TRES 3 DOS

Sistema de Gestión Integral de Eventos para Organizaciones Culturales

1. Introducción

- **Propósito**:
    - El propósito de este documento es especificar, de manera detallada y formal, los requerimientos del sistema Tres 3 Dos, una aplicación web destinada a optimizar la planificación y ejecución de eventos culturales o sociales de pequeña y mediana escala. El documento servirá como contrato entre el equipo de desarrollo y los interesados (stakeholders), brindando una referencia única para el diseño, implementación, pruebas y mantenimiento del sistema.
- **Alcance**: Se permitirá a organizaciones culturales autogestionadas o cooperativas:
    - Gestionar inventario y consumo de productos durante eventos.
    - Administrar presupuestos, ingresos, egresos y deudas.
    - Organizar y controlar la venta de entradas.
    - Distribuir responsabilidades entre varios usuarios con trazabilidad y control de roles.
- **Público destinatario:**
    - Usuarios finales:
        - Integrantes de organizaciones culturales (tesorería, organización, abastecimiento, boletería).
    - Equipo de desarrollo:
        - Programadores
        - diseñadores
        - testers
    - Responsables de mantenimiento y soporte técnico.

2. Descripción general

- **Perspectiva del producto**: El sistema se desarrollará como una aplicación web modular, con una arquitectura de tres capas:
    - Presentación
        - Interfaz web responsiva para navegadores modernos.
    - Lógica de negocio:
        - API que gestiona procesos de inventario, tesorería y eventos.
    - Acceso a datos:
        - SQLite
- **Funcionalidad del producto:**
    - Registro y control de inventario en tiempo real.

    - Administración de ingresos, egresos y deudas
    - Creación y gestión de eventos, incluyendo venta de entradas y seguimiento de ventas
    - Roles de usuario y permisos diferenciados
    - Reportes y predicciones basadas en datos históricos.
- **Características de los usuarios**:
    - **Organizadores/Tesoreros**: gestionan finanzas y pagos.
    - **Cajeros/Abastecimiento**: registran ventas y movimientos de stock.
    - **Administradores**: gestionan usuarios y permisos.

      Los usuarios poseen conocimientos básicos de informática, por lo que la interfaz debe ser simple e intuitiva.
- **Restricciones**:
    - El sistema debe funcionar en entornos con conectividad a Internet estándar.
    - Debe operar bajo licencias de software libre o de bajo costo.
    - La base de datos debe permitir respaldos automáticos.
- **Suposiciones y Dependencias**
    - Los usuarios contarán con dispositivos con navegadores actualizados.
    - Se utilizará un servidor con soporte para Python/Node.js y base de datos relacional.
    - La funcionalidad de predicciones depende de disponer de datos de al menos tres eventos previos.

3. Requerimientos funcionales

- Módulo de gestión de inventario
    - Registrar productos en inventario con sus atributos básicos (nombre, categoría, proveedor, costo).
    - Actualizar automáticamente el stock al registrar ventas o compras.
    - Consultar en tiempo real el stock disponible.
    - Generar alertas de reposición basadas en patrones históricos de consumo.
    - Calcular consumos promedio por evento.
- Módulo de presupuesto y tesorería
    - Registrar ingresos y egresos vinculados a transacciones.
    - Registrar deudas (cuentas a pagar) y mantener su estado (pendiente / saldada).
    - Consultar el presupuesto consolidado en tiempo real.
    - Registrar el pago de deudas y actualizar el estado financiero.
    - Permitir reportes básicos de ingresos, egresos y deudas.
- Módulo de eventos y venta de entradas
    - Crear y administrar eventos con datos básicos (nombre, fecha, valor entrada).
    - Generar panel de control de ventas para cada evento.
    - Registrar ventas de entradas y actualizar automáticamente el presupuesto.
    - Consultar en tiempo real el total de entradas vendidas.
    - Estimar la demanda de productos según el historial de eventos similares.
- Acceso multiusuario y control de roles
    - Permitir acceso a múltiples usuarios de manera simultánea.

    - Gestionar permisos diferenciados según roles (cajero, tesorería, abastecimiento, organización).
    - Garantizar la trazabilidad de las operaciones (registro con usuario y timestamp).
- Usabilidad y transparencia
    - Ofrecer una interfaz web accesible y fácil de usar para organizaciones sin estructura empresarial.
    - Centralizar la información para que esté disponible a todos los miembros autorizados.
    - Permitir la descentralización de tareas administrativas (no depender de una sola persona).
- Funcionalidades avanzadas (escalables)
    - Implementar módulo de análisis de datos históricos para predicciones de ventas y consumos.
    - Generar reportes estadísticos básicos para la toma de decisiones.
    - Ofrecer exportación de datos (ej. CSV, PDF) para documentación externa.

4. Requerimientos no funcionales

- **Rendimiento**
    - El sistema debe responder a operaciones de consulta o registro en un tiempo máximo de 2 segundos bajo una carga de hasta 100 usuarios concurrentes.
    - El tiempo de carga inicial de la interfaz no debe superar 5 segundos en una conexión de 10 Mbps.
- **Seguridad**
    - Las comunicaciones deben realizarse bajo protocolos HTTPS
    - Las contraseñas se pueden hashear
    - Implementar control de sesión y expiración automática tras 30 min de inactividad.
    - Control de permisos por rol, con auditoría de cada acción crítica.
- **Disponibilidad y fiabilidad**
    - Debe permitir copias de seguridad de la base de datos
- **Mantenibilidad y escalabilidad**
    - Código modular, documentado y versionado con Git.
- **Portabilidad**
    - Compatible con navegadores modernos (Chrome, Brave)
    - Diseño “responsive” de ser posible

5. Casos de uso

- **CU-1: Registro de venta de producto**
    - Actor: Usuario con rol de cajero u organizador.
    - Flujo principal: Selección de producto → Verificación de stock → Descuento automático → Registro de transacción → Actualización de presupuesto.
- **CU-2: Registro de compra de insumos**
    - Actor: Usuario con rol de abastecimiento o tesorería.
    - Flujo principal: Carga de datos de proveedor y producto → Actualización de stock → Registro de egreso y deuda (opcional).
- **CU-3: Pago de deuda**
    - Actor: Usuario tesorería.

    - Flujo principal: Selección de deuda → Ingreso de monto abonado → Actualización de estado a “saldada”.
- **CU-4: Venta de entradas**
    - Actor: Usuario boletería/difusión.
    - Flujo principal: Creación de evento → Panel de control de ventas → Registro de cada venta → Actualización de presupuesto y proyecciones.

6. Modelo de datos

- Poner foto de diagrama EER o mencion de la estructura de datos aca: Podria ser:
    - Tablas:
        - Usuarios
        - Productos
        - Movimientos_Stock
        - Eventos
        - Entradas
        - Ventas_Entradas
        - Ventas
        - … etc

7. Requisitos de interfaz de usuario

- Diseño responsive
- Menú principal: acceso a →
    - Tesorería
    - Inventario
    - Eventos/Entradas
    - Reportes
    - Administración de reportes
- Formularios para obtención de entradas

8. Dependencias y infraestructura

- Lenguaje backend
- Lenguaje frontend
- Base de datos
- Frameworks
- ¿Análisis de datos?
- ¿Obligación de acceso a internet?

9. Tests

- Pruebas unitarias
- Pruebas de integración
- Pruebas de seguridad
- Pruebas de usabilidad

10. Requerimientos de dominio

- Modelo de Aplicación Web
    - Pensado para Redes LAN.
    - Motor de base de datos SQLite.
    - Maquetado web HTML.
    - Lógica backend JavaScript.
    - Frontend algún framework en caso de que haga falta.