UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

INGENIERIA DE SOFTWARE

ASIGNACIÓN:

Proyecto Final - Documentación UML - Sistema Financiero

UNIDAD:

Proyecto Semestral - Sistema Financiero

FACILITADORA:

IRINA FONG

URL REPOSITORIO:

https://github.com/diegojaen18/sistema-financiero/

ESTUDIANTES:

CARLOS DELGADO

EDWIN ZHONG

DIEGO JAEN

MAURICIO PARRA

GRUPO:

1SF132

Documentación UML

Modelo Dinámico

Diagrama de Casos de Uso:

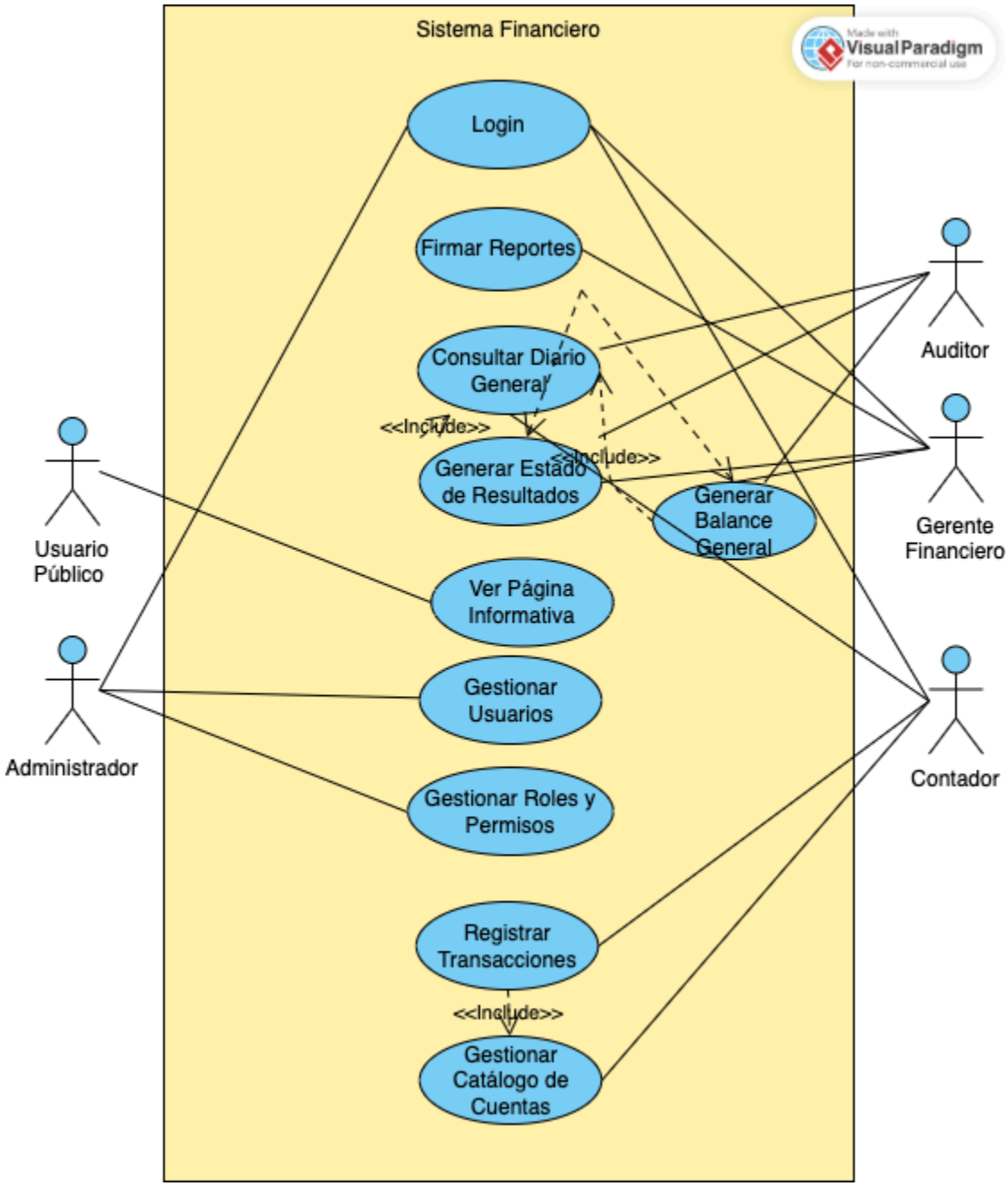


Diagrama de Actividad - Registro de Transacción:

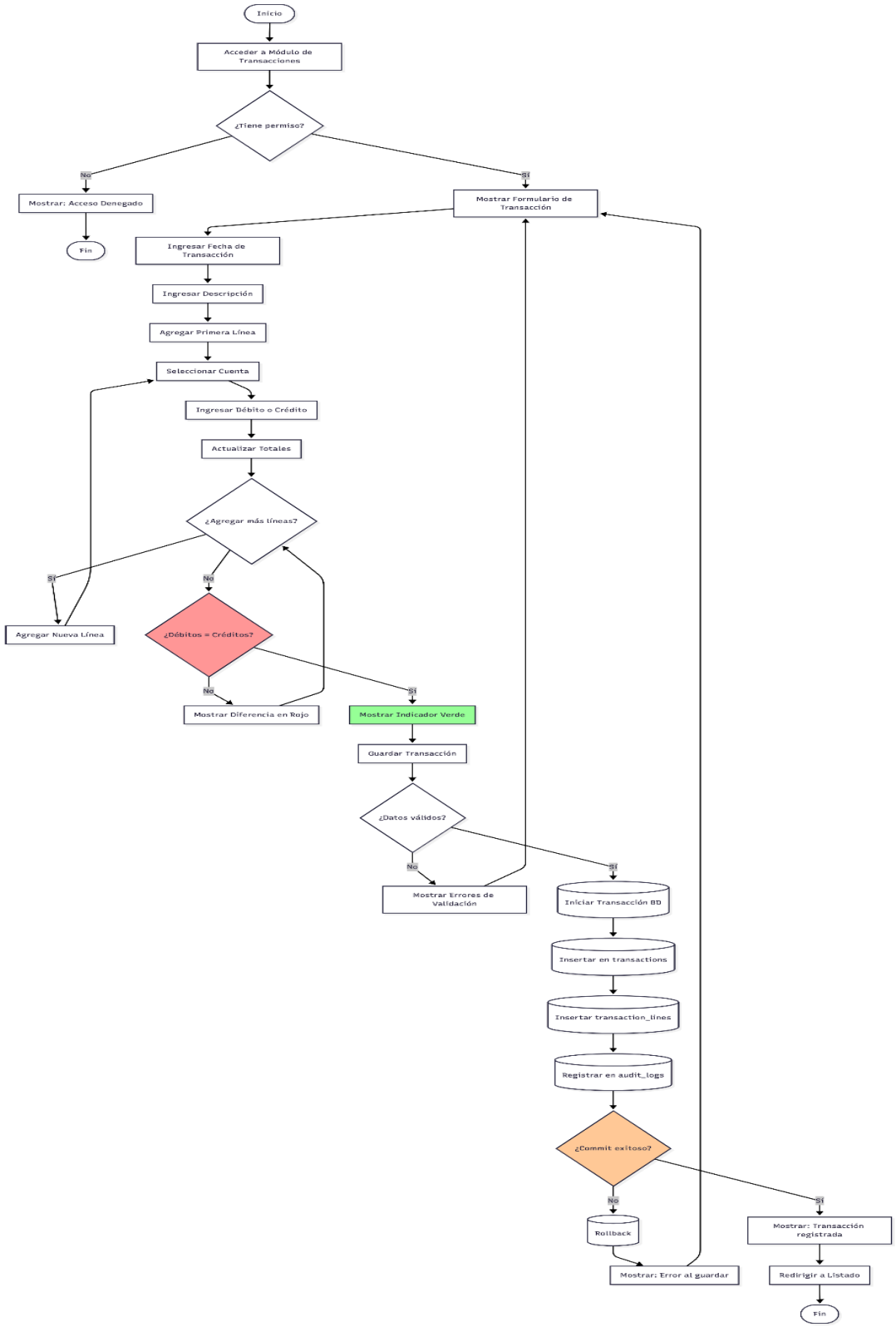


Diagrama de Secuencia - Registro de Transacción:

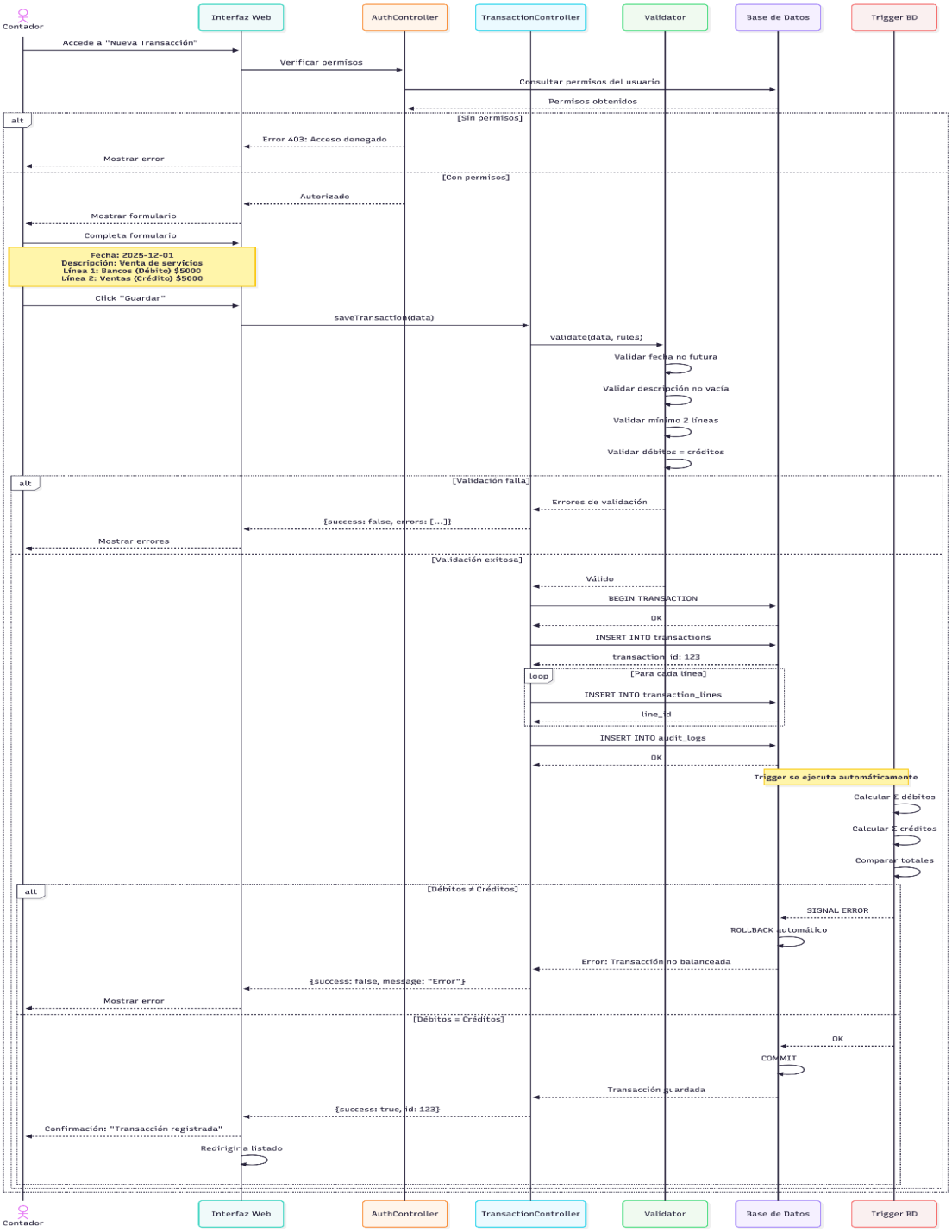


Diagrama de Secuencia: Login

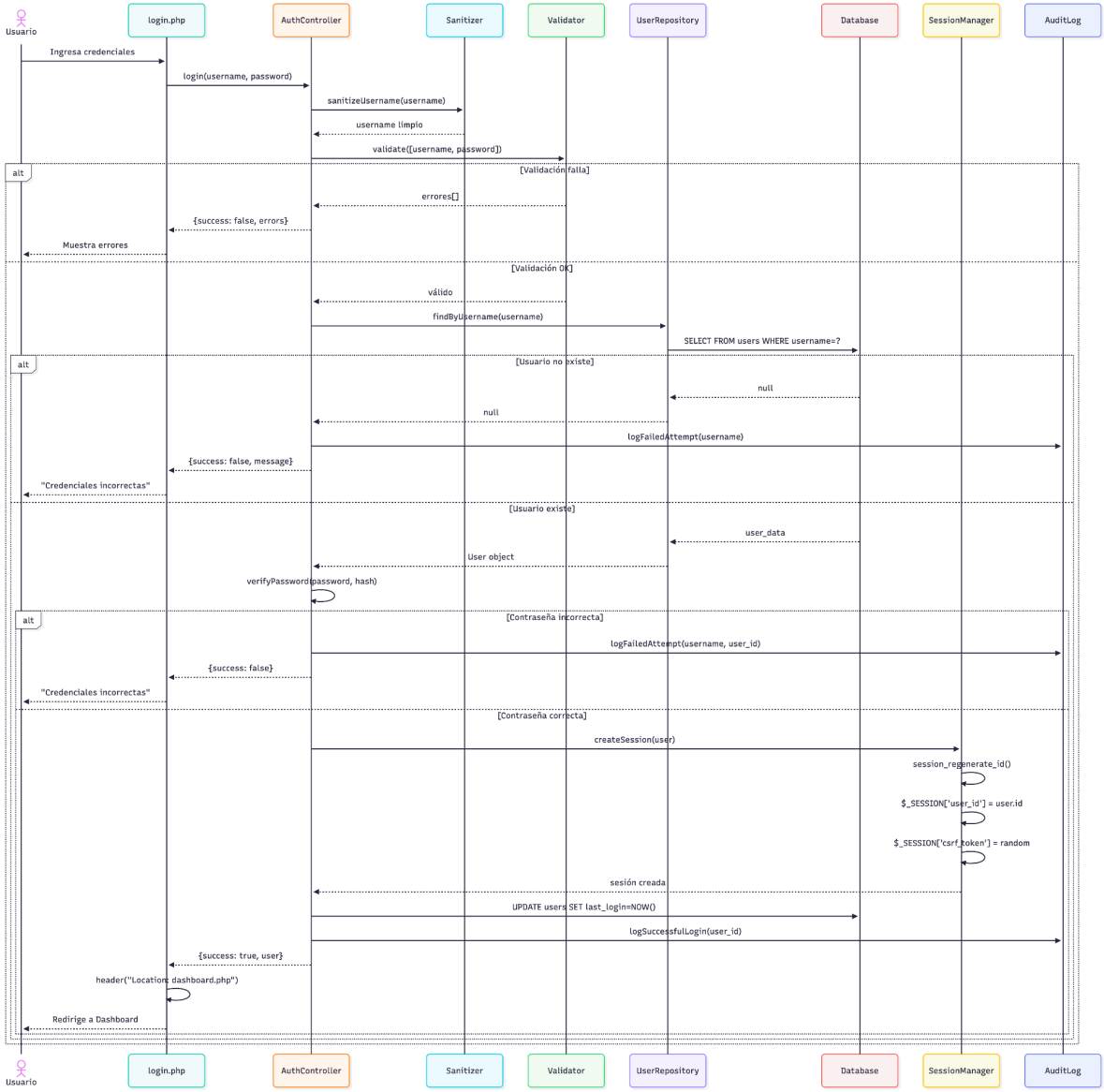


Diagrama de Estados: Transacción:

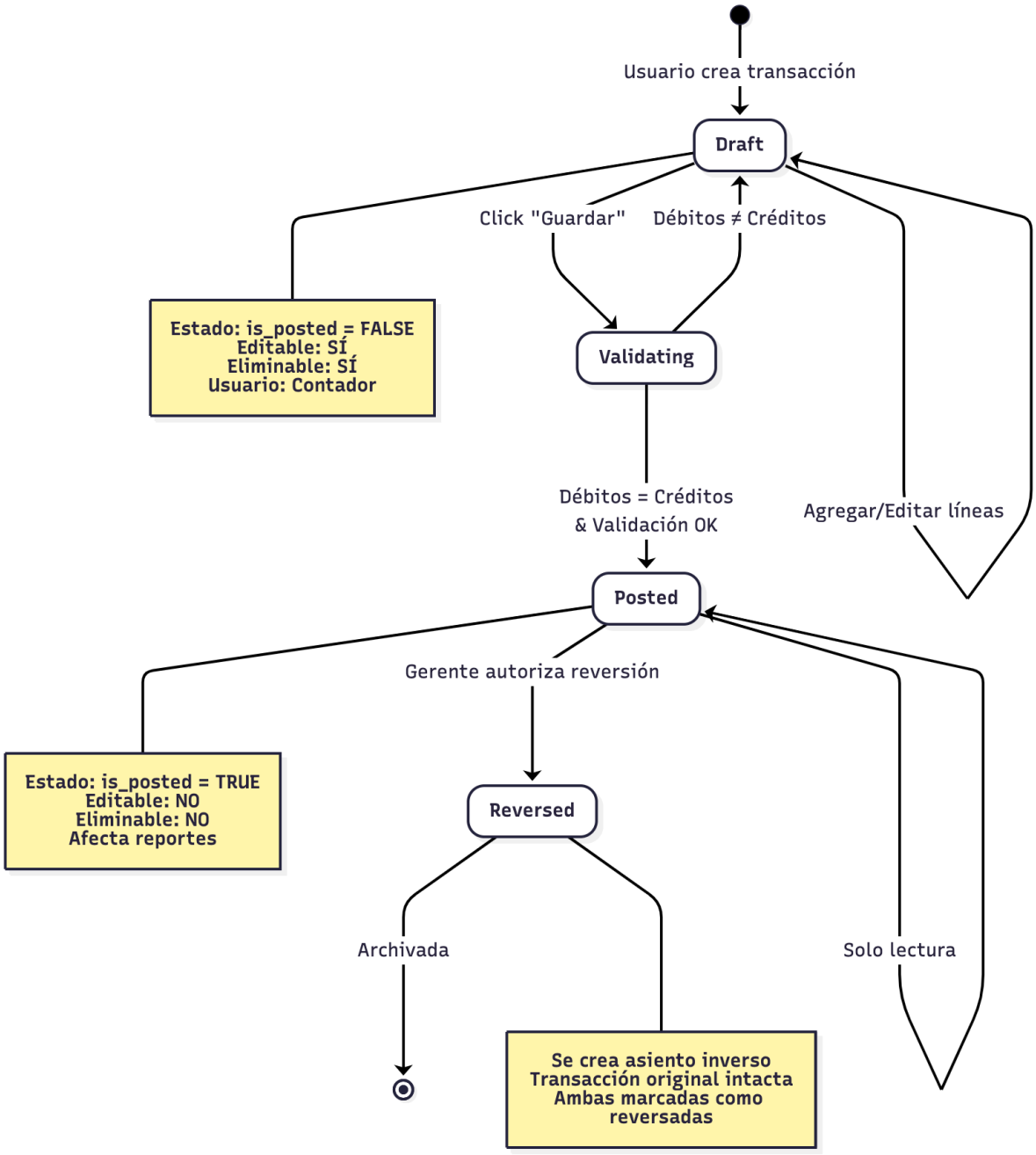
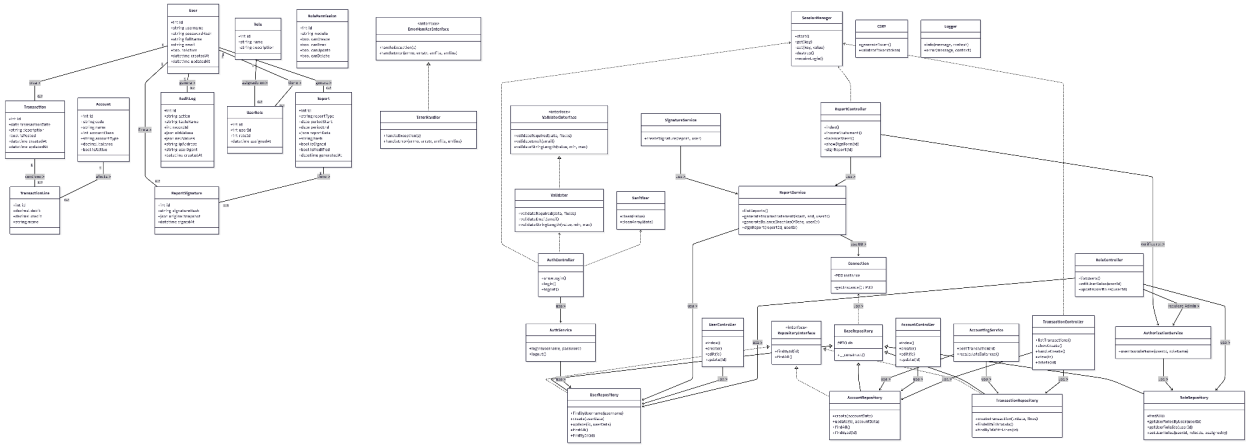

Usuario crea transacción
Draft
Click "Guardar"
Débitos ≠ Créditos
Estado: is_posted = FALSE
Editable: SÍ
Eliminable: SÍ
Usuario: Contador
Validating
Débitos = Créditos
& Validación OK
Agregar/Editar líneas
Posted
Gerente autoriza reversión
Estado: is_posted = TRUE
Editable: NO
Eliminable: NO
Afecta reportes
Reversed
Archivada
Solo lectura
Se crea asiento inverso
Transacción original intacta
Ambas marcadas como reversadas

Diagrama de Clases:



Modelado de Funcional (Casos de Uso, Descripción Textual):

CASO DE USO 01: LOGIN

Información General

Campo	Descripción
ID	**CU-01**
Nombre	**Iniciar Sesión en el Sistema**
Actores	**Administrador, Contador, Gerente Financiero, Usuario Publico**
Tipo	**Primario, Esencial**
Prioridad	**Alta**
Complejidad	**Media**

Descripción: Permite a los usuarios autenticarse en el sistema mediante credenciales (usuario y contraseña) para acceder a las funcionalidades según su rol asignado.
Precondiciones

- El usuario debe estar registrado en el sistema
- El usuario debe tener estado activo (is_active = true)
- El servidor web debe estar en funcionamiento

Postcondiciones
Éxito:

- El usuario queda autenticado con sesión activa
- Se genera un token CSRF para la sesión
- Se actualiza la fecha del último login
- Se registra el evento en audit_logs

Fallo:

- El usuario permanece sin autenticar
- Se registra el intento fallido en audit_logs
- Se muestra mensaje de error genérico

Flujo Normal (Básico)

1. El usuario accede a la página de login
2. El sistema muestra el formulario de autenticación
3. El usuario ingresa su nombre de usuario
4. El usuario ingresa su contraseña
5. El usuario hace clic en "Ingresar al Sistema"
6. El sistema valida que los campos no estén vacíos
7. El sistema sanitiza los datos ingresados
8. El sistema busca el usuario en la base de datos
9. El sistema verifica que el usuario esté activo
10. El sistema compara la contraseña con el hash almacenado
11. El sistema crea la sesión del usuario
12. El sistema regenera el ID de sesión (prevención de session fixation)
13. El sistema actualiza la fecha de último login
14. El sistema registra el login exitoso en auditoría
15. El sistema redirige al dashboard correspondiente al rol del usuario

Flujos Alternativos

FA-01: Usuario no existe

- En el paso 8, si el usuario no existe en la base de datos:
    1. El sistema registra el intento fallido
    2. El sistema muestra: "Usuario o contraseña incorrectos"
    3. El sistema retorna al formulario de login

FA-02: Contraseña incorrecta

- En el paso 10, si la contraseña no coincide:
    1. El sistema registra el intento fallido con user_id
    2. El sistema muestra: "Usuario o contraseña incorrectos"
    3. El sistema retorna al formulario de login

FA-03: Usuario inactivo

- En el paso 9, si el usuario está desactivado:
    1. El sistema muestra: "Usuario desactivado. Contacte al administrador"
    2. El sistema retorna al formulario de login

FA-04: Token CSRF inválido

- En el paso 6, si el token CSRF no coincide:

1. El sistema muestra: "Token de seguridad inválido. Recargue la página"
2. El sistema retorna al formulario de login

Flujos de Excepción

FE-01: Error de conexión a base de datos

- En cualquier paso que requiera acceso a BD:
    1. El sistema captura la excepción PDOException
    2. El sistema registra el error en logs del sistema
    3. El sistema muestra: "Error en el sistema. Intente nuevamente"
    4. El sistema retorna al formulario de login

FE-02: Sesión no puede iniciarse

- En el paso 11, si session_start() falla:
    1. El sistema registra el error
    2. El sistema muestra mensaje de error técnico
    3. El sistema sugiere limpiar cookies del navegador

Requisitos Especiales

- Seguridad: Contraseñas deben estar hasheadas con bcrypt
- Auditoría: Todos los intentos (exitosos y fallidos) deben registrarse
- Timeout: La sesión debe expirar después de 30 minutos de inactividad
- Protección: Debe implementar protección contra fuerza bruta
- HTTPS: En producción debe usar conexión segura

Reglas de Negocio

- RN-01: Las contraseñas nunca deben almacenarse en texto plano
- RN-02: Los mensajes de error no deben revelar si el usuario existe
- RN-03: Después de 3 intentos fallidos, debe aplicarse captcha
- RN-04: La sesión debe regenerarse después de autenticación exitosa

CASO DE USO 02: GESTIONAR USUARIOS

Información General

Campo	Descripción
ID	**CU-02**
Nombre	**Gestionar Usuarios Administrativos**
Actores	**Administrador**
Tipo	**Primario, Esencial**
Prioridad	**Alta**
Complejidad	**Alta**

Descripción: Permite al administrador realizar operaciones CRUD (Crear, Leer, Actualizar, Desactivar) sobre los usuarios del sistema, incluyendo la asignación de roles.

Precondiciones

- El actor debe estar autenticado
- El actor debe tener rol de Administrador
- El actor debe tener permiso de lectura/escritura en módulo "users"

Postcondiciones

Éxito:

- El usuario es creado/actualizado/desactivado según la operación
- Se registra la operación en audit_logs
- Se actualiza el campo updated_at con timestamp actual
- El administrador recibe confirmación visual

Fallo:

- No se realizan cambios en la base de datos
- Se muestra mensaje de error específico
- Se registra el intento fallido en logs

Flujo Normal - Crear Usuario

1. El administrador accede al módulo de usuarios
2. El sistema muestra la lista de usuarios existentes
3. El administrador selecciona "Crear Nuevo Usuario"
4. El sistema muestra el formulario de registro con campos:
    - Username (único)
    - Nombre completo
    - Email (único)
    - Contraseña
    - Confirmar contraseña
    - Roles disponibles (checkboxes)
5. El administrador completa el formulario
6. El administrador hace clic en "Guardar Usuario"
7. El sistema valida todos los campos según reglas establecidas
8. El sistema verifica que username y email sean únicos
9. El sistema sanitiza todos los inputs
10. El sistema hashea la contraseña con bcrypt
11. El sistema inicia una transacción de base de datos
12. El sistema inserta el registro en tabla users
13. El sistema inserta los roles en tabla user_roles
14. El sistema registra la operación en audit_logs
15. El sistema confirma la transacción (commit)
16. El sistema muestra: "Usuario creado exitosamente"
17. El sistema redirige a la lista de usuarios

Flujo Normal - Listar/Consultar Usuarios

1. El administrador accede al módulo de usuarios
2. El sistema consulta todos los usuarios activos e inactivos
3. El sistema muestra tabla con columnas:
    - ID
    - Username
    - Nombre completo
    - Email
    - Roles asignados
    - Estado (Activo/Inactivo)
    - Fecha de creación
    - Acciones (Ver, Editar, Desactivar)
4. El sistema pagina los resultados (20 por página)
5. El administrador puede filtrar por estado o buscar por nombre

Flujo Normal - Actualizar Usuario

1. El administrador selecciona "Editar" en un usuario específico
2. El sistema carga el formulario con datos actuales
3. El administrador modifica los campos deseados
4. El administrador hace clic en "Actualizar Usuario"
5. El sistema valida los cambios
6. El sistema verifica unicidad si cambió username o email
7. El sistema actualiza el registro en base de datos
8. El sistema registra los cambios en audit_logs (old_values, new_values)
9. El sistema muestra: "Usuario actualizado exitosamente"
10. El sistema redirige a la lista de usuarios

Flujo Normal - Desactivar Usuario (Soft Delete)

1. El administrador selecciona "Desactivar" en un usuario
2. El sistema verifica si el usuario tiene transacciones registradas
3. El sistema muestra confirmación: "¿Desea desactivar este usuario?"
4. El administrador confirma la acción
5. El sistema actualiza is_active = false
6. El sistema NO elimina el registro físicamente
7. El sistema registra la desactivación en audit_logs
8. El sistema muestra: "Usuario desactivado exitosamente"
9. El usuario ya no podrá iniciar sesión

Flujos Alternativos

FA-01: Username duplicado

- En paso 8 de creación o paso 6 de actualización:
    1. El sistema muestra: "El nombre de usuario ya existe"
    2. El sistema mantiene el formulario con los datos ingresados
    3. El sistema resalta el campo username en rojo

FA-02: Email duplicado

- En paso 8 de creación o paso 6 de actualización:
    1. El sistema muestra: "El email ya está registrado"
    2. El sistema mantiene el formulario con los datos ingresados

FA-03: Contraseñas no coinciden

- En paso 7 de creación:
    1. El sistema muestra: "Las contraseñas no coinciden"
    2. El sistema limpia los campos de contraseña

FA-04: Usuario con actividad no puede eliminarse

- En paso 2 de desactivación:
    1. Si el usuario tiene transacciones registradas:
    2. El sistema muestra: "No se puede eliminar. Usuario tiene registros asociados"
    3. El sistema solo permite desactivación (soft delete)

Reglas de Negocio

- RN-01: Username debe tener entre 4-50 caracteres alfanuméricos
- RN-02: Email debe ser válido según RFC 5322
- RN-03: Contraseña debe tener mínimo 8 caracteres
- RN-04: No se permite eliminación física si hay registros asociados
- RN-05: El campo created_by debe registrar quién creó el usuario
- RN-06: Un usuario puede tener múltiples roles

Requisitos Especiales

- Auditoría: Debe registrarse quién creó/modificó/desactivó cada usuario
- Seguridad: Solo administradores pueden gestionar usuarios
- Integridad: No permitir huérfanos (usuarios sin rol)
- Validación: Todos los campos deben validarse en servidor

CASO DE USO 03: GESTIONAR CATÁLOGO DE CUENTAS

Información General

Campo	Descripción
ID	**CU-03**
Nombre	**Gestionar Catálogo de Cuentas Contables**
Actores	**Contador, Administrador**
Tipo	**Primario, Esencial**
Prioridad	**Alta**
Complejidad	**Alta**

Descripción: Permite crear, consultar, actualizar y activar/desactivar cuentas del catálogo contable, clasificándolas según estándares NIIF (clases 1-7).

Precondiciones

- El actor debe estar autenticado
- El actor debe tener rol de Contador o Administrador
- El actor debe tener permisos en módulo "accounts"

Postcondiciones

Éxito:

- La cuenta es creada/actualizada en el catálogo
- Se registra quién creó/modificó la cuenta y cuándo
- Se mantiene la integridad de la clasificación contable
- El balance inicial queda en 0.00

Flujo Normal - Crear Cuenta Contable

1. El contador accede al módulo de catálogo de cuentas
2. El sistema muestra el listado de cuentas existentes
3. El contador selecciona "Crear Nueva Cuenta"
4. El sistema muestra formulario con campos:
    - Código de cuenta (formato: X.X.XX)
    - Nombre de la cuenta
    - Clase contable (1-7)
    - Tipo de cuenta (Débito/Crédito)
    - Estado (Activa/Inactiva)
5. El contador completa los campos
6. El contador hace clic en "Guardar Cuenta"
7. El sistema valida el formato del código
8. El sistema verifica que el código sea único
9. El sistema valida que la clase esté entre 1 y 7
10. El sistema valida consistencia entre clase y tipo
11. El sistema inserta el registro con balance = 0.00
12. El sistema registra el user_id y created_at
13. El sistema registra la operación en audit_logs
14. El sistema muestra: "Cuenta creada exitosamente"
15. El sistema redirige al listado actualizado

Flujo Normal - Consultar Cuentas

1. El contador accede al módulo
2. El sistema muestra listado con columnas:
    - Código
    - Nombre
    - Clase (con descripción: Activo, Pasivo, etc.)
    - Tipo (Débito/Crédito)

        - Balance actual
        - Estado
        - Fecha de creación
        - Creado por
3. El sistema permite filtrar por:
        - Clase contable
        - Estado (Activa/Inactiva)
        - Búsqueda por código o nombre
4. El sistema ordena por código ascendente

Flujos Alternativos

FA-01: Código duplicado

- En paso 8:
    1. El sistema muestra: "El código de cuenta ya existe"
    2. El sistema sugiere el siguiente código disponible

FA-02: Formato de código inválido

- En paso 7:
    1. El sistema muestra: "El código debe tener formato X.X.XX"
    2. El sistema muestra ejemplos: 1.1.01, 2.1.05

FA-03: Inconsistencia clase-tipo

- En paso 10:
    1. Si clase 1 (Activo) tiene tipo "Crédito":
    2. El sistema muestra advertencia: "Los activos normalmente son de tipo Débito"
    3. El sistema permite continuar si el contador confirma

Reglas de Negocio

- RN-01: Clases contables:
    - 1 = Activo (naturaleza débito)
    - 2 = Pasivo (naturaleza crédito)
    - 3 = Patrimonio (naturaleza crédito)
    - 4 = Ingresos (naturaleza crédito)
    - 5 = Gastos (naturaleza débito)
    - 6 = Costos (naturaleza débito)
    - 7 = Otros Gastos (naturaleza débito)
- RN-02: El código debe ser único en todo el catálogo
- RN-03: El balance inicial siempre es 0.00
- RN-04: Debe registrarse quién creó la cuenta y cuándo
- RN-05: No se eliminan cuentas físicamente, solo se desactivan

CASO DE USO 04: REGISTRAR TRANSACCIONES

Información General

Campo	Descripción

ID	CU-04
Nombre	Registrar Transacciones en Diario General
Actores	Contador
Tipo	Primario, Esencial
Prioridad	Crítica
Complejidad	Muy Alta

Descripción: Permite registrar asientos contables aplicando el principio de partida doble, donde cada transacción debe tener débitos y créditos balanceados.

Precondiciones

- El actor debe estar autenticado como Contador
- Debe existir al menos una cuenta en el catálogo
- El actor debe tener permiso de creación en módulo "transactions"

Postcondiciones

Éxito:

- La transacción queda registrada en el Diario General
- Los débitos igualan los créditos (partida doble cumplida)
- Se registra quién creó la transacción y cuándo
- Las cuentas afectadas mantienen su balance actualizado

Flujo Normal

1. El contador accede al módulo de transacciones
2. El sistema muestra el listado de transacciones
3. El contador selecciona "Nueva Transacción"
4. El sistema muestra formulario con:
    - Fecha de transacción
    - Descripción general
    - Tabla de líneas con campos:
        - Cuenta (selector)
        - Débito (input decimal)
        - Crédito (input decimal)
        - Memo (opcional)
    - Botón "Agregar línea"
    - Totales: Σ Débitos | Σ Créditos | Diferencia
5. El contador ingresa la fecha
6. El contador ingresa la descripción
7. El contador agrega la primera línea:
    - Selecciona cuenta de débito
    - Ingresa monto en columna débito
    - Ingresa memo (opcional)

8. El sistema actualiza el total de débitos
9. El contador agrega la segunda línea:
    - Selecciona cuenta de crédito
    - Ingresa monto en columna crédito
10. El sistema actualiza el total de créditos
11. El sistema calcula la diferencia (débitos - créditos)
12. El sistema muestra indicador:
    - Verde si diferencia = 0 (balanceado)
    - Rojo si diferencia ≠ 0 (desbalanceado)
13. El contador verifica que esté balanceado
14. El contador hace clic en "Guardar Transacción"
15. El sistema valida que débitos = créditos
16. El sistema valida que haya al menos 2 líneas
17. El sistema valida todos los montos sean > 0
18. El sistema inicia transacción de base de datos
19. El sistema inserta en tabla transactions
20. El sistema inserta cada línea en transaction_lines
21. El sistema registra el created_by con user_id actual
22. El sistema confirma la transacción (commit)
23. El sistema registra la operación en audit_logs
24. El sistema muestra: "Transacción registrada exitosamente"
25. El sistema redirige al listado

Flujos Alternativos

FA-01: Transacción desbalanceada

- En paso 15:
    1. Si débitos ≠ créditos:
    2. El sistema muestra: "Error: La transacción no está balanceada"
    3. El sistema muestra la diferencia exacta
    4. El sistema NO guarda la transacción
    5. El sistema mantiene el formulario con los datos

FA-02: Línea con débito Y crédito

- En pasos 7-10:
    1. Si el usuario ingresa monto en ambas columnas:
    2. El sistema muestra: "Una línea debe tener SOLO débito O crédito, no ambos"
    3. El sistema limpia la línea

FA-03: Menos de 2 líneas

- En paso 16:
    1. Si hay solo 1 línea:
    2. El sistema muestra: "Debe tener al menos 2 líneas (partida doble)"

FA-04: Monto negativo o cero

- En paso 17:

1. El sistema muestra: "Los montos deben ser mayores a cero"

Reglas de Negocio

- RN-01: PARTIDA DOBLE: Σ Débitos = Σ Créditos (SIEMPRE)
- RN-02: Una línea tiene débito O crédito, nunca ambos
- RN-03: Toda transacción debe tener mínimo 2 líneas
- RN-04: Los montos deben tener máximo 2 decimales
- RN-05: La fecha no puede ser futura
- RN-06: Se registra quién creó la transacción (created_by)
- RN-07: Una vez publicada, una transacción no se modifica (se reversa)

Requisitos Especiales

- Precisión: Usar DECIMAL(15,2) para evitar errores de redondeo
- Atomicidad: Toda la transacción debe guardarse o nada
- Auditoría: Registrar timestamp y usuario
- Validación: El trigger de BD debe validar balance antes de commit

CASO DE USO 05: GESTIONAR ROLES Y PERMISOS

Información General

Campo	Descripción
ID	**CU-05**
Nombre	**Gestionar Roles y Permisos del Sistema**
Actores	**Administrador**
Tipo	**Primario, Esencial**
Prioridad	**Alta**
Complejidad	**Alta**

Descripción: Permite definir roles del sistema y asignar permisos granulares por módulo, controlando qué usuarios pueden crear, leer, actualizar o eliminar información.

Precondiciones

- El actor debe ser Administrador
- El sistema debe tener al menos un rol definido

Postcondiciones

Éxito:

- Los roles quedan configurados con sus permisos
- Los usuarios con ese rol adquieren los permisos inmediatamente
- El sistema valida permisos en cada acción

Flujo Normal - Asignar Rol a Usuario

1. El administrador accede a gestión de usuarios
2. El administrador selecciona "Editar" en un usuario
3. El sistema muestra formulario con sección "Roles"
4. El sistema muestra checkboxes con roles disponibles:
    - Administrador
    - Contador
    - Gerente Financiero
    - Auditor
5. El administrador marca los roles deseados
6. El administrador hace clic en "Guardar"
7. El sistema elimina roles anteriores del usuario
8. El sistema inserta los nuevos roles en user_roles
9. El sistema registra quién asignó los roles
10. El sistema muestra: "Roles actualizados exitosamente"

Flujo Normal - Configurar Permisos de Rol

1. El administrador accede a módulo de roles
2. El sistema muestra lista de roles
3. El administrador selecciona "Configurar Permisos" en un rol
4. El sistema muestra tabla con módulos y permisos:
    - Usuarios: [x] Crear [ ] Leer [x] Actualizar [ ] Eliminar
    - Cuentas: [x] Crear [x] Leer [x] Actualizar [ ] Eliminar
    - Transacciones: [x] Crear [x] Leer [ ] Actualizar [ ] Eliminar
    - Reportes: [ ] Crear [x] Leer [ ] Actualizar [ ] Eliminar
    - Firmas: [ ] Crear [x] Leer [ ] Actualizar [ ] Eliminar
5. El administrador marca/desmarca permisos
6. El administrador guarda los cambios
7. El sistema actualiza tabla role_permissions
8. Los cambios aplican inmediatamente a todos los usuarios con ese rol

Reglas de Negocio

- RN-01: Un usuario puede tener múltiples roles
- RN-02: Los permisos son acumulativos (OR lógico)
- RN-03: Los permisos se verifican en cada acción del controlador
- RN-04: El rol Administrador tiene todos los permisos por defecto
- RN-05: El rol Gerente Financiero es el único que puede firmar reportes

CASO DE USO 06: GENERAR ESTADO DE RESULTADOS

Información General

Campo	Descripción
ID	**CU-06**
Nombre	**Generar Estado de Resultados (Income Statement)**
Actores	**Contador, Gerente Financiero**
Tipo	**Primario, Esencial**
Prioridad	**Crítica**

Complejidad	Alta

Descripción: Genera un reporte que resume los ingresos y gastos para calcular la utilidad neta durante un período específico.

Precondiciones

- Deben existir transacciones publicadas en el período
- El usuario debe tener permiso de lectura en "reports"

Postcondiciones

Éxito:

- El reporte se genera con datos precisos
- Se calcula correctamente la utilidad neta
- El reporte queda guardado en la base de datos

Flujo Normal

1. El usuario accede al módulo de reportes
2. El usuario selecciona "Estado de Resultados"
3. El sistema muestra formulario con:
    - Fecha inicio
    - Fecha fin
4. El usuario ingresa el período (ej: 2025-01-01 a 2025-12-31)
5. El usuario hace clic en "Generar Reporte"
6. El sistema valida que fecha inicio ≤ fecha fin
7. El sistema consulta todas las cuentas clase 4 (Ingresos)
8. El sistema suma los balances de ingresos
9. El sistema consulta todas las cuentas clase 6 (Costos)
10. El sistema suma los balances de costos
11. El sistema calcula: Utilidad Bruta = Ingresos - Costos
12. El sistema consulta todas las cuentas clase 5 (Gastos)
13. El sistema suma los balances de gastos
14. El sistema consulta todas las cuentas clase 7 (Otros Gastos)
15. El sistema suma otros gastos
16. El sistema calcula: Utilidad Neta = Utilidad Bruta - Gastos - Otros Gastos
17. El sistema construye el reporte con estructura: INGRESOS 4.1.01 Ventas: $10,000.00Total Ingresos: $10,000.00COSTOS 6.1.01 Costo de Ventas: $4,000.00Total Costos: $4,000.00UTILIDAD BRUTA: $6,000.00GASTOS OPERATIVOS 5.1.01 Sueldos: $2,000.00 5.2.01 Alquiler: $500.00Total Gastos: $2,500.00UTILIDAD NETA: $3,500.00
18.
19. El sistema guarda el reporte en tabla reports
20. El sistema genera hash SHA-256 del contenido
21. El sistema muestra el reporte en pantalla

Reglas de Negocio

- RN-01: Solo se incluyen transacciones con is_posted = true
- RN-02: Solo se usan cuentas de clases 4, 5, 6, 7
- RN-03: Los ingresos aumentan con crédito
- RN-04: Los gastos y costos aumentan con débito

CASO DE USO 07: GENERAR BALANCE GENERAL

Información General

Campo	Descripción
ID	**CU-07**
Nombre	**Generar Balance General (Balance Sheet)**
Actores	**Contador, Gerente Financiero**
Tipo	**Primario, Esencial**
Prioridad	**Crítica**
Complejidad	**Alta**

Descripción: Genera un reporte que muestra el saldo de Activos, Pasivos y Patrimonio en un momento dado, validando la ecuación contable.

Precondiciones

- Deben existir cuentas de clases 1, 2 y 3
- El usuario debe tener permiso de lectura en "reports"

Postcondiciones

Éxito:

- El reporte cumple: Activo = Pasivo + Patrimonio
- El reporte queda guardado en la base de datos
- Se valida la integridad contable

Flujo Normal

1. El usuario accede al módulo de reportes
2. El usuario selecciona "Balance General"
3. El sistema muestra formulario con:
    - Fecha de corte (a qué fecha generar el balance)
4. El usuario ingresa la fecha (ej: 2025-12-31)
5. El usuario hace clic en "Generar Reporte"
6. El sistema consulta cuentas clase 1 (Activos) con sus balances
7. El sistema suma: Total Activos
8. El sistema consulta cuentas clase 2 (Pasivos) con sus balances
9. El sistema suma: Total Pasivos
10. El sistema consulta cuentas clase 3 (Patrimonio) con sus balances
11. El sistema suma: Total Patrimonio
12. El sistema calcula: Pasivo + Patrimonio

13. El sistema valida: Activo = Pasivo + Patrimonio
14. Si la ecuación no se cumple:
    - El sistema muestra ERROR CRÍTICO
    - El sistema NO guarda el reporte
    - El sistema sugiere revisar las transacciones
15. Si la ecuación se cumple:
    - El sistema construye el reporte
    - El sistema guarda en base de datos
    - El sistema muestra el reporte
16. El reporte tiene estructura: ACTIVOS 1.1.01 Caja: $5,000.00 1.1.02 Bancos: $15,000.00Total Activos: $20,000.00PASIVOS 2.1.01 Cuentas por Pagar: $5,000.00Total Pasivos: $5,000.00PATRIMONIO 3.1.01 Capital Social: $15,000.00Total Patrimonio: $15,000.00TOTAL PASIVO + PATRIMONIO: $20,000.00ECUACIÓN CONTABLE: ✓ VÁLIDA
17.

Reglas de Negocio

- RN-01: CRITERIO DE ÉXITO: Activo = Pasivo + Patrimonio
- RN-02: Si la ecuación no se cumple, el reporte es inválido
- RN-03: Los activos tienen naturaleza débito
- RN-04: Los pasivos y patrimonio tienen naturaleza crédito

CASO DE USO 08: FIRMAR REPORTES

Información General

Campo	Descripción
ID	**CU-08**
Nombre	**Firmar Reporte con Firma Digital**
Actores	**Gerente Financiero**
Tipo	**Primario, Esencial**
Prioridad	**Crítica**
Complejidad	**Muy Alta**

Descripción: Permite al Gerente Financiero certificar un reporte mediante firma digital usando hash SHA-256, estableciendo un snapshot inmutable que detecta cualquier modificación posterior.

Precondiciones

- El reporte debe estar generado y guardado
- El reporte NO debe estar firmado previamente
- El usuario debe tener rol "Gerente Financiero"
- El usuario debe tener permiso en módulo "signatures"

Postcondiciones

Éxito:

- El reporte queda firmado digitalmente
- Se genera un hash SHA-256 del contenido
- Se guarda snapshot inmutable en report_signatures
- El campo is_signed se actualiza a TRUE
- Cualquier modificación posterior será detectada

Flujo Normal

1. El Gerente accede al módulo de reportes
2. El sistema muestra lista de reportes generados
3. El Gerente selecciona un reporte sin firmar
4. El sistema muestra el contenido completo del reporte
5. El sistema muestra botón "Firmar Reporte" (solo visible para Gerentes)
6. El Gerente revisa el contenido del reporte
7. El Gerente hace clic en "Firmar Reporte"
8. El sistema muestra confirmación: "¿Está seguro de firmar este reporte?"
9. El Gerente confirma la acción
10. El sistema inicia transacción de base de datos
11. El sistema captura snapshot completo del reporte:
    - ID del reporte
    - Tipo (income_statement o balance_sheet)
    - Período
    - Datos completos (JSON)
    - Fecha de generación
    - Usuario que generó
12. El sistema serializa el snapshot a JSON
13. El sistema genera hash SHA-256 del JSON
14. El sistema inserta registro en report_signatures:
    - report_id
    - user_id (Gerente que firma)
    - signature_hash
    - original_snapshot
    - signed_at (timestamp actual)
15. El sistema actualiza tabla reports:
    - is_signed = TRUE
    - hash = signature_hash
16. El sistema registra la operación en audit_logs
17. El sistema confirma transacción (commit)
18. El sistema muestra: "Reporte firmado exitosamente"
19. El sistema muestra indicador visual de firma: ✓ REPORTE FIRMADOFirmado por: [Nombre del Gerente]Fecha: [2025-12-09 14:30:00]Hash: a3f5b2c8...
20.

Flujo Normal - Verificar Integridad

1. Cualquier usuario abre un reporte firmado
2. El sistema obtiene el hash original de report_signatures
3. El sistema recalcula el hash del contenido actual
4. El sistema compara ambos hashes

5. Si coinciden:
    - El sistema muestra: ✓ "Reporte íntegro y confiable"
    - Indicador verde
6. Si NO coinciden:
    - El sistema actualiza is_modified = TRUE
    - El sistema muestra: "ADVERTENCIA: Este reporte ha sido modificado"
    - Indicador rojo
    - El sistema registra violación de integridad en audit_logs

Flujos Alternativos

FA-01: Reporte ya firmado

- En paso 3:
    1. Si el reporte tiene is_signed = TRUE:
    2. El sistema NO muestra el botón "Firmar Reporte"
    3. El sistema muestra información de la firma existente

FA-02: Usuario sin permiso

- En paso 7:
    1. Si el usuario no es Gerente Financiero:
    2. El sistema muestra: "Solo Gerentes Financieros pueden firmar reportes"
    3. El sistema redirige al listado

FA-03: Error al generar hash

- En paso 13:
    1. Si hash() falla:
    2. El sistema hace rollback de la transacción
    3. El sistema muestra: "Error al firmar reporte"
    4. El sistema registra el error en logs

Reglas de Negocio

- RN-01: Solo Gerentes Financieros pueden firmar
- RN-02: Un reporte solo puede firmarse UNA vez
- RN-03: La firma es irreversible (solo administradores pueden revocarla)
- RN-04: Se usa SHA-256 para el hash
- RN-05: El snapshot debe incluir TODOS los datos del reporte
- RN-06: La verificación de integridad se realiza en CADA acceso
- RN-07: Si hay modificación, se marca is_modified = TRUE automáticamente

Requisitos Especiales

- Algoritmo: SHA-256 para hashing
- Inmutabilidad: El snapshot NO debe modificarse nunca
- Detección: La verificación debe ser automática
- Auditoría: Registrar quién firmó, cuándo y desde qué IP

CASO DE USO 09: VER PÁGINA INFORMATIVA

Información General

Campo	Descripción
ID	**CU-09**
Nombre	**Ver Página Pública Informativa**
Actores	**Usuario Público (sin autenticar)**
Tipo	**Secundario**
Prioridad	**Media**
Complejidad	**Baja**

Descripción: Muestra una página pública que explica la importancia de llevar un registro contable organizado y permite acceder al sistema de login.

Precondiciones

- Ninguna (acceso público)

Postcondiciones

- El visitante conoce el propósito del sistema
- El visitante puede acceder al login

Flujo Normal

1. El visitante accede a la URL raíz del sistema
2. El sistema muestra la página pública con:
    - Título: "Sistema Financiero de Contabilidad"
    - Sección: "¿Por qué registrar tus transacciones?"
    - Beneficios de la contabilidad organizada
    - Enlace: "Ingresar al Sistema"
3. El visitante lee la información
4. El visitante hace clic en "Ingresar al Sistema"
5. El sistema redirige a la página de login

Contenido Sugerido

¿Por qué registrar tus transacciones?
La contabilidad organizada permite:
• Tomar decisiones informadas basadas en datos reales
• Cumplir con obligaciones fiscales y legales
• Detectar fraudes y errores oportunamente
• Planificar el futuro financiero de tu empresa
• Obtener financiamiento con reportes confiables
• Medir el rendimiento del negocio
Nuestro sistema te ofrece:

✓ Registro de transacciones con partida doble
✓ Generación automática de reportes financieros
✓ Control de acceso por roles
✓ Firma digital de documentos
✓ Auditoría completa de operaciones

CASO DE USO 10: Consultar Diario General

Campo	Descripción
ID	CU-05
Nombre	Consultar Diario General
Actores	Contador, Gerente Financiero, Auditor
Tipo	Primario, Esencial
Prioridad	Alta
Complejidad	Media

Descripción: Permite a los usuarios autorizados consultar el registro completo de todas las transacciones contables registradas en el sistema (Diario General), con opciones de filtrado por fecha, cuenta, usuario registrador y estado de publicación.

Precondiciones

- El actor debe estar autenticado
- El actor debe tener permiso de lectura en módulo "transactions"
- Deben existir transacciones registradas en el sistema

Postcondiciones

Éxito:

- El usuario visualiza el listado de transacciones solicitado
- Se muestran todos los detalles de las transacciones
- Los filtros aplicados quedan guardados en la sesión
- Se puede exportar la consulta (opcional)

Fallo:

- Se muestra mensaje si no hay transacciones en el período seleccionado
- Se muestra error si los filtros son inválidos

Flujo Normal (Básico)

1. El usuario accede al módulo "Diario General"
2. El sistema muestra la interfaz de consulta con:
    - Filtros disponibles:
        - Rango de fechas (desde - hasta)
        - Cuenta contable (selector múltiple)
        - Usuario que registró (selector)
        - Estado (Todas / Publicadas / Borradores)
        - Búsqueda por descripción
    - Botón "Consultar"
    - Tabla de resultados (inicialmente vacía o con últimos 30 días)
3. El usuario opcionalmente configura filtros:
    - Selecciona rango de fechas (ej: 01/01/2025 - 31/12/2025)
    - Selecciona cuentas específicas (ej: Bancos, Ventas)
    - Selecciona estado: "Solo Publicadas"
4. El usuario hace clic en "Consultar"
5. El sistema valida los filtros:
    - Fecha desde ≤ Fecha hasta
    - Fechas en formato válido
    - Cuentas existen en el catálogo
6. El sistema construye la consulta SQL con los filtros aplicados
7. El sistema consulta la base de datos

8. El sistema muestra tabla de resultados con columnas:

- Fecha de transacción
- No. Asiento (ID)
- Descripción
- Total Débitos
- Total Créditos
- Estado (Publicada/Borrador)
- Registrado por
- Fecha de registro
- Acciones (Ver detalle, Editar si es borrador)

9. El usuario puede:

- Ver el listado paginado
- Hacer clic en "Ver detalle" para expandir líneas de la transacción
- Navegar entre páginas
- Exportar a Excel/PDF (opcional)

10. Al hacer clic en "Ver detalle" de una transacción

11. El sistema mantiene los filtros en la sesión para consultas posteriores

Flujos Alternativos

FA-01: No hay transacciones en el período

- En paso 7, si la consulta no retorna resultados:
    1. El sistema muestra mensaje: "No se encontraron transacciones con los filtros aplicados"
    2. El sistema sugiere ampliar el rango de fechas
    3. El sistema mantiene el formulario de filtros visible

FA-02: Rango de fechas inválido

- En paso 5, si fecha desde > fecha hasta:
    1. El sistema muestra: "La fecha inicial debe ser anterior a la fecha final"
    2. El sistema resalta los campos de fecha en rojo
    3. El sistema NO ejecuta la consulta

FA-03: Consulta muy amplia (más de 1000 resultados)

- En paso 7, si la consulta retornaría más de 1000 registros:
    1. El sistema muestra advertencia: "Su búsqueda retorna más de 1000 registros"
    2. El sistema sugiere: "Recomendamos filtrar por un período más específico"
    3. El sistema ofrece opciones:
        - "Ver primeros 1000 resultados"
        - "Ajustar filtros"
        - "Exportar todo a Excel" (para procesamiento externo)

FA-04: Ver solo transacciones propias

- Si el usuario es Contador (no Gerente):
    1. El sistema automáticamente filtra: WHERE created_by = [user_id]
    2. El usuario solo ve transacciones que él mismo registró
    3. Se muestra nota: "Visualizando solo sus transacciones"

FA-05: Exportar a Excel

- En paso 9, si el usuario hace clic en "Exportar":
    1. El sistema genera archivo CSV/Excel con los resultados
    2. El sistema incluye todas las columnas visibles
    3. El sistema incluye líneas detalladas de cada transacción
    4. El sistema inicia descarga del archivo
    5. Nombre del archivo: `diario_general_YYYYMMDD_HHMMSS.xlsx`

Flujos de Excepción

FE-01: Error de base de datos

- En paso 7, si la consulta SQL falla:
    1. El sistema captura la excepción PDOException
    2. El sistema registra el error en logs
    3. El sistema muestra: "Error al consultar transacciones. Intente nuevamente"
    4. El sistema NO expone detalles técnicos del error

FE-02: Timeout de consulta

- En paso 7, si la consulta tarda más de 30 segundos:
    1. El sistema cancela la consulta
    2. El sistema muestra: "La consulta está tardando demasiado"
    3. El sistema sugiere: "Intente reducir el rango de fechas o agregar más filtros"

RESUMEN DE CASOS DE USO

ID	Caso de Uso	Actor Principal	Prioridad
CU-01	**Login**	**Todos**	**Alta**
CU-02	**Gestionar Usuarios**	**Administrador**	**Alta**
CU-03	**Gestionar Catálogo de Cuentas**	**Contador**	**Alta**
CU-04	**Registrar Transacciones**	**Contador**	**Crítica**
CU-05	**Gestionar Roles y Permisos**	**Administrador**	**Alta**
CU-06	**Generar Estado de Resultados**	**Contador, Gerente**	**Crítica**
CU-07	**Generar Balance General**	**Contador, Gerente**	**Crítica**
CU-08	**Firmar Reporte Digitalmente**	**Gerente Financiero**	**Crítica**
CU-09	**Ver Página Informativa**	**Usuario Público**	**Media**
CU-10	**Consultar Diario General**	**Contador**	**Alta**

Requisitos de Instalación del Sistema (Requisitos Funcionales y No Funcionales):

1. REQUISITOS FUNCIONALES

RF-01: Autenticación y Sesiones

ID	RF-01
Descripción	El sistema debe permitir a los usuarios autenticarse mediante usuario y contraseña
Prioridad	Alta
Criterios de Aceptación	- Login exitoso con credenciales válidas - Rechazo de credenciales inválidas - Sesión se mantiene durante 30 minutos de inactividad - Logout destruye completamente la sesión

RF-02: Gestión de Usuarios

ID	RF-02
Descripción	El sistema debe permitir crear, consultar y actualizar usuarios administrativos
Prioridad	Alta
Criterios de Aceptación	- CRUD completo de usuarios - Usuarios con actividad NO se eliminan físicamente - Solo desactivación lógica (soft delete) - Auditoría de quién creó cada usuario

RF-03: Catálogo de Cuentas Contables

ID	RF-03
Descripción	El sistema debe gestionar un catálogo de cuentas clasificadas según NIIF
Prioridad	Alta
Criterios de Aceptación	- Clasificación por clases 1-7 - Código único por cuenta - Registro de quién y cuándo se creó - Identificación de naturaleza (débito/crédito)

RF-04: Registro de Transacciones

ID	RF-04
Descripción	El sistema debe registrar transacciones contables aplicando partida doble
Prioridad	Crítica
Criterios de Aceptación	- Débitos deben igualar créditos (validación obligatoria) -Mínimo 2 líneas por transacción - Registro de usuario que creó la transacción - Timestamp automático

RF-05: Sistema de Roles y Permisos

ID	RF-05
Descripción	El sistema debe controlar el acceso mediante roles y permisos granulares
Prioridad	Alta
Criterios de Aceptación	- Un usuario puede tener múltiples roles -Permisos por módulo (crear, leer, actualizar, eliminar) -Validación de permisos en cada acción -Roles: Administrador, Contador, Gerente Financiero

RF-06: Generación de Estado de Resultados

ID	RF-06
Descripción	El sistema debe generar Estado de Resultados automáticamente
Prioridad	Crítica
Criterios de Aceptación	- Cálculo: Ingresos - Costos - Gastos = Utilidad Neta - Solo usa cuentas de clases 4, 5, 6, 7 - Filtrado por período específico - Solo transacciones publicadas

RF-07: Generación de Balance General

ID	RF-07
Descripción	El sistema debe generar Balance General validando la ecuación contable
Prioridad	Crítica
Criterios de Aceptación	- Validación: Activo = Pasivo + Patrimonio -Solo usa cuentas de clases 1, 2, 3 - Si no cuadra, muestra error y NO guarda - Balance a una fecha específica

RF-08: Firma Digital de Reportes

ID	RF-08
Descripción	El sistema debe permitir firmar reportes digitalmente y detectar modificaciones
Prioridad	Crítica
Criterios de Aceptación	- Solo Gerentes Financieros pueden firmar - Generación de hash SHA-256 - Detección automática de alteraciones - Indicador visual de integridad comprometida

RF-09: Auditoría Completa

ID	RF-09
Descripción	El sistema debe registrar automáticamente todas las operaciones críticas
Prioridad	Alta
Criterios de Aceptación	- Registro de quién hizo qué y cuándo - Logs de login exitoso y fallido - Registro de creación/modificación de datos - Almacenamiento de valores antiguos y nuevos

RF-10: Página Pública Informativa

ID	RF-10
Descripción	El sistema debe tener una página pública que explique su importancia
Prioridad	Media

Criterios de Aceptación	- Accesible sin autenticación - Explica beneficios de contabilidad organizada - Enlace a login visible - Información institucional

2. REQUISITOS NO FUNCIONALES

RNF-01: Seguridad

RNF-01.1: Gestión Segura de Contraseñas

ID	RNF-01.1
Descripción	Las contraseñas deben almacenarse de forma segura
Criterios	- Hash con bcrypt (password_hash) - Nunca almacenar en texto plano - Mínimo 8 caracteres - Verificación con password_verify()

RNF-01.2: Sesiones Seguras

ID	RNF-01.2
Descripción	Las sesiones deben configurarse de forma segura
Criterios	- HttpOnly cookies - Secure flag (HTTPS en producción) - Regeneración de session_id después de login - Timeout de 30 minutos

RNF-02: Rendimiento

RNF-02.1: Tiempo de Respuesta

ID	RNF-02.1
Descripción	El sistema debe responder rápidamente a las peticiones
Criterios	- Login: < 2 segundos - Listados: < 3 segundos - Generación de reportes: < 5 segundos - Guardado de transacciones: < 2 segundos

RNF-02.2: Optimización de Consultas

ID	RNF-02.2
Descripción	Las consultas a base de datos deben estar optimizadas
Criterios	- Uso de índices en columnas frecuentes - LIMIT en consultas grandes - Evitar SELECT * cuando no sea necesario - Uso de JOINs eficientes

RNF-02.3: Paginación

ID	RNF-02.3
Descripción	Los listados grandes deben paginarse

Criterios	- Máximo 20 registros por página - Navegación entre páginas - Indicador de página actual

RNF-03: Usabilidad

RNF-03.1: Interfaz Intuitiva

ID	RNF-03.1
Descripción	La interfaz debe ser fácil de usar
Criterios	- Mensajes de error claros - Confirmaciones de acciones exitosas - Formularios con validación en tiempo real - Etiquetas descriptivas

RNF-03.2: Mensajes de Error

ID	RNF-03.2
Descripción	Los errores deben comunicarse claramente
Criterios	- Mensajes en español - Específicos pero no técnicos - Sugerencias de solución - Nunca exponer información sensible

RNF-04: Mantenibilidad

RNF-04.1: Arquitectura MVC

ID	RNF-04.1
Descripción	El código debe seguir el patrón MVC
Criterios	- Separación de capas (Modelo, Vista, Controlador) - Service Layer para lógica compleja - Repository Pattern para acceso a datos

RNF-04.2: Código Documentado

ID	RNF-04.2
Descripción	El código debe estar bien documentado
Criterios	- PHPDoc en todas las clases y métodos - Comentarios explicativos en lógica compleja - README.md con instrucciones de instalación

RNF-04.3: Nomenclatura Consistente

ID	RNF-04.3
Descripción	Los nombres deben seguir convenciones
Criterios	- Clases: PascalCase - Métodos y variables: camelCase - Constantes: SNAKE_CASE en mayúsculas - Nombres descriptivos en inglés

RNF-05: Confiabilidad

RNF-05.1: Precisión Decimal

ID	RNF-05.1
Descripción	Las operaciones monetarias deben ser precisas
Criterios	- Uso de DECIMAL(15,2) en BD - number_format() en PHP

	- Nunca usar tipo float para dinero - Redondeo a 2 decimales

RNF-05.2: Transaccionalidad

ID	RNF-05.2
Descripción	Las operaciones críticas deben ser atómicas
Criterios	- BEGIN TRANSACTION para operaciones multi-paso - COMMIT solo si todo es exitoso - ROLLBACK en caso de error - Uso de try-catch

RNF-05.3: Validación de Integridad

ID	RNF-05.3
Descripción	El sistema debe validar la integridad contable
Criterios	- Trigger que valida débitos = créditos - Validación de ecuación contable en Balance - Foreign keys para integridad referencial

RNF-06: Escalabilidad

RNF-06.1: Estructura Modular

ID	RNF-06.1
Descripción	El sistema debe permitir agregar funcionalidades fácilmente
Criterios	- Módulos independientes - Interfaces para contratos - Bajo acoplamiento - Alta cohesión

RNF-07: Portabilidad

RNF-07.1: Compatibilidad de Servidor

ID	RNF-07.1
Descripción	El sistema debe funcionar en diferentes entornos
Criterios	- Compatible con PHP 7.4+ - Compatible con MySQL 5.7+ - Funciona en MAMP, XAMPP, WAMP - Rutas relativas (no absolutas)

RNF-08: Disponibilidad

RNF-08.1: Manejo de Errores

ID	RNF-08.1
Descripción	El sistema debe manejar errores gracefully
Criterios	- Try-catch en operaciones críticas -Logging de errores -Mensajes amigables al usuario -No exponer stack traces

3. REQUISITOS DE INSTALACIÓN

REQ-INST-01: Requisitos de Hardware

Servidor de Desarrollo (Mínimo)

- **Procesador**: Intel Core i5 o equivalente
- **RAM**: 4 GB mínimo (8 GB recomendado)
- **Almacenamiento**: 2 GB libres mínimo
- **Red**: Conexión a Internet para descargar dependencias

Recomendado para MacBook Air 2017

- **RAM disponible**: Mínimo 4 GB libres
- **Almacenamiento SSD**: 5 GB libres (incluye MAMP + BD + proyecto)
- **Sistema operativo**: macOS High Sierra o superior

REQ-INST-02: Requisitos de Software

Software Obligatorio
MAMP (Macintosh, Apache, MySQL, PHP)

- **Versión**: MAMP 6.x o superior
- **Componentes**:
    - Apache 2.4+
    - MySQL 8.0+ o MariaDB 10.x
    - PHP 7.4+ (recomendado: PHP 8.0+)
    - phpMyAdmin 5.x

Navegador Web Moderno

- Google Chrome 90+
- Safari 14+
- Firefox 88+
- Debe soportar JavaScript

Editor de Código (Opcional)

- Visual Studio Code (recomendado)
- Sublime Text
- PHPStorm

Control de Versiones

- Git 2.30+
- Cuenta de GitHub

REQ-INST-03: Configuración de PHP

Configuración Requerida en php.ini:
memory_limit = 256M
max_execution_time = 60
upload_max_filesize = 10M
post_max_size = 10M
display_errors = On (solo desarrollo)
error_reporting = E_ALL (solo desarrollo)
date.timezone = America/Panama

```
# Extensiones requeridas
extension=pdo_mysql
extension=mysqli
extension=mbstring
extension=openssl
extension=json
```

REQ-INST-04: Configuración de MySQL

Configuración Requerida en my.cnf:

```
[mysqld]
max_connections = 50
innodb_buffer_pool_size = 512M
character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci
[client]
default-character-set = utf8mb4
```

REQ-INST-05: Pasos de Instalación

1. Instalar MAMP

1. Descargar MAMP desde https://www.mamp.info/
2. Instalar en /Applications/MAMP/
3. Iniciar MAMP
4. Verificar que Apache y MySQL estén corriendo (luz verde)

2. Configurar Base de Datos

1. Abrir phpMyAdmin: http://localhost:8888/phpMyAdmin/
2. Crear base de datos "sistema_financiero"
3. Importar script SQL: docs/database/sistema_financiero.sql
4. Verificar creación de tablas (9 tablas + vistas)

3. Clonar Repositorio

```
cd /Applications/MAMP/htdocs/
git clone [URL_DEL_REPOSITORIO] sistema-financiero
cd sistema-financiero
```

4. Configurar Conexión

```
# Editar: config/database.php
# Ajustar credenciales según tu configuración de MAMP:
DB_HOST = 'localhost'
DB_PORT = '8889'  # Puerto por defecto de MAMP
DB_NAME = 'sistema_financiero'
DB_USER = 'root'
DB_PASS = 'root'  # Contraseña por defecto de MAMP
```

5. Configurar Permisos (macOS)

```
chmod -R 755 /Applications/MAMP/htdocs/sistema-financiero
chmod -R 777 storage/logs
```

6. Acceder al Sistema

URL: http://localhost:8888/sistema-financiero/public/
Usuario: admin
Contraseña: Admin123!

REQ-INST-06: Verificación de Instalación

Checklist de Verificación:

- [ ] MAMP muestra luces verdes en Apache y MySQL
- [ ] phpMyAdmin es accesible
- [ ] Base de datos "sistema_financiero" existe con 9 tablas
- [ ] Usuario admin puede hacer login
- [ ] Página pública es accesible sin login
- [ ] No hay errores en logs de PHP
- [ ] Catálogo de cuentas tiene registros iniciales

REQ-INST-07: Solución de Problemas Comunes

Problema: "Connection refused" al acceder a MySQL
Solución:
1. Verificar que MySQL esté corriendo en MAMP
2. Verificar el puerto en config/database.php (usualmente 8889 en MAMP)
3. Reiniciar MAMP

Problema: "Table doesn't exist"
Solución:
1. Re-importar el script SQL
2. Verificar que la base de datos seleccionada sea "sistema_financiero"
3. Verificar que todas las tablas se crearon correctamente

Problema: Página en blanco (White Screen of Death)
Solución:
1. Activar display_errors en php.ini
2. Revisar logs: /Applications/MAMP/logs/php_error.log
3. Verificar permisos de archivos

Problema: "Headers already sent"
Solución:
1. Verificar que no haya salida antes de header()
2. Revisar BOM en archivos PHP
3. Usar ob_start() al inicio si es necesario

REQ-INST-08: Optimización para MacBook Air 2017

Recomendaciones específicas:

```
# 1. Limitar logs
max_log_size = 10M
# 2. Cerrar aplicaciones innecesarias
# 3. Usar Activity Monitor para verificar RAM disponible
# 4. Detener MAMP cuando no se use
# 5. Trabajar con máximo 10 tabs abiertos en navegador
```

REQ-INST-09: Credenciales por Defecto

Usuario Administrador:

- Username: admin
- Password: Admin123!
- Roles: Administrador

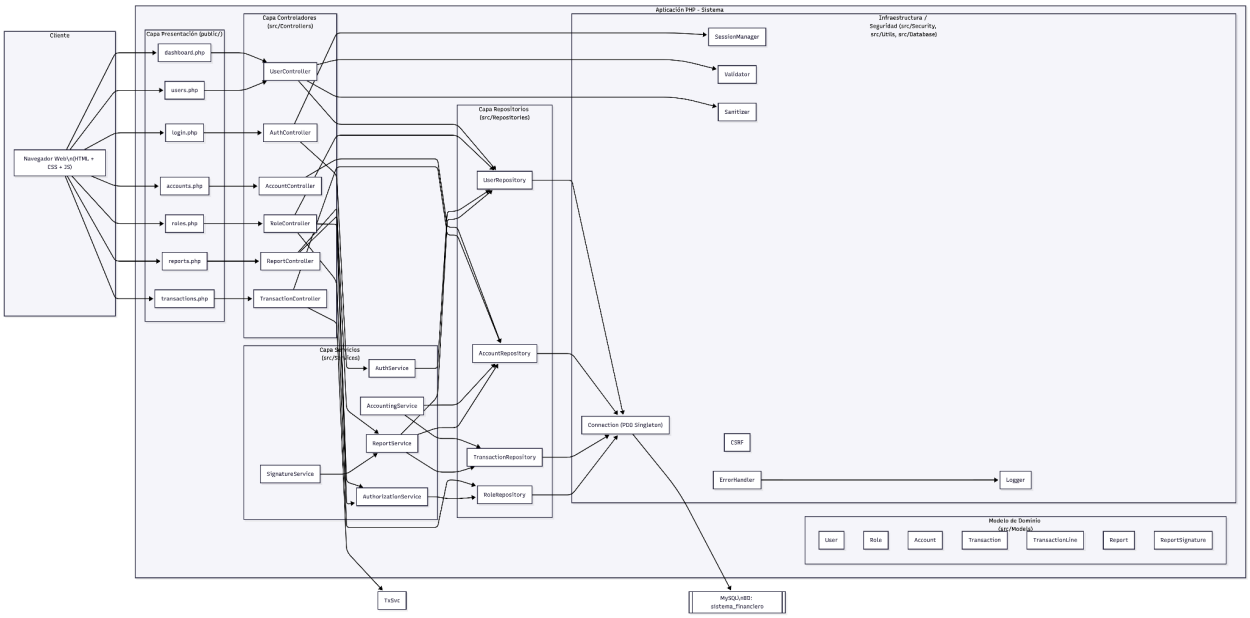
IMPORTANTE: Cambiar contraseña en producción

REQ-INST-10: Estructura de Archivos de Instalación

```
sistema-financiero/
├── docs/
```

```
│   ├── database/
│   │   └── sistema_financiero.sql  ← SCRIPT A IMPORTAR
│   ├── uml/
│   └── README.md  ← INSTRUCCIONES DETALLADAS
├── config/
│   └── database.php  ← CONFIGURAR AQUÍ
├── public/
│   └── index.php  ← PUNTO DE ENTRADA
└── storage/
    └── logs/  ← VERIFICAR PERMISOS
```

Estructura del Sistema Modelo Estructural:



sistema-financiero/

```
│
├── config/
│   ├── database.php          # Configuración de BD
│   ├── constants.php         # Constantes globales
│   └── security.php          # Headers de seguridad
│
├── src/
│   ├── Controllers/          # Controladores MVC
│   │   ├── AuthController.php
│   │   ├── UserController.php
│   │   ├── AccountController.php
│   │   ├── TransactionController.php
│   │   └── ReportController.php
│   │
```

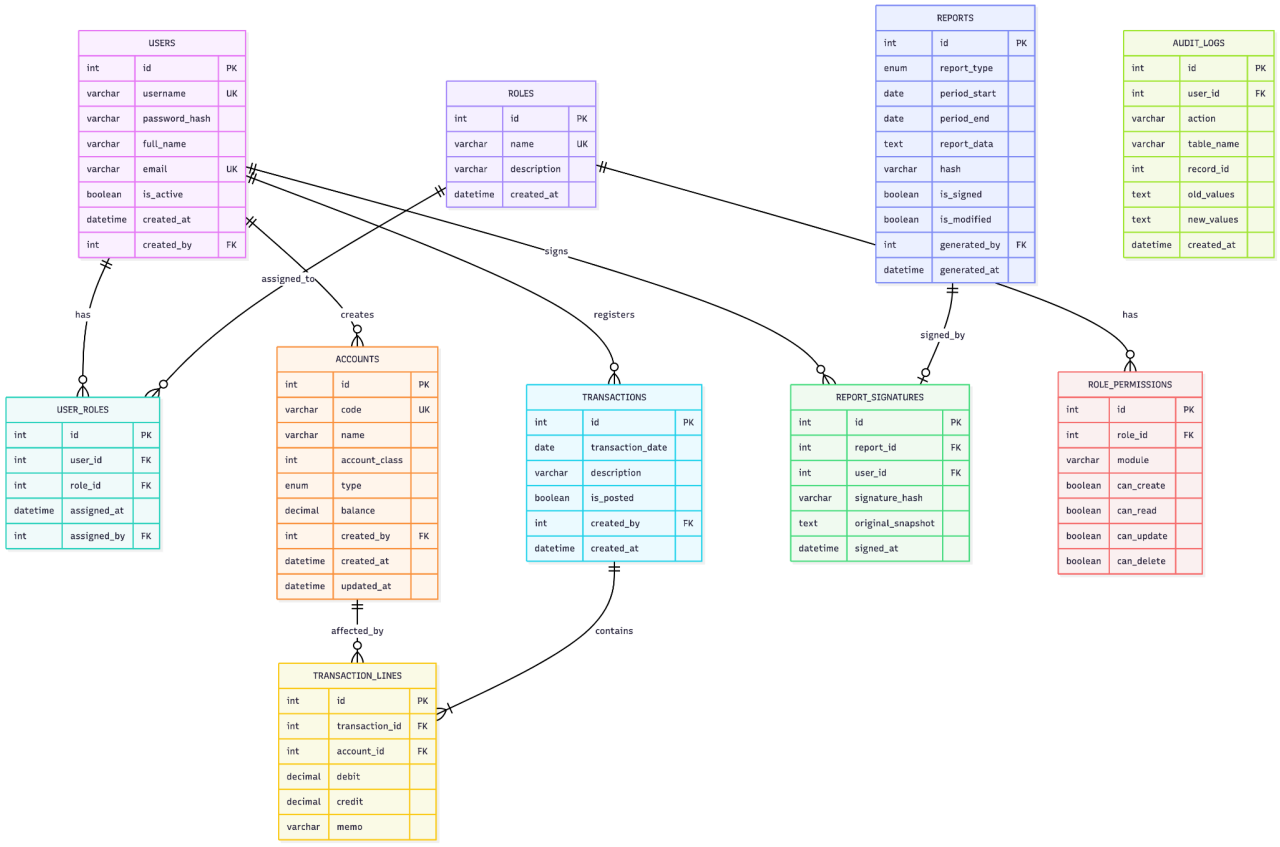
```
│   ├── Models/              # Entidades del dominio
│   │   ├── User.php
│   │   ├── Role.php
│   │   ├── Account.php
│   │   ├── Transaction.php
│   │   ├── Report.php
│   │   └── ReportSignature.php
│   │
│   ├── Services/            # Lógica de negocio compleja
│   │   ├── AuthService.php
│   │   ├── ReportService.php
│   │   ├── SignatureService.php
│   │   └── AccountingService.php
│   │
│   ├── Repositories/        # Acceso a datos
│   │   ├── BaseRepository.php
│   │   ├── UserRepository.php
│   │   ├── AccountRepository.php
│   │   └── TransactionRepository.php
│   │
│   ├── Database/
│   │   └── Connection.php     # Singleton PDO
│   │
│   ├── Security/
│   │   ├── Validator.php      # Validación de datos
│   │   ├── Sanitizer.php      # Sanitización
│   │   ├── CSRF.php           # Tokens CSRF
│   │   └── SessionManager.php # Manejo de sesiones
│   │
│   ├── Interfaces/          # Contratos
│   │   ├── ValidatorInterface.php
│   │   ├── RepositoryInterface.php
│   │   └── ErrorHandlerInterface.php
│   │
│   └── Utils/
│       ├── ErrorHandler.php   # Manejo centralizado
│       ├── Logger.php         # Sistema de logs
│       └── MoneyHelper.php    # Operaciones monetarias
│
├── public/                  # Carpeta pública (web root)
│   ├── index.php            # Página pública
│   ├── login.php            # Login
│   ├── dashboard.php        # Panel principal
│   ├── users.php            # CRUD usuarios
│   ├── accounts.php         # CRUD catálogo
│   ├── transactions.php     # CRUD transacciones
│   ├── reports.php          # Reportes
│   ├── logout.php           # Cerrar sesión
│   ├── assets/
```

```
│   │   ├── css/
│   │   │   └── style.css      # ← CSS PRINCIPAL (OBLIGATORIO)
│   │   ├── js/
│   │   │   ├── validator.js   # Validación cliente
│   │   │   └── main.js        # Scripts generales
│   │   └── images/
│   │       └── logo.png
│   └── .htaccess              # Configuración Apache
│
├── views/                     # Plantillas PHP
│   ├── layouts/
│   │   ├── header.php         # Header común
│   │   ├── footer.php         # Footer común
│   │   └── nav.php            # Navegación
│   ├── auth/
│   │   └── login.php          # Vista login
│   ├── users/
│   │   ├── list.php
│   │   ├── create.php
│   │   └── edit.php
│   ├── accounts/
│   │   ├── list.php
│   │   ├── create.php
│   │   └── edit.php
│   ├── transactions/
│   │   ├── list.php
│   │   ├── create.php
│   │   └── view.php
│   └── reports/
│       ├── income_statement.php
│       ├── balance_sheet.php
│       └── sign.php
│
├── storage/
│   ├── logs/                  # Logs del sistema
│   │   └── .gitkeep
│   └── signatures/            # Snapshots de firmas
│       └── .gitkeep
│
├── docs/
│   ├── uml/                   # Diagramas UML (PNG)
│   │   ├── casos-de-uso.png
│   │   ├── clases.png
│   │   ├── secuencia-login.png
│   │   ├── secuencia-transaccion.png
│   │   ├── secuencia-firma.png
│   │   ├── actividad-transaccion.png
│   │   ├── actividad-reporte.png
│   │   ├── estados-transaccion.png
```

```
│   │   ├── componentes.png
│   │   └── er-diagram.png
│   ├── database/              # Scripts SQL
│   │   └── sistema_financiero.sql
│   ├── Vision-del-Proyecto.md
│   ├── Casos-de-Uso.md
│   ├── Requisitos-Sistema.md
│   ├── Requisitos-Instalacion.md
│   ├── Modelo-Estructural.md
│   ├── Modelo-Dinamico.md
│   └── README.md
│
├── tests/                     # Pruebas (opcional)
│   └── AccountingTest.php
│
├── .gitignore
├── README.md
└── INSTALL.md                 # Instrucciones instalación
```

Base de Datos (Script de la Base de Datos y Diagrama de E/R):

Diagrama E/R:



Script Base de Datos:

```
-- ========================================================
-- SISTEMA FINANCIERO - BASE DE DATOS
-- Universidad Tecnológica de Panamá
-- Ingeniería Web - Sistema Financiero
-- ========================================================

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

CREATE DATABASE IF NOT EXISTS sistema_financiero
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

USE sistema_financiero;

-- ========================================================
-- TABLA: users
-- Almacena usuarios del sistema con auditoría
-- ========================================================
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by INT NULL,
            updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    last_login TIMESTAMP NULL,
    CONSTRAINT fk_user_created_by FOREIGN KEY (created_by)
       REFERENCES users(id) ON DELETE SET NULL,
    INDEX idx_username (username),
    INDEX idx_email (email),
    INDEX idx_is_active (is_active)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ========================================================
-- TABLA: roles
-- Define roles del sistema (Admin, Contador, Gerente)
```

```
-- ====================================================
CREATE TABLE roles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX idx_name (name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ====================================================
-- TABLA: user_roles
-- Relación muchos-a-muchos entre usuarios y roles
-- ====================================================
CREATE TABLE user_roles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    role_id INT NOT NULL,
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    assigned_by INT NULL,
    CONSTRAINT fk_user_role_user FOREIGN KEY (user_id)
        REFERENCES users(id) ON DELETE CASCADE,
    CONSTRAINT fk_user_role_role FOREIGN KEY (role_id)
        REFERENCES roles(id) ON DELETE CASCADE,
    CONSTRAINT fk_user_role_assigned_by FOREIGN KEY (assigned_by)
        REFERENCES users(id) ON DELETE SET NULL,
    UNIQUE KEY unique_user_role (user_id, role_id),
    INDEX idx_user_id (user_id),
    INDEX idx_role_id (role_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ====================================================
-- TABLA: role_permissions
-- Permisos granulares por módulo para cada rol
-- ====================================================
CREATE TABLE role_permissions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    role_id INT NOT NULL,
    module VARCHAR(50) NOT NULL,
    can_create BOOLEAN DEFAULT FALSE,
    can_read BOOLEAN DEFAULT FALSE,
    can_update BOOLEAN DEFAULT FALSE,
    can_delete BOOLEAN DEFAULT FALSE,
    CONSTRAINT fk_role_permission_role FOREIGN KEY (role_id)
        REFERENCES roles(id) ON DELETE CASCADE,
```

```
    UNIQUE KEY unique_role_module (role_id, module),
    INDEX idx_role_id (role_id),
    INDEX idx_module (module)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- TABLA: accounts (Catálogo de Cuentas)
-- Clasificación contable según NIIF
-- Clases: 1=Activo, 2=Pasivo, 3=Patrimonio,
--         4=Ingresos, 5=Gastos, 6=Costos, 7=Otros Gastos
-- ======================================================
CREATE TABLE accounts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    code VARCHAR(20) UNIQUE NOT NULL,
    name VARCHAR(150) NOT NULL,
    account_class TINYINT NOT NULL CHECK (account_class BETWEEN 1 AND 7),
    account_type ENUM('debit', 'credit') NOT NULL,
    balance DECIMAL(15, 2) DEFAULT 0.00,
    is_active BOOLEAN DEFAULT TRUE,
    created_by INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    CONSTRAINT fk_account_created_by FOREIGN KEY (created_by)
        REFERENCES users(id) ON DELETE RESTRICT,
    INDEX idx_code (code),
    INDEX idx_class (account_class),
    INDEX idx_active (is_active)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- TABLA: transactions (Diario General)
-- Registra transacciones contables con partida doble
-- ======================================================
CREATE TABLE transactions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    transaction_date DATE NOT NULL,
    description VARCHAR(255) NOT NULL,
    is_posted BOOLEAN DEFAULT FALSE,
    created_by INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    CONSTRAINT fk_transaction_created_by FOREIGN KEY (created_by)
```

```
        REFERENCES users(id) ON DELETE RESTRICT,
    INDEX idx_date (transaction_date),
    INDEX idx_posted (is_posted),
    INDEX idx_created_by (created_by)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- TABLA: transaction_lines (Líneas de Transacción)
-- Detalle de cada movimiento contable
-- CONSTRAINT: Debe cumplir Débitos = Créditos por transacción
-- ======================================================
CREATE TABLE transaction_lines (
    id INT AUTO_INCREMENT PRIMARY KEY,
    transaction_id INT NOT NULL,
    account_id INT NOT NULL,
    debit DECIMAL(15, 2) DEFAULT 0.00 CHECK (debit >= 0),
    credit DECIMAL(15, 2) DEFAULT 0.00 CHECK (credit >= 0),
    memo VARCHAR(255),
    CONSTRAINT fk_line_transaction FOREIGN KEY (transaction_id)
        REFERENCES transactions(id) ON DELETE CASCADE,
    CONSTRAINT fk_line_account FOREIGN KEY (account_id)
        REFERENCES accounts(id) ON DELETE RESTRICT,
    CONSTRAINT chk_debit_or_credit CHECK (
        (debit > 0 AND credit = 0) OR (credit > 0 AND debit = 0)
    ),
    INDEX idx_transaction_id (transaction_id),
    INDEX idx_account_id (account_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- TABLA: reports (Informes Financieros)
-- Almacena reportes generados con integridad
-- ======================================================
CREATE TABLE reports (
    id INT AUTO_INCREMENT PRIMARY KEY,
    report_type ENUM('income_statement', 'balance_sheet') NOT NULL,
    period_start DATE NOT NULL,
    period_end DATE NOT NULL,
    report_data JSON NOT NULL,
    hash VARCHAR(64) NOT NULL,
    is_signed BOOLEAN DEFAULT FALSE,
    is_modified BOOLEAN DEFAULT FALSE,
    generated_by INT NOT NULL,
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    CONSTRAINT fk_report_generated_by FOREIGN KEY (generated_by)
        REFERENCES users(id) ON DELETE RESTRICT,
    INDEX idx_type (report_type),
    INDEX idx_period (period_start, period_end),
    INDEX idx_signed (is_signed)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- TABLA: report_signatures (Firmas de Reportes)
-- Sistema de firma digital para autenticación
-- ======================================================
CREATE TABLE report_signatures (
    id INT AUTO_INCREMENT PRIMARY KEY,
    report_id INT UNIQUE NOT NULL,
    user_id INT NOT NULL,
    signature_hash VARCHAR(64) NOT NULL,
    original_snapshot JSON NOT NULL,
    signed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_signature_report FOREIGN KEY (report_id)
        REFERENCES reports(id) ON DELETE CASCADE,
    CONSTRAINT fk_signature_user FOREIGN KEY (user_id)
        REFERENCES users(id) ON DELETE RESTRICT,
    INDEX idx_report_id (report_id),
    INDEX idx_user_id (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- TABLA: audit_logs (Logs de Auditoría)
-- Registra todas las operaciones críticas del sistema
-- ======================================================
CREATE TABLE audit_logs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NULL,
    action VARCHAR(50) NOT NULL,
    table_name VARCHAR(50) NOT NULL,
    record_id INT,
    old_values JSON,
    new_values JSON,
    ip_address VARCHAR(45),
    user_agent VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_audit_user FOREIGN KEY (user_id)
        REFERENCES users(id) ON DELETE SET NULL,
    INDEX idx_user_id (user_id),
```

```
    INDEX idx_action (action),
    INDEX idx_table_name (table_name),
    INDEX idx_created_at (created_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ======================================================
-- DATOS INICIALES: Roles del Sistema
-- ======================================================
INSERT INTO roles (name, description) VALUES
('Administrador', 'Control total del sistema, gestión de usuarios y roles'),
('Contador', 'Registro de transacciones, gestión del catálogo de cuentas'),
('Gerente Financiero', 'Generación y firma de reportes financieros'),
('Auditor', 'Consulta de reportes y logs, sin capacidad de modificación');

-- ======================================================
-- DATOS INICIALES: Usuario Administrador por defecto
-- Password: Admin123! (cambiar en producción)
-- ======================================================
INSERT INTO users (username, password_hash, full_name, email, is_active) VALUES
('admin', '$2y$10$rZ8vN5K.Mj3xGqJ5qH6X2eZJYqP8wL3X6qR8vN5K.Mj3xGqJ5qH6X2',
 'Administrador del Sistema', 'admin@sistemafinanciero.com', TRUE);

-- Asignar rol de Administrador al usuario admin
INSERT INTO user_roles (user_id, role_id, assigned_by)
VALUES (1, 1, NULL);

-- ======================================================
-- PERMISOS: Configuración por rol
-- ======================================================
-- Administrador: Acceso completo
INSERT INTO role_permissions (role_id, module, can_create, can_read, can_update, can_delete) VALUES
(1, 'users', TRUE, TRUE, TRUE, FALSE),
(1, 'roles', TRUE, TRUE, TRUE, TRUE),
(1, 'accounts', TRUE, TRUE, TRUE, TRUE),
(1, 'transactions', TRUE, TRUE, TRUE, TRUE),
(1, 'reports', TRUE, TRUE, FALSE, FALSE);

-- Contador: Gestión contable
INSERT INTO role_permissions (role_id, module, can_create, can_read, can_update, can_delete) VALUES
(2, 'accounts', TRUE, TRUE, TRUE, FALSE),
(2, 'transactions', TRUE, TRUE, TRUE, FALSE),
(2, 'reports', FALSE, TRUE, FALSE, FALSE);
```

```
-- Gerente Financiero: Reportes y firma
INSERT INTO role_permissions (role_id, module, can_create, can_read, can_update, can_delete) VALUES
(3, 'reports', TRUE, TRUE, FALSE, FALSE),
(3, 'signatures', TRUE, TRUE, FALSE, FALSE),
(3, 'transactions', FALSE, TRUE, FALSE, FALSE);

-- ======================================================
-- DATOS EJEMPLO: Catálogo de Cuentas Básico
-- ======================================================
INSERT INTO accounts (code, name, account_class, account_type, created_by) VALUES
-- CLASE 1: ACTIVOS
('1.1.01', 'Caja General', 1, 'debit', 1),
('1.1.02', 'Bancos', 1, 'debit', 1),
('1.2.01', 'Cuentas por Cobrar', 1, 'debit', 1),
('1.3.01', 'Inventarios', 1, 'debit', 1),
('1.4.01', 'Equipos de Oficina', 1, 'debit', 1),

-- CLASE 2: PASIVOS
('2.1.01', 'Cuentas por Pagar', 2, 'credit', 1),
('2.2.01', 'Préstamos Bancarios', 2, 'credit', 1),

-- CLASE 3: PATRIMONIO
('3.1.01', 'Capital Social', 3, 'credit', 1),
('3.2.01', 'Utilidades Retenidas', 3, 'credit', 1),
('3.3.01', 'Utilidad del Ejercicio', 3, 'credit', 1),

-- CLASE 4: INGRESOS
('4.1.01', 'Ventas', 4, 'credit', 1),
('4.2.01', 'Intereses Ganados', 4, 'credit', 1),

-- CLASE 5: GASTOS
('5.1.01', 'Sueldos y Salarios', 5, 'debit', 1),
('5.2.01', 'Alquiler', 5, 'debit', 1),
('5.3.01', 'Servicios Públicos', 5, 'debit', 1);

-- ======================================================
-- TRIGGERS: Validación de partida doble
-- ======================================================
DELIMITER $$

CREATE TRIGGER check_balanced_transaction
BEFORE UPDATE ON transactions
```

```
FOR EACH ROW
BEGIN
   DECLARE total_debit DECIMAL(15,2);
   DECLARE total_credit DECIMAL(15,2);

   IF NEW.is_posted = TRUE AND OLD.is_posted = FALSE THEN
      SELECT COALESCE(SUM(debit), 0), COALESCE(SUM(credit), 0)
      INTO total_debit, total_credit
      FROM transaction_lines
      WHERE transaction_id = NEW.id;

      IF total_debit != total_credit THEN
         SIGNAL SQLSTATE '45000'
              SET MESSAGE_TEXT = 'Error: La transacción no está balanceada. Débitos deben
igualar créditos.';
      END IF;
   END IF;
END$$

-- ====================================================
-- TRIGGER: Actualizar balance de cuentas
-- ====================================================
CREATE TRIGGER update_account_balance
AFTER INSERT ON transaction_lines
FOR EACH ROW
BEGIN
   DECLARE tx_posted BOOLEAN;

   SELECT is_posted INTO tx_posted
   FROM transactions
   WHERE id = NEW.transaction_id;

   IF tx_posted = TRUE THEN
      UPDATE accounts
      SET balance = balance + NEW.debit - NEW.credit
      WHERE id = NEW.account_id;
   END IF;
END$$

-- ====================================================
-- TRIGGER: Log de auditoría automático
-- ====================================================
CREATE TRIGGER audit_account_changes
AFTER UPDATE ON accounts
```

```
FOR EACH ROW
BEGIN
    INSERT INTO audit_logs (user_id, action, table_name, record_id, old_values, new_values)
    VALUES (
        NEW.created_by,
        'UPDATE',
        'accounts',
        NEW.id,
        JSON_OBJECT('balance', OLD.balance, 'name', OLD.name),
        JSON_OBJECT('balance', NEW.balance, 'name', NEW.name)
    );
END$$

DELIMITER ;

-- ======================================================
-- VISTAS: Consultas frecuentes optimizadas
-- ======================================================

-- Vista: Estado de Resultados
CREATE VIEW view_income_statement AS
SELECT
    a.code,
    a.name,
    a.account_class,
    SUM(CASE
        WHEN a.account_type = 'credit' THEN tl.credit - tl.debit
        ELSE tl.debit - tl.credit
    END) as amount
FROM accounts a
INNER JOIN transaction_lines tl ON a.id = tl.account_id
INNER JOIN transactions t ON tl.transaction_id = t.id
WHERE t.is_posted = TRUE
  AND a.account_class IN (4, 5, 6, 7)
GROUP BY a.id, a.code, a.name, a.account_class
ORDER BY a.code;

-- Vista: Balance General
CREATE VIEW view_balance_sheet AS
SELECT
    a.code,
    a.name,
    a.account_class,
    a.balance as amount
```

```
FROM accounts a
WHERE a.account_class IN (1, 2, 3)
  AND a.is_active = TRUE
ORDER BY a.code;

COMMIT;
```