:$33.500**  Descuento (10%):-$3350  **Total neto:$30.150** | Success |
| CP-05 | Actualización de inventario al agregar pedido | - Inventario inicial “Pizza Napolitana”: “tomateSan Manzano” = 40 unidades  - Cada pizza consume 1 unidad | 1. Registrar pedido con 1 pizzas  2. Consultar inventario | “tomate San Manzano” = 39 unidades | cantidad  39  (número)  nombre  "Tomates San Marzano" | Success |
| CP-06 | Reversión de inventario al cancelar pedido | - Pedido con 2 pizzas registrado (inventario = 48)  - Pedido en estado “pendiente” | 1. Cancelar pedido desde interfaz de caja  2. Consultar inventario | “aceite de oliva” vuelve a 40 unidades | **“aceite de oliva” vuelve a 40 unidades** | cumple |
| CP-07 | No permitir venta si inventario insuficiente | - “Masa pizza” = 1 unidad  - Pedido requiere 2 unidades | Intentar registrar pedido con 2 pizzas | Sistema muestra alerta: “Inventario insuficiente” y no permite confirmar pedido | Sistema muestra: Inventario insuficiente: -2 unidades” y no permite confirmar el pedido | modificado |

* Notas:
  + Todos los precios están en CLP.
  + El IVA siempre se calcula sobre el subtotal antes de los descuentos, según la normativa chilena.
  + Los descuentos se aplican al total con IVA, salvo que se especifique lo contrario.

# Herramientas y Métodos

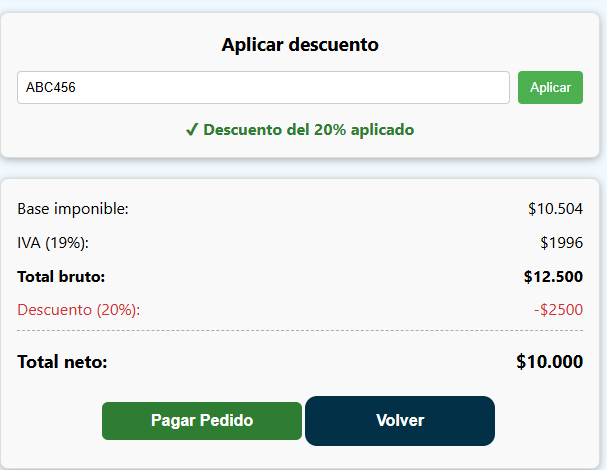
* Entorno de prueba: Firebase Hosting + Firestore (entorno de desarrollo).
* Método: Pruebas manuales funcionales + validación visual en UI.
* Registro de resultados: Se completará la columna “Resultado Obtenido” y “Estado” (Success/Error) durante la ejecución en sprint 6.
* Criterio de aceptación: Todos los casos deben pasar (estado: Success) para considerar el módulo de caja listo para entrega final.

# Responsables

* Diseño de pruebas: Fernanda Monsalve
* Ejecución de pruebas: Matías Gutiérrez y Fernanda Monsalve (pruebas cruzadas)
* Corrección de errores: Matías Gutiérrez (backend), Fernanda Monsalve (frontend)

# Anexos

* Captura de pantalla del desglose de IVA/descuento en UI.



* Enlace a colección de Firestore con datos de prueba.

const admin = require('firebase-admin');

const serviceAccount = require('./serviceAccountKey.json');

admin.initializeApp({

credential: admin.credential.cert(serviceAccount),

projectId: 'fastpos-b9692',

});

const db = admin.firestore();

// CREAR CLIENTES

async function createClients() {

const clientes = [

{ email: 'carlos.lopez@email.com', nombre: 'Carlos López' },

{ email: 'ana.martinez@email.com', nombre: 'Ana Martínez' },

{ email: 'luis.rodriguez@email.com', nombre: 'Luis Rodríguez' },

];

for (const c of clientes) {

await db.collection('Clientes').add({

...c,

createdAt: admin.firestore.FieldValue.serverTimestamp()

});

}

console.log('Success: Clientes creados');

}

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

// === USUARIOS ===

match /Usuarios/{userId} {

allow read: if request.auth != null;

allow write: if request.auth != null && (request.auth.uid == userId || isAdmin());

}

// === PRODUCTOS ===

match /Productos/{productoId} {

allow read: if request.auth != null;

allow create, update, delete: if isAdmin();

}

// === CATEGORÍAS ===

match /categorias/{categoriaId} {

allow read: if request.auth != null;

allow create, update, delete: if isAdmin();

}

// === VENTAS ===

match /ventas/{ventaId} {

allow create: if request.auth != null &&

request.resource.data.cajeroId == request.auth.uid;

allow read, update, delete: if request.auth != null;

}

// === ÓRDENES ===

match /ordenes/{ordenId} {

allow read, create, update: if request.auth != null &&

(resource == null || resource.data.usuarioId == request.auth.uid);

allow delete: if isAdmin() ||

(request.auth != null && resource.data.usuarioId == request.auth.uid);

}

// === CLIENTES ===

match /clientes/{clienteId} {

allow read, create, update: if request.auth != null;

allow delete: if isAdmin();

}

// === CONFIGURACIONES ===

match /config/{configId} {

allow read, write: if isAdmin();

}

// === FUNCIÓN AUXILIAR ===

function isAdmin() {

return request.auth != null &&

exists(/databases/$(database)/documents/Usuarios/$(request.auth.uid)) &&

get(/databases/$(database)/documents/Usuarios/$(request.auth.uid)).data.rol == 'admin';

}

// === BLOQUEO POR DEFECTO ===

match /{document=\*\*} {

allow read, write: if request.auth != null;

}

}

}

## Programación

* Módulo de autenticación: roles (mesero, caja, admin) implementados con Firebase Auth.
  + 📁 src/components/Auth/Recuperar.js
* Módulo de caja funcional: permite agregar productos, calcular subtotal, IVA (19%) y total.
  + 📁 src/components/Caja.js
* Conexión a Firestore: lectura/escritura de pedidos y productos.
  + 📁 src/components/firebase.js

## Almacenamiento de datos

* Modelo de datos en Firestore: colecciones Categorias, Clientes, Productos, Usuarios, config, inventario, ventas.
* Reglas de seguridad: acceso restringido por rol (ej: solo admin puede editar precios).
  + 📁 firestore.rules
* Todas las evidencias están versionadas en el repositorio Git del equipo, con commits descriptivos y fechas alineadas al cronograma.
  + https://github.com/FernandaMonsalveLuco/fastpos-desktop

# Cumplimiento de Competencias del Perfil de Egreso

| **Competencia** | **Indicador aplicado** | **Evidencia** |
| --- | --- | --- |
| Desarrollar solución de software | 4.1: Construye solución con técnicas sistematizadas | Uso de componentes reutilizables en React, separación frontend/backend |
| Construir modelos de datos | 3.2: Implementa modelo escalable | Firestore con estructura flexible y reglas de seguridad |
| Realizar pruebas de certificación | 1.2: Aplica pruebas de validación | Plan de pruebas ejecutado en módulo de caja |
| Gestionar proyectos informáticos | 2.2: Controla proyecto con alternativas | Planificación de sprint 5 sin afectar entrega final |

# Aspectos Formales y Repositorio Git

* Lenguaje técnico preciso: se utilizan términos como Firestore, autenticación basada en roles, módulo de caja, IVA, componente reutilizable, etc.
* Ortografía y redacción: revisado con corrector; sin errores.
* Estructura del informe: sigue el formato solicitado (portada, índice, abstract, desarrollo, conclusiones en inglés).
* Repositorio Git: https://github.com/FernandaMonsalveLuco/fastpos-desktop
* Carpetas organizadas: /docs, /src, /auth, /usuarios, /main
* Commits atribuidos: Fernanda (UI, pruebas), Matías (backend, base de datos)
* Nombres de archivos según estándar: authService.js, Caja.js, modeloDatosFirestore.png

# Trabajo en Equipo

Ambos integrantes:

* Participan activamente en reuniones y toma de decisiones.
* Pueden explicar cualquier componente del sistema.
* Han realizado aportes equitativos, visibles en el historial de commits y en la asignación de tareas en Trello.

# Conclusiones Individuales (en inglés)

### Fernanda Monsalve

This phase reinforced my ability to adapt technical decisions based on real constraints and feedback. Transitioning to Firebase improved our development speed and allowed us to focus on core functionalities like order management and tax calculation. I’ve deepened my understanding of agile practices and the importance of user-centered design, which aligns with my professional goal of building efficient business software.

### Matías Gutiérrez

Working on the backend integration taught me how to balance flexibility and structure in data modeling. Migrating to Firestore required rethinking our initial relational approach, but it ultimately made the system more scalable for a small business context. I also improved my collaboration skills by coordinating closely with Fernanda on UI-backend alignment, ensuring a seamless user experience.

# Reflexión (en inglés)

This project has been a powerful bridge between academic theory and real-world software development. We learned that technical decisions must be guided not only by best practices but also by project constraints and stakeholder feedback. The shift to Firebase, though unplanned, turned into a strategic advantage. More importantly, we discovered that clear communication, version control discipline, and iterative testing are as critical as coding skills. This experience has solidified our confidence in delivering professional-grade software solutions.

# Anexos

Captura de pantalla del tablero de Trello

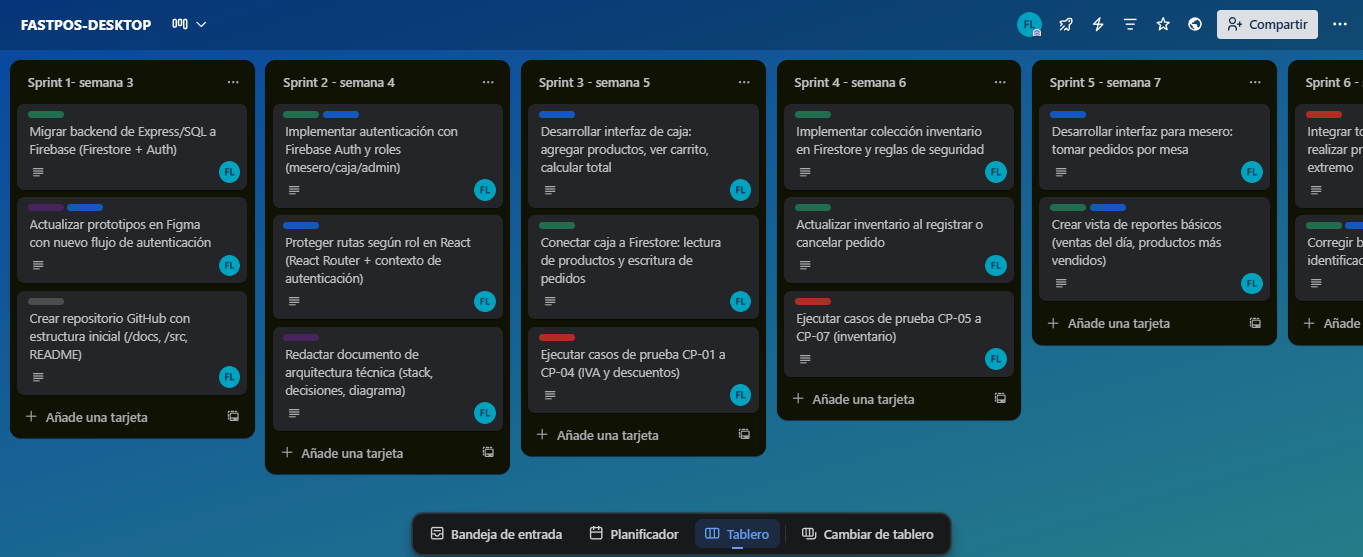


Diagrama del modelo de datos en Firestore

* Colección Categorias (Almacena información sobre categoría de productos para posteriormente organizar la caja)
  + Documento: {categoriaId}
  + Campos: nombre (string) → valores: “acompañamientos”, “bebidas”, “platillos”, “postres”, “combos” .
* Colección: config (Guarda las preferencias del usuario)
  + Documento: {configId}
  + Campos: impuesto (number), modoImpresion (string) → valores: “ticket”, “factura”, moneda (string), nombreNegocio (string), notificaciones (boolean).
* Colección Usuarios (Almacena información de los usuarios del sistema)
  + Documento: {userId}
  + Campos: name (string), email (string), rol (string)→ valores: "mesero", "caja", "admin"; createdAt (timestamp), activo: (boolean).
  + Regla de seguridad: Solo usuarios autenticados pueden leer su propio documento. Solo admin puede crear/editar usuarios.
* Colección: Productos (Catálogo de productos: pizzas, bebidas, ingredientes, etc.)
  + Documento: {productId}
  + Campos: categoría (string), descripcion (string), insumos (array), nombre (string), precio (number), stock (number).
  + Regla de seguridad: Lectura pública para roles mesero y caja. Solo admin puede escribir.
* Colección: ventas (Pedidos registrados en el sistema)
  + Documento: {orderId}
  + Campos: cajeroId (string), cajeroNombre (string), estado (string) → valores: “completado”, “pendiente”, cancelado”, fecha (timeStamp), metodoPago (string) → valores: “efectivo”, “tarjeta”, “transferencia”, montoRecibido (number), productos (array) 0 (map), cantidad (number), id (string ), nombre (string), precio (number), subtotal (number), tipo (string), total (number), vuelto (number).
  + Actualización de inventario: Al crear un pedido con estado "pendiente", se reduce el stock en inventario. Al cancelar, se revierte.
* Colección: inventario (Control de stock por insumo o producto)
  + Documento: {inventarioId} → generalmente igual al productId
  + Campos: Cantidad (number), nombre (string).
  + Nota: Aunque Firestore no tiene relaciones formales, mantiene coherencia mediante triggers o lógica en el frontend/backend al registrar pedidos.
* Relaciones Lógicas (sin joins, pero con referencias)
  + Un usuario (Usuario) puede crear muchos pedidos (ventas).
  + Cada pedido contiene múltiples productos (Productos), referenciados por productId.
  + El stock de cada producto se gestiona en Inventario, actualizado en tiempo real al:
    - Crear un pedido → currentStock -= quantity
    - Cancelar un pedido → currentStock += quantity

[users] ────< CREA >─── [orders]

[products] ← CONTIENE ← [orders.items.productId]

[inventory] ↔ ACTUALIZA ↔ [orders]