PLAN DE PRUEBAS UNITARIAS - Sistema de Gestión de Comprobantes

Información del Proyecto

- **Nombre:** Sistema de Gestión y Validación de Comprobantes para Eventos
- **Integrantes:** Samir Mideros, Alison Miranda, David Moran, Gabriel Vivanco
- **Versión:** 1.0.0
- **Fecha:** 21 de Enero, 2026
- **Responsable:** Equipo de Desarrollo
- **Framework de Pruebas:** Vitest + React Testing Library
- **Cobertura Objetivo:** 80% mínimo

---

1. OBJETIVOS DE LAS PRUEBAS

1.1 Objetivo General

Verificar mediante pruebas unitarias automatizadas que todos los componentes, controladores y modelos del sistema funcionan correctamente de forma aislada y cumplen con los requisitos funcionales (RF01-RF12).

1.2 Objetivos Específicos

- Validar la lógica de negocio en controladores
- Verificar validaciones de modelos
- Probar componentes React de forma aislada
- Validar integración con API backend
- Asegurar manejo correcto de errores
- Verificar estados y efectos de React Hooks
- Alcanzar cobertura mínima del 80%

---

2. ALCANCE DE LAS PRUEBAS

2.1 Componentes a Probar

Backend (Node.js + Express)

- server.js - Servidor Express con MongoDB
- Rutas API REST (20+ endpoints)
- Conexión a MongoDB Atlas

- Middleware CORS

Frontend (React + Vite)

- Controladores (8 archivos)
- Modelos (5 archivos)
- Vistas/Componentes (8+ archivos)
- Configuración API
- Patrones de diseño (Observer, Singleton)

Configuración

- Archivos de configuración (package.json, vite.config.js)
- ESLint configuration
- Path aliases (@/)

---

3. TIPOS DE PRUEBAS UNITARIAS

3.1 Pruebas de Modelos

3.1.1 Validaciones de Datos

Objetivo: Verificar que las validaciones de campos funcionen correctamente

Modelos a probar:

- `StaffMemberModel.js`
- `ComprobanteModel.js`
- `PagoExcepcionalModel.js`
- `GastoOperativoModel.js`
- `UserModel.js`

Casos de prueba por modelo:

```
// Ejemplo: ComprobanteModel.test.js
describe('ComprobanteModel', () => {
  test('debe validar campos requeridos', () => {
    const comprobante = new ComprobanteModel({});
    const result = comprobante.validate();
    expect(result.isValid).toBe(false);
    expect(result.errors).toContain('staffCedula es requerido');
  });

  test('debe aceptar datos válidos', () => {
    const comprobante = new ComprobanteModel({
      staffCedula: '1234567890',
      staffNombre: 'Juan Pérez',
      monto: 100,
```

```
      proveedor: 'Proveedor X'
    });
    expect(comprobante.validate().isValid).toBe(true);
  });
});
```

Criterios de aceptación:

- Validaciones de campos requeridos funcionan
- Validaciones de formato (email, cédula, etc.)
- Validaciones de rangos (montos positivos, fechas)

3.2 Pruebas de Controladores

3.2.1 Lógica de Negocio

Objetivo: Verificar que los controladores ejecuten correctamente la lógica de negocio

Controladores a probar:

- `StaffController.js` (RF02)
- `ComprobanteController.js` (RF03, RF04, RF05)
- `PagoExcepcionalController.js` (RF10)
- `GastoOperativoController.js` (RF11)
- `BusquedaController.js` (RF12)

Casos de prueba ejemplo:

```
// StaffController.test.js
describe('StaffController', () => {
  let controller;

  beforeEach(() => {
    controller = new StaffController();
  });

  test('debe importar Excel correctamente', async () => {
    const mockFile = new File(['content'], 'staff.xlsx');
    const result = await controller.importarExcel(mockFile);

    expect(result.success).toBe(true);
    expect(result.data.registrosProcesados).toBeGreaterThan(0);
  });

  test('debe rechazar archivos con formato inválido', async () => {
    const mockFile = new File(['content'], 'staff.txt');
    const result = await controller.importarExcel(mockFile);
```

```
    expect(result.success).toBe(false);
    expect(result.message).toContain('formato');
  });
});
```

Criterios de aceptación:

- Operaciones CRUD funcionan correctamente
- Validaciones de datos antes de enviar a API
- Manejo de errores de red
- Notificaciones se disparan correctamente

3.3 Pruebas de Componentes React

3.3.1 Renderizado y UI

Objetivo: Verificar que los componentes rendericen correctamente

Componentes a probar:

- `Login.jsx` (RF01)
- `ImportarExcel.jsx` (RF02)
- `RegistrarComprobante.jsx` (RF03)
- `ValidarComprobantes.jsx` (RF05)
- `RegistrarPagoExcepcional.jsx` (RF10)
- `RegistrarGastoOperativo.jsx` (RF11)
- `BusquedaRegistros.jsx` (RF12)

Casos de prueba ejemplo:

```
// Login.test.jsx
import { render, screen, fireEvent } from '@testing-library/react';
import { BrowserRouter } from 'react-router-dom';
import Login from './Login';

describe('Login Component', () => {
  test('debe renderizar formulario de login', () => {
    render(<BrowserRouter><Login /></BrowserRouter>);

    expect(screen.getByLabelText(/usuario/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/contraseña/i)).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /iniciar/i })).toBeInTheDocument()
  });

  test('debe mostrar error con credenciales inválidas', async () => {
    render(<BrowserRouter><Login /></BrowserRouter>);
```

```
    fireEvent.change(screen.getByLabelText(/usuario/i), { target: { value: 'inva
    fireEvent.change(screen.getByLabelText(/contraseña/i), { target: { value: 'v
    fireEvent.click(screen.getByRole('button', { name: /iniciar/i }));

    expect(await screen.findByText(/credenciales inválidas/i)).toBeInTheDocument
  });

  test('debe navegar al dashboard con credenciales correctas', async () => {
    const mockNavigate = jest.fn();
    jest.mock('react-router-dom', () => ({
      ...jest.requireActual('react-router-dom'),
      useNavigate: () => mockNavigate
    }));

    render(<BrowserRouter><Login /></BrowserRouter>);

    fireEvent.change(screen.getByLabelText(/usuario/i), { target: { value: 'admi
    fireEvent.change(screen.getByLabelText(/contraseña/i), { target: { value: 'a
    fireEvent.click(screen.getByRole('button', { name: /iniciar/i }));

    await waitFor(() => expect(mockNavigate).toHaveBeenCalledWith('/dashboard'))
  });
});
```

Criterios de aceptación:

- Componentes renderizan sin errores
- Formularios manejan inputs correctamente
- Validaciones del lado del cliente funcionan
- Estados de carga se muestran correctamente
- Mensajes de error/éxito se despliegan

3.3.2 Hooks y Estado

Objetivo: Verificar que los hooks de React funcionen correctamente

Casos de prueba:

```
describe('useEffect hooks', () => {
  test('debe cargar datos al montar componente', async () => {
    const mockFetch = jest.fn().mockResolvedValue({ data: [] });

    render(<ComponenteConUseEffect fetch={mockFetch} />);

    await waitFor(() => expect(mockFetch).toHaveBeenCalledTimes(1));
```

```
  });
});
```

3.2 Pruebas de Seguridad en Dependencias

3.2.1 Auditoría de npm

Comando:

```
npm audit
npm audit --production
```

Criterios:

- 0 vulnerabilidades críticas
- Máximo 5 vulnerabilidades altas (revisar y justificar)
- Actualizar dependencias vulnerables cuando sea posible

Paquetes críticos a revisar:

- express
- mongodb
- cors
- react
- vite

---

4. CASOS DE PRUEBA POR REQUISITO FUNCIONAL

RF01: Login y Autenticación

Archivo de prueba: `src/views/Login/Login.test.jsx`

Casos de prueba:

```
describe('RF01 - Login y Autenticación', () => {
  test('UT-RF01-001: Debe renderizar formulario de login', () => {
    // Verificar que los campos usuario y contraseña existen
  });

  test('UT-RF01-002: Debe validar campos vacíos', async () => {
    // Intentar login sin datos, debe mostrar error
  });

  test('UT-RF01-003: Debe rechazar credenciales inválidas', async () => {
    // Login con credenciales incorrectas
    // Verificar mensaje de error
  });
```

```
  test('UT-RF01-004: Debe autenticar staff correctamente', async () => {
    // Login como staff
    // Verificar navegación a /registrar-comprobante
  });

  test('UT-RF01-005: Debe autenticar contadora correctamente', async () => {
    // Login como contadora
    // Verificar navegación a /pago-excepcional
  });

  test('UT-RF01-006: Debe guardar sesión en localStorage', async () => {
    // Login exitoso
    // Verificar que currentUser esté en localStorage
  });
});
```

Criterios de éxito:

- 6/6 pruebas pasan
- Cobertura > 80% en Login.jsx
- Todos los roles autentican correctamente

RF02: Importación de Excel

Archivos de prueba:

- `src/controllers/StaffController.test.js`
- `src/views/JefeTicketera/ImportarExcel.test.jsx`

Casos de prueba:

```
describe('RF02 - Importación de Excel del Staff', () => {
  describe('StaffController', () => {
    test('UT-RF02-001: Debe rechazar archivo sin seleccionar', async () => {
      const result = await controller.importarExcel(null);
      expect(result.success).toBe(false);
    });

    test('UT-RF02-002: Debe rechazar formato inválido', async () => {
      const mockFile = new File([''], 'staff.txt');
      const result = await controller.importarExcel(mockFile);
      expect(result.message).toContain('formato');
    });

    test('UT-RF02-003: Debe validar estructura de Excel', async () => {
      // Excel sin columnas requeridas
      const result = await controller.importarExcel(mockInvalidExcel);
```

```
      expect(result.errors).toContain('Columnas faltantes');
    });

    test('UT-RF02-004: Debe importar registros válidos', async () => {
      const result = await controller.importarExcel(mockValidExcel);
      expect(result.success).toBe(true);
      expect(result.data.registrosProcesados).toBeGreaterThan(0);
    });

    test('UT-RF02-005: Debe validar datos de cada registro', async () => {
      // Excel con registros inválidos
      const result = await controller.importarExcel(mockPartialExcel);
      expect(result.data.registrosConError).toBeGreaterThan(0);
    });

    test('UT-RF02-006: Debe llamar a API para guardar', async () => {
      const apiSpy = jest.spyOn(api, 'importStaff');
      await controller.importarExcel(mockValidExcel);
      expect(apiSpy).toHaveBeenCalled();
    });
  });

  describe('ImportarExcel Component', () => {
    test('UT-RF02-007: Debe mostrar zona de drop para archivo', () => {
      render(<ImportarExcel />);
      expect(screen.getByText(/arrastre.*archivo/i)).toBeInTheDocument();
    });

    test('UT-RF02-008: Debe procesar archivo al seleccionar', async () => {
      render(<ImportarExcel />);
      const input = screen.getByLabelText(/seleccionar archivo/i);
      fireEvent.change(input, { target: { files: [mockFile] } });
      await waitFor(() => expect(screen.getByText(/procesando/i)).toBeInTheDocur
    });
  });
});
```

Criterios de éxito:

- 8/8 pruebas pasan
- Cobertura > 85% en StaffController
- Validaciones de Excel funcionan correctamente

RF03: Registro de Comprobantes

Pruebas ESLint:

- ☐ Validar `src/controllers/ComprobanteController.js`
- ☐ Validar `src/views/Staff/RegistrarComprobante.jsx`
- ☐ Verificar imports de modelos y API
- ☐ Validar formularios controlados

Criterios de éxito:

- Estados de formulario correctos
- Validaciones de campos implementadas
- API calls correctas

RF04: Subida de Documentos

Pruebas ESLint:

- ☐ Validar manejo de archivos (File API)
- ☐ Verificar validación de formatos
- ☐ Validar estados de carga (loading, error, success)

Criterios de éxito:

- FileReader usado correctamente
- Validación de tipos MIME
- Manejo de errores

RF05: Validación de Comprobantes

Pruebas ESLint:

- ☐ Validar `src/views/Validacion/ValidarComprobantes.jsx`
- ☐ Verificar lógica de validación
- ☐ Validar integración con StaffController

Criterios de éxito:

- Lógica de validación sin errores
- Estados de validación correctos
- Notificaciones implementadas

RF10: Pagos Excepcionales

Pruebas ESLint:

- ☐ Validar `src/controllers/PagoExcepcionalController.js`
- ☐ Validar `src/models/PagoExcepcionalModel.js`
- ☐ Validar `src/views/Contadora/RegistrarPagoExcepcional.jsx`

Criterios de éxito:

- Modelo validado correctamente
- Controller usa API correctamente

- Vista sin errores de React

RF11: Gastos Operativos

Pruebas ESLint:

- ☐ Validar `src/controllers/GastoOperativoController.js`
- ☐ Validar `src/models/GastoOperativoModel.js`
- ☐ Validar `src/views/Shared/RegistrarGastoOperativo.jsx`

Criterios de éxito:

- CRUD completo sin errores
- Validaciones de campos
- Integración con backend

RF12: Búsqueda y Filtrado

Pruebas ESLint:

- ☐ Validar `src/controllers/BusquedaController.js`
- ☐ Validar `src/views/Shared/BusquedaRegistros.jsx`
- ☐ Verificar lógica de filtros

Criterios de éxito:

- Query builders sin errores
- Filtros dinámicos correctos
- Renderizado de resultados

---

5. PROCEDIMIENTO DE EJECUCIÓN

5.1 Instalación de Framework de Pruebas

```
# 1. Instalar Vitest y dependencias
npm install --save-dev vitest @vitest/ui jsdom

# 2. Instalar React Testing Library
npm install --save-dev @testing-library/react @testing-library/jest-dom @testing

# 3. Verificar instalación
npx vitest --version
```

5.2 Configuración de Vitest

Crear `vitest.config.js`:

```
import { defineConfig } from 'vitest/config';
import react from '@vitejs/plugin-react';
import path from 'path';

export default defineConfig({
  plugins: [react()],
  test: {
    globals: true,
    environment: 'jsdom',
    setupFiles: './src/test/setup.js',
    coverage: {
      provider: 'v8',
      reporter: ['text', 'json', 'html'],
      exclude: [
        'node_modules/',
        'src/test/',
        '*.config.js'
      ],
      statements: 80,
      branches: 80,
      functions: 80,
      lines: 80
    }
  },
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src')
    }
  }
});
```

Crear `src/test/setup.js`:

```
import { expect, afterEach } from 'vitest';
import { cleanup } from '@testing-library/react';
import '@testing-library/jest-dom';

afterEach(() => {
  cleanup();
});
```

5.3 Ejecución de Pruebas Unitarias

Ejecutar Todas las Pruebas

```
# Ejecutar todas las pruebas
npm test

# Ejecutar con UI
npm test -- --ui

# Ejecutar con cobertura
npm test -- --coverage

# Modo watch (desarrollo)
npm test -- --watch
```

Pruebas por Categoría

```
# Solo modelos
npm test -- src/models

# Solo controladores
npm test -- src/controllers

# Solo componentes
npm test -- src/views

# Solo patrones
npm test -- src/patterns
```

Prueba Específica

```
# Ejecutar un archivo de prueba específico
npm test -- StaffController.test.js

# Ejecutar pruebas que coincidan con un patrón
npm test -- --grep "validación"
```

5.3 Análisis de Dependencias

```
# Auditoría de seguridad
npm audit

# Auditoría solo de dependencias de producción
npm audit --production

# Ver detalles de vulnerabilidades
npm audit --json
```

```
# Listar dependencias obsoletas
npm outdated
```

5.4 Verificación de Servidores

Backend

```
# Iniciar backend
npm run server

# Verificar en otra terminal
curl http://localhost:5000/api/staff
```

Frontend

```
# Iniciar frontend
npm run dev

# Abrir navegador en http://localhost:3000
# Verificar consola del navegador (F12)
```

---

6. CRITERIOS DE ACEPTACIÓN

6.1 Criterios Generales de Pruebas Unitarias

- **Cobertura de Código:** Mínimo 80% en todos los módulos
- **Tasa de Éxito:** 100% de pruebas pasando
- **Tiempo de Ejecución:** Todas las pruebas < 30 segundos
- **Sin Tests Flaky:** 0 pruebas intermitentes
- **Assertions:** Mínimo 3 assertions por prueba
- **Mocks:** API mockeada correctamente en todas las pruebas
- **Cleanup:** Sin memory leaks entre pruebas

6.2 Criterios por Componente

Backend (server.js)

- Sin errores de sintaxis
- Todas las rutas API definidas
- Middleware configurado correctamente
- Manejo de errores implementado

Controladores

- Imports correctos (API client)

- Métodos asíncronos con async/await
- Manejo de errores try/catch
- No usar dbConnection directamente

Modelos

- Validaciones implementadas
- Métodos toJSON() definidos
- Sin variables no usadas

Vistas React

- Hooks usados correctamente
- Props validadas
- Estados inicializados
- Event handlers definidos
- No hay memory leaks (cleanup en useEffect)

---

7. REGISTRO DE DEFECTOS

Los defectos encontrados durante las pruebas serán registrados en el archivo `INFORME_ERRORES.md` con la siguiente estructura:

```
### ERROR-XXX: [Título descriptivo]
- **Severidad:** Crítico / Alto / Medio / Bajo
- **Tipo:** Sintaxis / Lógica / Seguridad / Performance
- **Archivo:** ruta/al/archivo.js:línea
- **Descripción:** Detalle del error
- **Regla ESLint:** nombre-de-regla
- **Solución propuesta:** Cómo corregirlo
```

---

8. HERRAMIENTAS UTILIZADAS

8.1 Análisis de Código

- **ESLint** v9.x - Análisis estático de JavaScript/JSX
- **eslint-plugin-react** - Reglas específicas de React
- **eslint-plugin-react-hooks** - Validación de React Hooks
- **@eslint/js** - Configuración base de ESLint

8.2 Dependencias

- **npm audit** - Análisis de seguridad
- **npm outdated** - Detección de paquetes desactualizados

8.3 Navegador

- **DevTools Console** - Errores en runtime
- **Network Tab** - Verificar llamadas API
- **React DevTools** - Inspeccionar componentes

---

9. CRONOGRAMA DE PRUEBAS

Fase	Actividad	Duración Estimada	Estado
1	Configuración de ESLint	15 min	Completado
2	Ejecución de ESLint completo	5 min	Pendiente
3	Análisis de resultados	30 min	Pendiente
4	Generación de informe	15 min	Pendiente
5	Auditoría npm	10 min	Pendiente
6	Pruebas funcionales manuales	1 hora	Pendiente
7	Corrección de errores críticos	2 horas	Pendiente
8	Re-validación	30 min	Pendiente

Tiempo Total Estimado: 4.5 horas

---

10. RESPONSABILIDADES

Rol	Responsable	Responsabilidad
Ejecutor de Pruebas	Equipo QA	Ejecutar ESLint y análisis
Analista de Resultados	Lead Developer	Revisar y clasificar errores
Corrector	Developers	Implementar correcciones
Validador	Tech Lead	Aprobar cambios

---

11. ENTREGABLES

Documentos

- `PLAN_DE_PRUEBAS.md` - Este documento
- `INFORME_ERRORES.md` - Reporte de errores encontrados
- `eslint-report.json` - Reporte técnico de ESLint
- `npm-audit.txt` - Resultado de npm audit

Código Corregido

- Archivos con errores críticos corregidos
- Pull Request con correcciones
- Documentación de cambios

---

12. NOTAS ADICIONALES

12.1 Exclusiones

No se realizarán:

- Pruebas con Postman (según requerimiento)
- Pruebas unitarias automatizadas (fuera de alcance)
- Pruebas de carga/performance
- Pruebas end-to-end automatizadas

12.2 Limitaciones Conocidas

- MongoDB driver requiere backend (no funciona en navegador)
- Algunos controladores tienen código legacy de dbConnection
- Path aliases (@/) pueden causar warnings en ESLint

12.3 Recomendaciones Post-Pruebas

1. Implementar pre-commit hooks con ESLint
2. Configurar CI/CD con validación automática
3. Agregar pruebas unitarias con Jest/Vitest
4. Implementar cobertura de código mínima del 70%

---

13. APROBACIONES

Nombre	Rol	Firma	Fecha
_____	Product Owner	_____	//___
_____	Tech Lead	_____	//___
_____	QA Lead	_____	//___

Documento creado: 21 de Enero, 2026
Última actualización: 21 de Enero, 2026
Versión: 1.0