

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 1 de 326
	Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Pruebas de software automatizadas con Jest

Desarrollado por:
 Darwin Steven Gómez
 Constanza Quira
 Liliana Pérez

Centro de teleinformática y producción industrial
 Tecnología en análisis y desarrollo de software
 Sena regional cauca
 Popayán cauca
 2025



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe permitir controlar el acceso a las rutas protegidas mediante un middleware de autorización, garantizando que solo los usuarios autenticados y con roles válidos puedan acceder a los recursos correspondientes.
Objetivo	Validar el funcionamiento integral del middleware de autorización y control de acceso del sistema, asegurando que los flujos de autenticación y permisos respondan correctamente ante distintos escenarios válidos e inválidos.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Solicitudes HTTP simuladas con y sin encabezado Authorization (token JWT), usuarios existentes y no existentes, roles permitidos y no permitidos. Ambiente: Node.js + Express + Jest
Procedimiento de Prueba	Prueba 1 – Responde 401 si no hay token: Paso 1: Iniciar el entorno de pruebas con Jest y cargar el middleware authorization. Paso 2: Enviar una solicitud simulada sin encabezado Authorization. Paso 3: Observar la respuesta generada por el middleware.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: El middleware debe responder con el código 401 Unauthorized y mensaje “Token no proporcionado”.</p> <p>Prueba 2 – Responde 401 si el token es inválido:</p> <p>Paso 1: Configurar una solicitud con encabezado Authorization: Bearer token_invalido.</p> <p>Paso 2: Ejecutar la prueba del middleware authorization.</p> <p>Paso 3: Revisar la respuesta del sistema ante el token inválido o expirado.</p> <p>Resultado esperado: El sistema debe devolver 401 Unauthorized con mensaje “Token inválido o expirado”.</p> <p>Prueba 3 – Responde 404 si el usuario no existe:</p> <p>Paso 1: Simular un token JWT válido que referencia un usuario inexistente.</p> <p>Paso 2: Ejecutar la prueba del middleware.</p> <p>Paso 3: Comprobar la respuesta generada.</p> <p>Resultado esperado: El middleware debe devolver 404 Not Found con mensaje “Usuario no encontrado”.</p> <p>Prueba 4 – Llama next() cuando el token y usuario son válidos:</p> <p>Paso 1: Enviar una solicitud con token JWT válido y usuario existente en la base de datos.</p> <p>Paso 2: Ejecutar el middleware authorization.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 3: Verificar que se llame correctamente la función next().</p> <p>Resultado esperado: El middleware permite el acceso a la siguiente función sin errores.</p> <p>Prueba 5 – Permite si el rol está incluido:</p> <p>Paso 1: Configurar el middleware verificarRol con roles permitidos.</p> <p>Paso 2: Enviar una solicitud con rol autorizado.</p> <p>Paso 3: Validar la ejecución.</p> <p>Resultado esperado: El middleware debe permitir el acceso y continuar con la ejecución normal.</p> <p>Prueba 6 – Bloquea si el rol no está incluido:</p> <p>Paso 1: Configurar el middleware con una lista de roles que no incluye al usuario autenticado.</p> <p>Paso 2: Ejecutar la solicitud simulada.</p> <p>Paso 3: Observar la respuesta.</p> <p>Resultado esperado: El sistema debe devolver 403 Forbidden con mensaje “Acceso denegado”.</p> <p>Prueba 7 – permitirDoctorOPropietarioPorIdHistorial 400 id inválido:</p> <p>Paso 1: Enviar una solicitud con parámetro idHistorial no numérico.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Paso 2: Ejecutar el middleware.

Paso 3: Validar la respuesta.

Resultado esperado: El sistema debe devolver 400 Bad Request con mensaje “ID inválido”.

Prueba 8 – permitirDoctorOPropietarioPorIdHistorial 404 no encontrado:

Paso 1: Enviar una solicitud con idHistorial numérico que no existe en la base de datos.

Paso 2: Ejecutar el middleware.

Paso 3: Revisar la respuesta.

Resultado esperado: El middleware debe devolver 404 Not Found.

Prueba 9 – permitirDoctorOPropietarioPorIdHistorial permite doctor o propietario:

Paso 1: Simular solicitud con usuario que tiene rol “doctor” o es propietario del historial.

Paso 2: Ejecutar el middleware.

Paso 3: Observar el flujo.

Resultado esperado: El middleware debe permitir el acceso y llamar next() correctamente.

Prueba 10 – permitirDoctorOPropietarioPorIdHistorial 403 si no cumple:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Enviar solicitud con usuario que no es ni doctor ni propietario.</p> <p>Paso 2: Ejecutar el middleware.</p> <p>Paso 3: Verificar la respuesta.</p> <p>Resultado esperado: El sistema debe devolver 403 Forbidden con mensaje “Acceso denegado”.</p> <p>Prueba 11 – permitirDoctorOPropietarioPorIdUsuario permite doctor o mismo usuario:</p> <p>Paso 1: Enviar solicitud con usuario autenticado que es doctor o coincide con el ID solicitado.</p> <p>Paso 2: Ejecutar el middleware correspondiente.</p> <p>Paso 3: Observar si continúa la ejecución.</p> <p>Resultado esperado: El middleware debe permitir el acceso sin errores.</p> <p>Prueba 12 – permitirDoctorOPropietarioPorIdUsuario 403 si no cumple:</p> <p>Paso 1: Enviar solicitud con usuario autenticado que no tiene permisos ni coincide con el ID solicitado.</p> <p>Paso 2: Ejecutar el middleware.</p> <p>Paso 3: Revisar la respuesta del sistema.</p> <p>Resultado esperado: El middleware debe devolver 403 Forbidden, bloqueando el acceso.</p>
Datos de salida - Resultado Esperado	1. El middleware responde con el código y mensaje adecuados (401, 403, 404, 400) según el caso.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>2. Se garantiza que solo usuarios autenticados con roles válidos puedan continuar a las rutas protegidas.</p> <p>3. Las rutas que usan authorization, verificarRol, permitirDoctorOPropietarioPorIdHistorial y permitirDoctorOPropietarioPorIdUsuario funcionan conforme al control de acceso definido.</p> <p>4. No se presentan errores inesperados durante la validación de tokens o roles.</p>	1.
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El middleware respondió correctamente ante todos los escenarios simulados, cumpliendo las validaciones de autenticación y permisos.	Si(x) No ()

AuthorizationMiddleware.test.js

Validar el funcionamiento integral del middleware de **autorización y control de acceso** del sistema, asegurando que los flujos de autenticación y permisos respondan correctamente ante distintos escenarios (válidos e inválidos).

Prueba 1: Responde 401 si no hay token

Validar que el middleware authorization deniegue el acceso cuando la solicitud no incluye el encabezado de autenticación con token JWT.

```
it('responde 401 si no hay token', async () => {
  const req = { header: () => null, method: 'GET', originalUrl: '/x' };
  const res = makeRes();
  await authorization(req, res, next);
  expect(res.status).toHaveBeenCalledWith(401);
  expect(res.json).toHaveBeenCalledWith({ mensaje: 'Acceso denegado, token no proporcionado' });
  expect(next).not.toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 2: Responde 401 si el token es inválido

Comprobar que el middleware rechace tokens corruptos o expirados, devolviendo un mensaje de error estándar.

```
it('responde 401 si el token es inválido', async () => {
  const req = { header: () => `Bearer badtoken`, method: 'GET', originalUrl: '/x' };
  const res = makeRes();
  jest.spyOn(jwt, 'verify').mockImplementation(() => { throw new Error('invalid'); });
  await authorization(req, res, next);
  expect(res.status).toHaveBeenCalledWith(401);
  expect(res.json).toHaveBeenCalledWith({ mensaje: 'Token inválido o expirado' });
});
```

Prueba 3: Responde 404 si el usuario no existe

```
const req = { header: () => `Bearer ${token}`, method: 'GET', originalUrl: '/x' };
const res = makeRes();
await authorization(req, res, next);
expect(res.status).toHaveBeenCalledWith(404);
expect(res.json).toHaveBeenCalledWith({ mensaje: 'Usuario no encontrado' });
});
```

Verificar que el middleware gestione el caso en el que el token es válido, pero el usuario asociado no se encuentra en la base de datos



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 4: Llama next() cuando el token y usuario son válido

Confirmar que el middleware permite el acceso cuando el token JWT es válido y el usuario existe en la base de datos

```
it('llama next cuando el token y usuario son válidos', async () => {
  const token = jwt.sign({ id: 1 }, process.env.JWT_SECRET);
  jest.spyOn(jwt, 'verify').mockReturnValue({ id: 1 });
  models.usuarios.findByPk.mockResolvedValue({ id: 1, rol: 'doctor' });

  const req = { header: () => `Bearer ${token}`, method: 'GET', originalUrl: '/x' };
  const res = makeRes();
  await authorization(req, res, next);
  expect(next).toHaveBeenCalled();
  expect(res.usuario).toEqual({ id: 1, rol: 'doctor' });
});
```

Prueba 5: Permite si el rol está incluido

Verificar que el middleware verificarRol permita el acceso cuando el rol del usuario autenticado se encuentra dentro de la lista de roles autorizados

```
describe('verificarRol', () => {
  it('permite si el rol está incluido', () => {
    const req = { usuario: { rol: 'doctor' }, originalUrl: '/x', method: 'GET' };
    const res = makeRes();
    const next = jest.fn();
    verificarRol(['doctor', 'asistente'])(req, res, next);
    expect(next).toHaveBeenCalled();
  });
});
```

Prueba 6: Bloquea si el rol no está incluido



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

Comprobar que el middleware rechace solicitudes cuando el usuario autenticado tiene un rol no autorizado para la acción solicitada.

```
it('bloquea si el rol no está incluido', () => {
  const req = { usuario: { rol: 'usuario' }, originalUrl: '/x', method: 'GET' };
  const res = makeRes();
  const next = jest.fn();
  verificarRol(['doctor'])(req, res, next);
  expect(res.status).toHaveBeenCalledWith(403);
  expect(res.json).toHaveBeenCalledWith({ mensaje: 'Acceso denegado por rol' });
  expect(next).not.toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 7: permitirDoctorOPropietarioPorIdHistorial 400 id inválido

Propósito

Verificar que el middleware detecte y rechace parámetros de ID no válidos (no numéricos) al intentar acceder a un historial clínico.

```
describe('permisos por pertenencia/rol', () => {
  test('permitirDoctorOPropietarioPorIdHistorial -> 400 id inválido', async () => {
    const req = { params: { id: 'abc' } };
    const res = makeRes();
    const next = jest.fn();
    await permitirDoctorOPropietarioPorIdHistorial(req, res, next);
    expect(res.status).toHaveBeenCalledWith(400);
  });
});
```

Prueba 8: permitirDoctorOPropietarioPorIdHistorial 404 no encontrado

Confirmar que el middleware gestione apropiadamente el caso en el que el historial clínico no existe.

```
test('permitirDoctorOPropietarioPorIdHistorial -> 404 no encontrado', async () => {
  models.historialclinico.findByPk.mockResolvedValue(null);
  const req = { params: { id: '1' } };
  const res = makeRes();
  const next = jest.fn();
  await permitirDoctorOPropietarioPorIdHistorial(req, res, next);
  expect(res.status).toHaveBeenCalledWith(404);
});
```

Prueba 9: permitirDoctorOPropietarioPorIdHistorial permite doctor o propietario

Verificar que el middleware autorice correctamente tanto al médico como al propietario del historial clínico

```
test('permitirDoctorOPropietarioPorIdHistorial -> permite doctor o propietario', async () => {
  models.historialclinico.findByPk.mockResolvedValue({ id: 1, id_usuario: 9 });
  const res = makeRes();
  const next = jest.fn();
  let req = { params: { id: '1' }, usuario: { id: 2, rol: 'doctor' } };
  await permitirDoctorOPropietarioPorIdHistorial(req, res, next);
  expect(next).toHaveBeenCalledTimes(1);
  next.mockClear();
  req = { params: { id: '1' }, usuario: { id: 9, rol: 'usuario' } };
  await permitirDoctorOPropietarioPorIdHistorial(req, res, next);
  expect(next).toHaveBeenCalledTimes(1);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 10: permitirDoctorOPropietarioPorIdHistorial 403 si no cumple
Validar que el middleware rechace solicitudes de usuarios que no sean ni doctores ni propietarios del historial.

```
test('permitirDoctorOPropietarioPorIdHistorial -> 403 si no cumple', async () => {
  models.historialclinico.findByPk.mockResolvedValue({ id: 1, id_usuario: 9 });
  const req = { params: { id: '1' }, usuario: { id: 3, rol: 'usuario' } };
  const res = makeRes();
  const next = jest.fn();
  await permitirDoctorOPropietarioPorIdHistorial(req, res, next);
  expect(res.status).toHaveBeenCalledWith(403);
});
```

Prueba 11: permitirDoctorOPropietarioPorIdUsuario permite doctor o mismo usuario

```
test('permitirDoctorOPropietarioPorIdUsuario -> permite doctor o mismo usuario', async () => {
  const res = makeRes();
  const next = jest.fn();
  let req = { params: { id: '7' }, usuario: { id: 1, rol: 'doctor' } };
  permitirDoctorOPropietarioPorIdUsuario(req, res, next);
  expect(next).toHaveBeenCalledTimes(1);
  next.mockClear();
  req = { params: { id: '7' }, usuario: { id: 7, rol: 'usuario' } };
  permitirDoctorOPropietarioPorIdUsuario(req, res, next);
  expect(next).toHaveBeenCalledTimes(1);
});
```

Prueba 12: permitirDoctorOPropietarioPorIdUsuario 403 si no cumple

Comprobar que el middleware bloquee accesos no autorizados en rutas de usuarios.

```
test('permitirDoctorOPropietarioPorIdUsuario -> 403 si no cumple', () => {
  const res = makeRes();
  const next = jest.fn();
  const req = { params: { id: '7' }, usuario: { id: 8, rol: 'usuario' } };
  permitirDoctorOPropietarioPorIdUsuario(req, res, next);
  expect(res.status).toHaveBeenCalledWith(403);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Resultado

```
Archivo Editar Ver Terminal Pestañas Ayuda
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend/_tests $ cd /home/dstevenngz/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend
npm test -- _tests_/Authorization.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/Authorization.test.js

PASS  _tests_/Authorization.test.js
  Middleware de autorización
    responde 401 si no hay token (7 ms)
    responde 401 si el token es inválido (3 ms)
    responde 404 si usuario no existe (8 ms)
    llama next cuando el token y usuario son válidos (3 ms)
  verificarRol
    permite si el rol está incluido (1 ms)
    bloquea si el rol no está incluido (1 ms)
  permisos por pertenencia/rol
    permitirDoctorPropietarioPorIdHistorial -> 400 id inválido (1 ms)
    permitirDoctorPropietarioPorIdHistorial -> 404 no encontrado (1 ms)
    permitirDoctorPropietarioPorIdHistorial -> permite doctor o propietario (1 ms)
    permitirDoctorPropietarioPorIdHistorial -> 403 si no cumple (1 ms)
    permitirDoctorPropietarioPorIdUsuario -> permite doctor o mismo usuario (2 ms)
    permitirDoctorPropietarioPorIdUsuario -> 403 si no cumple (1 ms)

Test Suites: 1 passed, 1 total
Tests:    12 passed, 12 total
Snapshots: 0 total
Time:    0.49 s, estimated 1 s
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	
Requerimiento Funcional de la prueba	El sistema debe permitir al usuario cambiar, solicitar restablecimiento y restablecer su contraseña mediante las funciones del controlador CambioDeContrasena, garantizando respuestas HTTP adecuadas (200 y 400) según el resultado de las operaciones.
Objetivo	Validar el funcionamiento integral del controlador de cambio de contraseña, asegurando que los tres métodos (cambiarcontraseña, solicitarReset, resetearPassword) gestionen correctamente las solicitudes exitosas y los errores del servicio asociado, respondiendo con los códigos adecuados y mensajes coherentes.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: cambiarcontraseña, solicitarReset, resetearPassword



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Simulaciones con jest.fn() para los servicios asociados.</p> <p>Escenarios con resultados exitosos y errores simulados.</p> <p>Ambiente de prueba: Node.js, Express, Jest</p>
Procedimiento de Prueba	<p>Prueba 1 – cambiarcontraseña 200 con resultado del servicio:</p> <p>Paso 1: Simular la ejecución del servicio cambiarcontraseña devolviendo un resultado exitoso.</p> <p>Paso 2: Llamar al método del controlador cambiarcontraseña.</p> <p>Paso 3: Observar la respuesta generada.</p> <p>Resultado esperado: El controlador debe responder con código 200 OK y devolver el resultado del servicio en formato JSON.</p> <p>Prueba 2 – cambiarcontraseña 400 cuando el servicio lanza error:</p> <p>Paso 1: Simular que el servicio cambiarcontraseña lanza un error.</p> <p>Paso 2: Ejecutar el método del controlador.</p> <p>Paso 3: Observar la respuesta devuelta.</p> <p>Resultado esperado: El controlador debe responder con 400 Bad Request, mostrando un mensaje de error coherente con la falla.</p> <p>Prueba 3 – solicitarReset 200 con resultado del servicio:</p> <p>Paso 1: Simular que el servicio solicitarReset devuelve un resultado exitoso.</p> <p>Paso 2: Llamar al método del controlador correspondiente.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El controlador debe responder con 200 OK y un mensaje confirmando el envío del correo de restablecimiento.</p> <p>Prueba 4 – solicitarReset 400 cuando el servicio lanza error:</p> <p>Paso 1: Simular error en el servicio solicitarReset.</p> <p>Paso 2: Ejecutar la función del controlador.</p>



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 3: Observar el manejo del error.</p> <p>Resultado esperado: El controlador debe devolver 400 Bad Request, indicando que no se pudo enviar el correo de recuperación.</p> <p>Prueba 5 – resetearPassword 200 con resultado del servicio:</p> <p>Paso 1: Simular ejecución correcta del servicio resetearPassword con token válido.</p> <p>Paso 2: Ejecutar el método del controlador.</p> <p>Paso 3: Validar la respuesta generada.</p> <p>Resultado esperado: El controlador debe devolver 200 OK, confirmando que la contraseña fue restablecida exitosamente.</p> <p>Prueba 6 – resetearPassword 400 cuando el servicio lanza error:</p> <p>Paso 1: Simular error del servicio por token inválido o expirado.</p> <p>Paso 2: Ejecutar el método resetearPassword.</p> <p>Paso 3: Observar la respuesta.</p> <p>Resultado esperado: El controlador debe devolver 400 Bad Request, informando que el token de restablecimiento no es válido o ha expirado.</p>		
Datos de salida - Resultado Esperado	<ol style="list-style-type: none">1. Cada método del controlador devuelve el código HTTP correcto según éxito o error.2. Los mensajes de error son claros y coherentes con la causa.3. No se producen excepciones no controladas.4. Los resultados exitosos se retornan en formato JSON adecuado		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El controlador respondió correctamente ante escenarios exitosos y de error, cumpliendo los códigos HTTP esperados.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

2. . CambiarContrasena.Controllers.test.js

Validar el **comportamiento del controlador de Cambio de Contraseña**, que incluye tres operaciones principales:

1. cambiarcontrasena — cambio directo por parte del usuario autenticado.
2. solicitarReset — solicitud de restablecimiento por correo electrónico.
3. resetearPassword — reinicio de contraseña mediante token temporal.

El archivo verifica que cada método del controlador responda con los **códigos HTTP 200 y 400**, según el resultado exitoso o fallido del servicio asociado.

```
jest.mock('../services/CambiarContrasenaServices', () => ({  
    cambiarcontrasena: jest.fn(),  
    solicitarReset: jest.fn(),  
    resetearPassword: jest.fn(),  
}););
```

Prueba 1: cambiarcontrasena 200 con resultado del servicio

```
const mockReqRes = (overrides = {}) => {  
    const req = { body: {}, params: {}, usuario: { id: 1 }, ...overrides };  
    const res = {  
        statusCode: 200,  
        body: undefined,  
        status(code) { this.statusCode = code; return this; },  
        json(payload) { this.body = payload; return this; },  
    };  
    return { req, res };  
};
```

Verificar que el controlador devuelva una respuesta **200 OK** cuando el servicio cambiarcontrasena se ejecuta correctamente.

```
test('cambiarcontrasena -> 200 con result del servicio', async () => {  
    usuarioService.cambiarcontrasena.mockResolvedValue({ message: 'ok' });  
    const { req, res } = mockReqRes({ body: { actual: 'a', nueva: 'b' } });  
    await controller.cambiarcontrasena(req, res);  
    expect(res.statusCode).toBe(200);  
    expect(res.body).toEqual({ message: 'ok' });  
    expect(usuarioService.cambiarcontrasena).toHaveBeenCalledWith(1, 'a', 'b');  
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 2: cambiarcontraseña 400 cuando el servicio lanza error

Asegurar que el controlador capture errores generados por el servicio y responda con código **400 (Bad Request)**.

```
test('cambiarcontraseña -> 400 cuando servicio lanza error', async () => {
    usuarioService.cambiarcontraseña.mockRejectedValue(new Error('boom'));
    const { req, res } = mockReqRes({ body: { actual: 'a', nueva: 'b' } });
    await controller.cambiarcontraseña(req, res);
    expect(res.statusCode).toBe(400);
    expect(res.body).toEqual({ error: 'boom' });
});
```

Prueba 3: solicitarReset 200 con resultado del servicio

Verificar que el controlador responda correctamente cuando se solicita el restablecimiento de contraseña y el servicio retorna un resultado exitoso.

```
test('solicitarReset -> 200 con resultado del servicio', async () => {
    usuarioService.solicitarReset.mockResolvedValue({ message: 'enviado' });
    const { req, res } = mockReqRes({ body: { correo: 'test@gmail.com' } });
    await controller.solicitarReset(req, res);
    expect(res.statusCode).toBe(200);
    expect(res.body).toEqual({ message: 'enviado' });
    expect(usuarioService.solicitarReset).toHaveBeenCalledWith('test@gmail.com');
});
```

Prueba 4: solicitarReset 400 cuando el servicio lanza error

Probar que el controlador capture y comunique apropiadamente un error generado por el servicio al intentar enviar el correo de recuperación.

```
test('solicitarReset -> 400 cuando servicio lanza error', async () => {
    usuarioService.solicitarReset.mockRejectedValue(new Error('correo no existe'));
    const { req, res } = mockReqRes({ body: { correo: 'bad@x.com' } });
    await controller.solicitarReset(req, res);
    expect(res.statusCode).toBe(400);
    expect(res.body).toEqual({ error: 'correo no existe' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 5: resetearPassword 200 con resultado del servicio

Confirmar que el controlador devuelva **200 OK** cuando la contraseña es restablecida exitosamente mediante un token válido.

```
test('resetearPassword -> 200 con resultado del servicio', async () => {
  usuarioService.resetearPassword.mockResolvedValue({ message: 'ok' });
  const { req, res } = mockReqRes({ params: { token: 't' }, body: { nueva: 'x' } });
  await controller.resetearPassword(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ message: 'ok' });
  expect(usuarioService.resetearPassword).toHaveBeenCalledWith('t', 'x');
});
```

Prueba 6: resetearPassword 400 cuando el servicio lanza error

```
test('resetearPassword -> 400 cuando servicio lanza error', async () => {
  usuarioService.resetearPassword.mockRejectedValue(new Error('token invalido'));
  const { req, res } = mockReqRes({ params: { token: 't' }, body: { nueva: 'x' } });
  await controller.resetearPassword(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'token invalido' });
});
```

Validar que se gestione adecuadamente un token inválido o expirado al intentar restablecer la contraseña.

Resultado

```
dstevengz@dstevenguez:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends npm test -- _tests/_CambiarContrasena.Controllers.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests/_CambiarContrasena.Controllers.test.js

PASS  _tests/_CambiarContrasena.Controllers.test.js
  Controlador de Cambio de Contraseña
    ✓ cambiarcontraseña -> 200 con resultado del servicio (15 ms)
    ✓ cambiarcontraseña -> 400 cuando servicio lanza error (2 ms)
    ✓ solicitarReset -> 200 con resultado del servicio (3 ms)
    ✓ solicitarReset -> 400 cuando servicio lanza error (3 ms)
    ✓ resetearPassword -> 200 con resultado del servicio (4 ms)
    ✓ resetearPassword -> 400 cuando servicio lanza error (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.438 s, estimated 1 s
Run all test suites matching _tests/_CambiarContrasena.Controllers.test.js/i
dstevengz@dstevenguez:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 19 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe permitir al usuario cambiar, solicitar restablecimiento y restablecer su contraseña mediante el servicio CambioDeContrasenaService, aplicando validaciones de seguridad, control de expiración de tokens y respuestas adecuadas ante casos exitosos o fallidos.
Objetivo	Validar el comportamiento integral del servicio de cambio y recuperación de contraseñas, garantizando que las funciones internas gestionen correctamente todos los escenarios: éxito, error lógico, error de validación y expiración de token, protegiendo la integridad y seguridad del proceso.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos del servicio: cambiarcontraseña, solicitarReset, resetearPassword. Datos simulados de usuario, contraseña, correo y token. Casos de error y éxito controlados con jest.fn(). Ambiente: Node.js + Jest.
Procedimiento de Prueba	Prueba 1 – cambiarcontraseña error si usuario no existe: Paso 1: Simular un ID de usuario inexistente. Paso 2: Ejecutar la función cambiarcontraseña. Paso 3: Observar la respuesta.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Resultado esperado: El servicio debe lanzar un error indicando “Usuario no encontrado”.

Prueba 2 – cambiarcontraseña error si actual incorrecta:

Paso 1: Simular un usuario con contraseña actual distinta a la ingresada.

Paso 2: Ejecutar la función del servicio.

Paso 3: Revisar el resultado.

Resultado esperado: El servicio debe devolver error “Contraseña actual incorrecta”.

Prueba 3 – cambiarcontraseña error si nueva muy corta:

Paso 1: Simular una nueva contraseña con menos de 6 caracteres.

Paso 2: Ejecutar el método cambiarcontraseña.

Paso 3: Validar la respuesta.

Resultado esperado: El servicio debe rechazar el cambio con mensaje “Contraseña demasiado corta”.

Prueba 4 – cambiarcontraseña error si nueva igual a actual:

Paso 1: Usar una nueva contraseña igual a la actual.

Paso 2: Ejecutar el método del servicio.

Paso 3: Observar el resultado.

Resultado esperado: El servicio debe arrojar error “La nueva contraseña no puede ser igual a la actual”.

Prueba 5 – cambiarcontraseña actualiza y guarda:



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Paso 1: Simular un flujo completo con credenciales correctas.

Paso 2: Ejecutar la función cambiarcontraseña.

Paso 3: Verificar que se actualice y guarde la nueva contraseña.

Resultado esperado: El servicio debe devolver éxito con mensaje “Contraseña actualizada correctamente”.

Prueba 6 – solicitarReset error si correo no registrado:

Paso 1: Simular solicitud de restablecimiento con correo inexistente.

Paso 2: Ejecutar solicitarReset.

Paso 3: Revisar la respuesta.

Resultado esperado: El servicio debe lanzar error “Correo no registrado”.

Prueba 7 – solicitarReset envía correo con link y token válido:

Paso 1: Simular correo registrado.

Paso 2: Ejecutar solicitarReset y capturar el envío de correo.

Paso 3: Validar que se genere token y enlace correctamente.

Resultado esperado: El servicio debe enviar correo con enlace de restablecimiento válido y token temporal.

Prueba 8 – resetearPassword error token inválido:

Paso 1: Simular token incorrecto o manipulado.

Paso 2: Ejecutar resetearPassword.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Paso 3: Revisar la respuesta.

Resultado esperado: El servicio debe lanzar error “Token inválido”.

Prueba 9 – resetearPassword expirado si tiempo pasó:

Paso 1: Simular token con tiempo de expiración vencido.

Paso 2: Ejecutar el método.

Paso 3: Observar el resultado.

Resultado esperado: El servicio debe rechazar la solicitud con mensaje “Token expirado”.

Prueba 10 – resetearPassword actualiza, guarda y borra token:

Paso 1: Simular token válido y usuario existente.

Paso 2: Ejecutar el flujo de restablecimiento completo.

Paso 3: Validar que la nueva contraseña se guarde y el token se elimine.

Resultado esperado: El servicio debe devolver mensaje “Contraseña restablecida con éxito” y borrar el token.

Prueba 11 – resetearPassword lanza error si nueva contraseña es muy corta:

Paso 1: Simular restablecimiento con contraseña nueva de menos de 6 caracteres.

Paso 2: Ejecutar el método.

Paso 3: Revisar la respuesta.

Resultado esperado: El servicio debe lanzar error “Contraseña demasiado corta”.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 12 – resetearPassword lanza error si usuario no existe:</p> <p>Paso 1: Simular token válido pero usuario eliminado de la base de datos.</p> <p>Paso 2: Ejecutar resetearPassword.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El servicio debe arrojar error “Usuario no encontrado”.</p>		
Datos de salida - Resultado Esperado	<ul style="list-style-type: none">1 Los flujos de cambio, solicitud y restablecimiento de contraseña se validan correctamente.2 Los errores de usuario, token y validaciones se gestionan con mensajes claros.3 Se garantiza la seguridad de las contraseñas y expiración de tokens.4 No se presentan excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El servicio gestionó correctamente todos los escenarios simulados, aplicando las validaciones de seguridad y los códigos esperados.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

3. CambiarContrasenaServices.test.js

Validar el comportamiento completo del **servicio de cambio y recuperación de contraseña**, abarcando los siguientes procesos:

- Cambio de contraseña con validaciones de seguridad.
- Solicitud de restablecimiento mediante correo electrónico.
- Restablecimiento de contraseña con token y expiración.

El objetivo es garantizar que el servicio responda adecuadamente a todos los posibles escenarios: éxito, error lógico, error de validación y expiración del token.

```
jest.mock("nodemailer", () => ({
  createTransport: jest.fn(),
}));

jest.mock("bcryptjs", () => ({
  compare: jest.fn(),
  hash: jest.fn(),
}));

jest.mock("../models", () => ({
  usuarios: { findByPk: jest.fn(), findOne: jest.fn() },
}));

const nodemailer = require("nodemailer");
const bcrypt = require("bcryptjs");
const models = require("../models");

const FIXED_TOKEN_HEX = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
jest.mock("crypto", () => ({
  randomBytes: jest.fn(() =>
    Buffer.from("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa", "hex")
  ),
}));
const crypto = require("crypto");
```

Prueba 1: Cambiarcontraseña error si usuario no existe

Verificar que el servicio arroje un error cuando el ID de usuario no se encuentra en la base de datos.

```
test("cambiarcontraseña -> error si usuario no existe", async () => {
  models.usuarios.findByPk.mockResolvedValue(null);
  await expect(svc.cambiarcontraseña(1, "old", "newpass")).rejects.toThrow(
    /Usuario no encontrado/
  );
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 2: cambiarcontraseña error si actual incorrecta

Garantizar que el servicio rechace cambios si la contraseña actual no coincide.

```
test("cambiarcontraseña -> error si actual incorrecta", async () => {
    models.usuarios.findByPk.mockResolvedValue({ id: 1, contraseña: "hash" });
    bcrypt.compare.mockResolvedValueOnce(false);
    await expect(svc.cambiarcontraseña(1, "bad", "newpass")).rejects.toThrow(
        /actual es incorrecta/
    );
});
```

Prueba 3: cambiarcontraseña error si nueva muy corta

Validar que se rechacen contraseñas de longitud inferior a 6 caracteres.

```
test("cambiarcontraseña -> error si actual incorrecta", async () => {
    models.usuarios.findByPk.mockResolvedValue({ id: 1, contraseña: "hash" });
    bcrypt.compare.mockResolvedValueOnce(false);
    await expect(svc.cambiarcontraseña(1, "bad", "newpass")).rejects.toThrow(
        /actual es incorrecta/
    );
});
```

Prueba 4: cambiarcontraseña error si nueva igual a actual

Evitar que el usuario reemplace su contraseña actual con la misma.

```
test("cambiarcontraseña -> error si nueva igual a actual", async () => {
    const usuario = { id: 1, contraseña: "hash" };
    models.usuarios.findByPk.mockResolvedValue(usuario);
    bcrypt.compare.mockResolvedValueOnce(true).mockResolvedValueOnce(true);
    await expect(svc.cambiarcontraseña(1, "old", "samepass")).rejects.toThrow(
        /no puede ser igual/
    );
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 5: cambiarcontraseña actualiza y guarda

Verificar el flujo completo de actualización de contraseña cuando todos los datos son válidos.

```
test("cambiarcontraseña -> actualiza y guarda", async () => {
  const save = jest.fn();
  const usuario = { id: 1, contraseña: "oldhash", save };
  models.usuarios.findByPk.mockResolvedValue(usuario);
  bcrypt.compare.mockResolvedValueOnce(true).mockResolvedValueOnce(false);
  bcrypt.hash.mockResolvedValue("newhash");
  const res = await svc.cambiarcontraseña(1, "old", "newpass");
  expect(usuario.contraseña).toBe("newhash");
  expect(save).toHaveBeenCalled();
  expect(res).toEqual({ message: expect.stringMatching(/actualizada/i) });
});
```

Prueba 6: solicitarReset error si correo no registrado

Evitar solicitudes de recuperación para correos inexistentes.

```
test("solicitarReset -> error si correo no registrado", async () => {
  models.usuarios.findOne.mockResolvedValue(null);
  await expect(svc.solicitarReset("no@site.com")).rejects.toThrow(
    /Correo no registrado/
  );
});
```

Prueba 7: solicitarReset → envía correo con link y token fijo

Verificar que se envíe correctamente el correo de restablecimiento con el enlace válido.

```
test("solicitarReset -> envía correo con link y token fijo", async () => {
  models.usuarios.findOne.mockResolvedValue({ id: 9, correo: "u@site.com" });
  const baseTime = 1_000_000;
  Date.now = jest.fn(() => baseTime);
  const res = await svc.solicitarReset("u@site.com");
  expect(nodemailer.createTransport).toHaveBeenCalledWith({
    service: "gmail",
    auth: { user: "sender@gmail.com", pass: "pass" },
  });
  const sendArgs =
    nodemailer.createTransport.mock.results[0].value.sendMail.mock
      .calls[0][0];
  expect(sendArgs.to).toBe("u@site.com");
  expect(sendArgs.html).toMatch(
    new RegExp(
      `${process.env.FRONTEND_URL}/resetearcontraseña/${FIXED_TOKEN_HEX}`
    )
  );
  expect(res).toEqual({ message: expect.stringMatching(/Correo enviado/) });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 8: resetearPassword error token inválido

Verificar que se rechacen intentos de restablecimiento con tokens incorrectos.

```
test("resetearPassword -> error token inválido", async () => {
    await expect(svc.resetearPassword("deadbeef", "newpass")).rejects.toThrow(
        /Token inválido/
    );
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 9: resetearPassword expirado si tiempo pasó
Validar la expiración del token después del tiempo límite.

```
test("resetearPassword -> expirado si tiempo pasó", async () => {
    models.usuarios.findOne.mockResolvedValue({ id: 2, correo: "x@site.com" });
    Date.now = jest.fn(() => 2_000_000);
    await svc.solicitarReset("x@site.com");
    Date.now = jest.fn(() => 2_000_000 + 3_600_000 + 10);
    await expect(
        svc.resetearPassword(FIXED_TOKEN_HEX, "newpass")
    ).rejects.toThrow(/Token inválido/);
});
```

Prueba 10: resetearPassword actualiza, guarda y borra token
Confirmar el flujo completo de restablecimiento exitoso.

```
test("resetearPassword -> actualiza, guarda y borra token", async () => {
    const save = jest.fn();
    models.usuarios.findOne.mockResolvedValue({ id: 5, correo: "z@site.com" });
    Date.now = jest.fn(() => 3_000_000);
    await svc.solicitarReset("z@site.com");

    models.usuarios.findByPk.mockResolvedValue({ id: 5, save });
    bcrypt.hash.mockResolvedValue("hashed");
    const r = await svc.resetearPassword(FIXED_TOKEN_HEX, "newpass");
    expect(save).toHaveBeenCalled();
    expect(r).toEqual({ message: expect.stringMatching(/restablecida/) });

    await expect(
        svc.resetearPassword(FIXED_TOKEN_HEX, "another")
    ).rejects.toThrow(/Token inválido/);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 11: resetearPassword lanza error si nueva contraseña es muy corta
Evitar que se establezcan contraseñas inseguras durante el restablecimiento.

```
test("resetearPassword -> lanza error si nueva contraseña es muy corta", async () => {
  models.usuarios.findOne.mockResolvedValue({
    id: 10,
    correo: "test@site.com",
  });
  Date.now = jest.fn(() => 4_000_000);
  await svc.solicitarReset("test@site.com");

  await expect(svc.resetearPassword(FIXED_TOKEN_HEX, "123")).rejects.toThrow(
    /al menos 6/
  );
});
```

Prueba 12: resetearPassword lanza error si usuario no existe

Verificar que se controle el caso en que el usuario desaparece de la base de datos entre solicitud y restablecimiento.

```
test("resetearPassword -> lanza error si usuario no existe", async () => {
  models.usuarios.findOne.mockResolvedValue({
    id: 88,
    correo: "ghost@site.com",
  });
  Date.now = jest.fn(() => 8_000_000);
  await svc.solicitarReset("ghost@site.com");

  models.usuarios.findByPk.mockResolvedValue(null);

  await expect(
    svc.resetearPassword(FIXED_TOKEN_HEX, "newpass123")
  ).rejects.toThrow(/Usuario no encontrado/);
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 31 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-SENA-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/CambiarContrasenaServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/CambiarContrasenaServices.test.js

PASS  _tests_/CambiarContrasenaServices.test.js
  CambiarContrasenaServices
    cambiarcontraseña -> error si usuario no existe (26 ms)
    cambiarcontraseña -> error si actual incorrecta (3 ms)
    cambiarcontraseña -> error si nueva muy corta (2 ms)
    cambiarcontraseña -> error si nueva igual a actual (1 ms)
    cambiarcontraseña -> actualiza y guarda (3 ms)
    solicitarReset -> error si correo no registrado (2 ms)
    solicitarReset -> envia correo con link y token fijo (3 ms)
    resetearPassword -> error token invalido (2 ms)
    resetearPassword -> expirado si tiempo pasó (2 ms)
    resetearPassword -> actualiza, guarda y borra token (2 ms)
    resetearPassword -> lanza error si nueva contraseña es muy corta (2 ms)
    resetearPassword -> lanza error si usuario no existe (1 ms)

Test Suites: 1 passed, 1 total
Tests:    12 passed, 12 total
Snapshots: 0 total
Time:    0.795 s, estimated 1 s
Ran all test suites matching _tests_/CambiarContrasenaServices.test.js/i.

dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-SENA-Clinica-Estetica-Frontend-Backend$ █
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe permitir la gestión completa del carrito de compras del usuario, incluyendo las operaciones de listar, agregar, eliminar y limpiar ítems, garantizando respuestas correctas (200, 201, 400, 500) según el resultado de cada operación y asegurando el manejo adecuado de errores.
Objetivo	Validar el funcionamiento del controlador CarritoControllers, comprobando que todos los endpoints del carrito respondan correctamente ante escenarios exitosos y fallidos, gestionen los datos del usuario autenticado y devuelvan códigos HTTP coherentes con cada situación.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos del controlador: listarMiCarrito, agregarAlCarrito, eliminarDelCarrito, limpiarMiCarrito. Usuario autenticado simulado mediante req.usuario.id. Servicios mockeados para casos de éxito y error. Ambiente: Node.js + Express + Jest.
Procedimiento de Prueba	Prueba 1 – listarMiCarrito 200: Paso 1: Simular una solicitud GET al endpoint /carrito. Paso 2: Mockear el servicio para devolver una lista de ítems. Paso 3: Ejecutar el controlador listarMiCarrito.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: El controlador debe devolver 200 OK con los ítems del carrito del usuario autenticado.</p> <p>Prueba 2 – listarMiCarrito 500 en error:</p> <p>Paso 1: Simular un error interno en el servicio de carrito.</p> <p>Paso 2: Ejecutar la función del controlador.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El controlador debe devolver 500 Internal Server Error con mensaje “Error al listar el carrito”.</p> <p>Prueba 3 – agregarAlCarrito 201 con ítem:</p> <p>Paso 1: Simular el envío de un procedimiento válido para agregar al carrito.</p> <p>Paso 2: Ejecutar el controlador agregarAlCarrito.</p> <p>Paso 3: Validar la respuesta.</p> <p>Resultado esperado: El sistema debe devolver 201 Created con los datos del ítem agregado.</p> <p>Prueba 4 – agregarAlCarrito 400 si ya existe:</p> <p>Paso 1: Simular que el procedimiento ya está en el carrito.</p> <p>Paso 2: Ejecutar el método del controlador.</p> <p>Paso 3: Observar la respuesta.</p> <p>Resultado esperado: El sistema debe devolver 400 Bad Request con mensaje “El ítem ya existe en el carrito”.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 5 – agregarAlCarrito 500 en otro error:

Paso 1: Simular error inesperado al agregar un ítem.

Paso 2: Ejecutar el método del controlador.

Paso 3: Revisar la respuesta.

Resultado esperado: El sistema debe devolver 500 Internal Server Error con mensaje genérico de error.

Prueba 6 – eliminarDelCarrito 200:

Paso 1: Simular solicitud DELETE con ID de ítem válido.

Paso 2: Ejecutar el método eliminarDelCarrito.

Paso 3: Validar la respuesta.

Resultado esperado: El sistema debe devolver 200 OK confirmando que el ítem fue eliminado correctamente.

Prueba 7 – eliminarDelCarrito 500 en error:

Paso 1: Simular error interno al intentar eliminar un ítem.

Paso 2: Ejecutar el controlador.

Paso 3: Observar la respuesta.

Resultado esperado: El sistema debe devolver 500 Internal Server Error con mensaje “Error al eliminar del carrito”.

Prueba 8 – limpiarMiCarrito 200:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Simular una solicitud para vaciar completamente el carrito.</p> <p>Paso 2: Ejecutar el controlador limpiarMiCarrito.</p> <p>Paso 3: Verificar la respuesta.</p> <p>Resultado esperado: El controlador debe devolver 200 OK, confirmando que el carrito se vació exitosamente.</p> <p>Prueba 9 – limpiarMiCarrito 500 en error:</p> <p>Paso 1: Simular un error del servicio al intentar limpiar el carrito.</p> <p>Paso 2: Ejecutar el controlador.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El sistema debe devolver 500 Internal Server Error indicando fallo en la limpieza del carrito.</p>		
Datos de salida - Resultado Esperado	<p>1 El sistema responde con los códigos correctos según el resultado de cada operación.</p> <p>2 Se confirman las funciones CRUD del carrito (listar, agregar, eliminar, limpiar).</p> <p>3 Los errores son manejados correctamente sin generar excepciones no controladas.</p> <p>4 Las respuestas se devuelven en formato JSON coherente y comprensible.</p>		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El controlador del carrito respondió correctamente en todos los escenarios simulados,	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	cumpliendo las validaciones y retornando los códigos esperados.		
--	--	--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

4. CarritoControllers.test.js

Validar el funcionamiento del **controlador de carrito de compras** dentro de la aplicación, asegurando que los endpoints de **listar, agregar, eliminar y limpiar** gestionen correctamente tanto los casos exitosos como los errores controlados.

Configuración y entorno de pruebas

```
jest.mock('../services/CarritoServices', () => ({
  listarCarritoPorUsuario: jest.fn(),
  agregarAlCarrito: jest.fn(),
  eliminarDelCarrito: jest.fn(),
  limpiarCarritoUsuario: jest.fn(),
}));
```

Función auxiliar:

```
const mockReqRes = (overrides = {}) => {
  const req = { body: {}, params: {}, usuario: { id: 7 }, ...overrides }
  const res = {
    statusCode: 200,
    body: undefined,
    status(code) { this.statusCode = code; return this; },
    json(payload) { this.body = payload; return this; },
  };
  return { req, res };
};
```

Prueba 1: Listar mi carrito

Verificar que el controlador retorne correctamente los ítems del carrito de un usuario autenticado.

```
test('listarMiCarrito -> 200 con items', async () => [
  carritoService.listarCarritoPorUsuario.mockResolvedValue([{ id: 1 }]),
  const { req, res } = mockReqRes();
  await controller.listarMiCarrito(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual([{ id: 1 }]);
  expect(carritoService.listarCarritoPorUsuario).toHaveBeenCalledWith(7),
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 2: listarMiCarrito 500 en error

Comprobar que el controlador maneje adecuadamente los errores al listar el carrito.

```
test('listarMiCarrito -> 500 en error', async () => [
  carritoService.listarCarritoPorUsuario.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes();
  await controller.listarMiCarrito(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: 'Error al obtener el carrito' });
]);
```

Prueba 3: agregarAlCarrito 201 con item

Validar que el controlador agregue correctamente un procedimiento al carrito y retorne el código **201**

```
test('agregarAlCarrito -> 201 con item', async () => {
  carritoService.agregarAlCarrito.mockResolvedValue({ id: 2 });
  const { req, res } = mockReqRes({ body: { id_procedimiento: 5 } });
  await controller.agregarAlCarrito(req, res);
  expect(res.statusCode).toBe(201);
  expect(res.body).toEqual({ id: 2 });
  expect(carritoService.agregarAlCarrito).toHaveBeenCalledWith({ id_procedimiento: 5, id_usuario: 7 });
});
```

Prueba 4: agregarAlCarrito 400 si ya existe

Comprobar que el controlador devuelva **400 (Bad Request)** si el ítem ya estaba agregado.

```
test('agregarAlCarrito -> 400 si ya existe', async () => [
  carritoService.agregarAlCarrito.mockRejectedValue(new Error('El procedimiento ya está en el carrito.'));
  const { req, res } = mockReqRes({ body: { id_procedimiento: 5 } });
  await controller.agregarAlCarrito(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'El procedimiento ya está en el carrito.' });
]);
```

Prueba 5: agregarAlCarrito 500 en otro error

Verificar que cualquier otro error inesperado genere una respuesta genérica **500**.

```
test('agregarAlCarrito -> 500 en otro error', async () => {
  carritoService.agregarAlCarrito.mockRejectedValue(new Error('db down'));
  const { req, res } = mockReqRes({ body: { id_procedimiento: 5 } });
  await controller.agregarAlCarrito(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: 'Error al agregar al carrito' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 6: eliminarDelCarrito 200**Confirmar que se elimine correctamente un ítem del carrito.**

```
test('eliminarDelCarrito -> 200', async () => {
  carritoService.eliminarDelCarrito.mockResolvedValue(1);
  const { req, res } = mockReqRes({ params: { id: 9 } });
  await controller.eliminarDelCarrito(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ mensaje: 'Procedimiento eliminado del carrito' });
  expect(carritoService.eliminarDelCarrito).toHaveBeenCalledWith(9);
});
```

Prueba 7: eliminarDelCarrito 500 en error

Asegurar que ante un fallo interno el controlador responda correctamente.

```
test('eliminarDelCarrito -> 500 en error', async () => {
  carritoService.eliminarDelCarrito.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes({ params: { id: 9 } });
  await controller.eliminarDelCarrito(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: 'Error al eliminar del carrito' });
});
```

Prueba 8: limpiarMiCarrito 200

Verificar que se limpie el carrito completo del usuario correctamente.

```
test('limpiarMiCarrito -> 200', async () => {
  carritoService.limpiarCarritoUsuario.mockResolvedValue(1);
  const { req, res } = mockReqRes();
  await controller.limpiarMiCarrito(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ mensaje: 'Carrito limpiado correctamente' });
  expect(carritoService.limpiarCarritoUsuario).toHaveBeenCalledWith(7);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 9: limpiarMiCarrito 500 en error

```
test('limpiarMiCarrito -> 500 en error', async () => {
  carritoService.limpiarCarritoUsuario.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes();
  await controller.limpiarMiCarrito(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: 'Error al limpiar el carrito' });
});
```

Resultado

```
no tests found, exiting with code 0
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/CarritoControllers.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/CarritoControllers.test.js
PASS  tests/_CarritoControllers.test.js
Controlador de Carrito
  ✓ listarMiCarrito -> 200 con items (7 ms)
  ✓ listarMiCarrito -> 500 en error (2 ms)
  ✓ agregarAlCarrito -> 201 con item (2 ms)
  ✓ agregarAlCarrito -> 400 si ya existe (1 ms)
  ✓ agregarAlCarrito -> 500 en otro error (1 ms)
  ✓ eliminarDelCarrito -> 200 (1 ms)
  ✓ eliminarDelCarrito -> 500 en error (1 ms)
  ✓ limpiarMiCarrito -> 200 (1 ms)
  ✓ limpiarMiCarrito -> 500 en error (1 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        0.769 s, estimated 1 s
Run all test suites matching _tests_/_CarritoControllers.test.js/i.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	Requerimiento Funcional de la prueba



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Objetivo	Validar el comportamiento completo del módulo CarritoServices, asegurando que las operaciones de consulta, inserción y eliminación deleguen correctamente a los métodos del modelo (findAll, create, destroy), evitando duplicados y propagando los errores generados por el ORM.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Servicios: listarCarritoPorUsuario, agregarAlCarrito, eliminarDelCarrito, limpiarCarritoUsuario. Modelos Sequelize mockeados (carrito, procedimientos). Simulación de casos exitosos y fallidos. Ambiente: Node.js + Jest.
Procedimiento de Prueba	Prueba 1 – listarCarritoPorUsuario delega a findAll con include: Paso 1: Simular usuario con ID válido. Paso 2: Ejecutar listarCarritoPorUsuario. Paso 3: Verificar que el método findAll del modelo carrito sea llamado con where: { usuarioid } e include de procedimientos. Resultado esperado: El servicio debe invocar findAll correctamente con los parámetros esperados. Prueba 2 – agregarAlCarrito evita duplicados: Paso 1: Simular un carrito que ya contiene el procedimiento. Paso 2: Ejecutar agregarAlCarrito. Paso 3: Observar la respuesta.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: El servicio debe rechazar el intento y lanzar un error “El procedimiento ya existe en el carrito”.</p> <p>Prueba 3 – agregarAlCarrito crea ítem y devuelve procedimiento:</p> <p>Paso 1: Simular un carrito vacío.</p> <p>Paso 2: Ejecutar agregarAlCarrito con datos válidos.</p> <p>Paso 3: Verificar que se llame create y se devuelva el procedimiento agregado.</p> <p>Resultado esperado: El servicio debe crear correctamente el ítem y devolver el objeto del procedimiento agregado.</p> <p>Prueba 4 – eliminarDelCarrito delega a destroy:</p> <p>Paso 1: Simular ID válido del ítem del carrito.</p> <p>Paso 2: Ejecutar eliminarDelCarrito.</p> <p>Paso 3: Comprobar la llamada al método destroy.</p> <p>Resultado esperado: El servicio debe invocar destroy con los parámetros correctos y eliminar el ítem.</p> <p>Prueba 5 – limpiarCarritoUsuario delega a destroy por usuario:</p> <p>Paso 1: Simular ID de usuario.</p> <p>Paso 2: Ejecutar limpiarCarritoUsuario.</p> <p>Paso 3: Revisar la ejecución.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: El servicio debe invocar destroy con where: { usuariold } y eliminar todos los ítems del usuario.</p> <p>Prueba 6 – listarCarritoPorUsuario lanza error si findAll falla:</p> <p>Paso 1: Simular que el método findAll lanza un error.</p> <p>Paso 2: Ejecutar listarCarritoPorUsuario.</p> <p>Paso 3: Observar el resultado.</p> <p>Resultado esperado: El servicio debe propagar el error correctamente sin alterar el mensaje.</p> <p>Prueba 7 – agregarAlCarrito lanza error si create falla:</p> <p>Paso 1: Simular error del método create.</p> <p>Paso 2: Ejecutar agregarAlCarrito.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El servicio debe lanzar el error capturado del modelo create.</p> <p>Prueba 8 – eliminarDelCarrito lanza error si destroy falla:</p> <p>Paso 1: Simular que el método destroy arroja un error.</p> <p>Paso 2: Ejecutar eliminarDelCarrito.</p> <p>Paso 3: Observar el manejo del error.</p> <p>Resultado esperado: El servicio debe propagar el error correctamente sin detener la ejecución general del sistema.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 9 – limpiarCarritoUsuario maneja errores correctamente:</p> <p>Paso 1: Simular un fallo interno al ejecutar destroy en limpiarCarritoUsuario.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Observar el resultado.</p> <p>Resultado esperado: El servicio debe capturar y lanzar el error con un mensaje claro sin afectar la estabilidad del sistema.</p>		
Datos de salida - Resultado Esperado	1 Todas las funciones del servicio delegan correctamente a los métodos Sequelize (findAll, create, destroy). 2 Se evita la duplicación de ítems en el carrito. 3 Los errores son manejados y propagados correctamente. 4 Las respuestas mantienen coherencia y no dependen de la base de datos real.		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – Los servicios del módulo Carrito funcionan correctamente, delegando operaciones y manejando errores conforme al diseño esperado.	Si(<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

5. CarritoServices.test.js

Validar la correcta ejecución y manejo de errores en los servicios del módulo **Carrito**, garantizando que las operaciones de consulta, inserción y eliminación interactúen correctamente con los modelos Sequelize mockeados, sin dependencias externas a la base de datos.

Mocks iniciales:

```
jest.mock("../models", () => ({
  carrito: {
    findAll: jest.fn(),
    findOne: jest.fn(),
    create: jest.fn(),
    destroy: jest.fn(),
  },
  procedimientos: { findByPk: jest.fn() },
));
const models = require("../models");
jest.spyOn(console, "log").mockImplementation(() => {});
```

Prueba 1 : listarCarritoPorUsuario delega a findAll con include

Verificar que el servicio listarCarritoPorUsuario llame correctamente a findAll con los parámetros where y include.

```
test("listarCarritoPorUsuario delega a findAll con include", async () => {
  models.carrito.findAll.mockResolvedValue([{ id: 1 }]);
  const res = await svc.listarCarritoPorUsuario(9);
  expect(models.carrito.findAll).toHaveBeenCalledWith({
    where: { id_usuario: 9 },
    include: [{ model: models.procedimientos, as: "procedimiento" }],
  });
  expect(res).toEqual([{ id: 1 }]);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 2: agregarAlCarrito evita duplicados

Garantizar que el servicio no permita agregar un procedimiento duplicado al carrito.

```
test("agregarAlCarrito evita duplicados", async () => {
  models.carrito.findOne.mockResolvedValue({ id: 1 });
  await expect(
    svc.agregarAlCarrito({ id_usuario: 1, id_procedimiento: 2 })
  ).rejects.toThrow(/ya está en el carrito/);
});
```

Prueba 3: agregarAlCarrito crea item y devuelve procedimiento

Verificar el flujo exitoso al agregar un nuevo procedimiento al carrito.

```
test("agregarAlCarrito crea item y devuelve procedimiento", async () => {
  models.carrito.findOne.mockResolvedValue(null);
  models.carrito.create.mockResolvedValue({ id: 10 });
  models.procedimientos.findByPk.mockResolvedValue({ id: 2, nombre: "Proc" });
  const res = await svc.agregarAlCarrito({
    id_usuario: 1,
    id_procedimiento: 2,
  });
  expect(models.carrito.create).toHaveBeenCalledWith({
    id_usuario: 1,
    id_procedimiento: 2,
  });
  expect(models.procedimientos.findByPk).toHaveBeenCalledWith(2);
  expect(res).toEqual({ id: 10, procedimiento: { id: 2, nombre: "Proc" } });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 4: eliminarDelCarrito delega a destroy

Confirmar que el servicio invoque correctamente el método destroy del modelo carrito.

```
test("eliminarDelCarrito delega a destroy", async () => {
  models.carrito.destroy.mockResolvedValue(1);
  const res = await svc.eliminarDelCarrito(3);
  expect(models.carrito.destroy).toHaveBeenCalledWith({ where: { id: 3 } });
  expect(res).toBe(1);
});
```

Prueba 5: limpiarCarritoUsuario delega a destroy por usuario

Comprobar que el servicio borre todos los ítems de un usuario específico.

```
test("limpiarCarritoUsuario delega a destroy por usuario", async () => {
  models.carrito.destroy.mockResolvedValue(2);
  const res = await svc.limpiarCarritoUsuario(7);
  expect(models.carrito.destroy).toHaveBeenCalledWith({
    where: { id_usuario: 7 },
  });
  expect(res).toBe(2);
});
```

Prueba 6: listarCarritoPorUsuario lanza error si findAll falla

Verificar que los errores del modelo findAll sean propagados correctamente.

```
test("listarCarritoPorUsuario lanza error si findAll falla", async () => {
  models.carrito.findAll.mockRejectedValue(new Error("DB error"));
  await expect(svc.listarCarritoPorUsuario(1)).rejects.toThrow("DB error");
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 7: agregarAlCarrito → lanza error si create falla

Confirmar que los errores durante la creación del ítem se propaguen adecuadamente.

```
test("agregarAlCarrito crea item y devuelve procedimiento", async () => {
    models.carrito.findOne.mockResolvedValue(null);
    models.carrito.create.mockResolvedValue({ id: 10 });
    models.procedimientos.findByPk.mockResolvedValue({ id: 2, nombre: "Proc" });
    const res = await svc.agregarAlCarrito({
        id_usuario: 1,
        id_procedimiento: 2,
    });
    expect(models.carrito.create).toHaveBeenCalledWith({
        id_usuario: 1,
        id_procedimiento: 2,
    });
    expect(models.procedimientos.findByPk).toHaveBeenCalledWith(2);
    expect(res).toEqual({ id: 10, procedimiento: { id: 2, nombre: "Proc" } });
});
```

Prueba 8: eliminarDelCarrito → lanza error si destroy falla

```
test("eliminarDelCarrito lanza error si destroy falla", async () => {
    models.carrito.destroy.mockRejectedValue(new Error("falló eliminar"));
    await expect(svc.eliminarDelCarrito(5)).rejects.toThrow("falló eliminar");
});
```

Prueba 9: limpiarCarritoUsuario → maneja errores correctamente

Confirmar que, si ocurre un error al limpiar el carrito, el servicio

```
test("limpiarCarritoUsuario maneja errores correctamente", async () => {
    // Simula que falla la base de datos
    const fakeError = new Error("DB error");
    models.carrito.destroy.mockRejectedValue(fakeError);

    // Ejecuta el método
    await expect(svc.limpiarCarritoUsuario(7)).rejects.toThrow("DB error");

    // Verifica que el log fue llamado (opcional)
    expect(console.log).toHaveBeenCalledWith(
        "Error al limpiar el carrito del usuario:",
        fakeError
    );
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/CarritoServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/CarritoServices.test.js

PASS  tests/_CarritoServices.test.js
  Servicios de Carrito
    ✓ listarCarritoPorUsuario delega a findAll con include (9 ms)
    ✓ agregarAlCarrito evita duplicados (17 ms)
    ✓ agregarAlCarrito crea item y devuelve procedimiento (2 ms)
    ✓ eliminarDelCarrito delega a destroy (1 ms)
    ✓ limpiarCarritoUsuario delega a destroy por usuario (2 ms)
    ✓ listarCarritoPorUsuario lanza error si findAll falla (3 ms)
    ✓ agregarAlCarrito lanza error si create falla (1 ms)
    ✓ eliminarDelCarrito lanza error si destroy falla (1 ms)
    ✓ limpiarCarritoUsuario maneja errores correctamente (1 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        0.73 s, estimated 1 s
Ran all test suites matching _tests/_CarritoServices.test.js/i.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe permitir gestionar las categorías de procedimientos a través del controlador CategoriaProcedimientosControllers, realizando correctamente las operaciones CRUD (listar, buscar, crear, actualizar y eliminar), validando los datos de entrada y devolviendo respuestas HTTP coherentes según el resultado de cada operación.
Objetivo	Verificar el correcto funcionamiento integral del controlador de categorías de procedimientos, asegurando que maneje adecuadamente los flujos exitosos y los errores controlados en cada operación, con manejo apropiado de validaciones, duplicados y excepciones internas.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos probados: listarCategorias, buscarCategoria, crearCategoria, actualizarCategoria, eliminarCategoria. Servicios mockeados con jest.fn(). Datos de entrada simulados (ID válidos, inválidos, nombres duplicados, valores booleanos como string). Ambiente: Node.js + Express + Jest.
Procedimiento de Prueba	Prueba 1 – listarCategorias 200 (éxito): Paso 1: Simular una solicitud GET para obtener categorías.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Mockear el servicio listarCategorias devolviendo una lista de objetos.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: Devuelve 200 OK con el listado completo de categorías.</p> <p>Prueba 2 – buscarCategoria 200 cuando existe:</p> <p>Paso 1: Simular un ID existente.</p> <p>Paso 2: Mockear el servicio buscarCategoria devolviendo una categoría válida.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: Devuelve 200 OK con la información de la categoría solicitada.</p> <p>Prueba 3 – buscarCategoria 404 cuando no existe:</p> <p>Paso 1: Simular un ID inexistente.</p> <p>Paso 2: Mockear el servicio para devolver null.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: Devuelve 404 Not Found con mensaje “Categoría no encontrada”.</p> <p>Prueba 4 – buscarCategoria 500 en error:</p> <p>Paso 1: Simular un error interno en el servicio.</p> <p>Paso 2: Ejecutar la función del controlador.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Paso 3: Revisar la respuesta.

Resultado esperado: Devuelve 500 Internal Server Error con mensaje genérico de error.

Prueba 5 – crearCategoria 201 con datos válidos:

Paso 1: Enviar un cuerpo de solicitud con nombre válido y campo booleano “true” (string).

Paso 2: Ejecutar el controlador crearCategoria.

Paso 3: Validar la respuesta.

Resultado esperado: Devuelve 201 Created, convirtiendo el valor “true” (string) en true (boolean) correctamente.

Prueba 6 – crearCategoria 409 conflicto nombre duplicado:

Paso 1: Simular que el nombre de la categoría ya existe.

Paso 2: Ejecutar el controlador.

Paso 3: Observar la respuesta.

Resultado esperado: Devuelve 409 Conflict con mensaje “La categoría ya existe”.

Prueba 7 – crearCategoria 400 nombre requerido:

Paso 1: Simular creación sin incluir el campo “nombre”.

Paso 2: Ejecutar el controlador.

Paso 3: Revisar la respuesta.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: Devuelve 400 Bad Request con mensaje “El nombre es requerido”.</p> <p>Prueba 8 – crearCategoria 500 error genérico:</p> <p>Paso 1: Simular un error inesperado al crear la categoría.</p> <p>Paso 2: Ejecutar la función del controlador.</p> <p>Paso 3: Observar el resultado.</p> <p>Resultado esperado: Devuelve 500 Internal Server Error con mensaje de error genérico.</p> <p>Prueba 9 – actualizarCategoria 400 ID inválido:</p> <p>Paso 1: Simular ID no numérico.</p> <p>Paso 2: Ejecutar actualizarCategoria.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: Devuelve 400 Bad Request con mensaje “ID inválido”.</p> <p>Prueba 10 – actualizarCategoria 404 cuando no existe:</p> <p>Paso 1: Simular ID correcto pero categoría inexistente.</p> <p>Paso 2: Mockear el servicio para devolver 0 (sin filas actualizadas).</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: Devuelve 404 Not Found con mensaje “Categoría no encontrada”.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 11 – actualizarCategoria 200 cuando actualiza correctamente:

Paso 1: Simular datos válidos de actualización.

Paso 2: Mockear el servicio devolviendo resultado exitoso.

Paso 3: Ejecutar el controlador.

Resultado esperado: Devuelve 200 OK confirmando que la categoría fue actualizada.

Prueba 12 – actualizarCategoria 400 si no actualiza:

Paso 1: Simular intento de actualización sin cambios.

Paso 2: Ejecutar el controlador.

Paso 3: Revisar la respuesta.

Resultado esperado: Devuelve 400 Bad Request con mensaje “No se realizaron cambios”.

Prueba 13 – actualizarCategoria 500 en error servicio:

Paso 1: Simular error del servicio al actualizar.

Paso 2: Ejecutar el controlador.

Paso 3: Observar el resultado.

Resultado esperado: Devuelve 500 Internal Server Error con mensaje genérico.

Prueba 14 – eliminarCategoria 200 cuando elimina:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Simular ID válido y categoría existente.</p> <p>Paso 2: Mockear eliminación exitosa en el servicio.</p> <p>Paso 3: Ejecutar eliminarCategoria.</p> <p>Resultado esperado: Devuelve 200 OK confirmando la eliminación de la categoría.</p> <p>Prueba 15 – eliminarCategoria 404 cuando no existe:</p> <p>Paso 1: Simular ID inexistente.</p> <p>Paso 2: Mockear servicio para devolver 0 (sin registros eliminados).</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: Devuelve 404 Not Found con mensaje “Categoría no encontrada”.</p> <p>Prueba 16 – eliminarCategoria 500 en error:</p> <p>Paso 1: Simular error interno en la eliminación.</p> <p>Paso 2: Ejecutar el método.</p> <p>Paso 3: Observar la respuesta.</p> <p>Resultado esperado: Devuelve 500 Internal Server Error con mensaje de error genérico.</p>
Datos de salida - Resultado Esperado	<p>1 El sistema responde con los códigos HTTP correctos (200, 201, 400, 404, 409, 500) según cada escenario.</p> <p>2 Se validan correctamente las entradas, duplicados y conversiones de datos.</p>

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 56 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	3 Los errores se manejan de forma controlada sin excepciones no capturadas. 4 Las operaciones CRUD de categorías funcionan conforme al diseño del sistema		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El controlador gestionó correctamente las operaciones CRUD, validando entradas, manejando duplicados y controlando errores según los códigos esperados.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

6. CategoriaProcedimientosControllers.test

Validar el funcionamiento integral del **controlador de categorías de procedimientos**, verificando que gestione correctamente las operaciones CRUD (**listar, buscar, crear, actualizar y eliminar**) y que responda con los códigos HTTP apropiados

```
1 jest.mock('../services/CategoriaProcedimientosServices', () => ({
2   listarLasCategorias: jest.fn(),
3   buscarLaCategoria: jest.fn(),
4   crearLaCategoria: jest.fn(),
5   actualizarLaCategoria: jest.fn(),
6   eliminarLaCategoria: jest.fn(),
7 });
8
```

```
const mockReqRes = (overrides = {}) => {
  const req = { body: {}, params: {}, ...overrides };
  const res = [
    statusCode: 200,
    body: undefined,
    status(code) { this.statusCode = code; return this; },
    json(payload) { this.body = payload; return this; },
  ];
  return { req, res };
};
```

Prueba 1

```
test('listarCategorias -> 200', async () => {
  categoriaService.listarLasCategorias.mockResolvedValue([ { id: 1 } ]);
  const { req, res } = mockReqRes();
  await controller.listarCategorias(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual([ { id: 1 } ]);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 2: buscarCategoria 200 cuando existe

```
test('listarCategorias -> 500 en error', async () => {
  categoriaService.listarLasCategorias.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes();
  await controller.listarCategorías(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: 'Error al listar categorías' });
});
```

Prueba 3 : buscarCategoria 200 cuando existe

Verificar la búsqueda exitosa de una categoría por su ID.

```
test('buscarCategoria -> 200 cuando existe', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue({ id: 5 });
  const { req, res } = mockReqRes({ params: { id: 5 } });
  await controller.buscarCategoria(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ id: 5 });
});
```

Prueba4: buscarCategoria 404 cuando no existe

```
test('buscarCategoria -> 404 cuando no existe', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue(null);
  const { req, res } = mockReqRes({ params: { id: 5 } });
  await controller.buscarCategoria(req, res);
  expect(res.statusCode).toBe(404);
  expect(res.body).toEqual({ error: 'Categoría no encontrada' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 5: buscarCategoria 500 en error

```
test('buscarCategoria -> 500 en error', async () => {
  categoriaService.buscarLaCategoria.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes({ params: { id: 5 } });
  await controller.buscarCategoria(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: 'Error al buscar categoría' });
});
```

Prueba 6: crearCategoria → 201 con datos válidos

Validar creación correcta y conversión automática de "true" (string) a true (boolean).

```
test('crearCategoria -> 201 con body boolean string a boolean', async () => {
  categoriaService.crearLaCategoria.mockResolvedValue({ id: 10, nombre: 'Facial', estado: true });
  const { req, res } = mockReqRes({ body: { nombre: 'Facial', estado: 'true' } });
  await controller.crearCategoria(req, res);
  expect(res.statusCode).toBe(201);
  expect(res.body).toEqual({ id: 10, nombre: 'Facial', estado: true });
  expect(categoriaService.crearLaCategoria).toHaveBeenCalledWith({ nombre: 'Facial', estado: true });
});
```

Prueba 7: crearCategoria → 409 conflicto nombre duplicado

Comprobar manejo de duplicados.

```
test('crearCategoria -> 409 conflicto nombre', async () => {
  const e = new Error('dup'); e.status = 409;
  categoriaService.crearLaCategoria.mockRejectedValue(e);
  const { req, res } = mockReqRes({ body: { nombre: 'Facial' } });
  await controller.crearCategoria(req, res);
  expect(res.statusCode).toBe(409);
  expect(res.body).toEqual({ message: 'Ya existe una categoría con ese nombre' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 8: crearCategoria → 400 nombre requerido

```
test('crearCategoria -> 400 nombre requerido', async () => {
  const e = new Error('faltan datos');
  e.status = 400;
  categoriaService.crearLaCategoria.mockRejectedValue(e);
  const { req, res } = mockReqRes({ body: { estado: true } });
  await controller.crearCategoria(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ message: 'El nombre de la categoría es requerido' });
});
```

Prueba 9: crearCategoria → 500 error genérico

Verificar el manejo de errores no controlados.

```
test('crearCategoria -> 500 error genérico', async () => {
  const e = new Error('boom');
  categoriaService.crearLaCategoria.mockRejectedValue(e);
  const { req, res } = mockReqRes({ body: { nombre: 'Facial' } });
  await controller.crearCategoria(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ message: 'Hubo un error al crear la categoría', error: 'boom' });
});
```

Prueba 10: actualizarCategoria → 400 ID inválido

Evitar actualizaciones con IDs no numéricos.

```
test('actualizarCategoria -> 400 id inválido', async () => {
  const { req, res } = mockReqRes({ params: { id: 'abc' } });
  await controller.actualizarCategoria(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'ID inválido' });
});
```

Prueba 11: actualizarCategoria → 404 cuando no existe

```
test('actualizarCategoria -> 404 cuando no existe', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue(null);
  const { req, res } = mockReqRes({ params: { id: '5' } });
  await controller.actualizarCategoria(req, res);
  expect(res.statusCode).toBe(404);
  expect(res.body).toEqual({ error: 'Categoría no encontrada' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 12: actualizarCategoria → 200 cuando actualiza correctamente

```
test('actualizarCategoria -> 200 cuando actualiza', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue({ id: 5, estado: false });
  categoriaService.actualizarLaCategoria.mockResolvedValue([1]);
  const { req, res } = mockReqRes({ params: { id: '5' }, body: { estado: 'true' } });
  await controller.actualizarCategoria(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ mensaje: 'Categoría actualizada correctamente' });
  expect(categoriaService.actualizarLaCategoria).toHaveBeenCalledWith('5', { estado: true });
});
```

Prueba 13: actualizarCategoria → 400 si no actualiza

```
test('actualizarCategoria -> 400 si no actualiza', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue({ id: 5, estado: false });
  categoriaService.actualizarLaCategoria.mockResolvedValue([0]);
  const { req, res } = mockReqRes({ params: { id: '5' }, body: { estado: true } });
  await controller.actualizarCategoria(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'No se pudo actualizar la categoría' });
});
```

Prueba 14: actualizarCategoria → 500 en error servicio

```
test('actualizarCategoria -> 500 en error', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue({ id: 5 });
  categoriaService.actualizarLaCategoria.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes({ params: { id: '5' }, body: { estado: true } });
  await controller.actualizarCategoria(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({
    error: 'Error en el servidor al actualizar la categoría',
    detalle: 'x',
  });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 14: actualizarCategoria → 500 en error servicio

```
test('actualizarCategoria -> 500 en error', async () => {
  categoriaService.buscarLaCategoria.mockResolvedValue({ id: 5 });
  categoriaService.actualizarLaCategoria.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes({ params: { id: '5' }, body: { estado: true } });
  await controller.actualizarCategoria(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({
    error: 'Error en el servidor al actualizar la categoría',
    detalle: 'x',
  });
});
```

Prueba 15: eliminarCategoria → 200 cuando elimina

```
test('eliminarCategoria -> 200 cuando elimina', async () => {
  categoriaService.eliminarLaCategoria.mockResolvedValue(1);
  const { req, res } = mockReqRes({ params: { id: '8' } });
  await controller.eliminarCategoria(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ message: 'Categoría eliminada' });
});
```

Prueba 16: eliminarCategoria → 404 cuando no existe

```
test('eliminarCategoria -> 404 cuando no existe', async () => {
  categoriaService.eliminarLaCategoria.mockResolvedValue(0);
  const { req, res } = mockReqRes({ params: { id: '8' } });
  await controller.eliminarCategoria(req, res);
  expect(res.statusCode).toBe(404);
  expect(res.body).toEqual({ error: 'Categoría no encontrada' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 17: eliminarCategoria → 500 en error

```
test('eliminarCategoria -> 500 en error', async () => {
    categoriaService.eliminarLaCategoria.mockRejectedValue(new Error('x'));
    const { req, res } = mockReqRes({ params: { id: '8' } });
    await controller.eliminarCategoria(req, res);
    expect(res.statusCode).toBe(500);
    expect(res.body).toEqual({ error: 'Error al eliminar la categoría' });
});
```

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/CategoriaProcedimientosControllers.test.js
PASS  _tests_/CategoriaProcedimientosControllers.test.js
  Controlador de Categoría de Procedimientos
    ✓ ListarCategorías -> 200 (5 ms)
    ✗ ListarCategorías -> 500 en error (2 ms)
    ✗ buscarCategoria -> 200 cuando existe (1 ms)
    ✗ buscarCategoria -> 404 cuando no existe (1 ms)
    ✗ buscarCategoria -> 500 en error (1 ms)
    ✗ crearCategoria -> 201 con body boolean string a boolean (1 ms)
    ✗ crearCategoria -> 409 conflicto nombre
    ✗ crearCategoria -> 400 nombre requerido (1 ms)
    ✗ crearCategoria -> 500 error genérico (1 ms)
    ✗ actualizarCategoria -> 400 id inválido (1 ms)
    ✗ actualizarCategoria -> 404 cuando no existe (1 ms)
    ✗ actualizarCategoria -> 200 cuando actualiza (1 ms)
    ✗ actualizarCategoria -> 400 si no actualiza (1 ms)
    ✗ actualizarCategoria -> 500 en error (1 ms)
    ✗ eliminarCategoria -> 200 cuando elimina (1 ms)
    ✗ eliminarCategoria -> 404 cuando no existe (1 ms)
    ✗ eliminarCategoria -> 500 en error (1 ms)

Test Suites: 1 passed, 1 total
Tests:    17 passed, 17 total
Snapshots: 0 total
Time:    0.817 s, estimated 1 s
Run all test suites matching _tests_/CategoriaProcedimientosControllers.test.js/i.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización:

Duración de la prueba: 5 min



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

15-09-2025

Requerimiento Funcional de la prueba	El sistema debe garantizar la correcta ejecución de los servicios encargados de listar, buscar, crear, actualizar y eliminar categorías de procedimientos, asegurando validaciones de datos, manejo de duplicados, propagación de errores de base de datos y ejecución correcta de funciones utilitarias como normalizeNombre.
Objetivo	Validar el comportamiento integral del módulo CategoriaProcedimientosServices, comprobando que sus métodos gestionen adecuadamente todos los escenarios posibles: consultas exitosas, errores de validación, duplicados, fallas de base de datos y normalización correcta de nombres.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Servicios: listarLasCategorias, buscarLaCategoria, crearLaCategoria, actualizarLaCategoria, eliminarLaCategoria, normalizeNombre. Modelos Sequelize simulados con jest.fn(). Datos de entrada variados (nombre válido, duplicado, vacío, nulo, ID inexistente). Ambiente: Node.js + Jest.
Procedimiento de Prueba	Prueba 1 – listarLasCategorias devuelve lista: Paso 1: Mockear el modelo findAll para devolver una lista simulada. Paso 2: Ejecutar el servicio listarLasCategorias. Paso 3: Observar el resultado. Resultado esperado: El servicio debe devolver la lista de categorías obtenida del modelo.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 2 – listarLasCategorias lanza error si falla DB:</p> <p>Paso 1: Simular que findAll lanza un error de base de datos.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El servicio debe propagar el error sin modificar el mensaje original.</p> <p>Prueba 3 – buscarLaCategoria devuelve categoría por ID:</p> <p>Paso 1: Mockear findByPk para devolver una categoría válida.</p> <p>Paso 2: Ejecutar el servicio con un ID existente.</p> <p>Paso 3: Validar la respuesta.</p> <p>Resultado esperado: Devuelve la categoría correspondiente al ID solicitado.</p> <p>Prueba 4 – buscarLaCategoria lanza error si falla:</p> <p>Paso 1: Simular fallo en findByPk.</p> <p>Paso 2: Ejecutar buscarLaCategoria.</p> <p>Paso 3: Revisar el manejo del error.</p> <p>Resultado esperado: El servicio debe propagar el error sin alterar el formato.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 5 – crearLaCategoria lanza error si nombre vacío:</p> <p>Paso 1: Enviar un objeto con campo nombre vacío o espacios.</p> <p>Paso 2: Ejecutar el servicio crearLaCategoria.</p> <p>Paso 3: Validar la respuesta.</p> <p>Resultado esperado: El servicio debe lanzar error “El nombre de la categoría es obligatorio”.</p> <p>Prueba 6 – crearLaCategoria lanza error si ya existe:</p> <p>Paso 1: Simular que findOne devuelve una categoría con el mismo nombre.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Observar la respuesta.</p> <p>Resultado esperado: El servicio debe rechazar la creación con error “La categoría ya existe”.</p> <p>Prueba 7 – crearLaCategoria crea una nueva categoría:</p> <p>Paso 1: Mockear create para devolver una categoría creada exitosamente.</p> <p>Paso 2: Ejecutar crearLaCategoria con nombre válido.</p> <p>Paso 3: Observar el resultado.</p> <p>Resultado esperado: Devuelve la nueva categoría con el nombre normalizado.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 8 – crearLaCategoria lanza error genérico si falla DB:</p> <p>Paso 1: Simular error inesperado en el método create.</p> <p>Paso 2: Ejecutar crearLaCategoria.</p> <p>Paso 3: Revisar el manejo del error.</p> <p>Resultado esperado: El servicio debe lanzar error genérico sin interrumpir la ejecución general.</p> <p>Prueba 9 – actualizarLaCategoria lanza error si nombre vacío:</p> <p>Paso 1: Simular un nombre vacío en la actualización.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Observar la respuesta.</p> <p>Resultado esperado: El servicio debe devolver error “El nombre no puede estar vacío”.</p> <p>Prueba 10 – actualizarLaCategoria lanza error si nombre duplicado:</p> <p>Paso 1: Simular que existe otra categoría con el mismo nombre.</p> <p>Paso 2: Ejecutar actualizarLaCategoria.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El servicio debe rechazar la actualización con error “El nombre ya existe”.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 11 – actualizarLaCategoria actualiza correctamente:</p> <p>Paso 1: Simular un ID válido y nombre nuevo.</p> <p>Paso 2: Mockear update devolviendo éxito.</p> <p>Paso 3: Ejecutar el servicio.</p> <p>Resultado esperado: El servicio debe devolver confirmación de actualización exitosa.</p> <p>Prueba 12 – actualizarLaCategoria lanza error si falla DB:</p> <p>Paso 1: Simular fallo interno en el método update.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El servicio debe propagar el error generado por la base de datos.</p> <p>Prueba 13 – eliminarLaCategoria elimina correctamente:</p> <p>Paso 1: Mockear el método destroy devolviendo 1 (eliminación exitosa).</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Validar el resultado.</p> <p>Resultado esperado: El servicio debe confirmar la eliminación de la categoría.</p> <p>Prueba 14 – eliminarLaCategoria lanza error si falla:</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Simular error interno en destroy.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Paso 3: Revisar la respuesta.</p> <p>Resultado esperado: El servicio debe lanzar error genérico sin romper la ejecución.</p> <p> </p> <p>Prueba 15 – normalizeNombre devuelve cadena vacía si valor no es string:</p> <p>Paso 1: Llamar la función normalizeNombre con un valor no tipo texto (null, número, etc.).</p> <p>Paso 2: Observar la salida.</p> <p>Resultado esperado: La función debe devolver \"\" (cadena vacía).</p> <p> </p> <p>Prueba 16 – actualizarLaCategoria funciona cuando datos no tienen nombre:</p> <p>Paso 1: Simular actualización parcial (solo campo descripcion).</p> <p>Paso 2: Ejecutar actualizarLaCategoria.</p> <p>Paso 3: Verificar el flujo.</p> <p>Resultado esperado: El servicio debe realizar la actualización correctamente sin requerir el campo nombre.</p>
Datos de salida - Resultado Esperado	Las funciones del servicio interactúan correctamente con los modelos Sequelize.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	1 Los errores y duplicados son gestionados apropiadamente. 2 Las validaciones de nombre se aplican de forma coherente. 3 La función normalizeNombre se comporta correctamente con distintos tipos de entrada. 4 Las operaciones CRUD funcionan sin dependencias externas a la base de datos.		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – Los servicios de categorías funcionaron correctamente, aplicando las validaciones, normalizaciones y propagación de errores esperadas en cada escenario.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

7. CategoriaProcedimientosServices.test.js

Validar la funcionalidad de los servicios encargados de **listar, buscar, crear, actualizar y eliminar categorías de procedimientos**, asegurando el manejo adecuado de validaciones, duplicados y errores de base de datos. Además, se validan funciones utilitarias como la normalización de nombres (normalizeNombre).

Entorno de pruebas

```
jest.mock("../models", () => ({
  categoriaprocedimientos: {
    findAll: jest.fn(),
    findByPk: jest.fn(),
    findOne: jest.fn(),
    create: jest.fn(),
    update: jest.fn(),
    destroy: jest.fn(),
  },
  procedimientos: {},
}));
```



```
const {
  service: svc,
  normalizeNombre,
} = require("../services/CategoriaProcedimientosServices");

const models = require("../models");
```

Prueba 1: listarLasCategorias devuelve lista

Verifica que el servicio llame correctamente a findAll del modelo y retorne el resultado esperado.

```
test("listarLasCategorias devuelve lista", async () => {
  const data = [{ id: 1, nombre: "Faciales" }];
  models.categoriaprocedimientos.findAll.mockResolvedValue(data);
  const res = await svc.listarLasCategorias();
  expect(models.categoriaprocedimientos.findAll).toHaveBeenCalled();
  expect(res).toEqual(data);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 2: listarLasCategorias lanza error si falla DB

Confirma que el servicio propague un error cuando el modelo falla en la consulta.

```
test("listarLasCategorias lanza error si falla DB", async () => {
  models.categoriaprocedimientos.findAll.mockRejectedValue(
    new Error("DB error")
  );
  await expect(svc.listarLasCategorias()).rejects.toThrow("DB error");
});
```

Prueba 3: buscarLaCategoria devuelve categoría por id

Valida que findByPk sea llamado con el ID correcto y devuelva la categoría encontrada.

```
test("buscarLaCategoria devuelve categoría por id", async () => {
  models.categoriaprocedimientos.findByPk.mockResolvedValue({
    id: 2,
    nombre: "Corporal",
  });
  const res = await svc.buscarLaCategoria(2);
  expect(models.categoriaprocedimientos.findByPk).toHaveBeenCalledWith(
    2,
    expect.objectContaining({ include: expect.any(Array) })
  );
  expect(res).toEqual({ id: 2, nombre: "Corporal" });
});
```

Prueba 4: buscarLaCategoria lanza error si falla

Simula un fallo en el método findByPk y comprueba que el servicio propague el error.

```
test("buscarLaCategoria lanza error si falla", async () => {
  models.categoriaprocedimientos.findByPk.mockRejectedValue(
    new Error("fail")
  );
  await expect(svc.buscarLaCategoria(1)).rejects.toThrow("fail");
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 5: crearLaCategoria lanza error si nombre vacío

Valida que el servicio no permita crear una categoría sin nombre válido.

```
test("crearLaCategoria lanza error si nombre vacío", async () => {
    await expect(svc.crearLaCategoria({ nombre: " " })).rejects.toThrow(
        /requerido/i
    );
});
```

Prueba 6: crearLaCategoria lanza error si ya existe

Evita duplicar categorías con el mismo nombre.

```
test("crearLaCategoria lanza error si ya existe", async () => [
    models.categoriaprocedimientos.findOne.mockResolvedValue({ id: 1 });
    await expect(svc.crearLaCategoria({ nombre: "Faciales" })).rejects.toThrow(
        /ya existe/i
    );
]);
```

Prueba 7: crearLaCategoria crea una nueva categoría

Verifica la creación exitosa de una nueva categoría y la normalización del nombre.

```
test("crearLaCategoria crea una nueva categoría", async () => [
    models.categoriaprocedimientos.findOne.mockResolvedValue(null);
    models.categoriaprocedimientos.create.mockResolvedValue({
        id: 10,
        nombre: "Nueva",
    });
    const res = await svc.crearLaCategoria({ nombre: " Nueva " });
    expect(models.categoriaprocedimientos.create).toHaveBeenCalledWith({
        nombre: "Nueva",
    });
    expect(res).toEqual({ id: 10, nombre: "Nueva" });
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 8: crearLaCategoria lanza error genérico si falla DB

Valida que los errores inesperados de base de datos sean propagados correctamente.

```
test("crearLaCategoria lanza error genérico si falla DB", async () => {
  models.categoriacprocedimientos.findOne.mockRejectedValue(
    new Error("fallo DB")
  );
  await expect(svc.crearLaCategoria({ nombre: "Test" })).rejects.toThrow(
    "fallo DB"
  );
});
```

Prueba 9: actualizarLaCategoria lanza error si nombre vacío

Evita actualizar categorías con nombre vacío o espacios.

```
test("actualizarLaCategoria lanza error si nombre vacío", async () => {
  await expect(
    svc.actualizarLaCategoria(1, { nombre: " " })
  ).rejects.toThrow(/requerido/i);
});
```

Prueba 10: actualizarLaCategoria lanza error si nombre duplicado

Impide actualizar una categoría con un nombre ya existente.

```
test("actualizarLaCategoria lanza error si nombre duplicado", async () => {
  models.categoriacprocedimientos.findOne.mockResolvedValue({ id: 9 });
  await expect(
    svc.actualizarLaCategoria(1, { nombre: "Duplicada" })
  ).rejects.toThrow(/ya existe/i);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 11: actualizarLaCategoria actualiza correctamente

Simula una actualización exitosa de la categoría.

```
test("actualizarLaCategoria actualiza correctamente", async () => {
  models.categoriaprocedimientos.findOne.mockResolvedValue(null);
  models.categoriaprocedimientos.update.mockResolvedValue([1]);
  const res = await svc.actualizarLaCategoria(5, { nombre: "Ok" });
  expect(models.categoriaprocedimientos.update).toHaveBeenCalledWith(
    { nombre: "Ok" },
    { where: { id: 5 } }
  );
  expect(res).toEqual([1]);
});
```

Prueba 12: actualizarLaCategoria lanza error si falla DB

Confirma que los errores del método update se propagan correctamente.

```
test("actualizarLaCategoria lanza error si falla DB", async () => {
  models.categoriaprocedimientos.update.mockRejectedValue(
    new Error("Fallo update")
  );
  await expect(
    svc.actualizarLaCategoria(1, { nombre: "Nuevo" })
  ).rejects.toThrow("Fallo update");
});
```

Prueba 13: eliminarLaCategoria elimina correctamente

Verifica la eliminación exitosa de una categoría existente.

```
test("eliminarLaCategoria elimina correctamente", async () => {
  models.categoriaprocedimientos.destroy.mockResolvedValue(1);
  const res = await svc.eliminarLaCategoria(7);
  expect(models.categoriaprocedimientos.destroy).toHaveBeenCalledWith(
    { where: { id: 7 } }
  );
  expect(res).toBe(1);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 14: eliminarLaCategoria lanza error si falla
Valida el manejo de errores durante la eliminación.

```
test("eliminarLaCategoria lanza error si falla", async () => {
  models.categoriacprocedimientos.destroy.mockRejectedValue(new Error("fail"));
  await expect(svc.eliminarLaCategoria(1)).rejects.toThrow("fail");
});
```

Prueba 15: normalizeNombre devuelve cadena vacía si valor no es string

Verifica que la función normalizeNombre retorne "" cuando el argumento no sea texto.

```
test("normalizeNombre devuelve cadena vacía si el valor no es string", () => {
  expect(normalizeNombre(null)).toBe("");
  expect(normalizeNombre(123)).toBe("");
});
```

Prueba 16: actualizarLaCategoria funciona cuando datos no tiene nombre

Confirma que la actualización funciona incluso si el campo nombre no está presente (por ejemplo, solo cambia descripción).

```
test("actualizarLaCategoria funciona cuando datos no tiene nombre", async () => {
  models.categoriacprocedimientos.update.mockResolvedValue([1]);
  const res = await svc.actualizarLaCategoria(9, { descripcion: "extra" });
  expect(models.categoriacprocedimientos.update).toHaveBeenCalledWith(
    { descripcion: "extra" },
    { where: { id: 9 } }
  );
  expect(res).toEqual([1]);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests/_CategoriaProcedimientosServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests/_CategoriaProcedimientosServices.test.js

PASS  tests/_CategoriaProcedimientosServices.test.js
  Servicios de Categoría de Procedimientos
    listarLasCategorias devuelve lista (6 ms)
    listarLasCategorias lanza error si falla DB (18 ms)
    buscarLaCategoria devuelve categoría por id (2 ms)
    buscarLaCategoria lanza error si falla (1 ms)
    crearLaCategoria lanza error si nombre vacío (9 ms)
    crearLaCategoria lanza error si ya existe (2 ms)
    crearLaCategoria crea una nueva categoría (2 ms)
    crearLaCategoria lanza error genérico si falla DB (1 ms)
    actualizarLaCategoria lanza error si nombre vacío (2 ms)
    actualizarLaCategoria lanza error si nombre duplicado (1 ms)
    actualizarLaCategoria actualiza correctamente (1 ms)
    actualizarLaCategoria lanza error si falla DB (1 ms)
    eliminarLaCategoria elimina correctamente (1 ms)
    eliminarLaCategoria lanza error si falla (1 ms)
    normalizeNombre devuelve cadena vacía si el valor no es string
    actualizarLaCategoria funciona cuando datos no tiene nombre

Test Suites: 1 passed, 1 total
Tests:    16 passed, 16 total
Snapshots: 0 total
Time:    0.782 s, estimated 1 s
Run all test suites matching _tests/_CategoriaProcedimientosServices.test.js/i.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	
Requerimiento Funcional de la prueba	El sistema debe permitir la correcta gestión de solicitudes al endpoint consultarchat a través del controlador de chat, validando la entrada del usuario, comunicándose con el servicio correspondiente y manejando apropiadamente los errores internos para garantizar la disponibilidad y consistencia de las respuestas.
Objetivo	Verificar que el controlador ChatControllers procese correctamente las solicitudes del usuario, validando que los mensajes enviados sean válidos, que la comunicación con el servicio se realice sin errores y que se gestionen correctamente las fallas internas del servicio o los casos de entrada inválida.
Tipo de Prueba	Prueba Unitaria (Jest)



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Datos de entrada de la prueba	Endpoint: POST /consultarchat. Campos esperados: { mensaje: "texto" }. Servicios mockeados con jest.fn(). Escenarios probados: entrada inválida, respuesta exitosa, error del servicio. Ambiente: Node.js + Express + Jest.
Procedimiento de Prueba	Prueba 1 – Retorna 400 si el mensaje es inválido: Paso 1: Simular una solicitud sin el campo mensaje en el cuerpo (req.body). Paso 2: Ejecutar el método del controlador consultarChat. Paso 3: Observar la respuesta. Resultado esperado: El controlador debe devolver 400 Bad Request con mensaje “El campo mensaje es obligatorio” o equivalente, indicando validación fallida. Prueba 2 – Retorna 200 con respuesta del servicio: Paso 1: Simular una solicitud válida con req.body = { mensaje: "Hola" }. Paso 2: Mockear el servicio consultarchat para devolver una respuesta de texto. Paso 3: Ejecutar el método del controlador. Resultado esperado: El controlador llama correctamente al servicio con el mensaje recibido.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Devuelve 200 OK con la respuesta del servicio en formato JSON.</p> <p>Prueba 3 – Retorna 500 cuando el servicio falla:</p> <p>Paso 1: Simular un error interno del servicio (por ejemplo, error de conexión o timeout).</p> <p>Paso 2: Ejecutar el controlador con el mensaje válido.</p> <p>Paso 3: Revisar la respuesta del sistema.</p> <p>Resultado esperado: El controlador debe devolver 500 Internal Server Error con mensaje “Error al procesar la solicitud del chat” o equivalente.</p>		
Datos de salida - Resultado Esperado	1 El controlador valida correctamente la presencia del campo mensaje. 2 Se comunica exitosamente con el servicio consultarchat cuando la entrada es válida. 3 Los errores internos del servicio se gestionan adecuadamente sin afectar la estabilidad del sistema. 4 Las respuestas devuelven los códigos HTTP correctos (400, 200, 500) según el escenario.		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El controlador de chat gestionó correctamente las validaciones, la comunicación con el servicio y el manejo de errores internos, cumpliendo con los códigos esperados en todos los casos.	Si(<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

8. ChatControllers.test.js

Validar que el **controlador de chat** gestione correctamente la solicitud del usuario al endpoint consultarchat, incluyendo:

- Validación de entrada (mensaje presente y válido).
- Llamado al servicio de chat.
- Manejo de errores internos del servicio.

Configuración del entorno de prueba

```
jest.mock('../services/ChatServices', () => ({
  consultarr: jest.fn(),
}));

const ChatServices = require('../services/ChatServices');
const ChatController = require('../controllers/ChatControllers');

const mockReqRes = (overrides = {}) => {
  const req = { body: {}, params: {}, ...overrides };
  const res = {
    statusCode: 200,
    body: undefined,
    status(code) { this.statusCode = code; return this; },
    json(payload) { this.body = payload; return this; },
  };
  return { req, res };
};
```

Prueba 1: Retorna 400 si el mensaje es inválido

Verifica que el controlador valide correctamente la entrada del usuario.

Si no se proporciona un mensaje en el cuerpo (req.body), debe responder con error.

```
test('400 si mensaje inválido', async () => {
  const { req, res } = mockReqRes({ body: {} });
  await ChatController.consultarchat(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ ok: false, mensaje: 'Mensaje inválido' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 2: Retorna 200 con respuesta del servicio

Valida el flujo exitoso: cuando el mensaje es válido y el servicio consultarr devuelve una respuesta.

Comprueba que el controlador:

- Llame correctamente al servicio con el mensaje.
- Devuelva el resultado esperado con código **200**.

```
test('200 con respuesta del servicio', async () => [
  ChatServices.consultarr.mockResolvedValue({ ok: true, respuesta: 'hola', items: [] });
  const { req, res } = mockReqRes({ body: { mensaje: 'botox' } });
  await ChatController.consultarchat(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ ok: true, respuesta: 'hola', items: [] });
  expect(ChatServices.consultarr).toHaveBeenCalledWith('botox');
]);
```

Prueba 3: Retorna 500 cuando el servicio falla

Simula una falla del servicio de chat (por ejemplo, error de conexión o timeout) y valida que el controlador maneje el error correctamente.

```
test('500 cuando el servicio falla', async () => [
  ChatServices.consultarr.mockRejectedValue(new Error('down'));
  const { req, res } = mockReqRes({ body: { mensaje: 'botox' } });
  await ChatController.consultarchat(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ ok: false, mensaje: 'Error en el chat' });
]);
```

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/_ChatControllers.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/_ChatControllers.test.js

PASS  tests/_ChatControllers.test.js
  Controlador de Chat - consultarchat
    ✓ 400 si mensaje inválido (5 ms)
    ✓ 200 con respuesta del servicio (3 ms)
    ✓ 500 cuando el servicio falla (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.667 s, estimated 1 s
Run all test suites matching _tests_/_ChatControllers.test.js/1.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe garantizar el correcto funcionamiento de los servicios del módulo Chat, encargados de procesar los mensajes de los usuarios, normalizar texto, extraer palabras clave, buscar procedimientos relacionados en la base de datos y generar respuestas coherentes según el contexto detectado.
Objetivo	Validar el comportamiento integral de los servicios del chat, asegurando que las funciones normalizar, extraerPalabrasClaveBasico, buscarProcedimientosPorKeywords y consultarr ejecuten correctamente sus procesos internos: limpieza de texto, filtrado de palabras irrelevantes, consultas de procedimientos y generación de respuestas coherentes ante distintos escenarios.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Servicios probados: normalizar, extraerPalabrasClaveBasico, buscarProcedimientosPorKeywords, consultarr. Datos de entrada: texto con tildes, palabras comunes, cadenas vacías y arreglos de palabras clave. Modelos Sequelize simulados (procedimientos, categorias). Ambiente: Node.js + Jest.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Procedimiento de Prueba	<p>Prueba 1 – normalizar convierte texto a minúsculas y quita tildes:</p> <p>Paso 1: Enviar una cadena con mayúsculas y tildes (por ejemplo, "Ácido Hialurónico Facial").</p> <p>Paso 2: Ejecutar la función normalizar.</p> <p>Paso 3: Observar la salida.</p> <p>Resultado esperado: Devuelve "acido hialuronico facial", sin tildes y en minúsculas.</p> <p>Prueba 2 – normalizar devuelve cadena vacía si texto no se pasa:</p> <p>Paso 1: Ejecutar normalizar sin argumentos.</p> <p>Paso 2: Observar el resultado.</p> <p>Resultado esperado: Retorna "" (cadena vacía) sin lanzar errores.</p> <p>Prueba 3 – extraerPalabrasClaveBasico usa normalizar y filtra correctamente:</p> <p>Paso 1: Simular texto con palabras comunes ("Procedimiento de la piel").</p> <p>Paso 2: Ejecutar extraerPalabrasClaveBasico.</p> <p>Paso 3: Validar la salida.</p> <p>Resultado esperado: Llama internamente a normalizar y devuelve solo las palabras relevantes: ["procedimiento", "piel"].</p>
--------------------------------	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 4 – extraerPalabrasClaveBasico devuelve [] si texto vacío:</p> <p>Paso 1: Ejecutar extraerPalabrasClaveBasico("") .</p> <p>Paso 2: Revisar la salida.</p> <p>Resultado esperado: Devuelve [] (arreglo vacío).</p> <p>Prueba 5 – extraerPalabrasClaveBasico sin argumento devuelve []:</p> <p>Paso 1: Ejecutar la función sin parámetros.</p> <p>Paso 2: Observar el resultado.</p> <p>Resultado esperado: Devuelve [] sin generar excepciones.</p> <p>Prueba 6 – extraerPalabrasClaveBasico devuelve [] si el texto solo contiene palabras vacías:</p> <p>Paso 1: Simular texto compuesto únicamente por “stopwords” (“de la el los las”).</p> <p>Paso 2: Ejecutar extraerPalabrasClaveBasico.</p> <p>Resultado esperado: Devuelve [] filtrando todas las palabras irrelevantes.</p> <p>Prueba 7 – buscarProcedimientosPorKeywords devuelve [] si lista vacía o no es array:</p> <p>Paso 1: Ejecutar buscarProcedimientosPorKeywords con entrada vacía o tipo string.</p> <p>Paso 2: Observar la respuesta.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: Devuelve [] sin llamar al modelo findAll.</p> <p>Prueba 8 – buscarProcedimientosPorKeywords maneja ambos casos de validación del parámetro:</p> <p>Paso 1: Probar entradas de tipo string y arrays vacíos.</p> <p>Paso 2: Observar la salida.</p> <p>Resultado esperado: En ambos casos, retorna [] asegurando consistencia de validación.</p> <p>Prueba 9 – buscarProcedimientosPorKeywords sin argumentos retorna [] y no consulta:</p> <p>Paso 1: Ejecutar la función sin argumentos.</p> <p>Paso 2: Validar si el modelo findAll fue llamado.</p> <p>Resultado esperado: Retorna [] y no ejecuta consultas al modelo.</p> <p>Prueba 10 – buscarProcedimientosPorKeywords busca con LIKE y mapea resultados:</p> <p>Paso 1: Simular que findAll devuelve una lista de procedimientos y algunas categorías nulas.</p> <p>Paso 2: Ejecutar el método.</p> <p>Paso 3: Validar el mapeo.</p> <p>Resultado esperado: Retorna lista de objetos con nombre, descripción y categoría correctamente mapeados, asignando "Sin categoría" si es nula.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 11 – consultarr devuelve mensaje si no hay palabras clave:

Paso 1: Simular mensaje sin palabras relevantes ("de la el").

Paso 2: Ejecutar consultarr.

Paso 3: Observar la respuesta.

Resultado esperado: Devuelve mensaje informativo tipo "No se identificaron términos relevantes para la búsqueda."

Prueba 12 – consultarr devuelve mensaje si no encuentra coincidencias:

Paso 1: Simular texto con palabras clave válidas pero sin resultados.

Paso 2: Mockear buscarProcedimientosPorKeywords para devolver [].

Paso 3: Ejecutar consultarr.

Resultado esperado: Retorna mensaje tipo "No se encontraron procedimientos relacionados con tu consulta."

Prueba 13 – consultarr devuelve resultados cuando hay coincidencias:

Paso 1: Simular texto con palabras clave válidas y resultados en base de datos.

Paso 2: Mockear buscarProcedimientosPorKeywords para devolver lista de procedimientos.

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 87 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación:	1/02/2025	Elaborado por: Instructores área Software
Nombre del Documento: Formato para la construcción del Manual de Usuario				

	<p>Paso 3: Ejecutar consultarr.</p> <p>Resultado esperado: Devuelve 200 OK con lista de procedimientos y respuesta coherente al mensaje del usuario.</p>
Datos de salida - Resultado Esperado	<p>1 Las funciones normalizar y extraerPalabrasClaveBasico procesan correctamente los textos.</p> <p>2 Se eliminan tildes, mayúsculas y palabras vacías.</p> <p>3 Las consultas con buscarProcedimientosPorKeywords solo se ejecutan con datos válidos.</p> <p>4 consultarr responde correctamente según los escenarios: sin palabras clave, sin resultados o con coincidencias.</p> <p>5 No se generan errores no controlados durante la ejecución.</p>
Resultado Obtenido Prueba Exitosa	<p>Prueba exitosa – Los servicios de chat procesaron, filtraron y respondieron correctamente en todos los escenarios simulados, validando entradas y generando respuestas coherentes.</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

9. ChatServices.test.js

Validar la funcionalidad de los servicios de chat que permiten:

- Normalizar texto (eliminar tildes y pasar a minúsculas).
- Extraer palabras clave del mensaje del usuario.
- Buscar procedimientos según las palabras clave detectadas.
- Retornar respuestas coherentes en el flujo consultarr ante diferentes escenarios.

Configuración de entornos de prueba

```
jest.mock("../models", () => ({
  procedimientos: { findAll: jest.fn() },
  categoriaprocedimientos: {}
}));

const { procedimientos } = require("../models");
const ChatServices = require("../services/ChatServices");
```

Prueba 1: normalizar convierte texto a minúsculas y quita tildes

Verifica que el texto se procese correctamente eliminando tildes y convirtiendo todo a minúsculas.

```
test("normalizar convierte texto a minúsculas y quita tildes", async () => [
  const res = await ChatServices.normalizar("Árbol ELÉCTRICO");
  expect(res).toBe("arbol electrico");
]);
```

Prueba 2: normalizar devuelve cadena vacía si texto no se pasa

Valida que si no se envía ningún texto, la función retorne una cadena vacía sin errores.

```
test("normalizar devuelve cadena vacía si texto no se pasa", async () => [
  const res = await ChatServices.normalizar();
  expect(res).toBe("");
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3: extraerPalabrasClaveBasico usa normalizar y filtra correctamente
Comprueba que la función llame a normalizar internamente y elimine las “stopwords”
(palabras comunes sin valor semántico como *de*, *la*, *el*).

```
test("extraerPalabrasClaveBasico usa normalizar y filtra correctamente", async () => {
  const spyNorm = jest.spyOn(ChatServices, "normalizar");
  const palabras = await ChatServices.extraerPalabrasClaveBasico("Masaje facial con crema");
  expect(spyNorm).toHaveBeenCalledWith("Masaje facial con crema");
  expect(Array.isArray(palabras)).toBe(true);
  expect(palabras.length).toBeGreaterThan(0);
});
```

Prueba 4: extraerPalabrasClaveBasico devuelve [] si texto vacío
Verifica que el método retorne un arreglo vacío cuando se pasa una cadena vacía.

```
test("extraerPalabrasClaveBasico devuelve [] si texto vacío", async () => {
  const resultado = await ChatServices.extraerPalabrasClaveBasico("");
  expect(resultado).toEqual([]);
});
```

Prueba 5: extraerPalabrasClaveBasico sin argumento devuelve []
Asegura que la función retorne [] si no se pasa ningún argumento.

```
test("extraerPalabrasClaveBasico sin argumento devuelve []", async () => {
  const resultado = await ChatServices.extraerPalabrasClaveBasico();
  expect(resultado).toEqual([]);
});
```

Prueba 6: extraerPalabrasClaveBasico devuelve [] si el texto solo contiene palabras vacías
Confirma que las “stopwords” se filtren correctamente, devolviendo un arreglo vacío.

```
test("extraerPalabrasClaveBasico devuelve [] si el texto solo contiene palabras vacías", async () => {
  const resultado = await ChatServices.extraerPalabrasClaveBasico("de la y el o a los");
  expect(resultado).toEqual([]);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 7: buscarProcedimientosPorKeywords devuelve [] si lista vacía o no es array
Valida que el servicio retorne [] cuando la entrada no es un arreglo o es vacío, evitando consultas innecesarias al modelo.

```
test("buscarProcedimientosPorKeywords devuelve [] si lista vacía o no es array", async () => {
  const r1 = await ChatServices.buscarProcedimientosPorKeywords(null);
  const r2 = await ChatServices.buscarProcedimientosPorKeywords([]);
  expect(r1).toEqual([]);
  expect(r2).toEqual([]);
  expect(procedimientos.findAll).not.toHaveBeenCalled();
});
```

Prueba 8: buscarProcedimientosPorKeywords maneja ambos casos de validación del parámetro

Prueba redundante para asegurar que tanto las entradas tipo string como arrays vacíos produzcan el mismo resultado ([]).

```
test("buscarProcedimientosPorKeywords maneja ambos casos de validación del parámetro", async () => {
  const sinArray = await ChatServices.buscarProcedimientosPorKeywords("texto");
  const vacio = await ChatServices.buscarProcedimientosPorKeywords([]);
  expect(sinArray).toEqual([]);
  expect(vacio).toEqual([]);
});
```

Prueba 9: **buscarProcedimientosPorKeywords sin argumentos retorna [] y no consulta**

Garantiza que si no se pasa ningún argumento, el método devuelva un arreglo vacío y **no llame** al modelo findAll.

```
test("buscarProcedimientosPorKeywords sin argumentos retorna [] y no consulta", async () => {
  const res = await ChatServices.buscarProcedimientosPorKeywords();
  expect(res).toEqual([]);
  expect(procedimientos.findAll).not.toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 10: buscarProcedimientosPorKeywords busca con like y mapea resultados

Simula una búsqueda exitosa con resultados en base de datos.

Verifica que el método llame al modelo, mapee correctamente los campos y maneje los casos donde la categoría es nula.

```
test("buscarProcedimientosPorKeywords busca con like y mapea resultados", async () => {
  procedimientos.findAll.mockResolvedValue([
    {
      id: 1,
      nombre: "Limpieza facial",
      descripcion: "Tratamiento de piel",
      precio: 10000,
      imagen: "foto.jpg",
      categoria: { nombre: "Facial" },
    },
    {
      id: 2,
      nombre: "Masaje corporal",
      descripcion: "Relajante",
      precio: 200,
      imagen: "img.jpg",
      categoria: null,
    },
  ]);
  const resultado = await ChatServices.buscarProcedimientosPorKeywords(["facial", "piel"], 10);
  expect(procedimientos.findAll).toHaveBeenCalled();
  expect(resultado.length).toBe(2);
  expect(resultado[0]).toMatchObject({
    id: 1,
    nombre: "Limpieza facial",
    categoria: "Facial",
  });
  expect(resultado[1].categoria).toBe("Sin categoría");
});
```

Prueba 11: consultarr devuelve mensaje si no hay palabras clave

Simula un mensaje sin palabras clave relevantes (solo stopwords) y verifica que devuelva una respuesta informativa, no un error.

```
test("consultarr devuelve mensaje si no hay palabras clave", async () => {
  jest.spyOn(ChatServices, "extraerPalabrasClaveBasico").mockResolvedValueOnce([]);
  const res = await ChatServices.consultarr("de la y el");
  expect(res.ok).toBe(true);
  expect(res.items).toEqual([]);
  expect(res.respuesta).toMatch(/No pudo identificar palabras clave/);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 12: consultarr devuelve mensaje si no encuentra coincidencias

Comprueba el caso en el que hay palabras clave, pero no se encuentran procedimientos en la base de datos.

```
test("consultarr devuelve mensaje si no encuentra coincidencias", async () => {
  jest.spyOn(ChatServices, "extraerPalabrasClaveBasico").mockResolvedValueOnce(["facial"]);
  jest.spyOn(ChatServices, "buscarProcedimientosPorKeywords").mockResolvedValueOnce([]);
  const res = await ChatServices.consultarr("facial");
  expect(res.respuesta).toMatch(/No encontré coincidencias/);
});
```

Prueba 13: consultarr devuelve resultados cuando hay coincidencias

Valida el flujo exitoso cuando el mensaje contiene palabras clave y se encuentran procedimientos relacionados.

```
test("consultarr devuelve resultados cuando hay coincidencias", async () => {
  jest.spyOn(ChatServices, "extraerPalabrasClaveBasico").mockResolvedValueOnce(["facial"]);
  jest.spyOn(ChatServices, "buscarProcedimientosPorKeywords").mockResolvedValueOnce([
    { id: 1, nombre: "Facial básico" },
  ]);
  const res = await ChatServices.consultarr("facial");
  expect(res.ok).toBe(true);
  expect(res.items.length).toBe(1);
  expect(res.respuesta).toMatch(/Estos procedimientos/);
});
```

Resultado:

```
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests/_ChatServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests/_ChatServices.test.js
PASS  tests/_ChatServices.test.js
  Servicios de Chat
    ✓ normalizar convierte texto a minúsculas y quita tildes (4 ms)
    ✓ normalizar devuelve cadena vacía si texto no se pasa (1 ms)
    ✓ extraerPalabrasClaveBasico usa normalizar y filtra correctamente (12 ms)
    ✓ extraerPalabrasClaveBasico devuelve [] si texto vacío (1 ms)
    ✓ extraerPalabrasClaveBasico sin argumento devuelve [] (1 ms)
    ✓ extraerPalabrasClaveBasico devuelve [] si el texto solo contiene palabras vacías (1 ms)
    ✓ buscarProcedimientosPorKeywords devuelve [] si lista vacía o no es array (2 ms)
    ✓ buscarProcedimientosPorKeywords maneja ambos casos de validación del parámetro (1 ms)
    ✓ buscarProcedimientosPorKeywords sin argumentos retorna [] y no consulta (1 ms)
    ✓ buscarProcedimientosPorKeywords busca con like y mapea resultados (2 ms)
    ✓ consultarr devuelve mensaje si no hay palabras clave (1 ms)
    ✓ consultarr devuelve mensaje si no encuentra coincidencias (1 ms)
    ✓ consultarr devuelve resultados cuando hay coincidencias (1 ms)

Test Suites: 1 passed, 1 total
Tests:    13 passed, 13 total
Snapshots: 0 total
Time:    1.236 s
Ran all test suites matching _tests/_ChatServices.test.js/1.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe gestionar integralmente las citas médicas: creación, consulta (por rol), cancelación, actualización de estado, generación de PDFs y reportes; asegurando validaciones, manejo de errores controlados y flujos diferenciados por rol (doctor, asistente, admin/rol desconocido).
Objetivo	Validar exhaustivamente el controlador de Citas verificando que: Seleccione el flujo correcto según el rol autenticado. Valide parámetros (IDs, fechas, tipos de cita, rangos). Dele que a servicios y controle errores devolviendo los códigos HTTP correctos (200, 201, 400, 404, 409, 500). Genere PDFs (resultados/fórmulas) en escenarios felices y de falla.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Usuario autenticado (req.usuario: { id, rol } → doctor/asistente/admin). Parámetros y body simulados: doctorId, usuarioId, tipo (evaluación/procedimiento), fecha, hora, observaciones. Servicios mockeados (listar, crear, actualizar, cancelar, disponibilidad, PDFs). Ambiente: Node.js + Express + Jest.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Procedimiento de Prueba	Pre-check – Creación de Requerimientos (asignación id_doctor del autenticado): Paso 1: Simular req.usuario con rol doctor y id = 7. Paso 2: Enviar creación de requerimiento sin id_doctor en el body. Paso 3: Ejecutar controlador. Resultado esperado: Completa el payload con id_doctor = 7 y delega al servicio. Si el servicio falla, devuelve 500 con mensaje genérico; si faltan datos, 400. Prueba 1 – Listado de Citas y Pacientes (por rol): Paso 1: Simular doctor autenticado y ejecutar listados → debe listar solo sus citas. Paso 2: Simular asistente con doctorId válido → debe listar citas del doctor solicitado; sin doctorId, 400. Paso 3: Simular admin/rol desconocido → listado global. Paso 4: Forzar error del servicio para verificar 500. Resultado esperado: Doctor → 200 con sus citas. Asistente sin doctorId → 400. Con doctorId → 200. Admin/rol desconocido → 200 global. Falla servicio → 500. Prueba 2 – (duplicado funcional de 1) Validación de códigos y parámetros: Paso 1: Repetir matriz de roles y parámetros de la Prueba 1.
-------------------------	--



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Paso 2: Inyectar parámetros inválidos (IDs no numéricos, filtros erróneos).

Paso 3: Verificar códigos de respuesta por caso.

Resultado esperado: Idéntico a Prueba 1, con 400 ante parámetros inválidos y 500 ante fallas del servicio.

Prueba 3 – Cancelar y Eliminar Citas:

Paso 1: Simular cancelación de cita con id_doctor presente → debe enviar notificación.

Paso 2: Simular cancelación de cita sin id_doctor → no envía notificación.

Paso 3: Simular eliminación de cita existente → 200 con mensaje de éxito.

Paso 4: Forzar error del servicio → 500.

Resultado esperado:

Cancelación válida → 200, notifica si hay id_doctor.

Eliminación válida → 200 éxito.

Falla servicio → 500.

Prueba 4 – Crear y Actualizar Citas (roles/tipos/estados):

Paso 1: Crear cita como usuario (propia), como doctor (para paciente) y asistente (para doctor/paciente) → validar que cada rol sigue su flujo.

Paso 2: Probar tipos: evaluación (30 min) y procedimiento (150 min).

Paso 3: Actualizar estado de cita (p. ej. confirmada/cancelada/realizada) y notificar totales.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 4: Inyectar errores: 400 (datos inválidos), 404 (cita no existe), 409 (conflicto de horario o duplicidad), 500 (falla servicio).</p> <p>Resultado esperado:</p> <p>Creación válida → 201 con datos de la cita.</p> <p>Actualización válida → 200 y notificación de totales.</p> <p>Errores → 400/404/409/500 según el caso.</p> <p> </p> <p>Prueba 5 – Consultas de Horarios y Rangos:</p> <p>Paso 1: Consultar disponibilidad con fecha válida → retorna slots.</p> <p>Paso 2: Validar el mapeo de duración: 30 min (evaluación) / 150 min (procedimiento).</p> <p>Paso 3: Probar fechas/rangos inválidos (formato erróneo, rango inverso) → 400.</p> <p>Paso 4: Forzar error del servicio → 500.</p> <p>Resultado esperado:</p> <p>Petición válida → 200 con lista de horarios.</p> <p>Validación de duración aplicada según tipo.</p> <p>Parámetros inválidos → 400.</p> <p>Falla servicio → 500.</p> <p>Prueba 6 – Otras Operaciones (mis citas, marcarExámenesSubidos, totales asistente):</p> <p>Paso 1: Mis citas (usuario/doctor) → devuelve solo las del autenticado (200).</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: marcarExamenesSubidos con control de roles (solo doctor o rol permitido) → acceso 200; rol no permitido → 403.</p> <p>Paso 3: Total de citas para asistente → requiere permisos; sin permisos → 403; con permisos → 200.</p> <p>Resultado esperado:</p> <p>Mis citas → 200 filtrado por autenticado.</p> <p>marcarExamenesSubidos → 200 si rol permitido; 403 si no.</p> <p>Total asistente → 200 con permiso; 403 sin permiso.</p> <p>Prueba 7 – Generación de PDFs (resultados y fórmulas):</p> <p>Paso 1: Flujo feliz: cita existente con datos completos → genera PDF y retorna 200 con metadatos (URL/ID).</p> <p>Paso 2: 404 si la cita no existe.</p> <p>Paso 3: 400 si faltan datos mínimos (p. ej., sin observaciones cuando son obligatorias).</p> <p>Paso 4: 500 si falla la creación/subida del PDF.</p> <p>Paso 5: Casos particulares: imágenes faltantes y fechas inválidas → 400.</p> <p>Resultado esperado:</p> <p>Éxito → 200 y artefacto PDF disponible.</p> <p>Faltas/No existe → 400/404.</p> <p>Error interno → 500.</p>
Datos de salida - Resultado Esperado	1 Rutas de citas responden con 200/201 en éxito y 400/404/409/500 según validaciones/errores.

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 98 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	<p>2 Flujos por rol se respetan (doctor/asistente/admin/usuario).</p> <p>3 Validaciones de fechas, IDs, tipos, rangos aplicadas correctamente.</p> <p>4 Notificaciones se disparan cuando corresponde (cancelación con id_doctor, actualización de totales).</p> <p>5 PDFs se generan en escenarios válidos y se gestionan los fallos de forma controlada.</p> <p>6 No hay excepciones no controladas.</p>		
Resultado Obtenido Prueba Exitosa	Prueba exitosa – El controlador de Citas gestionó correctamente los escenarios definidos, aplicando reglas por rol, validaciones de negocio y manejo de errores conforme a lo esperado.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

10. CitasControllers.test.js

Validar exhaustivamente el **controlador de citas médicas** de la clínica estética: manejo de creación, consulta, cancelación, actualización de estado, generación de PDFs y reportes, asegurando respuestas correctas y flujos diferenciados por roles (doctor, usuario, asistente).

Creación de Requerimientos

Verifica que el requerimiento se cree correctamente usando el id_doctor del usuario autenticado.

Controla errores del servicio y devuelve un mensaje genérico.

Prueba 1: Listado de Citas y Pacientes

Comprueba flujos para:

- **Doctor:** lista sus citas.
- **Asistente:** necesita doctorId válido.
- **Admin o rol desconocido:** lista global.
- Valida códigos 200, 400, 500 según error o parámetros inválidos.

```
test("crearRequerimiento -> 201 con id_doctor del req", async () => {
  citasService.crearRequerimiento.mockResolvedValue({ id: 10 });
  const { req, res } = mockReqRes({ body: { x: 1 }, usuario: { id: 99 } });
  await controller.crearRequerimiento(req, res);
  expect(res.statusCode).toBe(201);
  expect(res.body).toEqual({ id: 10 });
  expect(citasService.crearRequerimiento).toHaveBeenCalledWith({
    x: 1,
    id_doctor: 99,
  });
});
```

```
test("listarCitas (asistente con doctorId válido) -> 200", async () => {
  citasService.listarLasCitas.mockResolvedValue([{ id: 5 }]);
  const { req, res } = mockReqRes({
    usuario: { id: 2, rol: "asistente" },
    query: { doctorId: "12" },
  });
  await controller.listarCitas(req, res);
  expect(res.statusCode).toBe(200);
  expect(citasService.listarLasCitas).toHaveBeenCalledWith(12);
  expect(res.body).toEqual([{ id: 5 }]);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test("listarCitas -> 500 si service falla", async () => [
  citasService.listarLasCitas.mockRejectedValue(new Error("fail"));
  const { req, res } = mockReqRes({ usuario: { id: 7, rol: "doctor" } });
  await controller.listarCitas(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body.error).toBe("Error al obtener las citas");
]);
```

Prueba 2: Listado de Citas y Pacientes

Comprueba flujos para:

- **Doctor:** lista sus citas.
- **Asistente:** necesita doctorId válido.
- **Admin o rol desconocido:** lista global.
- Valida códigos 200, 400, 500 según error o parámetros inválidos.

```
test("ListarCitas (asistente con doctorId válido) -> 200", async () => {
  citasService.listarLasCitas.mockResolvedValue([{ id: 5 }]);
  const { req, res } = mockReqRes({
    usuario: { id: 2, rol: "asistente" },
    query: { doctorId: "12" },
  });
  await controller.listarCitas(req, res);
  expect(res.statusCode).toBe(200);
  expect(citasService.listarLasCitas).toHaveBeenCalledWith(12);
  expect(res.body).toEqual([{ id: 5 }]);
});
```

```
test("listarCitas -> 500 si service falla", async () => [
  citasService.listarLasCitas.mockRejectedValue(new Error("fail"));
  const { req, res } = mockReqRes({ usuario: { id: 7, rol: "doctor" } });
  await controller.listarCitas(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body.error).toBe("Error al obtener las citas");
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3: Cancelar y Eliminar Citas

Asegura que las citas puedan cancelarse correctamente y disparen notificaciones solo si tienen id_doctor. Eliminaciones retornan 200 con mensaje de éxito.

```
test("cancelarCita -> 200 y dispara notificaciones si hay doctor", async () => {
    citasService.buscarLasCitas.mockResolvedValue([ { id: 5, id_doctor: 10 } ]);
    citasService.actualizarLasCitas.mockResolvedValue([1]);
    const { req, res } = mockReqRes({ params: { id: 5 } });
    await controller.cancelarCita(req, res);
    expect(res.statusCode).toBe(200);
    expect(res.body).toEqual({ mensaje: "Cita cancelada correctamente" });
    expect(citasService.notificarTotalCitasCanceladas).toHaveBeenCalledWith(10);
    expect(citasService.notificarTotalesMes).toHaveBeenCalledWith(10);
});
```

Prueba 4: Crear y Actualizar Citas

Se validan:

- Rol del creador (usuario, doctor, asistente).
- Tipos de cita (evaluación, procedimiento).
- Flujos de actualización y notificación de totales.
- Manejo de errores comunes (400, 404, 409, 500).

```
test("crearCitas (usuario) -> si no envia tipo, lo fuerza a evaluacion", async () => {
    citasService.crearLasCitas.mockResolvedValue([
        { id: 1,
          id_doctor: 9,
          tipo: "evaluacion",
        });
    const { req, res } = mockReqRes({
        usuario: { id: 8, rol: "usuario" },
        body: {},
    });
    await controller.crearCitas(req, res);
    expect(res.statusCode).toBe(201);
    expect(citasService.crearLasCitas).toHaveBeenCalledWith({
        _rol_creador: "usuario",
        tipo: "evaluacion",
    });
    expect(citasService.notificarTotalCitas).toHaveBeenCalledWith(9);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test("actualizarCitaDoctor -> 200 con requerimientos y notificaciones", async () => {
    citasService.actualizarLasCitas.mockResolvedValue([1]);
    citasService.buscarLasCitas.mockResolvedValue([
        {
            id: 9,
            id_usuario: 10,
            id_doctor: 11,
            tipo: "evaluacion",
        }
    ]);
    citasService.crearRequerimiento.mockResolvedValue({ id: 1 });

    const { req, res } = mockReqRes({
        params: { id: 9 },
        body: {
            requerimientos: [
                {
                    descripcion: "A",
                    frecuencia: "3",
                    repeticiones: "5",
                    fecha_inicio: "2025-01-01",
                },
                {
                    descripcion: "B",
                    frecuencia: "1",
                    repeticiones: "2",
                    fecha_inicio: "2025-01-02",
                },
            ],
        },
    });
    let call = 0;
    citasService.crearRequerimiento.mockImplementation(() => {
        call += 1;
        if (call === 2) return Promise.reject(new Error("inner"));
        return Promise.resolve({ id: call });
    });

    await controller.actualizarCitaDoctor(req, res);
    expect(res.statusCode).toBe(200);
    expect(citasService.notificarTotalCitasPendientesHoy).toHaveBeenCalledWith(
        11
    );
});
```



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test("actualizarEstadoCita -> 400 id inválido, 404 no encontrada, 200 ok y 409 error", async () => {
    let ctx = mockReqRes({ params: { id: "abc" }, body: {} });
    await controller.actualizarEstadoCita(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(400);

    citasService.cambiarEstadoCita.mockResolvedValue(null);
    ctx = mockReqRes({ params: { id: "9" }, body: {} });
    await controller.actualizarEstadoCita(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(404);

    citasService.cambiarEstadoCita.mockResolvedValue({ id: 9 });
    citasService.buscarLasCitas.mockResolvedValue([
        {
            id: 9,
            id_doctor: 3,
            tipo: "evaluacion",
        }
    ]);
    ctx = mockReqRes({
        params: { id: "9" },
        body: { estado: "realizada" },
        usuario: { id: 3 },
    });
    await controller.actualizarEstadoCita(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(200);
    expect(citasService.notificarTotalesMes).toHaveBeenCalledWith(3);

    const err = Object.assign(new Error("conflict"), { status: 409 });
    citasService.cambiarEstadoCita.mockRejectedValue(err);
    ctx = mockReqRes({ params: { id: "9" }, body: {} });
    await controller.actualizarEstadoCita(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(409);
})
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 104 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 5: Consultas de Horarios y Rangos

Evalúa los endpoints de disponibilidad y consulta, incluyendo:

- Validación de fechas.
- Mapeo de duración de cita (30 min evaluaciones / 150 min procedimientos).
- Manejo de errores del servicio.

```
test("obtenerHorariosOcupados -> 400 sin fecha, 200 mapea duraciones, 500 en error", async () => {
  let ctx = mockReqRes({ params: {}, query: {} });
  await controller.obtenerHorariosOcupados(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);

  citasService.obtenerCitasPorFecha.mockResolvedValue([
    { id: 1, fecha: "2025-01-01T10:00:00Z", tipo: "evaluacion" },
    { id: 2, fecha: "2025-01-01T12:00:00Z", tipo: "procedimiento" },
  ]);
  ctx = mockReqRes({
    params: { fecha: "2025-01-01" },
    query: { doctorId: "9" },
  });
  await controller.obtenerHorariosOcupados(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  expect(ctx.res.body[0].duracion).toBe(30);
  expect(ctx.res.body[1].duracion).toBe(150);

  citasService.obtenerCitasPorFecha.mockRejectedValue(new Error("x"));
  ctx = mockReqRes({ params: { fecha: "2025-01-01" }, query: {} });
  await controller.obtenerHorariosOcupados(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 6: Otras Operaciones

Verifica:

- Listado personal de citas.
- Control de roles en marcarExamenesSubidos.
- Consulta total de citas para asistentes.

```
test("marcarExamenesSubidos -> 403 no usuario, 200 ok, 500 con status", async () => {
  let ctx = mockReqRes({
    usuario: { id: 1, rol: "doctor" },
    params: { id: 2 },
  });
  await controller.marcarExamenesSubidos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(403);

  citasService.marcarExamenesSubidos.mockResolvedValue({ id: 2 });
  ctx = mockReqRes({ usuario: { id: 1, rol: "usuario" }, params: { id: 2 } });
  await controller.marcarExamenesSubidos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  const err = Object.assign(new Error("fail"), { status: 418 });
  citasService.marcarExamenesSubidos.mockRejectedValue(err);
  ctx = mockReqRes({ usuario: { id: 1, rol: "usuario" }, params: { id: 2 } });
  await controller.marcarExamenesSubidos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(418);
});
```

Prueba 7: Generación de PDFs

Simula creación de archivos PDF para resultados y fórmulas médicas:

- Casos exitosos (flujo feliz con datos completos).
- Casos de error (404 sin cita, 400 sin datos, 500 al fallar creación).
- Cubre observaciones, imágenes faltantes y fechas inválidas.

```
test("generarPDFExamenes -> 200 flujo completo", async () => {
    const ctrl = require("../controllers/CitasControllers");
    citasService.buscarLasCitas.mockResolvedValue({
        id: 1,
        fecha: "2025-01-01",
        tipo: "evaluacion",
        examenes_requeridos: "Hematologia\\nRX",
        doctor: { nombre: "Doc", ocupacion: "Medico" },
        usuario: { nombre: "Paciente" },
    });
    const { req, res } = {
        req: { params: { id: 1 } },
        res: {
            statusCode: 200,
            headers: {},
            setHeader(n, v) {
                this.headers[n] = v;
            },
            status(c) {
                this.statusCode = c;
                return this;
            },
            json(p) {
                this.body = p;
                return this;
            },
        },
    };
    await ctrl.generarPDFExamenes(req, res);
    expect(res.statusCode).toBe(200);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test("generarPDFMedicamentos -> 500 si falla creacion documento", async () => {
  const ctrl = require("../controllers/CitasControllers");
  citasService.buscarLasCitas.mockResolvedValue({
    id: 11,
    medicamentos_recetados: "Med",
    fecha: "2025-01-01",
    doctor: {},
    usuario: {},
    tipo: "evaluacion",
  });
  const { req, res } = {
    req: { params: { id: 11 } },
    res: {
      statusCode: 200,
      headers: {},
      setHeader(n, v) {
        this.headers[n] = v;
      },
      status(c) {
        this.statusCode = c;
        return this;
      },
      json(p) {
        this.body = p;
        return this;
      },
    },
  };
  await ctrl.generarPDFMedicamentos(req, res);
  expect(res.statusCode).toBe(500);
  expect(res.body).toEqual({ error: "Error al generar PDF" });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Resultado

```
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends$ npm test -- _tests_/CitasControllers.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/CitasControllers.test.js

PASS  __tests__/CitasControllers.test.js
Controlador de Citas (subset)
  ✓ crearRequerimiento -> 201 con id doctor del req (7 ms)
  ✓ crearRequerimiento -> 500 si servicio falla (2 ms)
  ✓ listarCitas (doctor) -> 200 (2 ms)
  ✓ listarPacientesPorDoctor -> 200 y 500 (1 ms)
  ✓ listarCitasPorUsuarioYDoctor -> 200 y 500 (1 ms)
  ✓ listarCitas (asistente sin doctorId) -> []
  ✓ listarCitas (asistente con doctorId válido) -> 200 (1 ms)
  ✓ listarCitas -> 500 si servicio falla (1 ms)
  ✓ listarCitas (rol desconocido) -> lista global con doctorId null (2 ms)
  ✓ listarCitas (usuario con doctorId inválido) -> 400 (1 ms)
  ✓ cancelarCita -> 404 si no existe (1 ms)
  ✓ cancelarCita -> 200 y dispara notificaciones si hay doctor (1 ms)
  ✓ cancelarCita -> 200 sin id_doctor (no notifica) (1 ms)
  ✓ cancelarCita -> 500 si servicio falla (1 ms)
  ✓ buscarCitas -> 200, 404 y 500 (1 ms)
  ✓ crearCitas (usuario) -> si no envia tipo, lo fuerza a evaluacion (1 ms)
  ✓ crearCitas -> valida tipo de cita cuando no es usuario
  ✓ crearCitas (usuario) -> tipo procedimiento devuelve 408
  ✓ crearCitas (no usuario) -> por defecto evaluacion y notifica (1 ms)
  ✓ crearCitas (no usuario - procedimiento) -> notifica procedimiento (1 ms)
  ✓ crearCitas -> 400 si servicio lanza con status
  ✓ actualizarCitaUsuario -> 200, 404 y 500
  ✓ actualizarCitaDoctor -> 404 si no se actualiza
  ✓ actualizarCitaDoctor -> 200 con requerimientos y notificaciones
  ✓ actualizarCitaDoctor -> 500 si servicio falla
  ✓ actualizarCitaDoctor -> notifica procedimiento cuando tipo es procedimiento
  ✓ crearOrdenDesdeCarrito -> 201 y 500 (1 ms)
  ✓ eliminarCitas -> 200 (1 ms)
  ✓ obtenerHorariosOcupados -> 400 sin fecha, 200 mapea duraciones, 500 en error (2 ms)
  ✓ citasPorDia -> 200, 404 y 500 (1 ms)
  ✓ citasPorDia -> 200, 404 y 500 (1 ms)
  ✓ citasPorRango -> 200, 404 y 500 (1 ms)
  ✓ citasPorTipo -> 200, 404 y 500
  ✓ misCitas -> 200 y 500
  ✓ actualizarEstadoCita -> 400 id inválido, 404 no encontrada, 200 ok y 409 error (1 ms)
  ✓ actualizarEstadoCita -> notifica procedimiento cuando tipo es procedimiento (1 ms)
  ✓ marcarExamenesSubidos -> 403 no usuario, 200 ok, 500 con status (5 ms)
  ✓ reagendar -> 200 y 500 (1 ms)
  ✓ consultarTodasLasCitasAsistente -> 200 y 500 (1 ms)
  ✓ generarPDFExamenes -> 404 y 400 (retornos tempranos) (1 ms)
  ✓ generarPDFMedicamentos -> 404 y 400 (retornos tempranos) (1 ms)

PDF happy path (mockeando pdfkit)
  ✓ generarPDFExamenes -> 200 flujo completo (31 ms)
  ✓ generarPDFMedicamentos -> 200 flujo completo (1 ms)
  ✓ generarPDFExamenes -> cubre observaciones y fallbacks de doctor/usuario (1 ms)
  ✓ generarPDFMedicamentos -> cubre bloque de observaciones (1 ms)
  ✓ generarPDFExamenes -> hace warn cuando image lanza (1 ms)
  ✓ generarPDFMedicamentos -> usa fallback de fecha cuando toLocaleString lanza (1 ms)

PDF errores (mockeando pdfkit para lanzar)
  ✓ generarPDFExamenes -> 500 si falla creación documento (2 ms)
  ✓ generarPDFMedicamentos -> 500 si falla creación documento (1 ms)

Test Suites: 1 passed, 1 total
Tests:       48 passed, 48 total
Snapshots:   0 total
Time:        0.911 s, estimated 2 s
Ran all test suites matching _tests_/_CitasControllers.test.js/i.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe garantizar el correcto funcionamiento del servicio de Citas, incluyendo: Cálculo y notificación de totales diarios por estado y tipo. Creación automatizada de requerimientos con agendamiento según HorariosDisponibles. Emisión de eventos en tiempo real mediante socket.io. Manejo robusto de errores en conteos y agendamiento.
Objetivo	Validar que las funciones del servicio de citas: <ol style="list-style-type: none">1. calculen correctamente los conteos por rango/estado/tipo,2. emitan notificaciones en tiempo real con datos verificados,3. creen requerimientos y agenden automáticamente la primera franja disponible, y4. manejen errores retornando valores seguros sin afectar la estabilidad.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones: calcularTotalesPorRango, notificarTotalesDiarios, crearRequerimientoYAgendar, notificarTotalesPorTipo. Mocks: modelos de citas, util HorariosDisponibles, socket.io (emisor), reloj/fechas.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Escenarios: rangos válidos/ inválidos, horarios disponibles/ no disponibles, éxito/ fallo de base de datos.
Procedimiento de Prueba	<p>Prueba 1 – Calcular Totales por Rango:</p> <p>Paso 1: Simular datos de citas en el rango con estados cancelada/pendiente/realizada y tipos evaluacion/procedimiento.</p> <p>Paso 2: Ejecutar calcularTotalesPorRango(fechalnicio, fechaFin).</p> <p>Paso 3: Inspeccionar el objeto devuelto.</p> <p>Resultado esperado: Devuelve un objeto resumen con conteos correctos por estado y tipo,</p> <p>Prueba 2 – Notificación de Totales Diarios:</p> <p>Paso 1: Mockear calcularTotalesPorRango para retornar un resumen conocido.</p> <p>Paso 2: Ejecutar notificarTotalesDiarios(doctorId, fecha) con socket.io mockeado.</p> <p>Paso 3: Verificar emisión a la sala doctor_<id> con el evento totalesDia.</p> <p>Resultado esperado: Se emite una vez el evento totalesDia a doctor_<id> usando los valores retornados por calcularTotalesPorRango.</p> <p>Prueba 3 – Creación de Requerimientos y Agendamiento Automático:</p> <p>Paso 1: Mockear creación de requerimiento exitosa.</p> <p>Paso 2: Mockear HorariosDisponibles para que retorne la primera franja libre y simular creación de cita en esa franja.</p> <p>Paso 3: Ejecutar crearRequerimientoYAgendar(payload).</p>



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado (éxito): Retorna objeto con requerimiento creado y cita asignada en la primera franja disponible.</p> <p>Caso de falla (sin horarios disponibles):</p> <p>Paso 1: Mockear HorariosDisponibles para retornar [].</p> <p>Paso 2: Ejecutar crearRequerimientoYAgendar(payload).</p> <p>Resultado esperado: Lanza el error "No se pudo asignar la cita" sin crear cita.</p> <p>Prueba 4 – Notificación de Totales por Tipo de Cita:</p> <p>Paso 1: Mockear conteos diarios por tipo (evaluacion, procedimiento).</p> <p>Paso 2: Ejecutar notificarTotalesPorTipo(doctorId, fecha).</p> <p>Paso 3: Verificar que se emitan eventos por tipo (p. ej. totalesDia:tipo) a la sala doctor_<id>.</p> <p>Resultado esperado (éxito): Se emiten eventos con los valores correctos por tipo.</p> <p>Resultado esperado (error): Si ocurre un error en el conteo, el servicio devuelve 0 para ese tipo y no rompe la ejecución global (no lanza excepción no controlada).</p>
Datos de salida - Resultado Esperado	1 El conteo por rango refleja con precisión estados y tipos. 2 Las notificaciones en tiempo real salen al canal 3 doctor_<id> con los valores calculados. 3 El agendamiento automático selecciona la primera franja disponible; si no hay, lanza el error esperado.

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 113 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	4 Fallas en conteos/notificaciones devuelven 0 o errores controlados, manteniendo la estabilidad del sistema.		
Resultado Obtenido Prueba Exitosa	El servicio de Citas calcula totales, emite notificaciones, agenda automáticamente y maneja errores según lo esperado.	Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 114 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

11. CitasServices.test.js

Validar las principales funciones del **servicio de citas** relacionadas con:

- Cálculo y notificación de totales diarios.
- Creación automatizada de requerimientos (agenda y validación de horarios).
- Emisión de eventos en tiempo real (via socket.io).
- Manejo de errores en flujos de conteo y agendamiento.

Prueba 1: Calcular Totales por Rango

Valida que el servicio cuente correctamente las citas agrupadas por estado (cancelada, pendiente, realizada) y tipo (evaluacion, procedimiento), devolviendo un objeto resumen.

```
test('calcularTotalesPorRango cuenta por estado y tipo', async () => {
  models.citas.findAll.mockResolvedValue([
    { estado: 'cancelada', tipo: 'evaluacion' },
    { estado: 'pendiente', tipo: 'procedimiento' },
    { estado: 'realizada', tipo: 'evaluacion' },
    { estado: 'realizada', tipo: 'procedimiento' },
    { estado: 'realizada', tipo: 'procedimiento' },
  ]);
  const res = await svc.calcularTotalesPorRango(7, new Date(), new Date());
  expect(res).toEqual({ total: 5, canceladas: 1, pendientes: 1, realizadasEvaluacion: 1, realizadasProcedimiento: 2 });
});

test('notificarTotalesDia emite al room del doctor', async () => {
  const fake = { total: 1 };
  jest.spyOn(svc, 'calcularTotalesPorRango').mockResolvedValueOnce(fake);
  await svc.notificarTotalesDia(7);
  expect(global.io.to).toHaveBeenCalledWith('doctor_7');
  expect(room.emit).toHaveBeenCalledWith('totalesDia', fake);
});
```

Prueba 2: Notificación de Totales Diarios

Verifica que se emita correctamente el evento totalesDia al canal correspondiente (doctor_<id>) usando socket.io, y que los valores provengan de calcularTotalesPorRango.

```
test('notificarTotalesDia emite al room del doctor', async () => {
  const fake = { total: 1 };
  jest.spyOn(svc, 'calcularTotalesPorRango').mockResolvedValueOnce(fake);
  await svc.notificarTotalesDia(7);
  expect(global.io.to).toHaveBeenCalledWith('doctor_7');
  expect(room.emit).toHaveBeenCalledWith('totalesDia', fake);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3: Creación de Requerimientos y Agendamiento Automático

Simula la creación de un requerimiento (tarea médica) y la asignación automática de la primera cita disponible según el horario configurado en HorariosDisponibles.

También se prueba la falla cuando no existen horarios disponibles, verificando el lanzamiento del error "No se pudo asignar la cita".

```
test('crearRequerimiento crea requerimiento y agenda primer slot disponible', async () => {
  const data = { id_usuario: 1, id_doctor: 2, fecha_inicio: '2025-10-06', repeticiones: 1, frecuencia: 1, descripcion: 'desc' };
  models.requerimientos.create.mockResolvedValue({ id: 99 });
  models.citas.findAll.mockResolvedValue([]);
  models.citas.create.mockResolvedValue({ id: 123 });

  const res = await svc.crearRequerimiento(data);
  expect(models.requerimientos.create).toHaveBeenCalledWith(data);
  expect(models.citas.create).toHaveBeenCalledWith(expect.objectContaining({ id_usuario: 1, id_doctor: 2, tipo: 'procedimiento', }));
  expect(res).toEqual({ id: 99 });
});

test('crearRequerimiento lanza error si no hay horarios en varios intentos', async () => {
  HorariosDisponibles.horarios.mockImplementation(() => []);
  models.requerimientos.create.mockResolvedValue({ id: 1 });
  models.citas.findAll.mockResolvedValue([]);
  await expect(svc.crearRequerimiento({ id_usuario: 1, id_doctor: 2, fecha_inicio: '2025-10-06', repeticiones: 1, frecuencia: 1, }))
    .rejects.toThrow(/No se pudo asignar la cita/);
});
```

Prueba 4: Notificación de Totales por Tipo de Cita

Valida que se emitan correctamente los eventos de conteo diario para cada tipo de cita y que, en caso de error, el servicio devuelva 0 sin romper la ejecución.

```
test('notificarTotalCitasRealizadasProcedimientoHoy emite y retorna total', async () => {
  models.citas.count.mockResolvedValue(5);
  const total = await svc.notificarTotalCitasRealizadasProcedimientoHoy(3);
  expect(models.citas.count).toHaveBeenCalled();
  expect(global.io.to).toHaveBeenCalledWith('doctor_3');
  expect(room.emit).toHaveBeenCalledWith('totalCitasRealizadasProcedimientoHoy', { total: 5 });
  expect(total).toBe(5);
});

test('notificarTotalCitasRealizadasProcedimientoHoy maneja error devolviendo 0', async () => {
  models.citas.count.mockRejectedValue(new Error('db'));
  const total = await svc.notificarTotalCitasRealizadasProcedimientoHoy();
  expect(total).toBe(0);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test('notificarTotalCitasRealizadasProcedimientoHoy maneja error devolviendo 0', async () => {
  models.citas.count.mockRejectedValue(new Error('db'));
  const total = await svc.notificarTotalCitasRealizadasProcedimientoHoy();
  expect(total).toBe(0);
});

test('notificarTotalCitasRealizadasEvaluacionHoy emite y retorna total', async () => [
  models.citas.count.mockResolvedValue(7);
  const total = await svc.notificarTotalCitasRealizadasEvaluacionHoy(4);
  expect(models.citas.count).toHaveBeenCalled();
  expect(global.io.to).toHaveBeenCalledWith('doctor_4');
  expect(room.emit).toHaveBeenCalledWith('totalCitasRealizadasEvaluacionHoy', { total: 7 });
  expect(total).toBe(7);
]);

test('notificarTotalCitasRealizadasEvaluacionHoy maneja error devolviendo 0', async () => {
  models.citas.count.mockRejectedValue(new Error('db'));
  const total = await svc.notificarTotalCitasRealizadasEvaluacionHoy();
  expect(total).toBe(0);
});

test('notificarTotalCitasCanceladasHoy emite y retorna total', async () => {
  models.citas.count.mockResolvedValue(3);
  const total = await svc.notificarTotalCitasCanceladasHoy(8);
  expect(models.citas.count).toHaveBeenCalled();
  expect(global.io.to).toHaveBeenCalledWith('doctor_8');
  expect(room.emit).toHaveBeenCalledWith('totalCitasCanceladasHoy', { total: 3 });
  expect(total).toBe(3);
});

test('notificarTotalCitasCanceladasHoy maneja error devolviendo 0', async () => {
  models.citas.count.mockRejectedValue(new Error('db'));
  const total = await svc.notificarTotalCitasCanceladasHoy();
  expect(total).toBe(0);
});
});
```

Resultado

```
dstevengz@dstevengz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests/_CitasServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests/_CitasServices.test.js

PASS  _tests/_CitasServices.test.js
  CitasServices (subset)
    calcularTotalesPorRango cuenta por estado y tipo (441 ms)
    notificarCitasEsbia unito al room del doctor_0 (6 ms)
    crearRequerimiento crea requerimiento y agenda primer slot disponible (9 ms)
    crearRequerimiento lanza error si no hay horarios en varios intentos (49 ms)
    notificarTotalCitasRealizadasProcedimientoHoy emite y retorna total (3 ms)
    notificarTotalCitasRealizadasEvaluacionHoy emite y retorna total (1 ms)
    notificarTotalCitasRealizadasEvaluacionHoy maneja error devolviendo 0 (1 ms)
    notificarTotalCitasCanceladasHoy emite y retorna total (3 ms)
    notificarTotalCitasCanceladasHoy maneja error devolviendo 0 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:  0 total
Time:        1.245 s
Run all test suites matching _tests/_CitasServices.test.js

dstevengz@dstevengz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe gestionar integralmente el ciclo del consentimiento informado : consulta, creación, eliminación y limpieza; además de generar y descargar PDFs firmados (metadatos, relaciones y firma digital), devolviendo códigos HTTP correctos (200, 404, 500) y manejando errores de forma robusta.
Objetivo	Validar que el Controlador de Consentimientos: Procese listados, creación, eliminación y limpieza por usuario autenticado. Genere/entregue PDFs firmados con metadatos y relaciones. Maneje fallos controlados (servicio/Cloudinary/relaciones) sin excepciones no capturadas y con códigos HTTP esperados.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: listarConsentimientos, crearConsentimiento, eliminarConsentimiento, limpiarConsentimientosUsuario, descargarConsentimiento.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Mocks: servicio de consentimientos, modelos relacionados (usuario, procedimiento, cita), generador PDF, Cloudinary (firma URL), req.usuario.id.</p> <p>Escenarios: éxito, 404 por inexistencia, 500 por error interno.</p>
Procedimiento de Prueba	<p>Prueba 1 – Listado de Consentimientos</p> <p>Paso 1: Simular solicitud GET al listado.</p> <p>Paso 2: Mockear servicio para devolver arreglo de consentimientos.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: 200 OK con la lista. Si el servicio lanza error → 500.</p> <p>Prueba 2 – Creación de Consentimiento</p> <p>Paso 1: Enviar body válido (usuariold, procedimentold, datos del consentimiento).</p> <p>Paso 2: Mockear servicio crear con retorno exitoso.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: 201 Created con el objeto creado. En datos inválidos → 400. Falla servicio → 500.</p> <p>Prueba 3 – Eliminación de Consentimiento</p> <p>Paso 1: Enviar id válido por params.</p> <p>Paso 2: Mockear eliminación exitosa.</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Paso 3: Ejecutar el controlador.

Resultado esperado (éxito): 200 OK con mensaje de eliminación.

Resultado esperado (error interno): Servicio lanza error → 500 con mensaje genérico.

Resultado esperado (no existe): Si se contempla, 404 cuando no se encuentre el ID.

Prueba 4 – Limpieza de Consentimientos de Usuario

Paso 1: Simular req.usuario.id (autenticado).

Paso 2: Ejecutar limpiarConsentimientosUsuario.

Paso 3: Verificar delegación al servicio con ese id.

Resultado esperado: 200 OK con mensaje confirmando la limpieza de todos los consentimientos del usuario. Si el servicio falla → 500.

Prueba 5 – Descarga del Consentimiento (PDF firmado)

Paso 1: Simular id de consentimiento y existencia de todas las relaciones (usuario, procedimiento, cita).

Paso 2: Generar PDF vía servicio (mock PDFKit/stream).

Paso 3: Firmar URL con Cloudinary (mock de upload/download/sign).

Paso 4: Permitir que metadatos faltantes se registren como warning sin impedir la entrega.

Paso 5: Simular fallo crítico en la generación para validar 500.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado (flujo feliz): 200 OK con payload de descarga (URL firmada/public_id/timestamp).</p> <p>Resultado esperado (404): Si el consentimiento o relaciones no existen → 404 Not Found.</p> <p>Resultado esperado (500): Falla de generación/subida → 500 Internal Server Error.</p>		
Datos de salida - Resultado Esperado	<p>1 200/201 en flujos correctos; 404 por inexistencia; 500 por errores internos; 400 en creación con datos inválidos.</p> <p>2 La limpieza usa req.usuario.id y confirma la operación.</p> <p>3 La descarga PDF entrega URL firmada y no se detiene por metadatos faltantes (solo warnings).</p> <p>4 No hay excepciones no controladas.</p>		
Resultado Obtenido Prueba Exitosa	El controlador de Consentimientos cumplió con los flujos de consulta/creación/eliminación/limpieza y descarga PDF firmada, retornando códigos correctos y manejando errores según lo esperado.	Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 122 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

12. ConsentimientoControllers.test.js

Validar el correcto funcionamiento del **Controlador de Consentimientos**, que gestiona el ciclo completo del consentimiento informado dentro del sistema clínico:

- **Consulta, creación, eliminación y limpieza** de consentimientos.
- **Generación y descarga en PDF** con metadatos, relaciones y firma digital.
- **Manejo robusto de errores y respuestas HTTP (200, 404, 500).**

prueba 1: Listado de Consentimientos

Valida la correcta respuesta del sistema al listar los consentimientos

```
test('obtenerConsentimientosPorUsuario -> 200', async () => {
  consentimientoService.obtenerConsentimientosPorUsuario.mockResolvedValue([{"id": 1}]);
  const { req, res } = mockReqRes();
  await controller.obtenerConsentimientosPorUsuario(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual([{"id": 1}]);
});
```

prueba 2: Creación de Consentimiento

Verifica que el sistema cree correctamente un consentimiento,

```
test('agregarConsentimiento -> 201', async () => {
  consentimientoService.crearConsentimiento.mockResolvedValue({ "id": 4 });
  const { req, res } = mockReqRes({ body: { "id_cita": 5, "texto_terminos": "ok" } });
  await controller.agregarConsentimiento(req, res);
  expect(res.statusCode).toBe(201);
  expect(res.body).toEqual({ "id": 4 });
  expect(consentimientoService.crearConsentimiento).toHaveBeenCalledWith({
    id_usuario: 9,
    id_cita: 5,
    texto_terminos: 'ok',
    fecha_firma: expect.any(Date),
    ip_firma: '127.0.0.1',
  });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3: Eliminación de Consentimiento

Comprueba la eliminación de un consentimiento por su ID.

Verifica tanto el caso exitoso (respuesta 200) como el manejo de error interno del servicio.

```
test('eliminarConsentimiento -> 200', async () => [
  const { req, res } = mockReqRes({ params: { id: 7 } });
  await controller.eliminarConsentimiento(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ mensaje: 'Consentimiento eliminado' });
  expect(consentimientoService.eliminarConsentimiento).toHaveBeenCalledWith(7);
]);
```

Prueba 4: Limpieza de Consentimientos de Usuario

Simula el proceso de limpiar todos los consentimientos de un usuario.

Se espera que la función use el id del usuario autenticado (req.usuario.id) y devuelva un mensaje confirmando la limpieza.

```
test('limpiarConsentimientos -> 200', async () => [
  const { req, res } = mockReqRes();
  await controller.limpiarConsentimientos(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ mensaje: 'Consentimientos limpiados correctamente' });
  expect(consentimientoService.limpiarConsentimientosPorUsuario).toHaveBeenCalledWith(9);
]);
```

Prueba 5: Descarga del Consentimiento

Simula el flujo completo para descargar un consentimiento firmado en PDF:

1. Valida existencia de todas las entidades relacionadas.
2. Genera el PDF mediante ConsentimientoService.
3. Firma la URL con Cloudinary.
4. Maneja fallos en metadatos sin interrumpir la entrega.
5. Lanza error 500 ante fallos críticos en la generación del PDF.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
describe('descargarConsentimiento', () => {
  test('404 si consentimiento no existe', async () => {
    consentimientoService.obtenerPorId.mockResolvedValue(null);
    const { req, res } = mockReqRes({ params: { id: 1 } });
    await controller.descargarConsentimiento(req, res);
    expect(res.statusCode).toBe(404);
    expect(res.body).toEqual({ error: 'Consentimiento no encontrado' });
  });

  test('404 si cita no existe', async () => {
    consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
    citas.findByPk.mockResolvedValue(null);
    const { req, res } = mockReqRes({ params: { id: 1 } });
    await controller.descargarConsentimiento(req, res);
    expect(res.statusCode).toBe(404);
    expect(res.body).toEqual({ error: 'Cita asociada no encontrada' });
  });

  test('404 si usuario no existe', async () => {
    consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
    citas.findByPk.mockResolvedValue({ id: 10, id_orden: 30, id_usuario: 20 });
    usuarios.findByPk.mockResolvedValue(null);
    const { req, res } = mockReqRes({ params: { id: 1 } });
    await controller.descargarConsentimiento(req, res);
    expect(res.statusCode).toBe(404);
    expect(res.body).toEqual({ error: 'Usuario asociado no encontrado' });
  });
});
```

```
test('404 si orden no existe', async () => {
  consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
  citas.findByPk.mockResolvedValue({ id: 10, id_orden: 30, id_usuario: 20 });
  usuarios.findByPk.mockResolvedValue({ id: 20 });
  jest.spyOn(ordenes, 'findByPK').mockResolvedValue(null);
  const { req, res } = mockReqRes({ params: { id: 1 } });
  await controller.descargarConsentimiento(req, res);
  expect(res.statusCode).toBe(404);
  expect(res.body).toEqual({ error: 'Orden asociada a la cita no encontrada' });
});

test('200 devuelve url de descarga cuando todo existe', async () => {
  consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
  citas.findByPk.mockResolvedValue({ id: 10, id_orden: 30, id_usuario: 20 });
  usuarios.findByPk.mockResolvedValue({ id: 20 });
  jest.spyOn(ordenes, 'findByPK').mockResolvedValue({ id: 30, procedimientos: [{ id: 1 }, { id: 2 }] });
  consentimientoService.gerararConsentimientoPDF.mockResolvedValue({ publicKey: 'pub', url: 'https://file.pdf', hash: 'abc' });
  consentimientoService.actualizarPDFMetadata.mockResolvedValue(true);

  const { req, res } = mockReqRes({ params: { id: 1 } });
  await controller.descargarConsentimiento(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ url: 'https://download/url.pdf', publicKey: 'pub' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test('200 y hace warn si no se pueden actualizar metadatos', async () => {
    consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
    citas.findByPk.mockResolvedValue({ id: 10, id_orden: 30, id_usuario: 20 });
    usuarios.findByPk.mockResolvedValue({ id: 20 });
    jest.spyOn(ordenes, 'findByPK').mockResolvedValue({ id: 30, procedimientos: [] });
    consentimientoService.generarConsentimientoPDF.mockResolvedValue({ publicId: 'pub', url: 'https://file.pdf', hash: 'abc' });
    consentimientoService.actualizarPDFMetadata.mockRejectedValue(new Error('warn'));

    const { req, res } = mockReqRes({ params: { id: 1 } });
    await controller.descargarConsentimiento(req, res);
    expect(res.statusCode).toBe(200);
    expect(res.body).toEqual({ url: 'https://download/url.pdf', publicId: 'pub' });
});

test('200 cuando orden.procedimientos no es array (usa [])', async () => {
    consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
    citas.findByPk.mockResolvedValue({ id: 10, id_orden: 30, id_usuario: 20 });
    usuarios.findByPk.mockResolvedValue({ id: 20 });
    jest.spyOn(ordenes, 'findByPK').mockResolvedValue({ id: 30, procedimientos: null });
    consentimientoService.generarConsentimientoPDF.mockResolvedValue({ publicId: 'pub', url: 'https://file.pdf', hash: 'abc' });
    consentimientoService.actualizarPDFMetadata.mockResolvedValue(true);

    const { req, res } = mockReqRes({ params: { id: 1 } });
    await controller.descargarConsentimiento(req, res);
    expect(res.statusCode).toBe(200);
    expect(res.body).toEqual({ url: 'https://download/url.pdf', publicId: 'pub' });
});
```

```
const { req, res } = mockReqRes({ params: { id: 1 } });
await controller.descargarConsentimiento(req, res);
expect(res.statusCode).toBe(200);
expect(res.body).toEqual({ url: 'https://download/url.pdf', publicId: 'pub' });
});

test('500 si falla la generación del PDF (catch global)', async () => {
    consentimientoService.obtenerPorId.mockResolvedValue({ id: 1, id_cita: 10, id_usuario: 20 });
    citas.findByPk.mockResolvedValue({ id: 10, id_orden: 30, id_usuario: 20 });
    usuarios.findByPk.mockResolvedValue({ id: 20 });
    jest.spyOn(ordenes, 'findByPK').mockResolvedValue({ id: 30, procedimientos: [] });
    consentimientoService.generarConsentimientoPDF.mockRejectedValue(new Error('fail'));

    const { req, res } = mockReqRes({ params: { id: 1 } });
    await controller.descargarConsentimiento(req, res);
    expect(res.statusCode).toBe(500);
    expect(res.body).toEqual({ error: 'Error al generar el PDF' });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Resultado

```
dstevenmri@stevenmgz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/ConsentimientoControllers.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/ConsentimientoControllers.test.js

PASS  Tests /ConsentimientoControllers.test.js
Controlador de Consentimientos
  ✓ obtenerConsentimientosPorUsuario -> 200 (42 ms)
  ✓ obtenerConsentimientosPorUsuario -> 500 en error (7 ms)
  ✓ obtenerConsentimientosPorCita -> 200 (5 ms)
  ✓ obtenerConsentimientosPorCita -> 500 en error (5 ms)
  ✓ agregarConsentimiento -> 201 (10 ms)
  ✓ agregarConsentimiento -> 500 en error (6 ms)
  ✓ eliminarConsentimiento -> 200 (6 ms)
  ✓ eliminarConsentimiento -> 500 en error (24 ms)
  ✓ limpiarConsentimientos -> 200 (5 ms)
  ✓ limpiarConsentimientos -> 500 en error (4 ms)
descargarConsentimiento:
  ✓ 404 si consentimiento no existe (5 ms)
  ✓ 404 si cita no existe (4 ms)
  ✓ 404 si usuario no existe (3 ms)
  ✓ 404 si orden no existe (5 ms)
  ✓ 200 devuelve url de descarga cuando todo existe (4 ms)
  ✓ 200 y hace warn si no se pueden actualizar metadatos (4 ms)
  ✓ 200 cuando orden.procedimientos no es array (usa []) (3 ms)
  ✓ 500 si falla la generación del PDF (catch global) (3 ms)

Test Suites: 1 passed, 1 total
Tests:    18 passed, 18 total
Snapshots: 0 total
Time:    3.262 s
Run all test suites matching _tests_ \ConsentimientoControllers.test.js/i.
dstevenmri@stevenmgz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe garantizar que la capa de servicios de Consentimientos delegue correctamente en el modelo Sequelize, genere el PDF firmado, lo suba a Cloudinary y actualice metadatos asociados, asegurando respuestas coherentes y manejo de errores básico.
Objetivo	Delegue sus operaciones CRUD en el modelo consentimiento,



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	genere un PDF con contenido mínimo (texto/fecha/IP), calcule hash de integridad y suba a Cloudinary, y actualice/consulte metadatos correctamente.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos del servicio: crearConsentimiento, obtenerConsentimientosPorUsuario, obtenerConsentimientosPorCita, eliminarConsentimiento, limpiarConsentimientosPorUsuario, generarPDFConsentimiento, actualizarPDFMetadata, obtenerPorId. Mocks: modelo Sequelize (create, findAll, findByPk, destroy, update), PDFKit, crypto.createHash, Cloudinary (upload_stream/secure_url/public_id). Datos simulados: id_usuario, id_cita, IP, fecha y contenido del consentimiento.
Procedimiento de Prueba	Prueba 1 – CRUD Delegations Paso 1: Mockear métodos del modelo consentimiento (create, findAll, destroy, update, findByPk). Paso 2: Ejecutar crearConsentimiento(payload) y verificar llamada a create con el payload. Paso 3: Ejecutar obtenerConsentimientosPorUsuario(id) y obtenerConsentimientosPorCita(id) y verificar findAll con where apropiado.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Paso 4: Ejecutar eliminarConsentimiento(id) y verificar destroy con where: { id }.

Paso 5: Ejecutar limpiarConsentimientosPorUsuario(idUsuario) y verificar destroy con where: { usuarioid: idUsuario }.

Resultado esperado: Cada función del servicio delegará correctamente en el método Sequelize correspondiente con los parámetros exactos.

Prueba 2 – Generación de Consentimiento (PDF firmado + Cloudinary)

Paso 1: Mockear PDFKit para capturar llamadas a text, fontSize, moveDown, end.

Paso 2: Ejecutar generarPDFConsentimiento({ usuario, procedimiento, cita, fecha, ip }).

Paso 3: Verificar que el servicio compone el PDF con texto mínimo (paciente/procedimiento/fecha/IP) y finaliza el stream.

Paso 4: Mockear crypto.createHash('sha256') → update(...).digest('hex') para obtener un hash determinístico.

Paso 5: Mockear Cloudinary upload_stream para devolver { secure_url, public_id }.

Resultado esperado (flujo feliz): La función retorna un objeto de metadatos con:

secure_url y public_id de Cloudinary,



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>hash_pdf calculado,</p> <p>información básica (ids/fecha/ip).</p> <p>Caso de error (opcional): si el stream/Cloudinary falla, el servicio lanza error controlado.</p> <p>Prueba 3 – Actualización de Metadatos y Búsqueda por ID</p> <p>Paso 1: Ejecutar actualizarPDFMetadata(id, { ruta_pdf, hash_pdf }).</p> <p>Paso 2: Verificar que el servicio llame update con { ruta_pdf, hash_pdf } y where: { id }.</p> <p>Paso 3: Ejecutar obtenerPorId(id) y verificar que use findByPk(id).</p> <p>Resultado esperado:</p> <p>actualizarPDFMetadata actualiza correctamente los campos ruta_pdf y hash_pdf.</p> <p>obtenerPorId retorna el consentimiento solicitado (o null si no existe).</p>
Datos de salida - Resultado Esperado	Delegaciones CRUD al modelo Sequelize correctas y con parámetros esperados.

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 131 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	<p>Generación de PDF compuesta y finalizada (mock PDFKit), hash SHA-256 calculado y subida a Cloudinary exitosa.</p> <p>Metadatos (ruta_pdf, hash_pdf) actualizados; obtenerPorId recupera el registro.</p> <p>Errores internos (stream/subida) propagados de forma controlada (si se simulan).</p>		
Resultado Obtenido Prueba Exitosa	El servicio de Consentimientos delega correctamente en el modelo, genera el PDF firmado, sube a Cloudinary y actualiza/consulta metadatos según lo esperado.	Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 132 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

13. ConsentimientoService.js

Validar el comportamiento de la **capa de servicios de Consentimientos**, que implementa la lógica de negocio detrás de la gestión de consentimientos clínicos.

Estas pruebas aseguran la correcta **delegación de operaciones al modelo Sequelize**, la **generación del documento PDF firmado**, la **subida a Cloudinary** y la **actualización de metadatos**.

Prueba 1. CRUD Delegations

Valida que las funciones del servicio delegan correctamente en el modelo consentimiento de Sequelize:

- crearConsentimiento: crea el registro.
- obtenerConsentimientosPorUsuario / obtenerConsentimientosPorCita: filtran por id_usuario o id_cita.
- eliminarConsentimiento: borra por ID.
- limpiarConsentimientosPorUsuario: borra por usuario.

```
test('CRUD delegations', async () => {
  models.consentimiento.create.mockResolvedValue({ id: 1 });
  expect(await ConsentimientoService.crearConsentimiento({ a: 1 })).toEqual({ id: 1 });
  expect(models.consentimiento.create).toHaveBeenCalledWith({ a: 1 });

  models.consentimiento.findAll.mockResolvedValue([{ id: 2 }]);
  expect(await ConsentimientoService.obtenerConsentimientosPorUsuario(5)).toEqual([{ id: 2 }]);
  expect(models.consentimiento.findAll).toHaveBeenCalledWith({ where: { id_usuario: 5 } });

  expect(await ConsentimientoService.obtenerConsentimientosPorCita(9)).toEqual([{ id: 2 }]);
  expect(models.consentimiento.findAll).toHaveBeenCalledWith({ where: { id_cita: 9 } });

  models.consentimiento.destroy.mockResolvedValue(1);
  expect(await ConsentimientoService.eliminarConsentimiento(7)).toBe(1);
  expect(models.consentimiento.destroy).toHaveBeenCalledWith({ where: { id: 7 } });

  expect(await ConsentimientoService.limpiarConsentimientosPorUsuario(3)).toBe(1);
  expect(models.consentimiento.destroy).toHaveBeenCalledWith({ where: { id_usuario: 3 } });
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 134 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Prueba.2 Generación de Consentimiento

Comprueba el flujo completo de generación del PDF del consentimiento, el cual:

1. Usa **PDFKit** para componer el documento con texto, fecha e IP.
2. Calcula un **hash de integridad** (crypto.createHash).
3. Sube el archivo a **Cloudinary** simulando la respuesta con `secure_url` y `public_id`.
4. Devuelve un objeto con los metadatos del documento.

```
test('generarConsentimientoPDF compone PDF, sube a cloudinary y retorna metadata', async () => {
  const result = await ConsentimientoService.generarConsentimientoPDF([
    { id: 1, texto_terminos: 't', fecha_firma: '2025-10-05', ip_firma: '1.2.3.4' },
    { id: 2, fecha: '2025-10-05', id_doctor: 7 },
    { id: 3 },
    { nombre: 'User', tipodocumento: 'CC', numerodocumento: '123', correo: 'a@b.com' },
    [{ nombre: 'Proc', precio: 100 }]
  ]);
  expect(result).toEqual({ publicId: 'consent/1', url: 'https://x/1.pdf', hash: 'hash123' });
});
```

Prueba:3 Actualización de Metadatos y Búsqueda por ID

Confirma que:

- `actualizarPDFMetadata` actualiza correctamente los campos `ruta_pdf` y `hash_pdf` mediante `update`.
- `obtenerPorId` usa `findById` para retornar el consentimiento solicitado.

```
test('actualizarPDFMetadata y obtenerPorId delegan al modelo', async () => {
  await ConsentimientoService.actualizarPDFMetadata('ruta.pdf', 'h');
  expect(models.consentimiento.update).toHaveBeenCalledWith({ ruta_pdf: 'ruta.pdf', hash_pdf: 'h' });

  models.consentimiento.findByPk.mockResolvedValue({ id: 5 });
  expect(await ConsentimientoService.obtenerPorId(5)).toEqual({ id: 5 });
  expect(models.consentimiento.findById).toHaveBeenCalledWith(5);
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 135 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación:	1/02/2025	Elaborado por: Instructores área Software
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultad0

```
dstevengmz1@Dstevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- __tests__/_ConsentimientoService.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests __tests__/_ConsentimientoService.test.js

PASS  __tests__/_ConsentimientoService.test.js
  Servicios de Consentimientos
    ✓ CRUD delegations (28 ms)
    ✓ generarConsentimientoPDF compone PDF, sube a cloudinary y retorna metadata (9 ms)
    ✘ actualizarPDFMetadata y obtenerPorId delegan al modelo (7 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        2.395 s
Ran all test suites matching __tests__/_ConsentimientoService.test.js/i.
dstevengmz1@Dstevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ █
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe procesar correctamente el formulario de contacto: validar entradas, delegar el envío al servicio (ContactoService.enviar) y responder con los códigos HTTP adecuados en escenarios válidos e inválidos, gestionando errores de servicio sin excepciones no controladas.
Objetivo	Verificar que el Controlador de Contacto: Valide presencia y formato de campos requeridos. Delegue el envío al servicio y devuelva 200 OK con messageld en éxito. Responda 400 ante entradas inválidas y 500 ante fallas del servicio (o integraciones).
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Endpoint: POST /contacto. Campos esperados: { nombre, email, asunto, mensaje } (strings). Servicio mockeado: ContactoService.enviar. Escenarios: éxito, faltantes/ inválidos, error del servicio. (Opcional) reCAPTCHA/token anti-bot si está habilitado.
Procedimiento de Prueba	Prueba 1 – Envío de mensaje exitoso Paso 1: Simular req.body válido:



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

{ nombre: "Constanza", email: "constanza@mail.com",
asunto: "Consulta", mensaje: "Hola, deseo información." }.

Paso 2: Mockear ContactoService.enviar para que
resuelva { messageld: "abc-123" }.

Paso 3: Ejecutar el controlador.

Resultado esperado: 200 OK y JSON { ok: true,
messageld: "abc-123" }. No se exponen detalles internos
del servicio.

Prueba 2 – Validaciones de entrada (400 Bad Request)

Caso A – Campos obligatorios faltantes

Paso 1: Enviar body sin uno o varios campos (p. ej. sin
email o mensaje).

Paso 2: Ejecutar el controlador.

Resultado esperado: 400 con lista/indicador de campos
requeridos faltantes.

Caso B – Email con formato inválido

Paso 1: Enviar email: "correo-invalido".

Paso 2: Ejecutar el controlador.

Resultado esperado: 400 con mensaje “Email inválido”.

Caso C – Longitudes mínimas/máximas

Paso 1: Enviar nombre o asunto demasiado cortos (p. ej.
< 2 chars) o mensaje demasiado corto (p. ej. < 10 chars).



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Repetir con mensaje excesivamente largo (p. ej. > 2000 chars).</p> <p>Resultado esperado: 400 indicando violación de longitudes.</p> <p>Caso D – Sanitización básica</p> <p>Paso 1: Enviar valores con espacios extra y HTML simple (holá).</p> <p>Paso 2: Ejecutar el controlador.</p> <p>Resultado esperado: Se trimean espacios; no se rompe el flujo (el detalle de escape/sanitización profunda se delega al servicio/capa inferior si aplica). Si política exige rechazo de HTML, 400.</p> <p>Caso E – (Opcional) Anti-bot / reCAPTCHA</p> <p>Paso 1: Simular token ausente/incorrecto si la validación está habilitada.</p> <p>Resultado esperado: 400 con mensaje de validación anti-bot.</p> <p>Prueba 3 – Error del servicio (500 Internal Server Error)</p> <p>Paso 1: Simular body válido.</p> <p>Paso 2: Mockear ContactoService.enviar para rechazar con un error (p. ej. timeout SMTP).</p> <p>Paso 3: Ejecutar el controlador.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Resultado esperado: 500 con mensaje genérico (“No se pudo enviar el mensaje”), sin filtrar stack traces.		
Datos de salida - Resultado Esperado	1 200 OK en envíos válidos con messageID retornado por el servicio. 2 400 Bad Request cuando faltan campos, hay formatos inválidos o violaciones de longitud. 3 500 Internal Server Error ante fallas del servicio de envío. 4 No hay excepciones no controladas; los mensajes de error son coherentes y seguros.		
Resultado Obtenido Prueba Exitosa	El controlador de Contacto valida entradas, delega al servicio y maneja correctamente respuestas y errores conforme a lo esperado.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

14. ContactoControl.js

Validar el correcto funcionamiento del **Controlador de Contacto**, encargado de procesar los datos del formulario de contacto, aplicar validaciones de entrada y delegar el envío de correo al servicio

Las pruebas garantizan la correcta respuesta HTTP para escenarios válidos e inválidos, y la gestión adecuada de errores de servicio.

Prueba:1 Envío de mensaje exitoso

Simula el envío exitoso del formulario de contacto con todos los campos válidos.

El servicio ContactoService.enviar devuelve un messageId, que se incluye en la respuesta JSON con estado **200 OK**.

```
test('200 y messageId cuando datos válidos', async () => {
  ContactoService.enviar.mockResolvedValue({ messageId: '123' });
  const { req, res } = mockReqRes({ body: { nombre: 'Ana', email: 'ana@test.com', asunto: 'Hola', me
  await ContactoController.enviar(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ ok: true, messageId: '123' });
  expect(ContactoService.enviar).toHaveBeenCalledWith({ nombre: 'Ana', email: 'ana@test.com', asunto
});
```

Prueba.2 Validaciones de entrada

```
test('400 cuando falta nombre', async () => {
  const { req, res } = mockReqRes({ body: { email: 'ana@test.com', mensaje: 'Mensaje' } });
  await ContactoController.enviar(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'El campo nombre es obligatorio' });
});
```

```
test('400 email inválido', async () => {
  const { req, res } = mockReqRes({ body: { nombre: 'Ana', email: 'mal', mensaje: 'Mensaje' } });
  await ContactoController.enviar(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'El email no es válido' });
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 141 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación:	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

```

test('400 asunto muy largo', async () => {
  const asunto = 'x'.repeat(151);
  const { req, res } = mockReqRes({ body: { nombre: 'Ana', email: 'ana@test.com', asunto, mensaje: '' } });
  await ContactoController.enviar(req, res);
  expect(res.statusCode).toBe(400);
  expect(res.body).toEqual({ error: 'El asunto es demasiado largo (máx 150)' });
});

```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba:3 Manejo de error en el servicio

Valida que si el servicio ContactoService.enviar lanza un error sin código de estado, el controlador responde con:

- Código HTTP **500 (Error interno)**.
- Mensaje de error genérico con el texto del Error.

```
test('500 cuando servicio lanza error sin status', async () => {
  ContactoService.enviar.mockRejectedValue(new Error('fail'))
  const { req, res } = mockReqRes({ body: { nombre: 'Ana', email: 'ana@test.com', mensaje: 'Mensaje' } })
  await ContactoController.enviar(req, res)
  expect(res.statusCode).toBe(500)
  expect(res.body).toEqual({ error: 'fail' })
})
});
```

Resultado:

```
dstevengz1@Dstevengz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/ContactoControllers.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/ContactoControllers.test.js

PASS _ tests /ContactoControllers.test.js
Controlador de Contacto - enviar
  ✓ 200 y messageId cuando datos válidos (27 ms)
  ✓ 400 cuando falta nombre (3 ms)
  ✓ 400-email inválido (22 ms)
  ✓ 400-asunto muy largo (3 ms)
  ✓ 500 cuando servicio lanza error sin status (5 ms)

Test Suites: 1 passed, 1 total
Tests:      5 passed, 5 total
Snapshots:  0 total
Time:       2.286 s
Ran all test suites matching _ tests /_ContactoControllers.test.js/i.
dstevengz1@Dstevengz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El servicio de contacto debe construir el transporte SMTP, sanitizar/escapar texto, componer el correo y enviarlo correctamente usando variables de entorno; debe manejar errores y cubrir todas las ramas lógicas: prioridad de SMTP_* sobre MAIL_*, secure true/false, auth presente/ausente, replyTo definido/indefinido, y destinatarios faltantes.
Objetivo	Asegurar la correcta creación del transporter, composición y envío del correo con nodemailer, incluida la normalización/sanitización del contenido.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones: buildTransporter(), enviar({ nombre, email, asunto, mensaje, to, replyTo? }), sanitizar(texto)/equivalente. ENV: SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_PASS, MAIL_HOST, MAIL_PORT, MAIL_USER, MAIL_PASSWORD. Mock: nodemailer.createTransport, sendMail.
Procedimiento de Prueba	Prueba 1 – Configuración del transporte (buildTransporter) Paso 1: Configurar ENV solo con SMTP_*.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Invocar buildTransporter() y verificar que prioriza SMTP_*.</p> <p>Resultado esperado: createTransport llamado con { host: SMTP_HOST, port: SMTP_PORT 587, secure: (SMTP_PORT==465) configurado, auth: { user: SMTP_USER, pass: SMTP_PASS } }.</p> <p>Caso B — Solo con MAIL_* (sin SMTP_*)</p> <p>Resultado esperado: Usa MAIL_* con puerto por defecto 587 si no se define; secure false si puerto ≠ 465; auth solo si hay credenciales.</p> <p>Caso C — Sin credenciales</p> <p>Resultado esperado: auth omitido (transporte sin auth).</p> <p>Caso D — Forzar secure: true y secure: false</p> <p>Resultado esperado: Se respetan ambas ramas y el puerto correspondiente.</p> <p>Prueba 2 – Envío de correos (enviar)</p> <p>Paso 1: Mockear transporter con sendMail resolviendo { messageId: "xyz-789" }.</p> <p>Paso 2: Llamar enviar con datos válidos (incluyendo to).</p> <p>Resultado esperado: Retorna { messageId: "xyz-789" } y sendMail recibe { from, to, subject, text, html, replyTo? } correctos.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Caso error — sendMail rechaza</p> <p>Resultado esperado: Lanza error controlado (propagado) con mensaje genérico.</p> <p>Prueba 3 – Sanitización y escape de texto</p> <p>Paso 1: Probar sanitizar con entradas con tildes, HTML simple y espacios extra.</p> <p>Paso 2: Llamar enviar y verificar que text/html usan el contenido normalizado (trim, escape mínimo, sin tags peligrosos).</p> <p>Resultado esperado: No se inyecta HTML malicioso; se normaliza y/o escapa.</p> <p>Prueba 4 – Ramas de destinatario configurado</p> <p>Caso A — to presente (string o array)</p> <p>Resultado esperado: Envía normalmente.</p> <p>Caso B — to ausente o vacío</p> <p>Resultado esperado: Lanza error de validación ("Destinatario requerido") antes de llamar sendMail.</p> <p>Caso C — replyTo definido/indefinido</p> <p>Resultado esperado: Si viene definido, se incluye; si no, se omite sin fallar.</p>
--	---

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 146 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Datos de salida - Resultado Esperado	1 buildTransporter respeta prioridad SMTP_* sobre MAIL_*; usa 587 por defecto; cubre secure true/false; incluye/omite auth según credenciales. 2 enviar compone y manda el correo; retorna messageld; propaga errores de envío de forma controlada. 3 El contenido es saneado/normalizado; no hay inyección ni HTML peligroso. 4 Se valida la presencia de destinatarios y se cubren ramas replyTo.		
Resultado Obtenido Prueba Exitosa	Prueba exitosa — El servicio de contacto crea el transporter correctamente, sanea contenidos, arma y envía correos con manejo adecuado de errores y todas las ramas cubiertas.	Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 147 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación:	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

15.ContactoSearces.test.js

Validar el comportamiento integral del **Servicio de Contacto**, encargado de construir el transporte SMTP o MAIL, sanitizar y escapar texto de entrada, componer los correos electrónicos, y enviarlos correctamente con las variables de entorno configuradas.

Las pruebas garantizan el correcto flujo de envío, manejo de errores y la cobertura completa de todas las ramas lógicas (replyTo definido o indefinido, destinatarios ausentes, normalización de texto y caracteres). Valida la creación correcta del **transporte SMTP** con nodemailer, considerando todas las combinaciones posibles de variables de entorno:

- Prioriza SMTP_* sobre MAIL_*.
- Configura secure y auth según presencia de credenciales.
- Usa puerto por defecto 587 cuando no hay configuración.

Cubre tanto secure: true como false.

Prueba 1.Configuración del transporte – buildTransporter()

```
test("buildTransporter usa variables de entorno y crea transport", async () => {
  const transportMock = { sendMail: jest.fn() };
  nodemailer.createTransport.mockReturnValue(transportMock);
  ContactoSearces = require("../services/ContactoSearces");
  const transporter = await ContactoSearces.buildTransporter();
  expect(nodemailer.createTransport).toHaveBeenCalledWith({
    host: "smtp.gmail.com",
    port: 587,
    secure: false,
    auth: undefined,
  });
  expect(transporter).toBe(transportMock);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 2.Envío de correos – enviar()

```
test("buildTransporter usa secure=true y auth cuando hay SMTP_USER y SMTP_PASS", async () => [
  const transportMock = { sendMail: jest.fn() };
  nodemailer.createTransport.mockReturnValue(transportMock);
  process.env.SMTP_SECURE = "true";
  process.env.SMTP_USER = "smtp_user@gmail.com";
  process.env.SMTP_PASS = "smtp_pass";
  ContactoServices = require("../services/ContactoServices");
  await ContactoServices.buildTransporter();
  expect(nodemailer.createTransport).toHaveBeenCalledWith(
    expect.objectContaining({
      secure: true,
      auth: { user: "smtp_user@gmail.com", pass: "smtp_pass" },
    })
  );
]);
```

```
test("enviar compone correo y llama sendMail", async () => [
  const transportMock = {
    sendMail: jest.fn().mockResolvedValue({ accepted: ["to@gmail.com"] }),
  };
  nodemailer.createTransport.mockReturnValue(transportMock);
  process.env.SMTP_USER = "from@gmail.com";
  process.env.CONTACT_EMAIL = "to@gmail.com";
  process.env.CONTACT_FROM_NAME = "Clinest";
  ContactoServices = require("../services/ContactoServices");
  const info = await ContactoServices.enviar({
    nombre: " <Juan> ",
    email: " user@site.com ",
    asunto: " Hola ",
    mensaje: "Lineal\nLinea2",
  });
  expect(transportMock.sendMail).toHaveBeenCalled();
  const args = transportMock.sendMail.mock.calls[0][0];
  expect(args.to).toBe("to@gmail.com");
  expect(args.from).toMatch(/Clinest/);
  expect(args.subject).toMatch(/Hola/);
  expect(args.text).toMatch(/Nombre:.*Juan/);
  expect(args.html).toMatch(/Lineal<br/>Linea2/);
  expect(info).toEqual({ accepted: ["to@gmail.com"] });
]);
```



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test("enviar lanza error si no hay destinatario en env", async () => {
  ContactoServices = require("../services/ContactoServices");
  await expect(
    ContactoServices.enviar({ nombre: "A", email: "a@b.com" })
  ).rejects.toThrow(/No hay destinatario configurado/);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

prueba 3.Sanitización y escape de texto

```
});
test("escapeHtml reemplaza todos los caracteres especiales correctamente", () => {
    ContactoServices = require("../services/ContactoServices");
    const texto = '<div> "hola" & \'adiós\' </div>';
    const esperado =
        "&lt;div&gt; &quot;hola&quot; &amp; #39;adiós&#39; &lt;/div&gt;";
    expect(ContactoServices.escapeHtml(texto)).toBe(esperado);
});
```

Prueba 4.Ramas de destinatario configurado

```
test("enviar cubre rama con destinatario configurado y email presente (replyTo definido)", async () =>
    // Mock del transporte
    const transportMock = {
        sendMail: jest.fn().mockResolvedValue({ accepted: ["ok@ok.com"] }),
    };
    nodemailer.createTransport.mockReturnValue(transportMock);

    // Configuramos entorno con CONTACT_EMAIL (para que no entre al if !to)
    process.env.CONTACT_EMAIL = "ok@ok.com";
    process.env.CONTACT_FROM_NAME = "Clinest";
    process.env.SMTP_USER = "";
    process.env.MAIL_USER = "";

    // Importamos servicio fresco
    delete require.cache[require.resolve("../services/ContactoServices")];
    const ContactoServices = require("../services/ContactoServices");

    // Ejecutamos enviar() con email lleno (replyTo definido)
    const resultado = await ContactoServices.enviar({
        nombre: "Usuario",
        email: "cliente@dominio.com",
        mensaje: "Prueba de envío",
    });

```

```
// Validaciones
expect(resultado).toEqual({ accepted: ["ok@ok.com"] });
const args = transportMock.sendMail.mock.calls[0][0];
expect(args.to).toBe("ok@ok.com");
expect(args.replyTo).toBe("cliente@dominio.com");
expect(args.from).toMatch(/Clinestetica|Clinest/);
expect(args.subject).toMatch(/Nuevo mensaje/);
expect(args.html).toMatch(/Usuario/);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Resultado:

```
istevengnz1@stevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/ContactoServices.test.js
- clinestetica@1.0.0 test
- jest --passWithNoTests _tests_/ContactoServices.test.js

PASS  tests/_ContactoServices.test.js (5.362 s)
  Servicios de Contacto
    buildTransporter usa variables de entorno y crea transporter (342 ms)
    buildTransporter usa secure=true y auth cuando hay SMTP_USER y SMTP_PASS (7 ms)
    buildTransporter cae a MAIL_+ cuando faltan SMTP_* (4 ms)
    buildTransporter usa puerto por defecto 587 y secure false cuando no hay puertos ni secure (5 ms)
    enviar lanza error si no hay destinatario en env (132 ms)
    enviar compone correo y llama sendMail (14 ms)
    enviar una asunto por defecto y replyTo:undefined con email vacío (7 ms)
    enviar escapa correctamente HTML en nombre y mensaje (9 ms)
    enviar con solo CONTACT_EMAIL usa from por defecto y normaliza CRLF (9 ms)
    enviar sin mensaje usa cadena vacía y sanitiza control chars en nombre/asunto (6 ms)
    escapeHTML reemplaza correctamente todos los caracteres especiales (4 ms)
    sanitizeText elimina caracteres de control y recorta espacios (2 ms)
    enviar no lanza error cuando CONTACT_EMAIL está configurado (6 ms)
    escapeHTML reemplaza todos los caracteres especiales correctamente (3 ms)
    sanitizeText elimina caracteres de control y espacios (4 ms)
    escapemail devuelve cadena vacía cuando se pasa undefined o null (3 ms)
    sanitizeText devuelve cadena vacía cuando se pasa undefined o null (2 ms)
    enviar usa replyTo undefined cuando email está vacío y lanza error si faltan destinatarios (7 ms)
    enviar cubre rama con destinatario configurado y email presente (replyTo definido) (5 ms)
    enviar cubre rama con destinatario configurado y sin email (replyTo undefined) (4 ms)
    enviar cubre rama con destinatario configurado (no lanza error) (5 ms)
    enviar cubre ramas restantes: con destinatario y replyTo undefined (4 ms)

Test Suites: 1 passed, 1 total
Tests:       22 passed, 22 total
Snapshots:  0 total
Time:        6.363 s
istevengnz1@stevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	Validar el correcto funcionamiento de la utilidad assets/corre.js para envío de correos: construcción del transporte SMTP con Nodemailer y envío con parámetros correctos (from, to, subject, text, html), leyendo adecuadamente variables de entorno.
Objetivo	Comprobar que: 1 se leen y aplican las variables de entorno para crear el transporter SMTP, y



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	2 se invoca sendMail con los valores esperados, retornando el messageld.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	<p>Módulo bajo prueba: assets/corre.js (función EnviarCorreo(payload) o similar).</p> <p>Mocks: nodemailer.createTransport y transporter.sendMail.</p> <p>ENV simuladas: MAIL_HOST, MAIL_PORT, MAIL_USER, MAIL_PASSWORD (o SMTP_* si tu util lo usa).</p> <p>Utilidades Jest: beforeEach, jest.resetModules(), jest.isolateModules(), jest.doMock().</p>
Procedimiento de Prueba	<p>Prueba 1 – Configuración del entorno y del transporte</p> <p>Paso 1: En beforeEach, definir variables de entorno (ejemplo):</p> <p>MAIL_HOST="smtp.test", MAIL_PORT="587", MAIL_USER="user@test", MAIL_PASSWORD="secret".</p> <p>Paso 2: Llamar jest.resetModules() para limpiar caché de require.</p> <p>Paso 3: Usar jest.isolateModules() + jest.doMock("nodemailer", ...) para mockear createTransport, devolviendo un objeto transporter falso con sendMail: jest.fn().</p> <p>Paso 4: Requerir dentro del bloque aislado el módulo assets/corre.js para que tome el mock y las ENV.</p> <p>Resultado esperado: createTransport es invocado una vez con un objeto que contenga:</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>host: "smtp.test", port: 587 (número),</p> <p>auth: { user: "user@test", pass: "secret" },</p> <p>(si tu util configura secure, validar rama correspondiente: false para 587).</p> <p>Prueba 2 – Envío del correo con parámetros correctos</p> <p>Paso 1: Dentro del mismo aislamiento, preparar sendMailMock = jest.fn().mockResolvedValue({ messageId: "abc-123" }) y usarlo en el transporter simulado.</p> <p>Paso 2: Importar { EnviarCorreo } desde assets/corre.js.</p> <p>Paso 3: Ejecutar EnviarCorreo({ recipients: "a@b.com", subject: "Hola", text: "Texto", html: "<p>Texto</p>", from?: opcional }).</p> <p>Paso 4: Esperar la promesa y capturar el retorno.</p> <p>Resultado esperado:</p> <p>sendMail es llamado una vez con un objeto que incluya:</p> <p>from (el valor por defecto de la utilidad o el pasado en payload),</p> <p>to: "a@b.com",</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>subject: "Hola",</p> <p>text: "Texto",</p> <p>html: "<p>Texto</p>".</p> <p>El resultado de EnviarCorreo incluye { messageld: "abc-123" }.</p> <p>No se filtran errores si sendMail resuelve OK.</p> <p>(Caso alterno opcional) Si sendMail rechaza, la utilidad debe propagar o mapear el error a un mensaje controlado (verificar throw o return de error según diseño).</p>		
Datos de salida - Resultado Esperado	<p>1 createTransport usa las ENV definidas y construye el transporter con host/port/auth/secure correctos.</p> <p>2 sendMail recibe los parámetros (from, to, subject, text, html) exactamente como espera la utilidad.</p> <p>3 La función retorna el messageld del envío exitoso.</p> <p>4 No hay excepciones no controladas; los errores de sendMail se manejan según el diseño del módulo.</p>		
Resultado Obtenido Prueba Exitosa	La utilidad assets/corre.js leyó correctamente las variables de entorno, construyó el transporte SMTP	Si(<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

y llamó a sendMail con los parámetros esperados, retornando el messageld.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

16 corre.test.js

Validar el correcto funcionamiento de la **utilidad de envío de correos** (assets/corre.js), la cual construye un transporte SMTP usando **Nodemailer** y envía mensajes con los parámetros adecuados (from, to, subject, text, html).

El objetivo es garantizar que se lean correctamente las variables de entorno y que el correo se envíe con los valores esperados.

Antes de cada prueba, se definen las variables de entorno necesarias para simular la conexión SMTP.

Luego, se aísla el módulo nodemailer mediante jest.isolateModules y jest.doMock para interceptar la creación del transporte y controlar el mock del método sendMail.

Prueba 1. Configuración del entorno y del transporte

```
describe('Utilidad EnviarCorreo', () => {
  beforeEach(() => {
    jest.resetModules();
    process.env.MAIL_HOST = "smtp.test";
    process.env.MAIL_PORT = "25";
    process.env.MAIL_USER = "user";
    process.env.MAIL_PASSWORD = "pass";
  });
});
```

Prueba 2. Envío del correo con parámetros correctos



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test("llama sendMail con los parámetros correctos", async () => {
  const sendMailMock = jest.fn().mockResolvedValue({ messageId: "abc" });
  let EnviarCorreo;
  jest.isolateModules(() => {
    jest.doMock("nodemailer", () => ({
      createTransport: () => ({ sendMail: sendMailMock }),
    }));
    ({ EnviarCorreo } = require("../assets/corre"));
  });

  EnviarCorreo({ recipients: "a@b.com", subject: "Hola", message: "Test" });
  await new Promise((r) => setImmediate(r));

  expect(sendMailMock).toHaveBeenCalledWith({
    from: '"Clinestetica" <juanmenxz9@gmail.com>',
    to: "a@b.com",
    subject: "Hola",
    text: "Test",
    html: "Test",
  });
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 158 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultado:

```
dstevengnz1@dstevengnz:~/Escritorio/Backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends$ npm test -- _tests_/corre.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/corre.test.js

PASS  tests/_corre.test.js
  Utilidad EnviarCorreo
    ✓ llama sendMail con los parametros correctos (153 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.893 s
Ran all test suites matching _tests_/_corre.test.js/i.
dstevengnz1@dstevengnz:~/Escritorio/Backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends$
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 159 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe gestionar el flujo completo de archivos de exámenes clínicos desde el controlador: subida, listado por cita, eliminación y descarga segura, validando autorización y tipo de archivo, y devolviendo códigos HTTP coherentes (201, 200, 400, 403, 404, 415, 500).
Objetivo	Verificar que el Controlador de Exámenes: Delegue a los servicios la subida (con Multer), listado, eliminación y descarga. Aplique validaciones de entrada, autorización (doctor/propietario/asistente permitido), existencia de cita/archivo y tipo MIME permitido. Responda con los códigos HTTP y mensajes adecuados.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: subir, listarPorCita, eliminar, descargarSeguro. Mocks: middleware de auth (req.usuario), Multer (archivos en req.files), servicio de exámenes (subir/listar/eliminar/descargar), util de firma o acceso seguro si aplica. Parámetros: citald, examenId, cabeceras Authorization.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Tipos MIME permitidos (ej.: pdf, jpg, jpeg, png).
Procedimiento de Prueba	<p>Prueba 1 – Subida de exámenes: subir()</p> <p>Paso 1: Simular req.usuario autorizado y req.files con al menos un archivo válido (application/pdf o imagen).</p> <p>Paso 2: Mockear servicio subirArchivos para resolver con metadatos de los archivos.</p> <p>Paso 3: Ejecutar controlador subir.</p> <p>Resultado esperado (éxito): 201 Created con JSON de archivos subidos.</p> <p>Caso 400 – Faltan archivos / validación falla</p> <p>Paso 1: Enviar req.files vacío o con error de validación.</p> <p>Paso 2: Mockear servicio para lanzar error de validación.</p> <p>Resultado esperado: 400 Bad Request con mensaje de validación.</p> <p>Caso 500 – Error interno</p> <p>Paso 1: Forzar rechazo del servicio por fallo inesperado.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p> <p>Prueba 2 – Listado por cita: listarPorCita()</p> <p>Paso 1: Enviar params.citald válido y usuario con permisos.</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Mockear servicio listarPorCita retornando arreglo de exámenes.</p> <p>Paso 3: Ejecutar controlador.</p> <p>Resultado esperado (éxito): 200 OK con lista de exámenes.</p> <p>Caso 404 – Cita sin exámenes / no existe</p> <p>Paso 1: Mockear servicio para devolver lista vacía o indicar inexistencia.</p> <p>Resultado esperado: 404 Not Found con mensaje informativo.</p> <p>Caso 500 – Error interno</p> <p>Paso 1: Forzar excepción en el servicio.</p> <p>Resultado esperado: 500 con mensaje genérico.</p> <p>Prueba 3 – Eliminación de exámenes: eliminar()</p> <p>Paso 1: Enviar params.examenId válido y usuario autorizado (doctor/propietario/rol permitido).</p> <p>Paso 2: Mockear servicio eliminar retornando eliminación exitosa.</p> <p>Paso 3: Ejecutar controlador.</p> <p>Resultado esperado (éxito): 200 OK con mensaje “Examen eliminado”.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Caso 404 – Examen no existe</p> <p>Paso 1: Mockear servicio para indicar que no hay registro a eliminar.</p> <p>Resultado esperado: 404 Not Found.</p> <p>Caso 500 – Error interno</p> <p>Paso 1: Forzar error del servicio.</p> <p>Resultado esperado: 500 Internal Server Error.</p> <p>Prueba 4 – Descarga segura: descargarSeguro()</p> <p>Paso 1: Enviar params.examenId, usuario autenticado con permisos; mockear servicio para devolver metadatos (ruta, mime).</p> <p>Paso 2: Validar tipo de archivo permitido (pdf/jpg/jpeg/png) y autorización (doctor/propietario/asistente).</p> <p>Paso 3: Simular flujo de stream/buffer y seteo de headers (Content-Type, Content-Disposition: attachment).</p> <p>Resultado esperado (éxito): 200 OK y cuerpo binario/stream; headers correctos.</p> <p>Caso 403 – Sin permisos</p> <p>Paso 1: Simular usuario sin rol/relación válida.</p> <p>Resultado esperado: 403 Forbidden.</p> <p>Caso 404 – Archivo no existe</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Paso 1: Mockear servicio para devolver null o inexistencia. Resultado esperado: 404 Not Found. Caso 415 – Tipo no permitido Paso 1: Devolver mime no permitido (p. ej. application/x-exe). Resultado esperado: 415 Unsupported Media Type. Caso 500 – Falla de descarga Paso 1: Forzar error en lectura/stream. Resultado esperado: 500 Internal Server Error.		
Datos de salida - Resultado Esperado	1 Subida: 201 en éxito; 400 por validaciones; 500 por fallas internas. 2 Listado: 200 con datos; 404 sin registros; 500 por error de servicio. 3 Eliminación: 200 si elimina; 404 si no existe; 500 en error. 4 Descarga segura: 200 con headers correctos; 403/404/415/500 según el caso. 5 No hay excepciones no controladas; se valida autorización y tipos MIME.		
Resultado Obtenido Prueba Exitosa	El Controlador de Exámenes gestiona subida, listado, eliminación y descarga segura con validaciones, permisos y códigos HTTP correctos.	Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 165 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

17. ExamenControllers.test.js

Validar el correcto funcionamiento del **Controlador de Exámenes**, encargado de gestionar el flujo completo de archivos asociados a exámenes clínicos: subida, listado por cita, eliminación, y descarga segura (con validación de autorización y tipo de archivo).

Estas pruebas aseguran que las operaciones de CRUD y descarga funcionen correctamente y que las respuestas HTTP sean coherentes según el escenario.

Evalúa el comportamiento del método de subida:

- **201 Created:** cuando el servicio subirArchivos responde correctamente.
- **400 Bad Request:** cuando el servicio lanza un error o faltan archivos.

Prueba 1. Subida de exámenes – subir()

```
test('subir -> 201 y 400', async () => {
  examenService.subirArchivos.mockResolvedValue([{"id": 1}]);
  let ctx = mockReqRes({ params: { id_cita: 9 }, files: [{ path: '/a' }] });
  await controller.subir(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(201);
  expect(ctx.res.body.examenes).toEqual([{ id: 1 }]);

  examenService.subirArchivos.mockRejectedValue(new Error('bad'));
  ctx = mockReqRes({ params: { id_cita: 9 }, files: [] });
  await controller.subir(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
});
```

Prueba 2. Listado por cita – listarPorCita()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test('listarPorCita -> 200 y 500', async () => {
  examenService.listarPorCita.mockResolvedValue([ { id: 1 } ]);
  let ctx = mockReqRes({ params: { id_cita: 2 } });
  await controller.listarPorCita(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  examenService.listarPorCita.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id_cita: 2 } });
  await controller.listarPorCita(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```

Prueba 3.Eliminación de exámenes – eliminar()

```
test('eliminar -> 404, 200 y 500', async () => [
  examenService.eliminar.mockResolvedValue(false);
  let ctx = mockReqRes({ params: { id: 3 } });
  await controller.eliminar(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);

  examenService.eliminar.mockResolvedValue(true);
  ctx = mockReqRes({ params: { id: 3 } });
  await controller.eliminar(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  examenService.eliminar.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 3 } });
  await controller.eliminar(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
]);
```

Prueba 4.Descarga segura – descargarSeguro()



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

```
test('descargarSeguro -> autorización y tipos de URL', async () => {
    // no registro
    examen.findByPk.mockResolvedValue(null);
    let ctx = mockReqRes({ params: { id: 5 }, usuario: { id: 10, rol: 'usuario' } });
    await controller.descargarSeguro(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(404);

    // registro y cita, pero no autorizado
    examen.findByPk.mockResolvedValue({ id: 5, id_cita: 7, archivo_examen: 'file.pdf' });
    citas.findByPk.mockResolvedValue({ id: 7, id_usuario: 99, id_doctor: 55 });
    ctx = mockReqRes({ params: { id: 5 }, usuario: { id: 10, rol: 'usuario' } });
    await controller.descargarSeguro(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(403);

    // autorizado paciente y legacy http
    examen.findByPk.mockResolvedValue({ id: 5, id_cita: 7, archivo_examen: 'http://legacy/url' });
    citas.findByPk.mockResolvedValue({ id: 7, id_usuario: 10, id_doctor: 55 });
    ctx = mockReqRes({ params: { id: 5 }, usuario: { id: 10, rol: 'usuario' } });
    await controller.descargarSeguro(ctx.req, ctx.res);
    expect(ctx.res.statusCode).toBe(200);
    expect(ctx.res.body).toEqual({ url: 'http://legacy/url', legacy: true });
});
```



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
// autorizado doctor y archivo pdf
examen.findByPk.mockResolvedValue({ id: 5, id_cita: 7, archivo_examen: 'folder/fi
citas.findByPk.mockResolvedValue({ id: 7, id_usuario: 10, id_doctor: 55 });
ctx = mockReqRes({ params: { id: 5 }, usuario: { id: 55, rol: 'doctor' } });
await controller.descargarSeguro(ctx.req, ctx.res);
expect(ctx.res.statusCode).toBe(200);
expect(ctx.res.body.url).toBe('https://signed/pdf');

// autorizado asistente y archivo imagen
examen.findByPk.mockResolvedValue({ id: 5, id_cita: 7, archivo_examen: 'folder/i
citas.findByPk.mockResolvedValue({ id: 7, id_usuario: 10, id_doctor: 55 });
ctx = mockReqRes({ params: { id: 5 }, usuario: { id: 1, rol: 'asistente' } });
await controller.descargarSeguro(ctx.req, ctx.res);
expect(ctx.res.statusCode).toBe(200);
expect(ctx.res.body.url).toBe('https://signed/image');

});
```

Resultado:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
Archivo Editar Ver Terminal Preferencias Ayuda
istevengnz1@stevengnz1:~/Escritorio/hackend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _ tests _/correo.test.js
+ clinestetica@1.0.0 test
+ jest --passWithNoTests _ tests _/correo.test.js

PASS  _ tests _/correo.test.js
  Utilidad EnviarCorreo
    ✓ llama sendMail con los parámetros correctos (153 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.893 s
All test suites matching _ tests _/correo.test.js/:
istevengnz1@stevengnz1:~/Escritorio/hackend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _ tests _/ExamenControllers.test.js
+ clinestetica@1.0.0 test
+ jest --passWithNoTests _ tests _/ExamenControllers.test.js

PASS  _ tests _/ExamenControllers.test.js
  Controlador de Exámenes
    ✓ subir > 200 y 400 (20 ms)
    ✓ listarPorCita > 200 y 500 (3 ms)
    ✓ eliminar > #04, 200 y 300 (3 ms)
    ✓ descargarSeguro > autorización y tipos de URL (11 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.6 s
All test suites matching _ tests _/ExamenControllers.test.js/:
istevengnz1@stevengnz1:~/Escritorio/hackend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	
Requerimiento Funcional de la prueba	



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Objetivo			
Tipo de Prueba	Prueba Unitaria (Jest)		
Datos de entrada de la prueba			
Procedimiento de Prueba			
Datos de salida - Resultado Esperado			
Resultado Obtenido Prueba Exitosa		Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 4. Descarga segura – descargarSeguro()

```
2 test('subirArchivos -> crea registros para cada archivo', async () => [
3   models.citas.findByPk.mockResolvedValue({ id: 1, examenes_cargados: false });
4   models.examen.create.mockImplementation(async (data) => data);
5   const archivos = [
6     { originalname: 'A.pdf', filename: 'pub1', path: '', secure_url: '', url: '' },
7     { originalname: 'B.png', filename: 'pub2', path: '', secure_url: '', url: '' }
8   ];
9   const res = await svc.subirArchivos({ id_cita: 1, archivos });
10  expect(res).toHaveLength(2);
11  expect(models.examen.create).toHaveBeenCalledTimes(2);
12];
13);
14 test('subirArchivos -> error si falta public id (filename) en algún archivo', async
```

Prueba 2: Listado por cita – listarPorCita()

```
test('listarPorCita -> delega en examen.findAll', async () => {
  models.examen.findAll.mockResolvedValue([ { id: 1 } ]);
  const res = await svc.listarPorCita(7);
  expect(models.examen.findAll).toHaveBeenCalledWith({ where: { id_cita: 7 }, order: [] });
  expect(res).toEqual([ { id: 1 } ]);
};

test('eliminar -> retorna null si no existe y true si elimina', async () => [
  models.examen.findByPk.mockResolvedValue(null);
  let res = await svc.eliminar(9);
  expect(res).toBeNull();

  const destroy = jest.fn();
  models.examen.findById.mockResolvedValue({ id: 9, destroy });
  res = await svc.eliminar(9);
  expect(destroy).toHaveBeenCalled();
  expect(res).toBe(true);
];
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3. Eliminación de exámenes – eliminar()

```
test('eliminar -> retorna null si no existe y true si elimina', async () => {
  models.examen.findByPk.mockResolvedValue(null);
  let res = await svc.eliminar(9);
  expect(res).toBeNull();

  const destroy = jest.fn();
  models.examen.findByPk.mockResolvedValue({ id: 9, destroy });
  res = await svc.eliminar(9);
  expect(destroy).toHaveBeenCalled();
  expect(res).toBe(true);
});
```

```
dstevengmz1@dstevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/ExamenServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/ExamenServices.test.js

PASS _tests_/ExamenServices.test.js
  Servicios de Exámenes
    ✓ subirArchivos -> error sin archivos (215 ms)
    ✓ subirArchivos -> error si cita no existe (6 ms)
    ✓ subirArchivos -> error si examenes_cargados true (6 ms)
    ✓ subirArchivos -> crea registros para cada archivo (7 ms)
    ✓ subirArchivos -> error si falta public_id (filename) en algún archivo (5 ms)
    ✓ listarPorCita -> delega en examen.findAll (23 ms)
    ✓ eliminar -> retorna null si no existe y true si elimina (9 ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:  0 total
Time:        5.245 s
Ran all test suites matching _tests_\\ExamenServices.test.js/i.
dstevengmz1@dstevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Resultado:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El sistema debe gestionar correctamente los Historiales Médicos desde el controlador: listar, buscar por ID, buscar por usuario, obtener el propio, crear, actualizar y eliminar, aplicando validaciones de entrada, control de permisos (usuario/doctor/asistente) y devolviendo códigos HTTP coherentes (200/201/400/403/404/500).
Objetivo	Verificar que el Controlador de Historial Médico: Delegue en el servicio y maneje errores sin excepciones no controladas. Valide IDs y datos obligatorios. Respete permisos (el usuario solo accede a su historial; el doctor/asistente según reglas). Devuelva respuestas JSON con códigos correctos.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: listarHistorialMedico, buscarHistorialMedico, buscarHistorialMedicoporUsuario, miHistorialMedico, crearHistorialMedico, actualizarHistorialMedico, eliminarHistorialMedico. Mocks: servicio de historiales (listar/buscar/crear/actualizar/eliminar), req.usuario (roles y id), validaciones de parámetros/ body. Parámetros: id (historial), usuariold (para búsquedas), body con datos clínicos mínimos.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Procedimiento de Prueba	Prueba 1 – Listar historiales médicos: listarHistorialMedico() Paso 1: Simular solicitud GET sin filtros (o con filtros válidos). Paso 2: Mockear servicio para devolver arreglo de historiales. Paso 3: Ejecutar controlador. Resultado esperado (éxito): 200 OK con lista de historiales. Errores: si el servicio falla → 500 con mensaje genérico. Prueba 2 – Buscar historial por ID: buscarHistorialMedico() Paso 1: Enviar params.id numérico. Paso 2: Mockear servicio para devolver un historial válido. Paso 3: Ejecutar controlador. Resultado esperado (éxito): 200 OK con el historial. 400: id inválido (no numérico). 404: historial no existe. 500: error del servicio. Prueba 3 – Buscar historial por usuario: buscarHistorialMedicoporUsuario() Paso 1: Enviar params.usuarioid válido.
-------------------------	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Mockear servicio para devolver historiales del usuario.</p> <p>Paso 3: Ejecutar controlador.</p> <p>Resultado esperado (éxito): 200 OK con lista del usuario.</p> <p>400: usuariold inválido.</p> <p>404: sin registros para ese usuario (si la regla lo contempla como 404).</p> <p>500: error del servicio.</p> <p> </p> <p>Prueba 4 – Historial médico propio: miHistorialMedico()</p> <p>Paso 1: Simular req.usuario.id autenticado.</p> <p>Paso 2: Mockear servicio para devolver historiales del autenticado.</p> <p>Paso 3: Ejecutar controlador.</p> <p>Resultado esperado (éxito): 200 OK con historiales del propio usuario.</p> <p>403: si el rol/estado no permite acceso (si aplica política).</p> <p>500: error del servicio.</p> <p> </p> <p>Prueba 5 – Crear historial médico: crearHistorialMedico()</p> <p>Paso 1: Preparar req.body con datos obligatorios (p. ej. usuariold, motivo/diagnóstico inicial, fecha).</p> <p>Paso 2: Validar permisos (doctor/asistente pueden crear para un usuario; el usuario quizás solo su propio historial, según regla).</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Paso 3: Mockear servicio para creación exitosa.</p> <p>Resultado esperado (éxito): 201 Created con el historial creado.</p> <p>400: faltan campos obligatorios / formato inválido.</p> <p>403: usuario sin permiso para crear ese historial.</p> <p>500: error del servicio.</p>
	<p>Prueba 6 – Actualizar historial médico: actualizarHistorialMedico()</p> <p>Paso 1: Enviar params.id válido y body con cambios (p. ej. notas/diagnóstico/indicaciones).</p> <p>Paso 2: Validar permisos (doctor o propietario según regla).</p> <p>Paso 3: Mockear servicio para actualización exitosa.</p> <p>Resultado esperado (éxito): 200 OK confirmando actualización.</p> <p>400: id inválido, datos vacíos o “no se realizaron cambios”.</p> <p>403: sin permisos para actualizar ese historial.</p> <p>404: historial no existe.</p> <p>500: error del servicio.</p>
	<p>Prueba 7 – Eliminar historial médico: eliminarHistorialMedico()</p> <p>Paso 1: Enviar params.id válido y usuario con permisos (p. ej. admin/doctor autorizado).</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Paso 2: Mockear servicio para eliminación exitosa. Paso 3: Ejecutar controlador. Resultado esperado (éxito): 200 OK con mensaje “Historial eliminado”. 400: id inválido. 403: sin permisos para eliminar. 404: historial no encontrado. 500: error del servicio.		
Datos de salida - Resultado Esperado	1 200/201 en operaciones válidas; 400/403/404/500 según validaciones, permisos o errores internos. 2 IDs y cuerpos validados (tipos, campos requeridos, formatos). 3 Permisos respetados (acceso al propio historial, o por rol). 4 Respuestas JSON claras; sin excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	El controlador de Historial Médico gestionó correctamente listados, búsquedas, creación, actualización y eliminación, con validaciones, permisos y códigos HTTP esperados.	Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 179 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

19 . HistorialMedicoControllers.test.js

Verificar la correcta funcionalidad del **Controlador de Historial Médico**, encargado de gestionar las operaciones CRUD sobre los historiales clínicos y sus validaciones asociadas.

Las pruebas aseguran el manejo adecuado de errores, validaciones de entrada, control de permisos (autorización de usuario) y estados HTTP coherentes.

Valida que el método devuelva correctamente una lista de historiales clínicos cuando el servicio responde exitosamente.

Prueba 1.star historiales médicos – listarHistorialMedico()

```
test('listarHistorialMedico -> 200', async () => {
  svc.listarLosHistorialesClinicos.mockResolvedValue([{ id: 1 }]);
  const { req, res } = mockReqRes();
  await controller.listarHistorialMedico(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual([{ id: 1 }]);
});
```

Prueba 2.Buscar historial médico – buscarHistorialMedico()

```
test('buscarHistorialMedico -> 200 y 404', async () => {
  svc.buscarLosHistorialesClinicos.mockResolvedValue({ id: 2 });
  let ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLosHistorialesClinicos.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 3. Buscar historial por usuario – buscarHistorialMedicoporUsuario()

```
test('buscarHistorialMedico -> 200 y 404', async () => {
  svc.buscarLosHistorialesClinicos.mockResolvedValue({ id: 2 });
  let ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLosHistorialesClinicos.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
});
```

Prueba 4. Historial médico propio – miHistorialMedico()

```
test('miHistorialMedico -> 403 si id distinto, 404 si no hay, 200 si existe y 500 en error', async () => {
  let ctx = mockReqRes({ params: { id: 11 }, usuario: { id: 10 } });
  await controller.miHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(403);

  svc.buscarLosHistorialesClinicosPorUsuario.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 10 }, usuario: { id: 10 } });
  await controller.miHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);

  svc.buscarLosHistorialesClinicosPorUsuario.mockResolvedValue({ id: 1 });
  ctx = mockReqRes({ params: { id: 10 }, usuario: { id: 10 } });
  await controller.miHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLosHistorialesClinicosPorUsuario.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 10 }, usuario: { id: 10 } });
  await controller.miHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 182 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Prueba 5. Crear historial médico – crearHistorialMedico()

```

test('crearHistorialMedico -> 201 y 500', async () => {
  svc.crearLosHistorialesClinicos.mockResolvedValue({ id: 9 });
  let ctx = mockReqRes({ body: { a: 1 } });
  await controller.crearHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(201);

  svc.crearLosHistorialesClinicos.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ body: { a: 1 } });
  await controller.crearHistorialMedico(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});

```

Prueba 6. Actualizar historial médico – actualizarHistorialMedico()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test('miHistorialMedico -> 403 si id distinto, 404 si no hay, 200 si existe y 500 en
let ctx = mockReqRes({ params: { id: 11 }, usuario: { id: 10 } });
await controller.miHistorialMedico(ctx.req, ctx.res);
expect(ctx.res.statusCode).toBe(403);

svc.buscarLosHistorialesClinicosPorUsuario.mockResolvedValue(null);
ctx = mockReqRes({ params: { id: 10 }, usuario: { id: 10 } });
await controller.miHistorialMedico(ctx.req, ctx.res);
expect(ctx.res.statusCode).toBe(404);

svc.buscarLosHistorialesClinicosPorUsuario.mockResolvedValue({ id: 1 });
ctx = mockReqRes({ params: { id: 10 }, usuario: { id: 10 } });
await controller.miHistorialMedico(ctx.req, ctx.res);
expect(ctx.res.statusCode).toBe(200);

svc.buscarLosHistorialesClinicosPorUsuario.mockRejectedValue(new Error('x'));
ctx = mockReqRes({ params: { id: 10 }, usuario: { id: 10 } });
await controller.miHistorialMedico(ctx.req, ctx.res);
expect(ctx.res.statusCode).toBe(500);
});
```

Prueba 7.Eliminar historial médico – eliminarHistorialMedico()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test('eliminarHistorialMedico -> 200', async () => {
  const { req, res } = mockReqRes({ params: { id: 1 } });
  await controller.eliminarHistorialMedico(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ message: 'Historialmedico eliminado' });
  expect(svc.eliminarLosHistorialesClinicos).toHaveBeenCalledWith(1);
});
});
```

Resultado

```
dstevengmz1@dstevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/HistorialMedicoControllers.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/HistorialMedicoControllers.test.js

PASS  _tests_/HistorialMedicoControllers.test.js
Controlador de Historial Médico
  ✓ listarHistoriaUMedico -> 200 (18 ms)
  ✓ buscarHistorialMedico -> 208 y 404 (4 ms)
  ✓ buscarHistoriaUMedicoporUsuario -> 208 y 404 (4 ms)
  ✓ niHistorialMedico -> 403 si id distinto, 404 si no hay, 200 si existe y 500 en error (9 ms)
  ✓ crearHistorialMedico -> 201 y 500 (5 ms)
  ✓ actualizarHistorialMedico -> 400 id inválido, 404 y 200, 500 (6 ms)
  ✓ eliminarHistorialMedico -> 200 (7 ms)

Test Suites: 1 passed, 1 total
Tests:    7 passed, 7 total
Snapshots: 0 total
Time:    2.388 s
Ran all test suites matching _tests_/HistorialMedicoControllers.test.js/i.

dstevengmz1@dstevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	Validar los flujos de acceso y validaciones internas del Controlador de Historial Médico en escenarios críticos de seguridad y errores: Control de acceso al historial médico propio. Validación del parámetro id en actualización.
Objetivo	Asegurar que el controlador: solo permita al usuario autenticado acceder a su propio historial, respetando reglas de rol, y valide correctamente el ID en las operaciones de actualización, rechazando valores inválidos antes de delegar al servicio.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: miHistorialMedico, actualizarHistorialMedico. Mocks: servicio de históricos (retornos/errores), req.usuario (id/rol), validación de parámetros. Parámetros: params.id (numérico), req.usuario.id, body con cambios mínimos.
Procedimiento de Prueba	Prueba 1 – Acceso a historial médico propio: miHistorialMedico()



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Simular req.usuario = { id: 10, rol: "usuario" }.</p> <p>Paso 2: Mockear servicio para devolver historiales pertenecientes al usuario 10.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado (éxito): 200 OK con el/los historiales del usuario autenticado.</p> <p>Caso 403 – Acceso no permitido (si aplica política):</p> <p>Simular un rol/estado que no tenga permiso para ver su propio historial (o flag de cuenta restringida).</p> <p>Resultado esperado: 403 Forbidden.</p> <p>Caso 500 – Error interno:</p> <p>Forzar error del servicio.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p> <p>Prueba 2 – Validación de ID en actualización: actualizarHistorialMedico()</p> <p>Paso 1: Enviar params.id = "abc" (no numérico) y un body cualquiera.</p> <p>Paso 2: Ejecutar el controlador.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: 400 Bad Request con mensaje “ID inválido”.</p> <p>Paso 3: Enviar params.id = 25 (válido) pero body vacío o sin cambios.</p> <p>Resultado esperado: 400 Bad Request con mensaje “Datos inválidos” o “No se realizaron cambios”.</p> <p>Paso 4: Enviar params.id = 25, body válido y usuario sin permiso para actualizar ese historial.</p> <p>Resultado esperado: 403 Forbidden.</p> <p>Paso 5: Enviar params.id = 25, body válido y servicio responde éxito.</p> <p>Resultado esperado (éxito): 200 OK confirmando la actualización.</p> <p>Paso 6: Forzar error del servicio (excepción).</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p>		
Datos de salida - Resultado Esperado	1 miHistorialMedico devuelve 200 con datos del autenticado; 403 si política lo restringe; 500 en falla del servicio. 2 actualizarHistorialMedico valida id y body: 400 en inválidos/sin cambios; 403 sin permisos; 200 al actualizar; 500 en error interno. 3 Sin excepciones no controladas; mensajes y códigos consistentes.		
Resultado Obtenido Prueba Exitosa	El controlador de Historial Médico aplica correctamente los controles de acceso al propio historial y la validación del ID en actualización,	Si(<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	respondiendo con códigos HTTP adecuados.		
--	--	--	--

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 189 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 190 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

20.HistorialMedicoControllers.unit.test.js

Validar los **flujos de acceso y validaciones internas** del Controlador de Historial Médico, enfocándose en los casos más críticos de seguridad y manejo de errores:

- Control de acceso al historial médico propio.
- Validación del parámetro id en la actualización.

Estas pruebas aseguran que el sistema maneje correctamente los casos donde un usuario intenta acceder a información que no le pertenece o envía datos inválidos.

Evaluá las condiciones de seguridad al consultar el historial del usuario autenticado.

Prueba 1. Acceso a historial médico propio – miHistorialMedico()

```
test("miHistorialMedico: bloquea acceso si id != token (403)", async () => {
  const req = { params: { id: "2" }, usuario: { id: 1 } };
  const res = mockRes();
  await HistorialController.miHistorialMedico(req, res);
  expect(res.status).toHaveBeenCalledWith(403);
});
```

```
test("miHistorialMedico: retorna historial si id coincide (200)", async () => {
  const req = { params: { id: "5" }, usuario: { id: 5 } };
  const res = mockRes();
  histService.buscarLosHistorialesClinicosPorUsuario.mockResolvedValueOnce({ id: 5 });
  await HistorialController.miHistorialMedico(req, res);
  expect(res.json).toHaveBeenCalledWith({ id: 5 });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 2. Validación de ID en actualización – actualizarHistorialMedico()

```
test("actualizarHistorialMedico: id inválido retorna 400", async () => {
  const req = { params: { id: "abc" }, body: {} };
  const res = mockRes();
  await HistorialController.actualizarHistorialMedico(req, res);
  expect(res.status).toHaveBeenCalledWith(400);
  expect(res.json).toHaveBeenCalledWith({ error: "ID inválido" });
});
```

Resultado:

```
dstevengmz1@stevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/HistorialMedicoControllers.unit.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/HistorialMedicoControllers.unit.test.js

PASS  tests /HistorialMedicoControllers.unit.test.js
  Controlador de Historial Médico (unit)
    ✓ miHistorialMedico: bloquea acceso si id != token (403) (28 ms)
    ✓ miHistorialMedico: retorna historial si id coincide (200) (5 ms)
    ✓ actualizarHistorialMedico: id inválido retorna 400 (6 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        2.096 s
Run all test suites matching _tests_/_HistorialMedicoControllers.unit.test.js/i.
dstevengmz1@stevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ dstevengmz1@stevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El servicio de Historial Médico debe ejecutar correctamente la lógica de negocio para listar, buscar, crear, actualizar y eliminar historiales clínicos, garantizando: Integridad de datos y control de duplicidad por usuario. Trazabilidad (logs en éxito/error). Uso de include para traer datos asociados (p. ej., Usuario). Propagación controlada de errores del ORM.
Objetivo	Validar que las funciones del servicio interactúen correctamente con los modelos Sequelize (Historial, Usuario) usando findAll, findByPk, create, update, destroy, apliquen validaciones/duplicidad y manejen errores sin excepciones no controladas.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones: listarLosHistorialesClinicos, buscarLosHistorialesClinicos, buscarLosHistorialesClinicosPorUsuario, crearLosHistorialesClinicos, actualizarLosHistorialesClinicos, eliminarLosHistorialesClinicos. Mocks: Historial.findAll/findByPk/create/update/destroy, Usuario (para include), util de logs (opcional).



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Datos: id, usuariold, cuerpo clínico mínimo (motivo/diagnóstico/fecha).
Procedimiento de Prueba	<p>Prueba 1 – Listar historiales: listarLosHistorialesClinicos()</p> <p>Paso 1: Mockear Historial.findAll para devolver arreglo de historiales con Usuario.</p> <p>Paso 2: Ejecutar el servicio.</p> <p>Resultado esperado (éxito): Retorna la lista completa e invoca findAll con include: [{ model: Usuario, as: "usuario" }]</p> <p>Resultado esperado (error): Si findAll falla, el servicio propaga el error (con log si existe).</p> <p>Prueba 2 – Buscar por ID: buscarLosHistorialesClinicos(id)</p> <p>Paso 1: Mockear Historial.findByPk(id, { include }) para devolver un historial.</p> <p>Paso 2: Ejecutar el servicio con id válido.</p> <p>Resultado esperado (éxito): Retorna el historial con su Usuario incluido.</p> <p>404 lógico: Si findByPk devuelve null, el servicio retorna null o lanza error “Historial no encontrado” (según diseño).</p> <p>Error: Si el ORM falla, propaga el error.</p> <p>Prueba 3 – Buscar por usuario: buscarLosHistorialesClinicosPorUsuario(usuariold)</p> <p>Paso 1: Mockear Historial.findAll({ where: { usuariold }, include }).</p>



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Ejecutar el servicio.</p> <p>Resultado esperado (éxito): Retorna historiales del usuariold con Usuario incluido.</p> <p>Vacio: Retorna [] si no hay registros.</p> <p>Error: Propaga fallo del ORM.</p> <p> </p> <p>Prueba 4 – Crear historial: crearLosHistorialesClinicos(datos)</p> <p>Paso 1: Mockear validación de duplicidad por usuario (p. ej. Historial.findOne({ where: { usuariold, ...criterio } })).</p> <p>Paso 2: Caso duplicado: findOne devuelve match.</p> <p>Resultado esperado: Lanza error de negocio “Ya existe un historial para este usuario/criterio”.</p> <p>Paso 3: Caso éxito: findOne → null y Historial.create(datos) exitoso.</p> <p>Resultado esperado: Retorna el historial creado con campos requeridos.</p> <p>Error: Si create falla, propaga el error (con log).</p> <p> </p> <p>Prueba 5 – Actualizar historial: actualizarLosHistorialesClinicos(id, cambios)</p> <p>Paso 1: Mockear Historial.findByPk(id) → devuelve registro existente.</p> <p>Paso 2: Validar cambios (no vacío / campos permitidos).</p> <p>Paso 3: Mockear Historial.update(cambios, { where: { id } }) o instancia.update(cambios).</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado (éxito): Retorna confirmación o el objeto actualizado (según implementación).</p> <p>400 lógico: Si cambios vacío o sin campos válidos, lanza “Datos inválidos / No se realizaron cambios”.</p> <p>404 lógico: Si findByPk → null, lanza “Historial no encontrado”.</p> <p>Error: Propaga fallo del ORM.</p> <p> </p> <p>Prueba 6 – Eliminar historial: eliminarLosHistorialesClinicos(id)</p> <p>Paso 1: Mockear Historial.destroy({ where: { id } }).</p> <p>Paso 2: Caso éxito: destroy devuelve 1.</p> <p>Resultado esperado: Retorna confirmación de eliminación (true/contador).</p> <p>Paso 3: Caso no encontrado: destroy devuelve 0.</p> <p>Resultado esperado: Lanza o retorna estado “Historial no encontrado” (según diseño).</p> <p>Error: Si destroy falla, propaga el error.</p>
Datos de salida - Resultado Esperado	<p>1 listar* usa include de Usuario y retorna lista; propaga errores.</p> <p>2 buscar por ID/usuario retorna datos (o null/error 404 lógico) con include.</p> <p>3 crear* previene duplicados por usuario y crea en éxito.</p> <p>4 actualizar* valida input, confirma actualización, maneja 404 y errores del ORM.</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	5 eliminar* confirma eliminación, maneja no-encontrado y errores. 6 Se registran logs (si existen) y no hay excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	El Servicio de Historial Médico listó, buscó, creó, actualizó y eliminó históricos con include de usuario, control de duplicados y manejo de errores conforme a lo esperado.	Si(x)	No ()

21. HistorialMedicoServices.test.js

Validar la funcionalidad del **Servicio de Historial Médico**, encargado de la lógica de negocio relacionada con la gestión de históricos clínicos.

Se evalúan los procesos de búsqueda, creación, actualización y manejo de errores, asegurando integridad de datos, trazabilidad en logs y control de duplicidad por usuario.

Verifica que el método obtenga todos los históricos clínicos e incluya los datos del usuario asociado mediante include.

Prueba 1.Listar históricos – listarLosHistorialesClinicos()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
test('listarLosHistorialesClinicos incluye datos de usuario', async () => {
  models.historialclinico.findAll.mockResolvedValue([ { id: 1 } ]);
  const res = await svc.listarLosHistorialesClinicos();
  expect(models.historialclinico.findAll).toHaveBeenCalledWith({
    include: [
      model: models.usuarios,
      as: 'usuario',
      attributes: [
        'nombre', 'correo', 'telefono', 'direccion', 'fecha_nacimiento', 'genero', 'r
      ],
    ],
  });
  expect(res).toEqual([ { id: 1 } ]);
});
```

Prueba 2. Buscar por ID – buscarLosHistorialesClinicos()

```
test('buscarLosHistorialesClinicos obtiene por id con include', async () => {
  models.historialclinico.findById.mockResolvedValue( { id: 7 } );
  const res = await svc.buscarLosHistorialesClinicos(7);
  expect(models.historialclinico.findById).toHaveBeenCalledWith(7, expect.objectContaine
  expect(res).toEqual( { id: 7 } );
});
```

```
test('buscarLosHistorialesClinicos captura error y loggea', async () => {
  models.historialclinico.findById.mockRejectedValue(new Error('db error'));
  const res = await svc.buscarLosHistorialesClinicos(9);
  expect(console.log).toHaveBeenCalled(expect.stringContaining('Error en el servidor al buscar e
  expect(res).toBeUndefined();
});
```

Prueba 3. Buscar por usuario – buscarLosHistorialesClinicosPorUsuario()



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('buscarLosHistorialesClinicosPorUsuario captura error y loggea', async () => {
  models.historialclinico.findOne.mockRejectedValue(new Error('db error'));
  const res = await svc.buscarLosHistorialesClinicosPorUsuario(4);
  expect(console.log).toHaveBeenCalledWith(expect.stringContaining('Error en el servidor al buscar'));
  expect(res).toBeUndefined();
});
```

Prueba 4. Crear historial – crearLosHistorialesClinicos()

```
test('crearLosHistorialesClinicos lanza si el usuario ya tiene historial', async () => {
  models.historialclinico.findOne.mockResolvedValue({ id: 5, id_usuario: 11 });
  await expect(svc.crearLosHistorialesClinicos({ id_usuario: 11, dato: 'x' })).rejects.toThrow('El usuario ya tiene un historial médico registrado.');
});
```

Prueba 5. Actualizar historial – actualizarLosHistorialesClinicos()

```
test('actualizarLosHistorialesClinicos actualiza correctamente', async () => {
  models.historialclinico.update.mockResolvedValue([1]);
  const res = await svc.actualizarLosHistorialesClinicos(20, { campo: 'valor' });
  expect(models.historialclinico.update).toHaveBeenCalledWith({ campo: 'valor' }, { where: { id: 20 } });
  expect(res).toEqual([1]);
});
```

```
test('actualizarLosHistorialesClinicos captura error y loggea', async () => {
  models.historialclinico.update.mockRejectedValue(new Error('update fail'));
  const res = await svc.actualizarLosHistorialesClinicos(21, { a: 1 });
  expect(console.log).toHaveBeenCalledWith(expect.stringContaining('Error en el servidor al actualizar el Historial'));
  expect(res).toBeUndefined();
});
```

Prueba 6. Eliminar historial – eliminarLosHistorialesClinicos()

```
test('eliminarLosHistorialesClinicos actualmente produce ReferenceError por sombreado de variable', async () => {
  await expect(svc.eliminarLosHistorialesClinicos(5)).rejects.toBeInstanceOf(ReferenceError);
});
```



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Resultado:

```
  tests ./HistorialMedicoServices.test.js (6.299 s)
  Servicios de Historial Médico
    ✓ listarLosHistorialesClinicos incluye datos de usuario (1953 ms)
    ✓ buscarLosHistorialesClinicos obtiene por id con include (8 ms)
    ✓ buscarLosHistorialesClinicos captura error y loggea (7 ms)
    ✓ buscarLosHistorialesClinicosPorUsuario retorna uno con include (5 ms)
    ✓ buscarLosHistorialesClinicosPorUsuario captura error y loggea (4 ms)
    ✓ crearLosHistorialesClinicos lanza si el usuario ya tiene historial (56 ms)
    ✓ crearLosHistorialesClinicos crea nuevo cuando no existe (6 ms)
    ✓ actualizarLosHistorialesClinicos actualiza correctamente (4 ms)
    ✓ actualizarLosHistorialesClinicos captura error y loggea (3 ms)
    ✓ eliminarLosHistorialesClinicos actualmente produce ReferenceError por sobrebladeo de variable (6 ms)

  ✘ Servicios de Historial Médico -> eliminarLosHistorialesClinicos actualmente produce ReferenceError por sobrebladeo de variable
    expect(Object).rejects.toBeInstanceOf(expected)
      Expected constructor: ReferenceError
      Received constructor: None

      test(
        it 'expect(svc.eliminarLosHistorialesClinicos) rejects toBeInstanceOf(referenceError)', async () => {
          expect(svc.eliminarLosHistorialesClinicos).rejects.toBeInstanceOf(ReferenceError);
        }
      )

      at Object.toBeInstanceOf (node modules/expect/build/index.js:218:22)
      at Object.toBeInstanceOf (tests ./HistorialMedicoServices.test.js:106:85)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 9 passed, 10 total
Snapshots:  0 total
Time:        6.901 s
 Ran all test suites matching ./tests ./HistorialMedicoServices.test.js/i.
stevenmz@Ostevengmz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$
```

Fecha de realización:

15-09-2025**Duración de la prueba: 5 min**



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Requerimiento Funcional de la prueba	La utilidad HorariosDisponibles debe: Generar una secuencia ordenada de horas hábiles desde 08:00 hasta 18:00 con incrementos regulares de 30 minutos, excluyendo la franja de almuerzo 12:00. Exponer la configuración de duraciones por tipo de cita y mantener consistencia con el incremento base (múltiplos de 30).
Objetivo	Verificar que: horarios() devuelva una grilla consistente (límite inicial/final, orden ascendente, salto fijo de 30 min, exclusión de 12:00 y sin duplicados), y duraciones() mapee correctamente evaluación → 30 minutos y procedimiento → 150 minutos, respetando múltiplos de 30.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones bajo prueba: horarios(), duraciones(). Supuestos del negocio: jornada 08:00–18:00, paso de 30 minutos, excluir 12:00 (pausa), tipos: evaluacion, procedimiento.
Procedimiento de Prueba	Prueba 1 – Generación de horarios: horarios() Paso 1: Invocar const lista = horarios(). Paso 2: Verificar que lista[0] === "08:00" (límite inferior). Paso 3: Verificar que existe "18:00" en lista (límite superior). Paso 4: Confirmar que no existe "12:00" en lista (pausa de almuerzo).



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 5: Comprobar que la lista esté estrictamente ordenada en forma ascendente.</p> <p>Paso 6: Validar que no haya duplicados y que todos los elementos cumplan el formato "HH:mm".</p> <p>Paso 7: Para cada par consecutivo, calcular el delta y verificar que sea exactamente 30 minutos (p. ej., 08:00→08:30, ..., 11:30→(salta 12:00)→12:30, ..., 17:30→18:00).</p> <p>Resultado esperado: horarios() devuelve una secuencia de marcas de tiempo desde 08:00 hasta 18:00, en pasos de 30 minutos, sin "12:00", ordenada, sin duplicados y con deltas uniformes.</p> <p>Prueba 2 – Duraciones de cita: duraciones()</p> <p>Paso 1: Invocar const map = duraciones().</p> <p>Paso 2: Verificar que map.evalucion === 30.</p> <p>Paso 3: Verificar que map.procedimiento === 150.</p> <p>Paso 4: Validar consistencia: todos los valores del mapa deben ser enteros positivos y múltiplos de 30 (para encajar en la grilla de horarios()).</p> <p>Paso 5: (Opcional) Si existen más tipos, iterar y validar el múltiplo de 30.</p> <p>Resultado esperado: duraciones() retorna un mapeo que incluye evaluación: 30 y procedimiento: 150 (ambos múltiplos de 30), asegurando compatibilidad con la grilla de horarios.</p>
Datos de salida - Resultado Esperado	1 horarios() → lista 08:00 ... 18:00, paso 30 min, sin 12:00, ordenada y sin duplicados.

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 202 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	2 duraciones() → mapeo evaluacion: 30, procedimiento: 150, todos múltiplos de 30 y valores válidos. 3 Coherencia entre la grilla y las duraciones (las citas encajan exactamente en los slots).		
Resultado Obtenido Prueba Exitosa	La utilidad HorariosDisponibles genera horarios consistentes con la política de la jornada y expone duraciones compatibles con incrementos de 30 minutos.	Si(x)	No ()

23. HorariosDisponibles.test.js

Verificar la **generación y consistencia lógica de los horarios disponibles** y la **configuración de duraciones** para los distintos tipos de citas en el sistema.

Esta utilidad es clave en la programación de citas médicas, asegurando coherencia en las franjas horarias y tiempos asignados por tipo de servicio

Confirma que la función horarios() genere una secuencia ordenada de horas desde **08:00 hasta 18:00**, con incrementos regulares de 30 minutos, y que excluya la pausa de almuerzo (12:00).

También valida la presencia de horas límite y consistencia interna de incrementos.

Prueba 1. Generación de horarios – horarios()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
const Horarios = require('../assets/HorariosDisponibles');

describe('HorariosDisponibles - utilidades de agenda', () => {
  test('horarios retorna lista esperada (08:00 a 18:00 con intervalos de 30m)', () => {
    const hs = Horarios.horarios();
    expect(Array.isArray(hs)).toBe(true);
    expect(hs.length).toBeGreaterThanOrEqual(1);
    expect(hs[0]).toBe('08:00');
    // En este proyecto hay pausa de almuerzo (no incluye 12:00)
    expect(hs.includes('12:00')).toBeFalsy();
    expect(hs.at(-1)).toBe('18:00');

    // Verificar presencia de 11:30 y 13:00 (pausa)
    expect(hs.includes('11:30')).toBeTruthy();
    expect(hs.includes('13:00')).toBeTruthy();

    // Verificar incremento de 30 min entre algunos consecutivos (muestra)
    const toMin = (s) => {
      const [h, m] = s.split(':').map(Number);
      return h * 60 + m;
    };
    const idx = hs.indexOf('10:00');
    if (idx > 0) {
      expect(toMin(hs[idx + 1]) - toMin(hs[idx])).toBe(30);
    }
  });
});
```

Prueba 2.Duraciones de cita – duraciones()

```
test('duraciones retorna mapa esperado', () => {
  const d = Horarios.duraciones();
  expect(d).toEqual({ evaluacion: 30, procedimiento: 60 });
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 204 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultado:

```
dstevengmz1@Dstevengmz:~/Escritorio/Backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/HorariosDisponibles.test.js
> climestetica@1.0.0 test
> jest --passWithNoTests _tests_/HorariosDisponibles.test.js

PASS  _tests_/HorariosDisponibles.test.js
  HorariosDisponibles - utilidades de agenda
    ✓ horarios retorna lista esperada (08:00 a 18:00 con intervalos de 30m) (19 ms)
    ✓ duraciones retorna mapa esperado (7 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        2.038 s
Ran all test suites matching _tests_/HorariosDisponibles.test.js/i.

dstevengmz1@Dstevengmz:~/Escritorio/Backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	<p>El middleware de seguridad de intentos fallidos debe prevenir ataques de fuerza bruta usando Redis para contar, bloquear y limpiar intentos por correo electrónico.</p> <p>Debe aplicar el umbral de bloqueo y el TTL de ventana configurados (p. ej. MAX_INTENTOS, TTL_BLOQUEO), retornando el código HTTP correspondiente al bloqueo (típicamente 429 Too Many Requests, o el definido por la política).</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Objetivo	Verificar que el middleware: verifique el número de intentos y permita/bloquee acceso según el umbral, registre intentos fallidos incrementando el contador y configurando TTL, y limpie el contador al producirse un inicio de sesión exitoso.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones/middlewares: verificarIntentos, registrarIntentoFallido, limpiarIntentos. Mocks de Redis: get, incr, expire, del (o setEx), con key tipo login:attempts:<email> Config: MAX_INTENTOS, TTL_BLOQUEO (en segundos), prefijo de keys. Request simulado: req.body.email.
Procedimiento de Prueba	Prueba 1 – Verificación de intentos: verificarIntentos() Paso 1: Simular req.body.email = "user@test.com". Paso 2: Caso A (debajo del umbral): mock Redis.get → "2" con MAX_INTENTOS = 5. Ejecutar middleware. Resultado esperado: Llama next() (permite continuar), no responde error. Paso 3: Caso B (en el umbral o por encima): mock Redis.get → "5" (\geq MAX_INTENTOS). Ejecutar middleware.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: Responde bloqueo con código configurado (p. ej. 429), cuerpo JSON con mensaje tipo "Demasiados intentos. Intente más tarde."; no llama next().</p> <p>Paso 4: Caso C (sin registro): mock Redis.get → null.</p> <p>Resultado esperado: Llama next() (sin bloqueos).</p> <p>Prueba 2 – Registro de intentos: registrarIntentoFallido()</p> <p>Paso 1: Simular fallo de login para email = "user@test.com".</p> <p>Paso 2: Caso A (primer intento del período): mock Redis.get → null; Redis.incr → 1. Ejecutar función.</p> <p>Resultado esperado: Se ejecuta incr(key) y se establece TTL del período con expire(key, TTL_BLOQUEO) (o setEx). El contador queda en 1.</p> <p>Paso 3: Caso B (intento subsecuente): mock Redis.get → "3"; Redis.incr → 4.</p> <p>Resultado esperado: incr incrementa a 4; no debe resetear TTL si ya existe (o lo renueva según política documentada—validar rama).</p> <p>Paso 4: Caso C (en/por encima del umbral): mock Redis.get → "4"; incr → 5 con MAX_INTENTOS = 5.</p> <p>Resultado esperado: El contador alcanza el umbral; la siguiente llamada a verificarIntentos debe bloquear (ver</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Prueba 1-B). La función puede devolver el valor actual o un indicador de restantes si está implementado.</p> <p>Prueba 3 – Limpieza de intentos: limpiarIntentos()</p> <p>Paso 1: Simular login exitoso para email = "user@test.com".</p> <p>Paso 2: Ejecutar limpiarIntentos(req, res).</p> <p>Resultado esperado: Llama Redis.del(key) y retorna (o permite continuar) sin error.</p> <p>Caso de error Redis (opcional): Si del rechaza, se registra el fallo (log) pero no debe romper el flujo de autenticación ya exitosa (según política).</p>		
Datos de salida - Resultado Esperado	verificarIntentos bloquea cuando el conteo es \geq MAX_INTENTOS y permite en caso contrario. registrarIntentoFallido incrementa el contador y configura TTL al primer fallo del período (manteniéndolo en sucesivos intentos). limpiarIntentos elimina la key de Redis tras un login exitoso. Códigos HTTP coherentes (p. ej. 429 para bloqueo) y sin excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	El middleware de intentos fallidos controla conteo, bloqueo y limpieza con Redis conforme a la configuración de seguridad, devolviendo estados HTTP correctos y manteniendo la estabilidad del sistema.	Si(<input checked="" type="checkbox"/>)	No (<input type="checkbox"/>)

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 208 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación:	1/02/2025	Elaborado por: Instructores área Software
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

24.

intentosfallidos.test.js

Validar el correcto funcionamiento del **middleware de seguridad que controla los intentos fallidos de autenticación**, evitando ataques de fuerza bruta mediante Redis. Las pruebas comprueban el conteo, bloqueo y limpieza de intentos por correo electrónico.

Evaluá el comportamiento del middleware ante solicitudes de login según el número de intentos registrados en Redis.

Prueba 1. Verificación de intentos – verificarIntentos()

```
test('verificarIntentos permite cuando no hay intentos', async () => {
  const req = { body: { correo: 'a@b.com' } };
  const res = makeRes();
  const next = jest.fn();
  await verificarIntentos(req, res, next);
  expect(next).toHaveBeenCalled();
});

test('verificarIntentos bloquea cuando supera MAX_INTENTOS', async () => {
  const correo = 'x@y.com';
  const key = `intentos:${correo}`;
  await redis.setEx(key, 300, '3');
  const req = { body: { correo } };
  const res = makeRes();
  const next = jest.fn();
  await verificarIntentos(req, res, next);
  expect(res.status).toHaveBeenCalledWith(429);
  expect(next).not.toHaveBeenCalled();
});
```

Prueba 2. Registro de intentos – registrarIntentoFallido()

```
test('registrarIntentoFallido crea y aumenta contador', async () => {
  const correo = 'z@z.com';
  const key = `intentos:${correo}`;
  await registrarIntentoFallido(correo);
  expect(await redis.get(key)).toBe('1');
  await registrarIntentoFallido(correo);
  expect(await redis.get(key)).toBe('2');
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 3. Limpieza de intentos – limpiarIntentos()

```
6 test('limpiarIntentos elimina el registro', async () => [
7   const correo = 'w@w.com';
8   const key = `intentos:${correo}`;
9   await registrarIntentoFallido(correo);
10  await limpiarIntentos(correo);
11  expect(await redis.get(key)).toBeUndefined();
12]);
13});
```

Resultado:

```
dstevengnz1@dstevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/intentosfallidos.test.js
> clinetestica@1.0.0 test
> jest --passWithNoTests _tests_/intentosfallidos.test.js

PASS _tests_/intentosfallidos.test.js
Middleware intentosfallidos (rate limit)
  ✓ verificarIntentos permite cuando no hay intentos (15 ms)
  ✓ verificarIntentos bloquea cuando supera MAX_INTENTOS (9 ms)
  ✓ registrarIntentoFallido crea y aumenta contador (6 ms)
  ✓ limpiarIntentos elimina el registro (4 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:  0 total
Time:        2.132 s
Ran all test suites matching _tests_/intentosfallidos.test.js/i.
dstevengnz1@dstevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización:	Duración de la prueba: 5 min		
15-09-2025			
Requerimiento Funcional de la prueba			
Objetivo			
Tipo de Prueba	Prueba Unitaria (Jest)		
Datos de entrada de la prueba			
Procedimiento de Prueba			
Datos de salida - Resultado Esperado			
Resultado Obtenido Prueba Exitosa		Si(x)	No ()

23. LimpiarNombreUtils.test.

Verificar la correcta **normalización y formateo de nombres** dentro del sistema, asegurando una presentación coherente para usuarios, doctores y pacientes, eliminando inconsistencias en espacios, mayúsculas y acentos

Verifica que la utilidad:

- Elimine los espacios al inicio, al final y entre palabras repetidos.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

- Capitalice la primera letra de cada palabra.
- Mantenga el resto de las letras en minúscula.

Prueba 1. Capitalización y limpieza de espacios

```
describe('Utilidad formatearNombre', () => {
  it('capitaliza cada palabra y quita espacios extra', () => {
    | expect(formatearNombre(' juan perez gomez ')).toBe('Juan Perez Gomez');
  });
});
```

Prueba 2. Manejo de mayúsculas, minúsculas y acentos

```
it('maneja nombres con mayúsculas/minúsculas mixtas', () => {
  | expect(formatearNombre('mArÍa loPeZ')).toBe('María Lopez');
});
```

Prueba 3. Entrada vacía o con solo espacios

```
'retorna cadena vacía si recibe solo espacios', () => {
  | expect(formatearNombre('   ')).toBe('');
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 212 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultado:

```
dstevengnz1@stevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$ npm test -- _tests_/LimpiarNombreUtils.test.js

> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/LimpiarNombreUtils.test.js

PASS  tests/_LimpiarNombreUtils.test.js
  Utilidad formatearNombre
    ✓ capitaliza cada palabra y quita espacios extra (13 ms)
    ✓ maneja nombres con mayúsculas/minúsculas mixtas (3 ms)
    ✓ retorna cadena vacía si recibe solo espacios (2 ns)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        1.987 s
Ran all test suites matching _tests/_LimpiarNombreUtils.test.js/i.
dstevengnz1@stevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	
Objetivo	
Tipo de Prueba	Prueba Unitaria (Jest)

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 213 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Datos de entrada de la prueba			
Procedimiento de Prueba			
Datos de salida - Resultado Esperado			
Resultado Obtenido Prueba Exitosa		Si(x)	No ()

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 214 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

24. Multer.test.js

Verificar el **cableado correcto del middleware Multer** encargado de manejar la carga de imágenes de procedimientos, garantizando que se configure con **CloudinaryStorage** usando la carpeta apropiada y las extensiones válidas, además de validar la generación del `public_id` dinámico.

Verifica que al importar el middleware:

- Se haya llamado correctamente a `multer()` una vez.
- Su configuración incluya un atributo `storage` asociado a `CloudinaryStorage`.
- El módulo `../config/cloudinary` esté correctamente inyectado en el almacenamiento.

Prueba 1.Creación del middleware con CloudinaryStorage

```
test('crea upload con CloudinaryStorage configurado', () => {
  const multer = require('multer');
  const { CloudinaryStorage } = require('multer-storage-cloudinary');
  const upload = require('../middleware/Multer');
```

Prueba 2.Validación de parámetros del almacenamiento

```
// params para procedimientos
const params = storageArgs.params;
expect(params.folder).toBe('procedimientos');
expect(params.allowed_formats).toEqual(expect.arrayContaining(['jpg', 'png', 'jpeg']));
const publicId = params.public_id({}, {fieldname: 'foto'});
expect(typeof publicId).toBe('string');
```

Prueba 3.Generación del identificador público (public_id)

```
const publicId = params.public_id({}, {fieldname: 'foto'});
expect(typeof publicId).toBe('string');
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 215 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultado:

```
dstevengnz1@Ostevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$ npm test -- _tests_/Multer.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/Multer.test.js

PASS _tests_/Multer.test.js
  Multer (procedimientos) - cableado de middleware
    ✓ crea upload con CloudinaryStorage configurado (88 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        3.265 s
Ran all test suites matching /_tests_/Multer.test.js/i.
dstevengnz1@Ostevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	
Requerimiento Funcional de la prueba	Validar la configuración y el cableado del middleware MulterExamenes para subir archivos clínicos (PDFs o imágenes) a Cloudinary, asegurando:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Límite de tamaño por archivo = 10 MB.</p> <p>Validación de tipos MIME permitidos (PDF/JPG/JPEG/PNG)</p> <p>Generación correcta de parámetros de almacenamiento según el tipo de archivo (folder, public_id, resource_type/format).</p> <p>Exportación del middleware listo para usar en rutas.</p>
Objetivo	Comprobar que MulterExamenes se inicializa con multer() usando storage (CloudinaryStorage), fileFilter y limits, valida MIME, respeta el límite de 10 MB y exporta la instancia correctamente.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Módulos mockeados: multer, multer-storage-cloudinary (CloudinaryStorage), cloudinary Archivo simulado: { originalname, mimetype, buffer }. Tipos válidos: application/pdf, image/jpeg, image/jpg, image/png. Llimate: 10 * 1024 * 1024 bytes.
Procedimiento de Prueba	Prueba 1 – Creación del middleware Paso 1: Mockear multer para capturar la llamada de inicialización. Paso 2: Requerir el módulo MulterExamenes (aislando módulos si es necesario). Paso 3: Inspeccionar la llamada a multer({ storage, fileFilter, limits }). Resultado esperado:



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

multer es llamado una vez con un objeto que tiene las claves storage, fileFilter, limits.

limits.fileSize === 10 * 1024 * 1024.

Prueba 2 – Configuración del almacenamiento (CloudinaryStorage)

Paso 1: Mockear CloudinaryStorage para registrar las params generadas por archivo.

Paso 2: Simular subida de PDF (application/pdf).

Paso 3: Simular subida de imagen (image/jpeg).

Resultado esperado (PDF):

resource_type acorde (comúnmente "raw" para PDFs) o configuración que soporte PDF,

folder tipo "clinica/exámenes" (o el folder configurado),

format: "pdf" (si aplica), public_id derivado de originalname (sanitizado).

Resultado esperado (Imagen):

resource_type: "image",



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

folder: "clinica/exámenes",

format acorde ("jpg"/"png"), public_id coherente.

(Ajusta los nombres de folder/format a tu implementación exacta si difieren; la prueba debe reflejar tus valores reales.)

Prueba 3 – Validación de tipos MIME (fileFilter)

Paso 1: Invocar el fileFilter con mimetype válido: application/pdf, image/jpeg, image/png.

Resultado esperado: Llama cb(null, true) (acepta el archivo).

Paso 2: Invocar el fileFilter con mimetype inválido (p. ej. application/zip).

Resultado esperado: Llama cb(new Error("Tipo de archivo no permitido"), false) o rechaza conforme a tu mensaje de error; el archivo no se acepta.

(Opcional) Exceso de tamaño: Si tu test simula el límite, verifica que Multer emite error LIMIT_FILE_SIZE cuando fileSize supera 10MB.

Prueba 4 – Exportación correcta del middleware

Paso 1: Importar el export de MulterExámenes.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	Paso 2: Verificar que es una instancia de multer (p. ej., posee métodos .single, .array, .fields). Resultado esperado: El módulo exporta el middleware listo para usar en rutas (por ejemplo, multerExamenes.array("files") o el que definas).		
Datos de salida - Resultado Esperado	1 multer() se inicializa con storage, fileFilter y limits.fileSize = 10MB. 2 CloudinaryStorage recibe params correctos por tipo (PDF → raw/pdf, Imagen → image/jpg png), con folder y public_id coherentes. 3 fileFilter acepta PDFs/imagenes válidas y rechaza tipos no permitidos con error claro 4 El export es el middleware utilizable en rutas (tiene los métodos de multer). 5 No hay excepciones no controladas; errores de validación se comunican adecuadamente.		
Resultado Obtenido Prueba Exitosa	MulterExamenes aplica límites, valida MIME y configura CloudinaryStorage correctamente, exportando el middleware listo para integrarse en las rutas de subida.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

25.MulterExamenes.test.js

alidar la configuración y correcto cableado del middleware **MulterExamenes**, encargado de subir archivos clínicos (PDFs o imágenes) a **Cloudinary**, asegurando que:

- Se apliquen los límites de tamaño adecuados.
- Se validen los tipos MIME permitidos.
- Se generen correctamente los parámetros de almacenamiento según el tipo de archivo.

Confirma que el middleware se inicializa correctamente:

- Usa multer() con las propiedades esperadas (storage, fileFilter, limits).
- Define el límite máximo de tamaño en **10 MB** por archivo.

Prueba 1.Creación del middleware

```
test('crea upload con fileFilter y limits; params usa raw para pdf y image para otros', async
  const multer = require('multer');
  const { CloudinaryStorage } = require('multer-storage-cloudinary');
  const upload = require('../middleware/MulterExamenes');

  // se invocó multer una vez
  expect(multer).toHaveBeenCalledTimes(1);
  const arg = multer.mock.calls[0][0];
  expect(arg).toHaveProperty('storage');
  expect(arg).toHaveProperty('fileFilter');
  expect(arg).toHaveProperty('limits');
  expect(arg.limits).toHaveProperty('fileSize', 10 * 1024 * 1024);
```

Prueba 2.Configuración del almacenamiento (CloudinaryStorage)

```
// CloudinaryStorage creado
expect(CloudinaryStorage).toHaveBeenCalledTimes(1);
expect(createdStorageOpts).toHaveProperty('params');
const paramsFn = createdStorageOpts.params;

// Caso PDF
const pdfParams = await paramsFn({}, { mimetype: 'application/pdf', originalname: 'reporte.pdf' });
expect(pdfParams.folder).toBe('examenes');
expect(pdfParams.resource_type).toBe('raw');
expect(pdfParams.type).toBe('private');
expect(pdfParams.format).toBe('pdf');
expect(typeof pdfParams.public_id).toBe('string');
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 221 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Prueba 3. Validación de tipos MIME (fileFilter)

```
// fileFilter acepta pdf y jpeg
const resAccept = { ok: false };
await new Promise((resolve) => arg.fileFilter({}, { mimetype: 'application/pdf' }), (err, v) => {
  resAccept.ok = true;
  resAccept.value = v;
});

// fileFilter rechaza mimetype no permitido
await new Promise((resolve) => arg.fileFilter({}, { mimetype: 'text/plain' }), (err, v) => {
  resAccept.ok = false;
  resAccept.value = v;
});
```

prueba 4. Exportación correcta del middleware

```
// el upload exportado es objeto tipo middleware de multer mockeado
expect(upload._upload).toBe(true);
});
```

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Front dstevengnz@dstevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests/_MulterExamenes.test.js

> clinestetica@1.0.8 test
> jest --passWithNoTests _tests/_MulterExamenes.test.js

PASS  tests/_MulterExamenes.test.js
  MulterExamenes - cableado de middleware
    crea upload con fileFilter y limits; params usa raw para pdf y image para otros (64 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        1.319 s, estimated 3 s
Run all test suites matching _tests/_MulterExamenes.test.js/i
dstevengnz@dstevengnz:~/Escritorio/backend/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ █
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Requerimiento Funcional de la prueba	El controlador de Órdenes debe gestionar correctamente las operaciones de listar, crear, actualizar y eliminar órdenes, devolviendo estados HTTP coherentes (200, 201, 400, 404, 500) según entradas, existencia de recursos y fallas de servicio.
Objetivo	Verificar que cada método del controlador: Valide parámetros (IDs numéricos, usuario autenticado). Delegue en el servicio y maneje errores internos sin excepciones no controladas. Responda con el código HTTP y payload adecuados.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: listarOrdenesElegiblesParaProcedimiento, listarMisOrdenes, listarOrdenes, buscarOrdenes, crearOrdenes, actualizarOrdenes, eliminarOrdenes. Mocks: servicio de órdenes (listar/buscar/crear/actualizar/eliminar), req.usuario (id), validadores de params/body. Parámetros: usuarioid, id de orden, body con datos de orden (p.ej. procedimientoid, totales).
Procedimiento de Prueba	Prueba 1 – listarOrdenesElegiblesParaProcedimiento Paso 1: Enviar query.usuarioid = "abc" (inválido). Resultado esperado: 400 Bad Request con “usuarioid inválido”. Paso 2: Enviar usuarioid = 10 y mockear servicio para devolver lista válida. Resultado esperado: 200 OK con arreglo de órdenes elegibles.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 3: Forzar excepción en el servicio.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p> <p>Prueba 2 – listarMisOrdenes</p> <p>Paso 1: Simular req.usuario.id = 7.</p> <p>Paso 2: Mockear servicio para devolver órdenes del usuario 7.</p> <p>Resultado esperado (éxito): 200 OK con lista.</p> <p>Paso 3: Forzar error del servicio.</p> <p>Resultado esperado: 500 con mensaje genérico.</p> <p>Prueba 3 – listarOrdenes</p> <p>Paso 1: Invocar el método sin filtros (o con filtros válidos).</p> <p>Paso 2: Mockear servicio para devolver lista global.</p> <p>Resultado esperado: 200 OK con listado de todas las órdenes.</p> <p>(Si el servicio falla → 500.)</p> <p>Prueba 4 – buscarOrdenes</p> <p>Paso 1: Enviar params.id = 25 y mockear servicio para devolver una orden.</p> <p>Resultado esperado (existe): 200 OK con la orden.</p> <p>Paso 2: Mockear servicio para devolver null.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado (no existe): 404 Not Found con “Orden no encontrada”. (Error interno → 500.)</p> <p>Prueba 5 – crearOrdenes</p> <p>Paso 1: Preparar req.body válido (p. ej. { usuarioid, procedimientoid, monto }).</p> <p>Paso 2: Mockear servicio crear devolviendo la orden creada.</p> <p>Resultado esperado: 201 Created con objeto creado (id, datos).</p> <p>(Validaciones fallidas → 400 si tu política lo contempla; error de servicio → 500.)</p> <p>Prueba 6 – actualizarOrdenes</p> <p>Paso 1: params.id = "xyz" (no numérico) + cualquier body.</p> <p>Resultado esperado: 400 Bad Request “ID inválido”.</p> <p>Paso 2: params.id = 50, body válido; mockear servicio para indicar no encontrado.</p> <p>Resultado esperado: 404 Not Found.</p> <p>Paso 3: params.id = 50, body válido; mockear actualización exitosa.</p> <p>Resultado esperado: 200 OK con confirmación/orden actualizada.</p> <p>Paso 4: Forzar error del servicio.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: 500 Internal Server Error.</p> <p>Prueba 7 – eliminarOrdenes</p> <p>Paso 1: params.id = 33, mockear servicio para eliminación exitosa (1 registro).</p> <p>Resultado esperado: 200 OK con mensaje “Orden eliminada”.</p> <p>Paso 2: (Opcional) Si la orden no existe, el servicio devuelve 0.</p> <p>Resultado esperado: 404 Not Found (si tu controlador lo contempla).</p> <p>Paso 3: Forzar error del servicio.</p> <p>Resultado esperado: 500 Internal Server Error.</p>
Datos de salida - Resultado Esperado	<p>1 200/201 en listados/creación/actualización/eliminación exitosos.</p> <p>2 400 por parámetros inválidos (IDs, usuariold).</p> <p>3 404 en recursos no encontrados (buscar/actualizar/eliminar).</p> <p>4 500 ante fallas del servicio.</p> <p>5 Respuestas JSON claras y sin excepciones no controladas.</p>
Resultado Obtenido Prueba Exitosa	<p>El controlador de Órdenes valida entradas, delega correctamente y responde con los códigos HTTP esperados en todos los escenarios definidos.</p>

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 227 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

26. OrdenControllers.test

Validar el comportamiento del controlador de órdenes en sus diferentes métodos: listar, crear, actualizar y eliminar, comprobando que retorne los códigos de estado HTTP esperados (200, 201, 400, 404 y 500) ante diferentes condiciones.

Prueba 1: listarOrdenesElegiblesParaProcedimiento

Verifica que el controlador responda correctamente cuando:

- Se pasa un usuarioid inválido (debe devolver **400**).
- El servicio retorna datos válidos (debe devolver **200**).
- El servicio lanza un error (debe devolver **500**).

```
test('listarOrdenesElegiblesParaProcedimiento -> 400 inválido, 200 ok y 500 error', async () => {
  let ctx = mockReqRes({ params: { usuarioid: 'abc' } });
  await controller.listarOrdenesElegiblesParaProcedimiento(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);

  svc.listarOrdenesEvaluacionRealizadaPorUsuario.mockResolvedValue([ { id: 1 } ]);
  ctx = mockReqRes({ params: { usuarioid: '7' } });
  await controller.listarOrdenesElegiblesParaProcedimiento(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.listarOrdenesEvaluacionRealizadaPorUsuario.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { usuarioid: '7' } });
  await controller.listarOrdenesElegiblesParaProcedimiento(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```

Prueba 2: listarMisOrdenes

Valida que el método devuelva **200** al listar órdenes del usuario autenticado y **500** si ocurre un error interno.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

```
test('listarMisOrdenes -> 200 y 500', async () => [
  svc.listarOrdenesPorUsuario.mockResolvedValue([ { id: 1 } ]);
  let ctx = mockReqRes({ usuario: { id: 9 } });
  await controller.listarMisOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.listarOrdenesPorUsuario.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ usuario: { id: 9 } });
  await controller.listarMisOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 3: listarOrdenes

Confirma que el método responda correctamente con **200** al listar todas las órdenes.

```
test('listarOrdenes -> 200', async () => {
  svc.listarLasOrdenes.mockResolvedValue([{ id: 1 }]);
  const { req, res } = mockReqRes();
  await controller.listarOrdenes(req, res);
  expect(res.statusCode).toBe(200);
});
```

Prueba 4:buscarOrdenes

Valida que se retorne **200** si la orden existe y **404** si no se encuentra.

```
test('buscarOrdenes -> 200 y 404', async () => {
  svc.buscarLasOrdenes.mockResolvedValue({ id: 1 });
  let ctx = mockReqRes({ params: { id: 3 } });
  await controller.buscarOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLasOrdenes.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 3 } });
  await controller.buscarOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
});
```

Prueba 5:crearOrdenes

Comprueba que el controlador cree correctamente una orden y devuelva **201**.

```
test('crearOrdenes -> 201', async () => {
  svc.crearLasOrdenes.mockResolvedValue({ id: 1 });
  const { req, res } = mockReqRes({ body: { a: 1 } });
  await controller.crearOrdenes(req, res);
  expect(res.statusCode).toBe(201);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 6: actualizarOrdenes

Evaluá los distintos escenarios:

- ID inválido → **400**
- No encontrado → **404**
- Actualización exitosa → **200**
- Error interno → **500**

```
test('actualizarOrdenes -> 400 inválido, 404, 200 y 500', async () => {
  let ctx = mockReqRes({ params: { id: 'abc' }, body: {} });
  await controller.actualizarOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);

  svc.actualizarLasOrdenes.mockResolvedValue([0]);
  ctx = mockReqRes({ params: { id: '7' }, body: {} });
  await controller.actualizarOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);

  svc.actualizarLasOrdenes.mockResolvedValue([1]);
  ctx = mockReqRes({ params: { id: '7' }, body: {} });
  await controller.actualizarOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.actualizarLasOrdenes.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: '7' }, body: {} });
  await controller.actualizarOrdenes(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
})
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

prueba 7: eliminarOrdenes

Verifica que el controlador elimine correctamente una orden y responda con **200** y un mensaje de confirmación.

```
test('eliminarOrdenes -> 200', async () => {
  const { req, res } = mockReqRes({ params: { id: 1 } });
  await controller.eliminarOrdenes(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ message: 'Orden eliminada' });
  expect(svc.eliminarLasOrdenes).toHaveBeenCalledWith(1);
});
```

Resultado:

```
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- tests/_OrdenControllers.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests/_OrdenControllers.test.js
PASS  _tests/_OrdenControllers.test.js
  Controlador de Órdenes
    ✓ listarOrdenesElegiblesParaProcedimiento -> 400 invalido, 200 ok y 500 error (5 ms)
    ✓ listarMisOrdenes -> 200 y 500 (1 ms)
    ✓ listarOrdenes -> 200 (1 ms)
    ✓ buscarOrdenes -> 200 y 404 (2 ms)
    ✓ crearOrdenes -> 201 (1 ms)
    ✓ actualizarOrdenes -> 400 invalido, 404, 200 y 500 (2 ms)
    ✓ eliminarOrdenes -> 200 (2 ms)

Test Suites: 1 passed, 1 total
Tests:    7 passed, 7 total
Snapshots: 0 total
Time:    1.042 s
Ran all test suites matching _tests/_OrdenControllers.test.js/i.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ █
```

Fecha de realización:

Duración de la prueba: 5 min



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

15-09-2025	
Requerimiento Funcional de la prueba	El Controlador de Órdenes de Procedimiento debe gestionar correctamente los flujos de listar, buscar, crear, actualizar y eliminar, devolviendo estados HTTP coherentes (200, 201, 400, 404, 500) según entradas, existencia del recurso y fallas del servicio.
Objetivo	Verificar que cada método: Valide parámetros (IDs numéricos, body requerido). Delegue en el servicio y maneje errores sin excepciones no controladas. Responda con el código HTTP y payload adecuados.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: listarOrdenesProcedimientos, buscarOrdenesProcedimientos, crearOrdenesProcedimientos, actualizarOrdenesProcedimientos, eliminaOrdenesProcedimientos. Mocks: servicio (listar/buscar/crear/actualizar/eliminar), req.usuario si aplica. Parámetros: id (orden-procedimiento), body con datos mínimos (p. ej. procedimientoId, ordenId, estado, monto).
Procedimiento de Prueba	Prueba 1 – Listar órdenes de procedimiento: listarOrdenesProcedimientos() Paso 1: Invocar sin filtros (o con filtros válidos). Paso 2: Mockear servicio para devolver una lista de órdenes de procedimiento. Paso 3: Ejecutar el controlador.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: 200 OK con la lista devuelta por el servicio.</p> <p>(Si deseas cubrir error del servicio, espera 500 con mensaje genérico.)</p> <p>Prueba 2 – Buscar orden de procedimiento: buscarOrdenesProcedimientos()</p> <p>Paso 1: Enviar params.id válido y mockear servicio para devolver un objeto existente.</p> <p>Resultado esperado (existe): 200 OK con la orden.</p> <p>Paso 2: Mockear servicio para devolver null.</p> <p>Resultado esperado (no existe): 404 Not Found con “Orden de procedimiento no encontrada”.</p> <p>(Error interno → 500.)</p> <p>Prueba 3 – Crear orden de procedimiento: crearOrdenesProcedimientos()</p> <p>Paso 1: Preparar req.body válido (p. ej. { orderId, procedimientoId, estado, monto }).</p> <p>Paso 2: Mockear servicio crear devolviendo el objeto creado.</p> <p>Paso 3: Ejecutar el controlador.</p> <p>Resultado esperado: 201 Created con el objeto creado.</p> <p>(Si faltan campos requeridos → 400; error del servicio → 500.)</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 4 – Actualizar orden de procedimiento: actualizarOrdenesProcedimientos()</p> <p>Paso 1: params.id = "abc" (no numérico) + cualquier body.</p> <p>Resultado esperado: 400 Bad Request “ID inválido”.</p> <p>Paso 2: params.id = 42, body válido; mockear servicio para indicar no encontrado.</p> <p>Resultado esperado: 404 Not Found.</p> <p>Paso 3: params.id = 42, body válido; mockear actualización exitosa.</p> <p>Resultado esperado: 200 OK con confirmación/objeto actualizado.</p> <p>Paso 4: Forzar error del servicio.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p> <p>Prueba 5 – Eliminar orden de procedimiento: eliminaOrdenesProcedimientos()</p> <p>Paso 1: params.id = 55; mockear servicio para eliminación exitosa (retorna 1).</p> <p>Paso 2: Ejecutar el controlador.</p> <p>Resultado esperado: 200 OK con mensaje “Orden de procedimiento eliminada”.</p> <p>(Opcional) Si el servicio retorna 0 (no existe), espera 404. En errores internos → 500.)</p>
Datos de salida - Resultado Esperado	1 200/201 en listados/creación/actualización/eliminación exitosos.

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 235 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación:	Elaborado por:	
Número del Documento:		1/02/2025		

	<p>2 400 ante IDs inválidos o body faltante.</p> <p>3 404 cuando el recurso no existe (buscar/actualizar/eliminar).</p> <p>4 500 en fallas del servicio.</p> <p>5 Respuestas JSON claras y sin excepciones no controladas.</p>		
Resultado Obtenido	Prueba Exitosa	El Controlador de Órdenes de Procedimiento valida entradas, delega correctamente y responde con los códigos HTTP esperados en todos los escenarios definidos.	Si(x)



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

27 OrdenProcedimientosControllers

Validar el correcto funcionamiento del **Controlador de Órdenes de Procedimiento**, asegurando que gestione de forma adecuada los flujos de **listar, buscar, crear, actualizar y eliminar**, manejando los códigos de estado HTTP correspondientes (**200, 201, 400, 404 y 500**) ante diferentes escenarios.

Prueba 1:listarOrdenesProcedimientos

Valida que el controlador retorne un código **200** cuando el servicio responde correctamente con una lista de órdenes de procedimiento.

```
test('listar -> 200', async () => [
  svc.listarLasOrdenesProcedimientos.mockResolvedValue([{ id: 1 }]);
  const { req, res } = mockReqRes();
  await controller.listarOrdenesProcedimientos(req, res);
  expect(res.statusCode).toBe(200);
]);
```

Prueba 2:buscarOrdenesProcedimientos

Verifica dos escenarios:

- Cuando la orden existe → debe retornar **200**.
- Cuando no se encuentra → debe retornar **404**.

```
test('buscar -> 200 y 404', async () => [
  svc.buscarLasOrdenesProcedimientos.mockResolvedValue({ id: 1 });
  let ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarOrdenesProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLasOrdenesProcedimientos.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarOrdenesProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3:crearOrdenesProcedimientos

Comprueba que el controlador cree una nueva orden de procedimiento correctamente y retorne código **201** con el objeto creado.

```
test('crear -> 201', async () => {
  svc.crearLasOrdenesProcedimientos.mockResolvedValue({ id: 1 });
  const { req, res } = mockReqRes({ body: { a: 1 } });
  await controller.crearOrdenesProcedimientos(req, res);
  expect(res.statusCode).toBe(201);
});
```

Prueba 4:actualizarOrdenesProcedimientos

Evalúa la actualización de órdenes de procedimiento en diferentes condiciones:

1. **ID inválido** → debe devolver **400**.
2. **Orden no encontrada** → debe devolver **404**.
3. **Actualización exitosa** → debe devolver **200**.
4. **Error interno** → debe devolver **500**.

```
test('actualizar -> 400 inválido, 404, 200 y 500', async () => {
  let ctx = mockReqRes({ params: { id: 'abc' }, body: {} });
  await controller.actualizarOrdenesProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);

  svc.actualizarLasOrdenesProcedimientos.mockResolvedValue([0]);
  ctx = mockReqRes({ params: { id: '7' }, body: { id_orden: 1, id_procedimiento: 2 } });
  await controller.actualizarOrdenesProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);

  svc.actualizarLasOrdenesProcedimientos.mockResolvedValue([1]);
  ctx = mockReqRes({ params: { id: '7' }, body: { id_orden: 1, id_procedimiento: 2 } });
  await controller.actualizarOrdenesProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.actualizarLasOrdenesProcedimientos.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: '7' }, body: { id_orden: 1, id_procedimiento: 2 } });
  await controller.actualizarOrdenesProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 5:eliminarOrdenesProcedimientos

Verifica que el controlador elimine correctamente una orden de procedimiento, devolviendo **200** y un mensaje confirmando la eliminación.

```
test('eliminar -> 200', async () => {
  const { req, res } = mockReqRes({ params: { id: 1 } });
  await controller.eliminarOrdenesProcedimientos(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ message: 'Orden Procedimiento eliminada' });
  expect(svc.eliminarLasOrdenesProcedimientos).toHaveBeenCalledWith(1);
});
```

Resultado

```
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$ npm test -- __tests__/_OrdenProcedimientosControllers.test.js
> clinesteticagi.0.0 test
> jest --passWithNoTests __tests__/_OrdenProcedimientosControllers.test.js

PASS  __tests__/_OrdenProcedimientosControllers.test.js
  Controlador de Ordenes de Procedimiento
    - listar -> 200 (5 ms)
    - buscar -> 200 y 404 (1 ms)
    - crear -> 201 (1 ms)
    - actualizar -> 400 inválido, 404, 200 y 500 (1 ms)
    - eliminar -> 200 (3 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.754 s
 Ran all test suites matching __tests__/_OrdenProcedimientosControllers.test.js/i.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Requerimiento Funcional de la prueba	El Servicio de Órdenes de Procedimiento debe ejecutar correctamente las operaciones CRUD sobre el modelo ordenprocedimiento, devolviendo los valores esperados y propagando de forma controlada los errores del ORM (Sequelize).
Objetivo	Validar que cada función del servicio delegue adecuadamente en el modelo (findAll, findByPk, create, update, destroy) y retorne resultados coherentes (lista, objeto, contadores), incluyendo manejo correcto de "no encontrado" y errores internos.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	listarLasOrdenesProcedimientos, buscarLasOrdenesProcedimientos(id), crearLasOrdenesProcedimientos(data), eliminarLasOrdenesProcedimientos(id), actualizarLasOrdenesProcedimientos(id, cambios). Mocks: ordenprocedimiento.findAll/findByPk/create/update/destroy Datos simulados: id, ordenId, procedimientoId, estado, monto.
Procedimiento de Prueba	Prueba 1 – listarLasOrdenesProcedimientos (findAll) Paso 1: Mockear findAll para devolver un arreglo de órdenes de procedimiento. Paso 2: Ejecutar listarLasOrdenesProcedimientos(). Resultado esperado: Retorna la lista recibida del modelo y findAll es llamado una vez con los parámetros esperados (filtros/orden si aplica).



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	(Error interno: si findAll lanza, el servicio propaga el error.)
	<p>Prueba 2 – buscarLasOrdenesProcedimientos (findById)</p> <p>Paso 1: Mockear findById(id) para devolver un registro válido.</p> <p>Paso 2: Ejecutar buscarLasOrdenesProcedimientos(id).</p> <p>Resultado esperado (existe): Retorna el objeto de la orden de procedimiento.</p> <p>Resultado esperado (no existe): Si findById devuelve null, el servicio retorna null (o lanza “No encontrado” según tu diseño).</p> <p>(Error interno: propaga la excepción del ORM.)</p>
	<p>Prueba 3 – crearLasOrdenesProcedimientos (create)</p> <p>Paso 1: Preparar data válido (p. ej. { orderId, procedimientoId, estado, monto }).</p> <p>Paso 2: Mockear create(data) para devolver el objeto creado.</p> <p>Paso 3: Ejecutar crearLasOrdenesProcedimientos(data).</p> <p>Resultado esperado: Retorna el objeto creado y create fue invocado con el payload exacto.</p> <p>(Validación fallida/ORM error: el servicio propaga el error.)</p>
	<p>Prueba 4 – eliminarLasOrdenesProcedimientos (destroy)</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Mockear destroy({ where: { id } }) para devolver 1 (una fila eliminada).</p> <p>Paso 2: Ejecutar eliminarLasOrdenesProcedimientos(id).</p> <p>Resultado esperado (éxito): Retorna un indicador de eliminación (true/contador=1).</p> <p>Resultado esperado (no existe): Si destroy devuelve 0, retornar false o lanzar “No encontrado” (según diseño).</p> <p>(Error interno: propaga la excepción del ORM.)</p> <p>Prueba 5 – actualizarLasOrdenesProcedimientos (update)</p> <p>Paso 1: Preparar cambios válidos (por ejemplo { estado: "pagada" }).</p> <p>Paso 2: Mockear update(cambios, { where: { id } }) para devolver [1] (una fila afectada) o usar findByPk + instance.update.</p> <p>Paso 3: Ejecutar actualizarLasOrdenesProcedimientos(id, cambios).</p> <p>Resultado esperado (éxito): Retorna confirmación (contador actualizado/objeto actualizado).</p> <p>Resultado esperado (sin cambios / no encontrado): Si el contador es 0 o findByPk es null, retornar indicador de “no actualizado/no encontrado” o lanzar error de negocio.</p> <p>(Error interno: propaga excepción del ORM.)</p>
Datos de salida - Resultado Esperado	1 listar* devuelve el arreglo de registros. 2 buscar* retorna el objeto o null (según implementación). 3 crear* retorna el objeto creado.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	4 actualizar* confirma filas afectadas o retorna “no encontrado/sin cambios”. 5 eliminar* confirma eliminación (1) o “no encontrado” (0). 6 Errores del ORM se propagan de forma controlada; no hay excepciones no capturadas.		
Resultado Obtenido Prueba Exitosa	El Servicio de Órdenes de Procedimiento interactúa correctamente con el modelo ordenprocedimiento, ejecutando los métodos CRUD y devolviendo resultados coherentes, con manejo de errores acorde.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

28. OrdenProcedimientoServices

Validar el comportamiento de los **Servicios de Órdenes de Procedimiento**, comprobando que los métodos CRUD (listar, buscar, crear, actualizar y eliminar) interactúan correctamente con el modelo ordenprocedimiento y devuelven los valores esperados.

Prueba 1: listarLasOrdenesProcedimientos

Verifica que el servicio consulte correctamente todos los registros de órdenes de procedimiento mediante el método findAll.

```
test('listarLasOrdenesProcedimientos', async () => {
  models.ordenprocedimiento.findAll.mockResolvedValue([{ id: 1 }]);
  const res = await svc.listarLasOrdenesProcedimientos();
  expect(res).toEqual([{ id: 1 }]);
});
```

Prueba 2:buscarLasOrdenesProcedimientos

Evaluá la función encargada de buscar una orden de procedimiento específica por su ID utilizando findByPk.

```
test('buscarLasOrdenesProcedimientos', async () => {
  models.ordenprocedimiento.findByPk.mockResolvedValue({ id: 2 });
  const res = await svc.buscarLasOrdenesProcedimientos(2);
  expect(res).toEqual({ id: 2 });
});
```

Prueba 3:crearLasOrdenesProcedimientos

Comprueba que el servicio cree correctamente un nuevo registro de orden de procedimiento utilizando create.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('crearLasOrdenesProcedimientos', async () => {
  models.orderprocedimiento.create.mockResolvedValue({ id: 3 });
  const res = await svc.crearLasOrdenesProcedimientos({ a: 1 });
  expect(models.orderprocedimiento.create).toHaveBeenCalledWith({ a: 1 });
  expect(res).toEqual({ id: 3 });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 4:eliminarLasOrdenesProcedimientos

Verifica que el servicio elimine correctamente una orden de procedimiento específica, utilizando el método destroy.

```
test('eliminarLasOrdenesProcedimientos', async () => {
  models.ordenprocedimiento.destroy.mockResolvedValue(1);
  const res = await svc.eliminarLasOrdenesProcedimientos(4);
  expect(models.ordenprocedimiento.destroy).toHaveBeenCalledWith({ where: { id: 4 } });
  expect(res).toBe(1);
});
```

Prueba 5:actualizarLasOrdenesProcedimientos

Valida la actualización de un registro de orden de procedimiento existente mediante update.

```
test('actualizarLasOrdenesProcedimientos', async () => {
  models.ordenprocedimiento.update.mockResolvedValue([1]);
  const res = await svc.actualizarLasOrdenesProcedimientos(5, { x: 2 });
  expect(models.ordenprocedimiento.update).toHaveBeenCalledWith({ x: 2 }, { where: { id: 5 } });
  expect(res).toEqual([1]);
});
```

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests/_OrdenProcedimientoServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests/_OrdenProcedimientoServices.test.js

[PASS]  tests/_OrdenProcedimientoServices.test.js
Servicios de Órdenes de Procedimiento
  listarLasOrdenesProcedimientos (37 ms)
  buscarLasOrdenesProcedimientos (2 ms)
  crearLasOrdenesProcedimientos (2 ms)
  eliminarLasOrdenesProcedimientos (1 ms)
  actualizarLasOrdenesProcedimientos (1 ms)

Test Suites: 1 passed, 1 total
Tests:      5 passed, 5 total
Snapshots:  0 total
Time:       0.7 s
Run all test suites matching _tests/_OrdenProcedimientoServices.test.js/1.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El Servicio de Órdenes debe ejecutar correctamente sus operaciones: listar, buscar, crear (con/sin procedimientos), actualizar, eliminar, y listados especializados por usuario y por evaluaciones realizadas, manejando errores (incluido logging) y propagación adecuada de excepciones del ORM.
Objetivo	Validar que las funciones del servicio deleguen correctamente en el modelo Sequelize (Orden, y asociaciones) usando findAll, findByPk, create, destroy, update, y que apliquen la lógica de negocio adicional (agregar procedimientos, filtrados por usuario, deduplicación de IDs, manejo/propagación de errores).
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones bajo prueba: listarLasOrdenes, buscarLasOrdenes, crearLasOrdenes, eliminarLasOrdenes, actualizarLasOrdenes, listarOrdenesPorUsuario, listarOrdenesEvaluacionRealizadaPorUsuario. Mocks: Orden.findAll/findByPk/create/destroy/update, modelos/relaciones (addProcedimientos en instancia creada, mocks de Cita si aplica), util de logs. Datos: id, usuarioid, procedimientos (array opcional), payload de orden.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Procedimiento de Prueba	Prueba 1 – listarLasOrdenes
	Paso 1: Mockear Orden.findAll para devolver arreglo de órdenes.
	Paso 2: Ejecutar listarLasOrdenes().
	Resultado esperado: Retorna la lista devuelta por el modelo; findAll llamado una vez con filtros/include esperados (si aplica).
	Error (opcional): si findAll lanza, el servicio propaga el error.
	Prueba 2 – buscarLasOrdenes
	Paso 1: Mockear Orden.findById devolviendo una orden válida.
	Paso 2: Ejecutar buscarLasOrdenes(id).
	Resultado esperado (existe): Retorna el objeto de la orden.
	Resultado esperado (no existe): Retorna null (o lanza “No encontrada” según diseño).
	Error: Propaga excepción del ORM.
	Prueba 3 – crearLasOrdenes (con procedimientos)
	Paso 1: Preparar data con procedimientos: [{ id: 1 }, { id: 2 }].
	Paso 2: Mockear Orden.create para retornar una instancia con método addProcedimientos = jest.fn().
	Paso 3: Ejecutar crearLasOrdenes(data).
	Resultado esperado:



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Se invoca create una vez con el payload base de la orden (sin mutar indebidamente los procedimientos).</p> <p>Se invoca addProcedimientos una vez con [1,2] (IDs mapeados).</p> <p>Retorna la orden creada (puede incluir relación si la función la agrega).</p> <p>Error: Si create o addProcedimientos fallan, el servicio propaga el error.</p>
	<p>Prueba 4 – crearLasOrdenes (sin procedimientos)</p> <p>Paso 1: data sin campo procedimientos o con [].</p> <p>Paso 2: Mockear Orden.create y NO definir addProcedimientos (o verificar que no se llame).</p> <p>Paso 3: Ejecutar la función.</p> <p>Resultado esperado:</p> <p>create llamado una vez.</p> <p>No se llama addProcedimientos.</p> <p>Retorna la orden creada.</p>



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario**Prueba 5 – eliminarLasOrdenes**

Paso 1: Mockear Orden.destroy({ where: { id } }) → 1.

Paso 2: Ejecutar eliminarLasOrdenes(id).

Resultado esperado (éxito): Retorna true (o contador=1).

No encontrado: Si devuelve 0, retorna false o lanza “No encontrada” (según diseño).

Error: Propaga la excepción del ORM.

Prueba 6 – actualizarLasOrdenes

Paso 1: Preparar cambios válidos (p. ej. { estado: "pagada" }).

Paso 2: Mockear Orden.update(cambios, { where: { id } }) → [1].

Paso 3: Ejecutar actualizarLasOrdenes(id, cambios).

Resultado esperado (éxito): Indica actualización (true/contador=1/objeto actualizado).

Sin cambios / no encontrada: [0] → retorna indicador de sin cambios/no encontrada o lanza error de negocio.

Error: Propaga excepción del ORM.

Prueba 7 – listarOrdenesPorUsuario (error controlado y log)



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Mockear una referencia interna mal definida (p. ej., acceso a procedimientos inexistente) para forzar un error propio de la función.</p> <p>Paso 2: Espiar el util de log (logger.error o console.error).</p> <p>Paso 3: Ejecutar listarOrdenesPorUsuario(usuarioId).</p> <p>Resultado esperado:</p> <p>El error se atrapa internamente y se registra (logger llamado).</p> <p>La función retorna [] o un valor seguro sin lanzar excepción (según especificación dada).</p> <p>Prueba 8 – listarOrdenesEvaluacionRealizadaPorUsuario (flujo base)</p> <p>Paso 1: Simular citas del usuario con evaluaciones realizadas y órdenes asociadas.</p> <p>Paso 2: Mockear consultas necesarias (Cita.findAll/Orden.findAll o lo que aplique).</p> <p>Resultado esperado: Retorna lista de órdenes correspondientes a las evaluaciones realizadas.</p> <p>Prueba 9 – listarOrdenesEvaluacionRealizadaPorUsuario (deduplicación de IDs)</p> <p>Paso 1: Simular múltiples citas que referencian las mismas órdenes válidas.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Paso 2: Ejecutar la función. Resultado esperado: Se filtran IDs únicos y se consultan/retornan órdenes sin duplicados. Prueba 10 – listarOrdenesEvaluacionRealizadaPorUsuario (propagación de error) Paso 1: Mockear Orden.findAll para rechazar la promesa. Paso 2: Ejecutar la función. Resultado esperado: El error se propaga al llamador (no se traga), respetando el flujo de manejo de errores global.		
Datos de salida - Resultado Esperado	1 listar* retorna listas coherentes (y propaga errores del ORM cuando corresponde). 2 buscar* retorna objeto o null según existencia. 3 crear* crea la orden y solo llama addProcedimientos cuando hay procedimientos. 4 eliminar* confirma eliminación (1) o indica no encontrado (0). 5 actualizar* indica filas afectadas o no-encontrado. 6 listarOrdenesPorUsuario atrapa y loguea su propio error, sin reventar la ejecución. 7 listarOrdenesEvaluacionRealizadaPorUsuario deduplica IDs y propaga errores de consulta. 8 No hay excepciones no controladas fuera de los casos diseñados para propagar.		
Resultado Obtenido Prueba Exitosa	El Servicio de Órdenes ejecuta correctamente sus operaciones CRUD, gestiona procedimientos asociados, controla errores (loguea/propaga según el caso) y	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	realiza listados especializados con deduplicación adecuada.		
--	---	--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

29. OrdenServices

Validar las funcionalidades del **Servicio de Órdenes**, verificando el correcto comportamiento de sus métodos:

Prueba 1:listarLasOrdenes

Verifica que el servicio obtenga correctamente todas las órdenes almacenadas en la base de datos.

```
test("listarLasOrdenes devuelve lista", async () => [
  models.ordenes.findAll.mockResolvedValue([{ id: 1 }, { id: 2 }]);
  const res = await OrdenServices.listarLasOrdenes();
  expect(models.ordenes.findAll).toHaveBeenCalledWith();
  expect(res).toEqual([{ id: 1 }, { id: 2 }]);
]);
```

Prueba 2:buscarLasOrdenes

Comprueba que el servicio busque una orden por su **ID** utilizando findByPk.

```
test("buscarLasOrdenes encuentra por id", async () => [
  models.ordenes.findByPk.mockResolvedValue({ id: 10 });
  const res = await OrdenServices.buscarLasOrdenes(10);
  expect(models.ordenes.findByPk).toHaveBeenCalledWith(10);
  expect(res).toEqual({ id: 10 });
]);
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3:crearLasOrdenes

Valida la creación de una nueva orden junto con sus procedimientos relacionados.
Usa el método create y la función addProcedimientos simulada.

```
test("crearLasOrdenes crea orden con procedimientos", async () => [
  const addProcedimientos = jest.fn().mockResolvedValue(undefined);
  models.ordenes.create.mockResolvedValue({ id: 1, addProcedimientos });
  const data = { id_usuario: 5, procedimientos: [1, 2, 3] };
  const res = await OrdenServices.crearLasOrdenes(data);
  expect(models.ordenes.create).toHaveBeenCalledWith({ id_usuario: 5 });
  expect(addProcedimientos).toHaveBeenCalledWith([1, 2, 3]);
  expect(res).toEqual({ id: 1, addProcedimientos });
]);
```

Prueba 4:crearLasOrdenes

Verifica que, al crear una orden sin procedimientos, **no se invoque** addProcedimientos.

```
test("crearLasOrdenes sin procedimientos no llama a addProcedimientos", async () => [
  models.ordenes.create.mockResolvedValue({ id: 2 });
  const res = await OrdenServices.crearLasOrdenes({ id_usuario: 7 });
  expect(models.ordenes.create).toHaveBeenCalledWith({ id_usuario: 7 });
  expect(res).toEqual({ id: 2 });
]);
```

Prueba 5:eliminarLasOrdenes

Comprueba que el servicio elimine una orden existente usando destroy.

```
test("eliminarLasOrdenes elimina por id", async () => [
  models.ordenes.destroy.mockResolvedValue(1);
  const res = await OrdenServices.eliminarLasOrdenes(9);
  expect(models.ordenes.destroy).toHaveBeenCalledWith({ where: { id: 9 } });
  expect(res).toBe(1);
]);
```

Número de
Documento:**FS-DOC Formato Manual de
Usuario**Fecha de Creación:
1/02/2025Elaborado por:
Instructores área
Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 6: actualizarLasOrdenes

Evaluá la actualización de los datos de una orden mediante update.

```
test("actualizarLasOrdenes actualiza datos", async () => {
  models.ordenes.update.mockResolvedValue([1]);
  const res = await OrdenServices.actualizarLasOrdenes(3, {
    estado: "nueva",
  });
  expect(models.ordenes.update).toHaveBeenCalledWith(
    { estado: "nueva" },
    { where: { id: 3 } }
  );
  expect(res).toEqual([1]);
});
```

Prueba 7:listarOrdenesPorUsuario

Simula un error en la función (por referencia no definida de procedimientos) y verifica que el error sea atrapado y logueado, sin lanzar excepción.

```
test("listarOrdenesPorUsuario atrapa ReferenceError por uso de Procedimientos y no lanza", async () => {
  const res = await OrdenServices.listarOrdenesPorUsuario(4);
  expect(console.log).toHaveBeenCalled();
  expect.stringContaining("Error al listar órdenes por usuario:");
  expect.any(Error)
);
  expect(res).toBeUndefined();
});
```

Prueba 8:listarOrdenesEvaluacionRealizadaPorUsuario



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test("listarOrdenesEvaluacionRealizadaPorUsuario retorna [] cuando no hay ids", async () => {
    models.citas.findAll.mockResolvedValue([]);
    const res = await OrdenServices.listarOrdenesEvaluacionRealizadaPorUsuario(
        8
    );
    expect(models.citas.findAll).toHaveBeenCalledWith({
        where: { id_usuario: 8, tipo: "evaluacion", estado: "realizada" },
        attributes: ["id_orden"],
    });
    expect(res).toEqual([]);
    expect(models.ordenes.findAll).not.toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 9:listarOrdenesEvaluacionRealizadaPorUsuario

Verifica que, cuando hay varias citas con órdenes válidas, se filtren los IDs únicos y se obtengan las órdenes correspondientes.

```
test("listarOrdenesEvaluacionRealizadaPorUsuario busca órdenes por IDs únicos", async () => {
  models.citas.findAll.mockResolvedValue([
    { id_orden: 1 },
    { id_orden: 2 },
    { id_orden: 1 },
    { id_orden: null },
  ]);
  models.ordenes.findAll.mockResolvedValue([{ id: 1 }, { id: 2 }]);
  const res = await OrdenServices.listarOrdenesEvaluacionRealizadaPorUsuario(
    9
  );
  expect(models.ordenes.findAll).toHaveBeenCalledWith({
    where: { id_usuario: 9, id: [1, 2] },
    include: [
      {
        model: models.procedimientos,
        as: "procedimientos",
        through: { attributes: [] },
      },
    ],
  });
  expect(res).toEqual([{ id: 1 }, { id: 2 }]);
});
```

Prueba 10:listarOrdenesEvaluacionRealizadaPorUsuario

Simula un fallo en la consulta de órdenes (findAll rechaza la promesa) y verifica que el error se propague correctamente.

```
test("listarOrdenesEvaluacionRealizadaPorUsuario propaga error si falla", async () => {
  models.citas.findAll.mockResolvedValue([{ id_orden: 5 }]);
  models.ordenes.findAll.mockRejectedValue(new Error("DB down"));
  await expect(
    OrdenServices.listarOrdenesEvaluacionRealizadaPorUsuario(1)
  ).rejects.toThrow("DB down");
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/OrdenServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/OrdenServices.test.js

PASS  _tests_/OrdenServices.test.js
  Servicios de Órdenes
    listarLasOrdenes devuelve lista (45 ms)
    buscarLasOrdenes encuentra por id (2 ms)
    crearLasOrdenes crea orden con procedimientos (2 ms)
    crearLasOrdenes sin procedimientos no llama a addProcedimientos (2 ms)
    eliminarLasOrdenes elimina por id (2 ms)
    actualizarLasOrdenes actualiza datos (1 ms)
    listarOrdenesPorUsuario atrapa ReferenceError por uso de Procedimientos y no lanza (1 ms)
    listarOrdenesEvaluacionRealizadaPorUsuario retorna [] cuando no hay ids (2 ms)
    listarOrdenesEvaluacionRealizadaPorUsuario busca órdenes por ids únicos (2 ms)
    listarOrdenesEvaluacionRealizadaPorUsuario propaga error si falla (17 ms)

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        0.792 s
Ran all test suites matching _tests_/OrdenServices.test.js/i.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	El middleware PrimeraMayusculaCategoria debe normalizar y capitalizar el campo nombre en objetos de categoría antes de su procesamiento en el backend. Debe incluir una función auxiliar capitalizarPrimera() y comportarse de forma robusta ante entradas inválidas, sin romper el flujo de la aplicación.
Objetivo	Verificar que: capitalizarPrimera() maneje correctamente mayúsculas/minúsculas, espacios, entradas vacías/no-string y caracteres con acento; el middleware transforme req.body.nombre cuando es string;



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	ante valores no válidos o ausencia de body, no lance errores y siempre invoque next().
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	<p>Función auxiliar: capitalizarPrimera(texto) Middleware: PrimeraMayusculaCategoria (req, res, next)</p> <p>Casos: nombre con acentos, espacios, mayúsculas/minúsculas, vacío, null, undefined, numérico, objeto.</p>
Procedimiento de Prueba	<p>Prueba 1 – capitalizarPrimera: strings y casos borde</p> <p>Paso 1: Llamar capitalizarPrimera(" ácido hialurónico ").</p> <p>Paso 2: Llamar capitalizarPrimera("laser").</p> <p>Paso 3: Llamar capitalizarPrimera("LÁSER").</p> <p>Paso 4: Llamar capitalizarPrimera(""), capitalizarPrimera(null), capitalizarPrimera(123).</p> <p>Resultado esperado:</p> <p>Para " ácido hialurónico " → "Ácido hialurónico" (trim + primera letra en mayúscula, resto en minúscula conservando acentos).</p> <p>Para "laser" → "Laser".</p> <p>Para "LÁSER" → "Láser" (normaliza a "Primera mayúscula + resto minúsculas", preservando acentos).</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Para "", null, 123 → "" (cadena vacía) o el comportamiento definido como seguro (no lanzar excepción).

Prueba 2 – middleware: transforma req.body.nombre cuando es string

Paso 1: Simular req.body = { nombre: " rejuvenecimiento FACIAL " }.

Paso 2: Ejecutar el middleware con un next espiado (jest.fn()).

Resultado esperado:

req.body.nombre pasa a "Rejuvenecimiento facial" (trim + capitalización de la primera letra; resto minúsculas).

Se invoca next() exactamente una vez.

(Si la implementación aplica “title case” o solo “sentence case”, validar según la especificación elegida; aquí se asume “Sentence case”.)

Prueba 3 – middleware: no rompe si no hay body o no es string

Paso 1: Caso A → req.body = { nombre: 100 }.

Paso 2: Caso B → req.body = { } sin nombre.

Paso 3: Caso C → req.body = null / undefined.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Paso 4: Ejecutar el middleware en cada caso con next espiado. Resultado esperado: No lanza errores en ningún caso. next() es llamado siempre. Si nombre no es string o falta, el middleware no modifica el valor y continúa el flujo.		
Datos de salida - Resultado Esperado	capitalizarPrimera() devuelve una cadena segura y normalizada, con primera letra mayúscula y resto minúsculas, respetando acentos y eliminando espacios sobrantes. El middleware transforma correctamente req.body.nombre solo cuando es string.		
Resultado Obtenido Prueba Exitosa	El middleware PrimeraMayusculaCategoria y su auxiliar capitalizarPrimera() cumplen con la normalización/capitalización esperada, manejan casos borde y mantienen la robustez del flujo sin excepciones no controladas.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

30. PrimeraMayusculaCategoria

Validar el correcto funcionamiento del middleware PrimeraMayusculaCategoria, encargado de normalizar y capitalizar el campo nombre dentro de los objetos de categoría antes de ser procesados por el backend.

Prueba 1:capitalizarPrimera: strings y casos borde

Evalúa el comportamiento de la función auxiliar capitalizarPrimera(), asegurando que maneje correctamente:

- Mayúsculas/minúsculas.
- Espacios innecesarios.
- Cadenas vacías, null, o tipos no string.
- Caracteres con acento.

```
describe('Middleware PrimeraMayusculaCategoria', () => {
  test('capitalizarPrimera: strings y casos borde', () => {
    expect(Categoría.capitalizarPrimera(' hola MUNDO ')).toBe('Hola mundo');
    expect(Categoría.capitalizarPrimera('')).toBe('');
    expect(Categoría.capitalizarPrimera(' ')).toBe('');
    expect(Categoría.capitalizarPrimera(null)).toBe('');
    expect(Categoría.capitalizarPrimera(123)).toBe('');
    expect(Categoría.capitalizarPrimera('áE í')).toBe('Áé í');
  });
});
```

Prueba 2:middleware: transforma req.body.nombre cuando es string

Comprueba que el middleware modifique correctamente el valor del campo nombre en req.body cuando este es una cadena de texto, aplicando la capitalización esperada.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('middleware: transforma req.body.nombre cuando es string', () => {
  const req = { body: { nombre: ' MULTI Palabras' } };
  const res = {};
  const next = jest.fn();
  Categoria.middleware(req, res, next);
  expect(req.body.nombre).toBe('Multi palabras');
  expect(next).toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3:middleware: no rompe si no hay body o no es string

Verifica que el middleware sea robusto ante valores no válidos (por ejemplo, req.body.nombre numérico o ausente), asegurando que **no lance errores y siempre invoque next()**.

```
test('middleware: no rompe si no hay body o no es string', () => {
  const req = { body: { nombre: 77 } };
  const res = {};
  const next = jest.fn();
  Categoria.middleware(req, res, next);
  expect(req.body.nombre).toBe(77);
  expect(next).toHaveBeenCalled();
});
```

Resultado

```
dstevengz@dstevengz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$ npm test -- __tests__/_PrimeraMayusculaCategoria.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests __tests__/_PrimeraMayusculaCategoria.test.js
PASS  __tests__/_PrimeraMayusculaCategoria.test.js
  Middleware.PrimeraMayusculaCategoria
    - capitalizarPrimerasLetras y casos borde (5 ms)
      middleware: transforma req.body.nombre cuando es string (2 ms)
      middleware: no rompe si no hay body o no es string (2 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.672 s
Run all test suites matching __tests__/_PrimeraMayusculaCategoria.test.js/i.
dstevengz@dstevengz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
--	------------------------------



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Requerimiento Funcional de la prueba	El middleware PrimerMayusculaProcedimientos debe normalizar y capitalizar los campos de texto de los procedimientos (p. ej., nombre, descripcion, recomendaciones_previas) antes de su almacenamiento/procesamiento. Debe incluir una función auxiliar capitalizarPrimera() y comportarse de forma robusta ante entradas no válidas, sin romper el flujo de la aplicación.
Objetivo	Verificar que: capitalizarPrimera() limpia espacios, unifique mayúsculas/minúsculas y maneje entradas vacías/no-string; el middleware aplique la capitalización a nombre, descripcion y recomendaciones_previas cuando sean string; ignore valores no-string y siempre invoque next().
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Función auxiliar: capitalizarPrimera(texto) Middleware: PrimerMayusculaProcedimientos (req, res, next) Campos objetivo: nombre, descripcion, recomendaciones_previas Casos: strings con acentos/espacios, MAYÚSCULAS, vacíos, null, undefined, numéricos, objetos.
Procedimiento de Prueba	Prueba 1 – capitalizarPrimera: normaliza espacios y capitaliza Paso 1: Llamar capitalizarPrimera(" peeling QUÍMICO "). Paso 2: Llamar capitalizarPrimera("laser fraccional").



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Paso 3: Llamar capitalizarPrimera("LIMPIEZA FACIAL PROFUNDA").</p> <p>Paso 4: Llamar capitalizarPrimera(""), capitalizarPrimera(null), capitalizarPrimera(123).</p> <p>Resultado esperado:</p> <p>" peeling QUÍMICO " → "Peeling químico" (trim + una sola primera mayúscula, resto minúsculas, preservando acentos).</p> <p>"laser fraccional" → "Laser fraccional".</p> <p>"LIMPIEZA FACIAL PROFUNDA" → "Limpieza facial profunda".</p> <p>Para "", null, 123 → "" (cadena segura) sin lanzar excepción.</p> <p>Prueba 2 – middleware: transforma nombre, descripcion y recomendaciones_previas</p> <p>Paso 1: Simular req.body = { nombre: "microdermoABRASIÓN ", descripcion: "TRATAMIENTO PARA MANCHAS", recomendaciones_previas: " EVITAR SOL 48h " }.</p> <p>Paso 2: Ejecutar el middleware con next = jest.fn().</p> <p>Resultado esperado:</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>req.body.nombre → "Microdermoabrasión".</p> <p>req.body.descripcion → "Tratamiento para manchas".</p> <p>req.body.recomendaciones_previas → "Evitar sol 48h".</p> <p>next() es llamado exactamente una vez.</p> <p>Prueba 3 – middleware: ignora campos no string y sigue con next</p> <p>Paso 1: Caso A → req.body = { nombre: 100, descripcion: { t: "x" }, recomendaciones_previas: undefined }.</p> <p>Paso 2: Caso B → req.body = {} (sin campos).</p> <p>Paso 3: Caso C → req.body = null / undefined.</p> <p>Paso 4: Ejecutar el middleware con next = jest.fn().</p> <p>Resultado esperado:</p> <p>No se lanzan errores en ningún caso.</p> <p>Los campos no-string no se modifican.</p> <p>next() se invoca siempre para continuar el flujo.</p>
--	--

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 269 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Datos de salida - Resultado Esperado	1 capitalizarPrimera() devuelve cadenas limpias y capitalizadas (primera letra mayúscula, resto minúsculas), preservando acentos y removiendo espacios sobrantes. 2 El middleware transforma nombre, descripcion y recoendaciones_previas solo si son string. 3 Ante valores no-string o ausencia de body, no falla y siempre llama next().		
Resultado Obtenido Prueba Exitosa	El middleware PrimerMayusculaProcedimientos y su auxiliar capitalizarPrimera() aplican la normalización/capitalización esperada, manejan casos borde y mantienen la robustez del flujo sin excepciones no controladas.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

31. PrimerMayusculaProcedimientos

Validar el correcto funcionamiento del **middleware PrimerMayusculaProcedimientos**, encargado de normalizar y capitalizar los campos de texto relacionados con los **procedimientos médicos o estéticos**. Este middleware asegura la consistencia de los datos antes de su almacenamiento o procesamiento en el sistema.

Prueba 1: capitalizarPrimera: normaliza espacios y capitaliza

Evalúa la función auxiliar capitalizarPrimera(), responsable de limpiar espacios innecesarios, ajustar mayúsculas y minúsculas, y manejar correctamente entradas no válidas o vacías.

```
describe('Middleware PrimerMayusculaProcedimientos', () => {
  test('capitalizarPrimera: normaliza espacios y capitaliza', () => [
    expect(Proc.capitalizarPrimera(' LA DESCripcion ')).toBe('La descripcion');
    expect(Proc.capitalizarPrimera('')).toBe('');
    expect(Proc.capitalizarPrimera(' ')).toBe('');
    expect(Proc.capitalizarPrimera(undefined)).toBe('');
  ]);
});
```

Prueba 2: middleware: transforma nombre, descripción y recomendaciones_previas

Verifica que el middleware procese correctamente los campos principales del cuerpo de la solicitud (req.body), aplicando la función capitalizarPrimera() a los textos de tipo string.

```
test('middleware: transforma nombre, descripción y recomendaciones_previas', () => {
  const req = { body: { nombre: ' PROCeDimiento X ', descripción: ' Desc extensa ', recomendaciones_previas: ' AGUA ayuno ' }};
  const res = {};
  const next = jest.fn();
  Proc.middleware(req, res, next);
  expect(req.body.nombre).toBe('Procedimiento x');
  expect(req.body.descripción).toBe('Desc extensa');
  expect(req.body.recomendaciones_previas).toBe('Agua ayuno');
  expect(next).toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3:middleware: ignora campos no string y sigue con next

Asegura que el middleware sea **tolerante a tipos de datos no válidos**, evitando errores si alguno de los campos no es de tipo cadena, y que **siempre llame a next()** para continuar el flujo.

```
test('middleware: ignora campos no string y sigue con next', () => {
  const req = { body: { nombre: 10, descripcion: null, recomendaciones_previas: {} } };
  const res = {};
  const next = jest.fn();
  Proc.middleware(req, res, next);
  expect(req.body.nombre).toBe(10);
  expect(req.body.descripcion).toBeNull();
  expect(typeof req.body.recomendaciones_previas).toBe('object');
  expect(next).toHaveBeenCalled();
});
```

Resultado

```
dstevengz@dstevengz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- __tests__/_PrimerMayusculaProcedimientos.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests __tests__/_PrimerMayusculaProcedimientos.test.js
PASS  Tests /PrimerMayusculaProcedimientos.test.js
  Middleware PrimerMayusculaProcedimientos
    · capitalizarPrimer: normaliza espacios y capitaliza (# ms)
    · middleware: transforma nombre, descripción y recomendaciones_previas (2 ms)
    · middleware: ignora campos no string y sigue con next (2 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.671 s
Run all test suites matching __tests__/_PrimerMayusculaProcedimientos.test.js/i.
dstevengz@dstevengz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$
```

Fecha de realización:	Duración de la prueba: 5 min
15-09-2025	
Requerimiento Funcional de la prueba	El Controlador de Procedimientos debe gestionar correctamente los endpoints para listar, buscar, crear, actualizar y eliminar procedimientos, incluyendo manejo de imágenes (alta/baja) y validaciones de parámetros,



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	devolviendo códigos HTTP coherentes (200, 201, 400, 404, 500).
Objetivo	Verificar que el controlador: Valide parámetros (id, categoriald) y entrada de datos. Dele que en servicios y maneje errores internos. Gestione imágenes (creación masiva, eliminación por ID/URL con Cloudinary, altas nuevas). Devuelva respuestas con los códigos y payloads esperados.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos: listarProcedimientos, buscarProcedimientos, crearProcedimientos, actualizarProcedimientos, listarProcedimientosPorCategoria, eliminarProcedimientos. Mocks: servicios de procedimientos e imágenes (procedimientoService, procedimientoImagenes.bulkCreate/destroy), cloudinary.v2.uploader.destroy, req.files (imágenes), req.usuario si aplica.
Procedimiento de Prueba	Prueba 1 – listarProcedimientos: por categoría y general; manejo de errores Paso 1: Con query.categoriald = 5 (válido), mockear servicio para devolver lista filtrada. Resultado esperado: 200 OK con lista filtrada. Paso 2: Sin categoriald, mockear servicio para devolver lista completa. Resultado esperado: 200 OK con lista general.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 3: Forzar error del servicio.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p> <p>(Si categoriad es inválido —no numérico— puede considerarse 400 dependiendo de la política).</p> <p>Prueba 2 – buscarProcedimientos</p> <p>Paso 1: params.id = 12 (válido), mockear servicio para devolver un procedimiento.</p> <p>Resultado esperado: 200 OK con el procedimiento.</p> <p>Paso 2: Mockear servicio para devolver null.</p> <p>Resultado esperado: 404 Not Found “Procedimiento no encontrado”.</p> <p>(Error interno → 500).</p> <p>Prueba 3 – crearProcedimientos (crea + imágenes)</p> <p>Paso 1: req.body válido + req.files con varias imágenes.</p> <p>Paso 2: Mockear creación de procedimiento en servicio y procedimientoimagenes.bulkCreate para registrar imágenes.</p> <p>Resultado esperado: 201 Created con el objeto creado y referencias de imágenes; bulkCreate invocado una vez con los metadatos correctos (url/public_id/procedimentold).</p> <p>Prueba 4 – crearProcedimientos (error del servicio)</p>
--	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: req.body válido; mockear servicio para lanzar excepción durante la creación.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico; no se debe llamar bulkCreate si no existe procedimiento.</p> <p>Prueba 5 – actualizarProcedimientos (validación previa de ID y existencia)</p> <p>Paso 1: params.id = "abc" (no numérico).</p> <p>Resultado esperado: 400 Bad Request “ID inválido”.</p> <p>Paso 2: params.id = 20, mockear servicio de búsqueda para devolver null.</p> <p>Resultado esperado: 404 Not Found “Procedimiento no existe”.</p> <p>Prueba 6 – actualizarProcedimientos (actualización completa + manejo de imágenes)</p> <p>Paso 1: params.id = 20, existe el procedimiento; req.body con cambios válidos.</p> <p>Paso 2: Eliminar imágenes obsoletas:</p> <p>Mockear procedimiento imagenes.destroy con where: { id: { [Op.in]: idsEliminar } } y/o</p> <p>Mockear cloudinary.v2.uploader.destroy(public_id) para URLs a borrar.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 3: Agregar imágenes nuevas: req.files con nuevas imágenes → procedimientoimagenes.bulkCreate.</p> <p>Paso 4: Mockear servicio de actualización para retornar éxito.</p> <p>Resultado esperado: 200 OK confirmando actualización; se llamó destroy/uploader.destroy para bajas, y bulkCreate para altas nuevas.</p> <p>Prueba 7 – actualizarProcedimientos (error del servicio en actualización)</p> <p>Paso 1: params.id válido y existente; forzar excepción en el servicio de actualización.</p> <p>Resultado esperado: 500 Internal Server Error con mensaje genérico.</p> <p>Prueba 8 – listarProcedimientosPorCategoria (validación y flujos)</p> <p>Paso 1: params.categoriald = "x" (inválido).</p> <p>Resultado esperado: 400 Bad Request “categoriald inválido”.</p> <p>Paso 2: categoriald = 7 (válido), mockear servicio para devolver lista.</p> <p>Resultado esperado: 200 OK con lista.</p> <p>Paso 3: Forzar error del servicio.</p> <p>Resultado esperado: 500 Internal Server Error.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	Prueba 9 – eliminarProcedimientos Paso 1: params.id = 77 (válido), mockear servicio para eliminación exitosa (retorna 1). Resultado esperado: 200 OK con mensaje “Procedimiento eliminado”. Paso 2: (Opcional) si servicio devuelve 0 → 404 Not Found. Paso 3: Si el servicio falla → 500.		
Datos de salida - Resultado Esperado	1 Datos de salida / Resultado esperado genera 2 200/201 en listados/creación/actualización/eliminación exiosos. 3 400 por id/categoriald inválidos. 4 404 cuando el recurso no existe (buscar/actualizar/eliminar). 5 500 ante errores del servicio. 6 Manejo correcto de imágenes: bulkCreate en altas, destroy/uploader.destroy en bajas, y altas nuevas registradas. 7 Respuestas JSON claras y sin excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	El Controlador de Procedimientos valida parámetros, gestiona imágenes y delega correctamente en servicios, respondiendo con los códigos HTTP esperados en todos los escenarios.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

32. ProcedimientoControllers

Validar el correcto funcionamiento del **Controlador de Procedimientos**, garantizando que los endpoints de **listar, buscar, crear, actualizar y eliminar** procedimientos respondan adecuadamente, manejen imágenes y realicen las validaciones necesarias sobre los parámetros recibidos.

Prueba 1: listarProcedimientos: listar por categoría y general, con manejo de errores

Valida que el controlador liste procedimientos de dos maneras:

1. Filtrando por categoría (categoryId).
2. Mostrando todos los procedimientos si no hay filtro.
También verifica el manejo del error interno (500).

```
test('listarProcedimientos -> por categoria y general, 500 en error', async () => {
  svc.listarLosProcedimientosPorCategoria.mockResolvedValue([{ id: 1 }]);
  let ctx = mockReqRes({ query: { categoryId: '5' } });
  await controller.listarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.listarLosProcedimientos.mockResolvedValue([{ id: 2 }]);
  ctx = mockReqRes();
  await controller.listarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.listarLosProcedimientos.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ query: {} });
  await controller.listarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 2:buscarProcedimientos

Evaluá que el controlador devuelva correctamente el procedimiento solicitado o un error 404 cuando no existe.

```
test('buscarProcedimientos -> 200 y 404', async () => [
  svc.buscarLosProcedimientos.mockResolvedValue({ id: 1 });
  let ctx = mockReqRes({ params: { id: 3 } });
  await controller.buscarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLosProcedimientos.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 3 } });
  await controller.buscarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
]);
```

Prueba 3:crearProcedimientos

Verifica que el controlador cree un nuevo procedimiento, almacene las imágenes asociadas y devuelva **201**.

También comprueba que se invoque correctamente bulkCreate para guardar las imágenes múltiples.

```
test('crearProcedimientos -> 201 crea y guarda imagenes', async () => [
  svc.crearLosProcedimientos.mockResolvedValue({ id: 9, imagen: null });
  const files = { imagen: { path: '/img/p.png' }, imagenes: [{ path: '/img/a.png' }, { path: '/img/b.png' }] };
  const { req, res } = mockReqRes({ body: { precio: '12.5', duracion: '45', requiere_evaluacion: 'true', categoriaId: '3', nombre: 'nuevo_procedimiento' } });
  await controller.crearProcedimientos(req, res);
  expect(res.statusCode).toBe(201);
  expect(procedimientoImagenes.bulkCreate).toHaveBeenCalled();
]);
```

Prueba 4:crearProcedimientos

Evaluá el comportamiento del controlador cuando el servicio lanza una excepción durante la creación.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

```
test('crearProcedimientos -> 500 en error', async () => [
  svc.crearLosProcedimientos.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes({ body: { precio: '12', duracion: '30' } });
  await controller.crearProcedimientos(req, res);
  expect(res.statusCode).toBe(500);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 5:actualizarProcedimientos

Comprueba que el controlador valide correctamente el ID y verifique la existencia del procedimiento antes de intentar actualizarlo.

```
test('actualizarProcedimientos -> 400 id inválido, 404 no existe', async () => {
  let ctx = mockReqRes({ params: { id: 'abc' }, body: {} });
  await controller.actualizarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);

  svc.buscarLosProcedimientos.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: '7' }, body: {} });
  await controller.actualizarProcedimientos(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
});
```

Prueba 6:actualizarProcedimientos

Valida el proceso completo de actualización:

- Actualiza los datos del procedimiento.
- Elimina imágenes obsoletas por ID o URL (mediante procedimientoimagenes.destroy y cloudinary.v2.uploader.destroy).
- Agrega imágenes nuevas al procedimiento.

```
test('actualizarProcedimientos -> 200 con cambios, elimina imágenes por ids/url', async () => {
  svc.buscarLosProcedimientos.mockResolvedValue([{ id: 7, imagen: '/img/principal.png' }]);
  svc.actualizarLosProcedimientos.mockResolvedValue([1]);
  procedimientoimagenes.findAll.mockResolvedValue([{ id: 1, url: '/cloud/v123/prod/a.png' }]);
  const { req, res } = mockReqRes({ params: { id: '7' }, body: { precio: '10', duracion: '30', requiere_evaluacion: 'false' } });
  await controller.actualizarProcedimientos(req, res);
  expect(res.statusCode).toBe(200);
  expect(procedimientoimagenes.destroy).toHaveBeenCalled();
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 7: actualizarProcedimientos

Simula un error en el servicio de actualización y verifica que se devuelva un código **500**.

```
test('actualizarProcedimientos -> 500 en error servicio', async () => {
  svc.buscarLosProcedimientos.mockResolvedValue({ id: 7, imagen: null });
  svc.actualizarLosProcedimientos.mockRejectedValue(new Error('x'));
  const { req, res } = mockReqRes({ params: { id: '7' }, body: { precio: '10', duracion: '30' } });
  await controller.actualizarProcedimientos(req, res);
  expect(res.statusCode).toBe(500);
});
```

Prueba 8:listarProcedimientosPorCategoria

Verifica que el controlador valide correctamente el parámetro categoriaId y maneje los diferentes escenarios:

ID inválido, consulta exitosa y error interno del servicio.

```
test('listarProcedimientosPorCategoria -> 400 inválido y 200 ok, 500 error', async () => {
  let ctx = mockReqRes({ params: { categoriaId: 'abc' } });
  await controller.listarProcedimientosPorCategoria(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);

  svc.listarLosProcedimientosPorCategoria.mockResolvedValue([{ id: 1 }]);
  ctx = mockReqRes({ params: { categoriaId: '3' } });
  await controller.listarProcedimientosPorCategoria(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.listarLosProcedimientosPorCategoria.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { categoriaId: '3' } });
  await controller.listarProcedimientosPorCategoria(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 282 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Prueba 9:eliminarProcedimientos

Verifica que el controlador elimine correctamente un procedimiento, devolviendo **200** y confirmación del mensaje.

```
test('eliminarProcedimientos -> 200', async () => {
  const { req, res } = mockReqRes({ params: { id: 1 } });
  await controller.eliminarProcedimientos(req, res);
  expect(res.statusCode).toBe(200);
  expect(res.body).toEqual({ message: 'Procedimiento eliminado' });
  expect(svc.eliminarLosProcedimientos).toHaveBeenCalledWith(1);
});

});
```

Resultado

```
dstevengmz@dstevengmz-/Escritorio/Clinestatica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$ npm test -- _tests/_ProcedimientoControllers.test.js
> clinestatica@1.0.0 test
> jest --passWithNoTests _tests/_ProcedimientoControllers.test.js

PASS  _tests/_ProcedimientoControllers.test.js
  Controlador de Procedimientos
    listarProcedimientos -> por categoría y general, 500 en error (6 ms)
    buscarProcedimientos -> 200 y 404 (2 ms)
    crearProcedimientos -> 201 crea y guarda imágenes (3 ms)
    crearProcedimientos -> 500 en error (1 ms)
    actualizarProcedimientos -> 400 id invalido, 404 no existe (2 ms)
    actualizarProcedimientos -> 200 con cambios, elimina imágenes por id/url (2 ms)
    actualizarProcedimientos -> 500 en error servicio (1 ms)
    listarProcedimientosPorCategoria -> 400 invalido y 200 ok, 500 error (1 ms)
    eliminarProcedimientos -> 200 (2 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:  0 total
Time:        1.486 s
Ran all test suites matching _tests/_ProcedimientoControllers.test.js/...
dstevengmz@dstevengmz-/Escritorio/Clinestatica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backend$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
---	-------------------------------------



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Requerimiento Funcional de la prueba	Validar que el Servicio de Procedimientos delegue correctamente las operaciones CRUD al modelo Sequelize procedimientos (y relaciones cuando aplique), retornando valores coherentes y propagando errores del ORM de forma controlada.
Objetivo	Comprobar que los métodos del servicio: Llamen a findAll, findByPk, create, update, destroy con los parámetros correctos. Retornen la lista, el objeto encontrado/creado, o los indicadores de actualización/eliminación. Soporten filtros por categoría y manejen “no encontrado” y errores internos.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones bajo prueba: listarLosProcedimientos(), buscarLosProcedimientos(id), crearLosProcedimientos(data), listarLosProcedimientosPorCategoria(categoriald), eliminarLosProcedimientos(id), actualizarLosProcedimientos(id, cambios). Mocks:models.procedimientos.findAll/findByPk/create/update/destroy (y include con relaciones si aplica). Datos simulados: id, categoriald, nombre, descripcion, precio, etc. Ambiente: Node.js + Jest.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Procedimiento de Prueba	Prueba 1 – Listar todos: listarLosProcedimientos() Paso 1: Mockear models.procedimientos.findAll para devolver un arreglo de procedimientos. Paso 2: Ejecutar listarLosProcedimientos(). Resultado esperado: Se invoca findAll una vez (con include si el servicio lo define) y se retorna la misma lista del modelo. Prueba 2 – Buscar por ID (con includes): buscarLosProcedimientos(id) Paso 1: Mockear models.procedimientos.findByPk(id, { include: [...] }) para devolver un procedimiento con relaciones (p. ej. categoria, imagenes). Paso 2: Ejecutar con id válido. Resultado esperado: Llama findByPk con id y include correctos; retorna el procedimiento. Si null → retorna null (o lanza “No encontrado” según diseño). Prueba 3 – Crear: crearLosProcedimientos(data) Paso 1: Preparar data válido (p. ej. { nombre, descripcion, categoriald, precio }). Paso 2: Mockear models.procedimientos.create(data) para devolver el creado. Paso 3: Ejecutar la función. Resultado esperado: create se llama una vez con el payload exacto y se retorna el objeto creado. Si falla → se propaga el error. Prueba 4 – Listar por categoría: listarLosProcedimientosPorCategoria(categoriald)
-------------------------	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: categoriald = 5.</p> <p>Paso 2: Mockear findAll({ where: { categoriald: 5 }, include: [...] }).</p> <p>Paso 3: Ejecutar la función.</p> <p>Resultado esperado: findAll se invoca con el filtro where correcto (y include si aplica) y se retorna la lista filtrada. Si falla → se propaga el error.</p> <p>Prueba 5 – Eliminar: eliminarLosProcedimientos(id)</p> <p>Paso 1: Mockear destroy({ where: { id } }) para devolver 1 (eliminado) y luego 0 (no encontrado).</p> <p>Paso 2: Ejecutar con id válido.</p> <p>Resultado esperado:</p> <p>Con 1 → retorna 1 (o true).</p> <p>Con 0 → retorna 0 (o señaliza “no encontrado”, según diseño).</p> <p>Si falla → se propaga el error.</p> <p>Prueba 6 – Actualizar: actualizarLosProcedimientos(id, cambios)</p> <p>Paso 1: Preparar cambios (p. ej. { precio: 150, descripcion: "Actualizada" }).</p> <p>Paso 2: Mockear update(cambios, { where: { id } }) devolviendo [1] (éxito) y [0] (sin cambios/no existe).</p> <p>Paso 3: Ejecutar la función.</p> <p>Resultado esperado:</p> <p>Con [1] → indica actualización exitosa (true/contador/objeto, según implementación)</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	Con [0] → retorna indicador de no actualizado/no encontrado (o lanza error de negocio, según diseño). Si falla → se propaga el error.		
Datos de salida - Resultado Esperado	listar* devuelve la lista proporcionada por el modelo (con include cuando aplique). buscar* retorna el objeto o null si no existe (o lanza "No encontrado", según política). crear* retorna el objeto creado por create. actualizar* reporta filas afectadas o "no actualizado/no encontrado". eliminar* devuelve 1 (eliminado) o 0 (no encontrado). Los errores del ORM se propagan; no hay excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	El Servicio de Procedimientos delega correctamente en el modelo procedimientos para todas las operaciones, retorna valores coherentes y maneja adecuadamente filtros por categoría, "no encontrado" y errores internos.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

33. ProcedimientosServices

Validar el correcto funcionamiento de los **Servicios de Procedimientos**, asegurando que deleguen correctamente las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) hacia los métodos del modelo Sequelize procedimientos y que retornen los valores esperados.

Prueba 1:listarLosProcedimientos

Verifica que el servicio llame correctamente a models.procedimientos.findAll() y devuelva la lista de resultados obtenidos.

```
test('listarLosProcedimientos -> delega a findAll con includes', async () => {
  models.procedimientos.findAll.mockResolvedValue([{ id: 1 }]);
  const res = await svc.listarLosProcedimientos();
  expect(models.procedimientos.findAll).toHaveBeenCalled();
  expect(res).toEqual([{ id: 1 }]);
});
```

Prueba 2:buscarLosProcedimientos

Comprueba que el servicio utilice correctamente findByPk para buscar un procedimiento por su ID, incluyendo las relaciones necesarias (includes).

```
test('buscarLosProcedimientos -> delega a findByPk con includes', async () => {
  models.procedimientos.findByPk.mockResolvedValue({ id: 5 });
  const res = await svc.buscarLosProcedimientos(5);
  expect(models.procedimientos.findByPK).toHaveBeenCalledWith(5, expect.any(Object));
  expect(res).toEqual({ id: 5 });
});
```

Prueba 3:crearLosProcedimientos

Valida que el servicio cree correctamente un nuevo procedimiento utilizando create con los datos recibidos.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('crearLosProcedimientos -> delega a create', async () => {
  models.procedimientos.create.mockResolvedValue({ id: 9 });
  const res = await svc.crearLosProcedimientos({ nombre: 'x' });
  expect(models.procedimientos.create).toHaveBeenCalledWith({ nombre: 'x' });
  expect(res).toEqual({ id: 9 });
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 4:listarLosProcedimientosPorCategoria

Comprueba que el servicio liste los procedimientos filtrando correctamente por el ID de categoría.

```
test('listarLosProcedimientosPorCategoria -> delega a findAll con where', async () => {
  models.procedimientos.findAll.mockResolvedValue([{ id: 2 }]);
  const res = await svc.listarLosProcedimientosPorCategoria(7);
  expect(models.procedimientos.findAll).toHaveBeenCalledWith(expect.objectContaining({ where: { categoriaId: 7 } }));
  expect(res).toEqual([{ id: 2 }]);
});
```

Prueba 5: eliminarLosProcedimientos

Evalúa que el servicio elimine correctamente un procedimiento usando el método destroy y devuelva la cantidad de registros afectados.

```
test('eliminarLosProcedimientos -> delega a destroy', async () => {
  models.procedimientos.destroy.mockResolvedValue(1);
  const res = await svc.eliminarLosProcedimientos(3);
  expect(models.procedimientos.destroy).toHaveBeenCalledWith({ where: { id: 3 } });
  expect(res).toBe(1);
});
```

Prueba 6: actualizarLosProcedimientos

Valida que el servicio actualice correctamente los datos de un procedimiento existente, delegando la operación al método update.

```
test('actualizarLosProcedimientos -> delega a update y devuelve resultado', async () => {
  models.procedimientos.update.mockResolvedValue([1]);
  const res = await svc.actualizarLosProcedimientos(4, { nombre: 'y' });
  expect(models.procedimientos.update).toHaveBeenCalledWith({ nombre: 'y' }, { where: { id: 4 } });
  expect(res).toEqual([1]);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends npm test -- _tests_/ProcedimientosServices.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/ProcedimientosServices.test.js

PASS  _tests_/ProcedimientosServices.test.js
  Servicios de Procedimientos
    listarLosProcedimientos > delega a findAll con includes (57 ms)
    buscarLosProcedimientos > delega a findByPk con includes (4 ms)
    crearLosProcedimientos > delega a create (3 ms)
    listarLosProcedimientosPorCategoria > delega a findAll con where (4 ms)
    eliminarLosProcedimientos > delega a destroy (2 ms)
    actualizarLosProcedimientos > delega a update y devuelve resultado (2 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:  0 total
Time:        0.999 s
Run all test suites matching _tests_/ProcedimientosServices.test.js/i.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 291 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 292 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	La función ValidarLaCita debe validar la fecha de registro de una cita médica/estética, garantizando: Formato de fecha/hora válido (usando moment-timezone). Comparación respecto al ahora en zona horaria "America/Bogota". Rechazo de fechas en el pasado y aceptación de fechas iguales o posteriores al momento actual.
Objetivo	Asegurar que ValidarLaCita(fechaStr): detecte formatos inválidos y arroje error, rechace fechas pasadas (según hora local de Bogotá), y acepte fechas futuras (o el mismo minuto/instante, según tu criterio de igualdad).
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Librería: moment-timezone (zona: "America/Bogota"). Función bajo prueba: ValidarLaCita(fechaStr) (retorna true o no lanza; lanza Error en inválidos/pasado). Ejemplos de entradas: Formatos válidos ISO: "2025-10-15T09:30:00", "2025-10-15 09:30". Formatos inválidos: "15/35/2025", "2025-13-01", "fecha", "", null.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Mock del “ahora”: usar jest.useFakeTimers().setSystemTime(...) o jest.spyOn(Date, "now") para fijar el tiempo de referencia en Bogotá (p. ej. 2025-10-11T10:00:00-05:00).</p>
Procedimiento de Prueba	<p>Prueba 1 – Fecha inválida</p> <p>Paso 1: Fijar “ahora” a 2025-10-11T10:00:00-05:00 (Bogotá).</p> <p>Paso 2: Llamar ValidarLaCita("fecha"), ValidarLaCita("2025-13-01"), ValidarLaCita("") .</p> <p>Resultado esperado: La función lanza Error con mensaje tipo "Fecha inválida" (o similar). No debe aceptar cadenas con mes/día fuera de rango ni texto libre.</p> <p>Prueba 2 – Fecha en el pasado</p> <p>Paso 1: Fijar “ahora” a 2025-10-11T10:00:00-05:00.</p> <p>Paso 2: Invocar ValidarLaCita("2025-10-11T09:59:00") y ValidarLaCita("2025-10-10 15:00").</p> <p>Resultado esperado: La función lanza Error con mensaje tipo "Fecha en el pasado" (comparación hecha en tz America/Bogota). No permite registrar citas retroactivas.</p> <p>Prueba 3 – Fecha futura (válida)</p> <p>Paso 1: Fijar “ahora” a 2025-10-11T10:00:00-05:00.</p> <p>Paso 2: Invocar ValidarLaCita("2025-10-11T10:01:00") y ValidarLaCita("2025-12-01 08:30").</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: No lanza excepción y/o retorna true. La fecha es aceptada por estar estrictamente posterior al "ahora" en Bogotá.</p> <p>(Si tu regla permite "igual al minuto actual" como válida, agrega un caso 2025-10-11T10:00:00 esperando aceptación.)</p>		
Datos de salida - Resultado Esperado	<p>Entradas con formato inválido → Error "Fecha inválida".</p> <p>Entradas anteriores al ahora (en Bogotá) → Error "Fecha en el pasado".</p> <p>Entradas futuras (o iguales, si tu política lo permite) → válidas (sin error / true).</p> <p>La comparación se realiza con zona horaria "America/Bogota" usando moment-timezone.</p> <p>No hay excepciones no controladas distintas a las esperadas por validación.</p>		
Resultado Obtenido Prueba Exitosa	ValidarLaCita valida formato y temporalidad en la zona America/Bogota, rechazando fechas inválidas/pasadas y aceptando fechas futuras como válidas.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

34. Validarfecharegistro

Comprobar el correcto funcionamiento de la función ValidarLaCita, encargada de validar las fechas de registro de citas médicas o estéticas, asegurando que estas cumplan con el formato correcto y no correspondan a fechas pasadas según la zona horaria **"America/Bogota"**.

Prueba 1: Fecha inválida

Evaluá que el validador arroje un error cuando la fecha ingresada no tiene un formato válido reconocido por moment-timezone.

```
describe('Validación de fecha de registro de cita', () => [
  it('lanza error si la fecha es inválida', () => {
    expect(() => ValidarLaCita({ fecha: 'fecha-mala' })).toThrow('Fecha de la cita no válida');
  });

  it('lanza error si la fecha es en el pasado', () => {
    const ayer = moment.tz('America/Bogota').subtract(1, 'day').format();
    expect(() => ValidarLaCita({ fecha: ayer })).toThrow('La fecha de la cita no puede ser pasada');
  });

  it('no lanza error si la fecha es futura', () => {
    const mañana = moment.tz('America/Bogota').add(1, 'day').format();
    expect(() => ValidarLaCita({ fecha: mañana })).not.toThrow();
  });
]);
```

Prueba 2: Fecha en el pasado

Comprueba que el sistema rechace las citas cuya fecha se encuentra en el pasado (respecto a la hora local de Bogotá), impidiendo que se registren citas retroactivas.

```
it('lanza error si la fecha es en el pasado', () => [
  const ayer = moment.tz('America/Bogota').subtract(1, 'day').format();
  expect(() => ValidarLaCita({ fecha: ayer })).toThrow('La fecha de la cita no puede ser pasada');
]);

it('no lanza error si la fecha es futura', () => {});
```

Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Prueba 3: Fecha futura

Confirma que el sistema acepte una fecha futura como válida, sin lanzar excepciones. Esto garantiza que el registro de citas solo sea posible para fechas iguales o posteriores al momento presente.

```
it('no lanza error si la fecha es futura', () => {
    const manana = moment.tz('America/Bogota').add(1, 'day').format();
    expect(() => ValidarLaCita({ fecha: manana })).not.toThrow();
});
```

```
dstevengnz@dstevengnz:~/Escritorio/clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends$ npm test -- __tests__/_Validarfecharegistro.test
> clinestetica@1.0.0 test
> jest --passWithNoTests __tests__/_Validarfecharegistro.test
PASS  tests/_Validarfecharegistro.test.js
  Validación de fecha de registro de cita
    ✓ lanza error si la fecha es inválida (15 ms)
    ✓ lanza error si la fecha es en el pasado (8 ms)
    ✓ no lanza error si la fecha es futura (2 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.712 s
Run all test suites matching __tests__/_Validarfecharegistro.test/i.
dstevengnz@dstevengnz:~/Escritorio/clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends$
```

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	Validar el comportamiento de los middlewares de validación de usuarios: validarUsuario (registros internos con rol). validarUsuarioPublico (registros públicos sin rol). Deben aplicar reglas de campos requeridos, formatos y condiciones antes de crear el usuario, retornando 400



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	cuento corresponda y permitiendo next() en entradas válidas.
Objetivo	Comprobar que: <ol style="list-style-type: none">se rechacen solicitudes con faltantes o formatos inválidos;terminos_condiciones sea obligatorio y verdadero para validarUsuario;el flujo pase con datos correctos;en validarUsuarioPublico no se exija rol;se validen email y número de documento en registros públicos.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Middlewares: validarUsuario(req, res, next), validarUsuarioPublico(req, res, next). Campos típicos esperados: nombres, apellidos, email, password, documento, telefono, rol (solo interno), terminos_condiciones. Formatos: email válido, documento numérico (string de dígitos o number), telefono numérico/regex, password longitud mínima (p. ej. ≥ 6).
Procedimiento de Prueba	Prueba 1 – validarUsuario: rechaza faltantes/vacíos (400) Paso 1: Prepara un cuerpo de solicitud con campos obligatorios vacíos o ausentes; por ejemplo, nombre vacío, email con formato inválido, contraseña vacía,



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>documento y teléfono vacíos, rol presente, términos aceptados.</p> <p>Paso 2: Ejecuta el middleware validarUsuario con objetos de respuesta simulados para capturar estado y mensaje.</p> <p>Resultado esperado: Respuesta 400 Bad Request con una lista de errores que mencione los campos vacíos/faltantes y el email inválido. No debe llamarse next().</p> <p>Prueba 2 – validarUsuario: términos y condiciones obligatorio (400)</p> <p>Paso 1: Prepara un cuerpo válido en todos los campos, pero con términos y condiciones en falso (ya sea booleano falso o el texto “false”).</p> <p>Paso 2: Ejecuta validarUsuario.</p> <p>Resultado esperado: Respuesta 400 con un error indicando que debe aceptar términos y condiciones. No debe llamarse next().</p> <p>Prueba 3 – validarUsuario: entrada válida (pasa a next)</p> <p>Paso 1: Prepara un cuerpo completo y válido: nombres y apellidos no vacíos, email con formato correcto, contraseña de longitud suficiente, documento y teléfono numéricos, rol presente y términos aceptados como verdaderos.</p> <p>Paso 2: Ejecuta validarUsuario.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Resultado esperado: La validación pasa y se llama a next() exactamente una vez; no se escribe respuesta.</p> <p>Prueba 4 – validarUsuarioPublico: sin rol (pasa a next)</p> <p>Paso 1: Prepara un cuerpo sin el campo rol, con el resto de campos válidos: nombres, apellidos, email válido, contraseña suficiente, documento y teléfono numéricos, y términos aceptados.</p> <p>Paso 2: Ejecuta validarUsuarioPublico.</p> <p>Resultado esperado: La validación pasa (no se exige rol) y se llama a next() una vez.</p> <p>Prueba 5 – validarUsuarioPublico: email/documento inválidos (400)</p> <p>Paso 1: Prepara un cuerpo con email en formato incorrecto y documento no numérico (por ejemplo, con letras mezcladas).</p> <p>Paso 2: Ejecuta validarUsuarioPublico.</p> <p>Resultado esperado: Respuesta 400 Bad Request con una lista de errores que incluya “Email inválido” y “Documento debe ser numérico”. No debe llamarse next().</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Datos de salida - Resultado Esperado	validarUsuario → 400 con lista de errores en faltantes/formatos; 400 si terminos_condiciones no es verdadero; next() en caso válido. validarUsuarioPublico → no exige rol; 400 por email inválido o documento no numérico; next() en caso válido. Las respuestas de error son JSON coherentes; sin excepciones no controladas.		
Resultado Obtenido Prueba Exitosa	Los middlewares validarUsuario y validarUsuarioPublico aplican correctamente reglas de campos requeridos, formatos y condiciones, bloqueando entradas inválidas (400) y permitiendo el flujo con next() cuando los datos son válidos.	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

35. Validaciones

Validar el comportamiento de los **middlewares de validación de usuarios**:

- validarUsuario (para registros internos con rol).
- validarUsuarioPublico (para registros públicos sin rol).

Estas pruebas aseguran que las reglas de validación de campos requeridos, formatos y condiciones se apliquen correctamente antes del registro del usuario

Prueba 1:validarUsuario

Verifica que el middleware validarUsuario rechace solicitudes con campos vacíos o faltantes, devolviendo un estado **400** y una lista de errores.

```
describe('Middlewares de Validaciones (usuarios)', () => {
  test('validarUsuario -> 400 si faltan campos', async () => {
    const app = mount([validarUsuario]);
    const res = await request(app).post('/x').send({});
    expect(res.status).toBe(400);
    expect(res.body.errores?.length).toBeGreaterThan(0);
  });
});
```

Prueba 2:validarUsuario

Comprueba que el campo terminos_condiciones sea obligatorio y válido.

Si es false o el string 'false', el middleware debe rechazar la solicitud.

```
test('validarUsuario -> 400 si términos es false/string false', async () => {
  const app = mount([validarUsuario]);
  const res = await request(app).post('/x').send({ ...basePayload, rol: 'usuario', terminos_condiciones: 'false' });
  expect(res.status).toBe(400);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Prueba 3: validarUsuario

Evaluá que el middleware permita el paso cuando todos los campos son válidos, incluyendo rol y terminos_condiciones: true.

```
test('validarUsuario -> 200 cuando todo es válido', async () => {
  const app = mount([validarUsuario]);
  const res = await request(app).post('/x').send({ ...basePayload, rol: 'usuario' });
  expect(res.status).toBe(200);
  expect(res.body).toEqual({ ok: true });
});
```

Prueba 4: validarUsuarioPublico

Confirma que los usuarios públicos puedan registrarse sin incluir el campo rol y que la validación se complete correctamente.

```
test('validarUsuarioPublico -> 200 sin rol si válido', async () => {
  const app = mount([validarUsuarioPublico]);
  const res = await request(app).post('/x').send({ ...basePayload });
  expect(res.status).toBe(200);
});
```

Prueba 5: validarUsuarioPublico

Verifica que el middleware rechace solicitudes con formato de correo no válido o número de documento no numérico.



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('validarUsuarioPublico -> 400 con correo inválido y doc no numérico', async () => {
  const app = mount([validarUsuarioPublico]);
  const res = await request(app).post('/x').send({
    ...basePayload,
    correo: 'mal-correo',
    numerodocumento: 'abc',
  });
  expect(res.status).toBe(400);
  expect(res.body.errores?.length).toBeGreaterThan(0);
});
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Resultado

```
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- _tests_/_ValidacionesMiddleware.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests _tests_/_ValidacionesMiddleware.test.js

 PASS  tests/_ValidacionesMiddleware.test.js
  Middlewares de Validaciones (usuarios)
    ✓ validarUsuario -> 400 si faltan campos (114 ms)
    ✓ validarUsuario -> 400 si términos es: false/string false (13 ms)
    ✓ validarUsuario -> 200 cuando todo es válido (11 ms)
    ✓ validarUsuarioPublico -> 200 sin rol si válido (8 ms)
    ✓ validarUsuarioPublico -> 400 con correo inválido y doc no numérico (9 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.921 s
Ran all test suites matching _tests_/_ValidacionesMiddleware.test.js/i.
dstevengnz@dstevengnz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ █
```

Fecha de realización: 15-09-2025	Duración de la prueba: 12 min
Requerimiento Funcional de la prueba	Validar integralmente el Controlador de Usuarios, cubriendo: registro, prerregistro, confirmación, inicio de sesión, perfil, edición/eliminación, activación de cuentas y manejo de notificaciones (doctor/usuario).
Objetivo	Comprobar que: la gestión de usuarios (listar, buscar, crear, actualizar, eliminar) responde con códigos correctos; los procesos de registro (prerregistro, confirmación, creación de administradores) aplican validaciones; el inicio de sesión gestiona credenciales erróneas, bloqueo y limpieza de intentos fallidos;



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	la activación de cuentas valida estados y responde adecuadamente; las notificaciones (obtener, marcar leídas, archivar, historial) funcionan para doctor/usuario.
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Métodos/Endpoints: listar/buscar/crear/actualizar/eliminar usuarios; prerregistro; confirmación; login; perfil; activar usuario; notificaciones (obtener, marcar como leídas, archivar, historial) para doctor y usuario. Parámetros y contexto: id de usuario, credenciales, token, req.usuario (roles admin/doctor/usuario). Dependencias mock: servicios de usuarios y notificaciones; middleware de intentos fallidos.
Procedimiento de Prueba	Prueba 1 – Listar/Buscar usuarios Paso 1: Invocar listar/buscar con id válido o sin filtros. Paso 2: Simular servicio con: (a) datos existentes, (b) sin registro, (c) excepción. Resultado esperado: 200 cuando existen datos. 404 cuando no existe el recurso solicitado. 500 ante error del servicio.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Prueba 2 – Listar doctores/usuarios</p> <p>Paso 1: Invocar endpoints de listado por rol (doctores / solo usuarios).</p> <p>Paso 2: Simular (a) datos válidos, (b) error del servicio.</p> <p>Resultado esperado: 200 con datos válidos; 500 en error.</p> <p>Prueba 3 – Crear usuarios/admin</p> <p>Paso 1: Enviar body válido para creación de usuario y de administrador.</p> <p>Paso 2: Simular (a) creación exitosa, (b) excepción en servicio.</p> <p>Resultado esperado: 201 con confirmación de creación; 500 error controlado.</p> <p>Prueba 4 – Preregistro y confirmación</p> <p>Paso 1: Preregistro con email válido y confirmación con token.</p> <p>Paso 2: Simular (a) éxito, (b) error lógico (token inválido/expirado o datos inválidos), (c) excepción.</p> <p>Resultado esperado: 200 éxito; 400 error lógico; 500 excepción.</p> <p>Prueba 5 – Perfil y actualización</p> <p>Paso 1: Obtener perfil con req.usuario.id; actualizar con id válido y body válido.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 2: Simular (a) entrada inválida (id/body), (b) no encontrado, (c) éxito, (d) error.</p> <p>Resultado esperado:</p> <p>200 en éxito; 400 por validación; 404 no encontrado; 500 error de servicio.</p> <p>Prueba 6 – Eliminar usuarios</p> <p>Paso 1: Enviar id válido y simular eliminación exitosa.</p> <p>Resultado esperado: 200 con confirmación; verificación de llamada al servicio.</p> <p>Prueba 7 – Inicio de sesión</p> <p>Paso 1: Enviar credenciales.</p> <p>Paso 2: Simular (a) credenciales erróneas, (b) éxito, (c) error de servicio.</p> <p>Resultado esperado: 401 rechazo por credenciales; 200 éxito; 500 error.</p> <p>Prueba 8 – Activación de usuario</p> <p>Paso 1: Enviar solicitud de activación con id/token.</p> <p>Paso 2: Simular (a) estado inválido, (b) no autorizado, (c) éxito, (d) error.</p> <p>Resultado esperado: 400 validación de estado; 401 no autorizado; 200 activación exitosa; 500 error.</p>
--	--



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	<p>Prueba 9 – Notificaciones (doctor y usuario)</p> <p>Paso 1: Para rol doctor y usuario: obtener, marcar como leídas, archivar, e historial.</p> <p>Paso 2: Simular (a) entradas válidas, (b) entrada inválida, (c) error del servicio.</p> <p>Resultado esperado: 200 datos/confirmación; 400 validación; 500 error.</p> <p>Prueba 10 – Intentos fallidos (middleware)</p> <p>Paso 1: Simular flujos de login con fallos repetidos y posterior éxito.</p> <p>Resultado esperado: Intentos registrados, bloqueo cuando corresponde, y limpieza tras éxito (verificado).</p>												
Datos de salida - Resultado Esperado	<table><thead><tr><th>Funcionalidad</th><th>Casos evaluados</th><th>Resultados esperados</th></tr></thead><tbody><tr><td>Listar/Buscar usuarios</td><td>200, 404, 500</td><td>Respuestas correctas según del servicio</td></tr><tr><td>Listar doctores/usuarios</td><td>200, 500</td><td>Manejo de errores y datos válidos</td></tr><tr><td>Crear usuarios/admin</td><td>201, 500</td><td>Confirmación de creación o err controlado</td></tr></tbody></table>	Funcionalidad	Casos evaluados	Resultados esperados	Listar/Buscar usuarios	200, 404, 500	Respuestas correctas según del servicio	Listar doctores/usuarios	200, 500	Manejo de errores y datos válidos	Crear usuarios/admin	201, 500	Confirmación de creación o err controlado
Funcionalidad	Casos evaluados	Resultados esperados											
Listar/Buscar usuarios	200, 404, 500	Respuestas correctas según del servicio											
Listar doctores/usuarios	200, 500	Manejo de errores y datos válidos											
Crear usuarios/admin	201, 500	Confirmación de creación o err controlado											



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	Prerreistro y confirmación	200, 400, 500	Validaciones de éxito excepción
	Perfil y actualización	200, 400, 404, 500	Validaciones de ID, ex correcta
	Eliminar usuarios	200	Eliminación confirmada y llamada a se verificada
	Inicio de sesión	401, 200, 500	Rechazo por credenciales errón errores
	Activación de usuario	400, 401, 200, 500	Validación de estado, err de activación
	Notificaciones (doctor y usuario)	200, 400, 500	Obtención, marcado según estado
	Intentos fallidos (middleware)	Verificado	Registra y limpia intent cada caso
Resultado Obtenido Prueba Exitosa	El controlador de Usuarios cumple con las validaciones y respuestas esperadas en todos los flujos	Si(x)	No ()



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	descritos, con manejo correcto de errores, roles y estados.		
--	---	--	--

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 312 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

36. UsuariosControllers

Validar integralmente el **Controlador de Usuarios**, cubriendo las operaciones de registro, prerregistro, confirmación, inicio de sesión, perfil, edición/eliminación, activación de cuentas y manejo de notificaciones tanto para doctores como para usuarios.

Cobertura general:

- Gestión de usuarios:** listar, buscar, crear, actualizar y eliminar.
- Procesos de registro:** prerregistro, confirmación y creación de administradores.
- Inicio de sesión:** control de errores, bloqueo y limpieza de intentos fallidos.
- Activación de cuentas:** validaciones de estado y control de respuestas.
- Notificaciones:** obtener, marcar como leídas, archivar e historial (doctor/usuario).

Funcionalidad	Casos evaluados	Resultados esperados
Listar/Buscar usuarios	200, 404, 500	Respuestas correctas según existencia y errores del servicio
Listar doctores/usuarios	200, 500	Manejo de errores y datos válidos
Crear usuarios/admin	201, 500	Confirmación de creación o error controlado
Prerregistro y confirmación	200, 400, 500	Validaciones de éxito, error lógico y excepción
Perfil y actualización	200, 400, 404, 500	Validaciones de ID, existencia y respuesta correcta
Eliminar usuarios	200	Eliminación confirmada y llamada a servicio verificada
Inicio de sesión	401, 200, 500	Rechazo por credenciales erróneas, éxito y manejo de errores
Activación de usuario	400, 401, 200, 500	Validación de estado, error de servicio y éxito de activación
Notificaciones (doctor y usuario)	200, 400, 500	Obtención, marcado, archivado e historial según estado
Intentos fallidos (middleware)	Verificado	Registra y limpia intentos correctamente en cada caso



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('crearUsuarios -> 201 y 500', async () => {
  svc.clearLosUsuarios.mockResolvedValue({ id: 1 });
  let ctx = mockReqRes({ body: { nombre: 'a' } });
  await controller.crearUsuarios(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(201);
  svc.clearLosUsuarios.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ body: { nombre: 'a' } });
  await controller.crearUsuarios(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});

test('preRegistro -> 200 ok y 400 por error de servicio', async () => {
  svc.preRegistrarUsuario.mockResolvedValue({ success: true });
  let ctx = mockReqRes({ body: { correo: 'a@b.c' } });
  await controller.preRegistro(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.preRegistrarUsuario.mockResolvedValue({ error: 'bad' });
  ctx = mockReqRes({ body: { correo: 'a@b.c' } });
  await controller.preRegistro(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
});

test('confirmarRegistro -> 200 ok, 400 por error y 500 por excepción', async () => {
  svc.confirmarRegistro.mockResolvedValue({ success: true });
  let ctx = mockReqRes({ body: { correo: 'a', codigo: '123' } });
  await controller.confirmarRegistro(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.confirmarRegistro.mockResolvedValue({ success: false, error: 'bad' });
  ctx = mockReqRes({ body: { correo: 'a', codigo: '123' } });
  await controller.confirmarRegistro(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
  svc.confirmarRegistro.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ body: { correo: 'a', codigo: '123' } });
  await controller.confirmarRegistro(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```

```
test('notificaciones_usuario_endpoints -> 200/400/500', async () => {
  expect(ctx.res.statusCode).toBe(200);
  svc.marcarNotificacionUsuarioCompleta.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 }, body: { notificaciónId: 2 } });
  await controller.marcarNotificaciónUsuarioCompleta(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);

  svc.marcarTodasNotificacionesUsuarioComplejadas.mockResolvedValue({ success: true });
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.marcarTodasNotificacionesUsuarioComplejadas(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.marcarTodasNotificacionesUsuarioComplejadas.mockRejectedValue({ success: false, error: 'bad' });
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.marcarTodasNotificacionesUsuarioComplejadas(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
  svc.marcarTodasNotificacionesUsuarioComplejadas.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.marcarTodasNotificacionesUsuarioComplejadas(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);

  svc.archivarNotificacionesUsuario.mockResolvedValue({ success: true, archivadas: 1, activas: 1 });
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.archivarNotificacionesUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.archivarNotificacionesUsuario.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.archivarNotificacionesUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);

  svc.obtenerHistorialNotificacionesUsuario.mockResolvedValue([]);
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.obtenerHistorialNotificacionesUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.obtenerHistorialNotificacionesUsuario.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.obtenerHistorialNotificacionesUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('listarUsuarios -> 200', async () => {
  svc.listarLosUsuarios.mockResolvedValue([ { id: 1 } ]);
  const { req, res } = mockReqRes();
  await controller.listarUsuarios(req, res);
  expect(res.statusCode).toBe(200);
});

test('buscarUsuarios -> 200 y 404', async () => {
  svc.buscarLosUsuarios.mockResolvedValue([ { id: 2 } ]);
  let ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarUsuarios(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);

  svc.buscarLosUsuarios.mockResolvedValue(null);
  ctx = mockReqRes({ params: { id: 2 } });
  await controller.buscarUsuarios(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(404);
});

test('listarDoctores/listarSoloUsuarios -> 200 y 500', async () => {
  svc.listarSoloDoctores.mockResolvedValue([ { id: 1 } ]);
  let ctx = mockReqRes();
  await controller.listarDoctores(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.listarSoloDoctores.mockRejectedValue(new Error('x'));
  ctx = mockReqRes();
  await controller.listarDoctores(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});

test('iniciarSesion -> 401 error, 200 ok y 500 excepción', async () => {
  svc.iniciarSesion.mockResolvedValue({ error: 'bad' });
  let ctx = mockReqRes({ body: { correo: 'a', contrasena: 'b' } });
  await controller.iniciarSesion(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(401);
  expect(registrarIntentoFallido).toHaveBeenCalled();
  svc.iniciarSesion.mockResolvedValue({ token: 't' });
  ctx = mockReqRes({ body: { correo: 'a', contrasena: 'b' } });
  await controller.iniciarSesion(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  expect(limpiarIntentos).toHaveBeenCalled();
  svc.iniciarSesion.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ body: { correo: 'a', contrasena: 'b' } });
  await controller.iniciarSesion(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});

test('activacionUsuario -> 400 estado inválido, 401 error servicio, 200 ok y 500 excepción', async () => {
  let ctx = mockReqRes({ params: { id: 1 }, body: { estado: 'no-bool' } });
  await controller.activacionUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
  svc.activarUsuario.mockResolvedValue({ error: 'bad' });
  ctx = mockReqRes({ params: { id: 1 }, body: { estado: true } });
  await controller.activacionUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(401);
  svc.activarUsuario.mockResolvedValue({ usuario: { estado: true } });
  ctx = mockReqRes({ params: { id: 1 }, body: { estado: false } });
  await controller.activacionUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.activarUsuario.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 }, body: { estado: true } });
  await controller.activacionUsuario(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
```

Número de Documento:

FS-DOC Formato Manual de Usuario

 Fecha de Creación:
1/02/2025

 Elaborado por:
 Instructores área Software

 Nombre del Documento: **Formato para la construcción del Manual de Usuario**

```

test('marcarNotificacionComoLeida y todas -> validaciones, 200 ok, 400 error, 500 excepción', async () => {
  let ctx = mockReqRes({ params: { id: 1 }, body: {} });
  await controller.marcarNotificacionComoLeida(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
  svc.marcarNotificacionComoLeida.mockResolvedValue({ success: true });
  ctx = mockReqRes({ params: { id: 1 }, body: { notificacionId: 9 } });
  await controller.marcarNotificacionComoLeida(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.marcarNotificacionComoLeida.mockResolvedValue({ success: false, error: 'bad' });
  ctx = mockReqRes({ params: { id: 1 }, body: { notificacionId: 9 } });
  await controller.marcarNotificacionComoLeida(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
  svc.marcarNotificacionComoLeida.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 }, body: { notificacionId: 9 } });
  await controller.marcarNotificacionComoLeida(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);

  svc.marcarTodasNotificacionesComoLeidas.mockResolvedValue({ success: true });
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.marcarTodasNotificacionesComoLeidas(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(200);
  svc.marcarTodasNotificacionesComoLeidas.mockResolvedValue({ success: false, error: 'bad' });
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.marcarTodasNotificacionesComoLeidas(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(400);
  svc.marcarTodasNotificacionesComoLeidas.mockRejectedValue(new Error('x'));
  ctx = mockReqRes({ params: { id: 1 } });
  await controller.marcarTodasNotificacionesComoLeidas(ctx.req, ctx.res);
  expect(ctx.res.statusCode).toBe(500);
});
    
```

Resultado

```

dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ npm test -- __tests__/_UsuariosControllers.test.js
> clinestetica@1.0.0 test
> jest --passWithNoTests __tests__/_UsuariosControllers.test.js

PASS  __tests__/_UsuariosControllers.test.js
  Controlador de Usuarios
    ✓ ListarUsuarios -> 200 (5 ms)
    ✓ buscarUsuarios -> 200 y 404 (3 ms)
    ✓ ListarDoctores /listarTodosusuarios -> 200 y 500 (4 ms)
    ✓ crearUsuarios -> 201 y 500 (2 ms)
    ✓ preRegistro -> 200 ok y 400 por error de servicio (2 ms)
    ✓ confirmarRegistro -> 200 ok, 400 por error y 500 por excepción (1 ms)
    ✓ crearUsuariosAdmin -> 201 y 500 (1 ms)
    ✓ perfilUsuario -> 200 y 404 (2 ms)
    ✓ actualizarUsuario -> 400 invalido, 404 no encontrado, 200 ok y 500 error (4 ms)
    ✓ eliminarUsuarios -> 200 (3 ms)
    ✓ iniciarSesion -> 401 error, 200 ok y 500 excepción (2 ms)
    ✓ activacionUsuario -> 400 estado inválido, 401 error servicio, 200 ok y 500 excepcion (1 ms)
    ✓ notificaciones doctor/usuario -> 200 y 500 (3 ms)
    ✓ marcarNotificacionComoLeida y todas -> validaciones, 200 ok, 400 error, 500 excepción (3 ms)
    ✓ archivar / historial notificaciones doctor -> 200 ok y 500 (1 ms)
    ✓ notificaciones usuario endpoints -> 200/400/500 (4 ms)

Test Suites: 1 passed, 1 total
Tests:    16 passed, 16 total
Snapshots: 0 total
Time:    0.725 s, estimated 2 s
Ran all test suites matching __tests__/_UsuariosControllers.test.js/i.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend$ 
    
```



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

Fecha de realización: 15-09-2025	Duración de la prueba: 5 min
Requerimiento Funcional de la prueba	Validar el correcto funcionamiento de la capa de Servicios de Usuarios: CRUD, inicio de sesión (JWT), activación, actualización/eliminación, prerregistro y confirmación (con Redis), y gestión de notificaciones para doctores y usuarios (Redis + BD).
Objetivo	Verificar llamadas correctas a modelos Sequelize y a Redis. Comprobar generación/retorno de JWT y validaciones de correo/estado/contraseña. Asegurar flujos de prerregistro/confirmación (código, expiración, limpieza de claves). Validar activación (existencia/estado/errores). Confirmar notificaciones (obtener, marcar una/todas, archivar, historial) con persistencia y caché. Controlar intentos y bloqueo temporal en verificación de código (Redis).
Tipo de Prueba	Prueba Unitaria (Jest)
Datos de entrada de la prueba	Funciones de servicio: listar, buscarPorId, crear, actualizar, eliminar, prerregistro, confirmar, login, activarUsuario, notificaciones.obtener/marcar/archivar/historial, verificarCodigo. Dependencias mock: modelos Sequelize (Usuarios, Notificaciones), Redis (get/setEx/del/incr/expire), util de JWT, util de email.



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

	Datos: id, email, password, estado, codigo, payloads de usuario y de notificación.
Procedimiento de Prueba	<p>Prueba 1 – CRUD Usuarios (listar/buscar/crear/actualizar/eliminar)</p> <p>Paso 1: Invocar cada método con entradas válidas.</p> <p>Paso 2: Simular: (a) éxito, (b) no encontrado (buscar/actualizar/eliminar), (c) excepción BD.</p> <p>Resultado esperado:</p> <p>Éxito → 200 lógico del servicio (retornos correctos).</p> <p>No encontrado → null/0 filas afectadas (equivalente a 404 a nivel controlador).</p> <p>Excepción → error propagado (equivalente a 500 a nivel controlador).</p> <p>Verificadas las llamadas a Sequelize (findAll/findByPk/create/update/destroy) con parámetros correctos.</p> <p>Prueba 2 – Registro y Pre-registro</p> <p>Paso 1: Validar correo/contraseña y duplicados.</p> <p>Paso 2: Simular uso de Redis para registro temporal (setEx) y envío de código por correo.</p> <p>Resultado esperado:</p> <p>Almacena temporalmente datos en Redis con TTL.</p> <p>Retorna confirmación de envío.</p> <p>En duplicados/formato inválido → error de negocio controlado.</p> <p>Prueba 3 – Confirmación de cuenta</p>



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Paso 1: Enviar código y email.</p> <p>Paso 2: Simular: (a) código correcto, (b) inválido, (c) expirado (clave inexistente), (d) excepción BD.</p> <p>Resultado esperado:</p> <p>Correcto → crea usuario, limpia Redis (del) y retorna confirmación</p> <p>Inválido/expirado → mensajes de error apropiados (equivalentes a 400).</p> <p>Excepción → error propagado (equivalente a 500)</p> <p>Prueba 4 – Inicio de sesión</p> <p>Paso 1: Probar: (a) éxito, (b) usuario inactivo, (c) correo inexistente, (d) contraseña errónea, (e) excepción.</p> <p>Paso 2: Verificar generación de JWT y retorno de usuario.</p> <p>Resultado esperado:</p> <p>Éxito → token JWT y datos de usuario.</p> <p>Inactivo/inexistente/contraseña errónea → errores equivalentes a 401.</p> <p>Excepción → error propagado (500 lógico).</p> <p>Prueba 5 – Activación de usuario</p> <p>Paso 1: Probar: (a) existente y actualizable, (b) no encontrado, (c) error BD.</p> <p>Resultado esperado:</p> <p>Actualiza estado a activo y confirma operación.</p> <p>No encontrado → retorno nulo/0 filas (equivalente a 404).</p> <p>Error BD → error propagado.</p>
--	---



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

	<p>Prueba 6 – Notificaciones (doctor y usuario)</p> <p>Paso 1: obtener, marcar (una/todas), archivar, historial.</p> <p>Paso 2: Simular persistencia (BD) y actualización de Redis (listas, contadores).</p> <p>Resultado esperado:</p> <p>Devuelve datos correctos; marca y archiva consistentemente en Redis y BD.</p> <p>En fallas de caché o BD → error controlado (propagado).</p> <p>Prueba 7 – Verificación de código (seguridad con Redis)</p> <p>Paso 1: Probar: (a) correcto, (b) erróneo, (c) bloqueado por demasiados intentos, (d) sin código almacenado.</p> <p>Paso 2: Validar incr/expire para intentos y bloqueo temporal.</p> <p>Resultado esperado:</p> <p>Correcto → éxito y limpieza de claves.</p> <p>Erróneo → incrementa intentos; devuelve error.</p> <p>Bloqueado → retorna estado de bloqueo temporal.</p> <p>Sin código → error de inexistencia/expiración.</p>
--	---

Datos de salida - Resultado Esperado	Módulo	Casos evaluados	Resultados esperados
	CRUD Usuarios	Crear, listar, buscar, actualizar y eliminar	Llamadas correctas a Sequelize ; retornos coherentes; errores 200/404/500 controlados a nivel controlador
	Registro y Pre-registro	Validaciones de correo, contraseña y duplicados	Uso de Redis para registro temporal y envío de código por correo
	Confirmación	Código correcto, inválido y expirado	Creación de usuario, limpieza de Redis y mensajes de error apropiados
	Inicio de sesión	Éxito, inactivo, correo inexistente, contraseña errónea, excepción	Retorno de JWT y usuario; errores tipo 401 y propagación de excepción
	Activación de usuario	Existente, no encontrado, error BD	Actualiza estado / retorna no encontrado / propaga error
	Notificaciones	Obtener, marcar, archivar, historial (usuario/doctor)	Actualiza Redis , persiste en BD y maneja errores de caché o base
	Verificación de código	Correcto, erróneo, bloqueado, sin código	Control de intentos en Redis con bloqueo temporal e incrementos

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 321 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			

Resultado Obtenido Prueba Exitosa	El Servicio de Usuarios cumple con las operaciones y flujos descritos: integra Sequelize y Redis correctamente, genera JWT, gestiona prerregistro/confirmación con códigos y expiraciones, activa usuarios, y administra notificaciones con consistencia entre caché y base.	Si(x)	No ()
--	--	----------------	---------------



Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento:

Formato para la construcción del Manual de Usuario

37. UsuariosServices

Validar el correcto funcionamiento de la capa de **Servicios de Usuarios**, incluyendo operaciones de **creación, inicio de sesión, activación, actualización/eliminación**, además de los flujos de **prerregistro, confirmación de cuenta y gestión de notificaciones** para doctores y usuarios.

Cobertura de pruebas:

- **CRUD básico:** listar, buscar, crear, actualizar y eliminar usuarios.
- **Inicio de sesión:** generación de token JWT, verificación de correo/estado/contraseña y manejo de errores.
- **Activación:** actualización de estado, validación de existencia y control de excepciones.
- **Prerregistro y confirmación:** validación de correo y contraseña, almacenamiento temporal en Redis, envío y verificación de código.
- **Notificaciones (doctor y usuario):** obtención, marcado de una/todas, archivado e historial (Redis y base de datos).

Módulo	Casos evaluados	Resultados esperados
CRUD Usuarios	Crear, listar, buscar, actualizar y eliminar	Llamadas correctas a modelos Sequelize, respuestas 200/404/500 controladas
Registro y Pre-registro	Validaciones de correo, contraseña y duplicados	Uso de Redis para registro temporal y envío de código por correo
Confirmación	Código correcto, inválido y expirado	Creación de usuario, limpieza de claves Redis y mensajes de error apropiados
Inicio de sesión	Éxito, usuario inactivo, correo inexistente, contraseña errónea y excepción	Retorno de token JWT y usuario; errores 401 simulados
Activación de usuario	Existente, no encontrado y error en BD	Actualización de estado o retorno de mensaje de error
Notificaciones	Obtener, marcar, archivar e historial (usuario/doctor)	Actualiza Redis, persiste en BD y maneja errores de caché o base
Verificación de código	Correcto, erróneo, bloqueado, sin código	Control de intentos con Redis e incremento de bloqueo temporal



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('iniciarSesion falla por contraseña incorrecta', async () => {
  models.usuarios.findOne.mockResolvedValue({ correo: 'x@x.com', contrasena: 'hash', estado: true });
  bcrypt.compare.mockResolvedValue(false);
  const res = await svc.iniciarSesion('x@x.com', 'bad');
  expect(res).toEqual({ error: 'Credenciales incorrectas' });
});

test('activarUsuario actualiza estado y devuelve mensaje', async () => {
  const save = jest.fn(async () => {});
  models.usuarios.findByPk.mockResolvedValue({ estado: false, save });
  const res = await svc.activarUsuario(1, true);
  expect(save).toHaveBeenCalled();
  expect(res).toHaveProperty('mensaje');
  expect(res.usuario.estado).toBe(true);
});

test('preRegistrarUsuario valida correo y contraseña y guarda en Redis y envía código', async () => {
  models.usuarios.findOne.mockResolvedValue(null);
  const result = await svc.preRegistrarUsuario({ correo: 'p@p.com', contrasena: '123', nombre: 'Pedro' });
  expect(redis.setEx).toHaveBeenCalled();
  expect(result).toHaveProperty('success', true);
});

test('confirmarRegistro feliz crea usuario y limpia Redis', async () => {
  jest.spyOn(svc, 'verificarCodigo').mockResolvedValue({ success: true });
  const payload = { nombre: 'Ana', correo: 'a@a.com', contrasena: 'hashed:pw', rol: 'usuario' };
  const redisMock = require('../config/redis');
  redisMock.get.mockResolvedValue(JSON.stringify(payload));
  models.usuarios.findOne.mockResolvedValue(null);
  models.usuarios.create.mockResolvedValue({ id: 10, nombre: 'Ana' });

  const res = await svc.confirmarRegistro('a@a.com', '000000');
  expect(models.usuarios.create).toHaveBeenCalledWith(expect.objectContaining({ correo: 'a@a.com' }));
  expect(redis.del).toHaveBeenCalled();
  expect(res).toHaveProperty('success', true);
});
```

```
test('listarLosUsuarios delega a modelo', async () => {
  models.usuarios.findAll.mockResolvedValue([{ id: 1 }]);
  const res = await svc.listarLosUsuarios();
  expect(models.usuarios.findAll).toHaveBeenCalled();
  expect(res).toEqual({ id: 1 });
});

test('crearLosUsuarios hashea contraseña y envía correo', async () => {
  models.usuarios.create.mockResolvedValue({ id: 2, nombre: 'Juan' });
  const res = await svc.crearLosUsuarios({ nombre: 'juan', correo: 'j@y.com', contrasena: '123' });
  expect(models.usuarios.create).toHaveBeenCalledWith(expect.objectContaining({
    nombre: expect.any(String),
    contrasena: 'hashed:123',
  }));
  expect(corre.EnviarCorreo).toHaveBeenCalled();
  expect(res).toEqual({ id: 2, nombre: 'Juan' });
});

test('iniciarSession feliz retorna token y usuario', async () => {
  models.usuarios.findOne.mockResolvedValue({
    id: 1, correo: 'a@a.com', contrasena: 'abc', rol: 'usuario', estado: true, nombre: 'Ana'
  });
  bcrypt.compare.mockResolvedValue(true);
  const res = await svc.iniciarSession('a@a.com', 'abc');
  expect(jwt.sign).toHaveBeenCalledWith({ id: 1, correo: 'a@a.com', rol: 'usuario' }, 'secret', { expiresIn: '3h' });
  expect(res).toHaveProperty('token', 'jwt-token');
  expect(res).toHaveProperty('usuario');
});

test('iniciarSession falla por correo no registrado', async () => {
  models.usuarios.findOne.mockResolvedValue(null);
  const res = await svc.iniciarSession('x@x.com', 'p');
  expect(res).toEqual({ error: 'Correo no registrado' });
});
```



Número de Documento:

FS-DOC Formato Manual de UsuarioFecha de Creación:
1/02/2025Elaborado por:
Instructores área SoftwareNombre del Documento: **Formato para la construcción del Manual de Usuario**

```
test('actualizarLosUsuario elimina rol si no es doctor', async () => {
  models.usuarios.update.mockResolvedValue([1]);
  const datos = { nombre: 'x', rol: 'admin' };
  const res = await svc.actualizarLosUsuario(5, datos, 'usuario');
  expect(models.usuarios.update).toHaveBeenCalledWith({ nombre: 'x' }, { where: { id: 5 } });
  expect(res).toEqual([1]);
});

test('actualizarLosUsuario retorna error si usuario no existe', async () => {
  models.usuarios.findByPk.mockResolvedValue(null);
  const res = await svc.actualizarLosUsuario(500, { nombre: 'y' }, 'doctor');
  expect(res).toEqual({ error: 'Usuario no encontrado' });
});

test('actualizarLosUsuario propaga error de update', async () => {
  models.usuarios.findByPk.mockResolvedValue({ id: 1 });
  models.usuarios.update.mockRejectedValue(new Error('fail'));
  await expect(svc.actualizarLosUsuario(1, { nombre: 'z' }, 'doctor')).rejects.toThrow('fail');
});

test('iniciarSesion maneja excepción y retorna error estándar', async () => {
  models.usuarios.findOne.mockRejectedValue(new Error('db down'));
  const res = await svc.iniciarSesion('e@e.com', 'pw');
  expect(res).toHaveProperty('error');
});

test('activarUsuario retorna error si no existe', async () => {
  models.usuarios.findByPk.mockResolvedValue(null);
  const res = await svc.activarUsuario(1, true);
  expect(res).toEqual({ error: 'Usuario no encontrado' });
});

test('activarUsuario maneja excepción y retorna error', async () => {
  models.usuarios.findByPk.mockRejectedValue(new Error('boom'));
  const res = await svc.activarUsuario(1, true);
  expect(res).toEqual({ error: 'Error al activar/desactivar usuario' });
});
```

Número de Documento:

FS-DOC Formato Manual de Usuario

Fecha de Creación:
1/02/2025Elaborado por:
Instructores área Software

Nombre del Documento: Formato para la construcción del Manual de Usuario

Resultado

```
PASS: __tests__\ UsuariosServices.test.js
Servicios de Usuarios
✓ listarLosUsuarios delega a modelo (20 ms)
✓ crearLosUsuarios hashea contraseña y envía correo (6 ms)
✓ iniciarSesion feliz retorna token y usuario (83 ms)
✓ iniciarSesion falla por correo no registrado (2 ms)
✓ iniciarSesion falla por usuario inactivo (4 ms)
✓ iniciarSesion falla por contraseña incorrecta (2 ms)
✓ activarUsuario actualiza estado y devuelve mensaje (3 ms)
✓ preRegistrarUsuario valida correo y contraseña y guarda en Redis y envía código (3 ms)
✓ confirmarRegistro feliz crea usuario y limpia Redis (3 ms)
✓ listarSoloDoctores usa filtro y atributos (1 ms)
✓ listarSoloUsuarios usa filtro y atributos (1 ms)
✓ buscarLosUsuarios por PK (1 ms)
✓ crearLosUsuariosAdmin hashea y envía correo (2 ms)
✓ eliminarLosUsuarios destruye cuando existe (1 ms)
✓ eliminarLosUsuarios retorna null cuando no existe (1 ms)
✓ actualizarLosUsuario elimina rol si no es doctor (2 ms)
✓ actualizarLosUsuario retorna error si usuario no existe (1 ms)
✓ actualizarLosUsuario propaga error de update (62 ms)
✓ iniciarSesion maneja excepción y retorna error estándar (32 ms)
✓ activarUsuario retorna error si no existe (1 ms)
✓ activarUsuario maneja excepción y retorna error (24 ms)
✓ obtenerNotificacionesPorUsuario migra leido y actualiza Redis (3 ms)
✓ obtenerNotificacionesPorUsuario maneja error y retorna error object (21 ms)
✓ obtenerNotificacionesDoctor migra y actualiza Redis (3 ms)
✓ obtenerNotificacionesDoctor en error retorna [] (24 ms)
✓ marcarNotificacionComoLeida actualiza BD y Redis (3 ms)
✓ marcarNotificacionComoLeida cuando no existe retorna error (2 ms)
✓ marcarNotificacionComoLeida maneja excepción (10 ms)
✓ marcarTodasNotificacionesComoLeidas actualiza todas (2 ms)
✓ marcarTodasNotificacionesComoLeidas maneja error (11 ms)
✓ archivarNotificacionesLeidas actualiza y limpia cache (1 ms)
✓ archivarNotificacionesLeidas maneja error (11 ms)
✓ obtenerHistorialNotificaciones devuelve lista ordenada (2 ms)
✓ obtenerHistorialNotificaciones maneja error (13 ms)
✓ marcarNotificacionUsuarioComoLeida actualiza BD y Redis (1 ms)
✓ marcarNotificacionUsuarioComoLeida maneja error (16 ms)
✓ archivarNotificacionesLeidasUsuario actualiza y limpia cache (2 ms)
✓ archivarNotificacionesLeidasUsuario maneja error (15 ms)
✓ marcarTodasNotificacionesUsuarioComoLeidas actualiza todas (1 ms)
✓ marcarTodasNotificacionesUsuarioComoLeidas maneja error (12 ms)
✓ obtenerHistorialNotificacionesUsuario devuelve lista (2 ms)
✓ obtenerHistorialNotificacionesUsuario en error retorna [] (15 ms)
✓ enviarCodigoVerificacion lanza si bloqueado (13 ms)
✓ enviarCodigoVerificacion exito escribe Redis y envia correo (2 ms)
✓ verificarCodigo lanza si no hay código (2 ms)
✓ verificarCodigo incorrecto incrementa intentos y no bloquea aún (1 ms)
✓ verificarCodigo incorrecto bloques al alcanzar límite (2 ms)
✓ verificarCodigo correcto limpia claves y retorna exito (1 ms)
✓ preRegistrarUsuario valida correo requerido (1 ms)
✓ preRegistrarUsuario valida contraseña requerida (1 ms)
✓ preRegistrarUsuario retorna error si ya existe usuario (1 ms)
✓ preRegistrarUsuario exito guarda en Redis y envia código (2 ms)
✓ confirmarRegistro retorna error si verificación falla (1 ms)
✓ confirmarRegistro retorna error si no hay registro pendiente (1 ms)
✓ confirmarRegistro retorna error si correo ya registrado (1 ms)

Test Suites: 1 passed, 1 total
Tests:      35 passed, 35 total
Snapshots:  0 total
Time:       1.129 s, estimated 2 s
Run all test suites matching __tests__\ UsuariosServices.test.js\1.
dstevengmz@dstevengmz:~/Escritorio/Clinestetica/Proyecto-Sena-Clinica-Estetica-Frontend-Backend/Backends
```

	Centro de Teleinformática y Producción Industrial - Regional Cauca			Página 326 de 326
Número de Documento:	FS-DOC Formato Manual de Usuario	Fecha de Creación: 1/02/2025	Elaborado por: Instructores área Software	
Nombre del Documento:	Formato para la construcción del Manual de Usuario			