# Registro de errores encontrados

## Error 1: Conexión a la base de datos no cerrada

**Descripción:**
Jest detectó un `open handle`, lo que indica que la conexión a MySQL no se estaba cerrando correctamente.

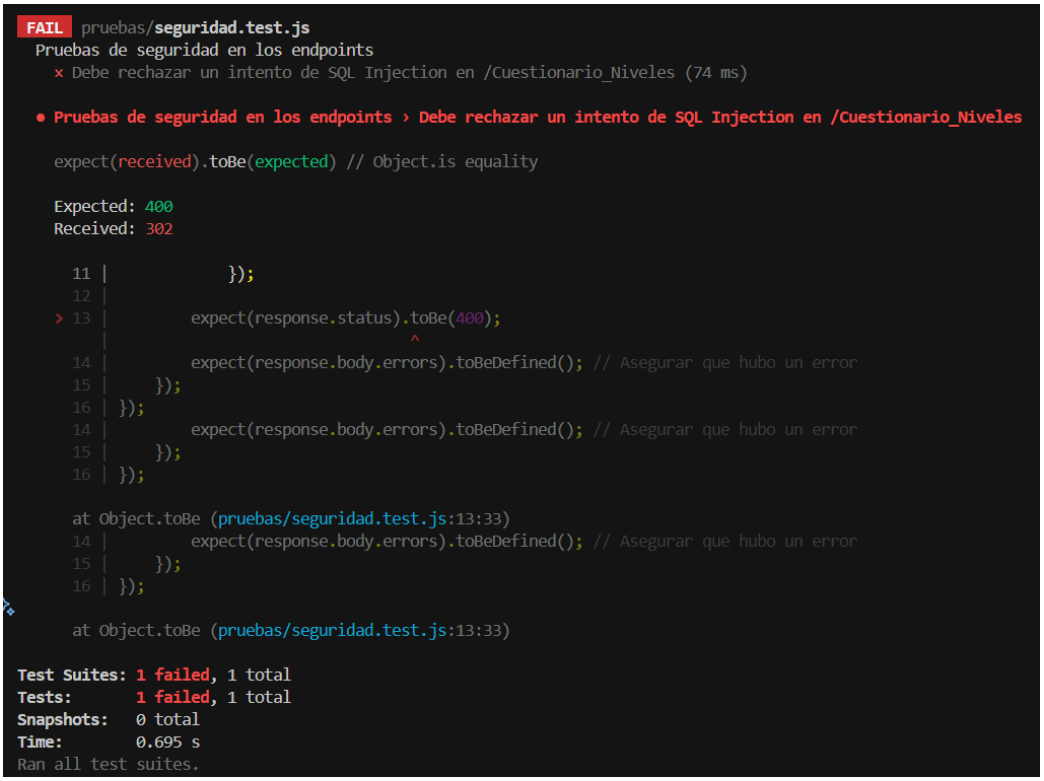
```
Jest has detected the following 1 open handle potentially keeping Jest from exiting:

  ● TCPWRAP

    10 | });
    11 |
  > 12 | pool.getConnection((err, connection) => {
       |      ^
    13 |     if (err) {
    14 |         console.error('Error de conexión a la BD:', err.code, err.sqlMessage);
    at new BaseConnection (node_modules/mysql2/lib/base/connection.js:49:27)
    at new BasePoolConnection (node_modules/mysql2/lib/base/pool_connection.js:7:5)
    at new PoolConnection (node_modules/mysql2/lib/pool_connection.js:5:1)
    at Pool.getConnection (node_modules/mysql2/lib/base/pool.js:49:20)
    at Object.getConnection (src/config/database.js:12:6)
    at Object.require (src/app/routes/rutas.js:3:14)
    at Object.require (src/config/server.js:53:20)
    at Object.require (pruebas/api.test.js:2:13)
```

**Solución aplicada:**
Se cerró la conexión de MySQL después de cada prueba con:

```
afterAll(async () => {
    await pool.end();
});
```

## Error 2: Conexión a la base de datos no cerrada

**Descripción:** Solo estás validando si nombre no está vacío, pero no estás evitando inyección SQL.




**Solución aplicada:** Agregar validaciones más estrictas, usando una expresión regular para evitar caracteres maliciosos:

```
// Procesar evaluación
router.post('/Cuestionario_Niveles', [
    body('nombre')
        .notEmpty().withMessage('El nombre es requerido')
        .matches(/^[a-zA-Z\s]+$/).withMessage('El nombre solo debe contener letras y espacios'),
    body('episodio').isInt({ min: 1 }).withMessage('Episodio debe ser un número válido')
], async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
    }
```

# Registro de pruebas completadas

## Pruebas de validación de datos con Jest + Supertest

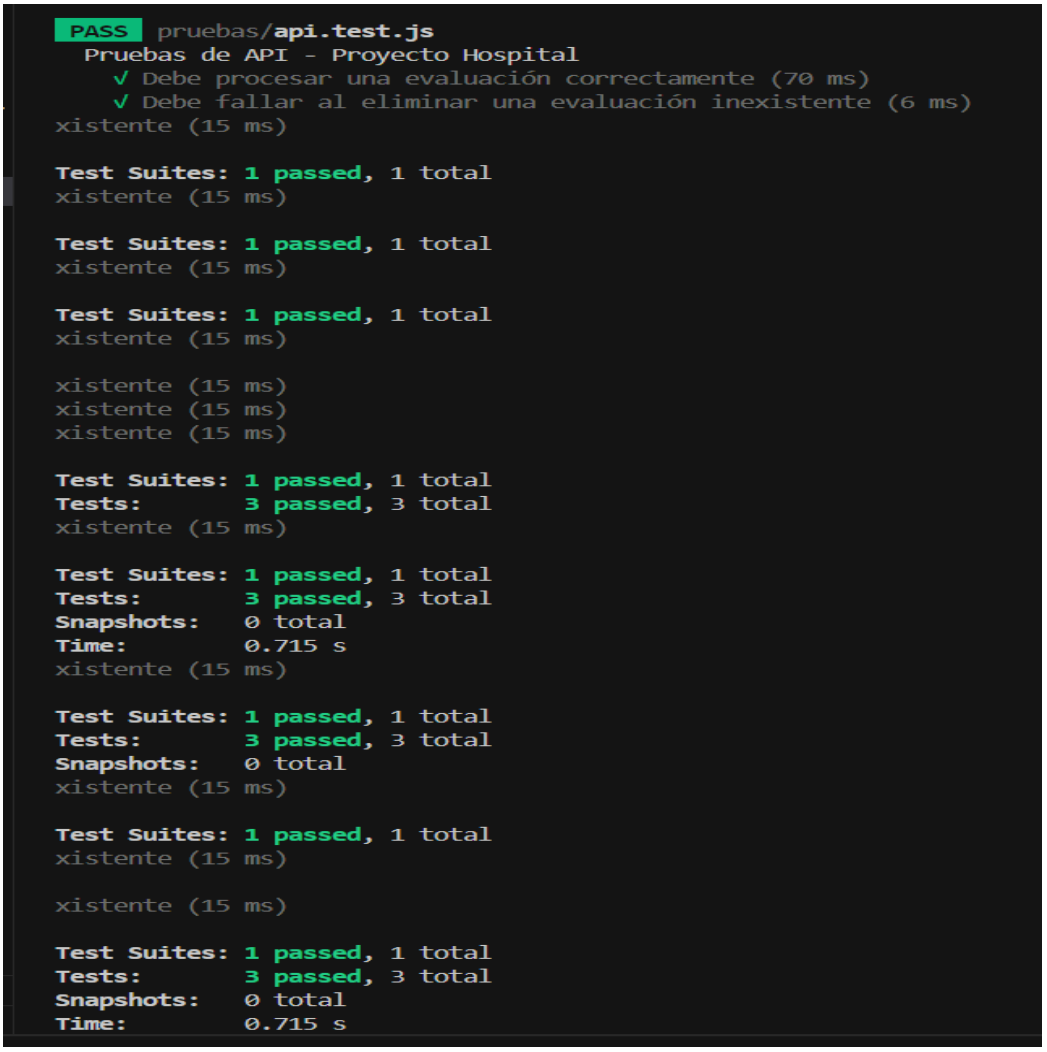
### Prueba 1: Pruebas API - Proyecto Hospital

**Descripción:** Prueba 1: Verifica que el cuestionario se envíe correctamente y la API redireccione (HTTP 302). Prueba 2: Intenta eliminar una evaluación con un ID inexistente y verifica que devuelva un error (HTTP 404). Prueba 3: Inserta una evaluación en la BD, luego la elimina y verifica que fue eliminada correctamente (HTTP 200).

`POST /Cuestionario_Niveles` → Envía datos de evaluación y los guarda en la BD.

`POST /eliminar-evaluacion/:id` → Elimina una evaluación según su ID.

**Salida en terminal**:

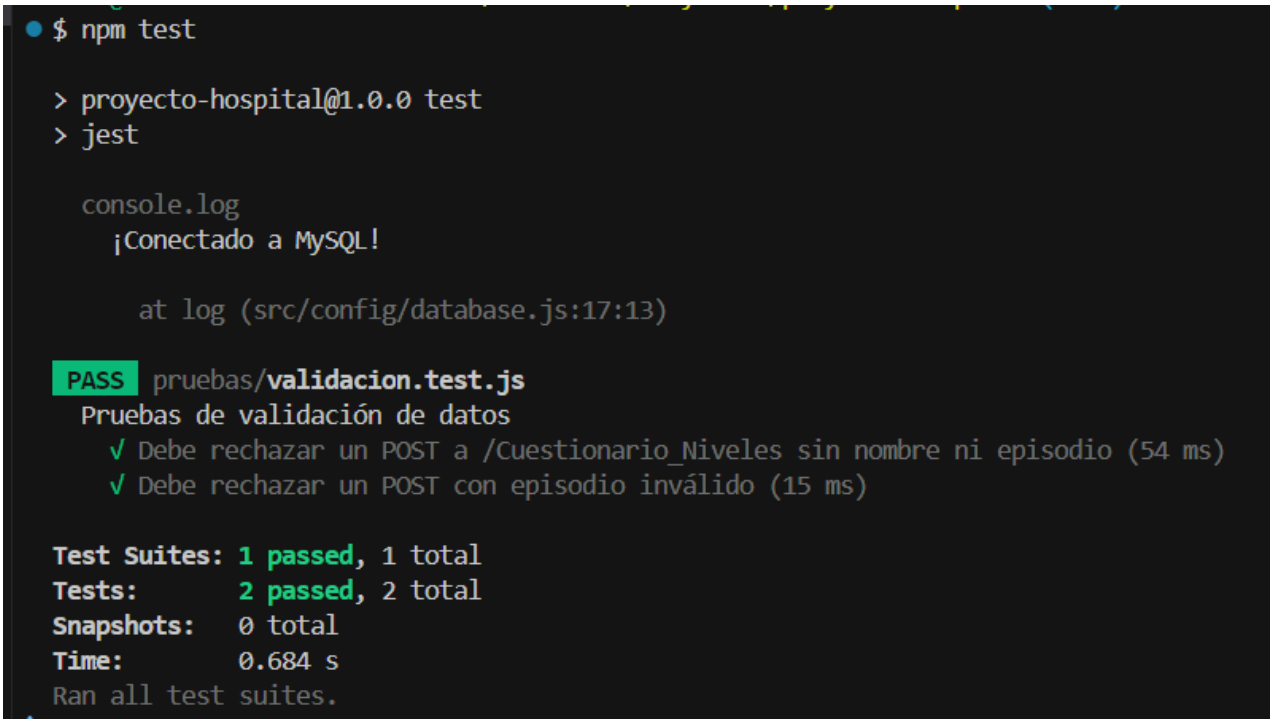


## Prueba 2: Enviar datos incompletos a /Cuestionario_Niveles

**Caso:** No enviar `nombre` ni `episodio`.

**Esperado:** La API debe responder con `400 Bad Request` y un mensaje de error.

**Salida en terminal**:



# Pruebas de Seguridad en los Endpoints

## Prueba 1: Intento de Inyección SQL en /Cuestionario_Niveles

**Caso:** Enviar un nombre con una consulta SQL maliciosa.

**Esperado:** El sistema debe rechazar la solicitud y no debe ejecutar la consulta maliciosa.

**Salida en terminal**:

```
PASS pruebas/seguridad.test.js
  Pruebas de seguridad en los endpoints
    √ Debe rechazar un intento de SQL Injection en /Cuestionario_Niveles (50 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.65 s, estimated 1 s
Ran all test suites.
```

## Prueba 2: Prueba 2: Intento de XSS (Cross-Site Scripting) en /Cuestionario_Niveles

**Caso:** Enviar un nombre con un script malicioso.

**Esperado:** El sistema debe rechazar la solicitud y no permitir la ejecución del script.

**Salida en terminal**:

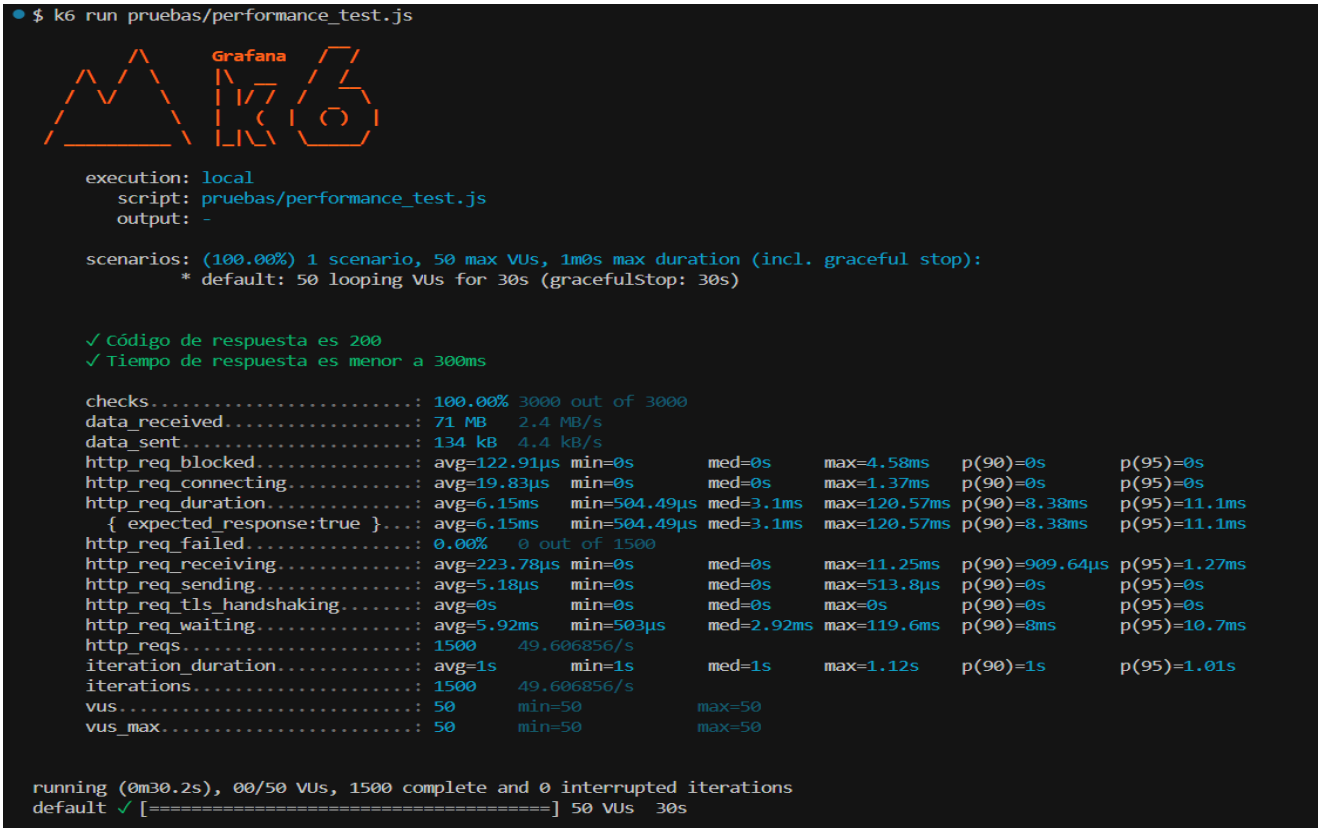
```
PASS pruebas/seguridad2.test.js
  √ Debe rechazar un intento de XSS en /Cuestionario_Niveles (52 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.674 s
Ran all test suites.
```

## Prueba de Rendimiento - API "Proyecto Hospital"

**Descripción:** Se utilizó la herramienta k6 para simular 30 usuarios virtuales (VUs) enviando solicitudes concurrentes durante 30 segundos, generando un alto volumen de peticiones en un corto periodo de tiempo.

**Objetivo:** Se realizó una prueba de rendimiento en el endpoint `/Historial` para evaluar su tiempo de respuesta bajo carga, el objetivo fue verificar que la API mantuviera tiempos de respuesta óptimos con múltiples usuarios concurrentes.

**Resultados Obtenidos:**




**Datos adicionales:**

- No hubo fallos en las solicitudes (http_req_failed: 0/1500).
- Se recibieron 71 MB de datos.
- El tiempo de espera (http_req_waiting) fue 9.92 ms en promedio.

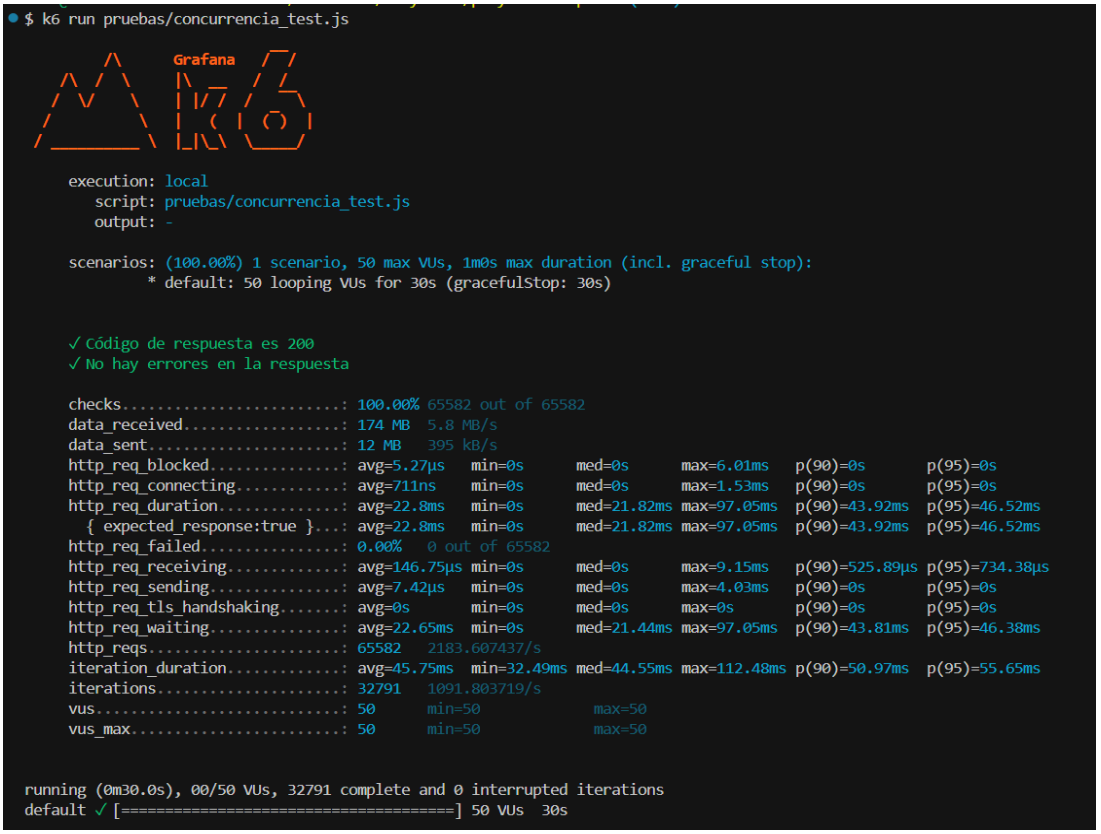
El endpoint `/Historial` respondió eficientemente, con un tiempo medio de 6.15 ms y un máximo de 120.57 ms, lo que está por debajo del umbral de 300 ms. No se registraron fallos en las solicitudes.

# Prueba de Concurrencia - API "Proyecto Hospital"

**Descripción:** Se utilizó la herramienta k6 para simular 50 usuarios virtuales (VUs) enviando solicitudes concurrentes durante 30 segundos, generando un alto volumen de peticiones en un corto periodo de tiempo.

**Objetivo:** Evaluar el comportamiento del sistema cuando múltiples usuarios realizan solicitudes simultáneamente para identificar posibles errores de concurrencia, tiempos de respuesta y estabilidad bajo carga.

**Resultados Obtenidos:**




Métricas clave:

- **Total de solicitudes procesadas:** 32,791
- **Código de respuesta esperado (200 OK):**100% (65,582 exitosas)
- **Errores de respuesta:** 0
- **Tiempo promedio de respuesta:** 22.8ms
- **Tiempo máximo de respuesta:** 97.05ms
- **Tasa de solicitudes por segundo:** 1091.8 req/s

La API manejó correctamente la concurrencia sin errores ni fallos en las solicitudes.
El tiempo promedio de respuesta es bajo, lo que indica que el sistema tiene un buen rendimiento bajo carga, no hubo solicitudes fallidas, lo que significa que la API respondió correctamente a cada petición sin pérdida de datos ni fallos de conexión.