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.

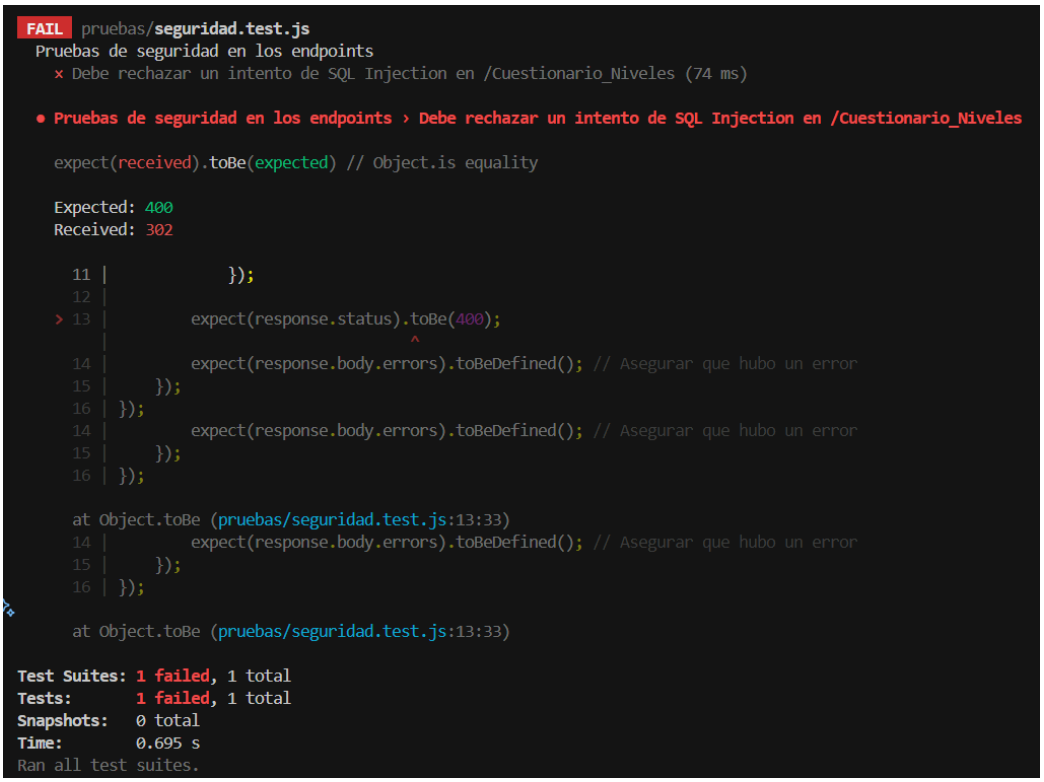
```
Jest has detected the following 1 open handle potentially keeping Jest from exiting:

  ● 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.



```
FAIL pruebas/seguridad.test.js
  Pruebas de seguridad en los endpoints
    × Debe rechazar un intento de SQL Injection en /Cuestionario_Niveles (74 ms)

  ● Pruebas de seguridad en los endpoints › Debe rechazar un intento de SQL Injection en /Cuestionario_Niveles

    expect(received).toBe(expected) // Object.is equality

    Expected: 400
    Received: 302

      11 |             });
      12 |
    > 13 |         expect(response.status).toBe(400);
         |                                 ^
      14 |         expect(response.body.errors).toBeDefined(); // Asegurar que hubo un error
      15 |     });
      16 | });
      14 |         expect(response.body.errors).toBeDefined(); // Asegurar que hubo un error
      15 |     });
      16 | });

      at Object.toBe (pruebas/seguridad.test.js:13:33)
      14 |         expect(response.body.errors).toBeDefined(); // Asegurar que hubo un error
      15 |     });
      16 | });

      at Object.toBe (pruebas/seguridad.test.js:13:33)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        0.695 s
Ran all test suites.
```

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

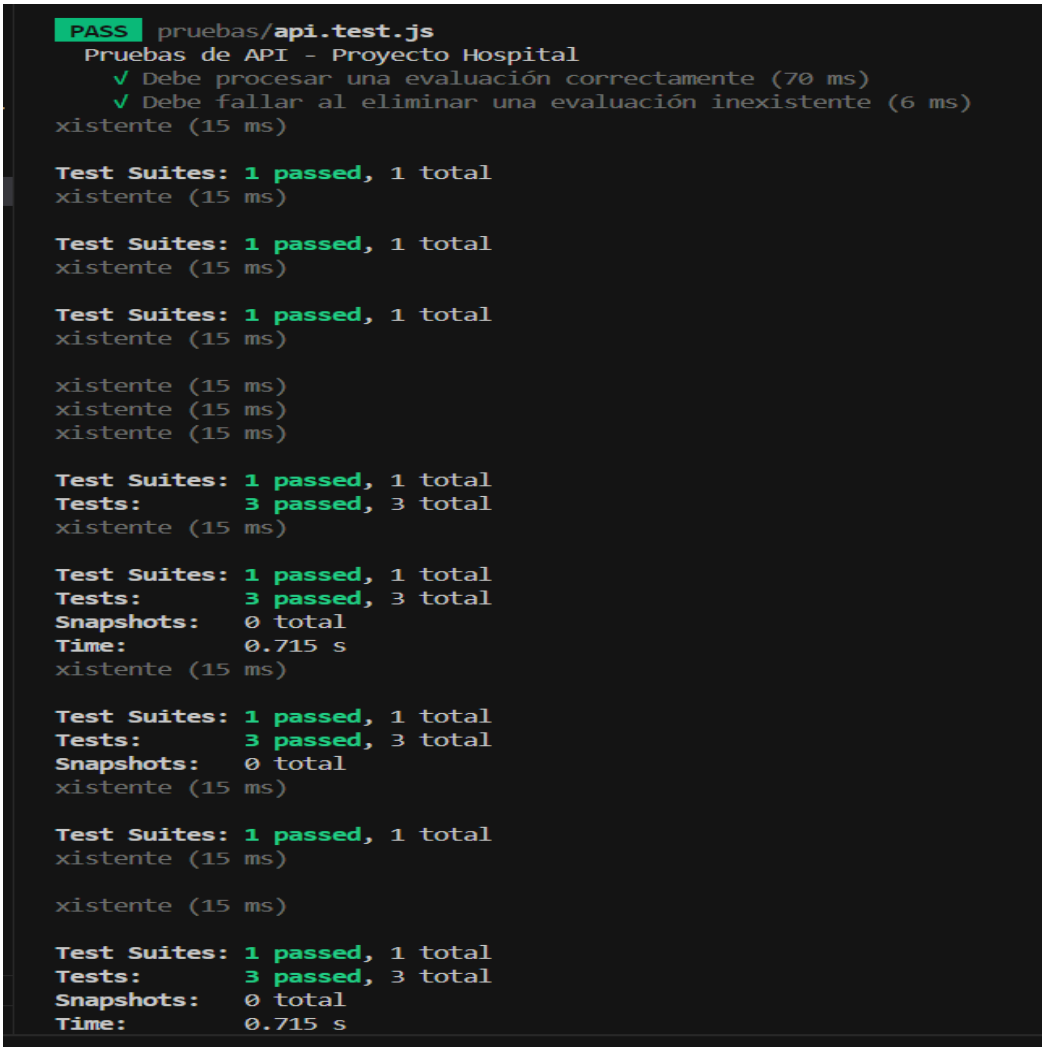
Prueba 1: Pruebas API - Proyecto Hospital

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:

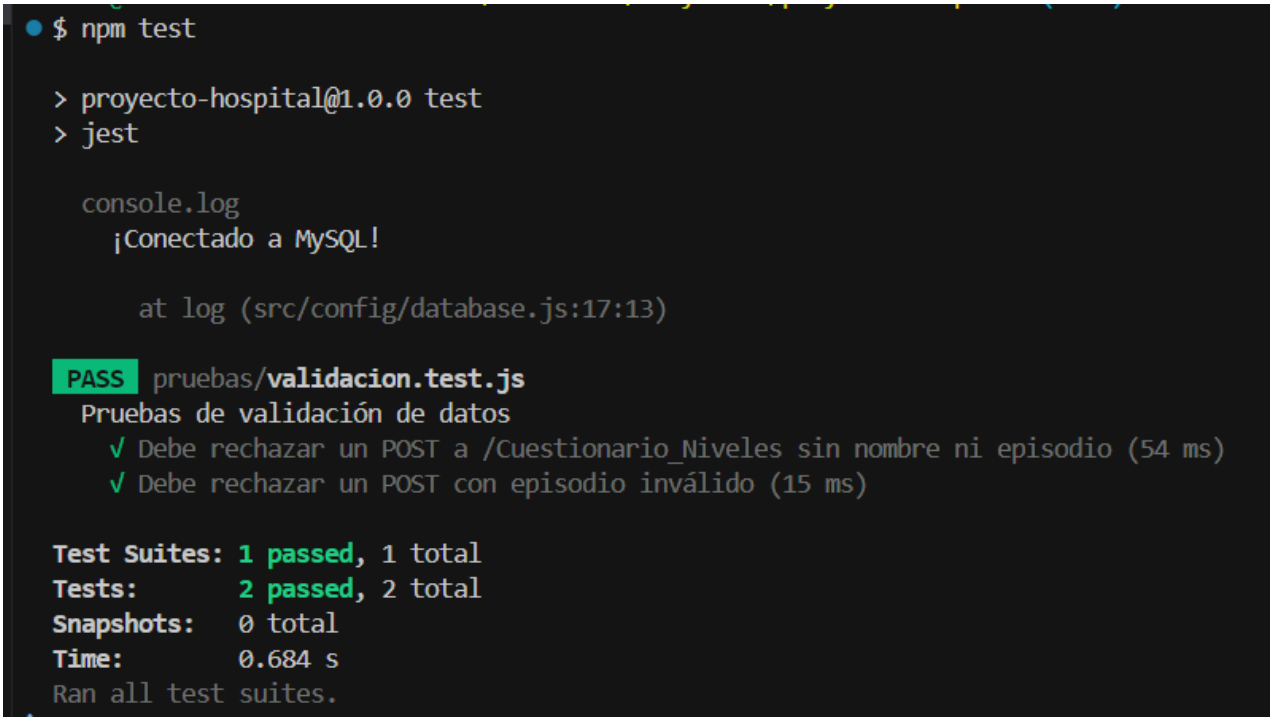


Prueba 2: Enviar datos incompletos a /Cuestionario_Niveles

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:

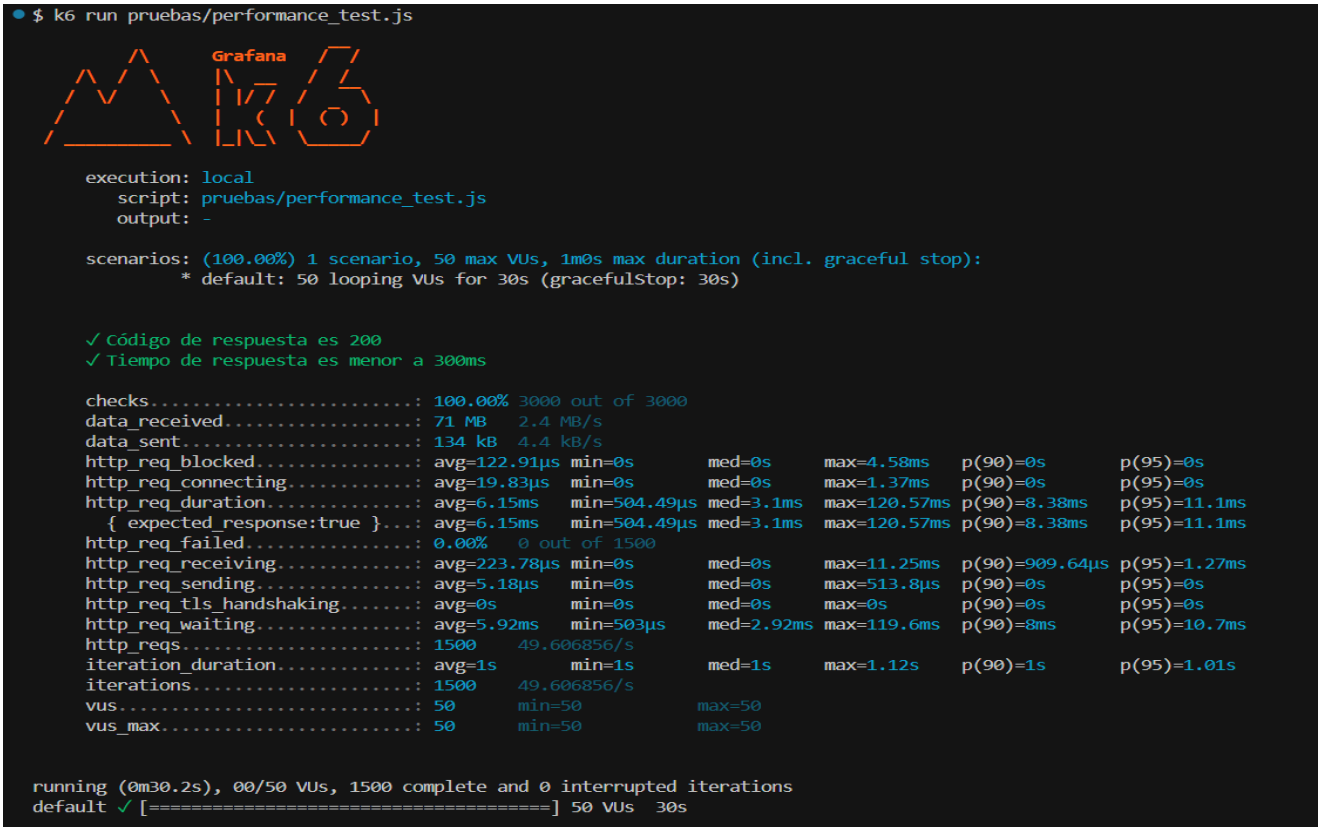
```
PASS pruebas/seguridad2.test.js
  √ Debe rechazar un intento de XSS en /Cuestionario_Niveles (52 ms)

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 50 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.

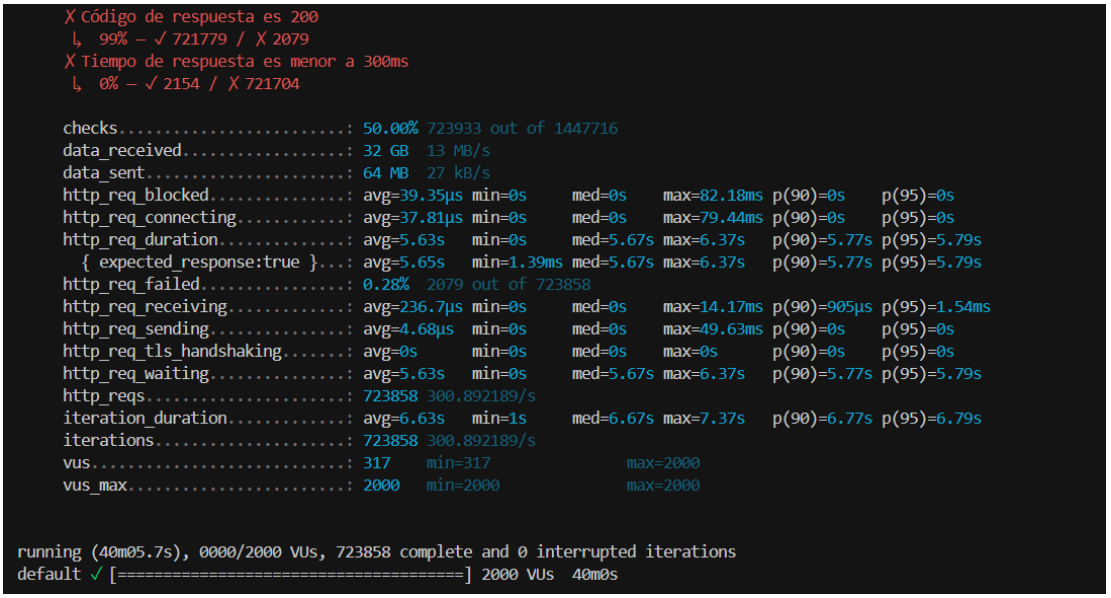
El endpoint /Historial respondió eficientemente, con un tiempo medio de 6.15 ms y un máximo de 120.57 ms, lo que está por debajo del umbral de 300 ms. No se registraron fallos en las solicitudes.

Prueba de Rendimiento - API "Proyecto Hospital" - Nivel Extremo

Descripción: Se ejecutó una prueba de rendimiento utilizando la herramienta K6, simulando 2000 usuarios virtuales (VUs) realizando solicitudes simultáneamente durante 40 minutos continuos, el objetivo fue ejercer máxima presión sobre el endpoint /Cuestionario_Niveles para evaluar su comportamiento bajo carga crítica.

Objetivo: Medir el desempeño y estabilidad del sistema en condiciones extremas, identificando tiempos de respuesta promedio, porcentaje de errores, y resistencia del backend ante una alta concurrencia y tráfico sostenido.

Resultados Obtenidos:



Métricas clave:

- **Total de solicitudes realizadas:** 723,858
- **Solicitudes exitosas (status 200):** 721,779
- **Tasa de solicitudes por segundo:** 300.89 req/s
- **Errores detectados:** 2,079 solicitudes fallidas (~0.28%)
- **Tiempo promedio de respuesta:** 5.63 segundos
- **Percentil 90 (p90):** 5.77 segundos
- **Percentil 95 (p95):** 5.79 segundos

Conclusión: Esta prueba representa un escenario de uso extremo. El sistema demostró una gran capacidad de manejo de carga en términos de estabilidad, ya que solo el 0.28% de las solicitudes fallaron. Sin embargo, se identificó un cuello de botella en el rendimiento:

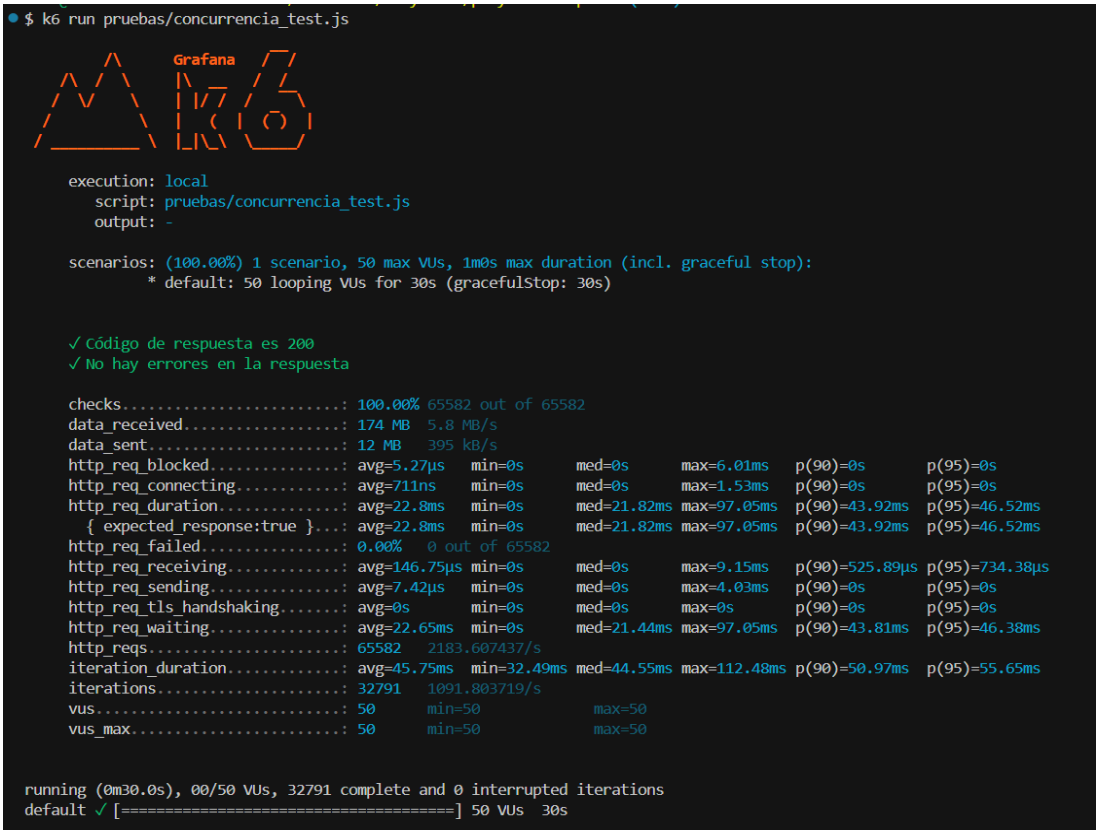
- El 99.7% de las solicitudes no cumplieron el objetivo de respuesta menor a 300ms, con un promedio de 5.63 segundos.
- Esto sugiere que, aunque el sistema no colapsó, sí presenta lentitud severa bajo alta concurrencia.

Prueba de Concurrencia - API "Proyecto Hospital" - Nivel Bajo

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

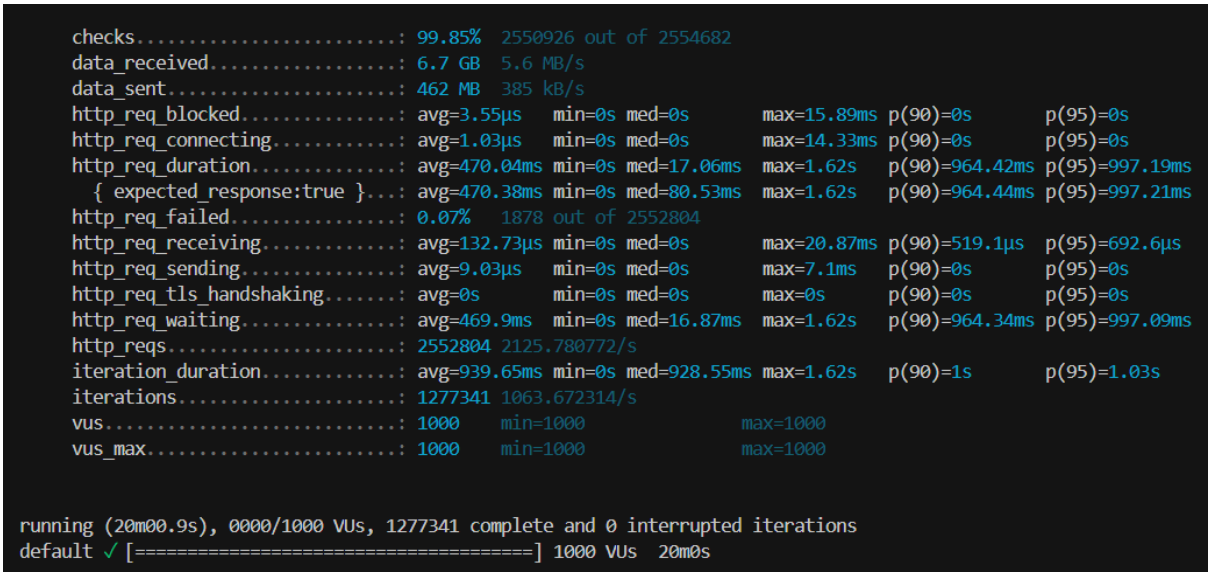
La API manejó correctamente la concurrencia sin errores ni fallos en las solicitudes.
El tiempo promedio de respuesta es bajo, lo que indica que el sistema tiene un buen rendimiento bajo carga, no hubo solicitudes fallidas, lo que significa que la API respondió correctamente a cada petición sin pérdida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

Prueba de Concurrencia - API "Proyecto Hospital" - Nivel Extremo

Descripción: Esta prueba simula una carga de 1,000 usuarios concurrentes interactuando constantemente con el formulario principal del sistema hospitalario durante 20 minutos.

Objetivo: Evaluar la capacidad de respuesta y estabilidad del formulario bajo una carga de alto tráfico, asegurando que el backend y la base de datos soporten múltiples envíos simultáneos sin fallos críticos.

Resultados Obtenidos:



```
     checks.........................: 99.85%  2550926 out of 2554682
     data_received..................: 6.7 GB  5.6 MB/s
     data_sent......................: 462 MB  385 kB/s
     http_req_blocked...............: avg=3.55µs   min=0s med=0s       max=15.89ms p(90)=0s       p(95)=0s
     http_req_connecting............: avg=1.03µs   min=0s med=0s       max=14.33ms p(90)=0s       p(95)=0s
     http_req_duration..............: avg=470.04ms min=0s med=17.06ms  max=1.62s   p(90)=964.42ms p(95)=997.19ms
       { expected_response:true }...: avg=470.38ms min=0s med=80.53ms  max=1.62s   p(90)=964.44ms p(95)=997.21ms
     http_req_failed................: 0.07%   1878 out of 2552804
     http_req_receiving.............: avg=132.73µs min=0s med=0s       max=20.87ms p(90)=519.1µs  p(95)=692.6µs
     http_req_sending...............: avg=9.03µs   min=0s med=0s       max=7.1ms   p(90)=0s       p(95)=0s
     http_req_tls_handshaking.......: avg=0s       min=0s med=0s       max=0s      p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=469.9ms  min=0s med=16.87ms  max=1.62s   p(90)=964.34ms p(95)=997.09ms
     http_reqs......................: 2552804 2125.780772/s
     iteration_duration.............: avg=939.65ms min=0s med=928.55ms max=1.62s   p(90)=1s       p(95)=1.03s
     iterations.....................: 1277341 1063.672314/s
     vus............................: 1000    min=1000              max=1000
     vus_max........................: 1000    min=1000              max=1000


running (20m00.9s), 0000/1000 VUs, 1277341 complete and 0 interrupted iterations
default ✓ [======================================] 1000 VUs  20m0s
```

Métricas clave:

- **Total de solicitudes realizadas:** 2,552,804
- **Solicitudes exitosas (status 200):** 1,275,463
- **Tasa de solicitudes por segundo:** 2,125 req/s
- **Errores detectados:** 1,878 (0.07%)
- **Tiempo promedio de respuesta:** 470 ms
- **Percentil 90 (p90):** 964 ms
- **Percentil 95 (p95):** 997 ms

Conclusión: La prueba de concurrencia con 1,000 usuarios virtuales durante 20 minutos demostró que el sistema es altamente estable y funcional bajo condiciones de alta carga simultánea. A pesar de un pequeño porcentaje de errores (0.07%), el sistema respondió correctamente en el 99.85% de los casos, lo cual es un rendimiento aceptable para aplicaciones en entornos reales. El tiempo promedio de respuesta fue de 470 ms, con los percentiles más exigentes (p90 y p95) por debajo del segundo, lo cual indica un comportamiento óptimo y consistente incluso en situaciones de estrés.

Análisis con OWASP ZAP

1. CSP: Falla al Definir Directiva sin Fallback

Riesgo: Medio
Confianza: Alta
Descripción: Se ha detectado que la política de seguridad de contenido (Content-Security-Policy) no define ciertas directivas que pueden representar un riesgo si no se proporciona un valor por defecto (fallback) con default-src.

Recomendación: Asegúrate de que todas las directivas necesarias estén definidas y que default-src cubra adecuadamente los recursos por defecto en caso de que falten otras.

2. CSP: script-src 'unsafe-inline'

Riesgo: Medio
Confianza: Alta
Descripción: La directiva script-src permite 'unsafe-inline', lo que habilita la ejecución de scripts embebidos. Esto facilita ataques XSS.

Recomendación: Evita usar 'unsafe-inline'. Usa hashes o nonces para scripts embebidos seguros.

3. CSP: style-src 'unsafe-inline'

Riesgo: Medio
Confianza: Alta
Descripción: Similar al punto anterior, el uso de 'unsafe-inline' en estilos permite ataques de inyección de CSS.

Recomendación: Evita 'unsafe-inline' en style-src. Usa archivos de estilo externos o hashes.

4. Configuración Incorrecta Cross-Domain

Riesgo: Medio
Confianza: Media
Descripción: Se ha detectado que el servidor permite peticiones desde cualquier dominio (Access-Control-Allow-Origin: *), lo cual podría permitir que otras páginas web interactúen libremente con tu API.

Recomendación: Restringe los orígenes permitidos con CORS solo a los dominios necesarios.

5. Ausencia de Tokens Anti-CSRF

Riesgo: Medio
Confianza: Baja
Descripción: No se detectó el uso de tokens CSRF en los formularios o peticiones sensibles, lo que deja tu app vulnerable a ataques CSRF.

Recomendación: Implementa tokens CSRF para cada sesión del usuario y verifica su validez en el servidor.

6. Inclusión de Archivos JavaScript de otros Dominios

Riesgo: Bajo
Confianza: Media
Descripción: La aplicación carga archivos JavaScript desde otros dominios, lo que podría ser riesgoso si esos recursos son comprometidos.

Recomendación: Revisa los archivos externos y asegúrate de que provienen de fuentes confiables. Usa integridad con SRI (subresource integrity) cuando sea posible.

7. CSP: Header & Meta

Riesgo: Informativo
Confianza: Alta
Descripción: Se detectó que la CSP está definida tanto en cabeceras HTTP como en etiquetas meta dentro del HTML.

Recomendación: Define CSP solamente en el encabezado HTTP para evitar conflictos y mantener una política unificada.

Solución a las Alertas Detectadas por OWASP ZAP

1. CSP: Falla al Definir Directiva sin Fallback

- **Estado:** Solucionado
- **Acción tomada:** Se estableció correctamente la directiva default-src: 'self' como fallback en la cabecera Content-Security-Policy, lo que garantiza una política por defecto cuando otras directivas no están presentes.

2. CSP: script-src 'unsafe-inline'

- **Estado:** Falso positivo (ZAP no interpreta nonces dinámicos)
- **Explicación:** En el código se utiliza Helmet con nonce generado dinámicamente por request. Esto reemplaza la necesidad de 'unsafe-inline' al permitir solo scripts con un nonce específico.
- **Conclusión:** La alerta se debe a que ZAP no interpreta funciones dinámicas dentro de las directivas CSP. Sin embargo, los navegadores aplican correctamente la protección con nonces. Se considera mitigada.

3. CSP: style-src 'unsafe-inline'

- **Estado:** Falso positivo (idéntico al punto anterior)
- **Explicación:** La directiva style-src también utiliza nonces generados dinámicamente, por lo que 'unsafe-inline' no está presente en la cabecera real enviada al navegador.

4. Configuración Incorrecta Cross-Domain

- **Estado:** Solucionado
- **Acción tomada:** Se configuró el middleware cors para aceptar solo orígenes definidos explícitamente desde variables de entorno:
- **Resultado:** Ya no se permite Access-Control-Allow-Origin: *, limitando las solicitudes a orígenes controlados.

5. Ausencia de Tokens Anti-CSRF

- **Estado:** Solucionado
- **Acción tomada:** Se implementó protección CSRF mediante el paquete csrf, generando un token único por sesión y validando en métodos sensibles (POST, PUT, DELETE). También se incluyeron cookies con atributos SameSite=Strict, HttpOnly y Secure para mitigar riesgos de robo de sesión.

6. Inclusión de Archivos JavaScript de otros Dominios

- **Estado:** Controlado
- **Acción tomada:** Se revisaron y permitieron únicamente fuentes confiables como cdn.jsdelivr.net y kit.fontawesome.com en la directiva script-src. Además, se podría considerar el uso futuro de **Subresource Integrity (SRI)** para archivos estáticos externos.

7. CSP: Header & Meta

- **Estado:** Solucionado
- **Acción tomada:** Se eliminó cualquier declaración CSP en etiquetas <meta> dentro del HTML. La política CSP ahora se define exclusivamente a través de la cabecera HTTP generada por Helmet.

Conclusión: Las alertas reportadas por ZAP fueron mitigadas adecuadamente o identificadas como falsos positivos causados por limitaciones de la herramienta al interpretar configuraciones dinámicas en CSP. Se mantiene un entorno seguro conforme a buenas prácticas de OWASP.