Plan de Trabajo: Sistema de Inventario para Bebidas y Desayunos

Tu Nombre y Nombre del Colaborador

May 28, 2025

Abstract

Este documento presenta un plan de trabajo estructurado para el desarrollo de un sistema de inventario modular y de bajo acoplamiento, enfocado en el negocio de bebidas y desayunos. Se detallan los conceptos clave, la caja de herramientas de Rust, patrones de diseño, consideraciones de seguridad, testing y una propuesta de demo mínima viable.

Contents

1 Introducción

El objetivo es desarrollar un sistema de inventario robusto que permita gestionar productos, ingredientes, recetas, ventas y empleados para un negocio de bebidas y desayunos. Este plan prioriza la **modularidad** y el **bajo acoplamiento** desde las primeras fases del desarrollo.

—

2 Conceptos Clave del Sistema (Alto Nivel)

Para este sistema, los conceptos fundamentales giran en torno a la gestión eficiente de recursos y la operativa de un negocio de alimentos y bebidas:

[label=•]

- **Alertas Proactivas**: Capacidad de notificar sobre la **baja existencia** de productos, un pilar para evitar la falta de stock y pérdidas de ventas.
- **Roles y Permisos**: Diferenciar las capacidades de los usuarios (**empleados**) (ej., administradores, personal de inventario, cajeros) para asegurar la integridad de los datos y la seguridad.
- **Registro de Transacciones**: Mantener un **historial** de ventas y movimientos de inventario para auditoría y análisis.
- **Persistencia de Datos**: La información del inventario, ventas y empleados debe **guardarse** de forma segura y **cargarse** al reiniciar la aplicación.

—

3 Caja de Herramientas de Rust Esencial

Aquí están las herramientas y conceptos específicos de Rust que utilizaremos para construir un sistema robusto y modular:

[label=•]

—

4 Patrones de Diseño (Alto Nivel)

Aunque no aplicaremos patrones complejos como CQRS o Event Sourcing de entrada, usaremos patrones fundamentales para la modularidad:

[label=•]

- **Servicios (Aplicación)**: Orquestra las operaciones de negocio, utilizando la lógica del dominio y los repositorios.

- **Repositorios (Infraestructura)**: Encapsula la lógica de persistencia de datos (cómo se guardan y cargan las entidades).
- **Presentación (CLI)**: La interfaz de usuario que interactúa con los servicios.

Inyección de Dependencias (mediante `traits`): Los servicios dependerán de `traits` de repositorio, no de implementaciones concretas, permitiendo flexibilidad y facilidad de prueba.

—

5 Seguridad con Hashing en Login

[label=•]

—

6 Concurrencia de Datos (Consideración Futura)

Para el demo inicial, la aplicación será **monohilo** y operará de forma secuencial. Si en el futuro el sistema necesita manejar múltiples usuarios simultáneamente o tareas en segundo plano, exploraremos conceptos como:

[label=•] —

7 Pruebas

[label=•]

—

8 Demo Básica: Inventario y Alertas

El objetivo del demo es mostrar la **funcionalidad central** de control de stock y aviso.

[label=•]

- **Listado de Inventario**: Mostrar el estado actual de todos los productos en el inventario.
- **Venta Simplificada de Recetas**:
  - Tener **2-3 recetas predefinidas** (ej. "Café Simple", "Jugo de Naranja").

    - Al "vender" una receta, el sistema descuenta los ingredientes correspondientes del inventario.
    - **Validación de Stock**: Si no hay suficientes ingredientes, la venta no procede y se notifica al usuario.
  - **Alertas de Bajo Stock**: Mostrar una lista de productos que han alcanzado o superado su `stock_minimo_alerta`.
  - **Persistencia**: Guardar y cargar el inventario en un archivo JSON para que los cambios persistan.

- **Interfaz**: Una aplicación de **línea de comandos (CLI)** simple con un menú de opciones (ej. "1. Ver Inventario", "2. Registrar Venta", "3. Ver Alertas", "4. Salir").

—

9 Plan de Trabajo Detallado (Para ti y tu Colaborador)

Este plan está diseñado para ser iterativo, construyendo funcionalidades paso a paso.

9.1 Fase 1: Configuración y Modelado Básico (1-2 días)

- `src/main.rs` (punto de entrada, CLI)
- `src/domain/mod.rs` (definición de structs: `Producto`, `IngredienteReceta`, `Receta`, `Error`)
- `src/repositories/mod.rs` (definición de `traits` de repositorio)
- `src/services/mod.rs` (definición de `traits` de servicios)
- Crea los `structs` para `Producto` y `Receta` con sus campos esenciales (ID, nombre, cantidad/ingredientes, stock mínimo).
- Define un `enum` básico para `Error` con al menos `NotFound` y `InsufficientStock`.

9.2 Fase 2: Persistencia y CRUD de Inventario (2-3 días)

- Crea un `struct InventoryService` en `src/services/mod.rs` que tome una instancia de `dyn ProductRepository` (mediante `Box<dyn ProductRepository>`) en su constructor.
- Implementa métodos para **añadir productos** y **listar todos los productos**.

- Cargar el inventario desde el archivo al inicio.
- Añadir un nuevo producto.
- Listar el inventario actual.
- Guardar los cambios al salir.

9.3 Fase 3: Recetas y Lógica de Descuento (3-4 días)

- Tome un `HashMap<String, Producto>` (el inventario) y una `Receta`.
- Verifique si hay suficiente stock de **todos los ingredientes**.
- Si hay, descuente las cantidades necesarias del inventario.
- Retorne un `Result` indicando éxito o un error de `InsufficientStock`.
- Añade un método al `InventoryService` (o crea un nuevo `SaleService` si prefieres una separación más estricta para ventas) que:
- Tome una `Receta` y una cantidad.
- Use la lógica de dominio para descontar los ingredientes.
- Actualice el inventario usando el `ProductRepository`.
- Añade una opción para "Vender Receta".
- Permite al usuario elegir una receta (ej., por su ID) y una cantidad.
- Muestra el resultado de la venta (éxito o error por falta de stock).

9.4 Fase 4: Alertas y Demo Final (2-3 días)

- Añade una opción de menú para "Ver Alertas de Bajo Stock".
- Al inicio del programa (después de cargar el inventario), muestra un resumen si hay alertas.
- Escribe pruebas unitarias para la lógica de descuento de ingredientes.
- Escribe una prueba de integración para el flujo de añadir producto y listarlo.
- Asegúrate de que los mensajes en la CLI sean claros.
- Prepara datos iniciales en tu archivo JSON para una demostración fluida.
- Practica el flujo de la demo.