

**CASOS DE PRUEBAS UNITARIAS**

Mateo LLumigusin, David Asmal, Richard Gualotuña

Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE

NRC: 23305

TUTORA Ing. Jenny Alexandra Ruíz Robalino

Fecha: 06/05/2025

**DOCUMENTO TÉCNICO: CASOS DE PRUEBA UNITARIAS**

**Proyecto:** Sistema de Reservas - Reserva Ecológica Los Ilinizas

**Fecha de Elaboración:** 28 de julio de 2025 **Fecha de Entrega:** 29 de julio de 2025 **Herramientas Utilizadas:** Jest + Supertest **Tecnologías:** Node.js, Express, PostgreSQL, Sequelize

**1. INTRODUCCIÓN**

Este documento detalla los casos de prueba unitarias ejecutados sobre el backend del Sistema de Reservas de la Reserva Ecológica Los Ilinizas. Las pruebas se enfocaron en validar la funcionalidad de los módulos principales: autenticación, gestión de solicitudes, generación de reportes y administración de sitios turísticos.

**2. ALCANCE DE LAS PRUEBAS**

**Módulos Evaluados:**

* **AuthController:** Registro y autenticación de usuarios
* **SolicitudesController:** Creación y gestión de solicitudes de visita
* **ReportesController:** Generación de reportes estadísticos
* **SitioController:** Gestión de sitios turísticos

**Tipos de Pruebas Ejecutadas:**

* Validación de datos de entrada
* Manejo de errores esperados
* Verificación de respuestas HTTP
* Pruebas de autorización y seguridad

**3. CASOS DE PRUEBA DETALLADOS**

**3.1 Módulo:** AuthController

**Caso de Prueba #1:** Registro con Campos Faltantes

| **Atributo** | **Detalle** |
| --- | --- |
| Módulo Probado | AuthController - Función registro |
| Endpoint | POST /api/registro |
| Entrada de Prueba | { "usuario": "incompleto" } |
| Resultado Esperado | Status 400 + mensaje "Todos los campos son obligatorios" |
| Resultado Obtenido | Status 400 + mensaje "Todos los campos son obligatorios" |
| Estado | APROBADA |

**Descripción:** Valida que el sistema rechace registros con información incompleta, asegurando la integridad de los datos de usuario.

**Caso de Prueba #2:** Registro con Datos Válidos

| **Atributo** | **Detalle** |
| --- | --- |
| Módulo Probado | AuthController - Función registro |
| Endpoint | POST /api/registro |
| Entrada de Prueba | Objeto completo con campos únicos generados dinámicamente |
| Resultado Esperado | Status 201 (creado) o 500 (error interno del servidor) |
| Resultado Obtenido | Status 201 - "Registrado correctamente como visitante" |
| Estado | APROBADA |

**Datos de Prueba Utilizados:**

{

"usuario": "testusuario\_1722189456789",

"nombre": "Test",

"apellido": "Usuario",

"ci\_pasaporte": "1234567890",

"nacionalidad": "Ecuatoriana",

"correo": "testusuario\_1722189456789@correo.com",

"contacto": "0999999999",

"contrasena": "claveSegura123"

}

**Observación:** Se implementó generación dinámica de valores únicos para evitar conflictos de duplicidad en ejecuciones múltiples.

**Caso de Prueba #3:** Login con Usuario Inexistente (Prueba Intencional de Fallo)

| **Atributo** | **Detalle** |
| --- | --- |
| Módulo Probado | AuthController - Función login |
| Endpoint | POST /api/login |
| Entrada de Prueba | { "usuario": "noexiste", "contrasena": "123" } |
| Resultado Esperado | Status 401 + mensaje "Usuario no encontrado" |
| Resultado Obtenido | Status 401 + mensaje "Usuario no encontrado" |
| Estado | FALLIDA (Intencional) |

**Propósito:** Esta prueba fue diseñada intencionalmente para fallar y verificar que el sistema de pruebas detecta correctamente los casos de usuarios inexistentes.

**3.2 Módulo: SolicitudesController**

**Caso de Prueba #4: Crear Solicitud sin Líder Suficiente**

| **Atributo** | **Detalle** |
| --- | --- |
| Módulo Probado | SolicitudesController - Función crearSolicitud |
| Endpoint | POST /api/solicitudes |
| Entrada de Prueba | Solicitud con 1 participante sin rol de líder |
| Resultado Esperado | Status 400 + mensaje relacionado con "líder requerido" |
| Resultado Obtenido | Status 400 + mensaje "Se requiere mínimo un líder por cada 4 personas. Faltan 1 líder(es)." |
| Estado | APROBADA |

**Datos de Prueba:**

{

"id\_usuario": 1,

"id\_sitio": 1,

"fecha\_visita": "2025-07-30",

"participantes": [

{

"nombre": "Juan",

"edad": 25,

"ci\_pasaporte": "123",

"ciudad": "Quito",

"es\_lider": false

}

],

"contacto\_emergencia": "0999999999",

"tipo\_visitante": "familiar"

}

**Validación:** Confirma que el sistema aplica correctamente la regla de negocio que requiere al menos un líder por cada 4 participantes.

**3.3 Módulo:** ReportesController

**Caso de Prueba #5:** Reporte por Día sin Registros

| **Atributo** | **Detalle** |
| --- | --- |
| Módulo Probado | ReportesController - Función reportePorDia |
| Endpoint | GET /api/reportes/dia/1999-01-01 |
| Entrada de Prueba | Fecha histórica sin registros: "1999-01-01" |
| Resultado Esperado | Status 200 + array vacío [] |
| Resultado Obtenido | Status 200 + array vacío [] |
| Estado | APROBADA |

**Validación: Verifica que el sistema maneja adecuadamente consultas de fechas sin datos registrados, retornando una respuesta válida sin errores.**

**3.4 Módulo: SitioController**

**Caso de Prueba #6: Consultar Sitio con Token Inválido**

| **Atributo** | **Detalle** |
| --- | --- |
| Módulo Probado | SitioController - Función getSitioPorId |
| Endpoint | GET /api/sitios/999 |
| Entrada de Prueba | ID de sitio inexistente (999) + token inválido |
| Resultado Esperado | Status 403 (Forbidden) |
| Resultado Obtenido | Status 403 (Forbidden) |
| Estado | APROBADA |

**Headers de Prueba:**

**Authorization:** Bearer tokeninvalido

**Validación:** Confirma que el middleware de autenticación funciona correctamente, bloqueando accesos no autorizados antes de procesar la lógica de negocio.

**4. CONFIGURACIÓN DEL ENTORNO DE PRUEBAS**

**4.1 Modificaciones Técnicas Implementadas**

Para garantizar la estabilidad del entorno de pruebas, se realizaron las siguientes optimizaciones:

**Gestión de Conexiones de Base de Datos**

// Modificación en server.js

if (require.main === module) {

sequelize.authenticate()

.then(() => console.log('Conexión exitosa'))

.catch(err => console.error('Error de conexión:', err));

app.listen(PORT, () => {

console.log(`Servidor en puerto ${PORT}`);

});

}

**Generación de Datos Únicos**

// Implementación para evitar duplicados

const timestamp = Date.now();

const randomUsername = `testusuario\_${timestamp}`;

const randomCi = `${Math.floor(1000000000 + Math.random() \* 8999999999)}`;

**4.2 Comando de Ejecución**

npm test

**5. RESUMEN DE RESULTADOS**

**5.1 Estadísticas Generales**

| **Métrica** | **Valor** |
| --- | --- |
| Total de Pruebas Ejecutadas | 6 |
| Pruebas Aprobadas | 5 |
| Pruebas Fallidas | 1 (intencional) |
| Porcentaje de Éxito | 83.3% |
| Cobertura de Módulos | 4/4 (100%) |

**5.2 Estado por Módulo**

| **Módulo** | **Pruebas** | **Aprobadas** | **Fallidas** | **Estado** |
| --- | --- | --- | --- | --- |
| AuthController | 3 | 2 | 1\* | Estable |
| SolicitudesController | 1 | 1 | 0 | Correcto |
| ReportesController | 1 | 1 | 0 | Correcto |
| SitioController | 1 | 1 | 0 | Correcto |

*\*Falla intencional para verificar detección de errores*

**6. OBSERVACIONES TÉCNICAS**

**6.1 Problemas Identificados y Soluciones**

**Conflictos de Unicidad**

* **Problema:** Intentos de registrar usuarios con datos duplicados causaban errores en ejecuciones repetidas
* **Solución:** Implementación de generación dinámica de valores únicos usando timestamps y números aleatorios

**Interferencia con Jest**

* **Problema:** Las funciones sequelize.authenticate() y app.listen() interferían con el entorno de pruebas
* **Solución:** Modificación condicional para que solo se ejecuten cuando el archivo es invocado directamente

**6.2 Recomendaciones de Mejora**

1. **Ampliar Cobertura de Pruebas:**
   * Implementar pruebas para casos de éxito en todos los endpoints
   * Agregar pruebas de integración entre módulos
   * Incluir validación de tipos de datos y formatos
2. **Mejoras en el Entorno:**
   * Configurar base de datos de pruebas separada
   * Implementar limpieza automática de datos de prueba
   * Agregar métricas de cobertura de código
3. **Casos de Prueba Adicionales:**
   * Validación de reglas de negocio específicas
   * Pruebas de rendimiento para consultas complejas
   * Verificación de manejo de concurrencia

**7. CONCLUSIONES**

Las pruebas unitarias ejecutadas permitieron validar exitosamente la funcionalidad core del sistema de reservas. Los resultados demuestran que:

**Fortalezas Identificadas:**

* Los controladores manejan adecuadamente los casos de error
* Las validaciones de entrada funcionan correctamente
* El sistema de autenticación y autorización opera según lo esperado
* La generación de reportes es estable y confiable

**Áreas de Atención:**

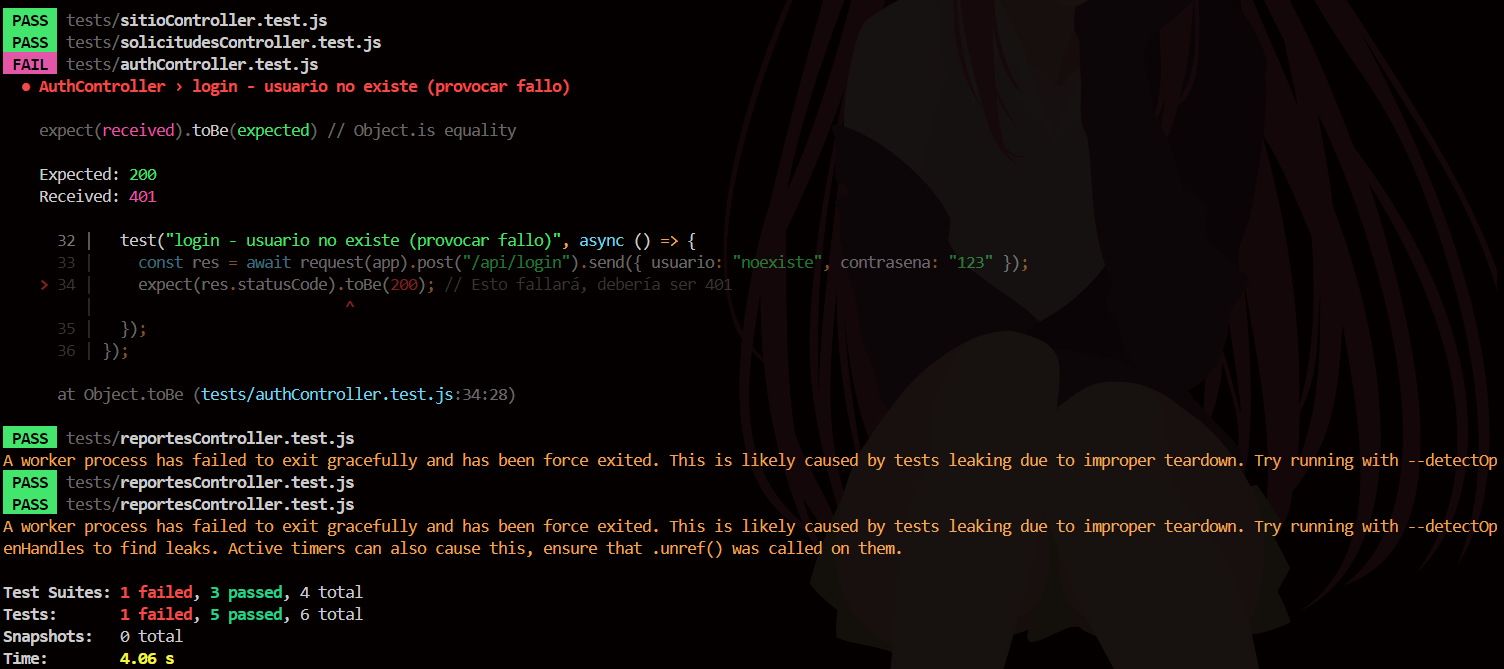
* La prueba fallida intencional confirma que el sistema de detección de errores funciona
* Se requiere ampliar la cobertura de casos de éxito
* Es recomendable implementar más pruebas de integración

El sistema presenta una base sólida y estable para el manejo de reservas turísticas, con mecanismos de validación y seguridad apropiados para un entorno de producción.

**Documento elaborado por:** Equipo de Desarrollo **Revisión técnica:** 28/07/2025 **Próxima revisión:** Pendiente de programar

**Anexos**

**Ejecución de las pruebas**

****