# Backlog

**Épicas:**

**EPIC-01: Autenticación, Seguridad y Gestión de Usuarios**

**Descripción general:**  
Esta épica abarca todos los procesos y módulos relacionados con el acceso seguro al sistema, la administración de cuentas de usuario y la gestión de permisos granulares. Incluye la autenticación basada en roles (vendedor, administrador), la creación, edición y desactivación de usuarios, la asignación de permisos específicos por usuario, y la trazabilidad de todas las acciones críticas mediante auditoría. El objetivo es garantizar que solo usuarios autorizados accedan a las funcionalidades adecuadas, proteger la información sensible y mantener un registro inmutable de todas las operaciones relevantes.

**Objetivos de negocio:**

* Proteger el acceso y la información del sistema.
* Controlar granularmente las acciones permitidas por cada usuario.
* Facilitar la gestión y recuperación de cuentas.
* Mantener trazabilidad y cumplimiento normativo.

**Alcance funcional:**

* Inicio y cierre de sesión seguro (JWT, cookies seguras).
* Gestión de usuarios: crear, editar, desactivar (no eliminar físico).
* Asignación de roles (Administrador, Vendedor) y permisos individuales (aplicar descuentos, crear reservas, editar inventario, etc.).
* Visualización de contraseñas solo al crear usuario.
* Auditoría automática de todas las acciones críticas.
* Recuperación de contraseña (si se implementa en MVP).
* Dashboard diferenciado por rol y permisos.

**Límites y exclusiones:**

* No incluye integración con SSO o autenticación social.
* No contempla eliminación física de usuarios (solo desactivación).
* Notificaciones por Gmail solo si es crítico para el MVP.

**Dependencias y relaciones:**

* Todas las funcionalidades dependen de la autenticación y permisos.
* Relación directa con auditoría y seguridad transversal.

**Escenarios clave:**

* Un administrador crea un nuevo vendedor, asigna permisos y puede auditar todas sus acciones.
* Un usuario desactivado no puede acceder al sistema ni realizar operaciones.

**EPIC-02: Gestión Integral de Inventario y Productos para Mundo Cartas**

**Descripción general:**  
Esta épica abarca la administración completa y centralizada del inventario de Mundo Cartas, incluyendo cartas TCG (Pokémon, Yu-Gi-Oh, Magic, One Piece, Digimon), juegos de mesa y accesorios. El objetivo es garantizar un control preciso, ágil y trazable de todos los productos, desde su ingreso hasta su venta o ajuste, permitiendo a la empresa mantener la disponibilidad adecuada, reducir errores y optimizar la toma de decisiones comerciales. La gestión de inventario es el corazón operativo de Mundo Cartas, ya que impacta directamente en la satisfacción del cliente, la rentabilidad y la eficiencia del negocio.

**Objetivos de negocio:**

* Eliminar errores y pérdidas de stock causados por gestión manual o planillas Excel.
* Facilitar la búsqueda, consulta y gestión de productos en tiempo real, con visualización clara de precios en CLP.
* Mantener trazabilidad completa de todos los movimientos de inventario, con registro de fecha, hora y responsable.
* Separar y controlar el stock disponible para venta inmediata y el stock reservado para clientes o preventas.
* Alertar proactivamente sobre productos con stock bajo, permitiendo reabastecimiento oportuno.
* Visualizar promociones activas y precios con descuento de forma destacada para potenciar ventas.
* Permitir ajustes manuales de inventario con justificación y registro de responsable, para corregir errores, registrar pérdidas o ingresos extraordinarios.
* Brindar a Mundo Cartas una herramienta robusta y escalable para soportar el crecimiento y la diversificación de su catálogo.

**Alcance funcional:**

* Consulta y búsqueda avanzada de productos por nombre, SKU, código de barras, categoría y estado.
* Gestión completa de productos: agregar, editar, eliminar (con validaciones de stock y movimientos).
* Visualización diferenciada de stock disponible, stock reservado y stock total.
* Ajustes manuales de inventario (entrada/salida) con motivos predefinidos y registro en auditoría.
* Visualización de promociones activas: precio original tachado, precio con descuento, badges visuales.
* Alertas automáticas de stock bajo según umbral configurable por producto.
* Registro de todos los movimientos en la tabla de auditoría, incluyendo usuario, motivo y cantidad.
* Exportación de inventario y movimientos a CSV/Excel para análisis externo (si es must have).
* Integración opcional con lectores de código de barras para agilizar la gestión en tienda física.

**Límites y exclusiones:**

* No incluye carga masiva de productos por archivo (puede ser futuro).
* No contempla integración con sistemas externos de inventario o ERP en el MVP.
* No incluye gestión avanzada de ubicaciones físicas ni logística de almacén (futuro).

**Dependencias y relaciones:**

* Relación directa con el POS para actualización automática de stock tras ventas.
* Integración con el módulo de reservas para separar stock reservado.
* Depende de la gestión de usuarios para permisos de edición y ajustes.
* Relación con auditoría para trazabilidad de todos los movimientos.

**Escenarios clave:**

* Un vendedor de Mundo Cartas detecta un error de stock tras un conteo físico, realiza un ajuste manual con motivo “Corrección de inventario” y el movimiento queda registrado para auditoría.
* Un producto de Pokémon entra en promoción: el sistema muestra el precio original tachado, el nuevo precio con descuento y un badge verde “En Promoción” en el inventario y POS.
* El sistema alerta automáticamente a los responsables cuando un producto de alta rotación baja del umbral mínimo, permitiendo reabastecimiento antes de que se agote.

**EPIC-03: Punto de Venta (POS) y Ventas**

**Descripción general:**  
Esta épica abarca el flujo completo de ventas en mostrador, desde la búsqueda y selección de productos, el manejo del carrito, la aplicación de descuentos manuales, la selección de método de pago, hasta la confirmación de la venta y la actualización automática del inventario. Incluye la visualización de promociones, la validación de stock y la auditoría de todas las transacciones.

**Objetivos de negocio:**

* Agilizar el proceso de venta y cobro.
* Registrar ventas con todos los detalles y descuentos aplicados.
* Actualizar inventario automáticamente tras cada venta.
* Mantener trazabilidad y control de descuentos.

**Alcance funcional:**

* Carrito de compras con búsqueda y selección rápida.
* Aplicación de descuentos manuales (porcentaje/monto fijo) solo para usuarios autorizados.
* Visualización de productos en promoción.
* Selección obligatoria de método de pago.
* Registro de venta y actualización automática de stock.
* Auditoría de ventas y descuentos.
* Visualización de ventas en historial.

**Límites y exclusiones:**

* No incluye generación de boletas electrónicas ni integración SII (futuro).
* No contempla venta de cartas sueltas con atributos avanzados (futuro).

**Dependencias y relaciones:**

* Depende de inventario para stock y productos.
* Relación con gestión de usuarios para permisos de descuento.

**Escenarios clave:**

* Un vendedor aplica un descuento manual, el administrador recibe notificación y la venta queda registrada con detalle.
* Al procesar una venta, el stock se descuenta automáticamente y se actualizan los badges de inventario.

**EPIC-04: Reservas y Preventas**

**Descripción general:**  
Gestiona la creación, confirmación y cancelación de reservas de productos para clientes, separando stock reservado del disponible y permitiendo la conversión de reservas en ventas. Incluye la validación de datos del cliente, la generación de códigos únicos de reserva y la trazabilidad de cada etapa.

**Objetivos de negocio:**

* Permitir apartar productos para clientes.
* Mantener control y trazabilidad de reservas.
* Facilitar la conversión de reservas en ventas.

**Alcance funcional:**

* Creación de reservas con datos obligatorios del cliente.
* Separación automática de stock reservado.
* Confirmación de reserva como venta.
* Cancelación de reservas y liberación de stock.
* Alertas de reservas vencidas.
* Permisos configurables para crear/confirmar reservas.

**Límites y exclusiones:**

* No incluye integración con sistemas externos de reservas.
* Alertas avanzadas de vencimiento pueden ser futuras.

**Dependencias y relaciones:**

* Depende de inventario para stock reservado.
* Relación con POS para conversión de reservas en ventas.

**Escenarios clave:**

* Un vendedor reserva productos para un cliente, y otro vendedor puede confirmar la venta posteriormente.
* Un administrador cancela una reserva vencida y el stock se libera automáticamente.

**EPIC-05: Cierres de Caja y Conciliación**

**Descripción general:**  
Permite a usuarios autorizados realizar el cierre de caja diario, conciliando ventas, métodos de pago y detectando diferencias. Incluye el registro de totales, diferencias, observaciones y la generación de un historial inmodificable para auditoría y control contable.

**Objetivos de negocio:**

* Conciliar ventas y recaudación diaria.
* Detectar y registrar diferencias o errores.
* Mantener historial de cierres para auditoría.

**Alcance funcional:**

* Resumen automático de ventas, anulaciones y descuentos.
* Ingreso de montos contados y cálculo de diferencias.
* Registro y auditoría de cierres.
* Historial de cierres accesible y no modificable.

**Límites y exclusiones:**

* No incluye integración con sistemas contables externos (futuro).
* Exportación avanzada de cierres puede ser futura.

**Dependencias y relaciones:**

* Depende de ventas y POS para totales diarios.
* Relación con auditoría para registro de cierres.

**Escenarios clave:**

* Un vendedor realiza el cierre de caja, detecta una diferencia y la justifica en observaciones.
* Un administrador revisa el historial de cierres para auditoría interna.

**EPIC-06: Gestión de Descuentos y Promociones**

**Descripción general:**  
Incluye la gestión de descuentos manuales durante la venta y la administración de promociones masivas por producto o categoría, con control de vigencia y visualización destacada. Garantiza la trazabilidad y la notificación de descuentos aplicados.

**Objetivos de negocio:**

* Permitir descuentos excepcionales y promociones planificadas.
* Controlar y auditar todos los descuentos aplicados.
* Visualizar promociones activas en inventario y POS.

**Alcance funcional:**

* Aplicación de descuentos manuales en POS (porcentaje/monto fijo, motivo opcional).
* Gestión de promociones masivas (por producto/categoría, vigencia).
* Visualización de precios originales y con descuento.
* Notificación automática a administradores por descuentos manuales.
* Registro de descuentos en historial de ventas y auditoría.

**Límites y exclusiones:**

* No incluye integración con sistemas externos de promociones.
* Reportes avanzados de efectividad pueden ser futuros.

**Dependencias y relaciones:**

* Depende de inventario y POS para aplicar descuentos.
* Relación con usuarios para permisos de descuento.

**Escenarios clave:**

* Un administrador crea una promoción masiva para una categoría y los productos se actualizan automáticamente en el POS.
* Un vendedor autorizado aplica un descuento manual y el administrador recibe notificación inmediata.

**EPIC-07: Documentación, Soporte y Capacitación**

**Descripción general:**  
Abarca la creación, mantenimiento y acceso universal a la documentación básica de uso y soporte del sistema, así como la provisión de canales de contacto para soporte técnico. Busca facilitar la adopción, reducir incidencias y transmitir el valor del sistema.

**Objetivos de negocio:**

* Facilitar la adopción y el uso correcto del sistema.
* Reducir incidencias y consultas repetitivas.
* Transmitir el valor y las mejores prácticas al cliente.

**Alcance funcional:**

* Documentación digital accesible desde el sistema (Markdown, PDF, web simple).
* Secciones claras por flujo, ejemplos, capturas y FAQ.
* Registro de actualizaciones y responsable.
* Sección de contacto para soporte técnico.
* Checklist de validación final para el cliente.

**Límites y exclusiones:**

* No incluye tutoriales en video ni chat de soporte en tiempo real (futuro).
* No contempla manuales avanzados para integraciones externas.

**Dependencias y relaciones:**

* Depende de la implementación de todas las funcionalidades must have.
* Relación con soporte y equipo de producto para mantenimiento.

**Escenarios clave:**

* Un usuario accede a la documentación desde el dashboard y resuelve una duda sin requerir soporte.
* El equipo de soporte actualiza la documentación tras un cambio crítico y notifica a los usuarios.

**Historias de Usuario:**

US-001: Iniciar Sesión (Login Seguro y Control de Permisos)

Como vendedor o administrador  
Quiero iniciar sesión con mi usuario y contraseña  
Para acceder al sistema de forma segura y realizar solo las funciones permitidas según mi rol

**Criterios de Aceptación**

**Pantalla de Login**

1. El sistema muestra una pantalla de login con campos de usuario y contraseña.
2. Ambos campos son obligatorios y muestran indicadores visuales de requerimiento.
3. El botón de login solo se habilita si ambos campos están completos.
4. Incluye enlace visible “¿Olvidaste tu contraseña?” (puede estar inactivo en MVP si no se implementa recuperación).

**Autenticación y Seguridad**

1. El sistema valida las credenciales contra la base de datos de usuarios activos (is\_active = True).
2. El sistema identifica automáticamente el rol del usuario (Vendedor/Administrador) y carga sus permisos granulares:
   * Puede aplicar descuentos
   * Puede crear reservas
   * Puede editar inventario
   * Puede cancelar reservas
   * Puede anular ventas
   * Puede realizar cierre de caja
3. Si el usuario está desactivado, muestra mensaje: “Usuario desactivado. Contacta a un administrador.”
4. Si las credenciales son incorrectas, muestra mensaje genérico: “Usuario o contraseña incorrectos.”
   * No revela si el error es en usuario o contraseña (prevención de enumeración).

**Generación y Manejo de Token**

1. Al autenticarse correctamente, el sistema genera un token JWT con expiración de 24 horas.
2. El token se almacena en una cookie httpOnly con flags de seguridad:
   * httpOnly=true (no accesible desde JavaScript)
   * secure=true (solo HTTPS en producción)
   * sameSite=Strict (protección contra CSRF)
3. El token incluye solo información básica no sensible (id, username, rol, permisos).

**Redirección y Estado Global**

1. Tras login exitoso, el sistema redirige automáticamente al dashboard correspondiente según el rol:
   * /admin/dashboard para administradores
   * /vendedor/dashboard para vendedores
2. El frontend almacena el rol y permisos del usuario en estado global (Context/Redux) para validación rápida y control de interfaz.

**Manejo de Errores y Seguridad Adicional**

1. Después de 5 intentos fallidos consecutivos del mismo usuario, la cuenta se bloquea temporalmente por 15 minutos.
2. Si se intenta login durante el bloqueo, muestra mensaje: “Cuenta bloqueada temporalmente por seguridad. Intenta en 15 minutos.”
3. El contador de intentos fallidos se resetea después de login exitoso o al cumplirse el tiempo de bloqueo.
4. El sistema no revela si un usuario existe o no durante errores de login.

**Expiración y Control de Sesión**

1. Si el token expira (después de 24 horas), el sistema detecta error 401 en las peticiones protegidas.
2. Redirige automáticamente a la pantalla de login y muestra mensaje: “Tu sesión ha expirado. Por favor inicia sesión nuevamente.”

**Auditoría**

1. Cada intento de login (exitoso o fallido) se registra en la tabla AuditLog con:
   * Timestamp (fecha y hora automática)
   * Usuario (si existe, null si el usuario no existe)
   * Acción: 'login\_success' o 'login\_failed'
   * Resultado: booleano (true para éxito, false para fallo)
   * Detalles: motivo del fallo si aplica (“Contraseña incorrecta”, “Usuario inexistente”, “Usuario desactivado”)
   * IP address (opcional)

**Tareas Técnicas**

**Backend (Django)**

* Modelo Usuario con campos: username (único), password (hasheado), email, rol (Vendedor/Administrador), is\_active, permisos granulares (booleanos), intentos\_fallidos, bloqueado\_hasta, created\_at, updated\_at.
* Modelo AuditLog con campos: timestamp, user (ForeignKey nullable), action, result, details, ip\_address (opcional).
* Instalar y configurar djangorestframework-simplejwt para generación de tokens JWT.
* Configurar expiración de tokens a 24 horas.
* Endpoint POST /api/auth/login con validación de credenciales y lógica de bloqueo temporal.
* Lógica de reseteo automático de intentos fallidos.
* Respuesta con cookie httpOnly y flags de seguridad.
* Registro automático en AuditLog para cada intento de login.
* Middleware de autenticación JWT para validación en requests subsecuentes.

**Frontend (React)**

* Componente LoginPage con formulario de usuario y contraseña.
* Validación básica de campos vacíos y feedback visual.
* Context de Autenticación (AuthContext) para gestión global de usuario, rol y permisos.
* Función login() que guarda datos del usuario en Context.
* Configuración de axios con withCredentials: true para envío de cookies.
* Llamada POST a /api/auth/login desde el formulario.
* Lógica de redirección condicional según rol.
* Componente PrivateRoute para protección de rutas.
* Interceptor de axios para detectar errores 401 (token expirado) y redirigir a login.
* Mensajes de error claros y específicos según respuesta del backend.
* Loading state durante autenticación.
* Almacenar información básica del usuario en localStorage para persistencia entre recargas.

**Configuración**

* Variables de entorno: SECRET\_KEY, DEBUG, SECURE\_COOKIES, ALLOWED\_HOSTS, CORS\_ALLOWED\_ORIGINS.
* Flags de seguridad en cookies según entorno (desarrollo/producción).
* Documentar endpoints de autenticación y configuración de cookies.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuario puede iniciar sesión con credenciales válidas.
* Redirección correcta al dashboard según rol.
* Mensajes de error genéricos y seguros.
* Usuario desactivado no puede iniciar sesión.
* Cuenta se bloquea tras 5 intentos fallidos y se libera automáticamente.
* Permisos granulares cargados y disponibles en frontend.

**Seguridad**

* Token JWT con expiración de 24 horas, almacenado en cookie httpOnly.
* Flags de seguridad en cookie (httpOnly, secure, sameSite=Strict).
* Contraseñas hasheadas en base de datos.
* No se revela información sensible en errores.
* Usuarios no autenticados no acceden a rutas protegidas.

**Auditoría**

* Todos los intentos de login (éxito/fallo) registrados en AuditLog.
* Registros incluyen timestamp, usuario, acción, resultado, detalles.
* Registros de auditoría inmutables.

**Experiencia de Usuario**

* Mensajes claros y no revelan información sensible.
* Validación de campos vacíos antes de enviar.
* Loading state visible durante autenticación.
* Redirección automática tras login y al expirar token.

**Testing (Prioritarios)**

* Autenticación exitosa y fallida.
* Login con usuario desactivado.
* Bloqueo tras 5 intentos fallidos.
* Reseteo de contador tras login exitoso.
* Flujo completo login → dashboard → navegación.
* Verificación de flags de seguridad en cookie.
* Expiración de token y redirección.
* Registro correcto en AuditLog.

**Documentación**

* Endpoint /api/auth/login documentado con ejemplos.
* Variables de entorno y configuración de cookies documentadas.
* Proceso de configuración y protección de rutas en frontend.

**Dependencias**

* Base de datos y modelos configurados.
* Librería djangorestframework-simplejwt instalada.
* Variables de entorno correctamente configuradas.

**Notas Técnicas**

* En desarrollo local, el flag secure=True debe configurarse como False.
* Django usa PBKDF2 para hashing de contraseñas.
* El bloqueo temporal de 15 minutos es automático.
* Permisos granulares se cargan una vez y se almacenan en el estado global.
* El token JWT contiene solo información básica no sensible.
* La auditoría puede incluir IP address para mayor trazabilidad.

US-002: Cerrar Sesión (Logout Seguro y Limpieza de Estado)

Como usuario autenticado (vendedor o administrador)  
Quiero cerrar mi sesión activa de forma inmediata y segura  
Para proteger mi cuenta y asegurar que nadie acceda a mi información cuando no estoy usando el sistema

**Criterios de Aceptación**

**Visualización y Acceso**

1. El botón “Cerrar Sesión” es visible en el navbar/header de todas las pantallas protegidas del sistema.
2. El botón incluye un ícono reconocible (ejemplo: puerta, flecha de salida, logout).
3. El botón está posicionado de forma consistente y accesible (esquina superior derecha o lugar estándar del navbar).
4. El texto “Cerrar Sesión” es claro y legible.

**Funcionalidad y Seguridad**

1. Al hacer clic en “Cerrar Sesión”, el sistema cierra la sesión inmediatamente, sin solicitar confirmación adicional.
2. El sistema elimina la cookie access\_token que contiene el token JWT, estableciendo max\_age=0 y expires en el pasado.
3. El estado de autenticación se limpia completamente en el Context/Redux.
4. Se eliminan todos los datos de usuario almacenados en localStorage (username, rol, permisos).
5. El frontend redirige automáticamente a la pantalla de login (/login) después del logout.
6. Se muestra un mensaje de confirmación visible: “Has cerrado sesión correctamente”, mediante toast notification o banner informativo.
7. El mensaje desaparece automáticamente después de 3 segundos.
8. Después del logout, cualquier intento de acceder a rutas protegidas redirige automáticamente a login.
9. El token JWT eliminado no puede ser reutilizado para autenticar requests.
10. Las rutas protegidas validan la ausencia de token y previenen acceso no autorizado.
11. El estado de autenticación se limpia completamente sin dejar información sensible en el navegador.

**Auditoría y Trazabilidad**

1. El sistema registra el evento de logout en la tabla AuditLog con:
   * Timestamp (fecha y hora automática)
   * Usuario que cerró sesión (ForeignKey)
   * Acción: 'logout'
   * Resultado: True (logout exitoso)
   * Detalles: “Logout manual del usuario”
2. El registro de auditoría es inmutable y no puede ser editado ni eliminado.

**Experiencia de Usuario**

1. El botón de logout es accesible desde cualquier pantalla protegida del sistema.
2. El proceso de logout es instantáneo (< 1 segundo).
3. No genera errores si el usuario hace clic múltiples veces consecutivas.
4. Si el endpoint de logout falla (por problemas de conectividad), el frontend limpia la sesión localmente de todas formas.
5. El mensaje de confirmación usa toast notification para mejor experiencia (menos intrusivo que alerts o modals).
6. El botón de logout está estilizado de manera destacada pero no intrusiva, manteniendo consistencia visual con el resto del navbar.

**Tareas Técnicas**

**Backend (Django)**

* Implementar endpoint POST /api/auth/logout con autenticación requerida.
* Lógica para eliminar la cookie access\_token (max\_age=0, expires en el pasado).
* Registrar el evento en AuditLog con los datos del usuario y acción.
* Validar que solo usuarios autenticados puedan acceder al endpoint.
* Configurar respuesta adecuada para frontend (status 200 y mensaje de éxito).

**Frontend (React)**

* Componente Navbar reutilizable que incluya el botón de cerrar sesión en todas las rutas protegidas.
* Función handleLogout que llama al endpoint /api/auth/logout.
* Actualizar AuthContext con función logout() que limpia el estado global.
* Limpiar localStorage después del logout.
* Redirección automática a /login tras logout exitoso.
* Instalar y configurar react-toastify para mostrar notificaciones.
* Configurar toast notification para mensaje de confirmación “Has cerrado sesión correctamente”, con auto-close de 3 segundos.
* Manejo de errores: si el endpoint falla, cerrar sesión localmente de todas formas.
* Aplicar estilos CSS consistentes al botón de logout (color, posición, hover effects).

**Configuración**

* Agregar react-toastify como dependencia (npm install).
* Configurar ToastContainer en el componente raíz de la aplicación.
* Documentar endpoint /api/auth/logout en README o Swagger.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Botón “Cerrar Sesión” visible en todas las pantallas protegidas.
* Al hacer clic, la sesión se cierra inmediatamente.
* Cookie access\_token eliminada correctamente.
* Estado de autenticación y localStorage limpiados.
* Redirección automática a /login.
* Mensaje de confirmación visible y desaparece tras 3 segundos.

**Seguridad**

* Usuario no puede acceder a rutas protegidas después del logout.
* Intentos de acceso sin autenticación redirigen automáticamente a login.
* Token JWT eliminado no puede ser reutilizado.
* No queda información sensible en el navegador.

**Auditoría**

* Evento de logout registrado en AuditLog con todos los datos relevantes.
* Registro es inmutable.

**Experiencia de Usuario**

* Botón accesible y proceso instantáneo.
* No genera errores en clics repetidos.
* Funciona correctamente incluso con problemas de conectividad.
* Mensaje de confirmación profesional y no intrusivo.

**Testing (Prioritarios)**

* Verificar que endpoint elimina cookie correctamente.
* Verificar que logout limpia estado de Context.
* Verificar que localStorage se limpia.
* Flujo completo login → dashboard → logout → login nuevamente.
* Validar que token eliminado no funciona en requests subsecuentes.
* Verificar registro correcto en AuditLog.
* Verificar que botón es visible y clickeable en todas las pantallas.

**Documentación**

* Endpoint /api/auth/logout documentado con ejemplos.
* Proceso de configuración y protección de rutas en frontend.

**Dependencias**

* US-001 (Iniciar Sesión) implementada y funcional.
* Componente Navbar incluido en todas las rutas protegidas.
* Librería react-toastify instalada y configurada.
* AuthContext con función logout() implementada.

**Notas Técnicas**

* El logout no requiere confirmación para agilizar el proceso y evitar fricciones.
* Si el endpoint de logout falla, el frontend debe limpiar la sesión localmente para garantizar la seguridad.
* El mensaje de confirmación usa toast notifications para mejor experiencia.
* El botón de logout debe ser estilizado de manera destacada pero no intrusiva.
* La eliminación de la cookie se hace estableciendo max\_age=0, lo que indica al navegador que la elimine inmediatamente.

US-003: Crear Usuario con Roles y Permisos Granulares

Como administrador  
Quiero crear nuevos usuarios del sistema con roles y permisos específicos  
Para permitir que vendedores y otros administradores accedan al sistema con permisos controlados y seguros

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con rol Administrador pueden acceder al módulo de gestión de usuarios.
2. Opción “Gestión de Usuarios” visible en el menú de navegación solo para administradores.
3. Vendedores y otros roles sin permiso “Gestionar usuarios” no ven esta opción.
4. Pantalla principal muestra listado de usuarios existentes y botón “Crear Usuario”.

**Formulario de Creación**

1. Al hacer clic en “Crear Usuario”, se muestra formulario con los siguientes campos:
   * Nombre de usuario (username): requerido, único.
   * Email: requerido, formato válido, único.
   * Contraseña: requerida, generada automáticamente o ingresada manualmente.
   * Confirmar contraseña: requerida.
   * Rol base: requerido, opciones: Vendedor o Administrador.
   * Gmail para notificaciones: opcional (solo relevante para administradores).
   * Estado: checkbox “Usuario activo” (marcado por defecto).
2. Botón “Generar contraseña segura” crea contraseña aleatoria automáticamente (12 caracteres, combinación de mayúsculas, minúsculas, números y símbolos).
3. Contraseña generada o ingresada es visible para el administrador que crea el usuario.

**Asignación de Permisos Granulares**

1. Sección “Permisos Específicos” con checkboxes independientes:
   * Aplicar descuentos
   * Crear reservas
   * Editar inventario
   * Cancelar reservas
   * Anular ventas
   * Realizar cierre de caja
   * Gestionar usuarios
2. Si el rol base es Administrador, todos los permisos se marcan automáticamente (pueden desmarcarse si es necesario).
3. Si el rol base es Vendedor, todos los permisos están desmarcados por defecto (pueden marcarse según necesidad).
4. Permiso “Gestionar usuarios” solo puede asignarse a usuarios con rol Administrador (deshabilitado para Vendedor).

**Validaciones del Formulario**

1. Valida que username sea único en el sistema.
2. Valida que email sea único en el sistema.
3. Valida formato válido de email.
4. Valida que contraseña cumpla requisitos mínimos:
   * Mínimo 8 caracteres
   * Al menos una letra mayúscula
   * Al menos una letra minúscula
   * Al menos un número
5. Valida que contraseña y confirmación coincidan.
6. Valida que Gmail para notificaciones tenga formato válido de email de Google (si se ingresa).
7. Muestra mensajes de error específicos bajo cada campo con problema.

**Visualización de Contraseña y Confirmación**

1. Campo contraseña incluye ícono de “ojo” para alternar visibilidad.
2. Al enviar el formulario, muestra modal de confirmación con todos los datos del usuario creado.
3. Modal incluye la contraseña en texto plano con opción “Copiar contraseña”.
4. Modal advierte: “Guarda esta contraseña. No podrás verla nuevamente después de cerrar esta ventana.”

**Creación del Usuario**

1. Hashea la contraseña con bcrypt antes de almacenarla en base de datos.
2. Crea registro en tabla Usuario con todos los campos completados.
3. Almacena permisos granulares en campos booleanos individuales.
4. Establece is\_active = True si el checkbox “Usuario activo” está marcado.
5. Establece created\_at con timestamp automático.
6. Inicializa intentos\_fallidos = 0 y bloqueado\_hasta = null.

**Confirmación y Redirección**

1. Muestra mensaje de éxito: “Usuario [username] creado exitosamente.”
2. Redirige al listado de usuarios donde el nuevo usuario aparece visible.
3. Nuevo usuario tiene badge “Nuevo” visible por 24 horas.

**Notificación al Usuario Creado (Opcional)**

1. Opcionalmente envía correo al email del nuevo usuario con:
   * Bienvenida al sistema Mundo Cartas
   * Username asignado
   * Enlace al login
   * Instrucciones para primer acceso
   * No incluye la contraseña (debe ser comunicada por administrador directamente).

**Auditoría y Trazabilidad**

1. Registra creación de usuario en AuditLog con:
   * Timestamp
   * Usuario: administrador que creó el usuario
   * Acción: 'user\_created'
   * Resultado: true
   * Detalles: username del nuevo usuario, rol asignado, permisos otorgados

**Tareas Técnicas**

**Backend (Django)**

* Endpoint POST /api/users/ accesible solo para administradores.
* Validaciones de unicidad para username y email.
* Validaciones de formato para email y Gmail.
* Validación de requisitos de contraseña.
* Lógica de generación de contraseña segura aleatoria.
* Serializer que hashea contraseña antes de guardar.
* Asignación automática de permisos según rol base.
* Validación: permiso “Gestionar usuarios” solo para rol Administrador.
* Registro en AuditLog después de creación exitosa.
* Endpoint GET /api/users/ para listar usuarios (solo administradores).

**Frontend (React)**

* Componente UserManagementPage accesible solo para administradores.
* Componente CreateUserForm con todos los campos especificados.
* Validaciones en tiempo real de campos requeridos.
* Validación de unicidad de username y email consultando al backend.
* Generador de contraseñas seguras aleatorias.
* Toggle de visibilidad de contraseña con ícono de ojo.
* Sección de permisos granulares con checkboxes independientes.
* Lógica: marcar todos los permisos si rol es Administrador, desmarcar si es Vendedor.
* Validación: deshabilitar permiso “Gestionar usuarios” si rol es Vendedor.
* Modal de confirmación que muestra contraseña en texto plano.
* Funcionalidad “Copiar contraseña” al clipboard.
* Llamada POST a /api/users/ con todos los datos del formulario.
* Mensajes de error específicos del backend.
* Redirección al listado después de creación exitosa.

**Configuración**

* Permisos en Django REST Framework: IsAuthenticated y custom IsAdministrator.
* Validaciones de contraseña en settings de Django.
* CORS configurado para permitir requests desde frontend.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo administradores pueden acceder al módulo de creación de usuarios.
* Formulario valida todos los campos correctamente.
* Username y email únicos en el sistema.
* Contraseña cumple requisitos mínimos de seguridad.
* Generador de contraseña segura funciona correctamente.
* Rol base y permisos granulares asignados correctamente.
* Permiso “Gestionar usuarios” solo para Administradores.
* Gmail para notificaciones guardado correctamente (opcional).
* Contraseña visible para administrador al momento de creación.
* Modal de confirmación permite copiar contraseña.
* Usuario creado correctamente en base de datos con contraseña hasheada.
* Nuevo usuario puede iniciar sesión con credenciales creadas.
* Usuario aparece en listado con badge “Nuevo”.

**Seguridad**

* Endpoint de creación solo accesible para administradores autenticados.
* Contraseña hasheada con bcrypt antes de almacenar.
* Validación de permisos en backend y frontend.
* Permiso “Gestionar usuarios” validado en backend.
* No se almacena contraseña en texto plano.

**Auditoría**

* Creación de usuario registrada en AuditLog con todos los datos relevantes.
* Registro inmutable.

**Experiencia de Usuario**

* Formulario intuitivo y validaciones en tiempo real.
* Mensajes de error claros y específicos.
* Generador de contraseña facilita creación segura.
* Modal de confirmación y advertencia sobre contraseña temporal.
* Funcionalidad “Copiar contraseña” fácil de usar.

**Testing (Prioritarios)**

* Integración: administrador crea usuario con rol y permisos → usuario puede iniciar sesión.
* Seguridad: vendedor sin permiso no puede acceder al módulo.
* Validación: username duplicado es rechazado.
* Validación: contraseña débil es rechazada.

**Documentación**

* Endpoint POST /api/users/ documentado con ejemplos.
* Campos requeridos y opcionales documentados.
* Requisitos de contraseña y permisos granulares documentados.

**Dependencias**

* US-001 (Iniciar Sesión) completada.
* Modelo Usuario con campos de permisos granulares.
* Modelo AuditLog disponible.
* Sistema de autenticación JWT funcional.

**Notas Técnicas**

* Generador de contraseñas puede usar librerías como secrets en Python o crypto en JavaScript.
* Contraseña mostrada al administrador no debe almacenarse después de cerrar el modal.
* Gmail para notificaciones solo relevante para administradores.
* Permisos granulares almacenados como campos booleanos individuales.
* Badge “Nuevo” implementado comparando created\_at con timestamp actual (< 24 horas).
* Considerar rate limiting en el endpoint para prevenir creación masiva de usuarios.

US-004: Editar Usuario y Permisos Granulares

Como administrador  
Quiero editar la información y permisos de usuarios existentes  
Para actualizar sus datos, modificar sus permisos o cambiar su estado según las necesidades del negocio

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con rol Administrador pueden acceder a la función de edición de usuarios.
2. En el listado de usuarios, cada registro tiene botón “Editar” visible solo para administradores.
3. Vendedores y otros roles sin permiso “Gestionar usuarios” no ven opción de edición.

**Carga de Datos Existentes**

1. Al hacer clic en “Editar”, redirige a formulario de edición con datos actuales del usuario precargados.
2. Formulario muestra todos los campos editables con valores actuales:
   * Nombre de usuario (username): solo visualización, no editable.
   * Email: editable.
   * Rol base: editable (Vendedor o Administrador).
   * Gmail para notificaciones: editable (opcional).
   * Estado: checkbox “Usuario activo”.
   * Permisos granulares: checkboxes con valores actuales marcados.
3. Campo username se muestra pero está deshabilitado para edición.
4. Contraseña NO se muestra (está hasheada en base de datos).

**Edición de Información Básica**

1. Administrador puede modificar email del usuario.
2. Administrador puede modificar Gmail para notificaciones.
3. Administrador puede cambiar el rol base entre Vendedor y Administrador.
4. Administrador puede activar/desactivar usuario mediante checkbox “Usuario activo”.

**Edición de Permisos Granulares**

1. Administrador puede marcar/desmarcar permisos independientemente:
   * Aplicar descuentos
   * Crear reservas
   * Editar inventario
   * Cancelar reservas
   * Anular ventas
   * Realizar cierre de caja
   * Gestionar usuarios
2. Si cambia rol a Administrador, todos los permisos se marcan automáticamente (pueden desmarcarse si es necesario).
3. Si cambia rol a Vendedor, permisos se mantienen como estén (no se desmarcan automáticamente).
4. Permiso “Gestionar usuarios” solo puede asignarse si el rol es Administrador (deshabilitado para Vendedor).
5. Si cambia rol de Administrador a Vendedor, permiso “Gestionar usuarios” se desmarca automáticamente.

**Cambio de Contraseña**

1. Sección separada “Cambiar Contraseña” con checkbox “¿Deseas cambiar la contraseña?”.
2. Al marcar checkbox, muestra campos:
   * Nueva contraseña
   * Confirmar nueva contraseña
3. Incluye botón “Generar contraseña segura” que crea contraseña aleatoria automáticamente.
4. Nueva contraseña es visible para el administrador.
5. Si no se marca el checkbox, la contraseña NO se modifica.

**Validaciones del Formulario**

1. Valida que email sea único (puede ser el mismo actual del usuario).
2. Valida formato válido de email.
3. Valida que Gmail para notificaciones tenga formato válido (si se ingresa).
4. Si se cambia contraseña, valida requisitos mínimos:
   * Mínimo 8 caracteres
   * Al menos una letra mayúscula
   * Al menos una letra minúscula
   * Al menos un número
5. Valida que nueva contraseña y confirmación coincidan.
6. Muestra mensajes de error específicos bajo cada campo con problema.

**Restricciones de Seguridad**

1. Administrador NO puede editarse a sí mismo para quitarse permisos críticos.
2. Administrador NO puede desactivarse a sí mismo.
3. Si el usuario editado está actualmente logueado, cambios de permisos se aplican en su próximo inicio de sesión.
4. Si se desactiva un usuario (is\_active = False), su token actual se invalida inmediatamente.

**Confirmación de Cambios**

1. Botón “Guardar Cambios” envía el formulario.
2. Si se cambió la contraseña, muestra modal de confirmación con nueva contraseña en texto plano.
3. Modal incluye opción “Copiar contraseña”.
4. Modal advierte: “Comunica esta contraseña al usuario. No podrás verla nuevamente.”
5. Muestra mensaje de éxito: “Usuario [username] actualizado exitosamente.”

**Actualización en Base de Datos**

1. Actualiza campos modificados en tabla Usuario.
2. Si se cambió contraseña, hashea con bcrypt antes de almacenar.
3. Actualiza permisos granulares según checkboxes seleccionados.
4. Actualiza updated\_at con timestamp automático.
5. Si se desactivó usuario, invalida su token JWT actual agregándolo a blacklist.

**Auditoría y Trazabilidad**

1. Registra edición de usuario en AuditLog con:
   * Timestamp
   * Usuario: administrador que realizó la edición
   * Acción: 'user\_updated'
   * Resultado: true
   * Detalles: username editado, campos modificados (antes y después), cambios en permisos
2. Si se cambió contraseña, registra acción adicional: 'password\_changed\_by\_admin'

**Tareas Técnicas**

**Backend (Django)**

* Endpoint GET /api/users/{id}/ para obtener datos del usuario (solo administradores).
* Endpoint PUT/PATCH /api/users/{id}/ para actualizar usuario (solo administradores).
* Validaciones de unicidad para email (excluyendo el email actual del usuario).
* Validación de requisitos de contraseña si se proporciona nueva contraseña.
* Lógica: permiso “Gestionar usuarios” solo para rol Administrador.
* Restricción: administrador no puede desactivarse a sí mismo ni quitarse permiso “Gestionar usuarios”.
* Lógica de invalidación de token si usuario se desactiva.
* Actualización de updated\_at automático.
* Registro en AuditLog con campos modificados (antes/después).

**Frontend (React)**

* Componente EditUserForm que recibe ID del usuario como parámetro.
* Llamada GET a /api/users/{id}/ para cargar datos al montar componente.
* Precarga todos los campos del formulario con datos actuales.
* Deshabilitar campo username (solo visualización).
* Validaciones en tiempo real de campos editables.
* Sección de cambio de contraseña con checkbox condicional.
* Generador de contraseñas seguras aleatorias.
* Toggle de visibilidad de contraseña.
* Lógica: marcar todos los permisos si cambia rol a Administrador.
* Lógica: desmarcar “Gestionar usuarios” si cambia rol a Vendedor.
* Validación: prevenir que administrador se desactive a sí mismo o se quite permiso crítico.
* Modal de confirmación para nueva contraseña (si se cambió).
* Funcionalidad “Copiar contraseña” al clipboard.
* Llamada PUT/PATCH a /api/users/{id}/ con campos modificados.
* Mensajes de error específicos del backend.
* Redirigir al listado de usuarios después de actualización exitosa.

**Configuración**

* Permisos en endpoint: IsAuthenticated y custom IsAdministrator.
* Serializer que solo hashee contraseña si se proporciona nueva.
* CORS configurado para permitir requests desde frontend.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo administradores pueden acceder a edición de usuarios.
* Formulario carga correctamente datos actuales del usuario.
* Username se muestra pero no es editable.
* Email puede modificarse y valida unicidad.
* Rol base puede cambiarse entre Vendedor y Administrador.
* Gmail para notificaciones puede modificarse.
* Estado activo/inactivo puede modificarse.
* Permisos granulares pueden modificarse independientemente.
* Permisos se ajustan automáticamente al cambiar rol.
* Permiso “Gestionar usuarios” solo asignable a Administradores.
* Contraseña puede cambiarse opcionalmente y cumple requisitos mínimos.
* Generador de contraseña segura funciona correctamente.
* Modal muestra nueva contraseña en texto plano si se cambió.
* Administrador no puede desactivarse a sí mismo ni quitarse permiso crítico.
* Usuario desactivado tiene token invalidado inmediatamente.
* Cambios se guardan correctamente en base de datos.
* Usuario actualizado puede iniciar sesión con nueva configuración.

**Seguridad**

* Endpoint de edición solo accesible para administradores autenticados.
* Nueva contraseña hasheada con bcrypt antes de almacenar.
* Restricciones de auto-edición validadas en backend.
* Usuario desactivado no puede usar tokens existentes.
* Validación de permisos en backend.

**Auditoría**

* Edición de usuario registrada en AuditLog con todos los datos relevantes.
* Cambio de contraseña registrado como acción separada.
* Registros de auditoría inmutables.

**Experiencia de Usuario**

* Formulario precarga datos actuales correctamente.
* Validaciones en tiempo real guían al administrador.
* Mensajes de error claros y específicos.
* Sección de contraseña opcional y clara.
* Modal de nueva contraseña permite copiarla fácilmente.
* Restricciones de auto-edición muestran mensajes informativos.

**Testing (Prioritarios)**

* Integración: administrador edita permisos de vendedor → vendedor tiene nuevos permisos al iniciar sesión.
* Seguridad: administrador no puede desactivarse a sí mismo ni quitarse permiso crítico.
* Validación: email duplicado (de otro usuario) es rechazado.
* Funcionalidad: usuario desactivado tiene token invalidado.

**Documentación**

* Endpoints GET y PUT/PATCH /api/users/{id}/ documentados con ejemplos.
* Campos editables y no editables documentados.
* Restricciones de auto-edición documentadas.

**Dependencias**

* US-003 (Crear Usuario) completada.
* Modelo Usuario con todos los campos necesarios.
* TokenBlacklist disponible para invalidar tokens.
* Modelo AuditLog disponible.

**Notas Técnicas**

* Username no es editable para mantener integridad histórica y auditoría.
* Si un usuario con sesión activa es desactivado, su token debe invalidarse inmediatamente.
* Cambios de permisos para usuarios logueados se aplican al verificar el token en cada request.
* Al cambiar rol de Administrador a Vendedor, se quita permiso “Gestionar usuarios” automáticamente.
* El administrador maestro podría tener protección adicional contra desactivación.
* Considerar confirmación adicional al desactivar usuarios para prevenir errores accidentales.

US-005: Listar Usuarios con Roles y Permisos

Como administrador  
Quiero visualizar un listado completo de todos los usuarios del sistema con sus roles y permisos  
Para tener control sobre quién tiene acceso al sistema y gestionar sus cuentas eficientemente

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con rol Administrador pueden acceder al módulo de gestión de usuarios.
2. Opción “Gestión de Usuarios” visible en el menú de navegación solo para administradores.
3. Al hacer clic, redirige a /admin/usuarios.

**Visualización del Listado**

1. Pantalla muestra tabla con todos los usuarios del sistema.
2. Tabla incluye columnas:
   * Username
   * Email
   * Rol (Administrador/Vendedor)
   * Estado (Activo/Inactivo)
   * Fecha de creación
   * Última actualización
   * Acciones (Editar, Ver Detalles)
3. Usuarios activos e inactivos se diferencian visualmente (badge verde “Activo” / gris “Inactivo”).
4. Usuarios con rol Administrador tienen badge distintivo.
5. Botón “Crear Usuario” visible en parte superior de la tabla.

**Ordenamiento**

1. Por defecto, usuarios ordenados por fecha de creación (más recientes primero).
2. Columnas ordenables al hacer clic en encabezado: username, email, rol, estado, fecha.
3. Indicador visual muestra columna y dirección de ordenamiento actual (ascendente/descendente).

**Búsqueda**

1. Barra de búsqueda en parte superior de la tabla.
2. Búsqueda en tiempo real filtra por: username, email.
3. Resultados se actualizan mientras el administrador escribe.
4. Muestra mensaje “No se encontraron usuarios” si búsqueda no arroja resultados.

**Filtros**

1. Filtro por rol con opciones: Todos, Administrador, Vendedor.
2. Filtro por estado con opciones: Todos, Activos, Inactivos.
3. Filtros aplicables simultáneamente.
4. Contador visible muestra cantidad de resultados: “Mostrando X usuarios”.

**Paginación**

1. Tabla muestra máximo 20 usuarios por página.
2. Controles de paginación visibles en parte inferior.
3. Muestra: Página actual, total de páginas, botones Anterior/Siguiente.
4. Opción de cambiar cantidad de resultados por página: 10, 20, 50, 100.

**Ver Detalles de Usuario**

1. Botón “Ver Detalles” en cada fila de la tabla.
2. Al hacer clic, muestra modal con información completa del usuario:
   * Username
   * Email
   * Rol base
   * Gmail para notificaciones
   * Estado (Activo/Inactivo)
   * Fecha de creación
   * Última actualización
   * Permisos granulares (listado con checkmarks visuales)
3. Modal incluye botón “Editar Usuario” que redirige al formulario de edición.
4. Modal incluye botón “Cerrar”.

**Visualización de Permisos**

1. En el modal de detalles, sección “Permisos Asignados” muestra:
   * Aplicar descuentos: ✓ o ✗
   * Crear reservas: ✓ o ✗
   * Editar inventario: ✓ o ✗
   * Cancelar reservas: ✓ o ✗
   * Anular ventas: ✓ o ✗
   * Realizar cierre de caja: ✓ o ✗
   * Gestionar usuarios: ✓ o ✗
2. Permisos activos se muestran con checkmark verde, inactivos con X gris.

**Acciones Rápidas**

1. Cada fila tiene menú de acciones con opciones:
   * Editar usuario
   * Ver detalles
   * Activar/Desactivar (toggle según estado actual)
2. Acción “Desactivar” muestra confirmación: “¿Estás seguro que deseas desactivar a [username]?”
3. Acción “Activar” no requiere confirmación.
4. Administrador no puede desactivarse a sí mismo desde el listado.

**Indicadores Visuales**

1. Badge “Tú” visible en la fila del administrador que está consultando el listado.
2. Badge “Nuevo” visible en usuarios creados hace menos de 24 horas.
3. Usuarios bloqueados temporalmente (por intentos fallidos) muestran badge “Bloqueado” con tiempo restante.

**Información de Actividad Reciente**

1. Columna adicional opcional “Último acceso” muestra fecha/hora del último login exitoso.
2. Si el usuario nunca ha iniciado sesión, muestra “Nunca”.

**Exportación**

1. Botón “Exportar” en parte superior permite descargar listado de usuarios.
2. Opciones de exportación: CSV, Excel.
3. Archivo exportado incluye: username, email, rol, estado, fecha creación, permisos.
4. Exportación respeta filtros y búsquedas activas.

**Tareas Técnicas**

**Backend (Django)**

* Endpoint GET /api/users/ que retorna listado completo de usuarios (solo administradores).
* Implementar paginación en el endpoint con parámetros: page, page\_size.
* Implementar ordenamiento con parámetro: ordering (ej: -created\_at, username).
* Implementar búsqueda con parámetro: search (busca en username y email).
* Implementar filtros con parámetros: role (Administrador/Vendedor), is\_active (true/false).
* Endpoint GET /api/users/{id}/ para detalles de usuario específico.
* Endpoint PATCH /api/users/{id}/toggle-active/ para activar/desactivar rápidamente.
* Endpoint GET /api/users/export/ que retorna CSV o Excel según parámetro format.
* Incluir en respuesta: contador total de usuarios, usuarios por página, página actual.
* Excluir campo password de todas las respuestas (nunca retornar contraseña).

**Frontend (React)**

* Componente UserListPage con tabla de usuarios.
* Llamada GET a /api/users/ al montar componente.
* Componente UserTable con columnas especificadas.
* Barra de búsqueda con debounce (300ms) para evitar requests excesivos.
* Filtros por rol y estado con dropdowns.
* Ordenamiento al hacer clic en encabezados de columna.
* Componente Pagination con controles de navegación.
* Modal UserDetailsModal que muestra información completa.
* Visualización de permisos con checkmarks visuales.
* Menú de acciones rápidas (Editar, Ver, Activar/Desactivar).
* Modal de confirmación para desactivar usuario.
* Acción de activar/desactivar con llamada a endpoint toggle.
* Funcionalidad de exportación con descarga de archivo.
* Mostrar badges distintivos (Activo/Inactivo, Administrador, Nuevo, Tú, Bloqueado).
* Estados de carga (loading) y errores apropiados.
* Actualizar listado automáticamente después de acciones (editar, activar/desactivar).

**Configuración**

* Tamaño de página por defecto: PAGE\_SIZE = 20.
* Permisos en endpoints: IsAuthenticated y custom IsAdministrator.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo administradores pueden acceder al listado de usuarios.
* Tabla muestra todos los usuarios del sistema correctamente.
* Columnas muestran información precisa y actualizada.
* Búsqueda filtra resultados en tiempo real correctamente.
* Filtros por rol y estado funcionan correctamente y pueden aplicarse simultáneamente.
* Ordenamiento por columnas funciona correctamente.
* Paginación muestra páginas correctamente con controles funcionales.
* Modal de detalles muestra información completa del usuario.
* Permisos se visualizan correctamente con checkmarks.
* Acción “Editar” redirige al formulario de edición.
* Acción “Desactivar” requiere confirmación y actualiza estado.
* Acción “Activar” actualiza estado inmediatamente.
* Administrador no puede desactivarse a sí mismo.
* Badges visuales se muestran correctamente (Activo, Administrador, Nuevo, Tú).
* Exportación descarga archivo con datos correctos respetando filtros.
* Listado se actualiza automáticamente después de acciones.

**Seguridad**

* Endpoint de listado solo accesible para administradores autenticados.
* Contraseñas nunca se incluyen en respuestas del backend.
* Validación de permisos en backend, no solo frontend.
* Vendedores sin permiso no pueden acceder al módulo.

**Experiencia de Usuario**

* Tabla clara y fácil de leer.
* Búsqueda rápida y responsiva.
* Filtros intuitivos y fáciles de usar.
* Badges visuales facilitan identificación rápida de estados.
* Modal de detalles organiza información de manera clara.
* Acciones rápidas accesibles y bien organizadas.
* Confirmación previene desactivaciones accidentales.
* Estados de carga muestran feedback apropiado.
* Mensajes de error claros y útiles.

**Testing (Prioritarios)**

* Integración: administrador accede al listado → visualiza todos los usuarios correctamente.
* Búsqueda: búsqueda por username filtra resultados correctamente.
* Filtros: filtrar por rol Vendedor muestra solo vendedores.
* Seguridad: vendedor sin permiso no puede acceder al módulo.

**Documentación**

* Endpoint GET /api/users/ documentado con parámetros y ejemplos.
* Endpoint GET /api/users/{id}/ documentado.
* Parámetros de búsqueda, filtros y ordenamiento documentados.
* Formato de exportación documentado.

**Dependencias**

* US-003 (Crear Usuario) completada.
* US-004 (Editar Usuario) completada.
* Modelo Usuario con todos los campos necesarios.
* Sistema de autenticación JWT funcional.

**Notas Técnicas**

* El debounce en búsqueda previene requests excesivos mientras el usuario escribe (espera 300ms de inactividad).
* La exportación debe respetar filtros activos para permitir exportaciones específicas.
* Badge “Nuevo” se calcula comparando created\_at con timestamp actual (< 24 horas).
* Badge “Bloqueado” se calcula verificando campo bloqueado\_hasta > timestamp actual.
* Último acceso puede implementarse agregando campo last\_login que se actualiza en US-001.
* Considerar actualización automática del listado con WebSockets o polling si múltiples administradores gestionan usuarios simultáneamente.
* La paginación del backend debe usar django.core.paginator o DRF's PageNumberPagination.

US-006: Dashboard Personalizado por Rol

Como vendedor o administrador  
Quiero ver un dashboard personalizado según mi rol y permisos al iniciar sesión  
Para acceder rápidamente a las funcionalidades y métricas relevantes para mi trabajo

**Criterios de Aceptación**

**Redirección Post-Login**

1. Después de iniciar sesión exitosamente, el sistema redirige automáticamente según el rol:
   * Administrador: /admin/dashboard
   * Vendedor: /vendedor/dashboard
2. El dashboard se carga automáticamente sin interacción adicional.

**Dashboard de Administrador**

1. Muestra sección de bienvenida: “Bienvenido, [nombre\_usuario]”.
2. Incluye tarjetas de métricas principales:
   * Total de ventas del día (en CLP).
   * Número de ventas realizadas hoy.
   * Total de ventas del mes actual (en CLP).
   * Número de productos con stock bajo (alerta).
   * Número de reservas pendientes.
   * Total de descuentos aplicados hoy (en CLP).
3. Sección “Alertas y Notificaciones” visible con:
   * Productos con stock bajo (lista con links a inventario).
   * Reservas vencidas pendientes de cancelar.
   * Descuentos aplicados por vendedores en las últimas 24 horas.
   * Badge con contador numérico de alertas pendientes en ícono de campana.
4. Sección “Accesos Rápidos” con botones a módulos principales:
   * Punto de Venta.
   * Gestión de Inventario.
   * Gestión de Usuarios.
   * Reportes y Análisis.
   * Gestión de Descuentos.
   * Ver Reservas.
5. Gráfico de ventas de los últimos 7 días (línea o barras).
6. Listado de últimas 5 ventas realizadas con: ID, fecha/hora, vendedor, total.

**Dashboard de Vendedor**

1. Muestra sección de bienvenida: “Bienvenido, [nombre\_usuario]”.
2. Incluye tarjetas de métricas personales:
   * Mis ventas del día (cantidad y total en CLP).
   * Mi última venta (hora y monto).
   * Productos vendidos hoy por mí.
3. Sección “Alertas” visible solo si tiene permiso:
   * Productos con stock bajo (si tiene permiso “Editar inventario”).
   * Reservas pendientes (si tiene permiso “Crear reservas”).
4. Sección “Accesos Rápidos” muestra solo módulos según permisos:
   * Punto de Venta (siempre visible).
   * Crear Reserva (si tiene permiso “Crear reservas”).
   * Ver Inventario (si tiene permiso “Editar inventario”).
   * Mis Reportes (si tiene permiso “Acceder reportes avanzados”).
5. Listado de mis últimas 5 ventas realizadas hoy con: hora, productos, total.

**Métricas en Tiempo Real**

1. Métricas del dashboard se calculan en tiempo real consultando la base de datos.
2. Valores monetarios siempre se muestran en formato CLP: $1.500, $15.000, $150.000.
3. Contadores numéricos se actualizan al recargar la página.
4. No hay auto-refresh automático; el usuario debe recargar manualmente.
5. Loading state visible mientras cargan las métricas.

**Navegación Desde Dashboard**

1. Todos los botones de “Accesos Rápidos” redirigen a módulos correspondientes.
2. Alertas tienen links directos a secciones relevantes (ej: click en producto con stock bajo redirige a ese producto en inventario).
3. Últimas ventas tienen link “Ver detalle” que abre modal con información completa.

**Permisos y Visibilidad**

1. Vendedores solo ven secciones y métricas para las que tienen permiso.
2. Si vendedor no tiene ningún permiso especial, solo ve métricas básicas personales y acceso a POS.
3. Administradores siempre ven todas las secciones del dashboard completo.

**Tareas Técnicas**

**Backend (Django)**

* Crear endpoint GET /api/dashboard/admin/ que retorna métricas para administradores:
  + Ventas del día (suma total, cantidad).
  + Ventas del mes (suma total, cantidad).
  + Productos con stock bajo (conteo y listado).
  + Reservas pendientes (conteo).
  + Descuentos aplicados hoy (suma total).
  + Últimas 5 ventas generales.
  + Datos para gráfico de ventas últimos 7 días.
* Crear endpoint GET /api/dashboard/vendedor/ que retorna métricas personales:
  + Ventas del día del vendedor actual (suma, cantidad).
  + Última venta del vendedor (hora, monto).
  + Productos vendidos hoy por el vendedor.
  + Últimas 5 ventas del vendedor hoy.
* Implementar filtros por fecha en queries (inicio y fin del día actual, mes actual).
* Implementar filtros por usuario en métricas de vendedor.
* Configurar permisos: /api/dashboard/admin/ solo para administradores, /api/dashboard/vendedor/ para vendedores.
* Optimizar queries con select\_related y prefetch\_related para evitar N+1.

**Frontend (React)**

* Crear componente AdminDashboard con todas las secciones especificadas.
* Crear componente VendedorDashboard con secciones personales.
* Implementar llamada GET a endpoints correspondientes según rol al montar componente.
* Crear componente MetricCard reutilizable para tarjetas de métricas.
* Crear componente AlertSection para mostrar alertas con badges numéricos.
* Crear componente QuickAccessButtons con botones condicionales según permisos.
* Implementar gráfico de ventas usando librería de charts (Chart.js o Recharts).
* Crear componente RecentSalesList para últimas ventas.
* Implementar modal de detalle de venta al hacer click en “Ver detalle”.
* Implementar loading states mientras cargan métricas.
* Implementar manejo de errores si falla carga de métricas.
* Formatear valores monetarios en CLP con separadores de miles.
* Implementar links desde alertas a secciones correspondientes.
* Renderizar accesos rápidos dinámicamente según permisos del usuario en Context.

**Configuración**

* Configurar formato de moneda CLP en constantes del frontend.
* Configurar zona horaria del servidor para cálculos correctos de “hoy”.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Después de login, sistema redirige automáticamente al dashboard apropiado.
* Dashboard de administrador muestra todas las métricas correctamente.
* Dashboard de vendedor muestra métricas personales correctamente.
* Métricas se calculan correctamente (ventas del día, mes, stock bajo, etc.).
* Valores monetarios se formatean en CLP correctamente.
* Alertas muestran información precisa y actualizada.
* Contadores de alertas se calculan correctamente.
* Accesos rápidos muestran solo módulos según permisos del usuario.
* Links desde alertas redirigen correctamente a secciones correspondientes.
* Gráfico de ventas muestra datos de últimos 7 días correctamente.
* Últimas ventas muestran información precisa con opción de ver detalle.
* Loading states son visibles durante carga de datos.
* Dashboard es responsive y se adapta a diferentes tamaños de pantalla.

**Seguridad**

* Endpoints de dashboard validan autenticación y rol apropiado.
* Administradores no pueden acceder a endpoint de vendedor y viceversa (o se permite pero retorna datos según rol).
* Vendedores solo ven sus propias métricas, no de otros vendedores.
* Permisos se validan en backend antes de retornar datos sensibles.

**Experiencia de Usuario**

* Dashboard se carga rápidamente después de login.
* Métricas son claras y fáciles de entender.
* Alertas son visibles y llamativas (colores de advertencia).
* Accesos rápidos están bien organizados y son accesibles.
* Gráfico es claro y fácil de interpretar.
* Formato de moneda es consistente en todo el dashboard.
* Loading states no bloquean la UI innecesariamente.
* Mensajes de error son claros si falla carga de métricas.

**Testing (Prioritarios)**

* Test de integración: administrador inicia sesión → redirige a dashboard admin → métricas se cargan correctamente.
* Test de integración: vendedor inicia sesión → redirige a dashboard vendedor → métricas personales se cargan.
* Test de permisos: vendedor solo ve accesos rápidos según sus permisos.
* Test de cálculo: métricas de ventas del día se calculan correctamente.

**Documentación**

* Endpoints /api/dashboard/admin/ y /api/dashboard/vendedor/ documentados con response examples.
* Estructura de métricas documentada.
* Permisos requeridos para cada endpoint documentados.

**Dependencias**

* US-001 (Iniciar Sesión) completada con redirección por rol.
* Modelo de Ventas existente con datos para métricas.
* Modelo de Inventario con campo de stock mínimo.
* Modelo de Reservas con estados.
* Sistema de permisos granulares funcional.
* AuthContext con información de usuario y permisos cargados.

**Notas Técnicas**

* Las métricas “del día” deben considerar zona horaria de Chile (UTC-3) para calcular correctamente inicio y fin del día.
* El gráfico de ventas puede usar Chart.js (más liviano) o Recharts (más react-friendly).
* Considerar caché de métricas en Redis para dashboards con alto tráfico (optimización futura).
* Las queries de métricas deben optimizarse con índices en campos de fecha y usuario.
* El formato CLP debe usar función reutilizable: formatCurrency(amount) que retorne $1.500.
* Badge de alertas en navegación debe actualizarse al navegar de vuelta al dashboard.
* Considerar implementar auto-refresh opcional cada 5 minutos para mantener métricas actualizadas.
* El conteo de “productos con stock bajo” compara stock\_disponible contra umbral configurado por producto.
* Las “reservas pendientes” son aquellas con estado “Pendiente” (no confirmadas ni canceladas).
* Los “descuentos aplicados por vendedores” se filtran de AuditLog con acción 'discount\_applied' donde el usuario no es administrador.

US-007: Consultar y Buscar Inventario con Filtros

Como administrador o vendedor  
Quiero consultar y buscar productos del inventario aplicando filtros y búsqueda por criterios  
Para localizar rápidamente productos, revisar stock disponible y tomar decisiones informadas sobre reposición o ventas

Criterios de Aceptación

Acceso y Permisos

* La opción "Inventario" está visible en el menú principal para todos los usuarios autenticados (Administrador y Vendedor).
* Al ingresar, se muestra pantalla /inventario con tabla de productos.
* Los vendedores pueden consultar el inventario pero solo pueden editar si tienen el permiso específico "Editar inventario".

Visualización del Inventario

* Tabla principal con columnas:
  + SKU
  + Nombre del producto
  + Categoría
  + Precio unitario (en CLP con formato $XX.XXX)
  + Stock disponible
  + Stock reservado
  + Stock total (calculado: disponible + reservado)
  + Estado visual (badge):
    - Verde "Stock Normal" si stock\_disponible ≥ umbral\_minimo\_stock
    - Amarillo "Stock Bajo" si stock\_disponible < umbral\_minimo\_stock y > 0
    - Rojo "Agotado" si stock\_disponible = 0
  + Acciones (botones Ver detalle, Editar, Ajustar stock, Eliminar según permisos)
* Los productos están ordenados alfabéticamente por nombre (orden por defecto).
* Paginación de 50 productos por página.

Búsqueda y Filtros

* Barra de búsqueda (texto libre):
  + Busca por nombre de producto o SKU
  + Actualización en tiempo real con debounce de 300 ms
  + Muestra "No se encontraron resultados" si no hay coincidencias
* Filtros disponibles (se pueden aplicar simultáneamente):
  + Por categoría (dropdown multiopción):
    - Pokemon
    - Yu-Gi-Oh
    - Magic
    - One Piece
    - Digimon
    - Accesorios
    - Juegos de Mesa
    - Opción "Todas las categorías" seleccionada por defecto
  + Por estado de stock (dropdown):
    - Todos
    - Stock Normal
    - Stock Bajo
    - Agotado
  + Por rango de precio (inputs numéricos):
    - Precio mínimo (≥0)
    - Precio máximo (≥precio mínimo)
* Botón "Limpiar filtros" que resetea todos los criterios a valores por defecto.
* Los filtros se aplican automáticamente al cambiar cualquier valor.
* La URL refleja los filtros aplicados para permitir compartir búsquedas (ej: /inventario?categoria=Pokemon&stock=bajo).

Acciones por Producto

* Ver detalle: Abre modal con información completa del producto (descripción, código de barras, imagen, historial reciente de movimientos).
* Editar: Disponible solo si el usuario tiene permiso "Editar inventario" o es Administrador.
* Ajustar stock: Disponible solo si el usuario tiene permiso "Editar inventario" o es Administrador.
* Eliminar: Disponible solo para Administradores y si stock\_disponible + stock\_reservado = 0.

Indicadores Visuales

* Badge de estado de stock con color correspondiente en cada fila.
* Tooltip al pasar mouse sobre badge que muestra: "Umbral mínimo: [X] unidades".
* Resaltado de fila al hacer hover para mejorar legibilidad.
* Ícono de advertencia junto a productos con stock disponible = 0 pero stock reservado > 0.

Rendimiento y UX

* Carga inicial no mayor a 2 segundos.
* Búsqueda y filtros responden en menos de 500 ms.
* Skeleton loaders durante carga de datos.
* Mensajes claros: "Cargando inventario...", "No hay productos que coincidan con tu búsqueda".

Tareas Técnicas

Backend (Django REST Framework)

* Endpoint: GET /api/inventario/
* Query parameters soportados:
  + search (texto libre para nombre o SKU)
  + categoria (array de strings)
  + estado\_stock (string: 'normal', 'bajo', 'agotado')
  + precio\_min (decimal)
  + precio\_max (decimal)
  + ordering (string: 'nombre', '-nombre', 'precio\_unitario', '-precio\_unitario', 'stock\_disponible', '-stock\_disponible')
  + page (int, para paginación)
* Serializer: ProductListSerializer con campos:
  + id, sku, nombre, categoria, precio\_unitario, stock\_disponible, stock\_reservado, stock\_total (calculado), umbral\_minimo\_stock, estado\_stock (calculado)
* Implementar filtros usando django-filter o lógica personalizada en queryset.
* Validaciones:
  + precio\_min ≥ 0
  + precio\_max ≥ precio\_min
  + categoria debe estar en lista permitida
* Cálculo de estado\_stock en el serializer:

python

**if** obj.stock\_disponible >= obj.umbral\_minimo\_stock:

**return** 'normal'

**elif** obj.stock\_disponible > 0:

**return** 'bajo'

**else**:

**return** 'agotado'

* Paginación estándar: 50 productos por página.
* Respuesta exitosa (200):

json

{

"count": 145,

"next": "http://api/inventario/?page=2",

"previous": **null**,

"results": [

{

"id": 1,

"sku": "PKM-001",

"nombre": "Pikachu VMAX",

"categoria": "Pokemon",

"precio\_unitario": 15000.00,

"stock\_disponible": 12,

"stock\_reservado": 3,

"stock\_total": 15,

"umbral\_minimo\_stock": 5,

"estado\_stock": "normal"

},

...

]

}

Frontend (React)

* Componente principal: InventoryPage.jsx
* Componentes auxiliares:
  + InventoryTable.jsx (tabla con productos)
  + InventoryFilters.jsx (barra de búsqueda y filtros)
  + ProductDetailModal.jsx (modal de detalle)
  + StockBadge.jsx (badge de estado)
* Estado local:
  + productos (array)
  + loading (boolean)
  + filters (objeto: search, categoria, estado\_stock, precio\_min, precio\_max)
  + page (number)
  + totalPages (number)
* Hooks:
  + useEffect para cargar productos al montar y cuando cambien filtros/página
  + useDebounce para búsqueda con delay de 300 ms
  + useSearchParams para sincronizar filtros con URL
* Lógica de filtros:
  + Al cambiar cualquier filtro, actualizar estado y hacer llamada GET con parámetros actualizados
  + Botón "Limpiar filtros" resetea todos los valores y recarga sin filtros
* Manejo de permisos:
  + Obtener permisos desde authService.getPermissions()
  + Mostrar/ocultar botones de Editar, Ajustar stock, Eliminar según permisos
* Paginación:
  + Componente Pagination con botones Anterior/Siguiente y números de página
  + Al cambiar página, scroll automático al inicio de la tabla

Configuración

* Paginación: 50 productos por página
* Debounce de búsqueda: 300 ms
* Timeout de carga: 10 segundos (mostrar error si excede)
* Formato de precio: separador de miles con punto (ej: $15.000)

Definición de Hecho (DoD)

Funcionalidad

* Todos los usuarios autenticados pueden acceder a la vista de inventario
* Tabla muestra productos con toda la información requerida
* Búsqueda por nombre y SKU funciona correctamente con debounce
* Todos los filtros se aplican correctamente y pueden combinarse
* Paginación funciona y muestra correctamente el número de páginas
* Badges de estado reflejan correctamente el nivel de stock
* URL refleja filtros aplicados y permite compartir búsquedas

Seguridad

* Validación de permisos para acciones de Editar, Ajustar y Eliminar
* Endpoint valida parámetros de entrada para evitar inyecciones
* Usuarios sin permiso no ven botones de acciones restringidas

Experiencia de Usuario

* Interfaz limpia, intuitiva y responsive
* Skeleton loaders durante carga de datos
* Mensajes claros de estado (cargando, sin resultados, error)
* Filtros y búsqueda responden rápidamente (< 500 ms)
* Tooltip informativos en badges de stock
* Hover en filas mejora legibilidad

Testing (Prioritarios)

* Búsqueda devuelve resultados correctos por nombre y SKU
* Cada filtro individual funciona correctamente
* Combinación de múltiples filtros devuelve resultados precisos
* Paginación muestra productos correctos en cada página
* Badges de stock muestran color correcto según nivel
* Botones de acciones aparecen/desaparecen según permisos
* URL se actualiza correctamente con filtros aplicados

Documentación

* Documentar endpoint con todos los parámetros soportados
* Documentar lógica de cálculo de estado\_stock
* Documentar estructura de respuesta paginada
* Documentar permisos requeridos para cada acción

Dependencias

* Modelo Producto completo y activo
* Sistema de autenticación y permisos implementado (US-001, US-003, US-024)
* Componentes base de UI (botones, inputs, badges, modals)

Notas Técnicas

* stock\_total es un campo calculado en el serializer, no se almacena en la base de datos.
* estado\_stock se calcula dinámicamente comparando stock\_disponible con umbral\_minimo\_stock.
* Los filtros se aplican en el backend mediante queryset de Django para optimizar rendimiento.
* La búsqueda por texto usa operadores \_\_icontains para búsqueda insensible a mayúsculas.
* Si un producto tiene stock\_disponible = 0 pero stock\_reservado > 0, el badge será "Agotado" pero con ícono de advertencia indicando que hay stock reservado.
* El umbral mínimo de stock por defecto es 5, pero puede configurarse por producto.

US-008: Consultar y Buscar Inventario con Filtros Avanzados

Como administrador o vendedor con permiso  
Quiero consultar el inventario completo y buscar productos específicos con filtros avanzados  
Para conocer el stock disponible, ubicar productos rápidamente y tomar decisiones de compra o venta

**Criterios de Aceptación**

**Acceso al Módulo**

1. Opción “Inventario” visible en menú de navegación para administradores y vendedores con permiso “Editar inventario”.
2. Vendedores sin permiso “Editar inventario” no ven esta opción (solo acceso desde POS).
3. Al hacer clic, redirige a /inventario.

**Visualización del Listado**

1. Pantalla muestra tabla con todos los productos del inventario.
2. Tabla incluye columnas:
   * Código/SKU del producto
   * Nombre del producto
   * Categoría (Pokémon, Yu-Gi-Oh, Magic, One Piece, Digimon, Accesorios, Juegos de Mesa)
   * Precio unitario (CLP)
   * Stock disponible
   * Stock reservado
   * Stock total (disponible + reservado)
   * Estado (badges visuales)
   * Acciones (Ver, Editar, Eliminar)
3. Productos se ordenan por defecto por nombre alfabéticamente.

**Badges y Estados Visuales**

1. Productos con stock bajo muestran badge rojo “Stock Bajo” si stock\_disponible <= umbral\_minimo configurado.
2. Productos con promoción activa muestran badge verde “En Promoción”.
3. Productos con stock reservado > 0 muestran badge azul “Con Reservas”.
4. Productos sin stock disponible (stock\_disponible = 0) muestran badge gris “Agotado”.
5. Múltiples badges pueden mostrarse simultáneamente en un mismo producto.

**Visualización de Precios y Promociones**

1. Si producto tiene promoción activa:
   * Muestra precio original tachado.
   * Muestra precio con descuento en color verde destacado.
   * Indica porcentaje o monto de descuento.
2. Si no tiene promoción, muestra solo precio normal en CLP.
3. Todos los precios con formato: $1.500, $15.000, $150.000.

**Barra de Búsqueda**

1. Barra de búsqueda en parte superior permite buscar por:
   * Nombre del producto
   * Código/SKU
   * Código de barras (si está registrado)
2. Búsqueda en tiempo real filtra resultados mientras el usuario escribe.
3. Búsqueda es case-insensitive (no distingue mayúsculas/minúsculas).
4. Muestra mensaje “No se encontraron productos” si búsqueda no arroja resultados.

**Filtros Avanzados**

1. Panel de filtros lateral o desplegable con opciones:
   * Filtro por categoría (multi-selección): Pokémon, Yu-Gi-Oh, Magic, One Piece, Digimon, Accesorios, Juegos de Mesa, Todos.
   * Filtro por estado de stock: Todos, Con stock, Stock bajo, Agotados, Con reservas.
   * Filtro por promoción: Todos, En promoción, Sin promoción.
   * Filtro por rango de precio: Desde $ \_\_\_ Hasta $ \_\_\_.
2. Filtros pueden aplicarse simultáneamente.
3. Botón “Limpiar filtros” resetea todos los filtros aplicados.
4. Contador visible muestra cantidad de resultados: “Mostrando X de Y productos”.

**Ordenamiento**

1. Columnas ordenables al hacer clic en encabezado:
   * Nombre (alfabético A-Z, Z-A)
   * Categoría (alfabético)
   * Precio (menor a mayor, mayor a menor)
   * Stock disponible (menor a mayor, mayor a menor)
   * Stock total (menor a mayor, mayor a menor)
2. Indicador visual muestra columna y dirección de ordenamiento actual.

**Paginación**

1. Tabla muestra máximo 50 productos por página por defecto.
2. Controles de paginación en parte inferior.
3. Opciones de resultados por página: 25, 50, 100, Todos.
4. Muestra: Página actual, total de páginas, botones Anterior/Siguiente.

**Ver Detalles de Producto**

1. Botón “Ver Detalles” en cada fila abre modal con información completa:
   * Código/SKU
   * Nombre completo
   * Descripción (si existe)
   * Categoría
   * Precio unitario actual (con promoción si aplica)
   * Stock disponible
   * Stock reservado
   * Stock total
   * Umbral mínimo de stock configurado
   * Código de barras (si existe)
   * Fecha de creación
   * Última actualización
   * Promoción activa (nombre, descuento, vigencia)
2. Modal incluye botón “Editar Producto” (solo para usuarios con permiso).
3. Modal incluye botón “Cerrar”.

**Acciones Rápidas**

1. Solo usuarios con permiso “Editar inventario” ven botones de Editar y Eliminar.
2. Vendedores sin permiso solo pueden ver detalles (solo lectura).
3. Administradores siempre ven todas las acciones.

**Exportación**

1. Botón “Exportar Inventario” permite descargar listado completo.
2. Opciones de exportación: CSV, Excel.
3. Archivo exportado incluye: código, nombre, categoría, precio, stock disponible, stock reservado, stock total.
4. Exportación respeta filtros y búsquedas activas.

**Vista de Stock Separado**

1. Columnas “Stock Disponible” y “Stock Reservado” claramente diferenciadas.
2. Stock disponible: productos listos para venta inmediata.
3. Stock reservado: productos apartados para clientes específicos (preventas).
4. Stock total = stock disponible + stock reservado (cálculo automático).
5. Tooltip en encabezados explica diferencia entre tipos de stock.

**Tareas Técnicas**

**Backend (Django)**

* Modelo Producto con campos:
  + codigo\_sku (CharField único, indexado)
  + nombre (CharField, indexado)
  + descripcion (TextField opcional)
  + categoria (CharField con choices predefinidas)
  + precio\_unitario (DecimalField para CLP)
  + stock\_disponible (IntegerField, default 0)
  + stock\_reservado (IntegerField, default 0)
  + umbral\_minimo (IntegerField, default 5)
  + codigo\_barras (CharField opcional, único)
  + promocion\_activa (ForeignKey a Promocion, nullable)
  + created\_at (DateTimeField auto)
  + updated\_at (DateTimeField auto)
* Endpoint GET /api/inventario/ que retorna listado de productos.
* Paginación con parámetros: page, page\_size.
* Búsqueda con parámetro: search (nombre, codigo\_sku, codigo\_barras).
* Filtros con parámetros:
  + categoria (multi-valor separado por comas)
  + estado (con\_stock, stock\_bajo, agotados, con\_reservas)
  + promocion (true/false)
  + precio\_min, precio\_max
* Ordenamiento con parámetro: ordering (nombre, categoria, precio\_unitario, stock\_disponible, stock\_total).
* Campo calculado stock\_total en serializer (stock\_disponible + stock\_reservado).
* Endpoint GET /api/inventario/{id}/ para detalles de producto específico.
* Endpoint GET /api/inventario/export/ que retorna CSV o Excel.
* Incluir en respuesta información de promoción activa si existe.
* Optimizar queries con select\_related para promocion\_activa.

**Frontend (React)**

* Componente InventoryPage con tabla de productos.
* Llamada GET a /api/inventario/ al montar componente.
* Componente ProductTable con todas las columnas especificadas.
* Barra de búsqueda con debounce (300ms).
* Panel de filtros avanzados con todos los filtros especificados.
* Lógica de aplicación de filtros múltiples simultáneos.
* Ordenamiento al hacer clic en encabezados.
* Componente Pagination reutilizable.
* Componente ProductBadges para badges visuales (Stock Bajo, En Promoción, Con Reservas, Agotado).
* Visualización diferenciada de precios con/sin promoción.
* Modal ProductDetailsModal con información completa.
* Función de exportación con descarga de archivo.
* Mostrar stock disponible y reservado en columnas separadas claramente.
* Cálculo visual de stock total.
* Renderizar botones de acción solo según permisos del usuario.
* Estados de carga y manejo de errores.
* Formatear precios en CLP consistentemente.
* Tooltips explicativos en encabezados de stock.

**Configuración**

* Choices de categorías en modelo: POKEMON, YU\_GI\_OH, MAGIC, ONE\_PIECE, DIGIMON, ACCESORIOS, JUEGOS\_MESA.
* Tamaño de página por defecto: PAGE\_SIZE = 50.
* Índices en base de datos para nombre, codigo\_sku, categoria para optimizar búsquedas.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuarios con permiso pueden acceder al módulo de inventario.
* Tabla muestra todos los productos correctamente con todas las columnas.
* Búsqueda filtra productos por nombre, SKU y código de barras en tiempo real.
* Filtros por categoría, estado, promoción y precio funcionan correctamente y pueden aplicarse simultáneamente.
* Ordenamiento por columnas funciona correctamente.
* Paginación muestra páginas correctamente con controles funcionales.
* Badges visuales se muestran correctamente según estado del producto.
* Productos en promoción muestran precio original tachado y precio con descuento.
* Precios se formatean en CLP correctamente.
* Stock disponible y stock reservado se muestran en columnas separadas.
* Stock total se calcula correctamente (disponible + reservado).
* Modal de detalles muestra información completa del producto.
* Exportación descarga archivo con datos correctos respetando filtros.
* Botones de acción solo visibles según permisos del usuario.
* Vendedores sin permiso solo pueden consultar (solo lectura).

**Seguridad**

* Endpoint de inventario valida autenticación.
* Vendedores sin permiso “Editar inventario” no ven botones de edición/eliminación.
* Validación de permisos en backend antes de permitir acciones.

**Experiencia de Usuario**

* Tabla clara, organizada y fácil de leer.
* Búsqueda rápida y responsiva.
* Filtros intuitivos y fáciles de aplicar.
* Badges visuales facilitan identificación rápida de estados.
* Diferenciación de precios con/sin promoción es clara.
* Stock disponible vs reservado es fácil de entender.
* Modal de detalles presenta información de manera organizada.
* Estados de carga muestran feedback apropiado.
* Mensajes de “sin resultados” son claros.

**Testing (Prioritarios)**

* Integración: usuario accede a inventario → visualiza productos correctamente con badges.
* Búsqueda: búsqueda por nombre filtra resultados correctamente.
* Filtros: filtro por categoría “Pokémon” muestra solo productos de esa categoría.
* Cálculo: stock total se calcula correctamente (disponible + reservado).

**Documentación**

* Endpoint GET /api/inventario/ documentado con parámetros y ejemplos.
* Estructura del modelo Producto documentada.
* Categorías disponibles documentadas.
* Parámetros de búsqueda, filtros y ordenamiento documentados.

**Dependencias**

* US-001 (Iniciar Sesión) completada.
* Sistema de permisos granulares funcional.
* Base de datos configurada (PostgreSQL o MySQL).

**Notas Técnicas**

* El campo stock\_total no se almacena en base de datos, se calcula en el serializer para evitar redundancia.
* El umbral\_minimo es configurable por producto para flexibilidad.
* Las categorías están hardcodeadas como choices en el modelo para consistencia.
* La separación entre stock\_disponible y stock\_reservado es fundamental para el sistema de reservas.
* El código de barras es opcional porque no todos los productos lo tienen inicialmente.
* Los índices en nombre, codigo\_sku y categoria son críticos para performance en búsquedas con inventario grande.
* El debounce en búsqueda previene requests excesivos (espera 300ms de inactividad antes de consultar).
* La exportación debe respetar filtros para permitir exportaciones específicas.
* Considerar agregar campo ubicacion\_fisica en futuras iteraciones para facilitar localización en el local.
* El precio\_unitario usa DecimalField con precisión suficiente para CLP (max\_digits=10, decimal\_places=0).

US-009: Agregar Producto Individual al Inventario

Como administrador o vendedor con permiso  
Quiero agregar nuevos productos al inventario de forma individual  
Para registrar mercadería nueva que llega al local y mantener el inventario actualizado

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con permiso “Editar inventario” pueden agregar productos.
2. Botón “Agregar Producto” visible en parte superior de la página de inventario.
3. Vendedores sin permiso no ven este botón.
4. Al hacer clic, redirige a formulario de creación /inventario/nuevo.

**Formulario de Creación**

1. Formulario muestra campos organizados en secciones:

**Información Básica**

* Código/SKU (requerido, único)
* Nombre del producto (requerido, mínimo 3 caracteres)
* Descripción (opcional, textarea)
* Categoría (requerido, dropdown con opciones predefinidas)

**Precio**

* Precio unitario en CLP (requerido, numérico, mayor a 0)

**Stock Inicial**

* Cantidad inicial (requerido, numérico, mayor o igual a 0)

**Configuración**

* Umbral mínimo de stock (requerido, numérico, default 5)
* Código de barras (opcional, numérico)

1. Todos los campos requeridos tienen indicador visual (asterisco rojo).
2. Botones “Guardar Producto” y “Cancelar” al final del formulario.

**Categorías Disponibles**

1. Dropdown de categoría incluye opciones:
   * Pokémon
   * Yu-Gi-Oh!
   * Magic: The Gathering
   * One Piece TCG
   * Digimon TCG
   * Accesorios
   * Juegos de Mesa
2. Campo categoría es obligatorio (no permite valor vacío).

**Validaciones del Formulario**

1. Valida que código/SKU sea único en el sistema.
2. Valida que nombre no esté vacío (mínimo 3 caracteres).
3. Valida que precio unitario sea numérico y mayor a 0.
4. Valida que cantidad inicial sea numérica y mayor o igual a 0.
5. Valida que umbral mínimo sea numérico y mayor a 0.
6. Valida formato de código de barras si se ingresa (numérico, opcional).
7. Muestra mensajes de error específicos bajo cada campo con problema.
8. Validación en tiempo real mientras el usuario completa el formulario.

**Generación Automática de SKU (Opcional)**

1. Botón “Generar SKU” que crea código único automáticamente.
2. SKU generado sigue patrón: [CATEGORIA]-[TIMESTAMP]-[RANDOM] (ej: POK-20251015-A3F9).
3. Usuario puede modificar SKU generado si lo desea.
4. Sistema valida unicidad antes de guardar.

**Escaneo de Código de Barras (Opcional)**

1. Si hay lector de código de barras conectado, campo “Código de barras” detecta input automáticamente.
2. Al escanear, completa el campo sin intervención manual.
3. Usuario puede ingresar código manualmente si no tiene lector.

**Creación del Producto**

1. Al hacer clic en “Guardar Producto”, valida todos los campos.
2. Crea registro en tabla Producto con todos los datos.
3. Establece stock\_disponible = cantidad inicial ingresada.
4. Establece stock\_reservado = 0 (sin reservas inicialmente).
5. Establece promocion\_activa = null (sin promoción inicial).
6. Establece created\_at con timestamp automático.
7. Genera ID único del producto automáticamente.

**Confirmación y Redirección**

1. Muestra mensaje de éxito: “Producto [nombre] agregado exitosamente al inventario”.
2. Redirige al listado de inventario donde el nuevo producto aparece visible.
3. Nuevo producto tiene badge “Nuevo” visible por 24 horas.
4. Producto aparece destacado visualmente (fondo verde claro temporal).

**Cancelación**

1. Botón “Cancelar” redirige al listado de inventario sin guardar.
2. Si hay cambios sin guardar, muestra confirmación: “¿Descartar cambios sin guardar?”
3. Opciones: “Continuar editando” o “Descartar”.

**Auditoría**

1. Registra creación de producto en AuditLog con:
   * Timestamp
   * Usuario que agregó el producto
   * Acción: 'producto\_creado'
   * Resultado: true
   * Detalles: código SKU, nombre, categoría, cantidad inicial

**Tareas Técnicas**

**Backend (Django)**

* Modelo Producto ya creado en US-008.
* Endpoint POST /api/inventario/ accesible para usuarios con permiso “Editar inventario”.
* Validaciones de campos requeridos y unicidad para codigo\_sku y codigo\_barras.
* Validación de valores numéricos positivos.
* Serializer ProductoCreateSerializer con validaciones personalizadas.
* Lógica de inicialización: stock\_reservado = 0, promocion\_activa = null.
* Registro en AuditLog después de creación exitosa.
* Endpoint GET /api/inventario/generar-sku/ que retorna SKU único generado automáticamente.
* Permisos: custom permission que verifica “Editar inventario”.

**Frontend (React)**

* Componente AddProductPage con formulario completo.
* Organizar formulario en secciones visuales claras.
* Validaciones en tiempo real de todos los campos.
* Validación de unicidad de SKU consultando al backend.
* Dropdown de categorías con opciones predefinidas.
* Campo numérico para precio con formato CLP mientras el usuario escribe.
* Campo numérico para cantidades (stock inicial, umbral).
* Botón “Generar SKU” con llamada a endpoint correspondiente.
* Detección de lector de código de barras en campo correspondiente.
* Modal de confirmación para cancelar con cambios sin guardar.
* Llamada POST a /api/inventario/ al enviar formulario.
* Mensajes de error específicos del backend bajo cada campo.
* Mensaje de éxito después de creación.
* Redirección al listado después de éxito.
* Validación que previene envío de formulario si hay errores.
* Estados de carga durante envío del formulario.

**Configuración**

* Choices de categorías consistentes entre backend y frontend.
* Permisos en endpoint: IsAuthenticated y custom CanEditInventory.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo usuarios con permiso “Editar inventario” pueden acceder al formulario.
* Formulario muestra todos los campos especificados correctamente.
* Validaciones en tiempo real funcionan correctamente.
* Código/SKU único se valida correctamente.
* Categorías se muestran en dropdown correctamente.
* Precio acepta solo valores numéricos positivos en CLP.
* Cantidad inicial acepta valores numéricos mayores o igual a 0.
* Umbral mínimo tiene valor por defecto de 5.
* Generador de SKU crea códigos únicos correctamente.
* Lector de código de barras completa campo automáticamente (si está conectado).
* Producto se crea correctamente en base de datos.
* Stock inicial se asigna a stock\_disponible correctamente.
* Stock reservado se inicializa en 0.
* Mensaje de éxito se muestra después de crear producto.
* Redirección al listado funciona correctamente.
* Nuevo producto aparece en listado con badge “Nuevo”.
* Botón cancelar muestra confirmación si hay cambios sin guardar.
* Producto creado queda registrado en auditoría.

**Seguridad**

* Endpoint de creación valida autenticación y permiso “Editar inventario”.
* Vendedores sin permiso no pueden acceder al formulario.
* Validaciones se ejecutan en backend (no solo frontend).
* SKU duplicado es rechazado por backend.

**Experiencia de Usuario**

* Formulario bien organizado por secciones.
* Campos requeridos tienen indicadores visuales claros.
* Validaciones en tiempo real guían al usuario.
* Mensajes de error específicos y útiles.
* Generador de SKU facilita creación de códigos únicos.
* Formato de precio se muestra en CLP mientras se escribe.
* Confirmación de cancelación previene pérdida accidental de datos.
* Mensaje de éxito claro y visible.
* Redirección después de éxito es fluida.

**Testing (Prioritarios)**

* Integración: usuario con permiso crea producto → producto aparece en inventario con datos correctos.
* Validación: SKU duplicado es rechazado.
* Validación: precio negativo es rechazado.
* Seguridad: vendedor sin permiso no puede acceder al formulario.

**Documentación**

* Endpoint POST /api/inventario/ documentado con ejemplos.
* Campos requeridos y opcionales documentados.
* Validaciones de cada campo documentadas.
* Categorías disponibles documentadas.

**Dependencias**

* US-008 (Consultar Inventario) completada con modelo Producto.
* Sistema de permisos granulares funcional.
* Modelo AuditLog disponible.

**Notas Técnicas**

* El código/SKU debe ser único para evitar duplicación de productos en inventario.
* El SKU generado automáticamente usa patrón que incluye categoría para facilitar identificación visual.
* El stock inicial puede ser 0 para productos que se registran antes de recibir mercadería.
* El umbral mínimo de 5 es un default razonable pero configurable por producto según rotación.
* El código de barras es opcional porque no todos los productos lo tienen (especialmente cartas sueltas).
* La detección de lector de código de barras se puede implementar escuchando eventos de teclado con patrón específico.
* El badge “Nuevo” se calcula comparando created\_at con timestamp actual (< 24 horas).
* El formato de precio debe permitir solo números enteros para CLP (sin decimales).
* Considerar agregar campo “Proveedor” en futuras iteraciones para trazabilidad de compras.
* El precio\_unitario en modelo usa DecimalField pero para CLP no necesita decimales.
* La auditoría de creación permite rastrear quién agregó cada producto y cuándo.

US-010: Editar Producto Existente en Inventario

Como administrador o vendedor con permiso  
Quiero editar la información de productos existentes en el inventario  
Para actualizar precios, corregir datos erróneos o modificar configuraciones según las necesidades del negocio

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con permiso “Editar inventario” pueden editar productos.
2. Botón “Editar” visible en cada fila del listado de inventario (US-008).
3. Vendedores sin permiso no ven este botón.
4. Al hacer clic, redirige a formulario de edición /inventario/editar/{id}.

**Carga de Datos Existentes**

1. Al abrir el formulario, carga automáticamente datos actuales del producto.
2. Formulario muestra todos los campos editables precargados:
   * Código/SKU (no editable, solo visualización)
   * Nombre del producto (editable)
   * Descripción (editable)
   * Categoría (editable, dropdown)
   * Precio unitario en CLP (editable)
   * Umbral mínimo de stock (editable)
   * Código de barras (editable)
3. Campo código/SKU se muestra pero está deshabilitado (no puede cambiarse).
4. Stock disponible y stock reservado NO son editables desde este formulario (se muestran solo como información).

**Información de Stock (Solo Lectura)**

1. Sección “Stock Actual” muestra información no editable:
   * Stock disponible: [cantidad]
   * Stock reservado: [cantidad]
   * Stock total: [cantidad]
2. Mensaje informativo: “El stock se modifica mediante ajustes de inventario (US-012) o ventas/reservas automáticamente.”
3. Link “Realizar ajuste de stock” redirige a US-012 para ese producto específico.

**Edición de Información Básica**

1. Usuario puede modificar nombre del producto.
2. Usuario puede modificar descripción.
3. Usuario puede cambiar categoría mediante dropdown.
4. Usuario puede modificar precio unitario en CLP.
5. Usuario puede modificar umbral mínimo de stock.
6. Usuario puede agregar/modificar/eliminar código de barras.

**Validaciones del Formulario**

1. Valida que nombre no esté vacío (mínimo 3 caracteres).
2. Valida que precio unitario sea numérico y mayor a 0.
3. Valida que umbral mínimo sea numérico y mayor a 0.
4. Valida formato de código de barras si se ingresa (numérico).
5. Valida que código de barras sea único (no puede duplicar el de otro producto).
6. Muestra mensajes de error específicos bajo cada campo con problema.
7. Validación en tiempo real mientras el usuario edita.

**Visualización de Promoción Activa**

1. Si el producto tiene promoción activa, muestra sección informativa:
   * Nombre de la promoción
   * Tipo de descuento (porcentaje/monto fijo)
   * Valor del descuento
   * Precio original vs precio con descuento
   * Vigencia (fecha inicio - fecha fin)
2. Mensaje: “Este producto tiene una promoción activa. Gestiona promociones desde el módulo correspondiente.”
3. Link “Ver promociones” redirige al módulo de gestión de descuentos (EPIC-07).
4. No permite modificar precio mientras tiene promoción activa (solo desde módulo de promociones).

**Actualización del Producto**

1. Botón “Guardar Cambios” envía el formulario.
2. Actualiza campos modificados en tabla Producto.
3. Actualiza campo updated\_at con timestamp automático.
4. NO modifica created\_at (se mantiene original).
5. NO modifica stock\_disponible ni stock\_reservado desde este formulario.

**Confirmación y Redirección**

1. Muestra mensaje de éxito: “Producto [nombre] actualizado exitosamente.”
2. Redirige al listado de inventario.
3. Producto actualizado muestra badge “Actualizado” por 1 hora.
4. Cambios son visibles inmediatamente en el listado.

**Cancelación**

1. Botón “Cancelar” redirige al listado sin guardar.
2. Si hay cambios sin guardar, muestra confirmación: “¿Descartar cambios sin guardar?”
3. Opciones: “Continuar editando” o “Descartar”.

**Auditoría**

1. Registra edición de producto en AuditLog con:
   * Timestamp
   * Usuario que editó el producto
   * Acción: 'producto\_editado'
   * Resultado: true
   * Detalles: código SKU, campos modificados (valores anteriores y nuevos)
2. Solo registra campos que realmente cambiaron (comparación antes/después).

**Tareas Técnicas**

**Backend (Django)**

* Endpoint GET /api/inventario/{id}/ para obtener datos del producto (ya existe en US-008).
* Endpoint PUT/PATCH /api/inventario/{id}/ para actualizar producto.
* Permisos: custom permission que verifica “Editar inventario”.
* Validaciones de campos editables.
* Validación de unicidad para codigo\_barras (excluyendo el código actual del producto).
* Lógica que previene modificación de codigo\_sku (campo read-only en serializer).
* Lógica que previene modificación de stock\_disponible y stock\_reservado desde este endpoint.
* Actualización automática de updated\_at.
* Comparación de valores anteriores vs nuevos para auditoría.
* Registro en AuditLog solo con campos modificados.
* Serializer ProductoUpdateSerializer que excluye campos no editables.

**Frontend (React)**

* Componente EditProductPage que recibe ID del producto como parámetro.
* Llamada GET a /api/inventario/{id}/ para cargar datos al montar componente.
* Precargar todos los campos editables con datos actuales.
* Deshabilitar campo codigo\_sku (solo visualización).
* Mostrar sección de stock actual como solo lectura.
* Validaciones en tiempo real de campos editables.
* Validación de unicidad de código de barras consultando al backend.
* Mostrar sección de promoción activa si existe (solo información, no editable).
* Modal de confirmación para cancelar con cambios sin guardar.
* Llamada PUT/PATCH a /api/inventario/{id}/ al guardar.
* Detectar campos modificados comparando con valores originales.
* Mensajes de error específicos del backend.
* Mensaje de éxito después de actualización.
* Redirección al listado después de éxito.
* Link “Realizar ajuste de stock” que redirige a formulario de ajustes.
* Estados de carga durante envío del formulario.

**Configuración**

* Permisos en endpoint: IsAuthenticated y custom CanEditInventory.
* Serializer con campos read-only: codigo\_sku, stock\_disponible, stock\_reservado, created\_at.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo usuarios con permiso “Editar inventario” pueden acceder al formulario.
* Formulario carga correctamente datos actuales del producto.
* Código/SKU se muestra pero no es editable.
* Stock disponible y reservado se muestran como solo lectura.
* Nombre, descripción, categoría, precio y umbral pueden modificarse.
* Código de barras puede agregarse/modificarse/eliminarse.
* Validaciones en tiempo real funcionan correctamente.
* Código de barras duplicado es rechazado.
* Precio negativo es rechazado.
* Si producto tiene promoción activa, se muestra información correctamente.
* Link a ajuste de stock redirige correctamente.
* Cambios se guardan correctamente en base de datos.
* Campo updated\_at se actualiza automáticamente.
* Campo created\_at NO se modifica.
* Mensaje de éxito se muestra después de actualizar.
* Redirección al listado funciona correctamente.
* Producto actualizado muestra badge “Actualizado” temporalmente.
* Botón cancelar muestra confirmación si hay cambios sin guardar.
* Auditoría registra solo campos que cambiaron.

**Seguridad**

* Endpoint de edición valida autenticación y permiso “Editar inventario”.
* Vendedores sin permiso no pueden acceder al formulario.
* Validaciones se ejecutan en backend (no solo frontend).
* Código/SKU no puede modificarse desde el frontend ni backend (campo protegido).
* Stock no puede modificarse desde este endpoint (solo desde ajustes/ventas/reservas).

**Experiencia de Usuario**

* Formulario precarga datos actuales correctamente.
* Campos no editables están claramente identificados.
* Validaciones en tiempo real guían al usuario.
* Mensajes de error específicos y útiles.
* Sección de stock es clara sobre por qué no es editable.
* Link a ajuste de stock facilita navegación.
* Información de promoción activa es clara.
* Confirmación de cancelación previene pérdida accidental de cambios.
* Mensaje de éxito claro y visible.

**Testing (Prioritarios)**

* Integración: usuario edita precio de producto → precio se actualiza correctamente en inventario.
* Validación: código de barras duplicado es rechazado.
* Seguridad: código/SKU no puede modificarse ni desde frontend ni backend.
* Auditoría: solo campos modificados se registran en log.

**Documentación**

* Endpoint PUT/PATCH /api/inventario/{id}/ documentado con ejemplos.
* Campos editables y no editables documentados claramente.
* Validaciones de cada campo documentadas.

**Dependencias**

* US-008 (Consultar Inventario) completada.
* US-009 (Agregar Producto) completada.
* Modelo Producto con todos los campos necesarios.
* Modelo AuditLog disponible.

**Notas Técnicas**

* El código/SKU no es editable porque es el identificador principal del producto en el sistema y modificarlo causaría inconsistencias en registros históricos.
* El stock NO se edita desde este formulario para mantener trazabilidad completa de movimientos (solo mediante ajustes, ventas o reservas).
* La auditoría debe registrar valores anteriores y nuevos de cada campo modificado para permitir rastreo completo de cambios.
* Si el producto tiene promoción activa, el precio mostrado en el inventario será el con descuento, pero el campo precio\_unitario almacena el precio base.
* El badge “Actualizado” se calcula comparando updated\_at con timestamp actual (< 1 hora).
* La validación de código de barras debe excluir el código actual del producto para permitir mantener el mismo código.
* Considerar confirmación adicional al modificar precio de productos con alta rotación o stock bajo.
* El campo updated\_at se actualiza automáticamente en Django usando auto\_now=True.
* La comparación de campos modificados puede implementarse en el frontend antes de enviar (enviar solo campos modificados) o en el backend (comparar con valores actuales antes de actualizar).

US-011: Ajuste Manual de Stock en Inventario

Como administrador o vendedor con permiso  
Quiero realizar ajustes manuales de stock (agregar o quitar cantidades)  
Para corregir errores de inventario, registrar pérdidas, daños, o ingresos de mercadería que no sean ventas

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con permiso “Editar inventario” pueden realizar ajustes.
2. En el listado de inventario (US-009), botón “Ajustar Stock” visible en cada fila.
3. Desde formulario de edición (US-010), link “Realizar ajuste de stock” disponible.
4. Vendedores sin permiso no ven estas opciones.
5. Al hacer clic, abre modal de ajuste de inventario.

**Modal de Ajuste de Inventario**

1. Modal muestra información del producto:
   * Código/SKU
   * Nombre del producto
   * Categoría
   * Stock disponible actual
   * Stock reservado actual (no modificable)
   * Stock total actual
2. Formulario de ajuste con campos:
   * Tipo de ajuste (requerido, radio buttons): Entrada (Agregar stock) / Salida (Quitar stock)
   * Cantidad (requerido, numérico, mayor a 0)
   * Motivo del ajuste (requerido, dropdown con opciones predefinidas)
   * Observaciones adicionales (opcional, textarea)
   * Botones “Confirmar Ajuste” y “Cancelar”

**Tipos de Ajuste**

1. Entrada (Agregar stock): incrementa stock\_disponible.
2. Salida (Quitar stock): decrementa stock\_disponible.
3. Ajustes solo afectan stock\_disponible, NO stock\_reservado.
4. Stock reservado se modifica únicamente mediante sistema de reservas (EPIC-04).

**Motivos Predefinidos**

1. Para ajustes de tipo Entrada:
   * Nueva compra a proveedor
   * Devolución de cliente
   * Corrección de inventario (conteo físico)
   * Recuperación de producto dañado
   * Transferencia desde otra ubicación
   * Otro (requiere especificar en observaciones)
2. Para ajustes de tipo Salida:
   * Producto dañado/defectuoso
   * Producto extraviado
   * Promoción/Regalo
   * Uso interno
   * Corrección de inventario (conteo físico)
   * Otro (requiere especificar en observaciones)
3. Si selecciona “Otro”, campo observaciones se vuelve requerido.

**Validaciones del Formulario**

1. Valida que tipo de ajuste esté seleccionado (Entrada o Salida).
2. Valida que cantidad sea numérica y mayor a 0.
3. Valida que motivo esté seleccionado.
4. Valida que observaciones no estén vacías si motivo es “Otro”.
5. Para ajustes de tipo Salida, valida que cantidad no exceda stock\_disponible actual.
6. Muestra mensaje de error: “No puedes quitar más stock del disponible. Stock actual: [cantidad]”.
7. No permite stock\_disponible negativo.

**Vista Previa del Ajuste**

1. Al ingresar cantidad y tipo, muestra cálculo en tiempo real:
   * Stock disponible actual: [cantidad]
   * Ajuste: + [cantidad] o - [cantidad]
   * Stock disponible resultante: [cantidad]
2. Resalta en verde si es entrada, en rojo si es salida.
3. Muestra advertencia si ajuste de salida deja stock por debajo del umbral mínimo.

**Confirmación del Ajuste**

1. Al hacer clic en “Confirmar Ajuste”, requiere confirmación adicional.
2. Modal de confirmación muestra resumen:
   * Producto: [nombre]
   * Tipo: Entrada/Salida
   * Cantidad: [cantidad]
   * Stock resultante: [cantidad]
   * Motivo: [motivo seleccionado]
   * Opciones: “Sí, confirmar ajuste” o “Cancelar”

**Ejecución del Ajuste**

1. Si tipo es Entrada: stock\_disponible = stock\_disponible + cantidad.
2. Si tipo es Salida: stock\_disponible = stock\_disponible - cantidad.
3. Actualiza campo updated\_at del producto con timestamp automático.
4. Crea registro en tabla MovimientoInventario con todos los detalles.
5. NO modifica stock\_reservado bajo ninguna circunstancia.

**Registro de Movimiento**

1. Cada ajuste crea registro en tabla MovimientoInventario con:
   * producto (ForeignKey al Producto)
   * tipo\_movimiento: 'entrada' o 'salida'
   * cantidad
   * stock\_anterior (stock\_disponible antes del ajuste)
   * stock\_nuevo (stock\_disponible después del ajuste)
   * motivo (seleccionado del dropdown)
   * observaciones (si se ingresaron)
   * usuario (ForeignKey al usuario que realizó el ajuste)
   * fecha\_hora (timestamp automático)
2. Registros de movimientos son inmutables (no editables ni eliminables).

**Confirmación y Actualización**

1. Muestra mensaje de éxito: “Ajuste realizado exitosamente. Stock actualizado”.
2. Cierra modal automáticamente.
3. Actualiza listado de inventario mostrando nuevo stock.
4. Producto ajustado muestra badge “Ajustado” temporalmente (10 minutos).

**Alertas Post-Ajuste**

1. Si ajuste de salida deja stock\_disponible por debajo del umbral\_minimo:
   * Muestra alerta: “¡Advertencia! Este producto ahora tiene stock bajo”.
   * Badge “Stock Bajo” aparece inmediatamente en el listado.
2. Si ajuste de entrada lleva stock por encima del umbral, remueve badge “Stock Bajo”.

**Auditoría**

1. Registra ajuste en AuditLog con:
   * Timestamp
   * Usuario que realizó el ajuste
   * Acción: 'ajuste\_inventario'
   * Resultado: true
   * Detalles: código SKU, tipo de ajuste, cantidad, motivo, stock anterior, stock nuevo

**Tareas Técnicas**

**Backend (Django)**

* Modelo MovimientoInventario con campos:
  + producto (ForeignKey a Producto)
  + tipo\_movimiento (CharField con choices: 'entrada', 'salida')
  + cantidad (IntegerField positivo)
  + stock\_anterior (IntegerField)
  + stock\_nuevo (IntegerField)
  + motivo (CharField)
  + observaciones (TextField opcional)
  + usuario (ForeignKey a Usuario)
  + fecha\_hora (DateTimeField auto\_now\_add)
* Endpoint POST /api/inventario/{id}/ajuste/ para realizar ajustes.
* Permisos: custom permission que verifica “Editar inventario”.
* Validaciones:
  + Cantidad mayor a 0.
  + Para salidas: cantidad <= stock\_disponible.
  + Motivo no vacío.
  + Observaciones no vacías si motivo es “Otro”.
* Lógica transaccional (atomic) para garantizar consistencia:
  + Obtener stock\_anterior.
  + Calcular stock\_nuevo según tipo de ajuste.
  + Actualizar stock\_disponible del producto.
  + Crear registro en MovimientoInventario.
  + Crear registro en AuditLog.
* Endpoint GET /api/inventario/{id}/movimientos/ para consultar historial de movimientos (futuro).
* Asegurar que transacción sea atómica (todo o nada) para prevenir inconsistencias.

**Frontend (React)**

* Componente AjusteInventarioModal que recibe producto como prop.
* Mostrar información actual del producto claramente.
* Formulario con tipo de ajuste (radio buttons), cantidad, motivo y observaciones.
* Dropdown de motivos dinámico según tipo de ajuste seleccionado.
* Validaciones en tiempo real.
* Vista previa de cálculo que muestra stock resultante.
* Resaltar en verde (entrada) o rojo (salida) el ajuste.
* Mostrar advertencia si stock resultante está por debajo del umbral.
* Validación: salida no puede exceder stock disponible.
* Modal de confirmación adicional con resumen del ajuste.
* Llamada POST a /api/inventario/{id}/ajuste/ al confirmar.
* Mensaje de éxito después de ajuste.
* Cerrar modal automáticamente después de éxito.
* Actualizar listado de inventario sin recargar página completa.
* Mostrar alertas visuales si stock queda bajo después del ajuste.

**Configuración**

* Choices de motivos en constantes del backend.
* Configurar transaction.atomic en Django para ajustes.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo usuarios con permiso “Editar inventario” pueden realizar ajustes.
* Modal se abre correctamente desde listado o formulario de edición.
* Información del producto se muestra correctamente.
* Tipo de ajuste (Entrada/Salida) se selecciona correctamente.
* Motivos se muestran dinámicamente según tipo de ajuste.
* Campo observaciones se vuelve requerido si motivo es “Otro”.
* Vista previa calcula stock resultante correctamente en tiempo real.
* Validación impide salidas mayores al stock disponible.
* Modal de confirmación muestra resumen correcto del ajuste.
* Ajuste de entrada incrementa stock\_disponible correctamente.
* Ajuste de salida decrementa stock\_disponible correctamente.
* Stock\_reservado NO se modifica bajo ninguna circunstancia.
* Registro en MovimientoInventario se crea correctamente con todos los datos.
* Campo updated\_at del producto se actualiza.
* Mensaje de éxito se muestra después de ajuste.
* Listado se actualiza mostrando nuevo stock.
* Alertas de stock bajo aparecen/desaparecen según corresponda.
* Auditoría registra ajuste correctamente.
* Transacción es atómica (todo o nada).

**Seguridad**

* Endpoint de ajuste valida autenticación y permiso “Editar inventario”.
* Vendedores sin permiso no pueden realizar ajustes.
* Validaciones se ejecutan en backend (no solo frontend).
* No permite stock\_disponible negativo.
* Registros de MovimientoInventario son inmutables.

**Experiencia de Usuario**

* Modal claro y fácil de usar.
* Diferencia entre entrada y salida es obvia.
* Motivos predefinidos cubren casos comunes.
* Vista previa ayuda a visualizar resultado del ajuste.
* Colores (verde/rojo) comunican claramente el tipo de ajuste.
* Advertencias claras y oportunas.
* Confirmación adicional previene errores accidentales.
* Mensajes de éxito y error claros.
* Actualización del listado es fluida.

**Testing (Prioritarios)**

* Integración: usuario realiza ajuste de entrada → stock se incrementa correctamente.
* Integración: usuario realiza ajuste de salida → stock se decrementa correctamente.
* Validación: salida mayor al stock disponible es rechazada.
* Transacción: si falla registro en MovimientoInventario, stock no se modifica (atomicidad).

**Documentación**

* Endpoint POST /api/inventario/{id}/ajuste/ documentado con ejemplos.
* Motivos predefinidos documentados para entradas y salidas.
* Estructura de MovimientoInventario documentada.

**Dependencias**

* US-009 (Consultar Inventario) completada.
* US-010 (Agregar Producto) completada.
* US-011 (Editar Producto) completada.
* Modelo Producto con campos stock\_disponible y stock\_reservado.
* Modelo AuditLog disponible.

**Notas Técnicas**

* La separación entre stock\_disponible y stock\_reservado es crítica: ajustes manuales solo afectan stock\_disponible.
* La tabla MovimientoInventario mantiene historial completo de todos los cambios de stock para auditoría y trazabilidad.
* Los registros deben ser inmutables para garantizar integridad de auditoría.
* La transacción atómica es fundamental: si falla algún paso (actualizar producto, crear movimiento, crear auditoría), todo debe revertirse.
* El stock\_anterior y stock\_nuevo en MovimientoInventario permiten verificar consistencia y detectar anomalías.
* Los motivos predefinidos facilitan reportes posteriores (ej: “¿Cuánto stock se perdió por daños este mes?”).
* El campo observaciones es especialmente útil para motivo “Otro” o para detalles adicionales.
* Considerar agregar campo “referencia\_documento” en futuras iteraciones para vincular ajustes con facturas o documentos externos.
* El badge “Ajustado” temporal ayuda a identificar visualmente qué productos fueron modificados recientemente.
* La advertencia de stock bajo después de ajuste de salida es crítica para prevenir quiebres de stock.
* El campo fecha\_hora en MovimientoInventario debe usar timezone-aware datetime para consistencia.
* Considerar implementar restricción adicional: ajustes de salida masivos (> 50% del stock) requieren confirmación de administrador.

US-012: Eliminar Producto del Inventario (MVP Simplificado)

Como administrador  
Quiero eliminar productos del inventario que ya no se venden o fueron registrados por error  
Para mantener el catálogo limpio y organizado sin productos obsoletos

**Criterios de Aceptación**

**Acceso al Módulo**

1. Solo usuarios con rol Administrador pueden eliminar productos.
2. Botón "Eliminar" visible en cada fila del listado de inventario solo para administradores.
3. Vendedores con permiso "Editar inventario" NO pueden eliminar (solo administradores).
4. Botón tiene ícono de basura en color rojo.

**Restricciones de Eliminación**

1. NO se puede eliminar producto si tiene stock\_disponible > 0.
2. NO se puede eliminar producto si tiene stock\_reservado > 0.
3. NO se puede eliminar producto si tiene movimientos de inventario registrados.
4. Mensaje de error: "No se puede eliminar este producto porque tiene stock disponible/reservado. Realiza ajustes para dejar stock en 0 primero".
5. Solo se pueden eliminar productos con stock total = 0 y sin historial de movimientos.

**Confirmación de Eliminación**

1. Al hacer clic en "Eliminar", muestra modal de confirmación.
2. Modal incluye:
   * Advertencia en rojo: "⚠️ Esta acción es permanente y no se puede deshacer".
   * Información del producto: código SKU, nombre, categoría.
   * Confirmación: "Stock disponible: 0, Stock reservado: 0".
   * Input de texto: "Escribe ELIMINAR para confirmar".
   * Botones: "Cancelar" (gris) y "Eliminar permanentemente" (rojo, deshabilitado hasta escribir correctamente).

**Proceso de Eliminación**

1. Valida que el texto ingresado sea exactamente "ELIMINAR" (case-sensitive).
2. Al confirmar, elimina el registro de la tabla Producto (eliminación física).
3. Eliminación es permanente, no se puede recuperar.
4. Proceso es atómico (transaccional).

**Confirmación Post-Eliminación**

1. Muestra mensaje de éxito: "Producto [nombre] eliminado permanentemente del inventario".
2. Listado de inventario se actualiza automáticamente sin el producto eliminado.
3. No hay opción de deshacer.

**Auditoría**

1. Registra eliminación en AuditLog con:
   * Timestamp
   * Usuario administrador que eliminó
   * Acción: 'producto\_eliminado'
   * Resultado: true
   * Detalles: código SKU, nombre, categoría del producto eliminado

**Tareas Técnicas**

**Backend (Django)**

* Endpoint DELETE /api/inventario/{id}/ accesible solo para administradores.
* Permisos: custom permission que verifica rol Administrador (no solo permiso "Editar inventario").
* Validaciones antes de eliminar:
  + Verificar stock\_disponible = 0.
  + Verificar stock\_reservado = 0.
  + Verificar que no existen registros en MovimientoInventario para este producto.
* Retornar error 400 con mensaje específico si no cumple validaciones.
* Eliminación física del registro (producto.delete()).
* Registro en AuditLog antes de eliminar (guardar datos del producto).
* Usar transaction.atomic para garantizar atomicidad.

**Frontend (React)**

* Botón "Eliminar" en cada fila del listado solo para administradores.
* Modal ConfirmDeleteModal con advertencia y requisitos.
* Input de texto para escribir "ELIMINAR".
* Validar texto ingresado en tiempo real.
* Deshabilitar botón "Eliminar permanentemente" hasta que texto sea correcto.
* Llamada DELETE a /api/inventario/{id}/.
* Mensaje de error específico si backend rechaza (tiene stock o movimientos).
* Mensaje de éxito después de eliminación.
* Actualizar listado de inventario sin recargar página completa (remover fila).
* Cerrar modal automáticamente después de éxito.

**Configuración**

* Permisos en endpoint: IsAuthenticated y custom IsAdministrator.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo administradores ven botón de eliminar.
* Vendedores con permiso "Editar inventario" NO pueden eliminar.
* Modal de confirmación muestra advertencia clara.
* Input de texto "ELIMINAR" funciona correctamente.
* Botón está deshabilitado hasta escribir correctamente.
* Validación rechaza eliminación si hay stock disponible > 0.
* Validación rechaza eliminación si hay stock reservado > 0.
* Validación rechaza eliminación si hay movimientos de inventario.
* Mensaje de error es claro y específico según el motivo.
* Producto se elimina permanentemente de base de datos.
* Listado se actualiza correctamente después de eliminación.
* Auditoría registra eliminación con datos del producto.

**Seguridad**

* Endpoint valida que usuario sea Administrador (no solo con permiso "Editar inventario").
* Vendedores no pueden acceder al endpoint de eliminación.
* Validaciones se ejecutan en backend antes de eliminar.
* Eliminación es transaccional (atomic).
* No se puede eliminar producto con stock o historial.

**Experiencia de Usuario**

* Advertencia de acción permanente es muy clara.
* Requisito de escribir "ELIMINAR" previene eliminaciones accidentales.
* Mensajes de error son claros y orientan sobre qué hacer.
* Proceso de confirmación es suficientemente "difícil" para prevenir errores.
* Feedback después de eliminación es inmediato.

**Testing (Prioritarios)**

* Test de seguridad: vendedor no puede eliminar productos.
* Test de validación: producto con stock > 0 no puede eliminarse.
* Test de integración: administrador elimina producto con stock 0 → producto desaparece del inventario.

**Documentación**

* Endpoint DELETE /api/inventario/{id}/ documentado con restricciones.
* Validaciones de eliminación documentadas.
* Restricción de permisos (solo Administrador) documentada.

**Dependencias**

* US-009 (Consultar Inventario) completada.
* US-011 (Ajustes de Inventario) para llevar stock a 0 antes de eliminar.
* Modelo Producto existente.
* Modelo MovimientoInventario para validación de historial.
* Modelo AuditLog disponible.

**Notas Técnicas**

* La eliminación es física (hard delete), no lógica (soft delete), por simplicidad del MVP.
* La validación de MovimientoInventario previene eliminar productos con historial de movimientos, manteniendo integridad de auditoría.
* El requisito de escribir "ELIMINAR" es un patrón UX común para acciones destructivas (GitHub, AWS lo usan).
* La restricción de solo administradores (no vendedores con permiso) es por seguridad: eliminar es más crítico que editar.
* Si en el futuro se necesita "recuperar" productos eliminados, implementar soft delete con campo is\_deleted.
* La auditoría debe registrar información del producto ANTES de eliminarlo (después ya no existirá en BD).
* Considerar agregar restricción adicional: no eliminar productos que aparezcan en historial de ventas (implementar validación en futuras iteraciones si es necesario).
* Para MVP, asumimos que productos que llegaron a stock 0 y sin movimientos pueden eliminarse sin problema.

US-013: Agregar Productos al Carrito y Procesar Venta

Como vendedor o administrador  
Quiero seleccionar productos en el POS, agregarlos a un carrito y procesar la venta  
Para registrar transacciones, descontar el stock automáticamente y facilitar el cobro al cliente

**Criterios de Aceptación**

**Acceso al POS**

1. Opción "Punto de Venta" visible en el menú principal para todos los usuarios con rol vendedor o administrador.
2. Al ingresar, muestra formulario de venta con búsqueda rápida de productos.

**Carrito de Compras**

1. Campo de búsqueda permite buscar productos por nombre, código SKU, o código de barras.
2. Vendedor puede escoger cantidad de cada producto a agregar al carrito (máximo stock disponible).
3. Al agregar producto, aparece en listado del carrito con:
   * Nombre
   * SKU
   * Cantidad
   * Precio unitario (CLP)
   * Subtotal
4. Carrito muestra total acumulado en CLP.

**Validación de Stock**

1. No se puede agregar cantidad mayor a stock disponible de ningún producto.
2. Si stock disponible = 0, producto no puede añadirse al carrito.
3. Mensaje de error si intenta agregar exceso: "No puedes agregar más unidades que el stock disponible".

**Métodos de Pago**

1. Campo obligatorio para seleccionar método de pago:
   * Efectivo
   * Transferencia
   * Redcompra
2. No permite finalizar venta sin método de pago seleccionado.

**Procesar Venta**

1. Botón "Procesar venta" habilitado solo si hay productos en el carrito y método de pago seleccionado.
2. Al hacer clic, muestra confirmación con resumen:
   * Productos, cantidades, precios, total
   * Método de pago seleccionado

**Actualización de Inventario**

1. Al confirmar venta, sistema descuenta automáticamente el stock vendido de cada producto.
2. Actualiza stock disponible y recalcula badges en inventario (stock bajo, agotado).
3. Si producto queda con stock <= umbral mínimo, agrega badge "Stock Bajo".

**Registro de Venta**

1. Venta se registra en base de datos con:
   * Productos vendidos
   * Cantidad por producto
   * Precio unitario y total en CLP
   * Vendedor responsable
   * Fecha y hora
   * Método de pago
   * Estado: "Venta confirmada"
2. Auditoría registra venta con todos los detalles para trazabilidad.

**Confirmación y Redirección**

1. Mensaje de éxito: "Venta registrada exitosamente".
2. Carrito se limpia y vuelve al estado inicial.
3. Redirige al dashboard o mantiene en POS para ventas consecutivas (configurable).

**Tareas Técnicas**

**Backend (Django)**

* Modelo Venta con campos:
  + productos\_vendidos (ManyToMany o JSON)
  + cantidad
  + precio\_unitario
  + total
  + vendedor (ForeignKey a Usuario)
  + fecha\_hora
  + metodo\_pago
  + estado (Venta confirmada)
* Endpoint POST /api/pos/venta/ para registrar venta.
* Validaciones:
  + Stock suficiente antes de vender.
  + Método de pago seleccionado.
* Transacción atómica para venta e inventario.
* Actualizar stock\_disponible de cada producto.
* Registrar venta en auditoría.

**Frontend (React)**

* Componente POSPage accesible por vendedores/administradores.
* Componente ProductSearchBar para buscar productos por nombre, código, o escáner de barras.
* Carrito: listar productos agregados, seleccionar cantidad, eliminar si es necesario.
* Selección de método de pago obligatorio (botones radio o dropdown).
* Botón "Procesar venta" con validación.
* Modal de confirmación antes de registrar venta.
* Llamada POST al endpoint para guardar venta.
* Mensaje de éxito y limpieza del carrito tras completar venta.
* Actualización de badges en inventario tras venta.
* Loading states y manejo de errores.

**Configuración**

* Formato CLP consistente en todo el POS.
* Permiso "Acceso al POS" asignado por defecto a vendedores y administradores.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuario accede al POS, busca productos y los agrega al carrito.
* Stock validado antes de agregar al carrito y antes de procesar venta.
* Método de pago es obligatorio.
* Al procesar venta, stock descontado automáticamente en inventario.
* Badge "Stock Bajo" aparece si corresponde después de la venta.
* Venta registrada con todos los detalles en base de datos y auditoría.
* Mensaje de éxito claro, carrito reset y listo para próxima transacción.

**Seguridad**

* Endpoint valida stock antes de descontar.
* Solo usuarios autenticados acceden al POS y procesan venta.
* Transacción es atómica, no se vende si falla cualquier parte.

**Experiencia de Usuario**

* Flujo rápido e intuitivo para ventas en mostrador.
* Errores y advertencias claras ante problemas de stock.
* Visualización del carrito organizada y continua.

**Testing (Prioritarios)**

* Test de venta: vendedor procesa venta con productos, stock descontado correctamente.
* Test de validación: intento de vender más que el stock disponible rechazado.
* Test de seguridad: solo usuario autenticado procesa venta.

**Documentación**

* Endpoint POST /api/pos/venta/ documentado con ejemplos de request/response.
* Campos requeridos y permisión de acceso documentados.

**Dependencias**

* Inventario funcional y actualizado con stock disponible.
* Modelo Usuario (vendedor/administrador).
* AuditLog activo para registrar ventas.

**Notas Técnicas**

* Stock debe descontarse solo de productos agregados al carrito y confirmados en venta.
* El estado inicial del carrito debe ser vacío tras cada venta.
* Badges de inventario deben recalcularse tras cada venta para alertar de stock bajo/agotar.
* Para MVP, descuentos y boletas NO se implementan (en futuras versiones pueden añadirse fácilmente).
* Si en el futuro se requiere impresión de boleta o integración con SII, diseñar endpoint adicional.

US-014: Confirmar Reserva como Venta

Como administrador o vendedor con permiso de gestionar reservas  
Quiero confirmar una reserva pendiente y convertirla en una venta completada  
Para registrar la transacción, liberar el stock reservado correctamente y que el cliente pueda retirar sus productos

Criterios de Aceptación

Acceso y Permisos

* La opción "Confirmar Venta" está visible en la vista de Reservas Pendientes para usuarios con permiso "Gestionar reservas" o rol Administrador.
* El botón aparece en cada fila de reserva pendiente dentro de la tabla.

Flujo de Confirmación

* Al hacer clic en "Confirmar Venta", se abre un modal que muestra:
  + Datos del cliente: RUT, nombre completo, teléfono
  + Lista de productos reservados (nombre, cantidad, precio unitario al momento de reserva)
  + Subtotal de productos
  + Campo selector obligatorio: Método de pago (Efectivo, Transferencia, Redcompra)
  + Campo opcional: Aplicar descuento manual (tipo: porcentaje o monto fijo, valor, motivo)
  + Total final calculado dinámicamente (subtotal - descuento)
* Botones disponibles: "Confirmar Venta" y "Cancelar"

Validaciones

* El sistema valida que la reserva esté en estado "Pendiente".
* Si la reserva está vencida (fecha\_vencimiento < hoy), mostrar alerta: "Esta reserva está vencida. ¿Deseas confirmarla de todas formas?" con opciones Sí/No.
* Si el usuario no tiene permiso para confirmar reservas, bloquear acción y mostrar mensaje de error.
* Campo "Método de pago" es obligatorio; botón "Confirmar Venta" deshabilitado hasta selección.

Procesamiento de Confirmación

Al confirmar, el sistema debe:

1. Crear registro en tabla Venta con:
   * usuario\_id (quien confirma la reserva)
   * Fecha y hora actuales
   * Subtotal, descuento (si aplica) y total calculado
   * Método de pago seleccionado
   * es\_preventa = TRUE
   * reserva\_id (FK a la reserva confirmada)
   * Datos del cliente (RUT, nombre, apellido)
2. Crear registros en VentaProducto por cada producto de la reserva:
   * producto\_id, cantidad, precio\_unitario (snapshot del momento de reserva)
3. Actualizar tabla Producto:
   * Reducir stock\_reservado por las cantidades reservadas
   * NO reducir stock\_disponible (ya fue reducido al crear la reserva)
4. Actualizar tabla Reserva:
   * Cambiar estado a "Confirmada"
   * Registrar fecha\_confirmacion (timestamp actual)
5. Crear registros en MovimientoInventario por cada producto:
   * tipo = 'Salida'
   * motivo = 'Reserva Confirmada'
   * cantidad, stock\_anterior, stock\_resultante
6. Registrar en AuditLog: "Confirmó reserva #[ID] como venta #[ID\_VENTA]"

Feedback al Usuario

* Mostrar notificación de éxito: "Reserva confirmada exitosamente. Venta #[ID] creada."
* Cerrar modal automáticamente tras confirmación.
* Refrescar lista de reservas (la confirmada desaparece de "Pendientes" y aparece en "Confirmadas").
* Actualizar historial de ventas para que la nueva venta sea visible inmediatamente.

Casos de Error

* Si hay error al procesar (conexión, validación), mostrar: "Error al confirmar reserva. Inténtalo nuevamente."
* No modificar estado de reserva ni stock si la operación falla (rollback transaccional).

Tareas Técnicas

Backend (Django REST Framework)

* Endpoint: POST /api/reservas/{id}/confirmar/
* Serializer: ReservationConfirmSerializer con campos:
  + metodo\_pago (obligatorio, ENUM)
  + descuento\_tipo (opcional, ENUM)
  + descuento\_valor (opcional, DECIMAL)
  + descuento\_motivo (opcional, TEXT)
* Validaciones:
  + Reserva existe y está en estado "Pendiente"
  + Usuario tiene permiso para confirmar reservas
  + Método de pago es válido
* Lógica transaccional (transaction.atomic()):
  + Crear Venta y múltiples VentaProducto
  + Actualizar stock\_reservado en Producto
  + Cambiar estado de Reserva a "Confirmada"
  + Crear registros en MovimientoInventario
  + Registrar en AuditLog
* Respuesta exitosa (201):

json

{

"mensaje": "Reserva confirmada exitosamente",

"venta\_id": 123,

"reserva\_id": 45,

"total": 25000.00

}

Frontend (React)

* Componente: ConfirmReservationModal.jsx
* Props: reserva (objeto completo), onConfirm (callback), onCancel
* Estado local:
  + metodoPago (string)
  + descuentoTipo (string | null)
  + descuentoValor (number | null)
  + descuentoMotivo (string | null)
  + loading (boolean)
  + totalFinal (number, calculado)
* Validaciones:
  + Método de pago obligatorio
  + Si aplica descuento, validar valor > 0 y ≤ subtotal
* Al confirmar:
  + Llamar POST /api/reservas/{id}/confirmar/ con datos del formulario
  + Manejar respuesta y mostrar notificación
  + Refrescar listas de reservas y ventas
  + Cerrar modal

Configuración

* Permiso: "Gestionar reservas"
* Transacciones atómicas obligatorias para garantizar consistencia de datos
* Log de auditoría habilitado

Definición de Hecho (DoD)

Funcionalidad

* Usuarios autorizados pueden confirmar reservas pendientes
* Modal muestra información correcta y calcula total dinámicamente
* Venta se crea correctamente con flag es\_preventa = TRUE
* Stock reservado se libera y se registra movimiento de inventario
* Reserva cambia de estado a "Confirmada"

Seguridad

* Validación estricta de permisos en backend
* Transacciones atómicas previenen inconsistencias
* Auditoría registra quién, cuándo y qué reserva fue confirmada

Experiencia de Usuario

* Modal claro e intuitivo con información completa
* Feedback inmediato tras confirmación
* Actualización automática de vistas de reservas y ventas
* Mensajes de error descriptivos y claros

Testing (Prioritarios)

* Confirmar reserva pendiente genera venta correctamente
* Stock reservado se actualiza y no afecta stock disponible
* Descuentos se aplican correctamente al total
* Reservas vencidas muestran alerta pero permiten confirmación
* Usuarios sin permiso no pueden confirmar reservas
* Rollback funciona correctamente en caso de error

Documentación

* Documentar endpoint con parámetros, validaciones y respuestas
* Documentar flujo transaccional y lógica de stock
* Documentar permisos requeridos

Dependencias

* US-013: Crear reserva básica de producto (completada)
* Modelo Venta, VentaProducto, Reserva, ReservaProducto activos
* Modelo MovimientoInventario y AuditLog implementados
* Endpoint POST /api/ventas/ funcional como referencia

Notas Técnicas

* La venta generada desde reserva tiene flag es\_preventa = TRUE para distinguirla de ventas directas.
* El precio\_unitario en VentaProducto es snapshot del momento de reserva, no el precio actual del producto.
* Stock reservado no se suma de vuelta a stock disponible; simplemente se reduce de stock\_reservado.
* Si la reserva está vencida, el sistema permite confirmarla tras advertencia (decisión de negocio con Don Claudio).

US-015: Confirmar Reserva como Venta

Como vendedor autorizado o administrador  
Quiero poder convertir una reserva pendiente en una venta registrada dentro del POS  
Para reflejar la transacción exacta en el sistema, liberar stock reservado y mantener la trazabilidad comercial

**Criterios de Aceptación**

**Acceso y Permisos**

1. Opción “Confirmar como Venta” visible en el módulo de reservas para usuarios con permiso “Confirmar reservas”, administradores incluidos.
2. Botón disponible exclusivamente en reservas con estado "Pendiente".

**Flujo de Confirmación**

1. Al hacer clic, se abre modal con resumen de la reserva:
   * Código de reserva (ej: RSV-20251015-AB12)
   * Cliente: RUT (formato chileno, validado), nombre, apellido
   * Fecha de creación de la reserva
   * Productos reservados:
     + Nombre de cada producto
     + Cantidad reservada por producto
     + Precio unitario (CLP)
     + Subtotal por producto
   * Total estimado en CLP de la reserva
2. Selección de método de pago (radio buttons obligatorios):
   * Efectivo
   * Transferencia
   * Redcompra
3. Campo adicional “Observaciones de la venta” (opcional).

**Validaciones**

1. Botón “Confirmar venta” activo solo si método de pago está seleccionado.
2. Antes de confirmar, backend y frontend validan:
   * Reserva sigue estado "Pendiente"
   * Stock reservado de cada producto debe ser suficiente
   * Método de pago seleccionado
3. Si el stock reservado ha cambiado (por ajuste manual, cancelación, etc.), muestra error:
   * “Stock reservado insuficiente para confirmar esta reserva. Modifique la reserva o ajuste el inventario.”

**Proceso de Confirmación**

1. Al confirmar:
   * Se registra la venta y se actualiza estado de la reserva a “Confirmada/Venta realizada”
   * Stock reservado disminuye por cantidad vendida de cada producto
   * Registro o decremento de stock\_disponible NO ocurre (ya se había restado en la creación de la reserva)
   * Fecha y usuario que confirma guardados en reserva y venta
   * Datos de transacción, método de pago y observaciones se vinculan en registro de venta

**Actualización de Inventario**

1. Productos involucrados:
   * stock\_reservado se reduce por la cantidad vendida
   * stock total del inventario se refleja correctamente
2. Si producto alcanza stock total = 0 tras venta, badge "Agotado" se activa automáticamente.

**Confirmación y Feedback**

1. Mensaje de éxito: “Reserva confirmada como venta. Stock actualizado correctamente. Código de venta: [ID]”
2. Modal se cierra y la reserva pasa a estado “Confirmada”.
3. Venta aparece inmediatamente en historial de ventas con referencia al código de reserva.

**Auditoría y Trazabilidad**

1. Registro en AuditLog de la transacción, incluyendo:
   * Timestamp
   * Usuario (vendedor/administrador que confirma)
   * Código de reserva y productos vendidos
   * Método de pago
   * Total final
   * Observaciones (si aplicaron)
   * Estado de la reserva antes y después

**Tareas Técnicas**

**Backend (Django)**

* Endpoint PATCH /api/reservas/{id}/confirmar/ accesible por usuarios autorizados.
* Validaciones:
  + Estado de la reserva ("Pendiente")
  + Stock reservado por producto suficiente (sin inconsistencias)
  + Método de pago recibido
* Proceso atómico:
  + Actualiza estado reserva a "Confirmada"
  + Disminuye stock\_reservado en producto(s) (mantiene stock\_disponible)
  + Crea registro en modelo Venta (referenciando reserva)
  + Guarda fecha, usuario y método de pago
  + Almacena observaciones opcionales
  + Registro en AuditLog con todos los detalles de la transacción
* Actualización visual inmediata en inventario y módulo de ventas.
* Errores claros (stock insuficiente, reserva cancelada o vencida).

**Frontend (React)**

* Botón “Confirmar como Venta” visible en reservas pendientes.
* Modal ConfirmReservationAsSaleModal muestra todos los datos relevantes para validación del vendedor.
* Selección obligatoria de método de pago en el modal (radio buttons).
* Campo de observaciones opcional.
* Llamada PATCH al endpoint con todos los datos: productos, cantidades, método de pago, observaciones.
* Loading states durante proceso.
* Mensaje de error si validación falla (stock, estado, etc.)
* Mensaje de éxito y cierre de modal tras confirmación.
* Actualización inmediata en la lista de reservas y ventas.
* Badge “Venta confirmada" en reserva y referencia cruzada en historial de ventas.

**Configuración**

* Permiso "Confirmar reservas" debe estar configurado en cada usuario que pueda realizar la acción.
* Métodos de pago definidos como enumeración en backend/frontend.
* Auditoría activa y tabulada en la base de datos/logs del sistema.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo usuarios autorizados pueden confirmar reservas como venta.
* Modal con datos completos: cliente, productos, cantidades, precios, total, método de pago.
* Validaciones estrictas de stock, método de pago y estado.
* Proceso de confirmación es transaccional y seguro.
* Inventario actualizado solo en stock\_reservado (jamás excede disponible).
* Reserva pasa a estado “Confirmada” y referenciada en ventas.
* Mensaje claro y feedback visual inmediato.

**Seguridad**

* Endpoint verifica rol y permisos antes de ejecutar acción.
* Validaciones de estado y stock tanto en backend como frontend.
* Registro completo en AuditLog de cada confirmación.

**Experiencia de Usuario**

* Flujo ágil y claro para vendedores.
* Modal con información completa y formato visual profesional.
* Errores y advertencias claros y orientados.
* Cierre de modal y actualización visual sin recarga manual.

**Testing (Prioritarios)**

* Test de integración: reserva confirmada → venta registrada → stock reservado actualizado.
* Test de validación: confirmación rechazada si stock reservado insuficiente.
* Test de seguridad: usuario sin permiso no puede confirmar reservas.

**Documentación**

* Endpoint PATCH /api/reservas/{id}/confirmar/ documentado con ejemplos de request y response.
* Estructura de permisos y modelo de reserva/venta documentados.
* Validaciones de estado y stock especificadas.

**Dependencias**

* Funcionamiento de inventario y esquema de reservas como en US-017.
* Tabla de ventas funcional y vinculable a reservas.
* Permiso “Confirmar reservas” asignado correctamente a usuarios.

**Notas Técnicas**

* El proceso de confirmación debe ser 100% atómico: si se produce error en cualquier paso, no se registra nada ni se modifica inventario.
* El código de reserva referenciado en modelo Venta para trazabilidad cruzada en auditoría.
* Si se habilita futura integración SII o boletas, la confirmación será el desencadenante del proceso de facturación.
* Las validaciones deben cubrir variaciones por ajuste manual de reservas, para evitar sobreventa de stock.
* Puede prepararse endpoint de historial de reservas con columna "Venta relacionada" para facilitar auditorías.

**Cambios realizados respecto a la versión anterior**

* Refuerzo de atomicidad: Explicitado que la confirmación y el registro de venta deben ser parte de la misma transacción atómica.
* Aclaración de experiencia de usuario: Se recomienda que la actualización visual sea inmediata y sin recarga manual.
* Reforzada la validación de stock: Se enfatiza que la validación debe ocurrir tanto en frontend como en backend antes de confirmar la venta.

**US-016: Consultar Historial Básico de Ventas**

Como administrador o vendedor con permiso  
Quiero consultar el historial de ventas realizadas en el módulo POS  
Para revisar transacciones, ver detalles de productos vendidos y filtrar ventas recientes según usuario o fecha

**Criterios de Aceptación**

**Acceso al Historial**

1. Opción "Historial de Ventas" visible desde el menú principal para administradores y vendedores con permiso.
2. Redirige a /ventas/historial

**Visualización Básica**

1. Tabla de ventas con columnas:
   * ID de venta
   * Fecha y hora
   * Vendedor responsable
   * Total venta (CLP)
   * Cantidad de productos vendidos
   * Método de pago
2. Por defecto, ordenado por fecha descendente (más reciente primero).

**Filtros Mínimos**

1. Barra de búsqueda permite:
   * Búsqueda por nombre del vendedor
   * Búsqueda por fecha (fecha específica o rango)
2. Actualiza resultados en tiempo real.

**Ver Detalle**

1. Botón "Ver Detalle" en cada fila abre modal con:
   * Productos vendidos (nombre, cantidad, precio unitario)
   * Subtotal y total en CLP
   * Método de pago
   * Vendedor responsable
   * Fecha y hora

**Auditoría y Seguridad**

1. Solo usuarios autenticados y con permiso acceden al historial.
2. No se muestran datos sensibles de clientes (solo transacciones internas).

**Exportar Historial (Opcional para MVP)**

1. Botón "Exportar a CSV" disponible para administradores.
2. Exporta listado actual (con filtros aplicados).

**Tareas Técnicas**

**Backend (Django)**

* Modelo Venta con todos los campos de la transacción, enlazando productos y vendedor.
* Endpoint GET /api/ventas/historial/ con paginación básica.
* Soporta filtros: vendedor, fecha inicial, fecha final.
* Endpoint GET /api/ventas/{id}/ para detalle completo.
* Endpoint GET /api/ventas/historial/export/ (opcional, CSV).
* Validación de permisos.

**Frontend (React)**

* Componente SalesHistoryPage con tabla y filtros.
* Modal SalesDetailModal con información completa de la transacción.
* Llamada GET a /api/ventas/historial/ con parámetros de filtro.
* Botón de exportar a CSV si usuario es administrador.

**Configuración**

* Paginación estándar: 25 ventas por página.
* Formato CLP consistente en toda la vista.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Administrador o vendedor accede y visualiza el historial con datos correctos.
* Filtros permiten búsqueda básica por vendedor y fecha.
* Modal muestra detalle completo de venta.
* Exportar CSV funciona para administradores.
* Paginación correcta y feedback visual de carga.

**Seguridad**

* Solo usuarios autenticados acceden.
* No se filtran datos de clientes al listado.

**Experiencia de Usuario**

* Tabla informativa y fácil de leer.
* Filtros rápidos e intuitivos.
* Modal de detalles claro y completo.

**Testing (Prioritarios)**

* Test de validación de acceso por permisos.
* Test de filtro por vendedor y fecha.
* Test de visualización y exportación.

**Documentación**

* Endpoint GET /api/ventas/historial/ y /api/ventas/{id}/ documentados.
* Parámetros y formato de exportación especificados.

**Dependencias**

* US-015 (Registrar venta) completada y funcional.
* Modelo Venta en base de datos.

**Notas Técnicas**

* Para MVP, detalle de productos vendido se almacena como array/JSON dentro de Venta o tabla relacional.
* El exportador de CSV puede implementarse con librería nativa de Django.
* Si se requiere ver el total vendido por vendedor, sumar con query agrupada en el futuro.
* La paginación mejora performance para negocios con muchas ventas al día.

US-017: Crear Reserva Básica de Producto

Como vendedor autorizado o administrador  
Quiero apartar productos específicos para un cliente mediante el sistema de reservas  
Para asegurar que no sean vendidos a otro cliente y registrar el compromiso de entrega

**Criterios de Aceptación**

**Acceso a Reservas**

1. Opción "Reservar producto" disponible en el POS y en inventario para usuarios con permiso "Crear reservas"
2. Puede acceder desde:
   * Venta individual (POS)
   * Listado de inventario por producto

**Formulario de Reserva**

1. Formulario muestra:
   * Producto(s) a reservar (selección directa, cantidad máxima = stock disponible)
   * RUT del cliente (formato chileno, validado)
   * Nombre y apellido del cliente (requeridos)
   * Teléfono de contacto (opcional)
   * Comentarios/observaciones (opcional)
2. Muestra stock disponible antes y cantidad a reservar.
3. Permite reservar solo si stock disponible es suficiente.

**Generación de Reserva**

1. Al confirmar:
   * Se crea reserva con código único de seguimiento (ej: RSV-20251015-AX31)
   * Stock reservado aumenta (stock\_disponible ↓, stock\_reservado ↑ solo por la cantidad apartada)
   * No se descuenta aún de inventario total; solo pasa de disponible a reservado
   * Reserva se marca como "Pendiente"

**Restricciones**

1. No permite reservar más cantidad que el stock disponible.
2. Si producto tiene stock insuficiente, muestra error y bloquea reserva.
3. Permiso "Crear reservas" configurable, solo usuarios habilitados pueden realizar la acción.

**Confirmación y Auditoría**

1. Muestra mensaje: "Reserva creada correctamente, código de reserva: [código único]"
2. Registro en base de datos con todos los datos del cliente, producto(s), cantidad, vendedor responsable, fecha y hora.
3. Auditoría registra la reserva con todos los detalles para trazabilidad.

**Tareas Técnicas**

**Backend (Django)**

* Modelo Reserva con campos:
  + productos\_reservados (ManyToMany o JSON)
  + cantidad (por producto)
  + rut\_cliente (string, validación chilena)
  + nombre\_cliente
  + apellido\_cliente
  + telefono
  + comentarios
  + vendedor (ForeignKey a Usuario)
  + estado (Pendiente)
  + fecha\_hora
  + codigo\_reserva (único, autogenerado)
* Endpoint POST /api/reservas/ para crear reserva
* Validación de stock suficiente antes de reservar
* Permiso para crear reservas
* Auditoría de creación de reservas

**Frontend (React)**

* Componente ReserveProductForm accesible por usuarios autorizados
* Formulario con validación de RUT chileno y campos requeridos
* Selección de productos y cantidad (máximo stock disponible)
* Mensajes de error claros al intentar reservar más de lo disponible
* Mensaje de éxito con código de reserva visible después de crear
* Actualizar inventario (stock reservado/disponible) tras creación

**Configuración**

* Permiso "Crear reservas" en roles de usuario

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuario autorizado accede a página de reservas y puede crear una nueva reserva básica
* Solo productos con stock suficiente pueden apartarse
* Formulario valida RUT, nombre, apellido obligatorios
* Teléfono y comentarios son opcionales
* Creación aumenta stock reservado y disminuye disponible
* Reserva visible en módulo de reservas y en inventario
* Mensaje claro con código único al confirmar

**Seguridad**

* Endpoint verifica permiso antes de permitir crear reserva
* Solo usuarios autorizados acceden
* Validación de stock y campos antes de crear

**Experiencia de Usuario**

* Formulario rápido y claro
* Validación de campos en tiempo real
* Mensajes de éxito y error claros
* Feedback visual de stock tras reservar

**Testing (Prioritarios)**

* Test de validación de stock: no permite reservar más que lo disponible
* Test de creación: reserva aparece con código único, datos correctos
* Test de seguridad: usuario sin permiso no puede reservar

**Documentación**

* Endpoint POST /api/reservas/ documentado
* Campos requeridos y de validación documentados

**Dependencias**

* Inventario funcional y actualizado con stock disponible/reservado
* Modelo Usuario con permisos asignados

**Notas Técnicas**

* El código de reserva se autogenera usando fecha + random para trazabilidad
* Validación de RUT chileno puede implementarse con librería dedicada
* La auditoría debe incluir detalles del vendedor, producto, cantidad, y cliente para trazabilidad legal/comercial
* En MVP, la reserva no descuenta producto definitivamente hasta confirmación de la venta; solo bloquea stock como reservado

US-018: Historial Básico de Ventas

Como administrador o vendedor autorizado  
Quiero consultar el historial de ventas registradas en el sistema  
Para revisar transacciones pasadas, validar datos y realizar seguimiento básico

**Criterios de Aceptación**

**Acceso y Permisos**

1. Opción “Historial de Ventas” visible en el menú principal solo para usuarios con permisos "Ver historial de ventas" o rol administrador.
2. Al ingresar, se muestra pantalla /ventas/historial con listado paginado de ventas.

**Visualización del Historial**

1. Tabla con columnas:
   * ID de venta
   * Fecha y hora de venta
   * Vendedor responsable
   * Total venta en CLP
   * Cantidad total de productos vendidos
   * Método de pago utilizado
2. Las ventas están ordenadas por fecha descendente (más recientes arriba).

**Filtros**

1. Filtros básicos que se pueden aplicar simultáneamente:
   * Rango de fechas (fecha inicio y fin)
   * Nombre o ID de vendedor
   * Método de pago
2. Barra de búsqueda por ID de venta o nombre de vendedor (texto libre).
3. Actualización en tiempo real de resultados al aplicar filtros y búsqueda.

**Detalle de Venta**

1. Cada fila incluye botón “Ver detalle” que abre modal con:
   * Lista de productos vendidos (nombre, cantidad, precio unitario)
   * Subtotal por producto
   * Total final
   * Datos del cliente (RUT, nombre) si disponible (opcional)
   * Estado de la venta (normal, preventa/confirmada de reserva)
   * Método de pago
   * Fecha y hora
   * Vendedor responsable

**Exportación (Opcional MVP)**

1. Botón “Exportar CSV” disponible para administradores.
2. Exporta datos visibles en tabla con filtros aplicados.

**Tareas Técnicas**

**Backend (Django)**

* Modelo Venta con campos necesarios ya definido.
* Endpoint GET /api/ventas/historial/ con soporte para filtros y paginación.
* Endpoint GET /api/ventas/{id}/ para detalle específico.
* Validación de permisos.
* Implementación de exportación CSV para administradores.
* Asegurar que datos sensibles de clientes solo sean visibles para roles autorizados.

**Frontend (React)**

* Componente SalesHistoryPage con tabla de ventas y filtros.
* Modal SaleDetailsModal para mostrar información completa.
* Implementar llamadas GET con parámetros filtrados.
* Botón de exportación para CSV, disponible para administradores.
* Manejo de estados de carga y error.
* Formato CLP consistente en toda la interfaz.

**Configuración**

* Permiso “Ver historial de ventas”.
* Paginación estándar: 25 por página.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuarios autorizados pueden acceder al historial.
* Filtros y búsqueda operan correctamente en combinación.
* Detalles completos accesibles y visualizados en modal.
* Exportación CSV funcional para administradores.

**Seguridad**

* Validación estricta de permisos para acceso a datos.
* Datos sensibles de clientes protegidos según rol.
* Exportación solo disponible para administradores.

**Experiencia de Usuario**

* Interfaz limpia y rápida.
* Filtros intuitivos.
* Mensajes claros de éxito y error.
* Modal de detalle fácil de navegar.

**Testing (Prioritarios)**

* Acceso correcto según permisos.
* Filtros y búsqueda devuelven resultados precisos.
* Detalle de venta muestra información correcta.
* Exportación CSV produce archivo válido.

**Documentación**

* Documentar endpoints, parámetros y formatos de respuesta.
* Documentar permisos y uso de exportación.

**Dependencias**

* US-015: Registro de venta funcional.
* Modelo Venta y Usuario activos.
* Auditoría implementada.

**Notas Técnicas**

* El historial es para consultas rápidas, sin análisis avanzado (implementar en EPIC-05).
* El detalle incluye venta asociada a reserva si aplica.
* CSV debe incluir sólo campos visibles y autorizados para usuario actual.
* Considerar paginación y filtros eficientes para grandes volúmenes de ventas.
* El modal de detalle puede incluir link a reserva asociada si corresponde.

US-019: Actualizar Stock Automáticamente tras Venta

Como sistema  
Quiero que después de confirmar una venta, el inventario se actualice automáticamente  
Para reflejar la reducción del stock disponible y garantizar un inventario preciso en tiempo real

**Criterios de Aceptación**

**Proceso Automático**

1. Para cada producto vendido, el sistema reduce automáticamente la cantidad vendida del stock disponible.
2. En caso de venta confirmada a través de reserva, reduce el stock reservado correspondiente.
3. El stock reservado se ajusta solo en ventas derivadas de reservas; para ventas normales solo se afecta stock disponible.
4. El sistema nunca permite que el stock disponible quede en valores negativos.
5. Actualiza la información del inventario visible en todos los módulos (POS, inventario, reservas) inmediatamente tras la venta.

**Validaciones**

1. Antes de descontar, el sistema valida que el stock disponible sea suficiente para la cantidad vendida.
2. Si la validación falla (stock insuficiente), bloquea la venta y muestra mensaje de error claro:
   * “No hay suficiente stock disponible para completar la venta.”
3. La actualización de stock debe ser parte de una transacción atómica para evitar inconsistencias.

**Feedback y Auditoría**

1. Cada decremento de stock genera un registro en la tabla MovimientoInventario con tipo 'salida' asociado a la venta, incluyendo:
   * Producto afectado
   * Cantidad descontada
   * Stock anterior y nuevo
   * Usuario responsable
   * Fecha y hora
   * Referencia a la venta
2. Decrementos de stock quedan reflejados inmediatamente para todos los usuarios conectados (actualización en tiempo real en frontend).
3. Mensajes claros en caso de errores o problemas durante la actualización.

**Tareas Técnicas**

**Backend (Django)**

* Implementar lógica en endpoint de registro de venta (US-015) para modificar stock disponible o reservado según corresponda.
* Validar cantidades antes de modificar stock.
* Actualizar tabla MovimientoInventario con detalles del ajuste (tipo 'salida', referencia a venta).
* Utilizar transacciones atómicas para garantizar la consistencia.
* Notificar errores y manejar excepciones de forma clara y segura.

**Frontend (React)**

* Feedback instantáneo tras venta exitosa mostrando stock actualizado.
* Error visual en caso de stock insuficiente en momento de venta.
* Actualización en tiempo real de badges y cantidades en listado de inventario y POS.

**Configuración**

* Configurar parámetros de transacciones atómicas si aplica.
* Definir tipos de movimientos en inventario para stock descontado por venta.

**Definición de Hecho (DoD)**

**Funcionalidad**

* El stock disponible y reservado se actualizan correctamente tras cada venta confirmada.
* No se permiten ventas si el stock no es suficiente.
* Movimientos de inventario quedan registrados con detalles completos.
* Usuarios ven la información actualizada inmediatamente en todos los módulos.

**Seguridad**

* Validaciones de stock y permiso en backend.
* Transacción atómica para evitar inconsistencias.

**Experiencia de Usuario**

* Mensajes claros y oportunos ante éxito o error.
* Refresco de datos sin necesidad de recargar página.

**Testing (Prioritarios)**

* Venta con stock suficiente descuenta correctamente.
* Venta con stock insuficiente bloquea la operación.
* Registro correcto en movimiento de inventario tras venta.

**Documentación**

* Actualización en endpoint de venta documentada con lógica de stock y movimientos.
* Detalles de movimientos en inventario especificados.

**Dependencias**

* US-015: Registro de venta.
* US-012: Movimientos de inventario.
* Modelo Producto con stock\_disponible y stock\_reservado.

**Notas Técnicas**

* La actualización de stock es crítica y debe ser segura, rápida y confiable.
* En ventas con reserva, la reducción afecta stock reservado primero.
* Para ventas normales, solo stock disponible se reduce.
* Esto evita sobreventa y mantiene inventario realista.
* El registro en MovimientoInventario es fundamental para trazabilidad y auditoría.

US-020: Gestión y Aplicación de Descuentos Manuales en Venta

Como vendedor autorizado o administrador  
Quiero aplicar descuentos manuales a productos durante la venta en POS  
Para ofrecer promociones excepcionales a clientes y registrar correctamente los descuentos aplicados

**Criterios de Aceptación**

**Acceso y Permisos**

1. Solo usuarios con permiso “Aplicar descuentos” pueden acceder a esta funcionalidad.
2. Usuarios sin permiso no ven opción de descuentos en la interfaz POS.

**Funcionalidad de Descuento**

1. Descuento puede aplicarse por producto o a la venta total.
2. Tipos de descuento permitidos:
   * Porcentaje (%) sobre precio o total
   * Monto fijo (en CLP)
3. Al seleccionar aplicar descuento:
   * Se muestra formulario para ingresar tipo, valor y motivo opcional.
   * El sistema valida que el descuento no sea mayor al subtotal o al total según aplique.
4. Descuentos deben reflejarse inmediatamente en el cálculo final del precio.

**Validaciones**

1. El descuento no puede reducir el precio total del producto o venta a menos de $0.
2. Si el descuento es incorrecto o excede límites, muestra error claro y no permite aplicar.
3. El motivo es opcional pero se sugiere agregarlo para mejor control y trazabilidad.

**Auditoría y Notificaciones**

1. Cada descuento aplicado genera notificación automática vía Gmail a todos los administradores con Gmail asociado (US-008).
2. Registro detallado en auditoría que incluye:
   * Usuario que aplicó el descuento
   * Tipo y valor del descuento
   * Motivo (si aplicado)
   * Productos afectados
   * ID de venta

**Confirmación y Visualización**

1. En POS, productos con descuento muestran visualmente:
   * Precio original tachado
   * Precio con descuento destacado
   * Total descontado visible
2. En resumen final de la venta se muestra el descuento total aplicado.

**Tareas Técnicas**

**Backend (Django)**

* Crear endpoint para aplicar descuentos manuales en la venta POS.
* Validar tipo de descuento, que no exceda monto total.
* Implementar mecanismo para notificar por correo a administradores.
* Registrar descuentos en tabla de auditoría vinculados a ventas.
* Validar permisos para aplicar descuentos.

**Frontend (React)**

* Implementar formulario emergente para ingresar descuento.
* Validar datos en tiempo real.
* Mostrar efecto visual inmediato del descuento en precios.
* Enviar datos al backend para registro y disparo de notificaciones.
* Mostrar mensajes de error o éxito.

**Configuración**

* Permiso "Aplicar descuentos" configurable por usuario.
* Integración con sistema de notificaciones por Gmail.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuarios con permiso pueden aplicar descuentos manuales durante venta.
* Descuentos se validan y no exceden valores permitidos.
* Notificación por correo enviada correctamente.
* Auditoría registra cada descuento aplicado.
* Visualización clara y actualizada en POS.

**Seguridad**

* Validación de permisos en backend.
* Control para evitar descuentos negativos o mayores al valor.

**Experiencia de Usuario**

* Formulario claro y sencillo.
* Mensajes en tiempo real durante ingreso del descuento.
* Visualización clara de descuentos aplicados.

**Testing (Prioritarios)**

* Aplicación correcta de descuento permitido.
* Bloqueo cuando descuento excede subtotal o total.
* Envío de notificación por correo.
* Registro correcto en auditoría.

**Documentación**

* Endpoint documentado con ejemplos de solicitud y respuesta.
* Especificaciones de permisos y formatos.

**Dependencias**

* US-001: Autenticación con permisos cargados.
* US-008: Gestión de Gmail para notificaciones.
* Sistema de auditoría funcional.

**Notas Técnicas**

* La notificación por correo debe incluir detalles relevantes sin sobrecargar datos.
* El cálculo de descuentos debe usar precisión y evitar redondeos erróneos.
* Se puede expandir para descuentos promocionales automáticos en futuras fases.
* El endpoint debe ser transaccional: si falla la notificación o la auditoría, el descuento no debe aplicarse.

US-021: Historial Básico de Ventas

Como administrador o vendedor autorizado  
Quiero consultar el historial de ventas registradas en el sistema  
Para revisar transacciones pasadas, validar datos y realizar seguimiento básico

**Criterios de Aceptación**

**Acceso y Permisos**

1. Opción “Historial de Ventas” visible en el menú principal solo para usuarios con permisos "Ver historial de ventas" o rol administrador.
2. Al ingresar, se muestra pantalla /ventas/historial con listado paginado de ventas.

**Visualización del Historial**

1. Tabla con columnas:
   * ID de venta
   * Fecha y hora de venta
   * Vendedor responsable
   * Total venta en CLP
   * Cantidad total de productos vendidos
   * Método de pago utilizado
2. Las ventas están ordenadas por fecha descendente (más recientes arriba).

**Filtros**

1. Filtros básicos que se pueden aplicar simultáneamente:
   * Rango de fechas (fecha inicio y fin)
   * Nombre o ID de vendedor
   * Método de pago
2. Barra de búsqueda por ID de venta o nombre de vendedor (texto libre)
3. Actualización en tiempo real de resultados al aplicar filtros.

**Detalle de Venta**

1. Cada fila incluye botón “Ver detalle” que abre modal con:
   * Lista de productos vendidos (nombre, cantidad, precio unitario)
   * Subtotal por producto
   * Total final
   * Datos del cliente (RUT, nombre) si disponible (opcional)
   * Estado de la venta (normal, preventa/confirmada de reserva)
   * Método de pago
   * Fecha y hora
   * Vendedor responsable

**Exportación (Opcional MVP)**

1. Botón “Exportar CSV” disponible para administradores
2. Exporta datos visibles en tabla con filtros aplicados

**Tareas Técnicas**

**Backend (Django)**

* Modelo Venta con campos necesarios ya definido.
* Endpoint GET /api/ventas/historial/ con soporte para filtros y paginación.
* Endpoint GET /api/ventas/{id}/ para detalle específico.
* Validación de permisos.
* Implementación de exportación CSV para administradores.

**Frontend (React)**

* Componente SalesHistoryPage con tabla de ventas y filtros.
* Modal SaleDetailsModal para mostrar información completa.
* Implementar llamadas GET con parámetros filtrados.
* Botón de exportación para CSV, disponible para administradores.
* Manejo de estados de carga y error.

**Configuración**

* Permiso “Ver historial de ventas”.
* Paginación estándar: 25 por página.
* Formato CLP consistente.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuarios autorizados pueden acceder al historial
* Filtros y búsqueda operan correctamente en combinación
* Detalles completos accesibles y visualizados en modal
* Exportación CSV funcional para administradores

**Seguridad**

* Validación estricta de permisos para acceso a datos
* Datos sensibles de clientes protegidos según rol

**Experiencia de Usuario**

* Interfaz limpia y rápida
* Filtros intuitivos
* Mensajes claros de éxito y error

**Testing (Prioritarios)**

* Acceso correcto según permisos
* Filtros y búsqueda devuelven resultados precisos
* Detalle de venta muestra información correcta
* Exportación CSV produce archivo válido

**Documentación**

* Documentar endpoints, parámetros y formatos de respuesta
* Documentar permisos y uso de exportación

**Dependencias**

* US-015: Registro de venta funcional
* Modelo Venta y Usuario activos
* Auditoría implementada

**Notas Técnicas**

* El historial es para consultas rápidas, sin análisis avanzado (implementar en EPIC-05)
* El detalle incluye venta asociada a reserva si aplica
* CSV debe incluir sólo campos visibles y autorizados para usuario actual

US-022: Actualizar Stock Automáticamente tras Venta

Como sistema  
Quiero que después de confirmar una venta, el inventario se actualice automáticamente  
Para reflejar la reducción del stock disponible y garantizar un inventario preciso en tiempo real

**Criterios de Aceptación**

**Proceso Automático**

1. Para cada producto vendido, el sistema reduce automáticamente la cantidad vendida del stock disponible.
2. En caso de venta confirmada a través de reserva, reduce el stock reservado correspondiente.
3. El stock reservado se ajusta solo en ventas derivadas de reservas; para ventas normales solo se afecta stock disponible.
4. Evita dejar el stock disponible en valores negativos.
5. Actualiza la información del inventario visible en todos los módulos (POS, inventario, reservas).

**Validaciones**

1. Antes de descontar, el sistema valida que el stock disponible sea suficiente para la cantidad vendida.
2. Si la validación falla (stock insuficiente), bloquea la venta y muestra mensaje de error claro.
3. La actualización de stock debe ser parte de una transacción atómica para evitar inconsistencias.

**Feedback y Auditoría**

1. Registro de movimientos en la tabla MovimientoInventario con tipo 'salida' asociado a la venta.
2. Decrementos de stock quedan reflejados inmediatamente para todos los usuarios conectados.
3. Mensajes claros en caso de errores o problemas durante la actualización.

**Tareas Técnicas**

**Backend (Django)**

* Implementar lógica en endpoint de registro de venta (US-015) para modificar stock disponible o reservado según corresponda.
* Validar cantidades antes de modificar stock.
* Actualizar tabla MovimientoInventario con detalles del ajuste.
* Utilizar transacciones atómicas para garantizar la consistencia.
* Notificar errores y manejar excepciones.

**Frontend (React)**

* Feedback instantáneo tras venta exitosa mostrando stock actualizado.
* Error visual en caso de stock insuficiente en momento de venta.
* Actualización en tiempo real de badges y cantidades en listado de inventario y POS.

**Configuración**

* Configurar parámetros de transacciones atómicas si aplica.
* Definir tipos de movimientos en inventario para stock descontado por venta.

**Definición de Hecho (DoD)**

**Funcionalidad**

* El stock disponible y reservado se actualizan correctamente tras cada venta confirmada.
* No se permiten ventas si el stock no es suficiente.
* Movimientos de inventario quedan registrados con detalles completos.
* Usuarios ven la información actualizada inmediatamente.

**Seguridad**

* Validaciones de stock y permiso en backend.
* Transacción atómica para evitar inconsistencias.

**Experiencia de Usuario**

* Mensajes claros y oportunos.
* Refresco de datos sin necesidad de recargar página.

**Testing (Prioritarios)**

* Venta con stock suficiente descuenta correctamente.
* Venta con stock insuficiente bloquea la operación.
* Registro correcto en movimiento de inventario tras venta.

**Documentación**

* Actualización en endpoint de venta documentada con lógica de stock y movimientos.
* Detalles de movimientos en inventario especificados.

**Dependencias**

* US-015: Registro de venta
* US-012: Movimientos de inventario
* Modelo Producto con stock\_disponible y stock\_reservado

**Notas Técnicas**

* La actualización de stock es crítica y debe ser segura, rápida y confiable.
* En ventas con reserva, la reducción afecta stock reservado primero.
* Para ventas normales, solo stock disponible se reduce.
* Esto evita sobreventa y mantiene inventario realista.

US-023: Gestión y Aplicación de Descuentos Manuales en Venta

Como vendedor autorizado o administrador  
Quiero aplicar descuentos manuales a productos durante la venta en POS  
Para ofrecer promociones excepcionales a clientes y registrar correctamente los descuentos aplicados

**Criterios de Aceptación**

**Acceso y Permisos**

1. Solo usuarios con permiso “Aplicar descuentos” pueden acceder a esta funcionalidad.
2. Usuarios sin permiso no ven opción de descuentos en la interfaz POS.

**Funcionalidad de Descuento**

1. Descuento puede aplicarse por producto o a la venta total.
2. Tipos de descuento permitidos:
   * Porcentaje (%) sobre precio o total
   * Monto fijo (en CLP)
3. Cuando se selecciona aplicar descuento:
   * Se muestra formulario para ingresar tipo, valor y motivo opcional.
   * El sistema valida que el descuento no sea mayor al subtotal o al total según aplique.
4. Descuentos deben reflejarse inmediatamente en el cálculo final del precio.

**Validaciones**

1. El descuento no puede reducir el precio total del producto o venta a menos de $0.
2. Si el descuento es incorrecto o excede límites, muestra error claro y no permite aplicar.
3. El motivo es opcional pero sugiere agregarlo para mejor control.

**Auditoría y Notificaciones**

1. Cada descuento aplicado genera notificación automática vía Gmail a todos los administradores con Gmail asociado (US-008).
2. Registro detallado en auditoría que incluye:
   * Usuario que aplicó el descuento
   * Tipo y valor del descuento
   * Motivo (si aplicado)
   * Productos afectados
   * ID de venta

**Confirmación y Visualización**

1. En POS, productos con descuento muestran visualmente:
   * Precio original tachado
   * Precio con descuento destacado
   * Total descontado visible
2. En resumen final de la venta se muestra el descuento total aplicado.

**Tareas Técnicas**

**Backend (Django)**

* Crear endpoint para aplicar descuentos manuales en la venta POS.
* Validar tipo de descuento, que no exceda monto total.
* Implementar mecanismo para notificar por correo a administradores.
* Registrar descuentos en tabla de auditoría vinculados a ventas.
* Validar permisos para aplicar descuentos.

**Frontend (React)**

* Implementar formulario emergente para ingresar descuento.
* Validar datos en tiempo real.
* Mostrar efecto visual inmediato del descuento en precios.
* Enviar datos al backend para registro y disparo de notificaciones.
* Mostrar mensajes de error o éxito.

**Configuración**

* Permiso "Aplicar descuentos" configurable por usuario.
* Integración con sistema de notificaciones por Gmail.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuarios con permiso pueden aplicar descuentos manuales durante venta.
* Descuentos se validan y no exceden valores permitidos.
* Notificación por correo enviada correctamente.
* Auditoría registra cada descuento aplicado.
* Visualización clara y actualizada en POS.

**Seguridad**

* Validación de permisos en backend.
* Control para evitar descuentos negativos o mayores al valor.

**Experiencia de Usuario**

* Formulario claro y sencillo.
* Mensajes en tiempo real durante ingreso del descuento.
* Visualización clara de descuentos aplicados.

**Testing (Prioritarios)**

* Aplicación correcta de descuento permitido.
* Bloqueo cuando descuento excede subtotal o total.
* Envío de notificación por correo.
* Registro correcto en auditoría.

**Documentación**

* Endpoint documentado con ejemplos de solicitud y respuesta.
* Especificaciones de permisos y formatos.

**Dependencias**

* US-001: Autenticación con permisos cargados.
* US-008: Gestión de Gmail para notificaciones.
* Sistema de auditoría funcional.

**Notas Técnicas**

* La notificación por correo debe incluir detalles relevantes sin sobrecargar datos.
* El cálculo de descuentos debe usar precisión y evitar redondeos erróneos.
* Se puede expandir para descuentos promocionales automáticos en futuras fases.

US-024: Cierre de Caja Básico

Como administrador o vendedor responsable  
Quiero realizar un cierre de caja al final del día  
Para conciliar las ventas registradas, los métodos de pago y detectar diferencias o errores en la recaudación diaria

**Criterios de Aceptación**

**Acceso y Permisos**

1. Opción “Cierre de caja” visible en el menú principal solo para usuarios con rol administrador o con permiso “Cierre de caja”.
2. Cada usuario puede realizar un solo cierre de caja por día.
3. El cierre de caja solo puede realizarse para el día actual (no días anteriores ni futuros).

**Proceso de Cierre**

1. Al acceder, el sistema muestra un resumen automático de:
   * Total de ventas del día, desglosado por método de pago (efectivo, transferencia, redcompra).
   * Total de ventas anuladas del día.
   * Total de descuentos aplicados en el día.
   * Detalle de ventas por usuario (si el cierre es de administrador).
2. El usuario debe ingresar los montos contados en caja para cada método de pago:
   * Campo obligatorio para cada método de pago utilizado en el día.
   * Validación de que los campos sean numéricos y mayores o iguales a cero.
3. El sistema calcula automáticamente la diferencia entre el monto registrado y el monto contado para cada método de pago.
4. Campo opcional para observaciones o comentarios del cierre.

**Validaciones**

1. No se puede realizar más de un cierre de caja por usuario y día.
2. No se puede realizar cierre si hay ventas pendientes de anulación o reservas pendientes de confirmación (opcional, configurable).
3. Si hay diferencias negativas o positivas, el sistema las resalta visualmente y solicita confirmación adicional antes de finalizar el cierre.

**Confirmación y Registro**

1. Al confirmar el cierre:
   * Se registra el cierre en la base de datos con:
     + Usuario responsable
     + Fecha y hora
     + Totales por método de pago (registrado y contado)
     + Diferencias detectadas
     + Observaciones (si las hay)
   * Se genera registro en AuditLog con todos los detalles del cierre.
   * El estado del cierre queda como “Cerrado” y no puede modificarse.
   * El cierre queda visible en el historial de cierres para consulta posterior.

**Feedback y Visualización**

1. Mensaje de éxito:
   * “Cierre de caja realizado correctamente. Diferencias detectadas: [detalle].”
2. El formulario se limpia y el usuario es redirigido al historial de cierres o dashboard.
3. El historial de cierres muestra:
   * Fecha
   * Usuario responsable
   * Totales por método de pago
   * Diferencias
   * Observaciones
   * Estado (Cerrado)

**Auditoría y Seguridad**

1. Solo usuarios autorizados pueden acceder y ejecutar el cierre.
2. Todos los intentos de cierre, exitosos o fallidos, quedan registrados en AuditLog.
3. No se permite modificar ni eliminar cierres ya realizados.

**Tareas Técnicas**

**Backend (Django)**

* Modelo CierreCaja con campos:
  + usuario (ForeignKey a Usuario)
  + fecha (DateField)
  + hora (TimeField)
  + totales\_registrados (por método de pago)
  + totales\_contados (por método de pago)
  + diferencias (por método de pago)
  + total\_descuentos
  + total\_ventas\_anuladas
  + observaciones (TextField opcional)
  + estado (Cerrado)
* Endpoint POST /api/cierre-caja/ para registrar cierre.
* Endpoint GET /api/cierre-caja/historial/ para consultar cierres anteriores.
* Validaciones:
  + Un cierre por usuario y día.
  + Validación de campos numéricos.
  + Validación de ventas/reservas pendientes (si aplica).
* Registro en AuditLog de cada cierre.
* Proceso atómico para evitar inconsistencias.

**Frontend (React)**

* Componente CashClosingPage accesible solo para usuarios autorizados.
* Visualización de resumen automático de ventas, anulaciones y descuentos.
* Formulario para ingreso de montos contados por método de pago.
* Validación en tiempo real de campos numéricos.
* Visualización de diferencias resaltadas.
* Campo de observaciones opcional.
* Botón de confirmación con advertencia si hay diferencias.
* Mensaje de éxito/error y redirección al historial de cierres.
* Componente CashClosingHistoryPage para consulta de cierres anteriores.

**Configuración**

* Permiso “Cierre de caja” asignado a administradores y responsables.
* Métodos de pago configurables.
* Parámetro para bloquear cierre si hay ventas/reservas pendientes (opcional).

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo usuarios autorizados pueden realizar cierre de caja.
* Un cierre por usuario y día.
* Resumen automático de ventas, anulaciones y descuentos.
* Ingreso y validación de montos contados.
* Cálculo y visualización de diferencias.
* Registro y auditoría completos.
* Historial de cierres accesible y no modificable.

**Seguridad**

* Endpoint y frontend restringidos por permisos.
* Validaciones robustas en backend.
* Proceso atómico y seguro.
* Auditoría de todos los intentos y cierres.

**Experiencia de Usuario**

* Interfaz clara y fácil de usar.
* Mensajes y advertencias precisos.
* Visualización intuitiva de diferencias y totales.

**Testing (Prioritarios)**

* Cierre exitoso con y sin diferencias.
* Bloqueo de segundo cierre en el mismo día.
* Validación de campos y diferencias.
* Registro correcto en historial y auditoría.

**Documentación**

* Endpoints /api/cierre-caja/ y /api/cierre-caja/historial/ documentados.
* Ejemplos de uso y parámetros.
* Explicación de validaciones y flujo.

**Dependencias**

* US-015: Registro de ventas funcional.
* US-026: Anulación de ventas implementada.
* Permisos y roles configurados.
* Auditoría activa.

**Notas Técnicas**

* El cierre de caja es fundamental para conciliación diaria y control de errores.
* Las diferencias deben ser fácilmente identificables y justificables.
* El historial de cierres es clave para auditoría y revisiones contables.
* En futuras versiones, se puede agregar exportación de cierres y reportes automáticos.

US-025: Confirmar Reserva como Venta desde POS

Como vendedor o administrador autorizado  
Quiero poder confirmar una reserva existente directamente desde el Punto de Venta (POS)  
Para convertir dicha reserva en una venta efectiva, ajustando el inventario y registrando correctamente la transacción

**Criterios de Aceptación**

**Acceso y Permisos**

1. La opción “Confirmar Reserva como Venta” es visible en el POS para usuarios con permiso “Confirmar reservas” y administradores.
2. Esta acción es accesible cuando se selecciona una reserva pendiente dentro del POS o desde el módulo de reservas.

**Interfaz y Flujo**

1. Al seleccionar “Confirmar Reserva como Venta”, se despliega un modal o pantalla con:
   * Código de reserva.
   * Detalles del cliente (RUT, nombre, apellido, teléfono).
   * Lista detallada de productos reservados (nombre, cantidad, precio unitario, subtotal).
   * Total a pagar calculado.
   * Campo obligatorio para seleccionar método de pago: Efectivo, Transferencia, Redcompra.
   * Campo opcional para observaciones o comentarios.
2. El método de pago debe ser obligatorio para habilitar la opción de confirmar venta.

**Validaciones Previas**

1. La reserva debe estar en estado “Pendiente”.
2. La cantidad de productos reservados aún debe estar disponible en stock reservado.
3. El método de pago debe estar seleccionado.
4. En caso de que el stock reservado haya cambiado y sea insuficiente, se debe bloquear la confirmación, mostrando un mensaje:
   * “Stock reservado insuficiente para confirmar la venta. Por favor, ajuste la reserva o actualice el inventario.”

**Proceso de Confirmación**

1. Al confirmar la venta:
   * Se registra la venta en la base de datos con todos los detalles: productos, cantidades, total, método de pago, vendedor y cliente.
   * La reserva cambia a estado “Confirmada / Venta Realizada”.
   * El stock reservado de cada producto disminuye según cantidad vendida.
   * El stock disponible permanece ajustado (ya descontado en reserva).
   * Fecha y usuario que confirma la venta quedan registrados.
2. La venta queda vinculada a la reserva para trazabilidad.

**Retroalimentación y Auditoría**

1. Se muestra mensaje de éxito claro:
   * “Venta realizada exitosamente, reserva confirmada.”
2. El modal se cierra automáticamente.
3. El historial de ventas refleja esta transacción con referencia a la reserva.
4. Se registra auditoría completa de la confirmación con todos los datos para seguimiento.

**Tareas Técnicas**

**Backend (Django)**

* Endpoint PATCH /api/pos/reservas/{id}/confirmar-venta/
* Validaciones de estado y stock reservado.
* Registro en tabla Venta con asociación a Reserva.
* Actualización atómica del stock reservado.
* Cambio de estado en Reserva.
* Registro en AuditLog de la acción.
* Manejo de errores claros y consistentes.

**Frontend (React)**

* Modal ConfirmReservationAsSaleModal con datos de reserva y formulario para método de pago.
* Validaciones en frontend para activación de botón Confirmar.
* Llamada PATCH al backend al confirmar.
* Mensajes de error o éxito con detalle.
* Actualización visual del inventario y estados.
* Refresco del listado de reservas si fuera visible.

**Configuración**

* Permiso “Confirmar reservas” activo.
* Métodos de pago configurables.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Usuarios autorizados pueden confirmar reservas en venta.
* Validaciones estrictas bloquean acciones inválidas.
* Stock reservado ajustado correctamente.
* Reserva cambia a estado “Confirmada”.
* Venta registrada y vinculada a reserva.
* Mensajes de usuario claros y concisos.
* Auditoría completa incluida.

**Seguridad**

* Endpoint restringido a permisos y roles definidos.
* Validaciones robustas en backend garantizan integridad.
* Acceso frontend limitado según permisos.

**Experiencia de Usuario**

* Modal con información completa y clara.
* Interacción sencilla e intuitiva.
* Confirmación visible y feedback inmediato.

**Testing (Prioritarios)**

* Confirmación exitosa con stock suficiente.
* Bloqueo correcto con stock insuficiente.
* Registro de auditoría completo y válido.
* Restricción de acceso para usuarios sin permiso.

**Documentación**

* Endpoint documentado con ejemplos y parámetros.
* Explicación clara de roles y permisos requeridos.
* Ejemplo de flujo de confirmación.

**Dependencias**

* US-017, US-018 y US-024 completadas y funcionando.
* Sistema de permisos configurado.
* Auditoría activa para escritura.

**Notas Técnicas**

* Confirmar que la operación es atómica para mantener consistencia de stock.
* En caso de fallo, se indica claramente al usuario el motivo sin perder datos.
* La vinculación entre venta y reserva es fundamental para seguimiento posterior.
* El método de pago se guarda para futuros reportes y conciliaciones.
* Para futuras funcionalidades, esta confirmación podrá desencadenar emisión electrónica (boleta/factura).

US-026: Anulación de Venta (Corrección de Error en POS)

Como administrador  
Quiero poder anular una venta registrada por error en el sistema  
Para corregir el inventario, mantener la trazabilidad y asegurar la exactitud de las transacciones

**Criterios de Aceptación**

**Acceso y Permisos**

1. Solo usuarios con rol Administrador pueden anular ventas.
2. Botón “Anular venta” visible en el historial de ventas, únicamente para administradores y solo en ventas del día actual.
3. Vendedores y otros roles no ven ni pueden acceder a esta opción.

**Proceso de Anulación**

1. Al seleccionar “Anular venta”, se abre un modal de confirmación que incluye:
   * Detalle completo de la venta:
     + ID de venta
     + Fecha y hora
     + Vendedor responsable
     + Método de pago
     + Productos vendidos (nombre, cantidad, precio unitario, subtotal)
     + Total de la venta
   * Campo obligatorio para motivo de anulación (mínimo 10 caracteres).
   * Advertencia visual:
     + “⚠️ Esta acción es irreversible y afectará el inventario y los reportes.”
   * Botones “Confirmar anulación” (rojo) y “Cancelar” (gris).

**Validaciones**

1. Solo pueden anularse ventas del día actual (no ventas históricas).
2. No se puede anular una venta que ya esté en estado “Anulada”.
3. El motivo de anulación es obligatorio y debe tener al menos 10 caracteres.
4. Si la venta fue generada por confirmación de reserva:
   * El stock reservado de cada producto se incrementa por la cantidad anulada.
   * El estado de la reserva asociada vuelve a “Pendiente” y queda visible para nueva confirmación o cancelación.
5. Si la venta fue directa (sin reserva):
   * El stock disponible de cada producto se incrementa por la cantidad anulada.
6. El proceso es atómico: si ocurre un error, no se modifica nada.

**Proceso Automático**

1. Al confirmar la anulación:
   * El estado de la venta cambia a “Anulada”.
   * El inventario se ajusta automáticamente según el tipo de venta (directa o por reserva).
   * Se crea un registro en la tabla MovimientoInventario con tipo “anulación”, incluyendo:
     + Producto afectado
     + Cantidad restituida
     + Stock anterior y nuevo
     + Motivo de anulación
     + Usuario responsable
     + Fecha y hora
   * Se registra la acción en AuditLog con todos los detalles:
     + ID de venta
     + Usuario administrador
     + Motivo
     + Productos y cantidades
     + Estado antes y después

**Feedback y Visualización**

1. Mensaje de éxito:
   * “Venta anulada correctamente. El stock ha sido restituido y la auditoría registrada.”
2. El modal se cierra automáticamente tras éxito.
3. La venta aparece como “Anulada” en el historial, con badge rojo y motivo visible.
4. El inventario se actualiza en tiempo real para todos los usuarios.
5. Si la venta estaba asociada a una reserva, la reserva vuelve a estado “Pendiente” y es visible en el módulo de reservas.

**Restricciones de Seguridad**

1. Endpoint de anulación solo accesible para administradores autenticados.
2. Validaciones de fecha, estado y motivo se ejecutan en backend antes de modificar datos.
3. Intentos de anulación no autorizados quedan registrados en AuditLog.

**Tareas Técnicas**

**Backend (Django)**

* Endpoint PATCH /api/ventas/{id}/anular/ accesible solo para administradores.
* Validaciones:
  + Venta debe ser del día actual.
  + Estado debe ser “Normal” (no “Anulada”).
  + Motivo obligatorio y longitud mínima.
* Proceso atómico:
  + Cambia estado de venta a “Anulada”.
  + Ajusta inventario según tipo de venta (directa o por reserva).
  + Si por reserva, actualiza estado de reserva asociada.
  + Crea registro en MovimientoInventario con tipo “anulación”.
  + Registra acción en AuditLog con todos los detalles.
* Manejo de errores claros y consistentes.

**Frontend (React)**

* Botón “Anular venta” visible solo para administradores en historial de ventas.
* Modal AnularVentaModal con:
  + Detalle completo de la venta.
  + Campo para motivo obligatorio.
  + Advertencia visual.
  + Botones de confirmación y cancelación.
* Validación en frontend para motivo mínimo 10 caracteres.
* Llamada PATCH al endpoint de backend.
* Mensaje de éxito/error y actualización visual inmediata.
* Badge “Anulada” en historial y motivo visible.

**Configuración**

* Permiso “Anular ventas” solo para administradores.
* Restricción de anulación a ventas del día actual (configurable en settings).

**Definición de Hecho (DoD)**

**Funcionalidad**

* Solo administradores pueden anular ventas.
* Modal muestra todos los datos relevantes y advierte sobre la acción.
* Motivo obligatorio y validado.
* Estado de venta cambiado correctamente a “Anulada”.
* Inventario ajustado según tipo de venta.
* Reserva asociada vuelve a “Pendiente” si aplica.
* Auditoría y movimientos registrados con detalle.
* Mensaje de éxito y feedback visual inmediato.

**Seguridad**

* Endpoint restringido y validaciones robustas en backend.
* Proceso atómico para evitar inconsistencias.
* Auditoría de intentos no autorizados.

**Experiencia de Usuario**

* Interfaz clara y advertencias visibles.
* Validaciones inmediatas y mensajes precisos.
* Actualización en tiempo real de historial y inventario.

**Testing (Prioritarios)**

* Anulación exitosa de venta directa y por reserva.
* Bloqueo de anulación en ventas históricas o ya anuladas.
* Validación de motivo obligatorio.
* Registro correcto en MovimientoInventario y AuditLog.
* Restricción de acceso para usuarios no administradores.

**Documentación**

* Endpoint PATCH /api/ventas/{id}/anular/ documentado con ejemplos y parámetros.
* Explicación de flujo y validaciones.
* Detalle de auditoría y movimientos.

**Dependencias**

* US-015: Registro de venta funcional.
* US-017/US-018/US-025: Gestión de reservas y ventas asociadas.
* Inventario y auditoría activos.

**Notas Técnicas**

* El proceso debe ser transaccional para evitar inconsistencias en inventario y ventas.
* El motivo de anulación es clave para trazabilidad y análisis de errores.
* La reversión de reservas permite reutilizar el stock para nuevos clientes.
* El badge “Anulada” y motivo deben ser visibles en historial para transparencia.
* En futuras versiones, se puede permitir anulación de ventas históricas con autorización especial.

US-027: Documentación Básica de Uso y Soporte para el MVP

Como usuario final, administrador o equipo de soporte  
Quiero contar con documentación clara, accesible y completa sobre el funcionamiento del sistema  
Para poder operar el MVP correctamente, resolver dudas frecuentes y transmitir el valor de la aplicación al cliente

**Criterios de Aceptación**

**Contenido de la Documentación**

1. La documentación debe cubrir todos los flujos y funcionalidades must have del MVP, incluyendo:
   * Acceso y login al sistema
   * Gestión de usuarios y permisos (creación, edición, roles)
   * Consulta, búsqueda, creación, edición, ajuste y eliminación de productos en inventario
   * Registro de ventas en POS, con y sin descuentos
   * Aplicación de descuentos manuales y validaciones
   * Creación, confirmación y cancelación de reservas (desde módulo y POS)
   * Anulación de ventas y reversión de inventario
   * Cierre de caja y conciliación de métodos de pago
   * Validación de permisos y roles en todas las acciones
   * Visualización y exportación de historial de ventas y reservas
   * Testing manual y checklist de validación final
2. Cada sección debe incluir:
   * Descripción funcional y objetivo
   * Pasos detallados para ejecutar cada acción
   * Capturas de pantalla o diagramas (si es posible)
   * Ejemplos de uso y casos frecuentes
   * Mensajes de error y cómo resolverlos
   * Preguntas frecuentes (FAQ) y soluciones rápidas

**Acceso y Visualización**

1. La documentación debe estar disponible desde el sistema:
   * Link visible en el menú principal o dashboard (“Ayuda” o “Documentación”)
   * Acceso sin restricciones para todos los usuarios
2. Debe estar disponible en formato digital:
   * Markdown, PDF, o página web simple
   * Descargable para consulta offline

**Actualización y Mantenimiento**

1. La documentación debe ser actualizada tras cada cambio crítico en el sistema.
2. Debe incluir fecha de última actualización y responsable de la documentación.

**Valor para el Cliente**

1. La documentación debe resaltar los beneficios y el valor agregado del sistema:
   * Agilidad en ventas y reservas
   * Control y trazabilidad de inventario
   * Seguridad y gestión de permisos
   * Facilidad de uso y soporte
   * Impacto en la gestión diaria y toma de decisiones

**Soporte y Contacto**

1. Debe incluir sección de contacto para soporte técnico o dudas:
   * Correo electrónico, WhatsApp, o formulario de contacto
   * Horarios de atención y tiempos de respuesta estimados

**Tareas Técnicas**

**Backend y Frontend**

* Redacción de documentación en formato Markdown, PDF o web simple.
* Integración de link visible en la aplicación.
* Inclusión de capturas de pantalla y diagramas explicativos.
* Revisión y validación por el equipo antes de entrega al cliente.
* Actualización tras cambios críticos.

**Configuración**

* Acceso universal a la documentación desde el sistema.
* Formato legible y navegable.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Documentación cubre todos los flujos y funcionalidades must have.
* Acceso fácil y universal desde la aplicación.
* Formato digital y descargable.
* Secciones claras, ejemplos y FAQ incluidos.

**Seguridad**

* Acceso sin restricciones, pero sin exponer datos sensibles.
* Actualización tras cambios críticos.

**Experiencia de Usuario**

* Navegación intuitiva y contenido claro.
* Ejemplos y capturas facilitan comprensión.
* Sección de contacto y soporte disponible.

**Testing (Prioritarios)**

* Revisión de contenido por equipo y usuarios de prueba.
* Validación de acceso y descarga.
* Pruebas de comprensión y utilidad (feedback de usuarios).

**Documentación**

* Documento principal en Markdown, PDF o web.
* Secciones, ejemplos, capturas y FAQ.
* Registro de actualizaciones y responsable.

**Dependencias**

* Todas las funcionalidades must have implementadas y probadas.
* Acceso a entorno de producción o demo para capturas y ejemplos.

**Notas Técnicas**

* La documentación es clave para transmitir el valor del sistema al cliente y facilitar la adopción.
* Debe ser actualizada y mantenida como parte del ciclo de vida del producto.
* En futuras versiones, se puede expandir con tutoriales en video, chat de soporte y manuales avanzados.

US-028: Validación y Control de Permisos en Todo el Sistema

Como sistema  
Quiero validar los permisos de cada usuario en cada acción crítica  
Para garantizar la seguridad, trazabilidad y evitar accesos indebidos o acciones no autorizadas

**Criterios de Aceptación**

**Validación de Permisos en Backend**

1. Todos los endpoints críticos (ventas, reservas, ajustes, anulaciones, cierres de caja, descuentos) verifican el permiso del usuario antes de ejecutar la acción.
2. Si el usuario no tiene el permiso requerido, el endpoint retorna error 403 Forbidden con mensaje:
   * “No tienes permiso para realizar esta acción.”
3. Los permisos se gestionan por rol (administrador, vendedor, responsable) y pueden ser configurados granularmente por el administrador del sistema.
4. Permisos incluyen, pero no se limitan a:
   * Registrar venta
   * Aplicar descuentos
   * Crear, confirmar y cancelar reservas
   * Ajustar inventario
   * Eliminar producto
   * Anular venta
   * Realizar cierre de caja
   * Ver historial de ventas y reservas

**Validación de Permisos en Frontend**

1. La interfaz oculta botones, acciones y opciones que el usuario no puede ejecutar según sus permisos.
2. Si el usuario intenta acceder a una ruta o acción no permitida (por manipulación de URL o error), se muestra mensaje claro:
   * “Acceso denegado. No tienes permiso para ver esta página.”
3. Los permisos se cargan y actualizan en el contexto de usuario al iniciar sesión y tras cualquier cambio de rol/permisos.

**Auditoría de Intentos No Autorizados**

1. Todo intento de acceso o acción no autorizada queda registrado en AuditLog con:
   * Usuario
   * Fecha y hora
   * Acción intentada
   * Permiso requerido
   * Resultado: “Denegado”
   * IP y dispositivo si es posible

**Configuración y Gestión**

1. El administrador puede modificar permisos de usuarios desde el módulo de gestión de usuarios.
2. Cambios de permisos se aplican en tiempo real y quedan registrados en AuditLog.

**Feedback y Mensajes**

1. Mensajes de error y acceso denegado son consistentes en todo el sistema, tanto en backend como frontend.
2. El usuario nunca ve opciones que no puede ejecutar, evitando confusión y errores.

**Tareas Técnicas**

**Backend (Django)**

* Implementar decoradores y validaciones de permisos en todos los endpoints críticos.
* Configurar modelo de permisos granular por rol y usuario.
* Registrar intentos fallidos en AuditLog con todos los detalles.
* Endpoint para modificar permisos de usuario, accesible solo para administradores.
* Actualización de permisos en tiempo real tras cambios.

**Frontend (React)**

* Cargar permisos en contexto de usuario al login y tras cambios.
* Ocultar/mostrar botones y acciones según permisos.
* Validar acceso a rutas protegidas y mostrar mensajes claros.
* Actualizar interfaz tras cambios de permisos.
* Mensajes de error uniformes y consistentes.

**Configuración**

* Permisos definidos y documentados por acción y rol.
* Interfaz de gestión de permisos accesible solo para administradores.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Todas las acciones críticas verifican permisos antes de ejecutarse.
* Acciones y rutas no permitidas están bloqueadas y ocultas.
* Intentos no autorizados quedan registrados en auditoría.
* Permisos pueden modificarse y se aplican en tiempo real.

**Seguridad**

* Validaciones robustas en backend y frontend.
* Auditoría completa de intentos y cambios de permisos.
* Acceso a gestión de permisos restringido a administradores.

**Experiencia de Usuario**

* Interfaz clara, sin opciones no permitidas.
* Mensajes de error y acceso denegado uniformes.
* Cambios de permisos reflejados inmediatamente.

**Testing (Prioritarios)**

* Pruebas de acceso a acciones y rutas según permisos.
* Pruebas de ocultamiento de botones y opciones.
* Pruebas de registro de intentos no autorizados.
* Pruebas de modificación y actualización de permisos.

**Documentación**

* Documentar todos los permisos por acción y rol.
* Documentar endpoints y lógica de validación.
* Ejemplos de mensajes y flujos de error.

**Dependencias**

* US-001: Login y gestión de usuarios funcional.
* Todas las historias de acciones críticas (ventas, reservas, ajustes, etc.).
* Modelo AuditLog activo.

**Notas Técnicas**

* La validación de permisos es fundamental para la seguridad y trazabilidad del sistema.
* El registro de intentos fallidos permite detectar intentos de acceso indebido y posibles brechas.
* La gestión granular de permisos facilita la adaptación a cambios organizacionales.
* En futuras versiones, se puede agregar notificación automática al administrador ante intentos reiterados de acceso no autorizado.

¿Quieres que continúe con la siguiente historia must have (testing manual y validación de flujo completo), o prefieres revisar/ajustar esta antes de avanzar?

US-029: Testing Manual y Validación de Flujo Completo del MVP

Como equipo de desarrollo  
Quiero realizar testing manual exhaustivo de todos los flujos críticos del sistema  
Para asegurar que el MVP es funcional, seguro y cumple los objetivos del negocio antes de la entrega

**Criterios de Aceptación**

**Alcance de Testing Manual**

1. Se deben probar manualmente todos los flujos críticos definidos en el MVP, incluyendo:
   * Login y gestión de usuarios (creación, edición, permisos)
   * Consulta, búsqueda, creación, edición, ajuste y eliminación de productos en inventario
   * Registro de ventas en POS, con y sin descuentos
   * Aplicación y validación de descuentos manuales
   * Creación, confirmación y cancelación de reservas (desde módulo y POS)
   * Anulación de ventas y reversión de inventario
   * Cierre de caja y conciliación de métodos de pago
   * Validación de permisos en todas las acciones críticas
   * Visualización y exportación de historial de ventas y reservas
2. Cada flujo debe ser probado por al menos dos usuarios con roles distintos (administrador y vendedor).

**Documentación de Casos de Prueba**

1. Se debe preparar una lista de casos de prueba para cada flujo, incluyendo:
   * Objetivo del caso
   * Pasos a seguir
   * Datos de entrada
   * Resultado esperado
   * Resultado obtenido (OK/ERROR)
   * Observaciones y bugs detectados
2. Los casos de prueba deben cubrir:
   * Flujos exitosos (happy path)
   * Flujos con errores y validaciones (edge cases)
   * Accesos no autorizados y bloqueos por permisos

**Corrección de Errores**

1. Todos los bugs críticos detectados deben ser corregidos antes de la entrega.
2. Se debe documentar cada bug corregido, con referencia al caso de prueba y commit asociado.

**Validación de Seguridad**

1. Pruebas de acceso indebido:
   * Intentos de acceder a acciones o rutas sin permiso
   * Manipulación de datos en frontend y backend
   * Validación de mensajes y bloqueos
2. Pruebas de consistencia de inventario tras ventas, anulaciones, reservas y ajustes.

**Validación de Experiencia de Usuario**

1. Pruebas de mensajes, feedback visual, estados de carga y error.
2. Pruebas de actualización en tiempo real de inventario, historial y badges.

**Entregables**

1. Documento de casos de prueba y resultados (en formato Markdown, Excel o PDF).
2. Listado de bugs corregidos y pendientes (si los hay).
3. Checklist de validación de funcionalidades y seguridad.

**Tareas Técnicas**

**Backend y Frontend**

* Preparar y ejecutar casos de prueba manuales para cada flujo.
* Documentar resultados y bugs.
* Corregir bugs críticos y validar correcciones.
* Validar consistencia de datos y seguridad en cada acción.
* Validar feedback visual y mensajes en frontend.

**Configuración**

* Acceso a entorno de testing con datos de prueba y usuarios de distintos roles.
* Herramienta de documentación de casos de prueba (Google Sheets, Notion, Excel, etc.).

**Definición de Hecho (DoD)**

**Funcionalidad**

* Todos los flujos críticos probados manualmente y validados.
* Bugs críticos corregidos y documentados.
* Casos de prueba y resultados disponibles para revisión.
* Checklist de funcionalidades y seguridad completado.

**Seguridad**

* Pruebas de acceso indebido y validación de bloqueos realizadas.
* Validación de permisos y roles en todas las acciones.

**Experiencia de Usuario**

* Mensajes y feedback visual validados en todos los flujos.
* Actualización en tiempo real de datos y estados.

**Testing (Prioritarios)**

* Pruebas de flujos exitosos y con errores.
* Pruebas de acceso y permisos.
* Pruebas de consistencia de inventario y datos.

**Documentación**

* Documento de casos de prueba y resultados.
* Listado de bugs y correcciones.
* Checklist de validación final.

**Dependencias**

* Todas las historias must have del MVP implementadas y funcionales.
* Acceso a entorno de testing y usuarios de prueba.

**Notas Técnicas**

* El testing manual es clave para asegurar calidad y funcionalidad antes de la entrega.
* La documentación de casos de prueba facilita la revisión y el soporte posterior.
* En futuras versiones, se puede implementar testing automatizado y unitario.

US-030: Documentación Básica de Uso y Soporte para el MVP

Como usuario final, administrador o equipo de soporte  
Quiero contar con documentación clara, accesible y completa sobre el funcionamiento del sistema  
Para poder operar el MVP correctamente, resolver dudas frecuentes y transmitir el valor de la aplicación al cliente

**Criterios de Aceptación**

**Contenido de la Documentación**

1. La documentación debe cubrir todos los flujos y funcionalidades must have del MVP, incluyendo:
   * Acceso y login al sistema
   * Gestión de usuarios y permisos (creación, edición, roles)
   * Consulta, búsqueda, creación, edición, ajuste y eliminación de productos en inventario
   * Registro de ventas en POS, con y sin descuentos
   * Aplicación de descuentos manuales y validaciones
   * Creación, confirmación y cancelación de reservas (desde módulo y POS)
   * Anulación de ventas y reversión de inventario
   * Cierre de caja y conciliación de métodos de pago
   * Validación de permisos y roles en todas las acciones
   * Visualización y exportación de historial de ventas y reservas
   * Testing manual y checklist de validación final
2. Cada sección debe incluir:
   * Descripción funcional y objetivo
   * Pasos detallados para ejecutar cada acción
   * Capturas de pantalla o diagramas (si es posible)
   * Ejemplos de uso y casos frecuentes
   * Mensajes de error y cómo resolverlos
   * Preguntas frecuentes (FAQ) y soluciones rápidas

**Acceso y Visualización**

1. La documentación debe estar disponible desde el sistema:
   * Link visible en el menú principal o dashboard (“Ayuda” o “Documentación”)
   * Acceso sin restricciones para todos los usuarios
2. Debe estar disponible en formato digital:
   * Markdown, PDF, o página web simple
   * Descargable para consulta offline

**Actualización y Mantenimiento**

1. La documentación debe ser actualizada tras cada cambio crítico en el sistema.
2. Debe incluir fecha de última actualización y responsable de la documentación.

**Valor para el Cliente**

1. La documentación debe resaltar los beneficios y el valor agregado del sistema:
   * Agilidad en ventas y reservas
   * Control y trazabilidad de inventario
   * Seguridad y gestión de permisos
   * Facilidad de uso y soporte
   * Impacto en la gestión diaria y toma de decisiones

**Soporte y Contacto**

1. Debe incluir sección de contacto para soporte técnico o dudas:
   * Correo electrónico, WhatsApp, o formulario de contacto
   * Horarios de atención y tiempos de respuesta estimados

**Tareas Técnicas**

**Backend y Frontend**

* Redacción de documentación en formato Markdown, PDF o web simple.
* Integración de link visible en la aplicación.
* Inclusión de capturas de pantalla y diagramas explicativos.
* Revisión y validación por el equipo antes de entrega al cliente.
* Actualización tras cambios críticos.

**Configuración**

* Acceso universal a la documentación desde el sistema.
* Formato legible y navegable.

**Definición de Hecho (DoD)**

**Funcionalidad**

* Documentación cubre todos los flujos y funcionalidades must have.
* Acceso fácil y universal desde la aplicación.
* Formato digital y descargable.
* Secciones claras, ejemplos y FAQ incluidos.

**Seguridad**

* Acceso sin restricciones, pero sin exponer datos sensibles.
* Actualización tras cambios críticos.

**Experiencia de Usuario**

* Navegación intuitiva y contenido claro.
* Ejemplos y capturas facilitan comprensión.
* Sección de contacto y soporte disponible.

**Testing (Prioritarios)**

* Revisión de contenido por equipo y usuarios de prueba.
* Validación de acceso y descarga.
* Pruebas de comprensión y utilidad (feedback de usuarios).

**Documentación**

* Documento principal en Markdown, PDF o web.
* Secciones, ejemplos, capturas y FAQ.
* Registro de actualizaciones y responsable.

**Dependencias**

* Todas las funcionalidades must have implementadas y probadas.
* Acceso a entorno de producción o demo para capturas y ejemplos.

**Notas Técnicas**

* La documentación es clave para transmitir el valor del sistema al cliente y facilitar la adopción.
* Debe ser actualizada y mantenida como parte del ciclo de vida del producto.
* En futuras versiones, se puede expandir con tutoriales en video, chat de soporte y manuales avanzados.

**Asociaciones (Épicas):**

**EPIC-01: Autenticación, Seguridad y Gestión de Usuarios**

* US-001: Inicio de sesión y cierre de sesión seguro
* US-003: Crear usuario con roles y permisos granulares
* US-004: Editar información y permisos de usuarios existentes
* US-005: Visualizar listado completo de usuarios, roles y permisos
* US-006: Dashboard personalizado según rol y permisos
* US-007: Recuperación de contraseña (si está en MVP)
* US-008: Auditoría y registro de acciones críticas de usuarios

**EPIC-02: Gestión Integral de Inventario y Productos para Mundo Cartas**

* US-008: Consultar inventario completo y buscar productos con filtros avanzados
* US-009: Agregar nuevos productos al inventario de forma individual
* US-010: Editar información de productos existentes en el inventario
* US-011: Realizar ajustes manuales de stock (agregar o quitar cantidades)
* US-012: Eliminar producto del inventario (con validaciones de stock y movimientos)
* US-019: Actualizar stock automáticamente tras venta (proceso automático)
* US-021: Visualización de promociones activas y badges en inventario (si aplica)
* US-022: Exportar inventario y movimientos a CSV/Excel (si es must have)

**EPIC-03: Punto de Venta (POS) y Ventas**

* US-013: Agregar productos al carrito y procesar venta
* US-014: Validación de stock y método de pago en POS
* US-015: Confirmar reserva como venta (flujo POS-reservas)
* US-016: Visualización de resumen de venta y confirmación
* US-017: Registro de venta y actualización automática de inventario
* US-018: Historial básico de ventas (consulta, filtros, exportación)
* US-020: Aplicación de descuentos manuales en venta (flujo POS)
* US-026: Anulación de ventas y reversión de inventario

**EPIC-04: Reservas y Preventas**

* US-015: Confirmar reserva como venta (flujo POS-reservas)
* US-017: Registro de venta desde reserva
* US-023: Creación, consulta y cancelación de reservas (desde módulo y POS)
* US-025: Alertas y gestión de reservas vencidas

**EPIC-05: Cierres de Caja y Conciliación**

* US-024: Cierre de caja básico (conciliación diaria)
* US-025: Historial de cierres de caja y consulta de diferencias

**EPIC-06: Gestión de Descuentos y Promociones**

* US-020: Aplicación de descuentos manuales en venta (flujo POS)
* US-021: Visualización de promociones activas y badges en inventario (si aplica)
* US-028: Gestión de promociones masivas (si es must have en MVP)

**EPIC-07: Documentación, Soporte y Capacitación**

* US-027: Documentación básica de uso y soporte para el MVP

Notas y aclaraciones:

* Algunas historias (como US-015 y US-017) cruzan POS y reservas, pero la épica principal es la que lidera el flujo (ejemplo: la confirmación de reserva como venta es principalmente POS, pero depende de reservas).
* Si tienes historias adicionales (por ejemplo, integración con boletas, reportes avanzados, carga masiva, etc.), normalmente quedarían fuera del MVP o se asignarían a épicas futuras.
* Si alguna historia no aparece aquí, dime el número y la reviso para ubicarla correctamente.

**Tareas:**

Sprint 1 – Nombre: “Inicio Seguro y Gestión de Usuarios”  
Meta Sprint 1  
“Establecer la base de seguridad y gestión de usuarios del sistema, entregando un módulo completamente funcional de autenticación y autorización. Al cierre del sprint, los usuarios podrán iniciar y cerrar sesión de forma segura, el administrador podrá crear, editar, listar y desactivar usuarios con roles y permisos granulares, y todas las rutas críticas estarán protegidas según los permisos definidos.”

TAREAS SPRINT 1 (15-21 OCTUBRE)

Historia US-001: Iniciar sesión

Tarea 1.1: Diseñar modelo de datos de Usuario

Descripción:  
Crear el modelo de base de datos para la tabla Usuario con los campos necesarios para autenticación y gestión.

Campos obligatorios:

* id (PK, auto-incremental)
* nombre\_usuario (único, no nulo)
* nombre\_completo (no nulo)
* email (único, no nulo)
* contraseña (hash, no nulo)
* rol (enum: 'Administrador', 'Vendedor')
* estado (enum: 'Activo', 'Inactivo', default 'Activo')
* fecha\_creacion (timestamp, default now)
* fecha\_actualizacion (timestamp, auto-update)

Criterio de éxito: Modelo creado en Django (models.py) con validaciones y migraciones generadas correctamente.

Tarea 1.2: Crear endpoint POST /api/auth/login/

Descripción:  
Implementar endpoint en Django REST Framework para autenticar usuarios y devolver token JWT.

Pasos:

1. Crear vista en views.py que reciba nombre\_usuario, contraseña y rol.
2. Validar credenciales contra la base de datos usando authenticate() de Django.
3. Verificar que el usuario esté activo (estado='Activo').
4. Generar token JWT con librería djangorestframework-simplejwt.
5. Devolver respuesta con token, rol, nombre completo y permisos del usuario.

Criterio de éxito: Endpoint funcional, responde 200 con token si credenciales correctas, 401 si incorrectas, 403 si usuario inactivo.

Tarea 1.3: Diseñar vista de login en React

Descripción:  
Crear componente LoginPage.jsx con formulario de inicio de sesión.

Elementos:

* Logo de Mundo Cartas
* Input: Nombre de usuario (text, obligatorio)
* Input: Contraseña (password, obligatorio)
* Dropdown: Rol (Administrador/Vendedor, obligatorio)
* Botón: "Iniciar sesión"
* Mensajes de error si credenciales incorrectas o campos vacíos

Pasos:

1. Crear componente funcional con useState para manejar inputs.
2. Validar que campos no estén vacíos antes de enviar.
3. Al hacer clic en "Iniciar sesión", hacer POST a /api/auth/login/.
4. Si respuesta es exitosa (200), guardar token en localStorage y redirigir a Dashboard.
5. Si falla, mostrar mensaje de error claro.

Criterio de éxito: Vista funcional, permite login exitoso y muestra errores apropiados.

Tarea 1.4: Implementar validación de token en frontend

Descripción:  
Crear función para verificar si el usuario tiene sesión activa al cargar la aplicación.

Pasos:

1. Crear archivo authService.js con funciones getToken(), isAuthenticated(), getRole().
2. En componente principal (App.jsx), verificar token al cargar.
3. Si no hay token válido, redirigir a /login.
4. Si hay token, redirigir a /dashboard según rol.

Criterio de éxito: Usuario no puede acceder a vistas protegidas sin token válido.

Historia US-002: Cerrar sesión

Tarea 2.1: Crear botón de cierre de sesión en barra lateral

Descripción:  
Agregar botón "Cerrar sesión" en el componente de navegación lateral.

Pasos:

1. En Sidebar.jsx, agregar botón al final de la lista de navegación.
2. Al hacer clic, ejecutar función logout().
3. Función logout() debe eliminar token de localStorage y redirigir a /login.

Criterio de éxito: Al hacer clic, usuario es redirigido a login y no puede volver a vistas protegidas sin iniciar sesión nuevamente.

Tarea 2.2: Implementar endpoint POST /api/auth/logout/ (opcional)

Descripción:  
Crear endpoint para invalidar token en backend (si se usa blacklist de tokens).

Pasos:

1. Crear vista en Django que reciba token.
2. Agregar token a blacklist (si usas simplejwt).
3. Responder con mensaje de éxito.

Criterio de éxito: Token invalidado en backend, no puede usarse después de logout.

Historia US-003: Crear usuario con Roles y Permisos

Tarea 3.1: Diseñar modelo de Permisos

Descripción:  
Extender modelo de Usuario para incluir permisos granulares.

Campos adicionales en tabla Usuario:

* permiso\_aplicar\_descuentos (boolean, default False)
* permiso\_crear\_reservas (boolean, default False)
* permiso\_editar\_inventario (boolean, default False)
* permiso\_cancelar\_reservas (boolean, default False)
* permiso\_cierre\_caja (boolean, default False)

Criterio de éxito: Modelo actualizado con permisos y migraciones aplicadas.

Tarea 3.2: Crear endpoint POST /api/usuarios/

Descripción:  
Implementar endpoint para crear usuarios (solo administradores).

Pasos:

1. Crear vista en Django con decorador @permission\_required('es\_administrador').
2. Recibir datos: nombre\_completo, nombre\_usuario, email, contraseña, rol, permisos[].
3. Validar que nombre\_usuario y email sean únicos.
4. Hashear contraseña con make\_password() de Django.
5. Crear usuario en base de datos con permisos asignados.
6. Registrar acción en tabla AuditLog.

Criterio de éxito: Usuario creado correctamente, contraseña hasheada, permisos asignados, auditoría registrada.

Tarea 3.3: Diseñar modal de creación de usuario en React

Descripción:  
Crear componente CreateUserModal.jsx para formulario de creación.

Elementos:

* Input: Nombre completo (obligatorio)
* Input: Nombre de usuario (único, obligatorio)
* Input: Email (único, obligatorio)
* Input: Contraseña (visible, obligatorio)
* Botón: "Generar contraseña segura"
* Dropdown: Rol (Administrador/Vendedor, obligatorio)
* Checkboxes de permisos (solo si rol = Vendedor):
  + Aplicar descuentos
  + Crear reservas
  + Editar inventario
  + Cancelar reservas
  + Realizar cierre de caja
* Botones: "Guardar Usuario" / "Cancelar"

Pasos:

1. Crear componente con useState para manejar inputs y permisos.
2. Si rol es Administrador, ocultar checkboxes de permisos.
3. Validar campos obligatorios antes de enviar.
4. Al hacer clic en "Guardar", hacer POST a /api/usuarios/.
5. Mostrar mensaje de éxito o error según respuesta.

Criterio de éxito: Modal funcional, permite crear usuarios con permisos correctos.

Tarea 3.4: Implementar validación de permisos en frontend

Descripción:  
Ocultar opciones y botones según permisos del usuario logueado.

Pasos:

1. Al guardar token en localStorage, guardar también permisos del usuario.
2. Crear función hasPermission(permission) en authService.js.
3. En cada vista, usar función para mostrar/ocultar botones según permisos.

Ejemplo:  
Si usuario no tiene permiso\_aplicar\_descuentos, ocultar sección de descuentos en POS.

Criterio de éxito: Usuario solo ve funcionalidades permitidas según sus permisos.

Historia US-004: Editar usuario y permisos

Tarea 4.1: Crear endpoint PATCH /api/usuarios/{id}/

Descripción:  
Implementar endpoint para editar datos y permisos de usuarios (solo administradores).

Pasos:

1. Crear vista en Django con validación de permisos de administrador.
2. Recibir datos actualizados (nombre, email, rol, permisos, contraseña opcional).
3. Validar que email siga siendo único si se cambió.
4. Si se cambió contraseña, hashear con make\_password().
5. Actualizar usuario en base de datos.
6. Registrar cambio en AuditLog.

Criterio de éxito: Usuario actualizado correctamente, cambios registrados en auditoría.

Tarea 4.2: Diseñar modal de edición de usuario en React

Descripción:  
Crear componente EditUserModal.jsx con formulario precargado.

Elementos:

* Igual que modal de creación, pero con datos precargados.
* Campo nombre\_usuario no editable (solo visualización).
* Checkbox opcional: "Cambiar contraseña" (muestra input de nueva contraseña si se activa).

Pasos:

1. Al abrir modal, cargar datos del usuario seleccionado.
2. Validar cambios antes de enviar.
3. Al hacer clic en "Guardar", hacer PATCH a /api/usuarios/{id}/.
4. Mostrar mensaje de éxito o error.

Criterio de éxito: Modal funcional, permite editar usuarios sin errores.

Historia US-005: Listar usuarios

Tarea 5.1: Crear endpoint GET /api/usuarios/

Descripción:  
Implementar endpoint para obtener lista de usuarios (solo administradores).

Pasos:

1. Crear vista en Django con validación de permisos.
2. Devolver lista de usuarios con campos: id, nombre\_usuario, email, rol, estado, fecha\_creacion.
3. Opcionalmente, agregar paginación (50 usuarios por página).

Criterio de éxito: Endpoint devuelve lista completa de usuarios correctamente.

Tarea 5.2: Diseñar vista de gestión de usuarios en React

Descripción:  
Crear componente UsersPage.jsx con tabla de usuarios.

Elementos:

* Tabla con columnas: Nombre de usuario, Email, Rol, Estado, Fecha de creación, Acciones.
* Botones por usuario: Ver detalles, Editar, Desactivar/Activar.
* Botón principal: "Crear Usuario" (abre modal).

Pasos:

1. Al cargar página, hacer GET a /api/usuarios/.
2. Renderizar tabla con datos recibidos.
3. Botón "Crear Usuario" abre CreateUserModal.
4. Botón "Editar" abre EditUserModal con datos del usuario.
5. Botón "Desactivar" hace PATCH cambiando estado a 'Inactivo'.

Criterio de éxito: Vista funcional, muestra usuarios y permite crear/editar.

Historia US-024: Validación y control de permisos en todo el sistema

Tarea 24.1: Implementar middleware de validación de permisos en backend

Descripción:  
Crear decorador reutilizable para proteger endpoints según permisos.

Pasos:

1. Crear archivo permissions.py con decoradores:
   * @require\_admin: Solo administradores.
   * @require\_permission('nombre\_permiso'): Valida permiso específico.
2. En cada endpoint protegido, agregar decorador correspondiente.
3. Si usuario no tiene permiso, devolver 403 Forbidden.

Criterio de éxito: Endpoints protegidos, usuarios sin permiso no pueden ejecutar acciones restringidas.

Tarea 24.2: Implementar guardas de rutas en frontend

Descripción:  
Crear componentes de protección de rutas en React.

Pasos:

1. Crear componente PrivateRoute.jsx que valide token antes de renderizar.
2. Crear componente PermissionRoute.jsx que valide permiso específico.
3. Envolver rutas protegidas con estos componentes.

Ejemplo:

jsx

<PermissionRoute permission="editar\_inventario" component={InventoryPage} />

Criterio de éxito: Usuario no puede acceder a rutas sin permiso, es redirigido a página de error o dashboard.

Tarea 24.3: Implementar tabla de auditoría (AuditLog)

Descripción:  
Crear modelo de base de datos para registrar todas las acciones críticas.

Campos obligatorios:

* id (PK)
* usuario\_id (FK a Usuario)
* accion (texto: "Creó usuario", "Editó producto", etc.)
* entidad\_afectada (texto: "Usuario", "Producto", etc.)
* id\_entidad (int: ID del objeto afectado)
* datos\_anteriores (JSON, nullable)
* datos\_nuevos (JSON)
* ip\_usuario (varchar, opcional)
* timestamp (datetime, auto)

Criterio de éxito: Modelo creado, migraciones aplicadas, registros se guardan automáticamente en acciones críticas.

Tarea 24.4: Registrar auditoría en todas las acciones críticas

Descripción:  
Integrar registro de auditoría en endpoints de creación, edición, eliminación.

Pasos:

1. En cada endpoint crítico (crear usuario, editar producto, procesar venta, etc.), agregar llamada a función registrar\_auditoria().
2. Función debe recibir: usuario, acción, entidad, id, datos anteriores, datos nuevos.
3. Guardar registro en tabla AuditLog.

Criterio de éxito: Todas las acciones críticas quedan registradas en auditoría con detalles completos.

Resumen de tareas para cronograma Sprint 1 (15-21 octubre)

| **Tarea** | **Historia** | **Nombre** | | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- | --- |
| 1.1 | US-001 | Diseñar modelo de datos de Usuario | | Backend | 2 horas |
| 1.2 | US-001 | Crear endpoint POST /api/auth/login/ | | Backend | 3 horas |
| 1.3 | US-001 | Diseñar vista de login en React | | Frontend | 4 horas |
| 1.4 | US-001 | Implementar validación de token en frontend | | Frontend | 2 horas |
| 2.1 | US-002 | Crear botón de cierre de sesión | | Frontend | 1 hora |
| 2.2 | US-002 | Implementar endpoint POST /api/auth/logout/ | | Backend | 1 hora |
| 3.1 | US-003 | Diseñar modelo de Permisos | | Backend | 2 horas |
| 3.2 | US-003 | Crear endpoint POST /api/usuarios/ | | Backend | 4 horas |
| 3.3 | US-003 | Diseñar modal de creación de usuario | | Frontend | 5 horas |
| 3.4 | US-003 | Implementar validación de permisos en frontend | | Frontend | 3 horas |
| 4.1 | US-004 | Crear endpoint PATCH /api/usuarios/{id}/ | | Backend | 3 horas |
| 4.2 | US-004 | Diseñar modal de edición de usuario | | Frontend | 4 horas |
| 5.1 | US-005 | Crear endpoint GET /api/usuarios/ | | Backend | 2 horas |
| 5.2 | US-005 | Diseñar vista de gestión de usuarios | | Frontend | 5 horas |
| 24.1 | US-024 | Implementar middleware de validación de permisos | Backend | | 3 horas |
| 24.2 | US-024 | Implementar guardas de rutas en frontend | | Frontend | 3 horas |
| 24.3 | US-024 | Implementar tabla de auditoría (AuditLog) | | Backend | 2 horas |
| 24.4 | US-024 | Registrar auditoría en acciones críticas | | Backend | 2 horas |

Total estimado: ~51 horas de desarrollo (distribuibles entre 2 personas en 1 semana)

Sprint 2 – Nombre: “Fundamentos de Inventario”  
Meta Sprint 2  
“Implementar el módulo de inventario esencial para Mundo Cartas, permitiendo a los usuarios consultar, buscar, crear, editar y ajustar el stock de productos. Al cierre del sprint, el inventario deberá ser totalmente funcional, con validaciones visuales y lógicas claras, y preparado para integrarse con futuras ventas automáticas.”

**TAREAS DETALLADAS PARA SPRINT 2 (22–28 Oct)**

**Historia US-007: Consultar y buscar inventario con filtros**

Tarea 7.1: Crear endpoint GET /api/inventario/

* Descripción: Implementar en Django REST Framework un endpoint que devuelva la lista de productos, con soporte de filtros por nombre, SKU, categoría y estado de stock.
* Pasos:
  1. Definir filtro backend usando django-filter.
  2. En views.py, crear InventoryListView que herede de ListAPIView.
  3. En serializers.py, exponer campos: id, sku, nombre, categoria, precio\_unitario, stock\_disponible, stock\_reservado.
  4. Añadir paginación (50 productos por página).
* Criterio de éxito: Al llamar /api/inventario/?categoria=Pokemon&stock\_\_lte=5, devuelve sólo los productos Pokémon con stock ≤5.

Tarea 7.2: Diseñar vista de inventario en React

* Descripción: Crear componente InventoryPage.jsx que muestre tabla de productos y permita filtrar.
* Pasos:
  1. Implementar barra de búsqueda (input text) con debounce de 300 ms.
  2. Dropdown de filtros: categoría, estado de stock.
  3. Al cambiar filtros, enviar GET a /api/inventario/ con parámetros.
  4. Renderizar tabla con columnas: SKU, Nombre, Categoría, Precio, Stock Disponible, Stock Reservado, Acciones.
  5. Manejar loading y “No se encontraron resultados”.
* Criterio de éxito: Vista muestra y filtra productos sin recargar la página, con feedback visual de carga.

**Historia US-008: Agregar producto individual al inventario**

Tarea 8.1: Crear endpoint POST /api/inventario/

* Descripción: Implementar endpoint que reciba datos de un nuevo producto y lo persista.
* Pasos:
  1. En serializers.py, crear ProductCreateSerializer con validaciones: SKU único, nombre obligatorio, precio ≥0, cantidad inicial ≥0.
  2. En views.py, añadir InventoryCreateView que herede de CreateAPIView.
  3. Validar umbral mínimo de stock (default 5) si no se envía.
  4. Registrar acción en AuditLog.
* Criterio de éxito: Al POSTear SKU nuevo, producto se crea y aparece en la tabla de inventario.

Tarea 8.2: Modal “Agregar Producto” en React

* Descripción: Crear componente AddProductModal.jsx para formulario de alta de productos.
* Pasos:
  1. Inputs: SKU, Nombre, Categoría (dropdown), Precio unitario, Cantidad inicial, Umbral stock.
  2. Validar local: campos no vacíos y valores numéricos ≥0.
  3. Al “Guardar”, POST a /api/inventario/.
  4. Mostrar éxito y refrescar lista; manejar errores del servidor.
* Criterio de éxito: Modal funciona, crea productos y actualiza inventario dinámicamente.

**Historia US-009: Editar producto existente en inventario**

Tarea 9.1: Crear endpoint PATCH /api/inventario/{id}/

* Descripción: Endpoint para editar datos no relacionados con stock (precio, nombre, categoría).
* Pasos:
  1. En serializers.py, crear ProductUpdateSerializer con validaciones: nombre, precio ≥0.
  2. En views.py, añadir InventoryUpdateView que herede de UpdateAPIView.
  3. No permitir cambiar SKU.
  4. Registrar cambio en AuditLog, con datos\_anteriores y datos\_nuevos.
* Criterio de éxito: PATCH modifica producto existente sin alterar stock; auditoría guardada.

Tarea 9.2: Modal “Editar Producto” en React

* Descripción: Crear EditProductModal.jsx con formulario precargado.
* Pasos:
  1. Al abrir modal, GET /api/inventario/{id}/ para datos iniciales.
  2. Inputs editables: Nombre, Categoría, Precio unitario, Umbral stock.
  3. Al “Guardar”, PATCH a /api/inventario/{id}/.
  4. Mostrar feedback de éxito y refrescar lista.
* Criterio de éxito: Modal actualiza producto correctamente y refleja cambios en la tabla.

**Historia US-010: Ajuste manual de stock en inventario**

Tarea 10.1: Crear endpoint POST /api/inventario/{id}/ajuste-stock/

* Descripción: Endpoint para realizar ajustes de stock, registrar motivo y cantidad.
* Pasos:
  1. En serializers.py, crear StockAdjustmentSerializer con campos: tipo (‘entrada’/‘salida’), cantidad (>0), motivo, observaciones.
  2. En views.py, añadir StockAdjustmentView que herede de CreateAPIView.
  3. Validar lógica: si ‘salida’, cantidad ≤ stock\_disponible.
  4. Actualizar stock\_disponible y persistir en MovimientoInventario con FK a Producto, usuario, timestamp.
  5. Registrar auditoría.
* Criterio de éxito: Ajuste crea registro en MovimientoInventario, modifica stock y se ve reflejado en inventario.

Tarea 10.2: Modal “Ajustar Stock” en React

* Descripción: Crear AdjustStockModal.jsx para entrada/salida de stock.
* Pasos:
  1. Inputs: Tipo (radio), Cantidad (number), Motivo (dropdown dinámico), Observaciones (textarea si motivo = Other).
  2. Validar cantidad >0 y ≤stock actual si salida.
  3. Previsualizar stock resultante en texto.
  4. Al “Confirmar”, POST a /api/inventario/{id}/ajuste-stock/.
  5. Feedback de éxito y refresco de tabla de inventario.
* Criterio de éxito: Modal realiza ajuste correctamente y muestra stock actualizado al cerrar.

## Sprint 2: Fundamentos de Inventario (22–28 Octubre)

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 7.1 | US-007 | Crear endpoint GET /api/inventario/ | Backend | 4 horas |
| 7.2 | US-007 | Diseñar vista de inventario en React | Frontend | 6 horas |
| 8.1 | US-008 | Crear endpoint POST /api/inventario/ | Backend | 4 horas |
| 8.2 | US-008 | Modal “Agregar Producto” en React | Frontend | 6 horas |
| 9.1 | US-009 | Crear endpoint PATCH /api/inventario/{id}/ | Backend | 3 horas |
| 9.2 | US-009 | Modal “Editar Producto” en React | Frontend | 5 horas |
| 10.1 | US-010 | Crear endpoint POST /api/inventario/{id}/ajuste-stock/ | Backend | 5 horas |
| 10.2 | US-010 | Modal “Ajustar Stock” en React | Frontend | 5 horas |

Cada una de estas 20–24 horas de trabajo puede distribuirse entre dos desarrolladores front/backend. Con estas tareas y la meta clara, podrán avanzar de inmediato sin perder tiempo en definir qué hacer.

¿Listo para empezar?

por favor, como antes hiciste, has una tabla mostrandome de titulo el nombre del sprint y el rango de fecha, y en la tabla las tareas que cubre el sprint, historia a la que pertenecen, nombre de la tarea, responsable sugerido (backend/frontend) y duración estimada.

**Sprint 2: Fundamentos de Inventario (22–28 Octubre)**

Total estimado: 38 horas de trabajo combinadas (distributed entre Backend y Frontend).

**Sprint 3 – Nombre: “Stock Automático y Dashboard”**

Rango de fechas: 29 Octubre – 4 Noviembre

Meta del Sprint 3:  
“Implementar la lógica de actualización automática de inventario tras ventas, la eliminación segura de productos y el dashboard personalizado por rol. Al cierre del sprint, el sistema debe descontar stock automáticamente, permitir eliminar productos con validaciones, y mostrar métricas clave en el dashboard para administradores y vendedores.”

**Tareas detalladas para Sprint 3**

**Historia US-011: Eliminar producto del inventario**

Tarea 11.1: Crear endpoint DELETE /api/inventario/{id}/

* Descripción: Implementar en Django REST Framework un endpoint que elimine un producto sólo si su stock disponible y reservado son cero.
* Pasos:
  1. En views.py, crear InventoryDeleteView que herede de DestroyAPIView.
  2. Antes de eliminar, verificar stock\_disponible == 0 y stock\_reservado == 0.
  3. Si no cumple, devolver 400 con mensaje: “No se puede eliminar un producto con stock”.
  4. Si cumple, proceder a eliminación física.
  5. Registrar acción en AuditLog.
* Criterio de éxito: Un producto con stock cero se elimina; otro con stock >0 devuelve error adecuado.

Tarea 11.2: Implementar botón “Eliminar” en React

* Descripción: En InventoryPage.jsx, conectar el botón de eliminar con el endpoint y manejar respuestas.
* Pasos:
  1. Al hacer clic en ícono de eliminar, mostrar modal de confirmación: “¿Eliminar producto [nombre]?”.
  2. Si confirma, llamar a DELETE /api/inventario/{id}/.
  3. Mostrar notificación de éxito o error según respuesta.
  4. Refrescar lista de inventario.
* Criterio de éxito: Eliminar funciona con confirmación y feedback, sin recargar manualmente la página.

**Historia US-018: Actualizar stock automáticamente tras venta**

Tarea 18.1: Ajustar lógica en endpoint de venta para stock automático

* Descripción: En el endpoint que procesa ventas (POST /api/ventas/), restar automáticamente stock.
* Pasos:
  1. En views.py para SalesCreateView, tras guardar la venta, iterar productos vendidos.
  2. Para cada producto, restar cantidad de stock\_disponible.
  3. Registrar movimiento en MovimientoInventario con tipo “salida-venta”.
  4. Registrar auditoría con AuditLog.
* Criterio de éxito: Al crear venta, stock se actualiza en base de datos y movimiento queda registrado.

Tarea 18.2: Verificar actualización en Inventario React

* Descripción: Asegurar que InventoryPage refleje los cambios de stock tras ventas.
* Pasos:
  1. Después de POST /api/ventas/, invalidar caché o refrescar data de inventario.
  2. Mostrar notificación: “Stock actualizado tras venta.”
  3. Validar badges de “Stock Bajo” o “Agotado” según nuevos valores.
* Criterio de éxito: Dashboard e inventario muestran stock correcto inmediatamente tras venta.

**Historia US-006: Dashboard personalizado por rol**

Tarea 6.1: Crear endpoint GET /api/dashboard/

* Descripción: Implementar vista que entregue métricas según rol del usuario.
* Pasos:
  1. En views.py, crear DashboardView que herede de APIView.
  2. Si rol=Administrador, devolver: total ventas hoy, número de ventas, productos stock bajo, reservas pendientes.
  3. Si rol=Vendedor, devolver: mis ventas hoy (cantidad y total), mi última venta, mis productos vendidos hoy.
  4. Registrar acceso a dashboard en auditoría (opcional).
* Criterio de éxito: /api/dashboard/ devuelve JSON con métricas correctas según rol.

Tarea 6.2: Diseñar componentes de métricas en React

* Descripción: Crear DashboardAdmin.jsx y DashboardSeller.jsx para consumir el endpoint y mostrar tarjetas.
* Pasos:
  1. En DashboardPage.jsx, usar rol de authService para decidir qué componente renderizar.
  2. Cada tarjeta es un componente reutilizable MetricCard con props: título, valor, icono.
  3. Mostrar accesos rápidos a POS, Inventario, Reservas (solo Admin).
  4. Manejar loading y errores.
* Criterio de éxito: Dashboard muestra métricas correctas y botones de acceso rápido, sin recargar la página.

**Tabla de Tareas Sprint 3**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 11.1 | US-011 | Crear endpoint DELETE /api/inventario/{id}/ | Backend | 3 horas |
| 11.2 | US-011 | Implementar botón “Eliminar” en React | Frontend | 4 horas |
| 18.1 | US-018 | Ajustar lógica en endpoint de venta para stock automático | Backend | 4 horas |
| 18.2 | US-018 | Verificar actualización en Inventario React | Frontend | 3 horas |
| 6.1 | US-006 | Crear endpoint GET /api/dashboard/ | Backend | 4 horas |
| 6.2 | US-006 | Diseñar componentes de métricas en React | Frontend | 5 horas |

Total estimado: 23 horas de trabajo.

**Sprint 4 – Nombre: “Ventas, Descuentos y Historial”**

Rango de fechas: 5 Noviembre – 11 Noviembre

Meta del Sprint 4:  
“Desarrollar el módulo de Punto de Venta (POS) completo con carrito de compras, procesamiento de ventas, aplicación de descuentos manuales y registro de historial de ventas. Al cierre del sprint, los usuarios podrán vender productos con descuentos, seleccionar métodos de pago y consultar un historial detallado de transacciones.”

**Tareas detalladas para Sprint 4**

**Historia US-012: Agregar productos al carrito y procesar venta**

Tarea 12.1: Crear endpoint POST /api/ventas/

* Descripción: Implementar en Django REST Framework el endpoint que reciba los detalles de la venta y la persista.
* Pasos:
  1. Definir modelo Venta con FK a Usuario, fecha/hora, total, método de pago, descuento aplicado.
  2. Definir modelo intermedio VentaProducto con FK a Venta y Producto, cantidad, precio unitario.
  3. En serializers.py, crear SaleCreateSerializer para recibir array de productos, descuento y método de pago.
  4. En views.py, añadir SalesCreateView que herede de CreateAPIView.
  5. Validar que cada producto tenga stock suficiente antes de guardar.
  6. Registrar auditoría y movimiento de inventario (parte de Sprint 3).
* Criterio de éxito: Al POSTear venta válida, se crea registro en Venta y VentaProducto, stock se actualiza y se devuelve respuesta 201 con resumen de la venta.

Tarea 12.2: Diseñar componente POS en React

* Descripción: Crear POSPage.jsx con sección de búsqueda y carrito integrados.
* Pasos:
  1. Reutilizar componentes de producto de inventario con imagen pequeña, nombre, precio y “Agregar al carrito”.
  2. En carrito, listar productos agregados con cantidad, subtotal individual, botones +/- y eliminar.
  3. Mostrar subtotal general y sección de descuento manual (tipo porcentaje/monto, motivo).
  4. Selector de método de pago con contorno y visual activo.
  5. Botones: “Procesar Venta” abre modal de confirmación, “Limpiar Carrito”.
  6. Al confirmar modal, llamar a POST /api/ventas/ y manejar respuesta: limpieza y notificación.
* Criterio de éxito: POS funcional en React, permite armar carrito, aplicar descuento, seleccionar pago y procesar venta con feedback correcto.

**Historia US-019: Gestión y aplicación de descuentos manuales en venta**

Tarea 19.1: Extender serializador de venta para descuento

* Descripción: Asegurar que SaleCreateSerializer acepte campos de descuento: tipo (porcentaje/fijo), valor, motivo.
* Pasos:
  1. Añadir campos descuento\_tipo, descuento\_valor, descuento\_motivo en serializer.
  2. En validación, asegurar que descuento\_valor ≤ subtotal y porcentaje ≤100.
  3. Calcular total final en create() del serializer.
* Criterio de éxito: Endpoint procesa descuentos correctamente y devuelve total ajustado.

Tarea 19.2: Validar UI de descuentos en POS React

* Descripción: Mejorar formulario de descuento en POSPage.jsx.
* Pasos:
  1. Inputs para tipo (%/CLP) y valor, con validaciones inmediatas.
  2. Textarea para motivo.
  3. Al aplicar, recalcular total en la UI sin llamar al backend.
  4. Mostrar monto descontado y total actualizado.
* Criterio de éxito: Descuento se aplica en interfaz y se envía correctamente al endpoint.

**Historia US-017: Historial básico de ventas**

Tarea 17.1: Crear endpoint GET /api/ventas/

* Descripción: Implementar en Django REST Framework un endpoint para listar ventas con filtros básicos.
* Pasos:
  1. En serializers.py, crear SaleListSerializer con campos: id, fecha, usuario, total, cantidadProductos, métodoPago, esPreventa (boolean).
  2. En views.py, añadir SalesListView que herede de ListAPIView y permita filtros por fecha, usuario, método de pago, tipo.
  3. Añadir paginación (20 ventas por página).
* Criterio de éxito: Al llamar /api/ventas/?usuario=3&tipo=preventa, devuelve ventas coincidentes con estructura correcta.

Tarea 17.2: Diseñar vista de Historial de Ventas en React

* Descripción: Crear SalesHistoryPage.jsx con tabla y filtros.
* Pasos:
  1. Implementar barra de búsqueda (ID, RUT/nombre cliente).
  2. Dropdowns para filtros: rango de fechas, vendedor, método de pago, tipo.
  3. Al cargar y cambiar filtros, llamar a GET /api/ventas/.
  4. Renderizar tabla con columnas: ID, Fecha/Hora, Vendedor, Total, Cantidad, Método, Tipo, Botón “Ver Detalle”.
  5. Modal SaleDetailModal.jsx para mostrar detalle de venta con productos, descuentos y cliente.
* Criterio de éxito: Historial muestra datos correctos, filtros funcionan y modal de detalle despliega información completa.

**Tabla de Tareas Sprint 4**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 12.1 | US-012 | Crear endpoint POST /api/ventas/ | Backend | 5 horas |
| 12.2 | US-012 | Diseñar componente POS en React | Frontend | 6 horas |
| 19.1 | US-019 | Extender serializador de venta para descuento | Backend | 3 horas |
| 19.2 | US-019 | Validar UI de descuentos en POS React | Frontend | 4 horas |
| 17.1 | US-017 | Crear endpoint GET /api/ventas/ | Backend | 4 horas |
| 17.2 | US-017 | Diseñar vista de Historial de Ventas en React | Frontend | 5 horas |

Total estimado: 27 horas de trabajo.

**Sprint 5 – Nombre: “Cierre de Caja, Pruebas y Documentación”**

Rango de fechas: 12 Noviembre – 18 Noviembre

Meta del Sprint 5:  
“Implementar el proceso de cierre de caja diario, ejecutar pruebas manuales e integrar la validación de todos los flujos completos del MVP, y completar la documentación de uso y soporte. Al cierre del sprint, el sistema deberá registrar cierres de caja con conciliación, pasar la batería de pruebas y contar con manual de usuario y guías de soporte disponibles.”

**Tareas detalladas para Sprint 5**

**Historia US-023: Cierre de caja básico**

Tarea 23.1: Crear endpoint POST /api/cierre-caja/

* Descripción: Implementar en Django un endpoint para registrar el cierre de caja diario con conciliación de métodos de pago.
* Pasos:
  1. Definir modelo CierreCaja con campos: usuario (FK), fecha (date), hora (time), monto\_efectivo\_registrado, monto\_efectivo\_contado, monto\_transferencia\_registrado, monto\_transferencia\_contado, monto\_redcompra\_registrado, monto\_redcompra\_contado, diferencias (JSON), observaciones.
  2. En serializers.py, crear CashCloseSerializer que reciba montos registrados y contados, calcule diferencias y valide datos.
  3. En views.py, añadir CashCloseCreateView que herede de CreateAPIView.
  4. Registrar auditoría del cierre.
* Criterio de éxito: Al POSTear datos válidos, se crea registro de cierre con diferencias calculadas y devuelve 201.

Tarea 23.2: Crear endpoint GET /api/cierre-caja/

* Descripción: Implementar listado de cierres realizados.
* Pasos:
  1. En views.py, añadir CashCloseListView que herede de ListAPIView.
  2. En serializers.py, crear CashCloseListSerializer con campos: fecha, hora, usuario, diferencias, observaciones.
  3. Permitir filtro por fecha.
* Criterio de éxito: GET devuelve lista de cierres con posibilidad de filtrar.

Tarea 23.3: Diseñar componente “CierreCajaPage.jsx” en React

* Descripción: Crear página con formulario de cierre y sección de historial.
* Pasos:
  1. Formularios de entrada de montos contados por cada método.
  2. Mostrar totales registrados (lectura del endpoint o cálculo previo).
  3. Calcular diferencias en tiempo real y resaltar con colores.
  4. Botón “Confirmar Cierre de Caja” abre modal de confirmación.
  5. Sección “Historial de Cierres” muestra tabla con cierres pasados y filtros por fecha.
* Criterio de éxito: Página permite registrar cierres con validación, muestra historial actualizado al guardar.

**Historia US-025: Testing manual y validación de flujo completo del MVP**

Tarea 25.1: Definir plan de pruebas unitarias y de integración

* Descripción: Elaborar documento en Markdown o Word describiendo casos de prueba unitarios e integrales para cada módulo.
* Pasos:
  1. Listar casos unitarios para endpoints críticos (login, inventario CRUD, ventas, cierre de caja).
  2. Listar casos de prueba de integración de flujos completos (login → POS → venta → cierre de caja → inventario).
  3. Asignar responsable y prioridad (crítico/alto/medio) a cada caso.
* Criterio de éxito: Documento con al menos 20 casos unitarios y 5 flujos integrales definidos.

Tarea 25.2: Ejecutar pruebas manuales de flujos completos

* Descripción: Validar manualmente los flujos críticos según plan de pruebas.
* Pasos:
  1. Preparar datos de prueba (usuarios, productos, stock, reservas).
  2. Ejecutar cada caso de flujo integral, registrar resultados (éxito/fallo).
  3. Documentar hallazgos y errores en un reporte de incidencias.
* Criterio de éxito: Todos los flujos críticos probados y reporte de incidencias con al menos 80% de casos exitosos.

**Historia US-026: Documentación básica de uso y soporte para MVP**

Tarea 26.1: Redactar Guía de Usuario

* Descripción: Crear documento en Markdown o Word con instrucciones para usuarios Administrador y Vendedor.
* Secciones:
  + Introducción al sistema
  + Login y Logout
  + Gestión de Usuarios
  + Gestión de Inventario
  + Punto de Venta y Ventas
  + Cierre de Caja
  + Reservas (breve)
* Criterio de éxito: Documento de al menos 10 páginas con capturas de pantalla o diagramas.

Tarea 26.2: Redactar Guía de Soporte Técnico

* Descripción: Incluir requerimientos del sistema, arquitectura, endpoints, credenciales de test.
* Secciones:
  + Arquitectura general (frontend, backend, base de datos)
  + Tecnologías y versiones
  + Endpoints disponibles y ejemplos de uso (cURL/Postman)
  + Procedimiento de despliegue
  + Mantenimiento y backups
* Criterio de éxito: Documento de al menos 5 páginas con ejemplos y diagramas.

Tarea 26.3: Publicar documentación en repositorio

* Descripción: Añadir archivos de documentación al repositorio GitHub y actualizar README.
* Pasos:
  1. Crear carpeta /docs con guías de usuario y soporte.
  2. Actualizar README.md con enlaces a documentos.
* Criterio de éxito: Documentación accesible desde la raíz del repositorio y enlazada en README.

**Tabla de Tareas Sprint 5**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 23.1 | US-023 | Crear endpoint POST /api/cierre-caja/ | Backend | 4 horas |
| 23.2 | US-023 | Crear endpoint GET /api/cierre-caja/ | Backend | 3 horas |
| 23.3 | US-023 | Diseñar componente CierreCajaPage.jsx | Frontend | 6 horas |
| 25.1 | US-025 | Definir plan de pruebas unitarias e integración | QA/Analista | 5 horas |
| 25.2 | US-025 | Ejecutar pruebas manuales de flujos completos | QA/Analista | 6 horas |
| 26.1 | US-026 | Redactar Guía de Usuario | Documentación | 5 horas |
| 26.2 | US-026 | Redactar Guía de Soporte Técnico | Documentación | 4 horas |
| 26.3 | US-026 | Publicar documentación en repositorio | DevOps/Documentación | 2 horas |

Total estimado: 35 horas de trabajo.

**Sprint 6 – Nombre: “Reservas Básicas”**

Rango de fechas: 19 Noviembre – 25 Noviembre

Meta del Sprint 6:  
“Implementar el módulo básico de reservas de Mundo Cartas, permitiendo crear reservas de productos, consultar su estado, confirmarlas como venta y cancelarlas. Al cierre del sprint, los usuarios podrán gestionar reservas completas con trazabilidad de quién las crea, fechas de vencimiento y liberación de stock.”

**Tareas detalladas para Sprint 6**

**Historia US-013: Crear reserva básica de producto**

Tarea 13.1: Crear endpoint POST /api/reservas/

* Descripción: Implementar en Django REST Framework un endpoint para crear una reserva que aparte stock.
* Pasos:
  1. Definir modelo Reserva con campos: usuario (FK), cliente\_rut, cliente\_nombre, cliente\_apellido, cliente\_telefono (opcional), fecha\_creacion, fecha\_vencimiento, observaciones.
  2. Definir modelo intermedio ReservaProducto con FK a Reserva y Producto, cantidad reservada.
  3. En serializers.py, crear ReservationCreateSerializer que valide stock suficiente y reste temporalmente stock\_disponible, incrementando stock\_reservado.
  4. En views.py, añadir ReservationCreateView que herede de CreateAPIView.
  5. Registrar auditoría con datos de reserva.
* Criterio de éxito: Al POSTear reserva válida, se crea Reserva y ReservaProducto, stock actualizado y devuelve 201 con datos de reserva.

Tarea 13.2: Crear modal “Nueva Reserva” en React

* Descripción: Diseñar componente NewReservationModal.jsx para capturar datos de reserva.
* Pasos:
  1. Inputs obligatorios: RUT (validar formato chileno), Nombre, Apellido; opcional: Teléfono, Observaciones.
  2. Selección de productos actuales en carrito: reutilizar CartItem pero sin procesar venta.
  3. Input: Vigencia en días (número, ≥1).
  4. Al “Confirmar Reserva”, llamar POST /api/reservas/.
  5. Mostrar feedback de éxito y refrescar lista de reservas.
* Criterio de éxito: Modal crea reservas y refleja nuevos registros en la tabla de reservas.

**Historia US-014: Confirmar reserva como venta**

Tarea 14.1: Crear endpoint POST /api/reservas/{id}/confirmar/

* Descripción: Implementar endpoint para confirmar una reserva, convirtiéndola en venta y ajustando stock reservado.
* Pasos:
  1. En views.py, crear ReservationConfirmView que herede de APIView.
  2. Validar que reserva exista y esté en estado “Pendiente”.
  3. Crear objeto Venta y VentaProducto con los mismos productos y cantidades de la reserva.
  4. Ajustar stock: reducir stock\_reservado y no volver a reducir stock\_disponible.
  5. Actualizar estado de Reserva a “Confirmada” y guardar fecha\_confirmacion.
  6. Registrar auditoría y movimiento de inventario tipo “salida-reserva”.
* Criterio de éxito: Al llamar a confirmar, devuelve venta creada y reserva cambia de estado, stock reservado actualizado.

Tarea 14.2: Implementar botón “Confirmar Venta” en React

* Descripción: En ReservationsPage.jsx, conectar botón de confirmar con endpoint.
* Pasos:
  1. Al hacer clic en “Confirmar Venta”, abrir modal para seleccionar método de pago (con contorno) y mostrar resumen.
  2. Validar selección y llamar POST /api/reservas/{id}/confirmar/ con método de pago y descuento si aplica.
  3. Mostrar notificación de éxito y refrescar tablas de reservations y sales history.
* Criterio de éxito: Confirmar funciona, reserva y venta creadas, UI actualiza estado en ambas vistas.

**Historia US-015: Ver reservas pendientes y confirmadas**

Tarea 15.1: Crear endpoint GET /api/reservas/

* Descripción: Implementar endpoint para listar reservas con filtros por estado.
* Pasos:
  1. En serializers.py, crear ReservationListSerializer con campos: id, cliente, fecha\_creacion, fecha\_vencimiento, estado, usuario\_creador.
  2. En views.py, añadir ReservationListView que herede de ListAPIView.
  3. Permitir filtro ?estado=Pendiente|Confirmada|Cancelada.
* Criterio de éxito: GET devuelve reservas filtradas correctamente según estado.

Tarea 15.2: Diseñar vista de reservas en React

* Descripción: Crear ReservationsPage.jsx con pestañas Pendientes, Confirmadas, Canceladas.
* Pasos:
  1. Al cargar, llamada a GET /api/reservas/?estado= según pestaña activa.
  2. Tabla con columnas: ID, Fecha Creación, Cliente (RUT, Nombre), Teléfono, Fecha Vencimiento, Estado, Usuario Creador, Acciones.
  3. Acciones: “Confirmar Venta” (solo pendientes), “Ver Detalle”, “Cancelar Reserva” (solo admin).
* Criterio de éxito: Vista lista reservas por estado y muestra usuario creador; cambiar pestaña actualiza datos.

**Historia US-016: Cancelar reserva (solo administradores)**

Tarea 16.1: Crear endpoint POST /api/reservas/{id}/cancelar/

* Descripción: Implementar endpoint para cancelar una reserva, liberando stock reservado.
* Pasos:
  1. En views.py, crear ReservationCancelView que herede de APIView con permiso de admin.
  2. Validar reserva en estado “Pendiente”.
  3. Reducir stock\_reservado y aumentar stock\_disponible por cantidades de ReservaProducto.
  4. Actualizar estado de Reserva a “Cancelada” y guardar fecha\_cancelacion.
  5. Registrar auditoría con motivo.
* Criterio de éxito: POST a cancelar libera stock, reserva cambia de estado y devuelve 200.

Tarea 16.2: Implementar botón “Cancelar Reserva” en React

* Descripción: En ReservationsPage.jsx, conectar acción de cancelación con confirmación.
* Pasos:
  1. Al hacer clic en “Cancelar Reserva”, abrir modal de motivo (textarea obligatorio).
  2. Validar motivo y llamar POST /api/reservas/{id}/cancelar/.
  3. Mostrar notificación y refrescar lista de reservas.
* Criterio de éxito: Cancelar funciona solo para admins, libera stock y la UI refleja cambio inmediato.

**Tabla de Tareas Sprint 6**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 13.1 | US-013 | Crear endpoint POST /api/reservas/ | Backend | 4 horas |
| 13.2 | US-013 | Modal “Nueva Reserva” en React | Frontend | 5 horas |
| 14.1 | US-014 | Crear endpoint POST /api/reservas/{id}/confirmar/ | Backend | 4 horas |
| 14.2 | US-014 | Botón “Confirmar Venta” en React | Frontend | 4 horas |
| 15.1 | US-015 | Crear endpoint GET /api/reservas/ | Backend | 3 horas |
| 15.2 | US-015 | Vista de reservas con pestañas en React | Frontend | 5 horas |
| 16.1 | US-016 | Crear endpoint POST /api/reservas/{id}/cancelar/ | Backend | 4 horas |
| 16.2 | US-016 | Botón “Cancelar Reserva” en React | Frontend | 4 horas |

Total estimado: 33 horas de trabajo.

**Sprint 7 – Nombre: “Reservas desde POS y Anulación de Ventas”**

Rango de fechas: 26 Noviembre – 2 Diciembre

Meta del Sprint 7:  
“Extender el flujo de reservas para operar directamente desde el Punto de Venta y habilitar la anulación de ventas con reversión automática de stock. Al cierre del sprint, los vendedores podrán crear reservas en el POS, confirmarlas como venta o anular ventas, y el inventario se ajustará correctamente en cada caso.”

**Tareas detalladas para Sprint 7**

**Historia US-020: Crear reserva básica desde POS**

Tarea 20.1: Extender endpoint POST /api/reservas/ para flujo POS

* Descripción: Adaptar el endpoint de reservas para aceptar datos desde el POS, incluyendo lista de productos en carrito.
* Pasos:
  1. Modificar ReservationCreateSerializer para aceptar array de productos con producto\_id y cantidad.
  2. Al crear reserva, grabar en ReservaProducto cada item y ajustar stock\_disponible y stock\_reservado.
  3. Registrar auditoría con detalle de usuario y origen “POS”.
* Criterio de éxito: POST desde POS crea reserva con productos y stock ajustado igual que desde modal de reserva.

Tarea 20.2: Integrar botón “Reservar” en POSPage.jsx

* Descripción: Añadir acción de reserva en la UI del POS, reusando modal de reserva pero con lista de productos del carrito.
* Pasos:
  1. En POSPage.jsx, agregar botón “Reservar” junto a “Procesar Venta”.
  2. Al hacer clic, abrir NewReservationModal prellenado con productos del carrito.
  3. Validar campos y llamar POST /api/reservas/.
  4. Mostrar notificación de éxito y limpiar carrito.
* Criterio de éxito: Reservas desde POS funcionan idéntico al módulo de reservas, stock y UI actualizados.

**Historia US-021: Confirmar reserva como venta desde POS**

Tarea 21.1: Extender endpoint POST /api/reservas/{id}/confirmar/ para POS

* Descripción: Ajustar lógica de confirmación de reserva para invocar desde POS con descuento y método de pago.
* Pasos:
  1. Modificar ReservationConfirmView para recibir descuento\_tipo, descuento\_valor y metodo\_pago.
  2. Crear venta con parámetros de reserva y aplicar descuento antes de grabar.
  3. Ajustar stock\_reservado, no tocar stock\_disponible.
  4. Registrar auditoría con origen “POS”.
* Criterio de éxito: Confirmar desde POS genera venta con descuento, reserva pasa a “Confirmada” y stock reservado actualizado.

Tarea 21.2: Añadir opción “Confirmar Reserva” en POS UI

* Descripción: En POSPage.jsx, mostrar lista de reservas activas y botón para confirmar venta directamente.
* Pasos:
  1. Al cargar POS, habilitar pestaña o dropdown “Reservas” que liste reservas pendientes del usuario.
  2. Cada reserva tiene botón “Confirmar” que abre modal para método de pago y descuento.
  3. Al confirmar, llamar POST /api/reservas/{id}/confirmar/.
  4. Notificar éxito y actualizar inventario y ventas.
* Criterio de éxito: Flujo completo de confirmación desde POS sin cambiar de vista, stock y ventas reflejados.

**Historia US-022: Anulación de venta (corrección de error en POS)**

Tarea 22.1: Crear endpoint POST /api/ventas/{id}/anular/

* Descripción: Implementar endpoint para anular una venta, restaurando stock y registrando motivo.
* Pasos:
  1. Definir vista SaleCancelView en views.py que herede de APIView.
  2. Validar estado de venta (solo si no ha sido anulada).
  3. Iterar productos en VentaProducto, aumentar stock\_disponible y registrar en MovimientoInventario tipo “entrada-anulación”.
  4. Marcar venta como anulada, guardar fecha\_anulacion y motivo\_anulacion.
  5. Registrar auditoría con detalle de usuario y motivo.
* Criterio de éxito: POST a anular devuelve éxito y stock queda restaurado.

Tarea 22.2: Implementar botón “Anular Venta” en POSPage.jsx

* Descripción: Agregar acción en UI de historial de ventas dentro del POS para anular directamente.
* Pasos:
  1. En componente SaleDetailModal.jsx, agregar botón “Anular Venta” visible si venta no anulada.
  2. Al hacer clic, abrir modal para ingresar motivo\_anulacion.
  3. Validar motivo y llamar POST /api/ventas/{id}/anular/.
  4. Mostrar notificación y refrescar inventario, carrito vacío y lista de ventas.
* Criterio de éxito: Anular venta restaura stock, modal de detalle actualiza estado de venta a “Anulada” y UI refleja cambio.

**Tabla de Tareas Sprint 7**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 20.1 | US-020 | Extender endpoint POST /api/reservas/ para flujo POS | Backend | 4 horas |
| 20.2 | US-020 | Integrar botón “Reservar” en POSPage.jsx | Frontend | 5 horas |
| 21.1 | US-021 | Extender endpoint POST /api/reservas/{id}/confirmar/ para POS | Backend | 4 horas |
| 21.2 | US-021 | Añadir opción “Confirmar Reserva” en POS UI | Frontend | 5 horas |
| 22.1 | US-022 | Crear endpoint POST /api/ventas/{id}/anular/ | Backend | 4 horas |
| 22.2 | US-022 | Implementar botón “Anular Venta” en SaleDetailModal.jsx | Frontend | 4 horas |

Total estimado: 26 horas de trabajo.

**Sprint 8 – Nombre: “Ajustes Finales e Integración”**

Rango de fechas: 3 Diciembre – 9 Diciembre

Meta del Sprint 8:  
“Realizar ajustes finales en el prototipo funcional, abordar incidencias halladas durante las pruebas, refinar la experiencia de usuario según feedback y preparar la presentación y demo del proyecto. Al cierre del sprint, el sistema debe estar libre de bugs críticos, con todos los flujos integrados, listo para ser entregado y presentado.”

**Tareas detalladas para Sprint 8**

**Ajustes y correcciones generales**

Tarea 8.1: Corregir bugs críticos reportados

* Descripción: Resolver errores documentados en el reporte de incidencias del Sprint 5, priorizando aquellos que bloquean flujos principales.
* Pasos:
  1. Revisar reporte de incidencias.
  2. Clasificar bugs por criticidad (bloqueante, alto, medio).
  3. Asignar bugs bloqueantes a desarrolladores para corrección inmediata.
  4. Validar correcciones con pruebas manuales.
* Criterio de éxito: 100% de bugs bloqueantes resueltos y validados.

Tarea 8.2: Refinar UI/UX según feedback de usuario

* Descripción: Implementar mejoras sugeridas por Don Claudio y testers para pulir detalles de diseño e interacción.
* Pasos:
  1. Recopilar feedback pendiente (tooltips, alineación, colores).
  2. Priorizar cambios de alto impacto UX.
  3. Aplicar correcciones en React y Figma.
  4. Validar visualmente cada cambio en demo.
* Criterio de éxito: Feedback crítico atendido, UI consistente y aprobada por usuario.

**Integración y pruebas finales**

Tarea 8.3: Realizar pruebas de integración completa

* Descripción: Ejecutar scripts de prueba automatizada (si aplican) y manual de todos los flujos del sistema.
* Pasos:
  1. Ejecutar pruebas unitarias automatizadas (pytest, jest).
  2. Ejecutar pruebas de integración: login, inventario, POS, reservas, cierre de caja.
  3. Documentar resultados y corregir fallos menores.
* Criterio de éxito: 95% de pruebas pasan sin errores; incidencias menores documentadas.

Tarea 8.4: Preparar demo y slides de presentación

* Descripción: Crear un guión de presentación, capturas de pantalla y demo en vivo para la entrega final.
* Pasos:
  1. Escribir guión destacando funcionalidades clave por historia épica.
  2. Preparar slides con screenshots y métricas del proyecto (historial, dashboard).
  3. Ensayar demo en ambiente local, asegurar datos de prueba.
* Criterio de éxito: Presentación lista, demo fluido sin errores y guión aprobado por el equipo.

Tarea 8.5: Documentar entrega final

* Descripción: Actualizar README y documentación con versión final, credenciales de demo y pasos de despliegue.
* Pasos:
  1. Añadir sección “Entrega Final” en README.md.
  2. Incluir credenciales de usuario Admin y Vendedor de demo.
  3. Describir pasos para levantar aplicación en local.
* Criterio de éxito: Documentación actualizada y verificada, entorno de demo reproducible.

**Tabla de Tareas Sprint 8**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Responsable sugerido** | **Duración estimada** |
| --- | --- | --- | --- | --- |
| 8.1 | US-025/varios | Corregir bugs críticos reportados | Backend/Frontend | 6 horas |
| 8.2 | UI/UX | Refinar UI/UX según feedback | Frontend/UX Designer | 5 horas |
| 8.3 | US-025 | Realizar pruebas de integración completa | QA/Developers | 5 horas |
| 8.4 | Presentación | Preparar demo y slides de presentación | PM/Dev Lead | 4 horas |
| 8.5 | Documentación | Documentar entrega final | DevOps/Documentation | 3 horas |

Total estimado: 23 horas de trabajo.

**Dependencias:**

| **Tarea** | **Historia** | **Nombre de la tarea** | **Depende de** |
| --- | --- | --- | --- |
| 1.1 | US-001 | Diseñar modelo de datos de Usuario | – |
| 1.2 | US-001 | Crear endpoint POST /api/auth/login/ | 1.1 |
| 1.3 | US-001 | Diseñar vista de login en React | 1.2 |
| 1.4 | US-001 | Implementar validación de token en frontend | 1.2 |
| 2.1 | US-002 | Crear botón de cierre de sesión | 1.4 |
| 2.2 | US-002 | Implementar endpoint POST /api/auth/logout/ | 1.2 |
| 3.1 | US-003 | Diseñar modelo de Permisos | 1.1 |
| 3.2 | US-003 | Crear endpoint POST /api/usuarios/ | 3.1 |
| 3.3 | US-003 | Diseñar modal de creación de usuario | 3.2 |
| 3.4 | US-003 | Implementar validación de permisos en frontend | 3.2 |
| 4.1 | US-004 | Crear endpoint PATCH /api/usuarios/{id}/ | 3.2 |
| 4.2 | US-004 | Diseñar modal de edición de usuario | 4.1 |
| 5.1 | US-005 | Crear endpoint GET /api/usuarios/ | 3.2 |
| 5.2 | US-005 | Diseñar vista de gestión de usuarios | 5.1 |
| 24.1 | US-024 | Implementar middleware de validación de permisos | 3.1 |
| 24.2 | US-024 | Implementar guardas de rutas en frontend | 1.4 & 3.4 |
| 24.3 | US-024 | Implementar tabla de auditoría (AuditLog) | 1.1 |
| 24.4 | US-024 | Registrar auditoría en acciones críticas | 24.3 |
| 7.1 | US-007 | Crear endpoint GET /api/inventario/ | 24.1 |
| 7.2 | US-007 | Diseñar vista de inventario en React | 7.1 |
| 8.1 | US-008 | Crear endpoint POST /api/inventario/ | 7.1 |
| 8.2 | US-008 | Modal “Agregar Producto” en React | 8.1 & 7.2 |
| 9.1 | US-009 | Crear endpoint PATCH /api/inventario/{id}/ | 8.1 |
| 9.2 | US-009 | Modal “Editar Producto” en React | 9.1 & 7.2 |
| 10.1 | US-010 | Crear endpoint POST /api/inventario/{id}/ajuste-stock/ | 9.1 |
| 10.2 | US-010 | Modal “Ajustar Stock” en React | 10.1 & 7.2 |
| 11.1 | US-011 | Crear endpoint DELETE /api/inventario/{id}/ | 10.1 |
| 11.2 | US-011 | Implementar botón “Eliminar” en React | 11.1 & 7.2 |
| 18.1 | US-018 | Ajustar lógica en endpoint de venta para stock automático | 1.2 & 10.1 |
| 18.2 | US-018 | Verificar actualización en Inventario React | 18.1 & 7.2 |
| 6.1 | US-006 | Crear endpoint GET /api/dashboard/ | 1.2 & 5.1 |
| 6.2 | US-006 | Diseñar componentes de métricas en React | 6.1 |
| 12.1 | US-012 | Crear endpoint POST /api/ventas/ | 18.1 |
| 12.2 | US-012 | Diseñar componente POS en React | 12.1 & 7.2 |
| 19.1 | US-019 | Extender serializador de venta para descuento | 12.1 |
| 19.2 | US-019 | Validar UI de descuentos en POS React | 19.1 & 12.2 |
| 17.1 | US-017 | Crear endpoint GET /api/ventas/ | 12.1 |
| 17.2 | US-017 | Diseñar vista de Historial de Ventas en React | 17.1 |
| 23.1 | US-023 | Crear endpoint POST /api/cierre-caja/ | 17.1 |
| 23.2 | US-023 | Crear endpoint GET /api/cierre-caja/ | 23.1 |
| 23.3 | US-023 | Diseñar componente CierreCajaPage.jsx | 23.2 & 6.2 |
| 25.1 | US-025 | Definir plan de pruebas unitarias e integración | all módulos iniciales |
| 25.2 | US-025 | Ejecutar pruebas manuales de flujos completos | 25.1 |
| 26.1 | US-026 | Redactar Guía de Usuario | all módulos funcionales |
| 26.2 | US-026 | Redactar Guía de Soporte Técnico | all módulos funcionales |
| 26.3 | US-026 | Publicar documentación en repositorio | 26.1 & 26.2 |
| 13.1 | US-013 | Crear endpoint POST /api/reservas/ | 7.1 & 18.1 |
| 13.2 | US-013 | Modal “Nueva Reserva” en React | 13.1 |
| 14.1 | US-014 | Crear endpoint POST /api/reservas/{id}/confirmar/ | 13.1 |
| 14.2 | US-014 | Botón “Confirmar Venta” en React | 14.1 |
| 15.1 | US-015 | Crear endpoint GET /api/reservas/ | 13.1 |
| 15.2 | US-015 | Vista de reservas con pestañas en React | 15.1 |
| 16.1 | US-016 | Crear endpoint POST /api/reservas/{id}/cancelar/ | 15.1 |
| 16.2 | US-016 | Botón “Cancelar Reserva” en React | 16.1 |
| 20.1 | US-020 | Extender endpoint POST /api/reservas/ para flujo POS | 13.1 |
| 20.2 | US-020 | Integrar botón “Reservar” en POSPage.jsx | 20.1 & 12.2 |
| 21.1 | US-021 | Extender endpoint POST /api/reservas/{id}/confirmar/ para POS | 14.1 |
| 21.2 | US-021 | Añadir opción “Confirmar Reserva” en POS UI | 21.1 & 12.2 |
| 22.1 | US-022 | Crear endpoint POST /api/ventas/{id}/anular/ | 12.1 |
| 22.2 | US-022 | Implementar botón “Anular Venta” en SaleDetailModal.jsx | 22.1 |

**Carta Gantt:**

Aquí se puede ver el cronograma de las tareas de manera más simplificada y rápida.

[Carta Gantt - Metodología - Mundo Cartas](https://docs.google.com/spreadsheets/d/1uVZvWagVEIsw3k_qBQZ0DC6BxF10lNWZlWI6n1QvxrY/edit?usp=sharing)