

# UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

Departamento de Ciencias de la Computación

Ingeniería en Software

## INFORME DE ERRORES

*Sistema de Gestión y Validación de Comprobantes para  
Eventos*

### **Integrantes:**

David Morán  
Gabriel Vivanco  
Samir Mideros  
Alison Miranda

Versión 1.0.0  
21 de enero de 2026

# Índice

<b>1. Resumen Ejecutivo</b>	<b>3</b>
1.1. Información del Proyecto . . . . .	3
1.2. Resultados Principales . . . . .	3
<b>2. Estado de Pruebas Unitarias</b>	<b>3</b>
2.1. Pruebas Implementadas . . . . .	3
2.1.1. ComprobanteModel.test.js . . . . .	4
2.1.2. UserModel.test.js . . . . .	4
2.2. Desglose por Módulo . . . . .	5
2.3. Pruebas Faltantes . . . . .	5
2.3.1. Modelos Restantes . . . . .	5
2.3.2. Controladores . . . . .	5
<b>3. Advertencias y Problemas Detectados</b>	<b>5</b>
3.1. WARNING-001: Variable no utilizada en StaffController . . . . .	6
3.2. WARNING-002: Dependencia faltante en useEffect . . . . .	6
3.3. WARNING-003: Variable no utilizada en RegistrarGastoOperativo . . . . .	7
3.4. WARNING-004: Dependencia faltante en useEffect . . . . .	7
<b>4. Vulnerabilidades de Seguridad</b>	<b>7</b>
4.1. Resumen de Vulnerabilidades . . . . .	7
4.2. VULN-001: React Router - XSS via Open Redirects . . . . .	7
4.3. VULN-002: React Router (Indirecta) . . . . .	9
4.4. VULN-003: esbuild - CORS Bypass en Development Server . . . . .	9
4.5. VULN-004: Vite (Indirecta) . . . . .	10
<b>5. Problemas Adicionales Detectados</b>	<b>10</b>
5.1. ISSUE-001: Uso de dbConnection en ComprobanteController . . . . .	10
5.2. ISSUE-002: Métodos duplicados en ComprobanteController . . . . .	10
<b>6. Análisis por Categoría</b>	<b>11</b>
6.1. Calidad de Código . . . . .	11
6.2. Seguridad . . . . .	11
6.3. Arquitectura . . . . .	11
<b>7. Plan de Implementación de Pruebas</b>	<b>11</b>
7.1. Fase 1: Configuración e Infraestructura . . . . .	12
7.2. Fase 2: Pruebas de Modelos . . . . .	12
7.3. Fase 3: Pruebas de Controladores . . . . .	13
7.4. Fase 4: Pruebas de Componentes React . . . . .	13
7.5. Fase 5: Pruebas de Patrones y Utilidades . . . . .	13
<b>8. Métricas de Calidad</b>	<b>14</b>
8.1. Cumplimiento de Estándares . . . . .	14
8.2. Cobertura de Requisitos Funcionales . . . . .	15

<b>9. Conclusiones y Recomendaciones</b>	<b>15</b>
9.1. Conclusiones . . . . .	15
9.2. Recomendaciones Prioritarias . . . . .	16
9.2.1. Acciones Obligatorias Antes de Producción . . . . .	16
9.2.2. Mejoras de Calidad Recomendadas . . . . .	16
9.2.3. Mejoras de Mejora Continua (Opcional) . . . . .	16
9.3. Aprobación para Producción . . . . .	17
<b>10. Anexos</b>	<b>17</b>
10.1. Anexo A: Comandos para Correcciones . . . . .	17
10.2. Anexo B: Archivos que Requieren Atención . . . . .	17
10.3. Anexo C: Métricas por Directorio . . . . .	18
10.4. Anexo D: Resumen de Hallazgos . . . . .	18
10.5. Anexo E: Próximos Pasos . . . . .	19
<b>11. Referencias</b>	<b>19</b>

# 1. Resumen Ejecutivo

El presente informe detalla el análisis exhaustivo de las pruebas unitarias implementadas en el Sistema de Gestión y Validación de Comprobantes para Eventos, desarrollado como proyecto académico en la Universidad de las Fuerzas Armadas ESPE. El sistema fue construido utilizando tecnologías modernas como React para el frontend, Node.js con Express para el backend, y MongoDB como base de datos.

## 1.1. Información del Proyecto

Atributo	Valor
Proyecto	Sistema de Gestión y Validación de Comprobantes
Versión	1.0.0
Fecha de Análisis	21 de enero de 2026
Framework de Pruebas	Vitest v4.0.17 + React Testing Library
Total de Pruebas	21
Archivos de Prueba	2 de 15 planificados
Duración de Ejecución	1.22 segundos

Cuadro 1: Información general del proyecto

## 1.2. Resultados Principales

A la fecha del análisis, el sistema cuenta con 21 pruebas unitarias implementadas, distribuidas en 2 archivos de prueba correspondientes a los modelos principales del sistema. La ejecución de estas pruebas fue exitosa en su totalidad, sin fallos detectados.

La cobertura de código alcanzada es del 90.19 %, superando el objetivo mínimo establecido del 80 %. Este porcentaje se desglosa de la siguiente manera:

Categoría	Cobertura	Estado
Statements	90.19 %	Excelente
Branches	94.04 %	Excelente
Functions	62.5 %	Necesita mejora
Lines	90.19 %	Excelente
Cobertura Global	90.19 %	Sobre objetivo

Cuadro 2: Estadísticas de cobertura actual

# 2. Estado de Pruebas Unitarias

## 2.1. Pruebas Implementadas

Las pruebas unitarias implementadas se encuentran en los modelos del sistema, que son componentes fundamentales para el manejo de datos y validaciones.

### 2.1.1. ComprobanteModel.test.js

Este archivo contiene 12 pruebas unitarias que verifican el correcto funcionamiento del modelo de comprobantes. Las pruebas implementadas son:

- UT-COMP-001: Crear comprobante con datos válidos
- UT-COMP-002: Crear con valores por defecto
- UT-COMP-003: Validar datos correctos
- UT-COMP-004: Rechazar número de comprobante vacío
- UT-COMP-005: Rechazar proveedor vacío
- UT-COMP-006: Rechazar monto inválido
- UT-COMP-007: Rechazar cédula de staff vacía
- UT-COMP-008: Convertir a JSON
- UT-COMP-009: Crear desde JSON
- UT-COMP-010: Marcar como validado
- UT-COMP-011: Aprobar comprobante
- UT-COMP-012: Rechazar comprobante

La cobertura alcanzada es del 84.84 %, con líneas sin cubrir en las posiciones 32 y 58-71.

### 2.1.2. UserModel.test.js

Este archivo contiene 9 pruebas unitarias que validan el modelo de usuarios del sistema:

- UT-USER-001: Crear usuario con datos válidos
- UT-USER-002: Crear con valores por defecto
- UT-USER-003: Validar datos correctos
- UT-USER-004: Rechazar email inválido
- UT-USER-005: Rechazar cédula inválida
- UT-USER-006: Rechazar contraseña vacía
- UT-USER-007: Validar roles permitidos
- UT-USER-008: Convertir a JSON sin contraseña
- UT-USER-009: Verificar createdAt es Date

La cobertura alcanzada es del 100 %, indicando que todas las líneas de código del modelo han sido probadas.

## 2.2. Desglose por Módulo

Categoría	Pruebas	Archivos	Estado
Modelos	21 (100 %)	2/5	Implementado
Controladores	0	0/6	Sin implementar
Componentes React	0	0/8	Sin implementar
Patrones de Diseño	0	0/3	Sin implementar
API Client	0	0/1	Sin implementar

Cuadro 3: Estado de pruebas por módulo

## 2.3. Pruebas Faltantes

El análisis revela que existen varios módulos del sistema que requieren implementación de pruebas unitarias:

### 2.3.1. Modelos Restantes

- **StaffMemberModel.js**: 0 pruebas (estimado: 8 necesarias)
- **PagoExcepcionalModel.js**: 0 pruebas (estimado: 8 necesarias)
- **GastoOperativoModel.js**: 0 pruebas (estimado: 9 necesarias)

Total estimado: 25 pruebas de modelos pendientes. El impacto de esta falta es medio, ya que se requiere completar la cobertura de validaciones.

### 2.3.2. Controladores

- **StaffController.js**: 0 pruebas (estimado: 12 necesarias)
- **ComprobanteController.js**: 0 pruebas (estimado: 15 necesarias)
- **PagoExcepcionalController.js**: 0 pruebas (estimado: 8 necesarias)
- **GastoOperativoController.js**: 0 pruebas (estimado: 10 necesarias)
- **BusquedaController.js**: 0 pruebas (estimado: 7 necesarias)
- **GlobalControllers.js**: 0 pruebas (estimado: 3 necesarias)

Total estimado: 55 pruebas de controladores necesarias. El impacto de esta falta es alto, ya que la lógica de negocio debe estar completamente probada.

## 3. Advertencias y Problemas Detectados

Durante el análisis del código, se identificaron cuatro advertencias (warnings) que, aunque no son errores críticos, representan áreas de mejora en las buenas prácticas de programación.

### 3.1. WARNING-001: Variable no utilizada en StaffController

- **Severidad:** Baja
- **Tipo:** Calidad de Código
- **Archivo:** src/controllers/StaffController.js:157
- **Regla ESLint:** no-unused-vars

**Descripción:** La variable `error` está definida en un bloque `catch` pero ESLint la considera no utilizada debido al sombreado (`shadowing`). Sin embargo, la variable se usa correctamente para acceder a `error.message`.

**Contexto del código:**

```
catch (error) {
  errors.push({
    fila: index + 2,
    errores: ['Error al procesar: ${error.message}']
  });
}
```

**Impacto:** Mínimo. No afecta la funcionalidad del sistema.

**Prioridad:** Baja.

### 3.2. WARNING-002: Dependencia faltante en useEffect

- **Severidad:** Baja
- **Tipo:** React Hooks
- **Archivo:** src/views/Contadora/RegistrarPagoExcepcional.jsx:27
- **Regla ESLint:** react-hooks/exhaustive-deps

**Descripción:** El hook `useEffect` tiene una dependencia faltante (`cargarPagos`) en su arreglo de dependencias.

**Contexto del código:**

```
const cargarPagos = async () => {
  const result = await pagoController.obtenerPagos();
  if (result.success) {
    setPagos(result.data);
  }
};

useEffect(() => {
  cargarPagos();
}, []); // Warning: falta 'cargarPagos' en dependencias
```

**Impacto:** Medio. Puede causar closures obsoletos en re-renderizados, aunque es improbable en este caso específico.

**Solución propuesta:** Mover la lógica dentro del `useEffect` o utilizar `useCallback` para estabilizar la función.

**Prioridad:** Media.

### 3.3. WARNING-003: Variable no utilizada en RegistrarGastoOperativo

- Severidad: Baja
- Tipo: Calidad de Código
- Archivo: `src/views/Shared/RegistrarGastoOperativo.jsx:25`
- Regla ESLint: `no-unused-vars`

**Descripción:** La variable `selectedGastoId` se asigna pero nunca se utiliza en el código.

**Impacto:** Mínimo. Representa código muerto que ocupa memoria innecesariamente.

**Prioridad:** Baja.

### 3.4. WARNING-004: Dependencia faltante en useEffect

- Severidad: Baja
- Tipo: React Hooks
- Archivo: `src/views/Shared/RegistrarGastoOperativo.jsx:29`
- Regla ESLint: `react-hooks/exhaustive-deps`

**Descripción:** Similar a WARNING-002, el hook `useEffect` tiene una dependencia faltante.

**Impacto:** Medio.

**Prioridad:** Media.

## 4. Vulnerabilidades de Seguridad

El análisis de seguridad mediante `npm audit` reveló la presencia de 4 vulnerabilidades que requieren atención.

### 4.1. Resumen de Vulnerabilidades

Severidad	Cantidad	Estado
Crítica	0	Ninguna
Alta	2	Requiere actualización
Moderada	2	Requiere actualización
Baja	0	Ninguna

Cuadro 4: Resumen de vulnerabilidades detectadas

### 4.2. VULN-001: React Router - XSS via Open Redirects

- Severidad: Alta

- **CVE:** GHSA-2w69-qvlg-hvix
- **CWE:** CWE-79 (Cross-site Scripting)
- **CVSS Score:** 8.0/10
- **Paquete Afectado:** @remix-run/router  $\leq$  1.23.1

**Descripción:** React Router es vulnerable a ataques de Cross-Site Scripting (XSS) mediante redirecciones abiertas. Un atacante podría:

- Crear URLs maliciosas que redirijan a sitios externos
- Ejecutar scripts en el contexto del dominio de la aplicación
- Comprometer sesiones de usuario

**Vector de Ataque:**

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:N

- Attack Vector: Network
- Attack Complexity: High
- Privileges Required: None
- User Interaction: Required
- Confidentiality Impact: High
- Integrity Impact: High

**Versión Vulnerable:**

- react-router-dom@6.20.0
- react-router@6.20.0
- @remix-run/router@1.14.0

**Versión Segura:**

- react-router-dom@7.0.2 o superior
- react-router@7.0.2 o superior
- @remix-run/router@1.24.0 o superior

**Solución:**

```
npm install react-router-dom@latest
npm audit
```

**Impacto en el Proyecto:**

- Probabilidad: Media (requiere interacción del usuario)
- Impacto: Alto (puede comprometer sesiones y datos)
- Mitigación Actual: Ninguna

**Prioridad:** Alta. Debe actualizarse antes de producción.

### 4.3. VULN-002: React Router (Indirecta)

- **Severidad:** Alta
- **Paquete Afectado:** react-router (6.4.0-pre.0 - 6.30.2)
- **Causa:** Heredada de @remix-run/router

**Descripción:** Vulnerabilidad transitiva desde @remix-run/router. Se soluciona automáticamente al actualizar react-router-dom.

**Prioridad:** Alta. Se soluciona con VULN-001.

### 4.4. VULN-003: esbuild - CORS Bypass en Development Server

- **Severidad:** Moderada
- **CVE:** GHSA-67mh-4wv8-2f99
- **CWE:** CWE-346 (Origin Validation Error)
- **CVSS Score:** 5.3/10
- **Paquete Afectado:** esbuild  $\leq$  0.24.2

**Descripción:** esbuild permite que cualquier sitio web envíe solicitudes al servidor de desarrollo y lea las respuestas, evadiendo las políticas CORS.

**Riesgo:**

- Solo afecta en desarrollo (servidor de Vite)
- NO afecta en producción (build compilado)
- Un atacante podría leer código fuente durante desarrollo

**Versión Vulnerable:** vite@5.0.8

**Versión Segura:** vite@7.3.1 o superior

**Mitigación sin Actualizar:**

- No exponer servidor de desarrollo públicamente
- Usar localhost solo, no 0.0.0.0
- No compartir URLs del servidor de desarrollo con terceros
- Usar VPN/firewall para desarrollo remoto

**Impacto en el Proyecto:**

- Probabilidad: Baja (solo en desarrollo, requiere acceso a la red)
- Impacto: Moderado (fuga de código fuente)
- Mitigación Actual: Servidor local protegido

**Prioridad:** Media. Mitigado por uso local.

## 4.5. VULN-004: Vite (Indirecta)

- **Severidad:** Moderada
- **Paquete Afectado:** vite (heredado de esbuild)

**Descripción:** Dependencia transitiva de esbuild vulnerable.

**Nota:** Actualizar Vite de 5.x a 7.x es un cambio mayor que puede requerir ajustes de configuración.

**Prioridad:** Media. Evaluar cambios incompatibles antes de actualizar.

## 5. Problemas Adicionales Detectados

### 5.1. ISSUE-001: Uso de dbConnection en ComprobanteController

- **Severidad:** Media
- **Tipo:** Arquitectura
- **Archivo:** src/controllers/ComprobanteController.js

**Descripción:** El método `validarContraDatosOficiales()` utiliza acceso directo a MongoDB a través de `dbConnection`, lo cual causará errores en el navegador.

**Líneas afectadas:**

- Línea ~192: `const db = await this.dbConnection.connect();`
- Línea ~193: `const collection = db.collection(this.collectionName);`

**Solución:** Migrar el método para usar la API REST del backend en lugar de acceso directo a la base de datos.

**Prioridad:** Media. Causará error al usar la funcionalidad RF05.

### 5.2. ISSUE-002: Métodos duplicados en ComprobanteController

- **Severidad:** Baja
- **Tipo:** Calidad de Código
- **Archivo:** src/controllers/ComprobanteController.js

**Descripción:** El archivo tiene múltiples definiciones de los mismos métodos:

- `getAllComprobantes()` aparece 2 veces (línea ~310 y ~345)
- `getComprobanteById()` aparece 2 veces (línea ~325 y ~358)

**Impacto:** La segunda definición sobrescribe la primera, causando comportamiento inconsistente.

**Prioridad:** Media. Requiere limpieza de código.

## 6. Análisis por Categoría

### 6.1. Calidad de Código

El análisis general de calidad de código arroja una calificación de Excelente con 91 % de cumplimiento.

Categoría	Estado	Notas
Sintaxis	100 %	Sin errores de compilación
Variables No Usadas	96 %	2 warnings menores
React Hooks	94 %	2 warnings de dependencias
Imports/Exports	100 %	Todos válidos
Patrones de Diseño	100 %	Correctamente implementados

Cuadro 5: Evaluación de calidad de código

### 6.2. Seguridad

La evaluación de seguridad obtiene una calificación de Buena con recomendaciones, con 75 % de cumplimiento.

Vulnerabilidad	Estado	Acción Requerida
React Router XSS	Pendiente	Actualizar a v7+
esbuild CORS	Mitigado	Solo desarrollo
Datos Sensibles	OK	Sin credenciales en código
SQL Injection	N/A	Usa MongoDB (NoSQL)

Cuadro 6: Evaluación de seguridad

### 6.3. Arquitectura

La arquitectura del sistema obtiene una calificación de Buena con 85 % de cumplimiento.

Componente	Estado	Observaciones
Backend API	Excelente	Express + MongoDB
Frontend Controllers	Parcial	1 método sin migrar
Modelos	Perfecto	Validaciones completas
Vistas React	Muy bueno	Hooks correctos
Separación de Concerns	Excelente	MVC bien implementado

Cuadro 7: Evaluación de arquitectura

## 7. Plan de Implementación de Pruebas

Para alcanzar una cobertura completa del sistema, se propone un plan de implementación dividido en 5 fases con una duración estimada total de 50 horas.

## 7.1. Fase 1: Configuración e Infraestructura

**Duración Estimada:** 2 horas

**Prioridad:** Alta

1. Configurar Vitest y Testing Library (1 hora)
  - Instalar dependencias necesarias
  - Crear archivo de configuración vitest.config.js
  - Crear archivo de setup src/test/setup.js
  - Configurar scripts en package.json
2. Crear estructura de archivos de prueba (30 minutos)
  - Crear carpetas \_\_tests\_\_ en cada módulo
  - Crear archivos .test.js y .test.jsx
3. Configurar mocks y utilidades (30 minutos)
  - Mock de API client
  - Mock de localStorage
  - Utilidades de testing (factories, fixtures)

## 7.2. Fase 2: Pruebas de Modelos

**Duración Estimada:** 8 horas

**Prioridad:** Alta

1. Implementar pruebas de StaffMemberModel (1.5 horas)
  - 8 casos de prueba
  - Validaciones: cédula, correo, monto
2. Implementar pruebas de PagoExcepcionalModel (1.5 horas)
  - 8 casos de prueba
  - Validaciones: pagos manuales, conceptos
3. Implementar pruebas de GastoOperativoModel (2 horas)
  - 9 casos de prueba
  - Validaciones: comprobantes, proveedores

### 7.3. Fase 3: Pruebas de Controladores

**Duración Estimada:** 16 horas

**Prioridad:** Alta

Los controladores contienen la lógica de negocio crítica del sistema, por lo que sus pruebas son de alta prioridad:

1. StaffController (3 horas) - 12 casos de prueba
2. ComprobanteController (4 horas) - 15 casos de prueba
3. PagoExcepcionalController (2 horas) - 8 casos de prueba
4. GastoOperativoController (2.5 horas) - 10 casos de prueba
5. BusquedaController (2 horas) - 7 casos de prueba
6. GlobalControllers (1 hora) - 3 casos de prueba

### 7.4. Fase 4: Pruebas de Componentes React

**Duración Estimada:** 20 horas

**Prioridad:** Media

Esta fase incluye las pruebas de interfaz de usuario y componentes visuales:

1. Login.jsx (2 horas) - 6 casos de prueba (RF01)
2. ImportarExcel.jsx (2.5 horas) - 5 casos de prueba (RF02)
3. RegistrarComprobante.jsx (3 horas) - 8 casos de prueba (RF03)
4. ValidarComprobantes.jsx (3 horas) - 7 casos de prueba (RF05)
5. RegistrarPagoExcepcional.jsx (2.5 horas) - 6 casos de prueba (RF10)
6. RegistrarGastoOperativo.jsx (3 horas) - 7 casos de prueba (RF11)
7. BusquedaRegistros.jsx (3.5 horas) - 8 casos de prueba (RF12)
8. App.jsx (1.5 horas) - 4 casos de prueba

### 7.5. Fase 5: Pruebas de Patrones y Utilidades

**Duración Estimada:** 4 horas

**Prioridad:** Baja

Esta fase final incluye las pruebas de los patrones de diseño implementados y utilidades del sistema:

1. NotificationSystem (1.5 horas) - 5 casos de prueba
  - Patrón Observer
  - Suscripción y notificaciones
2. SessionManager (1 hora) - 4 casos de prueba

- Patrón Singleton
  - Instancia única y gestión de sesión
3. UserFactory (1 hora) - 3 casos de prueba
- Patrón Factory
  - Creación de usuarios por rol
4. API client (1.5 horas) - 6 casos de prueba
- Todas las rutas API
  - Manejo de errores y respuestas

## 8. Métricas de Calidad

### 8.1. Cumplimiento de Estándares

El análisis mediante ESLint y npm audit proporciona las siguientes métricas de calidad:

Métrica	Valor
<b>Análisis ESLint</b>	
Archivos Analizados	33
Errores Críticos	0
Warnings	4
Tasa de Éxito	87.8 %
Calificación	A-
<b>Auditoría npm</b>	
Paquetes Totales	347
Vulnerabilidades Críticas	0
Vulnerabilidades Altas	2
Vulnerabilidades Moderadas	2
Calificación	B+
<b>Puntuación Global</b>	<b>85/100</b>
<b>Evaluación</b>	<b>MUY BUENO</b>

Cuadro 8: Métricas de cumplimiento de estándares

## 8.2. Cobertura de Requisitos Funcionales

RF	Descripción	Análisis	Estado
RF01	Login	Completo	Sin errores
RF02	Importar Excel	Completo	1 warning menor
RF03	Registrar Comprobante	Completo	Sin errores
RF04	Subir Documento	Completo	Sin errores
RF05	Validar Comprobante	Parcial	Método sin migrar
RF10	Pagos Excepcionales	Completo	1 warning menor
RF11	Gastos Operativos	Completo	2 warnings menores
RF12	Búsqueda	Completo	Sin errores

Cuadro 9: Estado del código por requisito funcional

## 9. Conclusiones y Recomendaciones

### 9.1. Conclusiones

Basándose en el análisis exhaustivo realizado, se presentan las siguientes conclusiones:

- Cobertura de Pruebas:** El sistema cuenta actualmente con 21 pruebas unitarias implementadas, alcanzando una cobertura del 90.19 % en los módulos probados. Sin embargo, esta cobertura está limitada a únicamente 2 de los 5 modelos del sistema, dejando sin probar componentes críticos como controladores y vistas.
- Riesgo de Regresión:** La ausencia de pruebas automatizadas en controladores y componentes React representa un riesgo significativo. Cualquier modificación en estos módulos puede introducir errores sin ser detectados durante el desarrollo.
- Alcance Requerido:** Para alcanzar una cobertura integral del sistema se requieren aproximadamente 96 pruebas unitarias adicionales, distribuidas entre los módulos faltantes.
- Tiempo de Implementación:** Se estima un tiempo total de 50 horas para implementar el conjunto completo de pruebas necesarias, distribuidas en las 5 fases propuestas en este informe.
- Priorización de Esfuerzos:** Los modelos y controladores deben ser la prioridad alta en la implementación de pruebas, ya que contienen la lógica de negocio crítica del sistema.
- Calidad del Código:** El sistema presenta una alta calidad general de código (91 %), con solo 4 advertencias menores que no afectan significativamente la funcionalidad.
- Vulnerabilidades de Seguridad:** Se identificaron 4 vulnerabilidades, 2 de alta severidad y 2 de severidad moderada, que requieren atención antes del despliegue en producción.

## 9.2. Recomendaciones Prioritarias

### 9.2.1. Acciones Obligatorias Antes de Producción

Las siguientes acciones deben completarse obligatoriamente antes de desplegar el sistema en ambiente de producción:

1. **Actualizar React Router:** Actualizar a la versión 7.x o superior para corregir la vulnerabilidad XSS identificada (VULN-001). Esta vulnerabilidad tiene un CVSS score de 8.0/10 y puede comprometer sesiones de usuario.
2. **Migrar método validarContraDatosOficiales:** El método en ComprobanteController debe migrarse para utilizar la API REST del backend en lugar de acceso directo a MongoDB (ISSUE-001). Esto evitará errores en el navegador al utilizar la funcionalidad RF05.
3. **Eliminar métodos duplicados:** Remover las definiciones duplicadas en ComprobanteController (ISSUE-002) para evitar comportamiento inconsistente del sistema.

### 9.2.2. Mejoras de Calidad Recomendadas

Se recomienda implementar las siguientes mejoras para aumentar la calidad del código:

1. **Corregir warnings de React Hooks:** Resolver las dependencias faltantes en los hooks useEffect (WARNING-002 y WARNING-004) para cumplir con las mejores prácticas de React y evitar posibles problemas de closures obsoletos.
2. **Limpiar código no utilizado:** Eliminar variables y estados que no se utilizan en el código (WARNING-001 y WARNING-003) para mejorar la legibilidad y reducir el consumo de memoria.
3. **Evaluar actualización de Vite:** Considerar la actualización de Vite a la versión 7.x, teniendo en cuenta que es un cambio mayor que puede requerir ajustes en la configuración del proyecto.

### 9.2.3. Mejoras de Mejora Continua (Opcional)

Para fortalecer el proceso de desarrollo a largo plazo, se sugieren las siguientes implementaciones:

1. **Implementar pre-commit hooks:** Configurar hooks de Git para validación automática del código antes de cada commit, utilizando herramientas como Husky y lint-staged.
2. **Completar suite de pruebas unitarias:** Implementar las 96 pruebas adicionales identificadas en el plan de implementación, siguiendo las 5 fases propuestas.
3. **Configurar CI/CD:** Establecer un pipeline de integración y despliegue continuo que incluya validación ESLint automática y ejecución de pruebas.
4. **Implementar Dependabot:** Configurar actualizaciones automáticas de seguridad para las dependencias del proyecto mediante GitHub Dependabot o herramientas similares.

### 9.3. Aprobación para Producción

**Estado Actual:** CONDICIONAL

El sistema puede ser considerado para paso a producción únicamente después de completar las siguientes acciones:

1. Actualizar React Router para corregir VULN-001
2. Migrar método validarContraDatosOficiales (ISSUE-001)
3. Realizar pruebas funcionales completas de los requisitos RF01 a RF12
4. Validar que todas las funcionalidades críticas operan correctamente después de las correcciones

Aspecto	Evaluación
Riesgo Actual	Moderado
Riesgo Post-Corrección	Bajo
Estado de Aprobación	Condicional
Tiempo Estimado de Corrección	4-6 horas

Cuadro 10: Evaluación para aprobación en producción

## 10. Anexos

### 10.1. Anexo A: Comandos para Correcciones

A continuación se presentan los comandos necesarios para aplicar las correcciones identificadas:

```
# 1. Actualizar React Router
npm install react-router-dom@latest
npm audit

# 2. Re-ejecutar ESLint despues de correcciones
npx eslint . --ext .js,.jsx

# 3. Verificar correcciones en ambiente de desarrollo
npm run dev
npm run server

# 4. Prueba final de auditoria
npm audit --production
```

### 10.2. Anexo B: Archivos que Requieren Atención

Los siguientes archivos requieren atención inmediata según su nivel de prioridad:

**Prioridad Alta:**

- src/controllers/ComprobanteController.js (ISSUE-001, ISSUE-002)
- package.json (VULN-001)

**Prioridad Media:**

- src/views/Contadora/RegistrarPagoExcepcional.jsx (WARNING-002)
- src/views/Shared/RegistrarGastoOperativo.jsx (WARNING-003, WARNING-004)

**Prioridad Baja:**

- src/controllers/StaffController.js (WARNING-001)

**10.3. Anexo C: Métricas por Directorio**

Directorio	Archivos	Errores	Warnings	Estado
src/controllers/	6	0	1	Excelente
src/models/	5	0	0	Perfecto
src/views/	8	0	3	Muy bueno
src/patterns/	3	0	0	Perfecto
src/config/	4	0	0	Perfecto
root/	7	0	0	Perfecto

Cuadro 11: Métricas de calidad por directorio

**10.4. Anexo D: Resumen de Hallazgos****Puntos Positivos:**

- Arquitectura sólida con separación clara entre backend y frontend
- Código compilable y funcional sin errores críticos
- Patrones de diseño correctamente implementados (Singleton, Factory, Observer)
- API REST completa con más de 20 endpoints documentados
- Modelos con validaciones robustas garantizando consistencia de datos
- 91 % de archivos sin problemas, indicando alta calidad general

**Áreas de Mejora:**

- Completar migración a API REST (1 método pendiente)
- Actualizar dependencias con vulnerabilidades de seguridad
- Corregir warnings de React Hooks para seguir mejores prácticas
- Limpiar código duplicado en controladores
- Implementar pruebas unitarias en módulos faltantes

## 10.5. Anexo E: Próximos Pasos

Se recomienda seguir el siguiente cronograma de actividades:

1. **Semana 1:** Aplicar correcciones de alta prioridad (2-4 horas)
  - Actualizar React Router
  - Migrar método validarContraDatosOficiales
  - Eliminar métodos duplicados
2. **Semana 2:** Re-ejecutar análisis y validación (2-3 horas)
  - Ejecutar ESLint y npm audit nuevamente
  - Verificar que las correcciones se aplicaron correctamente
  - Documentar cambios realizados
3. **Semana 3:** Pruebas funcionales completas (8-10 horas)
  - Probar requisitos RF01 a RF12
  - Validar flujos de usuario completos
  - Identificar y corregir errores encontrados
4. **Semana 4:** Despliegue a ambiente de staging (4 horas)
  - Configurar ambiente de staging
  - Desplegar aplicación
  - Realizar pruebas de aceptación
5. **Semana 5:** Aprobación final y producción (2 horas)
  - Revisión final de seguridad
  - Aprobación de stakeholders
  - Despliegue a producción

## 11. Referencias

- ESLint Documentation. (2024). Getting Started with ESLint. <https://eslint.org/docs/latest/use/getting-started>
- Vitest Documentation. (2024). Getting Started. <https://vitest.dev/guide/>
- React Testing Library Documentation. (2024). Introduction. <https://testing-library.com/docs/react-testing-library/intro/>
- OWASP Foundation. (2024). Cross-Site Scripting (XSS). <https://owasp.org/www-community/attacks/xss/>
- National Vulnerability Database. (2024). CVSS v3.1 Specification. <https://nvd.nist.gov/vuln-metrics/cvss>

- npm Documentation. (2024). npm audit. <https://docs.npmjs.com/cli/v10/commands/npm-audit>

**Documento generado:** 21 de enero de 2026

**Última actualización:** 21 de enero de 2026

**Versión:** 1.0

**Sistema de análisis:** ESLint + npm audit + Vitest

---

Firma del Equipo de Desarrollo

Fecha: \_\_\_\_\_