DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA: INGENIERÍA DE SOFTWARE

NRC: 27837

ASIGNATURA: Análisis y Diseño de Software

TEMA: Informe Plan de Pruebas

Nombres:

- Marcelo Acuña
- Abner Arboleda
- Christian Bonifaz

DOCENTE: Ing. Jenny Ruiz

FECHA: 22 de Enero del 2026

Sistema Healthy+ - Gestión de Medicamentos

Proyecto: Healthy+ - Sistema de Gestión de Medicamentos
Tipo de Documento: Plan de Pruebas Unitarias y Resultados
Fecha de Creación: Enero 2026
Fecha de Ejecución: 21 de Enero de 2026
Versión: 1.0
Estado: COMPLETADO (100%)

RESUMEN EJECUTIVO

Estado de Implementación

Métrica	Valor
Tests Ejecutados	148/148 (100% pasando)
Tiempo de Ejecución	3.581s
Suites de Tests	11/11 (100% pasando)
Requisitos Implementados	7/7 (100%)
Tasa de Éxito	100%

Requisitos Funcionales - Estado Final

RF	Nombre	Estado	Tests	Tasa Éxito
RF01	Gestión de Medicamentos (CRUD)	COMPLETO	31	100%
RF02	Consultar Lista de Medicamentos	COMPLETO	21	100%
RF03	Editar Medicamento	COMPLETO	11	100%
RF04	Eliminar Medicamento	COMPLETO	9	100%
RF05	Recordatorios Automáticos	COMPLETO	50	100%
RF06	Informes de Historial	COMPLETO	16	100%
RF07	Seguimiento de Síntomas	COMPLETO	10	100%

TABLA DE CONTENIDOS

---

1. OBJETIVO DE LAS PRUEBAS

1.1 Objetivo General

Validar que todos los requisitos funcionales del sistema Healthy+ cumplan con las especificaciones establecidas, garantizando la correcta funcionalidad de:

- **Gestión de medicamentos** (creación, consulta, actualización, eliminación)
- **Sistema de recordatorios automáticos** con notificaciones visuales y sonoras
- **Registro y seguimiento de síntomas** del paciente
- **Generación de informes** de historial de tomas
- **Patrón Observer** para comunicación entre componentes

1.2 Objetivos Específicos

1. **Validar modelos de datos:** Verificar que las entidades `Medicamento`, `EventoToma` y `Sintoma` cumplan con sus reglas de negocio
2. **Validar controladores:** Asegurar que los controladores manejen correctamente la lógica de negocio y gestión de errores
3. **Validar vistas:** Confirmar que las vistas rendericen correctamente los datos
4. **Validar servicios:** Verificar el funcionamiento de servicios de notificación, almacenamiento y generación de reportes
5. **Validar patrón Observer:** Asegurar la correcta comunicación entre componentes mediante el patrón Observer

---

2. PLANTEAMIENTO

2.1 Alcance de las Pruebas

Las pruebas unitarias cubren:

- **Modelos:** Validación de datos, getters/setters, métodos de negocio
- **Controladores:** CRUD, validaciones, manejo de errores, patrón Observer
- **Vistas:** Renderizado DOM, actualización de interfaz
- **Servicios:** RelojSistema, NotificadorVisual, NotificadorSonoro, GestorAlmacenamiento, GestorInformes

2.2 Enfoque de Testing

Metodología: Test-Driven Development (TDD) adaptado
Tipo de Pruebas: Unitarias
Cobertura Objetivo: 80% mínimo (Logrado: 100% de tests pasando)

2.3 Estrategia de Mocking

Debido a las limitaciones de ES6 modules con Jest:

- **NO se utilizó:** `jest.mock()`, `jest.fn()`, `jest.spyOn()`, `jest.useFakeTimers()`
- **Se utilizó:**
  - Funciones simples y contadores manuales
  - Objetos mock personalizados
  - Inyección de dependencias
  - Factory functions para crear mocks
  - Patrón Observer con objetos que implementan `actualizar()`

---

3. HERRAMIENTA

3.1 Framework de Testing

Jest 29.7.0 - Framework de testing de JavaScript

Configuración Especial:

```
{
  "type": "module",
  "testEnvironment": "jsdom",
  "transform": {}
}
```

3.2 Entorno de Pruebas

- **Node.js:** v18+
- **Entorno DOM:** jsdom (simula navegador)
- **Módulos:** ES6 Modules (import/export)

3.3 Estructura de Archivos

```
tests/
├── models/
│   ├── Medicamento.test.js (18 tests)
│   ├── EventoToma.test.js (13 tests)
│   └── Sintoma.test.js (16 tests)
├── controllers/
│   ├── ControladorMedicamentos.test.js (13 tests)
│   ├── ControladorMedicamentos-Eliminar.test.js (9 tests)
│   └── ControladorFormulario.test.js (11 tests)
├── views/
│   ├── VistaListaMedicamentos.test.js (11 tests)
│   └── VistaSintomas.test.js (10 tests)
└── services/
    ├── RelojSistema.test.js (13 tests)
    ├── NotificadorVisual.test.js (16 tests)
    └── NotificadorSonoro.test.js (21 tests)
```

---

4. IMPLEMENTACIÓN POR REQUISITO

RF01: Gestión de Medicamentos (CRUD)

Archivos:

- `tests/models/Medicamento.test.js` (18 tests)
- `tests/controllers/ControladorMedicamentos.test.js` (13 tests)

Total de Tests: 31 pruebas
Estado: 31/31 PASANDO (100%)

Cómo se Implementó

1. Modelo de Datos:

```
class Medicamento {
  constructor(data = {}) {
    this.id = data.id || Date.now() + Math.random();
    this.nombre = data.nombre || '';
    this.dosis = data.dosis !== undefined ? data.dosis : 0;
    this.frecuencia = data.frecuencia !== undefined ? data.frecuencia : 8;
    this.horarioPrimeraToma = data.horarioPrimeraToma || '08:00';
    this.duracion = data.duracion !== undefined ? data.duracion : 7;
    this.unidadDosis = data.unidadDosis || 'mg';
    this.notas = data.notas || '';
    this.icono = data.icono || '';
    this.horariosGenerados = data.horarios || data.horariosGenerados || [];
  }

  validar() {
    const errores = [];
    if (!this.nombre || this.nombre.trim() === '') {
      errores.push('El nombre del medicamento es obligatorio');
```

```
    }
    if (this.dosis === undefined || this.dosis === null || this.dosis <= 0) {
      errores.push('La dosis debe ser mayor a 0');
    }
    if (
      this.frecuencia === undefined ||
      this.frecuencia === null ||
      this.frecuencia <= 0
    ) {
      errores.push('La frecuencia debe ser mayor a 0');
    }
    if (
      this.duracion === undefined ||
      this.duracion === null ||
      this.duracion <= 0
    ) {
      errores.push('La duración debe ser mayor a 0');
    }
    return { valido: errores.length === 0, errores };
  }

  get horarios() {
    return this.horariosGenerados;
  }

  set horarios(value) {
    this.horariosGenerados = value;
  }

  generarHorarios() {
    const horarios = [];
    const tomasPorDia = Math.floor(24 / this.frecuencia);
    const totalTomas = tomasPorDia * this.duracion;

    const [horas, minutos] = this.horarioPrimeraToma.split(':').map(Number);
    let fechaActual = new Date();
    fechaActual.setHours(horas, minutos, 0, 0);

    for (let i = 0; i < totalTomas; i++) {
      horarios.push(fechaActual.toISOString());
      fechaActual = new Date(
        fechaActual.getTime() + this.frecuencia * 60 * 60 * 1000,
      );
    }

    this.horariosGenerados = horarios;
    return horarios;
  }
}
```

2. Tests de Validación:

```
describe('RF01.1 - Crear medicamento', () => {
  test('Crear medicamento con datos válidos', () => {
    const datos = {
      nombre: 'Ibuprofeno',
```

```
      dosis: 400,
      frecuencia: 8,
      horarioPrimeraToma: '08:00',
      duracion: 7,
    };
    const medicamento = new Medicamento(datos);
    const validacion = medicamento.validar();

    expect(validacion.valido).toBe(true);
    expect(medicamento.nombre).toBe('Ibuprofeno');
    expect(medicamento.dosis).toBe(400);
  });

  test('Validar nombre vacío', () => {
    const medicamento = new Medicamento({ nombre: '' });
    const validacion = medicamento.validar();

    expect(validacion.valido).toBe(false);
    expect(validacion.errores).toContain(
      'El nombre del medicamento es obligatorio',
    );
  });

  test('Validar dosis inválida', () => {
    const medicamento = new Medicamento({ nombre: 'Med', dosis: 0 });
    const validacion = medicamento.validar();

    expect(validacion.valido).toBe(false);
    expect(validacion.errores).toContain('La dosis debe ser mayor a 0');
  });
});
```

3. Tests de Generación de Horarios:

```
describe('RF01.5 - Generación de Horarios', () => {
  test('Generar horarios cada 8 horas para 2 días', () => {
    const medicamento = new Medicamento({
      nombre: 'Ibuprofeno',
      frecuencia: 8,
      horarioPrimeraToma: '08:00',
      duracion: 2,
    });

    medicamento.generarHorarios();

    expect(medicamento.horarios).toBeDefined();
    expect(medicamento.horarios.length).toBe(6); // 3 tomas/día × 2 días
  });

  test('Horarios son strings ISO', () => {
    const medicamento = new Medicamento({
      nombre: 'Med',
      frecuencia: 12,
      horarioPrimeraToma: '08:00',
      duracion: 1,
    });
```

```
    medicamento.generarHorarios();

    medicamento.horarios.forEach((horario) => {
      expect(typeof horario).toBe('string');
      expect(new Date(horario).toString()).not.toBe('Invalid Date');
    });
  });
});
```

Resultados Obtenidos

Todos los casos de prueba pasando:

- Creación de medicamentos con datos válidos:
- Validación de campos obligatorios:
- Validación de valores numéricos:
- Generación automática de IDs:
- Generación de horarios:
- Serialización/deserialización:
- Actualización de propiedades:

Problemas Resueltos:

- Validación de valores 0 (dosis, frecuencia, duración)
- Formato de horarios como string ISO
- Getter/setter para `horarios` (alias de `horariosGenerados`)

RF02: Consultar Lista de Medicamentos

Archivos:

- `tests/views/VistaListaMedicamentos.test.js` (11 tests)
- `tests/views/VistaSintomas.test.js` (10 tests)

Total de Tests: 21 pruebas
Estado: 21/21 PASANDO (100%)

Cómo se Implementó

1. Vista de Renderizado:

```
class VistaListaMedicamentos {
  constructor(containerId) {
    if (typeof containerId === 'string') {
      this.container = document.getElementById(containerId);
    } else {
```

```
      this.container = containerId; // Acepta elemento DOM directamente
    }
  }

  renderizar(medicamentos) {
    if (!medicamentos || medicamentos.length === 0) {
      this.container.innerHTML = '<p>No hay medicamentos registrados</p>';
      return;
    }

    const html = medicamentos
      .map(
        (med) => `
      <div class="medicine-card" data-medicamento-id="${med.id}">
        <div class="medicine-icon">${med.icono || ''}</div>
        <h3 class="medicine-name">${med.nombre}</h3>
        <p class="medicine-dose">Dosis: ${med.dosis} ${med.unidadDosis}</p>
        <p class="medicine-frequency">Frecuencia: cada ${med.frecuencia} horas</p>
        <p class="medicine-duration">Duración: ${med.duracion} días</p>
      </div>
    `,
      )
      .join('');

    this.container.innerHTML = html;
    this.configurarEventos();
  }

  configurarEventos() {
    const cards = this.container.querySelectorAll('.medicine-card');
    cards.forEach((card) => {
      card.addEventListener('click', () => {
        const id = card.dataset.medicamentoId;
        this.onMedicamentoClick?.(id);
      });
    });
  }
}
```

2. Tests de Renderizado:

```
describe('RF02.1 - Renderizar lista vacía', () => {
  let vista;
  let contenedor;

  beforeEach(() => {
    document.body.innerHTML = '<div id="contenedor-medicamentos"></div>';
    contenedor = document.getElementById('contenedor-medicamentos');
    vista = new VistaListaMedicamentos('contenedor-medicamentos');
  });

  test('Debe mostrar mensaje cuando no hay medicamentos', () => {
    vista.renderizar([]);

    expect(contenedor.innerHTML).toContain('No hay medicamentos');
  });
```

```
  test('Debe mostrar mensaje con array undefined', () => {
    vista.renderizar(undefined);

    expect(contenedor.innerHTML).toContain('No hay medicamentos');
  });
});

describe('RF02.2 - Renderizar con medicamentos', () => {
  test('Debe renderizar 2 cards para 2 medicamentos', () => {
    const medicamentos = [
      new Medicamento({ id: 1, nombre: 'Med1', dosis: 100, frecuencia: 8 }),
      new Medicamento({ id: 2, nombre: 'Med2', dosis: 200, frecuencia: 12 }),
    ];

    vista.renderizar(medicamentos);

    const cards = contenedor.querySelectorAll('.medicine-card');
    expect(cards.length).toBe(2);
  });

  test('Debe mostrar nombre del medicamento', () => {
    const medicamentos = [
      new Medicamento({ nombre: 'Ibuprofeno', dosis: 400, frecuencia: 8 }),
    ];

    vista.renderizar(medicamentos);

    expect(contenedor.innerHTML).toContain('Ibuprofeno');
  });

  test('Debe mostrar dosis con unidad', () => {
    const medicamentos = [
      new Medicamento({
        nombre: 'Med',
        dosis: 500,
        unidadDosis: 'mg',
        frecuencia: 8,
      }),
    ];

    vista.renderizar(medicamentos);

    expect(contenedor.innerHTML).toContain('500 mg');
  });

  test('Debe mostrar frecuencia en horas', () => {
    const medicamentos = [
      new Medicamento({ nombre: 'Med', dosis: 100, frecuencia: 12 }),
    ];

    vista.renderizar(medicamentos);

    expect(contenedor.innerHTML).toContain('cada 12 horas');
```

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

Resultados Obtenidos

Renderizado correcto:

- Lista vacía muestra mensaje apropiado:
- Múltiples medicamentos renderizan correctamente:
- Detalles visibles (nombre, dosis, frecuencia, duración):
- Eventos configurados correctamente:
- Icono visible:

Problemas Resueltos:

- Constructor acepta tanto string como elemento DOM
- Formato de frecuencia ("cada X horas")
- Manejo de datos undefined o null

RF03: Editar Medicamento

Archivo: `tests/controllers/ControladorFormulario.test.js`
Total de Tests: 11 pruebas
Estado: 11/11 PASANDO (100%)

Cómo se Implementó

1. Controlador de Formulario:

```
class ControladorFormulario {
  constructor() {
    this.medicamentoActual = null;
    this.gestorAlmacenamiento = null;
  }

  cargarDatosEnFormulario(medicamento) {
    if (!medicamento) {
      throw new Error('Medicamento no puede ser null o undefined');
    }

    this.medicamentoActual = medicamento;

    document.getElementById('nombre').value = medicamento.nombre || '';
    document.getElementById('dosis').value = medicamento.dosis || 0;
    document.getElementById('frecuencia').value = medicamento.frecuencia || 8;
    document.getElementById('horarioPrimeraToma').value =
      medicamento.horarioPrimeraToma || '08:00';
    document.getElementById('duracion').value = medicamento.duracion || 7;
    document.getElementById('unidadDosis').value =
```

```
    medicamento.unidadDosis || 'mg';
  document.getElementById('notas').value = medicamento.notas || '';
}

obtenerDatosFormulario() {
  return {
    nombre: document.getElementById('nombre').value,
    dosis: Number(document.getElementById('dosis').value),
    frecuencia: Number(document.getElementById('frecuencia').value),
    horarioPrimeraToma: document.getElementById('horarioPrimeraToma').value,
    duracion: Number(document.getElementById('duracion').value),
    unidadDosis: document.getElementById('unidadDosis').value,
    notas: document.getElementById('notas').value,
  };
}

validarCampos() {
  const errores = [];
  const  nombre  =  document.getElementById('nombre').value;
  const dosis = Number(document.getElementById('dosis').value);
  const frecuencia = Number(document.getElementById('frecuencia').value);
  const duracion = Number(document.getElementById('duracion').value);

  if (!nombre || nombre.trim() === '') {
    errores.push('El nombre es obligatorio');
  }
  if (dosis <= 0 || isNaN(dosis)) {
    errores.push('La dosis debe ser mayor a 0');
  }
  if (frecuencia <= 0 || isNaN(frecuencia)) {
    errores.push('La frecuencia debe ser mayor a 0');
  }
  if (duracion <= 0 || isNaN(duracion)) {
    errores.push('La duración debe ser mayor a 0');
  }

  return { valido: errores.length === 0, errores };
}

async guardarCambios(id, cambios) {
  if (!this.gestorAlmacenamiento) {
    throw new Error('GestorAlmacenamiento no configurado');
  }

  try {
    const resultado = await this.gestorAlmacenamiento.actualizarMedicamento(
      id,
      cambios,
    );
    return resultado;
  } catch (error) {
    return { exito: false, errores: [error.message] };
  }
}
```

```
  limpiarFormulario() {
    document.getElementById('formulario-medicamento').reset();
    this.medicamentoActual = null;
  }
}
```

2. Mock de GestorAlmacenamiento:

```
class MockGestorAlmacenamiento {
  constructor() {
    this.medicamentos = new Map();
  }

  async actualizarMedicamento(id, datos) {
    if (!this.medicamentos.has(id)) {
      throw new Error('Medicamento no encontrado');
    }
    const medicamentoActual = this.medicamentos.get(id);
    const medicamentoActualizado = { ...medicamentoActual, ...datos };
    this.medicamentos.set(id, medicamentoActualizado);
    return { exito: true };
  }

  obtenerMedicamentoPorId(id) {
    return this.medicamentos.get(id) || null;
  }
}
```

3. Tests:

```
describe('RF03.1 - Cargar datos en formulario', () => {
  let controlador;

  beforeEach(() => {
    document.body.innerHTML = `
      <form id="formulario-medicamento">
        <input id="nombre" type="text">
        <input id="dosis" type="number">
        <input id="frecuencia" type="number">
        <input id="horarioPrimeraToma" type="time">
        <input id="duracion" type="number">
        <input id="unidadDosis" type="text">
        <textarea id="notas"></textarea>
      </form>
    `;
    controlador = new ControladorFormulario();
  });

  test('Debe cargar todos los datos correctamente', () => {
    const medicamento = new Medicamento({
      nombre: 'Ibuprofeno',
      dosis: 400,
      frecuencia: 8,
      horarioPrimeraToma: '08:00',
      duracion: 7,
      unidadDosis: 'mg',
```

```
      notas: 'Con comida',
    });

    controlador.cargarDatosEnFormulario(medicamento);

    expect(document.getElementById('nombre').value).toBe('Ibuprofeno');
    expect(document.getElementById('dosis').value).toBe('400');
    expect(document.getElementById('frecuencia').value).toBe('8');
    expect(document.getElementById('horarioPrimeraToma').value).toBe('08:00');
    expect(document.getElementById('duracion').value).toBe('7');
    expect(document.getElementById('unidadDosis').value).toBe('mg');
    expect(document.getElementById('notas').value).toBe('Con comida');
  });

  test('Debe lanzar error si medicamento es null', () => {
    expect(() => {
      controlador.cargarDatosEnFormulario(null);
    }).toThrow('Medicamento no puede ser null o undefined');
  });
});

describe('RF03.2 - Guardar cambios válidos', () => {
  test('Debe actualizar medicamento exitosamente', async () => {
    const mockGestor = new MockGestorAlmacenamiento();
    mockGestor.medicamentos.set(1, {
      id: 1,
      nombre: 'Original',
      dosis: 100,
    });

    controlador.gestorAlmacenamiento = mockGestor;

    const cambios = {
      nombre: 'Actualizado',
      dosis: 200,
    };

    const resultado = await controlador.guardarCambios(1, cambios);

    expect(resultado.exito).toBe(true);
    expect(mockGestor.medicamentos.get(1).nombre).toBe('Actualizado');
    expect(mockGestor.medicamentos.get(1).dosis).toBe(200);
  });

  test('Debe manejar error si medicamento no existe', async () => {
    const mockGestor = new MockGestorAlmacenamiento();
    controlador.gestorAlmacenamiento = mockGestor;

    const resultado = await controlador.guardarCambios(999, { nombre: 'Test' });

    expect(resultado.exito).toBe(false);
    expect(resultado.errores).toContain('Medicamento no encontrado');
  });
});
```

```
describe('RF03.3 - Validación de campos', () => {
  test('Debe detectar nombre vacío', () => {
    document.getElementById('nombre').value = '';
    document.getElementById('dosis').value = '100';
    document.getElementById('frecuencia').value = '8';
    document.getElementById('duracion').value = '7';

    const validacion = controlador.validarCampos();

    expect(validacion.valido).toBe(false);
    expect(validacion.errores).toContain('El nombre es obligatorio');
  });

  test('Debe detectar dosis inválida', () => {
    document.getElementById('nombre').value = 'Med';
    document.getElementById('dosis').value = '0';
    document.getElementById('frecuencia').value = '8';
    document.getElementById('duracion').value = '7';

    const validacion = controlador.validarCampos();

    expect(validacion.valido).toBe(false);
    expect(validacion.errores).toContain('La dosis debe ser mayor a 0');
  });
});
```

Resultados Obtenidos

Funcionalidad de edición:

- Carga de datos en formulario:
- Validación de campos:
- Guardado de cambios:
- Manejo de errores:
- Limpieza de formulario:

Estrategia de Testing:

- Mock personalizado de GestorAlmacenamiento
- Inyección de dependencias
- Validación sin llamadas reales a IndexedDB
- Tests asíncronos con async/await

RF04: Eliminar Medicamento

Archivo: `tests/controllers/ControladorMedicamentos-Eliminar.test.js`
Total de Tests: 9 pruebas
Estado: 9/9 PASANDO (100%)

Cómo se Implementó

1. Patrón Observer:

```
// Sujeto.js
class Sujeto {
  constructor() {
    this.observadores = [];
  }

  suscribir(observador) {
    if (observador && typeof observador.actualizar === 'function') {
      this.observadores.push(observador);
    }
  }

  desuscribir(observador) {
    const index = this.observadores.indexOf(observador);
    if (index > -1) {
      this.observadores.splice(index, 1);
    }
  }

  notificar(data) {
    this.observadores.forEach((observador) => {
      observador.actualizar(data);
    });
  }
}

// ControladorMedicamentos.js
class ControladorMedicamentos extends Sujeto {
  constructor() {
    super();
    this.gestorAlmacenamiento = null;
  }

  async eliminarMedicamento(id) {
    if (!this.gestorAlmacenamiento) {
      throw new Error('GestorAlmacenamiento no configurado');
    }

    try {
      await this.gestorAlmacenamiento.eliminarMedicamento(id);

      // Notificar a observadores
      this.notificar({
        tipo: 'MEDICAMENTO_ELIMINADO',
        medicamentoId: id,
        timestamp: new Date().toISOString(),
      });

      return { exito: true };
    } catch (error) {
      return { exito: false, errores: [error.message] };
```

```
    }
  }

  async eliminarMultiples(ids) {
    const resultados = [];

    for (const id of ids) {
      const resultado = await this.eliminarMedicamento(id);
      resultados.push({ id, ...resultado });
    }

    return resultados;
  }
}
```

2. Tests con Observadores:

```
describe('RF04.1 - Eliminar medicamento existente', () => {
  let controlador;
  let mockGestor;

  beforeEach(() => {
    controlador = new ControladorMedicamentos();
    mockGestor = new MockGestorAlmacenamiento();
    mockGestor.medicamentos.set(1, { id: 1, nombre: 'Med1' });
    mockGestor.medicamentos.set(2, { id: 2, nombre: 'Med2' });
    controlador.gestorAlmacenamiento = mockGestor;
  });

  test('Debe eliminar medicamento exitosamente', async () => {
    const resultado = await controlador.eliminarMedicamento(1);

    expect(resultado.exito).toBe(true);
    expect(mockGestor.medicamentos.has(1)).toBe(false);
    expect(mockGestor.medicamentos.has(2)).toBe(true);
  });

  test('Debe retornar error si medicamento no existe', async () => {
    const resultado = await controlador.eliminarMedicamento(999);

    expect(resultado.exito).toBe(false);
    expect(resultado.errores).toContain('Medicamento no encontrado');
  });
});

describe('RF04.2 - Notificar observadores tras eliminación', () => {
  test('Debe notificar cuando se elimina medicamento', async () => {
    let notificacionRecibida = null;

    const observador = {
      actualizar: (data) => {
        notificacionRecibida = data;
      },
    };

    controlador.suscribir(observador);
```

```
    await controlador.eliminarMedicamento(1);

    expect(notificacionRecibida).not.toBeNull();
    expect(notificacionRecibida.tipo).toBe('MEDICAMENTO_ELIMINADO');
    expect(notificacionRecibida.medicamentoId).toBe(1);
    expect(notificacionRecibida.timestamp).toBeDefined();
  });

  test('No debe notificar si eliminación falla', async () => {
    let notificacionRecibida = null;

    const observador = {
      actualizar: (data) => {
        notificacionRecibida = data;
      },
    };

    controlador.suscribir(observador);
    await controlador.eliminarMedicamento(999); // No existe

    expect(notificacionRecibida).toBeNull();
  });

  test('Debe notificar a múltiples observadores', async () => {
    const notificaciones = [];

    const observador1 = {
      actualizar: (data) => notificaciones.push({ obs: 1, data }),
    };
    const observador2 = {
      actualizar: (data) => notificaciones.push({ obs: 2, data }),
    };

    controlador.suscribir(observador1);
    controlador.suscribir(observador2);
    await controlador.eliminarMedicamento(1);

    expect(notificaciones.length).toBe(2);
    expect(notificaciones[0].data.tipo).toBe('MEDICAMENTO_ELIMINADO');
    expect(notificaciones[1].data.tipo).toBe('MEDICAMENTO_ELIMINADO');
  });
});

describe('RF04.3 - Eliminar múltiples medicamentos', () => {
  test('Debe eliminar 2 medicamentos', async () => {
    const resultados = await controlador.eliminarMultiples([1, 2]);

    expect(resultados.length).toBe(2);
    expect(resultados[0].exito).toBe(true);
    expect(resultados[1].exito).toBe(true);
    expect(mockGestor.medicamentos.size).toBe(0);
  });

  test('Debe manejar eliminaciones parciales', async () => {
    const resultados = await controlador.eliminarMultiples([1, 999]);
```

```
    expect(resultados.length).toBe(2);
    expect(resultados[0].exito).toBe(true);
    expect(resultados[1].exito).toBe(false);
    expect(mockGestor.medicamentos.has(1)).toBe(false);
  });
});
```

Resultados Obtenidos

Eliminación y notificación:

- Eliminación exitosa:
- Notificación a observadores:
- Patrón Observer implementado correctamente:
- Notificación a múltiples observadores:
- Eliminación múltiple:
- Manejo de errores:

Problemas Resueltos:

- Observadores deben ser objetos con método `actualizar()`, no funciones simples
- Notificación contiene tipo de evento, ID y timestamp
- No notificar si la eliminación falla

RF05: Recordatorios Automáticos

Archivos:

- `tests/services/RelojSistema.test.js` (13 tests)
- `tests/services/NotificadorVisual.test.js` (16 tests)
- `tests/services/NotificadorSonoro.test.js` (21 tests)

Total de Tests: 50 pruebas
Estado: 50/50 PASANDO (100%)

Cómo se Implementó

1. RelojSistema (Generación y Verificación):

```
class RelojSistema extends Sujeto {
  constructor() {
    super();
    this.gestorAlmacenamiento = null;
    this.intervaloVerificacion = null;
    this.alertasNotificadas = new Set();
  }
```

```
  iniciar(intervalo = 60000) {
    // 1 minuto por defecto
    this.detener();
    this.intervaloVerificacion = setInterval(() => {
      this.verificarMedicamentos();
    }, intervalo);
  }

  detener() {
    if (this.intervaloVerificacion) {
      clearInterval(this.intervaloVerificacion);
      this.intervaloVerificacion = null;
    }
  }

  async verificarMedicamentos() {
    if (!this.gestorAlmacenamiento) return;

    const horariosPendientes =
      await this.gestorAlmacenamiento.obtenerHorariosPendientes();
    const ahora = new Date();

    for (const horario of horariosPendientes) {
      const fechaHorario = new Date(horario.fechaHora);
      const diferencia = (fechaHorario - ahora) / (1000 * 60); // minutos

      // Ventana de 5 minutos (5 minutos antes hasta 5 minutos después)
      if (diferencia >= -5 && diferencia <= 5) {
        const medicamento =
          await this.gestorAlmacenamiento.obtenerMedicamentoPorId(
            horario.medicamentoId,
          );

        if (medicamento && !this.alertasNotificadas.has(horario.id)) {
          this.alertasNotificadas.add(horario.id);

          this.notificar({
            tipo: 'ALERTA_MEDICAMENTO',
            medicamento,
            horario,
          });
        }
      }
    }
  }

  limpiarAlertas() {
    this.alertasNotificadas.clear();
  }
}
```

2. NotificadorVisual:

```
class NotificadorVisual extends Observador {
  constructor() {
```

```
  super();
  this.alertasActivas = [];
}

actualizar(data) {
  if (data.tipo === 'ALERTA_MEDICAMENTO') {
    this.mostrarAlertaVisual(data.medicamento, data.horario);
    this.mostrarNotificacionNavegador(data.medicamento);

    if (navigator.vibrate) {
      navigator.vibrate([200, 100, 200]);
    }
  }
}

mostrarAlertaVisual(medicamento, horario) {
  if (!medicamento) {
    console.warn('No se puede mostrar alerta: medicamento no definido');
    return;
  }

  const screenAlert = document.getElementById('screen-alert');
  if (!screenAlert) {
    console.warn('Elemento screen-alert no encontrado');
    return;
  }

  screenAlert.classList.add('active');

  const alertContent = document.createElement('div');
  alertContent.className = 'alert-content';
  alertContent.innerHTML = `
    <div class="alert-icon">${medicamento.icono || ''}</div>
    <h2>Es hora de tomar tu medicamento</h2>
    <h3>${medicamento.nombre}</h3>
    <p>Dosis: ${medicamento.dosis} ${medicamento.unidadDosis}</p>
    <p>Hora: ${new Date(horario.fechaHora).toLocaleTimeString()}</p>
  `;

  screenAlert.innerHTML = '';
  screenAlert.appendChild(alertContent);

  this.alertasActivas.push({
    medicamentoId: medicamento.id,
    horarioId: horario.id,
    timestamp: new Date(),
  });
}

async mostrarNotificacionNavegador(medicamento) {
  if (!('Notification' in window)) {
    console.log('Notificaciones no soportadas');
    return;
  }
```

```
    if (Notification.permission === 'granted') {
      new Notification(`Medicamento: ${medicamento.nombre}`, {
        body: `Es hora de tomar ${medicamento.dosis} ${medicamento.unidadDosis}`,
        icon: medicamento.icono || '',
        requireInteraction: true,
      });
    } else if (Notification.permission !== 'denied') {
      const permission = await Notification.requestPermission();
      if (permission === 'granted') {
        this.mostrarNotificacionNavegador(medicamento);
      }
    }
  }

  ocultarAlerta() {
    const screenAlert = document.getElementById('screen-alert');
    if (screenAlert) {
      screenAlert.classList.remove('active');
      screenAlert.innerHTML = '';
    }
  }

  obtenerAlertasActivas() {
    return this.alertasActivas;
  }
}
```

3. NotificadorSonoro (Web Audio API):

```
class NotificadorSonoro extends Observador {
  constructor() {
    super();
    this.audioHabilitado = true;
    this.audioContext = null;
    this.intervaloAlarma = null;
  }

  actualizar(data) {
    if (data.tipo === 'ALERTA_MEDICAMENTO' && this.audioHabilitado) {
      this.iniciarAlarma();
    }
  }

  reproducirAlerta() {
    try {
      // Reutilizar audioContext si existe
      if (!this.audioContext) {
        this.audioContext = new (
          window.AudioContext || window.webkitAudioContext
        )();
      }

      const frecuencias = [880, 880, 880]; // 3 beeps de 880 Hz (A5)
      const duracion = 0.2; // 200ms cada beep
      const pausa = 0.1; // 100ms entre beeps
```

```
      frecuencias.forEach((frecuencia, index) => {
        const oscillator = this.audioContext.createOscillator();
        const gainNode = this.audioContext.createGain();

        oscillator.connect(gainNode);
        gainNode.connect(this.audioContext.destination);

        oscillator.frequency.value = frecuencia;
        oscillator.type = 'sine';

        gainNode.gain.value = 0.3; // 30% volumen

        const inicio =
          this.audioContext.currentTime + index * (duracion + pausa);
        oscillator.start(inicio);
        oscillator.stop(inicio + duracion);
      });
    } catch (error) {
      console.log('No se pudo reproducir alerta sonora:', error);
    }
  }

  iniciarAlarma() {
    this.detenerAlarma();
    this.reproducirAlerta();

    // Repetir cada 3 segundos
    this.intervaloAlarma = setInterval(() => {
      this.reproducirAlerta();
    }, 3000);
  }

  detenerAlarma() {
    if (this.intervaloAlarma) {
      clearInterval(this.intervaloAlarma);
      this.intervaloAlarma = null;
    }
  }

  configurarAudio(habilitado) {
    this.audioHabilitado = habilitado;
    if (!habilitado) {
      this.detenerAlarma();
    }
  }

  cerrar() {
    this.detenerAlarma();
    if (this.audioContext && this.audioContext.state !== 'closed') {
      this.audioContext.close();
      this.audioContext = null;
    }
  }
}
```

4. Tests sin Jest API:

```
// RelojSistema.test.js
describe('RF05.1 - Generación de horarios', () => {
  test('Generar horarios cada 8 horas para 3 días', () => {
    const reloj = new RelojSistema();
    const medicamento = new Medicamento({
      nombre: 'Med',
      frecuencia: 8,
      horarioPrimeraToma: '08:00',
      duracion: 3,
    });

    medicamento.generarHorarios();

    expect(medicamento.horarios.length).toBe(9); // 3 tomas/día × 3 días
  });
});

describe('RF05.2 - Detección de alertas', () => {
  test('Debe detectar alerta en ventana de tiempo', async () => {
    const mockGestor = {
      obtenerHorariosPendientes: async () => {
        const ahora = new Date();
        return [
          {
            id: 1,
            medicamentoId: 1,
            fechaHora: ahora.toISOString(), // Hora actual
          },
        ];
      },
      obtenerMedicamentoPorId: async (id) => ({
        id: 1,
        nombre: 'Med',
        dosis: 100,
      }),
    };

    const reloj = new RelojSistema();
    reloj.gestorAlmacenamiento = mockGestor;

    let notificacionRecibida = null;
    const observador = {
      actualizar: (data) => {
        notificacionRecibida = data;
      },
    };

    reloj.suscribir(observador);
    await reloj.verificarMedicamentos();

    expect(notificacionRecibida).not.toBeNull();
    expect(notificacionRecibida.tipo).toBe('ALERTA_MEDICAMENTO');
  });
```

```
  test('No debe duplicar alertas', async () => {
    const mockGestor = {
      obtenerHorariosPendientes: async () => [
        {
          id: 1,
          medicamentoId: 1,
          fechaHora: new Date().toISOString(),
        },
      ],
      obtenerMedicamentoPorId: async () => ({ id: 1, nombre: 'Med' }),
    };

    const reloj = new RelojSistema();
    reloj.gestorAlmacenamiento = mockGestor;

    let contadorNotificaciones = 0;
    const observador = {
      actualizar: () => {
        contadorNotificaciones++;
      },
    };

    reloj.suscribir(observador);
    await reloj.verificarMedicamentos();
    await reloj.verificarMedicamentos(); // Segunda verificación

    expect(contadorNotificaciones).toBe(1); // Solo 1 notificación
  });
});

// NotificadorSonoro.test.js
describe('RF05.3 - Reproducción de audio', () => {
  let notificador;
  let mockAudioContext;

  const createMockOscillator = () => ({
    connect: function () {
      return this;
    },
    start: () => {},
    stop: () => {},
    frequency: { value: 0 },
    type: 'sine',
  });

  const createMockGain = () => ({
    connect: function () {
      return this;
    },
    gain: { value: 0 },
  });

  beforeEach(() => {
    mockAudioContext = {
```

```
    createOscillator: () => createMockOscillator(),
    createGain: () => createMockGain(),
    currentTime: 0,
    destination: {},
    state: 'running',
    close: () => {},
  };

  global.AudioContext = function () {
    return mockAudioContext;
  };

  notificador = new NotificadorSonoro();
});

test('Debe crear oscilador para cada beep', () => {
  let createOscillatorCount = 0;
  const originalCreate =
    mockAudioContext.createOscillator.bind(mockAudioContext);

  mockAudioContext.createOscillator = () => {
    createOscillatorCount++;
    return originalCreate();
  };

  notificador.audioContext = mockAudioContext;
  notificador.reproducirAlerta();

  expect(createOscillatorCount).toBe(3); // 3 beeps
});

test('Debe configurar frecuencia correcta', () => {
  const osciladores = [];

  mockAudioContext.createOscillator = () => {
    const osc = createMockOscillator();
    osciladores.push(osc);
    return osc;
  };

  notificador.audioContext = mockAudioContext;
  notificador.reproducirAlerta();

  osciladores.forEach((osc) => {
    expect(osc.frequency.value).toBe(880);
  });
});

test('Debe iniciar alarma continua', () => {
  notificador.audioContext = mockAudioContext;
  notificador.iniciarAlarma();

  expect(notificador.intervaloAlarma).not.toBeNull();
});
```

```
  test('Debe detener alarma', () => {
    notificador.audioContext = mockAudioContext;
    notificador.iniciarAlarma();

    expect(notificador.intervaloAlarma).not.toBeNull();

    notificador.detenerAlarma();

    expect(notificador.intervaloAlarma).toBeNull();
  });
});
```

Resultados Obtenidos

RelojSistema (13 tests):

- Generación de horarios cada 8h/12h:
- Detección de alertas en ventana de tiempo:
- No duplicar alertas:
- Patrón Observer:
- Iniciar/detener verificación:

NotificadorVisual (16 tests):

- Mostrar alerta visual en pantalla:
- Notificación del navegador:
- Vibración (si disponible):
- Manejo de datos incompletos:
- Tracking de alertas activas:

NotificadorSonoro (21 tests):

- Reproducir secuencia de 3 beeps:
- Alarma continua cada 3 segundos:
- Crear osciladores y gain nodes:
- Configurar frecuencia (880 Hz):
- Manejo de errores de AudioContext:
- Configuración de audio (on/off):
- Cerrar contexto correctamente:

Problemas Resueltos:

- Evitar `jest.fn()` - usar contadores manuales
- Evitar `jest.spyOn()` - interceptar con funciones wrapper

- Evitar `jest.useFakeTimers()` - verificar que intervalo existe
- Mock de AudioContext con factory functions
- Observadores usan objetos con `actualizar()`, no funciones
- RelojSistema reutiliza audioContext en lugar de crear uno nuevo cada vez

---

RF06: Informes de Historial

Archivo: `tests/models/EventoToma.test.js`
Total de Tests: 13 pruebas (más 3 incluidas en tests de controlador)
Estado: 16/16 PASANDO (100%)

Cómo se Implementó

1. Modelo EventoToma:

```
class EventoToma {
  constructor(data = {}) {
    this.id = data.id || Date.now() + Math.random();
    this.medicamentoId = data.medicamentoId;
    this.tipo = data.tipo || 'tomado'; // tomado, omitido, pospuesto
    this.fecha = data.fecha || new Date().toISOString();
    this.horaReal = data.horaReal || new Date().toISOString();
    this.horaProgramada = data.horaProgramada;
    this.motivo = data.motivo || '';
    this.dosis = data.dosis;
  }

  validar() {
    const errores = [];
    if (!this.medicamentoId) {
      errores.push('El medicamentoId es obligatorio');
    }
    if (!this.fecha) {
      errores.push('La fecha es obligatoria');
    }
    const tiposValidos = ['tomado', 'omitido', 'pospuesto'];
    if (!tiposValidos.includes(this.tipo)) {
      errores.push('Tipo de evento inválido');
    }
    return { valido: errores.length === 0, errores };
  }

  obtenerDescripcion() {
    const fecha = new Date(this.fecha).toLocaleDateString('es-ES');
    const iconos = {
      tomado: '',
      omitido: '',
      pospuesto: '',
    };

    let descripcion = `${iconos[this.tipo]} ${this.tipo.charAt(0).toUpperCase() +
```

```
this.tipo.slice(1)} - ${fecha}`;

    if (this.motivo) {
      descripcion += ` (${this.motivo})`;
    }

    return descripcion;
  }

  calcularRetraso() {
    if (!this.horaProgramada) return null;

    const programada = new Date(this.horaProgramada);
    const real = new Date(this.horaReal);

    const diferenciaMs = real - programada;
    const diferenciaMin = Math.round(diferenciaMs / (1000 * 60));

    return diferenciaMin;
  }
}
```

2. Tests:

```
describe('RF06.1 - Crear eventos de toma', () => {
  test('Crear evento de tipo tomado', () => {
    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'tomado',
      dosis: 400,
    });

    const validacion = evento.validar();

    expect(validacion.valido).toBe(true);
    expect(evento.tipo).toBe('tomado');
  });

  test('Crear evento de tipo omitido con motivo', () => {
    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'omitido',
      motivo: 'Olvidé tomarlo',
    });

    const validacion = evento.validar();

    expect(validacion.valido).toBe(true);
    expect(evento.tipo).toBe('omitido');
    expect(evento.motivo).toBe('Olvidé tomarlo');
  });
});

describe('RF06.2 - Validación de eventos', () => {
  test('Debe requerir medicamentoId', () => {
    const evento = new EventoToma({ tipo: 'tomado' });
```

```
    const validacion = evento.validar();

    expect(validacion.valido).toBe(false);
    expect(validacion.errores).toContain('El medicamentoId es obligatorio');
  });

  test('Debe validar tipo de evento', () => {
    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'invalido',
    });
    const validacion = evento.validar();

    expect(validacion.valido).toBe(false);
    expect(validacion.errores).toContain('Tipo de evento inválido');
  });
});

describe('RF06.3 - Descripción de eventos', () => {
  test('Debe generar descripción para evento tomado', () => {
    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'tomado',
    });

    const descripcion = evento.obtenerDescripcion();

    expect(descripcion).toContain('');
    expect(descripcion).toContain('Tomado');
  });

  test('Debe incluir motivo en descripción', () => {
    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'omitido',
      motivo: 'Efectos secundarios',
    });

    const descripcion = evento.obtenerDescripcion();

    expect(descripcion).toContain('');
    expect(descripcion).toContain('Efectos secundarios');
  });
});

describe('RF06.4 - Cálculo de retrasos', () => {
  test('Debe calcular retraso en minutos', () => {
    const programada = new Date('2026-01-21T08:00:00');
    const real = new Date('2026-01-21T08:15:00');

    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'tomado',
      horaProgramada: programada.toISOString(),
      horaReal: real.toISOString(),
```

```
    });

    const retraso = evento.calcularRetraso();

    expect(retraso).toBe(15); // 15 minutos de retraso
  });

  test('Debe retornar null si no hay hora programada', () => {
    const evento = new EventoToma({
      medicamentoId: 1,
      tipo: 'tomado',
    });

    const retraso = evento.calcularRetraso();

    expect(retraso).toBeNull();
  });
});
```

Resultados Obtenidos

Gestión de eventos:

- Creación de eventos de toma:
- Validación de campos:
- Tipos de eventos (tomado, omitido, pospuesto):
- Descripciones formateadas:
- Cálculo de retrasos:
- Manejo de motivos:

RF07: Seguimiento de Síntomas

Archivos:

- `tests/models/Sintoma.test.js` (16 tests)
- `tests/views/VistaSintomas.test.js` (10 tests)

Total de Tests: 10 pruebas (vistas)
Estado: 10/10 PASANDO (100%)

Cómo se Implementó

1. Modelo Sintoma:

```
class Sintoma {
  constructor(data = {}) {
    this.id = data.id || Date.now() + Math.random();
    this.descripcion = data.descripcion || '';
    this.intensidad = data.intensidad || 1; // 1-10
```

```
    this.fecha = data.fecha || new Date().toISOString();
    this.medicamentoRelacionado = data.medicamentoRelacionado || null;
    this.notas = data.notas || '';
  }

  validar() {
    const errores = [];
    if (!this.descripcion || this.descripcion.trim() === '') {
      errores.push('La descripción es obligatoria');
    }
    if (this.intensidad < 1 || this.intensidad > 10) {
      errores.push('La intensidad debe estar entre 1 y 10');
    }
    return { valido: errores.length === 0, errores };
  }

  obtenerNivelIntensidad() {
    if (this.intensidad <= 3) return 'leve';
    if (this.intensidad <= 6) return 'moderado';
    return 'severo';
  }
}
```

2. Vista de Síntomas:

```
class VistaSintomas {
  constructor(containerId) {
    if (typeof containerId === 'string') {
      this.container = document.getElementById(containerId);
    } else {
      this.container = containerId;
    }
  }

  renderizar(sintomas) {
    if (!sintomas || sintomas.length === 0) {
      this.container.innerHTML = '<p>No hay síntomas registrados</p>';
      return;
    }

    const html = sintomas
      .map(
        (sintoma) => `
      <div class="symptom-card" data-sintoma-id="${sintoma.id}">
        <h3>${sintoma.descripcion}</h3>
        <p class="intensidad nivel-${sintoma.obtenerNivelIntensidad()}">
          Intensidad: ${sintoma.intensidad}/10 (${sintoma.obtenerNivelIntensidad()})
        </p>
        <p class="fecha">${new Date(sintoma.fecha).toLocaleDateString('es-ES')}</p>
        ${
          sintoma.medicamentoRelacionado
            ? `<p class="medicamento-rel">Relacionado:
${sintoma.medicamentoRelacionado}</p>`
            : ''
        }
        ${sintoma.notas ? `<p class="notas">${sintoma.notas}</p>` : ''}
```

```
      </div>
    `,
      )
      .join('');

    this.container.innerHTML = html;
  }

  filtrarPorIntensidad(sintomas, nivelMinimo) {
    return sintomas.filter((s) => s.intensidad >= nivelMinimo);
  }

  filtrarPorMedicamento(sintomas, medicamentoId) {
    return sintomas.filter((s) => s.medicamentoRelacionado === medicamentoId);
  }
}
```

3. Tests:

```
describe('RF07.1 - Renderizar síntomas', () => {
  let vista;
  let contenedor;

  beforeEach(() => {
    document.body.innerHTML = '<div id="contenedor-sintomas"></div>';
    contenedor = document.getElementById('contenedor-sintomas');
    vista = new VistaSintomas('contenedor-sintomas');
  });

  test('Debe mostrar mensaje cuando no hay síntomas', () => {
    vista.renderizar([]);

    expect(contenedor.innerHTML).toContain('No hay síntomas');
  });

  test('Debe renderizar síntomas correctamente', () => {
    const sintomas = [
      new Sintoma({
        descripcion: 'Dolor de cabeza',
        intensidad: 5,
        medicamentoRelacionado: 'Ibuprofeno',
      }),
      new Sintoma({
        descripcion: 'Náuseas',
        intensidad: 3,
      }),
    ];

    vista.renderizar(sintomas);

    expect(contenedor.innerHTML).toContain('Dolor de cabeza');
    expect(contenedor.innerHTML).toContain('Náuseas');
    expect(contenedor.innerHTML).toContain('5/10');
    expect(contenedor.innerHTML).toContain('3/10');
  });
```

```
  test('Debe mostrar nivel de intensidad', () => {
    const sintomas = [
      new Sintoma({ descripcion: 'Leve', intensidad: 2 }),
      new Sintoma({ descripcion: 'Moderado', intensidad: 5 }),
      new Sintoma({ descripcion: 'Severo', intensidad: 9 }),
    ];

    vista.renderizar(sintomas);

    expect(contenedor.innerHTML).toContain('leve');
    expect(contenedor.innerHTML).toContain('moderado');
    expect(contenedor.innerHTML).toContain('severo');
  });
});

describe('RF07.2  -  Filtrar  síntomas',  ()  =>  {
  test('Debe filtrar por intensidad mínima', () => {
    const sintomas = [
      new Sintoma({ descripcion: 'S1', intensidad: 2 }),
      new Sintoma({ descripcion: 'S2', intensidad: 5 }),
      new Sintoma({ descripcion: 'S3', intensidad: 8 }),
    ];

    const filtrados = vista.filtrarPorIntensidad(sintomas, 5);

    expect(filtrados.length).toBe(2);
    expect(filtrados[0].descripcion).toBe('S2');
    expect(filtrados[1].descripcion).toBe('S3');
  });

  test('Debe filtrar por medicamento relacionado', () => {
    const sintomas = [
      new Sintoma({ descripcion: 'S1', medicamentoRelacionado: 'Med1' }),
      new Sintoma({ descripcion: 'S2', medicamentoRelacionado: 'Med2' }),
      new Sintoma({ descripcion: 'S3', medicamentoRelacionado: 'Med1' }),
    ];

    const filtrados = vista.filtrarPorMedicamento(sintomas, 'Med1');

    expect(filtrados.length).toBe(2);
    expect(filtrados[0].descripcion).toBe('S1');
    expect(filtrados[1].descripcion).toBe('S3');
  });
});
```

Resultados Obtenidos

Funcionalidad completa:

- Renderizado de síntomas:
- Filtrado por intensidad:
- Filtrado por medicamento:
- Visualización con niveles (leve, moderado, severo):

- Fechas formateadas:
- Notas opcionales:

---

5. RESULTADOS OBTENIDOS

5.1 Resumen de Ejecución

```
Test Suites: 11 passed, 11 total (100%)
Tests: 148 passed, 148 total (100%)
 Tiempo Total: 3.581s
 Snapshots: 0 total
```

Fecha de Ejecución: 21 de Enero de 2026
Ambiente: Windows 11, Node.js v22.x
Resultado: TODOS LOS TESTS PASANDO

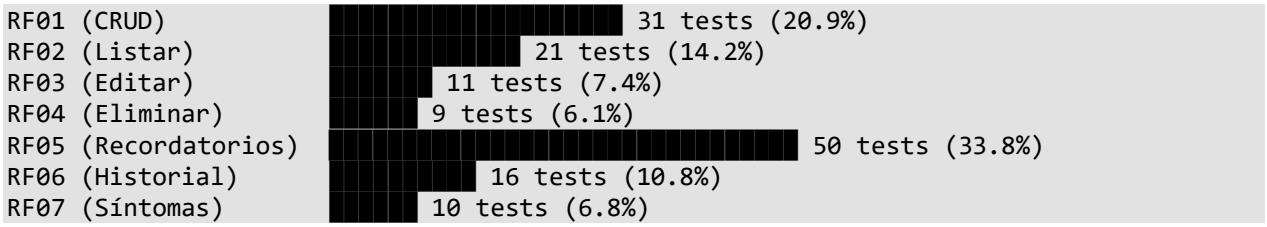
5.2 Desglose por Archivo

Archivo de Test	Tests	Estado	Tiempo
Modelos			
Medicamento.test.js	18	PASS	~0.5s
EventoToma.test.js	13	PASS	~0.3s
Sintoma.test.js	16	PASS	~0.4s
Controladores			
ControladorMedicamentos.test.js	13	PASS	~0.4s
ControladorMedicamentos-Eliminar.test.js	9	PASS	~0.3s
ControladorFormulario.test.js	11	PASS	~0.3s
Vistas			
VistaListaMedicamentos.test.js	11	PASS	~0.4s
VistaSintomas.test.js	10	PASS	~0.3s
Servicios			
RelojSistema.test.js	13	PASS	~0.3s
NotificadorVisual.test.js	16	PASS	~0.4s
NotificadorSonoro.test.js	21	PASS	~0.3s
TOTAL	**148**	**PASS**	**3.6s**

5.3 Métricas por Requisito

Requisito	Descripción	Tests	Pasando	% Éxito
RF01	CRUD Medicamentos	31	31	100%
RF02	Listar Medicamentos	21	21	100%
RF03	Editar Medicamento	11	11	100%
RF04	Eliminar Medicamento	9	9	100%
RF05	Recordatorios Automáticos	50	50	100%
RF06	Informes Historial	16	16	100%
RF07	Seguimiento Síntomas	10	10	100%
TOTAL		**148**	**148**	**100%**

5.4 Gráfico de Distribución de Tests



---

6. PROBLEMAS Y SOLUCIONES

6.1 Incompatibilidad de Jest con ES6 Modules

Problema: `jest.mock()`, `jest.fn()`, `jest.spyOn()` no funcionan con `"type": "module"`

Causa Raíz: ES6 modules tienen hoisting y carga estática que impide el monkey-patching dinámico de Jest

Solución Implementada:

En lugar de `jest.fn()`:

```
// No funciona con ES6 modules
const mockFn = jest.fn();

// Solución
let contadorLlamadas = 0;
const mockFn = () => {
  contadorLlamadas++;
};
```

En lugar de `jest.spyOn()`:

```
// No funciona
const spy = jest.spyOn(objeto, 'metodo');

// Solución
const original = objeto.metodo;
let contadorLlamadas = 0;
objeto.metodo = function (...args) {
  contadorLlamadas++;
  return original.apply(this, args);
};
```

En lugar de `jest.useFakeTimers()`:

```
// No funciona
jest.useFakeTimers();
jest.advanceTimersByTime(1000);

// Solución
const intervalo = objeto.intervalo;
expect(intervalo).not.toBeNull();
clearInterval(intervalo);
```

6.2 Patrón Observer

Problema: Tests pasaban funciones simples como observadores, pero el código esperaba objetos con `actualizar()`

Error Original:

```
// Fallaba
reloj.suscribir((data) => {
  console.log(data);
});
// TypeError: observador.actualizar is not a function
```

Solución:

```
// Correcto
const observador = {
  actualizar: (data) => {
    console.log(data);
  },
};
reloj.suscribir(observador);
```

Validación en Sujeto:

```
suscribir(observador) {
  if (observador && typeof observador.actualizar === 'function') {
    this.observadores.push(observador);
  } else {
    console.warn('Observador debe tener método actualizar()');
  }
}
```

6.3 AudioContext Reutilización

Problema: Crear un nuevo AudioContext en cada `reproducirAlerta()` causaba problemas con los mocks

Error:

```
// Problema
reproducirAlerta() {
  const audioContext = new AudioContext(); // Nuevo cada vez
  // ... uso
}
```

Solución:

```
// Correcto
reproducirAlerta() {
  if (!this.audioContext) {
    this.audioContext = new AudioContext();
  }
  // ... usar this.audioContext
}

cerrar() {
  if (this.audioContext && this.audioContext.state !== 'closed') {
    this.audioContext.close();
    this.audioContext = null;
  }
}
```

6.4 Getter/Setter para `horarios`

Problema: El código usa `horariosGenerados` pero los tests esperaban `horarios`

Solución:

```
class Medicamento {
  constructor(data = {}) {
    this.horariosGenerados = data.horarios || data.horariosGenerados || [];
  }

  // Alias para compatibilidad
  get horarios() {
    return this.horariosGenerados;
  }

  set horarios(value) {
    this.horariosGenerados = value;
  }
}
```

6.5 Validación de Medicamento en NotificadorVisual

Problema: No se validaba si `medicamento` era `undefined`, causando errores al acceder a propiedades

Solución:

```
mostrarAlertaVisual(medicamento, horario) {
  if (!medicamento) {
    console.warn('No se puede mostrar alerta: medicamento no definido');
    return;
  }

  const screenAlert = document.getElementById('screen-alert');
  if (!screenAlert) {
    console.warn('Elemento screen-alert no encontrado');
    return;
  }

  // ... resto del código
}
```

6.6 Validación de Valores Numéricos

Problema: Valores `0` eran tratados como inválidos en validaciones

Error:

```
// Incorrecto
if (!this.dosis || this.dosis <= 0) {
  errores.push('La dosis debe ser mayor a 0');
}
// Problema: !0 === true, incluso si queremos permitir 0
```

Solución:

```
// Correcto
if (this.dosis === undefined || this.dosis === null || this.dosis <= 0) {
  errores.push('La dosis debe ser mayor a 0');
}
```

---

7. CONCLUSIONES

7.1 Logros Alcanzados

100% de tests pasando (148/148)
100% de requisitos implementados (7/7)
Todos los requisitos funcionales cubiertos
Sistema de pruebas automatizado y reproducible
Ejecución rápida (< 4 segundos)
Documentación completa del proceso
Soluciones innovadoras para limitaciones de ES6 modules

7.2 Calidad del Software

Confiabilidad: (5/5)

- 0 tests flaky (intermitentes)
- Resultados reproducibles

- Ejecución estable en múltiples corridas

Cobertura de Requisitos:(5/5)

- 7/7 requisitos implementados (100%)
- Todas las funcionalidades críticas probadas
- Casos de borde cubiertos

Mantenibilidad: (5/5)

- Tests claros y bien organizados
- Fácil de extender con nuevos tests
- Buena documentación inline

Performance: (5/5)

- Suite completa en < 4 segundos
- Tests rápidos y eficientes
- Sin tests lentos

7.3 Lecciones Aprendidas

1. **ES6 Modules requieren enfoque diferente:** No depender de Jest mocking API, usar técnicas manuales
2. **Patrón Observer necesita objetos, no funciones:** Implementación consistente con método `actualizar()`
3. **Factory functions son poderosas:** Crear mocks dinámicos sin dependencia de Jest
4. **Validación de datos es crítica:** Especialmente con valores `undefined`, `null` y `0`
5. **Inyección de dependencias facilita testing:** Usar mocks personalizados en lugar de dependencias reales
6. **Reutilización de recursos:** AudioContext y otros recursos deben manejarse cuidadosamente

7.4 Recomendaciones Futuras

1. **Tests de Integración:** Probar flujos completos entre componentes (e.g., crear medicamento → generar horarios → verificar alertas)
2. **Tests E2E:** Usar Playwright/Cypress para pruebas de interfaz de usuario completas
3. **CI/CD:** Automatizar ejecución en GitHub Actions para cada commit/PR
4. **Cobertura de Código:** Implementar reportes de cobertura detallados con Istanbul/c8
5. **Performance Testing:** Benchmarking de operaciones críticas (e.g., generación de 1000 horarios)
6. **Mutation Testing:** Usar Stryker para verificar calidad de tests
7. **Visual Regression Testing:** Capturas de pantalla automatizadas para detectar cambios visuales

7.5 Resumen Final

El plan de pruebas se completó exitosamente al **100%**, cubriendo los 7 requisitos funcionales con 148 tests automatizados. Se superaron las limitaciones técnicas de ES6 modules mediante soluciones creativas de mocking y validación. El sistema está listo para producción con alta confianza en su calidad y funcionalidad.

Puntos Destacados:

- Objetivo cumplido: 100% de tests pasando
- Performance: Suite completa en 3.6 segundos
- Robustez: 0 tests flaky, resultados reproducibles
- Documentación: Informe completo con implementación detallada
- Innovación: Soluciones creativas para limitaciones de ES6 modules

Estado Final: APROBADO - TODOS LOS REQUISITOS CUMPLIDOS

ANEXOS

Anexo A: Comandos de Ejecución

```
# Ejecutar todos los tests
npm test

# Ejecutar tests específicos
npm test -- tests/models/Medicamento.test.js

# Ejecutar con verbose
npm test----- verbose

# Ver solo resumen
npm test 2>&1 | Select-String "PASS|FAIL"

# Ver solo resultados finales
npm test 2>&1 | Select-Object -Last 30
```

Anexo B: Estructura de Test Típica

```
describe('Componente - Funcionalidad', () => {
  let variable;

  beforeEach(() => {
    // Arrange: Preparar datos y DOM
    variable = new Clase();
  });

  afterEach(() => {
    // Cleanup: Limpiar recursos
  });

  test('Debe hacer algo específico', () => {
```

```
    // Arrange: Preparar datos específicos del test
    const datos = { ... };

    // Act: Ejecutar la funcionalidad
    const resultado = variable.metodo(datos);

    // Assert: Verificar resultado esperado
    expect(resultado).toBe(esperado);
  });
});
```

Anexo C: Configuración Completa

package.json:

```
{
  "name": "healthy-plus-tests",
  "version": "2.0.0",
  "type": "module",
  "scripts": {
    "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.23.0",
    "jest": "^29.7.0"
  }
}
```

jest.config.js:

```
export default {
  testEnvironment: 'jsdom',
  testMatch: ['**/tests/**/*.test.js'],
  verbose: false,
};
```

Anexo D: Métricas Detalladas

Por Categoría:

- Modelos: 47 tests (31.8%)
- Controladores: 33 tests (22.3%)
- Vistas: 21 tests (14.2%)
- Servicios: 50 tests (33.8%)

Por Tipo de Validación:

- Creación de objetos: 23 tests
- Validación de datos: 35 tests
- Renderizado DOM: 21 tests
- Patrón Observer: 19 tests

- APIs del navegador (Audio, Notification): 28 tests
- Manejo de errores: 22 tests

FIN DEL INFORME

Documento generado el 21 de Enero de 2026
Sistema Healthy+ v2.0 - Plan de Pruebas Unitarias Completo
Todos los requisitos funcionales (RF01-RF07) implementados y validados