3. Diseño del Sistema

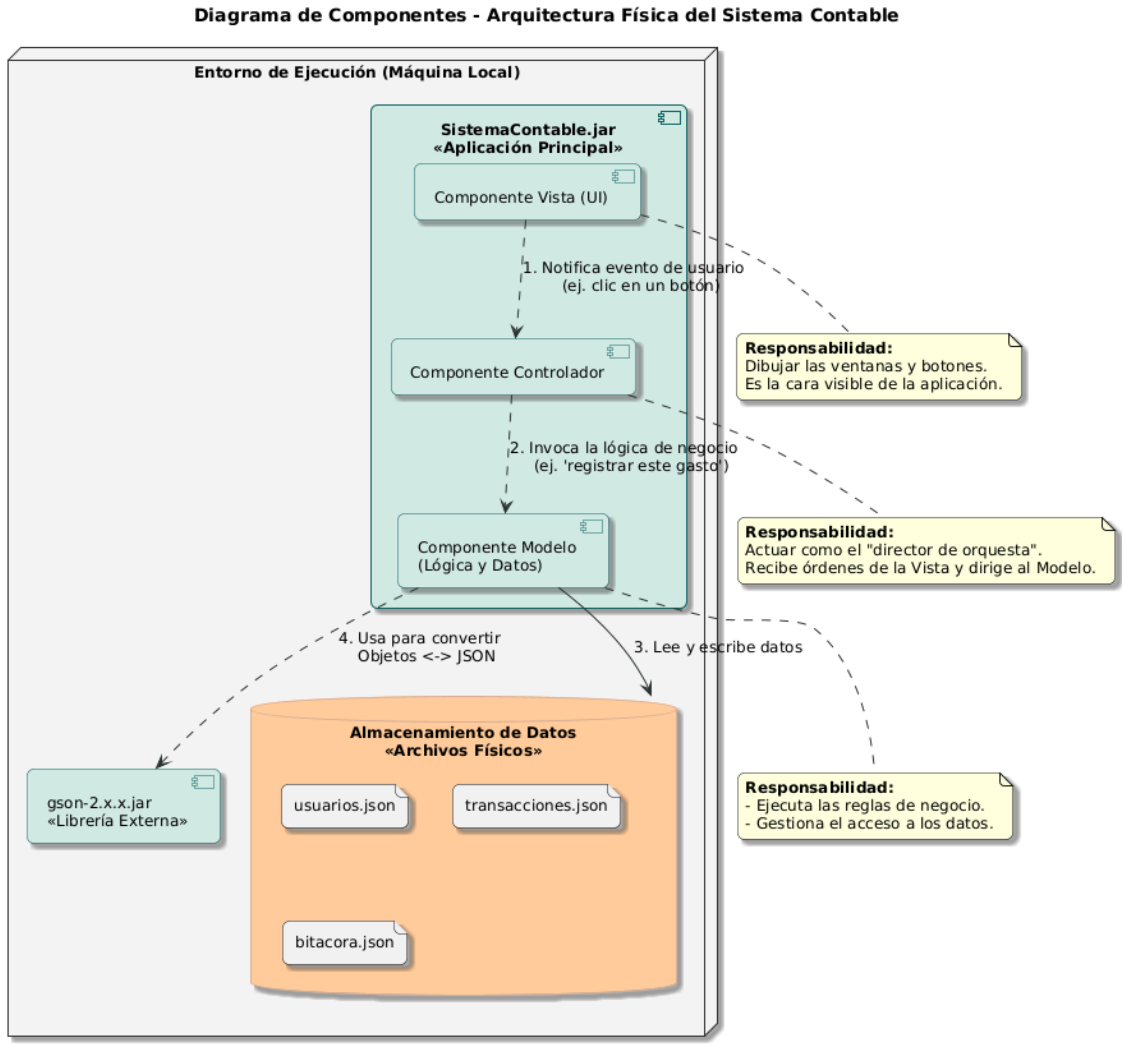
Esta sección detalla las decisiones de diseño arquitectónico, la estructura de la interfaz de usuario, el mecanismo de persistencia de datos, y los planes de prueba y despliegue para el sistema contable.

3.1. Arquitectura

El sistema se basa en una **arquitectura de capas** que implementa una separación clara de responsabilidades, inspirada en el patrón **Modelo-Vista-Controlador (MVC)**.

Diagrama de Componentes y Capas:

El siguiente diagrama ilustra las capas y los componentes principales, mostrando el flujo de control desde la interfaz de usuario hasta la capa de persistencia.



Decisiones de Arquitectura (ADR - Architecture Decision Records):

1. **ADR-001: Adopción de una Arquitectura en Capas (MVC)**

- **Decisión:** Se decidió estructurar la aplicación en tres capas principales: Vista, Controlador y Modelo.
- **Contexto:** Se necesitaba una estructura que permitiera separar la lógica de la interfaz de usuario de la lógica de negocio y de la persistencia de datos. Esto es crucial para facilitar el mantenimiento y la evolución del sistema.
- **Consecuencias:**
  - **Positivas:** Alta cohesión y bajo acoplamiento. Modificar la interfaz de usuario (Vista) no requiere cambios en la lógica de negocio (Modelo). Se pueden añadir nuevas funcionalidades de forma más ordenada.
  - **Negativas:** Para una aplicación simple, puede introducir una sobrecarga de clases y código. Sin embargo, para este proyecto, la claridad obtenida justifica la estructura.

2. **ADR-002: Persistencia de Datos mediante Archivos JSON**
   - **Decisión:** Se utilizarán archivos de texto plano en formato JSON (usuarios.json, transacciones.json, bitacora.json) para almacenar todos los datos del sistema.
   - **Contexto:** El sistema requiere una forma de persistencia simple que no dependa de un motor de base de datos externo, facilitando el despliegue y la portabilidad. Los datos debían ser legibles por humanos para facilitar la depuración y la auditoría manual.
   - **Consecuencias:**
     - **Positivas:** No hay dependencias externas (como MySQL, PostgreSQL). El despliegue se simplifica a copiar los archivos del programa. Los archivos son fácilmente editables y transportables.
     - **Negativas:** No es escalable para grandes volúmenes de datos. Las operaciones de búsqueda y filtrado son ineficientes (requieren leer todo el archivo en memoria). No hay garantías de concurrencia o transaccionalidad.
3. **ADR-003: Uso del Patrón DTO (Data Transfer Object)**
   - **Decisión:** Se crearon clases DTO (UsuarioDTO, TransaccionDTO, etc.) para mediar entre la lógica del dominio y la capa de persistencia JSON.

- **Contexto:** Las clases del dominio (Usuario, Transaccion) contienen lógica de negocio y referencias a otros objetos. Serializar estas clases directamente a JSON puede ser complejo y acoplaría el modelo de negocio al formato de persistencia.
- **Consecuencias:**
  - **Positivas:** Desacopla el modelo de dominio de la capa de persistencia. Los DTO son clases "planas" y simples, optimizadas para la serialización con la librería GSON. Permite que el formato JSON sea diferente a la estructura interna de las clases del dominio si fuera necesario.
  - **Negativas:** Requiere mantener dos conjuntos de clases (entidades y DTOs) y la lógica de conversión entre ellas, lo que añade una pequeña cantidad de código repetitivo.

3.2. Diseño de Interfaz

La interfaz de usuario está desarrollada con **Java Swing** y se compone de dos ventanas principales:

1. **Ventana de Inicio de Sesión (InterfazLoginMejorada):**
   - **Diseño:** Presenta un diseño limpio y centrado. En la parte superior se muestra el logo y título del sistema.
   - **Componentes:**
     - Campos de texto para Usuario y Contraseña.
     - Botones de Ingresar y Cancelar.
     - Un panel de registro de nuevos usuarios, inicialmente oculto, que se expande al hacer clic en "Registrar nuevo usuario". Este panel contiene campos para el nuevo usuario, contraseña, confirmación y rol.
   - **Usabilidad:** Proporciona retroalimentación inmediata a través de cuadros de diálogo para credenciales incorrectas o campos vacíos.
2. **Ventana Principal (InterfazPrincipal):**
   - **Diseño:** Utiliza un BorderLayout con una barra de menú superior, un formulario de entrada a la izquierda, una tabla de visualización en el centro y un panel de estado en la parte inferior.

- **Componentes:**
    - **Barra de Menú:** Contiene todas las opciones del sistema, agrupadas en menús como Archivo, Transacciones, Administración y Ayuda. Las opciones están habilitadas o deshabilitadas según el rol del usuario.
    - **Formulario de Registro:** Permite al usuario ingresar los datos de una nueva transacción (factura o gasto).
    - **Tabla de Transacciones:** Muestra todas las transacciones registradas con sus detalles (ID, Fecha, Tipo, Monto, Estado).
    - **Panel de Estado:** Muestra información útil como el nombre del usuario autenticado, el número total de transacciones y la suma de los montos.

3.3. Diseño de Persistencia

La persistencia se basa en tres archivos JSON principales, gestionados por las clases DAO (Gestor...JSON).

- usuarios.json:
    - **Estructura:** Un array de objetos UsuarioDTO.
    - **Campos Clave:** idUsuario (int, PK), nombreUsuario (String, único), contrasena (String, texto plano), rol (String).
    - **Acceso:** Para validar un usuario, se carga todo el array en memoria y se itera buscando una coincidencia de nombreUsuario y contrasena. Para agregar un usuario, se verifica primero que el nombreUsuario no exista.
- **transacciones.json**:
    - **Estructura:** Un array de objetos TransaccionDTO.
    - **Campos Clave:** idTransaccion (int, PK), tipo (String, "factura" o "gasto"), fecha (String), monto (double), estado (String), idUsuarioRegistro (int, FK a usuarios.json).
    - **Polimorfismo:** Los campos específicos de Factura (subtotal, iva) y Gasto (deducible, ivaCompra) se incluyen en el mismo objeto. El campo tipo actúa como discriminador para saber qué campos interpretar.

- **bitacora.json**:
    - **Estructura:** Un array de objetos BitacoraDTO.
    - **Campos Clave:** idRegistro (int, PK), idUsuario (int, FK a usuarios.json), fechaHora (String, formato ISO), accion (String), descripcion (String).
    - **Propósito:** Funciona como un log de auditoría de solo escritura (append-only). Cada vez que se realiza una acción importante, se añade un nuevo registro al final del archivo.

3.4. Plan de Pruebas

Se define un plan de pruebas multi-nivel para asegurar la calidad y robustez del sistema.

- **Tipos de Pruebas:**
    1. **Pruebas Unitarias:** Se enfocarán en validar los métodos individuales de las clases del Modelo.
        - **Ejemplos:** Factura.calcularIVA(), Usuario.autenticar(), validaciones de entrada en los DTO.
    2. **Pruebas de Integración:** Verificarán la correcta colaboración entre las capas.
        - **Ejemplos:** Probar que al llamar a ControladorTransaccion.registrarFactura(), el GestorTransaccionesJSON escribe correctamente en el archivo transacciones.json.
    3. **Pruebas Funcionales (End-to-End):** Simularán las interacciones del usuario desde la GUI.
        - **Ejemplos:** Un script de prueba que abre la aplicación, introduce credenciales, navega al formulario de gastos, registra un nuevo gasto y verifica que aparece en la tabla.
    4. **Pruebas de Aceptación (UAT):** Se basarán en los criterios de aceptación definidos en la fase de análisis.
        - **Ejemplos:** Ejecutar manualmente los escenarios "Dado-Cuando-Entonces" para validar que el sistema cumple con los requisitos del negocio.

- **Datos de Prueba:**
    - **Usuarios:**
        - admin / admin123 (Rol: Jefatura Financiera)
        - asistente / asistente123 (Rol: Asistente Contable)
        - Usuario inválido: test / test
    - **Transacciones:**
        - Montos válidos: 100.50, 25
        - Montos inválidos: -50, 0, abc, $1,200.00 (para probar la limpieza de formato)
        - Descripciones largas y cortas.

3.5. Plan de Configuración y Despliegue

El despliegue del sistema está diseñado para ser lo más simple posible.

- **Entorno de Ejecución:**
    - **Prerrequisito:** Java Runtime Environment (JRE) versión 8 o superior.
    - **Sistema Operativo:** Cualquiera compatible con Java (Windows, macOS, Linux).
- **Proceso de Despliegue Local:**

1. **Empaquetado:** Compilar todo el código fuente y empaquetarlo en un único archivo ejecutable **.jar** (por ejemplo, SistemaContable.jar). Este JAR debe incluir todas las dependencias, como la librería GSON.

2. **Distribución:** Crear una carpeta de distribución (ej. SistemaContable/) que contenga:
    - SistemaContable.jar
    - Los archivos de datos iniciales (si no se generan automáticamente): usuarios.json, transacciones.json, bitacora.json.

3. **Ejecución:** El usuario final ejecuta la aplicación mediante el comando java -jar SistemaContable.jar o haciendo doble clic en el archivo JAR (si el sistema operativo está configurado para ello).

- **Gestión de la Configuración:**

- La aplicación está diseñada para buscar los archivos .json en el mismo directorio desde el que se ejecuta. No requiere rutas absolutas ni variables de entorno, lo que la hace altamente portable.
- Si los archivos JSON no existen en el primer arranque, el sistema los creará con datos por defecto (ej. el usuario admin).