

Seguridad en funcionalidades de gestión de reservas en Restful Booker

Documento de Testing de Software

Versión 1.0.0

13/12/2025

Índice de Contenidos

1. Descripción general del Sistema	3
2. Descripción de funcionalidades	3
3. Planificación y enfoque de testing	4
3.1 Objetivo del Testing	
3.2 Alcance	
3.3 Enfoque de Testing	
4. Casos de Prueba	5
4.1 Vulnerabilidades SQLi	5
4.1.1 Detección de SQL Injection Boolean-based en GET /booking	
4.1.2 Error-Based SQLi en GET /booking	
4.1.3 SQLi Error Based Body — /booking	
4.1.4 SQLi Boolean-Based en Body	
4.1.5 SQL Injection Time-Based en POST Body	
4.2 Vulnerabilidades varias en validación y codificado de Input & Output	21
4.2.1 HTML Injection	
4.2.2 Stored XSS y Tag Injection	
4.3 Access control	25
4.3.1 Acceso a recurso protegido sin token	
4.3.2 Acceso a recurso protegido con token inválido	

1. Descripción general del Sistema

Restful-Booker es una API REST de ejemplo diseñada específicamente como un entorno de práctica (sandbox) para el aprendizaje y aplicación de pruebas de API (API Testing). Simula un sistema básico de reservas de hotel (*Bookings*), exponiendo los *endpoints* necesarios para realizar las operaciones fundamentales del protocolo CRUD (Crear, Leer, Actualizar y Eliminar reservas). Su arquitectura incluye un sistema de autenticación basado en *tokens* o *Basic Auth* requerido para las operaciones de modificación, y posee *endpoints* clave para la gestión de reservas y un *HealthCheck* (/ping).

2. Descripción de funcionalidades

Get Booking IDs

Devuelve los ids de todas las reservas existentes en la API. Opcionalmente puede recibir parámetros de filtrado como nombre de quién reserva y fechas de check in y check out.

Get Booking

Devuelve la información completa de una reserva basada en el id provisto.

Post Create Booking

Crea una nueva reserva en la API. Recibe un body en JSON o XML con los parámetros de la reserva. Ejemplo en JSON:

```
{ "firstname" : "Marco",
  "lastname" : "Polo",
  "totalprice" : 222,
  "depositpaid" : false,
  "bookingdates" : {
    "checkin" : "2018-01-01",
    "checkout" : "2019-01-01"
  },
  "additionalneeds" : "Breakfast" }
```

Delete Booking

Elimina una reserva previamente creada. Requiere un token de autenticación válido o un Basic Auth Header.

3. Planificación y enfoque de Testing

3.1 Objetivo del Testing

El objetivo de este ciclo de pruebas es evaluar la seguridad de la API Restful-Booker, enfocándose en la detección de vulnerabilidades comunes en APIs REST, tales como inyecciones SQL, fallas en la validación de entradas, problemas de codificación de salida y controles de acceso insuficientes.

El testing se orienta principalmente a identificar riesgos asociados al procesamiento de datos controlados por el usuario y a la protección de recursos sensibles.

3.2 Alcance

Las pruebas se realizaron sobre los siguientes endpoints expuestos por la API:

- GET /booking
- GET /booking/{id}
- POST /booking
- DELETE /booking/{id}
- Endpoint de autenticación (para pruebas de control de acceso)

Dentro del alcance se incluyen:

- Pruebas de SQL Injection (boolean-based, error-based y time-based).
- Pruebas de validación y sanitización de input
- Pruebas de codificación segura de output
- Pruebas de control de acceso y autorización

Fuera del alcance:

- Pruebas de performance y stress
- Pruebas de lógica de negocio compleja
- Pruebas de infraestructura o configuración del servidor

3.3 Enfoque de Testing

Se utilizó un enfoque de testing de seguridad dinámico (DAST), realizando pruebas manuales mediante la herramienta Postman.

Los casos de prueba fueron diseñados siguiendo:

- Buenas prácticas de OWASP (OWASP Top 10 y OWASP API Security Top 10)
- Técnicas clásicas de detección de vulnerabilidades en APIs
- Comparación de respuestas entre requests legítimos y maliciosos

3.4 Criterios de Ejecución

Para cada caso de prueba se evaluaron los siguientes criterios:

- Código de estado HTTP
- Contenido del cuerpo de la respuesta
- Presencia de mensajes de error internos
- Comportamientos anómalos ante payloads maliciosos
- Consistencia entre respuestas esperadas y obtenidas

3.5 Criterios de Aprobación / Rechazo

Un caso de prueba se considera:

- **Pass** cuando la API responde de forma segura y consistente ante el escenario probado.
- **Fail / Defectuoso** cuando se detecta una vulnerabilidad o comportamiento inseguro que podría ser explotado en un entorno real.

4. Casos de Prueba

4.1 Vulnerabilidades SQLi

Este bloque verifica que la API no presente vulnerabilidades a inyecciones SQL.

3.1.1 Detección de SQL Injection Boolean-based en GET /booking

IDENTIFICADOR	SQLi-001
DESCRIPCIÓN	Verificar si el endpoint GET /booking es vulnerable a SQL Injection boolean-based, evaluando si variaciones lógicas en los parámetros producen cambios anómalos en las respuestas.
PRECONDICIONES	<ul style="list-style-type: none">• Postman instalado.• Entorno Postman con variable: base_url = https://restful-booker.herokuapp.com• Scripts de test configurados en los requests A, B y C.• Base de datos poblada previamente con bookings válidos (ver sección Setup).• Endpoint /booking operativo.

SET-UP DE DATOS	<p>La API puede devolver [] si no hay reservas, lo cual impediría detectar variaciones entre TRUE y FALSE. Por eso se deben crear al menos 3 reservas antes de ejecutar la prueba.</p> <p>Request POST de ejemplo</p> <pre>POST {{base_url}}/booking</pre> <p>Body (JSON):</p> <pre>{ "firstname": "Marco", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast" }</pre> <p>Test asociado al POST para verificar la creación de los datos:</p> <pre>pm.test("Status code is 200", function () { pm.response.to.have.status(200); }); pm.test("Verifying booking id", function () { const jsonData = pm.response.json(); const bookingId = jsonData.bookingid; pm.environment.set("BookingId", bookingId); pm.expect(bookingId).to.not.be.null(); });</pre>
DATOS DE ENTRADA	<p>A (baseline) firstname: Marco (o cualquier existente)</p> <p>B (TRUE) firstname: ' OR '1'='1</p> <p>C (FALSE) firstname ' AND '1'='2</p>
PASOS A SEGUIR	<ul style="list-style-type: none"> Ejecutar Request A — Baseline Enviar GET {{base_url}}/booking?firstname=Marco. Confirmar que el status es 200. El script guarda baseline_body y baseline_len Ejecutar Request B — Inyección TRUE Enviar GET {{base_url}}/booking?firstname=' OR '1'='1'. Verificar ausencia de errores SQL. El script guarda inj_true_len Ejecutar Request C — Inyección FALSE Enviar GET {{base_url}}/booking?firstname=' AND '1'='2'. Verificar ausencia de errores SQL. El script guarda inj_false_len

	Postman ejecuta el test que compara TRUE vs FALSE (tolerancia: ±20).
RESULTADO ESPERADO	<p>Status code debe ser 200 o 400, nunca 500/502/503/504.</p> <p>El body no debe contener indicadores de error SQL: sql, mysql, syntax, exception, stacktrace, odbc, etc.</p> <p>La respuesta del request B (TRUE) no debe diferir significativamente de C (FALSE).</p> <p>La longitud de ambas respuestas debe ser similar (± 200).</p> <p>No debe observarse aumento de registros en B ni reducción en C.</p> <p>B y C similares o diferencias mínimas</p>
RESULTADO OBTENIDO	A: [{"bookingid": 296}] B: [] C: []
ESTADO	Pass

Evidencias de la prueba

GET A (BaseLine):

The screenshot shows the Postman interface with a GET request to `https://restful-booker.herokuapp.com/booking?firstname=Marco`. The 'Scripts' tab contains the following code:

```

1 pm.test("Baseline OK", () => {
2   pm.expect(pm.response.code).to.equal(200);
3   pm.environment.set("baseline_body", pm.response.text());
4   pm.environment.set("baseline_len", pm.response.text().length);
5 });

```

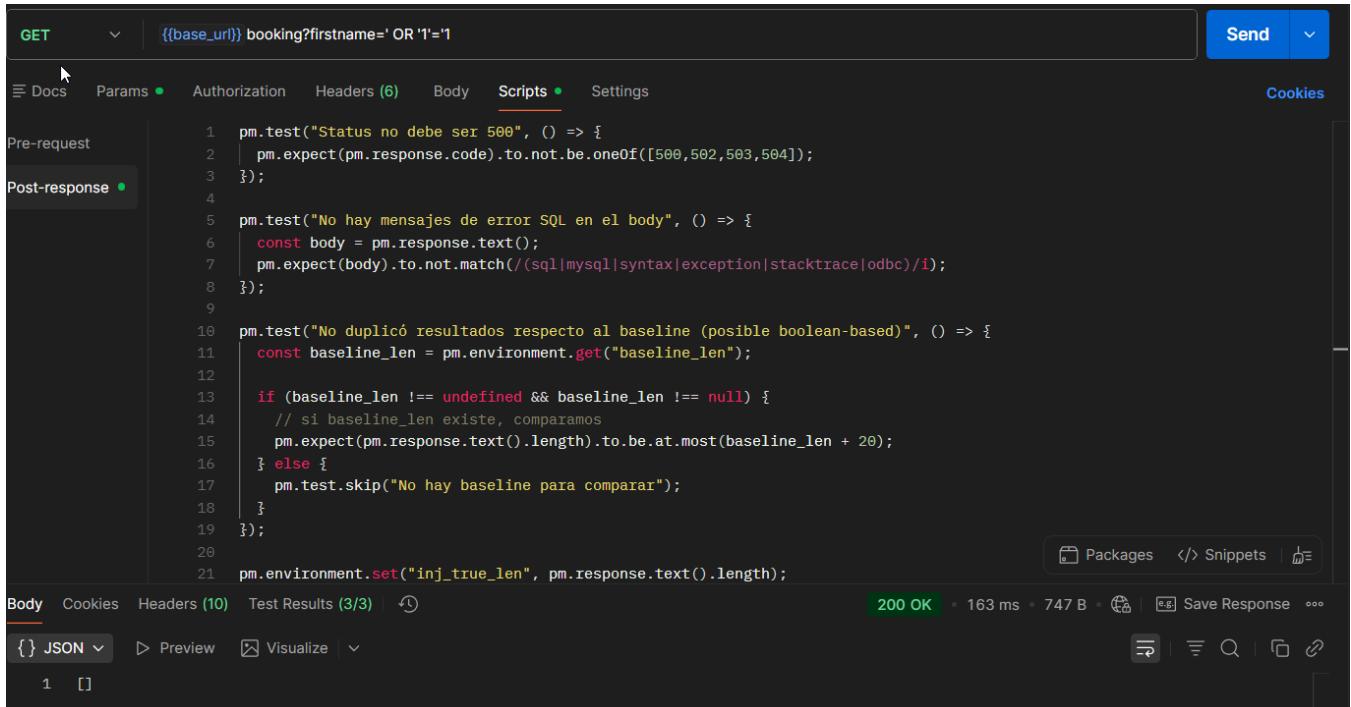
The response body is displayed as JSON:

```

1 [
2   {
3     "bookingid": 296
4   }
5 ]

```

At the bottom, the status is 200 OK with 166 ms response time and 762 B size.

GET B:

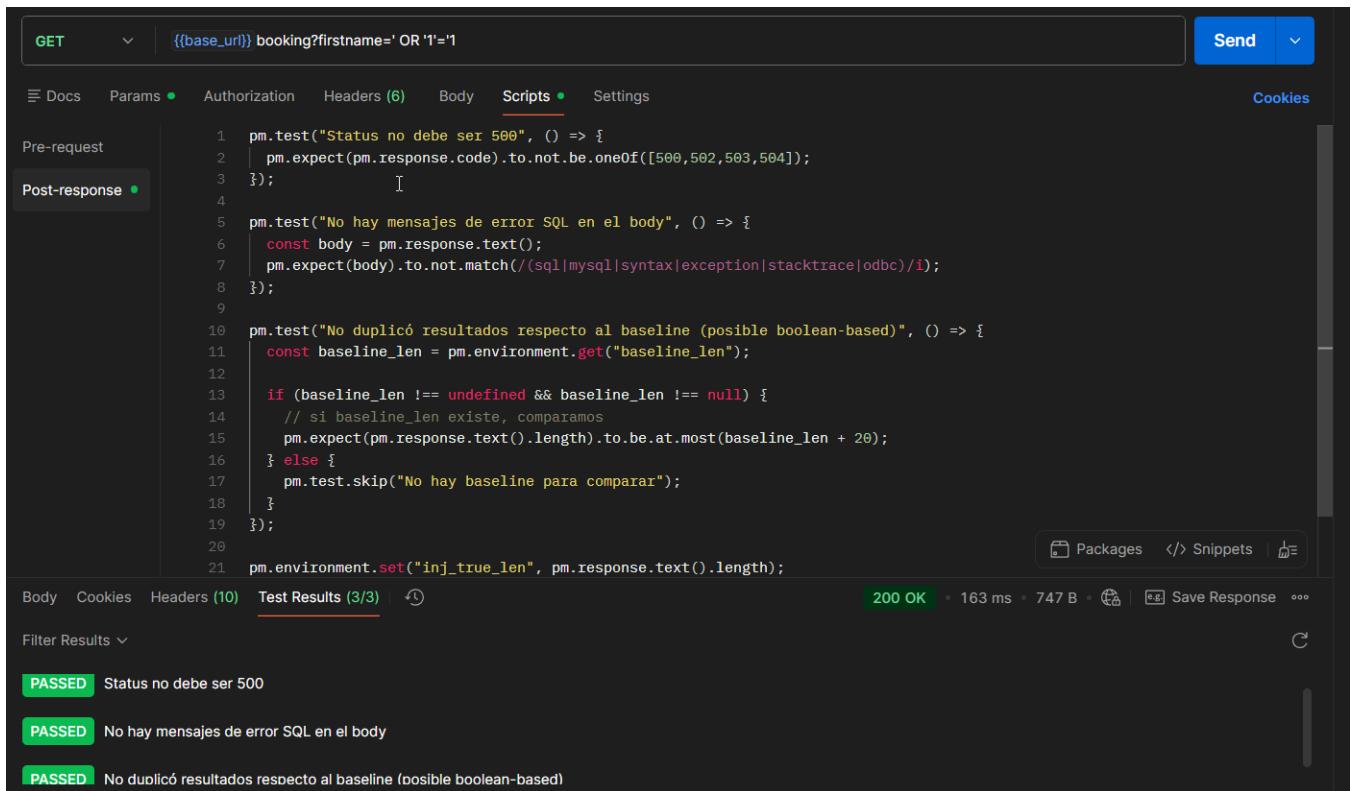
GET `{{base_url}} booking?firstname=' OR '1='1'` Send

Docs Params Authorization Headers (6) Body Scripts **Post-response** Cookies

```
1 pm.test("Status no debe ser 500", () => {
2   | pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
3 });
4
5 pm.test("No hay mensajes de error SQL en el body", () => {
6   | const body = pm.response.text();
7   | pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
8 });
9
10 pm.test("No duplicó resultados respecto al baseline (possible boolean-based)", () => {
11   | const baseline_len = pm.environment.get("baseline_len");
12
13   if (baseline_len !== undefined && baseline_len !== null) {
14     // si baseline_len existe, comparamos
15     pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
16   } else {
17     pm.test.skip("No hay baseline para comparar");
18   }
19 });
20
21 pm.environment.set("inj_true_len", pm.response.text().length);
```

Body Cookies Headers (10) Test Results (3/3) ↴ 200 OK • 163 ms • 747 B • ↗ Save Response ⋮

{ JSON ↴ Preview Visualize ↴ 1 []



GET `{{base_url}} booking?firstname=' OR '1='1'` Send

Docs Params Authorization Headers (6) Body Scripts **Post-response** Cookies

```
1 pm.test("Status no debe ser 500", () => {
2   | pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
3 });
4
5 pm.test("No hay mensajes de error SQL en el body", () => {
6   | const body = pm.response.text();
7   | pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
8 });
9
10 pm.test("No duplicó resultados respecto al baseline (possible boolean-based)", () => {
11   | const baseline_len = pm.environment.get("baseline_len");
12
13   if (baseline_len !== undefined && baseline_len !== null) {
14     // si baseline_len existe, comparamos
15     pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
16   } else {
17     pm.test.skip("No hay baseline para comparar");
18   }
19 });
20
21 pm.environment.set("inj_true_len", pm.response.text().length);
```

Body Cookies Headers (10) Test Results (3/3) ↴ 200 OK • 163 ms • 747 B • ↗ Save Response ⋮

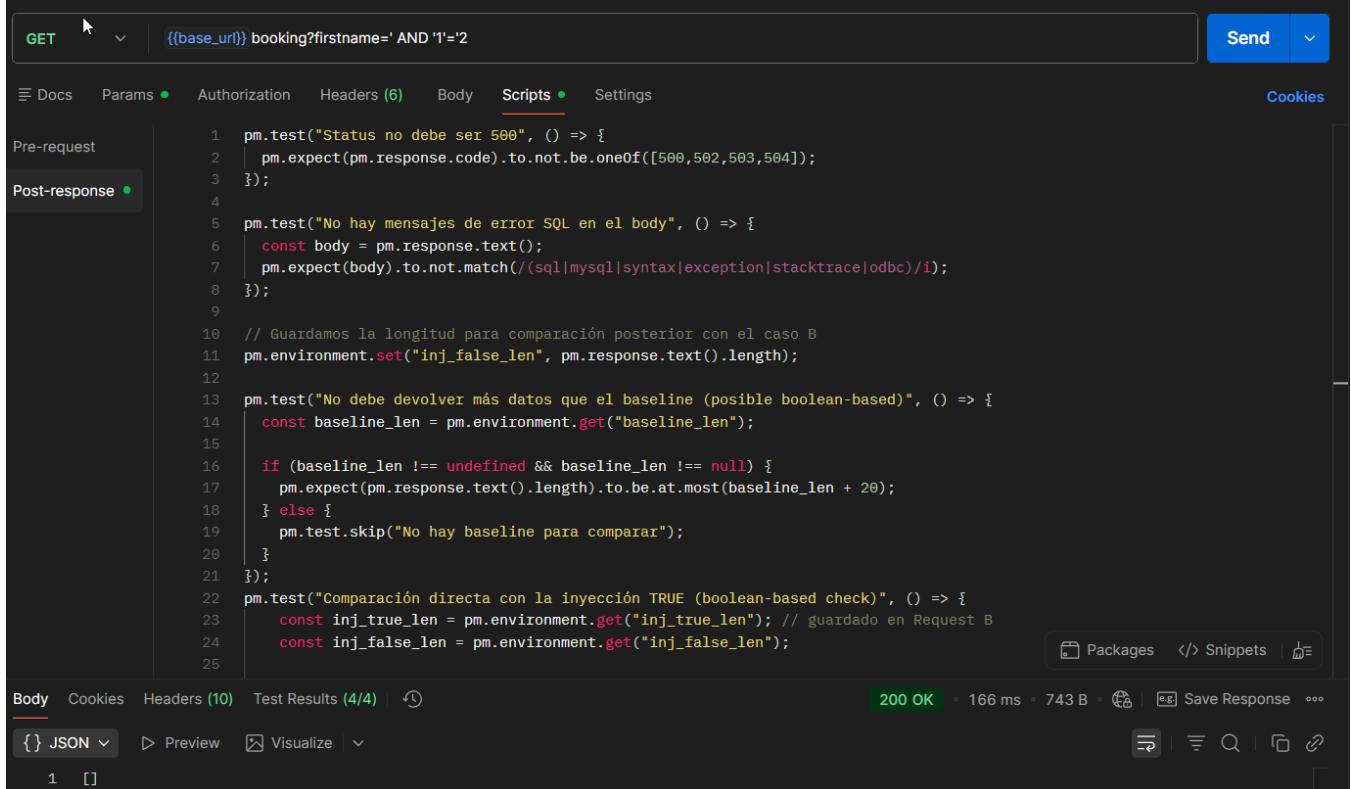
Filter Results ↴

PASSED Status no debe ser 500

PASSED No hay mensajes de error SQL en el body

PASSED No duplicó resultados respecto al baseline (possible boolean-based)

GET C:



```

GET {{base_url}} booking?firstname=' AND '1='2

pm.test("Status no debe ser 500", () => {
  pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
});

pm.test("No hay mensajes de error SQL en el body", () => {
  const body = pm.response.text();
  pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
});

// Guardamos la longitud para comparación posterior con el caso B
pm.environment.set("inj_false_len", pm.response.text().length);

pm.test("No debe devolver más datos que el baseline (possible boolean-based)", () => {
  const baseline_len = pm.environment.get("baseline_len");

  if (baseline_len !== undefined && baseline_len !== null) {
    pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
  } else {
    pm.test.skip("No hay baseline para comparar");
  }
});

pm.test("Comparación directa con la inyección TRUE (boolean-based check)", () => {
  const inj_true_len = pm.environment.get("inj_true_len"); // guardado en Request B
  const inj_false_len = pm.environment.get("inj_false_len");

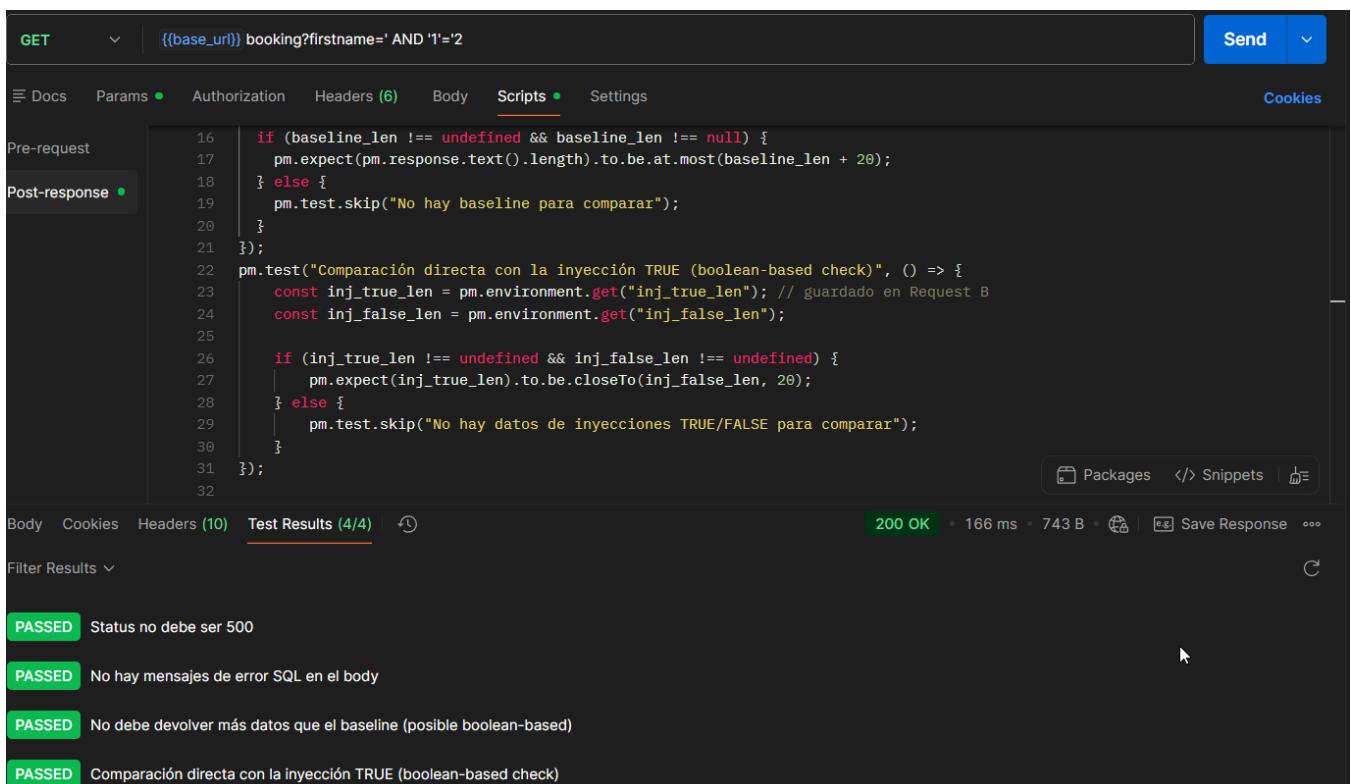
  if (inj_true_len !== undefined && inj_false_len !== undefined) {
    pm.expect(inj_true_len).to.be.closeTo(inj_false_len, 20);
  } else {
    pm.test.skip("No hay datos de inyecciones TRUE/FALSE para comparar");
  }
});

```

Body Cookies Headers (10) Test Results (4/4)

200 OK • 166 ms • 743 B • Save Response

{ } JSON ▾ Preview Visualize



```

GET {{base_url}} booking?firstname=' AND '1='2

pm.test("Status no debe ser 500", () => {
  pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
});

pm.test("No hay mensajes de error SQL en el body", () => {
  const body = pm.response.text();
  pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
});

pm.test("Comparación directa con la inyección TRUE (boolean-based check)", () => {
  const inj_true_len = pm.environment.get("inj_true_len"); // guardado en Request B
  const inj_false_len = pm.environment.get("inj_false_len");

  if (inj_true_len !== undefined && inj_false_len !== undefined) {
    pm.expect(inj_true_len).to.be.closeTo(inj_false_len, 20);
  } else {
    pm.test.skip("No hay datos de inyecciones TRUE/FALSE para comparar");
  }
});

```

Body Cookies Headers (10) Test Results (4/4)

200 OK • 166 ms • 743 B • Save Response

PASSED Status no debe ser 500

PASSED No hay mensajes de error SQL en el body

PASSED No debe devolver más datos que el baseline (possible boolean-based)

PASSED Comparación directa con la inyección TRUE (boolean-based check)

SETUP:

The screenshot shows a Postman interface with a POST request to <https://restful-booker.herokuapp.com/booking>. The request body is a JSON object representing a booking with fields like firstname, lastname, totalprice, depositpaid, bookingdates, and additionalneeds. The response status is 200 OK, and the response body is identical to the request body, indicating a successful creation of the booking.

```
POST https://restful-booker.herokuapp.com/booking

{
  "firstname": "Marco",
  "lastname": "Polo",
  "totalprice": 500,
  "depositpaid": false,
  "bookingdates": {
    "checkin": "1300-01-01",
    "checkout": "1300-02-01"
  },
  "additionalneeds": "Cuaderno de las maravillas"
}

200 OK
```

Body Cookies Headers (10) Test Results (2/2)

```
{}
{} JSON ▾ Preview Visualize ▾
```

```
1 {
  "bookingid": 4454,
  "booking": {
    "firstname": "Marco",
    "lastname": "Polo",
    "totalprice": 500,
    "depositpaid": false,
    "bookingdates": {
      "checkin": "1300-01-01",
      "checkout": "1300-02-01"
    },
    "additionalneeds": "Cuaderno de las maravillas"
}
```

The screenshot shows Postman test results for the booking creation. It includes two test cases: one for the status code and another for verifying the booking ID. Both tests passed.

PASSED Status code is 200

PASSED verifying booking id

3.1.2 Error-Based SQLi en GET /booking

IDENTIFICADOR	SQLi-002
DESCRIPCIÓN	Verificar si el endpoint GET /booking es vulnerable a SQL Injection Error-Based, enviando un parámetro con comillas malformadas para observar si la API revela errores internos del motor SQL (mensajes de error, stacktrace o status 500).
PRECONDICIONES	<ul style="list-style-type: none"> Postman instalado. El endpoint /booking operativo.
DATOS DE ENTRADA	firstname=""
PASOS A SEGUIR	<ul style="list-style-type: none"> Abrir Postman y seleccionar método GET. Enviar la solicitud a https://restful-booker.herokuapp.com/booking?firstname=';--/* Observar status code y body de respuesta
RESULTADO ESPERADO	<ul style="list-style-type: none"> No debe devolverse un status 500/502/503/504. No deben aparecer mensajes típicos de error SQL como sql, mysql, syntax, exception, stacktrace, odbc, pdo, ORA-. La API debe responder con un 200 o 400 y un body seguro (generalmente []).
RESULTADO OBTENIDO	Status Code 200 Response Body: []
ESTADO	Pass

Evidencias de la prueba

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** https://restful-booker.herokuapp.com/booking?firstname=';--/*'
- Headers:** (6 items)
- Body:** (Empty)
- Scripts:** (Script provided in the screenshot)
- Test Results:** 2/2 (2 successful, 0 failed)
- Response Status:** 200 OK
- Response Time:** 168 ms
- Response Size:** 747 B
- Response Content:** []

```

1 pm.test("Status no debe ser 500", () => {
2   pm.expect(pm.response.code).to.not.be.oneOf([500, 502, 503, 504]);
3 });
4
5 pm.test("No debe haber mensajes de error SQL", () => {
6   const body = pm.response.text();
7   pm.expect(body).to.not.match(
8     /(sql|mysql|syntax|exception|stacktrace|odbc|pdo|ora-|warning|error)/i
9   );
10 });

```

3.1.3 SQLi Error Based Body — /booking

IDENTIFICADOR	SQLi-003
DESCRIPCIÓN	Evaluar si el endpoint POST /booking es vulnerable a SQL Injection de tipo error-based , enviando un payload diseñado para romper la sintaxis SQL mediante comillas simples, dobles y otros caracteres especiales. El objetivo es observar si el servidor expone mensajes de error internos o comportamientos inesperados que indiquen falta de sanitización de entrada.
PRECONDICIONES	API en funcionamiento.
DATOS DE ENTRADA	Body: <pre>{ "firstname": "\/*/ /* ; --", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast" }</pre>
PASOS A SEGUIR	<ul style="list-style-type: none"> Abrir Postman y seleccionar POST /booking. URL: https://restful-booker.herokuapp.com/booking Ir a Body → raw → JSON. Pegar el payload malicioso y enviar la solicitud. Registrar el código de estado HTTP y el cuerpo de la respuesta.
RESULTADO ESPERADO	<ul style="list-style-type: none"> El servidor no debe devolver errores SQL. No deben aparecer mensajes como SQL syntax error, mysql, exception, stacktrace, ORA-00933. El status code debe ser 200 o algún código de validación lógica (400), pero nunca 500+. El valor malicioso debe ser tratado como un texto más (si no hay sanitización del lado del servidor). El backend no debe mostrar evidencia de ejecutar SQL.
RESULTADO OBTENIDO	<p>Status code 200 Response body:</p> <pre>{ "bookingid": 547, "booking": { "firstname": "\/*/ /* ; --", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast" } }</pre>

	<pre> "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast" } } </pre>
OBSERVACIONES	<ul style="list-style-type: none"> • La API acepta y almacena el payload malicioso sin filtrarlo. • No se generan errores SQL ni mensajes internos. • No se expone información sensible. • El comportamiento es consistente con un backend que no ejecuta SQL (no sabemos si Restful Booker lo hace).
ESTADO	Pass

Evidencias de la prueba:

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <https://restful-booker.herokuapp.com/booking>
- Body:** Raw JSON payload (shown below)
- Headers:** (9 items listed)
- Response Status:** 200 OK
- Response Body (Raw JSON):**

```

1  {
2      "bookingid": 547,
3      "booking": {
4          "firstname": "\/*/ /*; --",
5          "lastname": "Brown",
6          "totalprice": 111,
7          "depositpaid": true,
8          "bookingdates": {
9              "checkin": "2018-01-01",
10             "checkout": "2019-01-01"
11         },
12         "additionalneeds": "Breakfast"
13     }
}

```

3.1.4 SQLi Boolean-Based en Body

IDENTIFICADOR	SQLi-004
DESCRIPCIÓN	Evaluar si el endpoint POST /booking es vulnerable a SQL injection de tipo boolean-based comparando las respuestas a payloads que, interpretados como código SQL, serían siempre verdaderos o siempre falsos. Tipos o contenidos de respuestas significativamente distintas indicarían un patrón típico de SQL Injection inferencial y puede detectarse aunque no se expongan errores SQL.
PRECONDICIONES	API en funcionamiento.
DATOS DE ENTRADA	<pre>{ "firstname" : "" OR '1'='1", "lastname" : "Trump", "totalprice" : 6000, "depositpaid" : false, "bookingdates" : { "checkin" : "2019-01-01", "checkout" : "2029-01-01" }, "additionalneeds" : "Un portaavinoes" }</pre> <p>Body en caso False:</p> <pre>{ "firstname" : "" AND '1'='2", "lastname" : "Trump", "totalprice" : 6000, "depositpaid" : false, "bookingdates" : { "checkin" : "2019-01-01", "checkout" : "2029-01-01" }, "additionalneeds" : "Un portaavinoes" }</pre>
PASOS A SEGUIR	<ul style="list-style-type: none"> Enviar POST con el payload TRUE (' OR '1'='1) y guardar status code y body response Enviar POST con el payload FALSE (' OR '1'='2) y guardar status code y body response Comparar status code y body response de ambas request
RESULTADO ESPERADO	<ul style="list-style-type: none"> Ambos requests deben devolver el mismo status code

	<ul style="list-style-type: none"> El contenido de respuesta debe tener estructura equivalente No deben aparecer errores SQL ni stack traces Longitud de respuesta similar
RESULTADO OBTENIDO	<ul style="list-style-type: none"> Status code 200 en ambos casos Las longitudes idénticas No aparecen errores SQL
ESTADO	Pass

Evidencias de la prueba:

POST https://restful-booker.herokuapp.com/booking Send

Docs Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Schema Beautify

```

1 {
2   "firstname" : "' OR '1'='1",
3   "lastname" : "Trump",
4   "totalprice" : 6000,
5   "depositpaid" : false,
6   "bookingdates" : {
7     "checkin" : "2019-01-01",
8     "checkout" : "2029-01-01"
9   },
10  "additionalneeds" : "Un portaavinoes"
11 }
```

Body Cookies Headers (10) Test Results 200 OK 782 ms 953 B Save Response

{ } JSON Preview Visualize

```

1 {
2   "bookingid": 1514,
3   "booking": {
4     "firstname": "' OR '1'='1",
5     "lastname": "Trump",
6     "totalprice": 6000,
7     "depositpaid": false,
8     "bookingdates": {
9       "checkin": "2019-01-01",
10      "checkout": "2029-01-01"
11    },
12    "additionalneeds": "Un portaavinoes"
13 }
```

POST https://restful-booker.herokuapp.com/booking Send

Docs Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Schema Beautify

```

1 {
2   "firstname" : "' AND '1'='2",
3   "lastname" : "Trump",
4   "totalprice" : 6000,
5   "depositpaid" : false,
6   "bookingdates" : {
7     "checkin" : "2019-01-01",
8     "checkout" : "2029-01-01"
9   },
10  "additionalneeds" : "Un portaavinoes"
11 }
```

Body Cookies Headers (10) Test Results (2/2) 200 OK 812 ms 958 B Save Response

{ } JSON Preview Visualize

```

1 {
2   "bookingid": 2538,
3   "booking": {
4     "firstname": "' AND '1'='2",
5     "lastname": "Trump",
6     "totalprice": 6000,
7     "depositpaid": false,
8     "bookingdates": {
9       "checkin": "2019-01-01",
10      "checkout": "2029-01-01"
11    },
12    "additionalneeds": "Un portaavinoes"
13 }
```

3.1.5 SQL Injection Time-Based en POST Body

IDENTIFICADOR	SQLi-005
DESCRIPCIÓN	Esta prueba mide la diferencia de tiempo entre un request legítimo y uno que contiene un payload diseñado para introducir un retraso deliberado en el motor de base de datos. Con ello se podrá identificar una posible vulnerabilidad de SQL Injection Time-Based en el endpoint POST /booking.
PRECONDICIONES	API en funcionamiento. El tiempo de respuesta estándar del endpoint es estable (menos de 1000 ms).
DATOS DE ENTRADA	<p>Body de Request Legítimo:</p> <pre>{ "firstname": "Jim", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2024-01-01", "checkout": "2024-02-01" }, "additionalneeds": "Breakfast" }</pre> <p>Body malicioso:</p> <pre>{ "firstname": "test'; SELECT pg_sleep(5)--", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2024-01-01", "checkout": "2024-02-01" }, "additionalneeds": "Breakfast" }</pre> <p>Payloads alternativos por motor SQL:</p> <p>MySQL test' OR SLEEP(5)-- -</p>

	PostgreSQL test'; SELECT pg_sleep(5)-- SQL Server test'; WAITFOR DELAY '0:0:5'--
PASOS A SEGUIR	<ul style="list-style-type: none"> Enviar el body legítimo y registrar status code, body response y tiempo de respuesta. Enviar payload de time-based SQLi reemplazando firstname por el payload malicioso (en las tres versiones) y registrar status code, body response y tiempo de respuesta. Repetir el request malicioso 3 veces para confirmar consistencia, registrar todos los tiempos. Comparar tiempo de respuesta de los request maliciosos con el del legítimo.
RESULTADO ESPERADO	<ul style="list-style-type: none"> Status code en todos los casos 200 Body similar al request normal Tiempos de respuesta similar al request normal (diferencia <1000 ms)
RESULTADO OBTENIDO	<ul style="list-style-type: none"> Status code en todos los casos 200 Body similar al request normal Tiempos de respuesta similar al request normal (diferencia <1000 ms)
ESTADO	Pass

Evidencias de la prueba

Request legítimo

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** https://restful-booker.herokuapp.com/booking
- Body:** Raw JSON (selected)

```

1 {
2   "firstname": "Jim",
3   "lastname": "Brown",
4   "totalprice": 111,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-01-01",
8     "checkout": "2019-01-01"
9   },
10  "additionalneeds": "Breakfast"
11 }
```
- Response Status:** 200 OK
- Response Time:** 769 ms
- Response Size:** 941 B
- Response Headers:** (10) including Content-Type: application/json, Content-Length: 941, Date: Mon, 01 Jan 2018 10:30:00 GMT, Connection: keep-alive, Set-Cookie: JSESSIONID=0000uXWzJLjyfCgk:1qkVQ, Transfer-Encoding: chunked
- Response Body:** JSON (selected)

```

1 {
2   "bookingid": 3057,
3   "booking": {
4     "firstname": "Jim",
5     "lastname": "Brown",
6     "totalprice": 111,
7     "depositpaid": true,
8     "bookingdates": {
9       "checkin": "2018-01-01",
10      "checkout": "2019-01-01"
11    },
12    "additionalneeds": "Breakfast"
13  }
```

Malicioso PostgreSQL:

The screenshot shows a Postman request to `https://restful-booker.herokuapp.com/booking`. The request method is POST. The JSON body contains a malicious payload:

```
1 {  
2   "firstname" : "test'; SELECT pg_sleep(5)--",  
3   "lastname" : "Brown",  
4   "totalprice" : 111,  
5   "depositpaid" : true,  
6   "bookingdates" : {  
7     "checkin" : "2018-01-01",  
8     "checkout" : "2019-01-01"  
9   },  
10  "additionalneeds" : "Breakfast"  
11 }  
12 }
```

The response status is 200 OK, with a duration of 742 ms and a response size of 965 B. The response body is identical to the request body.

The screenshot shows a Postman request to `https://restful-booker.herokuapp.com/booking`. The request method is POST. The JSON body contains a malicious payload:

```
1 {  
2   "firstname" : "test'; SELECT pg_sleep(5)--",  
3   "lastname" : "Brown",  
4   "totalprice" : 111,  
5   "depositpaid" : true,  
6   "bookingdates" : {  
7     "checkin" : "2018-01-01",  
8     "checkout" : "2019-01-01"  
9   },  
10  "additionalneeds" : "Breakfast"  
11 }  
12 }
```

The response status is 200 OK, with a duration of 744 ms and a response size of 973 B. The response body is identical to the request body.

The screenshot shows a Postman request to `POST {{base_url}} /booking`. The request body is raw JSON:

```
1 {
2     "firstname" : "test'; SELECT pg_sleep(5)--",
3     "lastname" : "Brown",
4     "totalprice" : 111,
5     "depositpaid" : true,
6     "bookingdates" : {
7         "checkin" : "2018-01-01",
8         "checkout" : "2019-01-01"
9     },
10    "additionalneeds" : "Breakfast"
11 }
```

The response is 200 OK with 777 ms latency and 961 B size. The response body is identical to the request body.

Malicioso MySQL:

The screenshot shows a Postman request to `POST {{base_url}} /booking`. The request body is raw JSON with a SQL injection:

```
1 {
2     "firstname" : "test' OR SLEEP(5)--",
3     "lastname" : "Brown",
4     "totalprice" : 111,
5     "depositpaid" : true,
6     "bookingdates" : {
7         "checkin" : "2018-01-01",
8         "checkout" : "2019-01-01"
9     },
10    "additionalneeds" : "Breakfast"
11 }
```

The response is 200 OK with 723 ms latency and 967 B size. The response body is identical to the request body.

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' and the URL '{{base_url}} /booking'. Below the header are tabs for 'Docs', 'Params', 'Authorization', 'Headers (9)', 'Body' (which is selected), 'Scripts', 'Settings', and 'Cookies'. Under the 'Body' tab, there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected), 'binary', 'GraphQL', and 'JSON'. To the right of these options are 'Schema' and 'Beautify' buttons. The main area contains the raw JSON body of the request:

```
1 {  
2   "firstname" : "test' OR SLEEP(5)-- -- ",  
3   "lastname" : "Brown",  
4   "totalprice" : 2111,  
5   "depositpaid" : true,  
6   "bookingdates" : {  
7     "checkin" : "2018-01-01",  
8     "checkout" : "2019-01-01"  
9   },  
10  "additionalneeds" : "Breakfast"  
11 }
```

Below the body, there are tabs for 'Body', 'Cookies', 'Headers (10)', and 'Test Results (2/2)'. The 'Test Results' tab shows a single entry with status '200 OK', time '164 ms', size '964 B', and a 'Save Response' button. The response body is identical to the request body:

```
1 {  
2   "bookingid": 2159,  
3   "booking": {  
4     "firstname": "test' OR SLEEP(5)-- -- ",  
5     "lastname": "Brown",  
6     "totalprice": 2111,  
7     "depositpaid": true,  
8     "bookingdates": {  
9       "checkin": "2018-01-01",  
10      "checkout": "2019-01-01"  
11    },  
12    "additionalneeds": "Breakfast"  
13  }
```

The screenshot shows the Postman application interface, similar to the one above. At the top, there is a header bar with 'POST' and the URL '{{base_url}} /booking'. Below the header are tabs for 'Docs', 'Params', 'Authorization', 'Headers (9)', 'Body' (selected), 'Scripts', 'Settings', and 'Cookies'. Under the 'Body' tab, there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (selected), 'binary', 'GraphQL', and 'JSON'. To the right of these options are 'Schema' and 'Beautify' buttons. The main area contains the raw JSON body of the request:

```
1 {  
2   "firstname" : "test' OR SLEEP(5)-- -- ",  
3   "lastname" : "Brown",  
4   "totalprice" : 2111,  
5   "depositpaid" : true,  
6   "bookingdates" : {  
7     "checkin" : "2018-01-01",  
8     "checkout" : "2019-01-01"  
9   },  
10  "additionalneeds" : "Breakfast"  
11 }
```

Below the body, there are tabs for 'Body', 'Cookies', 'Headers (10)', and 'Test Results (2/2)'. The 'Test Results' tab shows a single entry with status '200 OK', time '169 ms', size '956 B', and a 'Save Response' button. The response body is identical to the request body:

```
1 {  
2   "bookingid": 2290,  
3   "booking": {  
4     "firstname": "test' OR SLEEP(5)-- -- ",  
5     "lastname": "Brown",  
6     "totalprice": 2111,  
7     "depositpaid": true,  
8     "bookingdates": {  
9       "checkin": "2018-01-01",  
10      "checkout": "2019-01-01"  
11    },  
12    "additionalneeds": "Breakfast"  
13  }
```

Malicioso SQL server:

POST {{base_url}} /booking

Body (raw) JSON

```
1 {  
2     "firstname" : "test'; WAITFOR DELAY '0:0:5'--",  
3     "lastname" : "Brown",  
4     "totalprice" : 111,  
5     "depositpaid" : true,  
6     "bookingdates" : {  
7         "checkin" : "2018-01-01",  
8         "checkout" : "2019-01-01"  
9     },  
10    "additionalneeds" : "Breakfast"  
11}  
12  
13
```

200 OK | 166 ms | 968 B | Save Response | ...

Body Cookies Headers (10) Test Results (2/2)

```
1 {  
2     "bookingid": 2443,  
3     "booking": {  
4         "firstname": "test'; WAITFOR DELAY '0:0:5'--",  
5         "lastname": "Brown",  
6         "totalprice": 111,  
7         "depositpaid": true,  
8         "bookingdates": {  
9             "checkin": "2018-01-01",  
10            "checkout": "2019-01-01"  
11        },  
12        "additionalneeds": "Breakfast"  
13    }  
14}
```

POST {{base_url}} /booking

Body (raw) JSON

```
1 {  
2     "firstname" : "test'; WAITFOR DELAY '0:0:5'--",  
3     "lastname" : "Brown",  
4     "totalprice" : 111,  
5     "depositpaid" : true,  
6     "bookingdates" : {  
7         "checkin" : "2018-01-01",  
8         "checkout" : "2019-01-01"  
9     },  
10    "additionalneeds" : "Breakfast"  
11}  
12  
13
```

200 OK | 164 ms | 964 B | Save Response | ...

Body Cookies Headers (10) Test Results (2/2)

```
1 {  
2     "bookingid": 2573,  
3     "booking": {  
4         "firstname": "test'; WAITFOR DELAY '0:0:5'--",  
5         "lastname": "Brown",  
6         "totalprice": 111,  
7         "depositpaid": true,  
8         "bookingdates": {  
9             "checkin": "2018-01-01",  
10            "checkout": "2019-01-01"  
11        },  
12        "additionalneeds": "Breakfast"  
13    }  
14}
```

The screenshot shows the Postman application interface. At the top, there's a header with 'POST' and the URL '{{base_url}} /booking'. Below the header are tabs for 'Docs', 'Params', 'Authorization', 'Headers (9)', 'Body' (which is selected), 'Scripts', and 'Settings'. Under 'Body', options include 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected), 'binary', 'GraphQL', and 'JSON'. To the right of these are 'Cookies', 'Schema', and 'Beautify' buttons. The main area shows a JSON payload with line numbers 1 through 11. The response section shows a status of '200 OK', a duration of '165 ms', a size of '964 B', and a 'Save Response' button.

```

1 {
2   "firstname" : "test'; WAITFOR DELAY '0:0:5'--",
3   "lastname" : "Brown",
4   "totalprice" : 111,
5   "depositpaid" : true,
6   "bookingdates" : {
7     "checkin" : "2018-01-01",
8     "checkout" : "2019-01-01"
9   },
10  "additionalneeds" : "Breakfast"
11 }
  
```

Below the main area, there are tabs for 'Body', 'Cookies', 'Headers (10)', 'Test Results (2/2)', and a refresh icon. The 'Body' tab is selected and shows the same JSON payload. The 'Responses' tab also shows the same JSON payload. The bottom right corner has icons for copy, export, search, and refresh.

4.2 Vulnerabilidades varias en validación y codificado de Input & Output

Se evaluaron múltiples vectores de inyección (HTML Injection, Script Tag Injection, Stored XSS). Dado que todos los vectores resultaron ejecutables, comparten la misma causa raíz y las mismas recomendaciones de mitigación.

3.2.1 HTML Injection

IDENTIFICADOR	IV-01
DESCRIPCIÓN	<p>Esta prueba verifica si la API valida y sanitiza correctamente contenido HTML enviado en campos controlados por el usuario.</p> <p>El objetivo es determinar si etiquetas HTML pueden ser almacenadas y devueltas sin escapar, lo que podría afectar a cualquier aplicación cliente que renderice estos datos y evidencia una validación de entradas insuficiente.</p>
PRECONDICIONES	<p>Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com</p> <p>API en funcionamiento</p>
DATOS DE ENTRADA	<p>Body de POST Request:</p> <pre>{ }</pre>

	<pre> "firstname": "<h1>Pedro</h1>", "lastname": "Test", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2020-01-01", "checkout": "2020-01-02" }, "additionalneeds": "Breakfast" } Request GET: https://restful-booker.herokuapp.com/booking/(id de POST) </pre>
PASOS A SEGUIR	<ul style="list-style-type: none"> Enviar una solicitud POST /booking con el payload indicado. Verificar el código de estado de la respuesta. Obtener el bookingid generado. Ejecutar una solicitud GET /booking/{bookingid}.
RESULTADO ESPERADO	<ul style="list-style-type: none"> Status 400 Bad Request, o el campo "<h1>...</h1>" devuelto escapado en los body response. Ejemplo: &lt;h1&gt;.
RESULTADO OBTENIDO	<ul style="list-style-type: none"> Status code casos 200 y body response no escapado.
ESTADO	FAIL
SEVERIDAD	ALTA
PRIORIDAD	ALTA
MITIGACIÓN SUGERIDA	<p>La vulnerabilidad debe ser mitigada implementando sanitización de entrada del lado del servidor y codificado seguro de salida, validacion estricta de modelos y testeo automatizado de seguridad.</p> <p>Además puede implementarse, como capa de seguridad adicional, configuración WAF de bloqueo de patrones típicos de XSS.</p>

Evidencias de la prueba

The screenshot shows a Postman request for the endpoint `POST {{base_url}}/booking`. The request body is a JSON object:

```

1  {
2   "firstname": "<h1>Pedro</h1>",
3   "lastname": "Brown",
4   "totalprice": 111,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-01-01",
8     "checkout": "2019-01-01"
9   },
10  "additionalneeds": "Breakfast"
11 }

```

The response status is 200 OK, and the response body is identical to the request body, indicating that the input was not properly sanitized.

3.2.2 Stored XSS y Tag Injection

IDENTIFICADOR	IV-02
DESCRIPCIÓN	Esta prueba verifica si el endpoint POST /booking permite almacenar código HTML/JavaScript malicioso (Stored XSS), lo cual podría ser ejecutado posteriormente en una interfaz cliente o panel administrativo que consuma los datos devueltos por la API.
PRECONDICIONES	Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com API en funcionamiento
DATOS DE ENTRADA	Body de POST Request: <pre>{ "firstname": "<script>alert('XSS')</script>", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast" }</pre>
PASOS A SEGUIR	<ul style="list-style-type: none"> Enviar request POST URL: {{base_url}}/booking con el payload malicioso anterior. Registrar status code y response body Enviar request GET URL: {{base_url}}/booking/(id del POST)
RESULTADO ESPERADO	<ul style="list-style-type: none"> Status 400 Bad Request, o el campo "<script>...</script>" devuelto escapado en los body response. Ejemplo: &lt;script&gt;alert('XSS')&lt;/script&gt;
RESULTADO OBTENIDO	<ul style="list-style-type: none"> Status code casos 200 y body response no escapado.
ESTADO	Fail
SEVERIDAD	ALTA
PRIORIDAD	ALTA
MITIGACIÓN SUGERIDA	Misma raíz de problema que el defecto de HTMLi y, por ende, mismas sugerencias: La vulnerabilidad debe ser mitigada implementando sanitización de entrada del lado del servidor y codificado

seguro de salida, validacion estricta de modelos y testeo automatizado de seguridad.
Además puede implementarse, como capa de seguridad adicional, configuración WAF de bloqueo de patrones típicos de XSS.

Evidencia de la prueba:

POST {{base_url}} /booking

Body raw JSON

```
1 {  
2   "firstname": "<script>alert('XSS')</script>",  
3   "lastname": "Brown",  
4   "totalprice": 111,  
5   "depositpaid": true,  
6   "bookingdates": {  
7     "checkin": "2018-01-01",  
8     "checkout": "2019-01-01"  
9   },  
10  "additionalneeds": "Breakfast"  
11 }  
12 }
```

200 OK 167 ms 975 B Save Response

Body Cookies Headers (10) Test Results (2/2)

```
1 {  
2   "bookingid": 4470,  
3   "booking": {  
4     "firstname": "<script>alert('XSS')</script>",  
5     "lastname": "Brown",  
6     "totalprice": 111,  
7     "depositpaid": true,  
8     "bookingdates": {  
9       "checkin": "2018-01-01",  
10      "checkout": "2019-01-01"  
11     },  
12     "additionalneeds": "Breakfast"  
13   }  
14 }
```

GET https://restful-booker.herokuapp.com/booking/4470

Body none

This request does not have a body.

200 OK 165 ms 938 B Save Response

Body Cookies Headers (10) Test Results

```
1 {  
2   "firstname": "<script>alert('XSS')</script>",  
3   "lastname": "Brown",  
4   "totalprice": 111,  
5   "depositpaid": true,  
6   "bookingdates": {  
7     "checkin": "2018-01-01",  
8     "checkout": "2019-01-01"  
9   },  
10  "additionalneeds": "Breakfast"  
11 }  
12 }
```

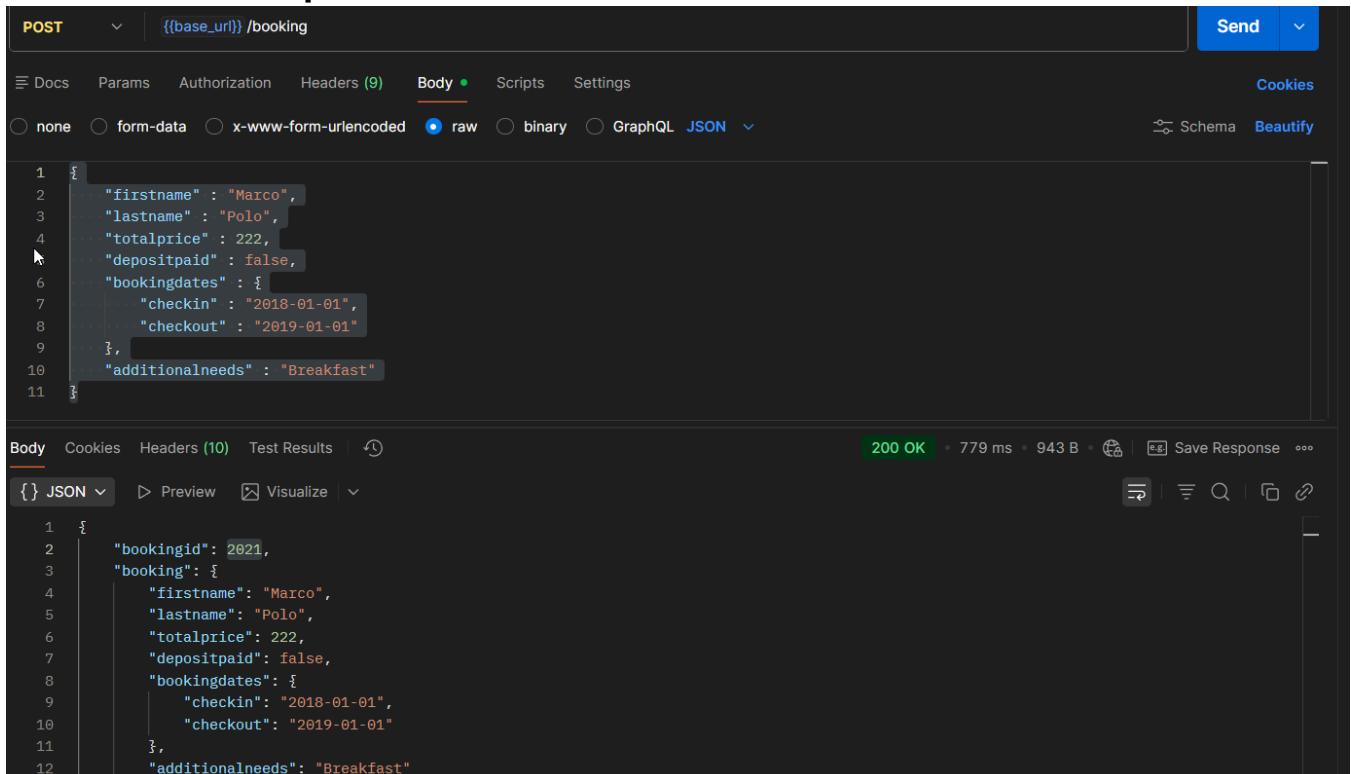
4.3 Access Control

Este bloque verifica que la API proteja correctamente los recursos sensibles y no permita operaciones críticas sin autenticación válida.

3.3.1 Acceso a recurso protegido sin token

IDENTIFICADOR	A-01
DESCRIPCIÓN	<p>Esta prueba valida que la API bloquee el acceso a endpoints protegidos cuando la solicitud no incluye un token de autenticación válido.</p> <p>El objetivo es verificar que operaciones sensibles (modificación o eliminación de recursos) no puedan ejecutarse por usuarios no autenticados.</p>
PRECONDICIONES	<p>Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com</p> <p>API en funcionamiento</p>
SET UP DE DATOS	<p>Body de POST:</p> <pre>{ "firstname" : "Marco", "lastname" : "Polo", "totalprice" : 222, "depositpaid" : false, "bookingdates" : { "checkin" : "2018-01-01", "checkout" : "2019-01-01" }, "additionalneeds" : "Breakfast" }</pre>
PASOS A SEGUIR	<ul style="list-style-type: none"> Enviar request POST URL: {{base_url}}/booking Enviar request DELETE URL: {{base_url}}/booking/(id del POST) sin token ni cookies de autorización
RESULTADO ESPERADO	<ul style="list-style-type: none"> Status code 401 o 403
RESULTADO OBTENIDO	<ul style="list-style-type: none"> Status code 401 o 403
ESTADO	Pass

Evidencias de la prueba:



POST `POST {{base_url}}/booking`

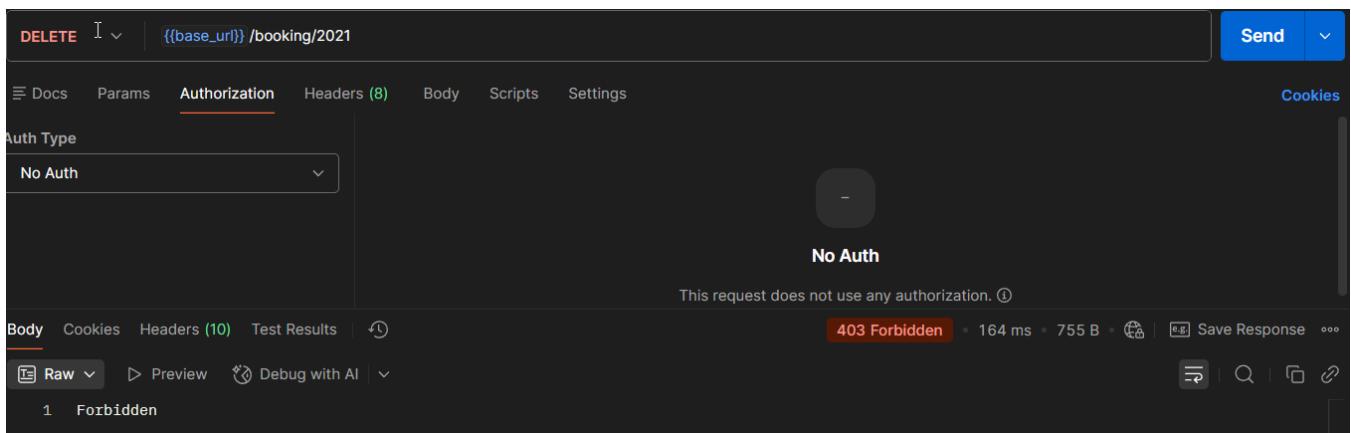
Body `{ "firstname": "Marco", "lastname": "Polo", "totalprice": 222, "depositpaid": false, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast" }`

200 OK | 779 ms | 943 B | Save Response | ...

Body Cookies Headers (10) Test Results | ↻

{ } JSON | Preview | Visualize | ↻

```
1 {  
2   "bookingid": 2021,  
3   "booking": {  
4     "firstname": "Marco",  
5     "lastname": "Polo",  
6     "totalprice": 222,  
7     "depositpaid": false,  
8     "bookingdates": {  
9       "checkin": "2018-01-01",  
10      "checkout": "2019-01-01"  
11    },  
12    "additionalneeds": "Breakfast"
```



DELETE `DELETE {{base_url}}/booking/2021`

Authorization | Headers (8)

No Auth

No Auth

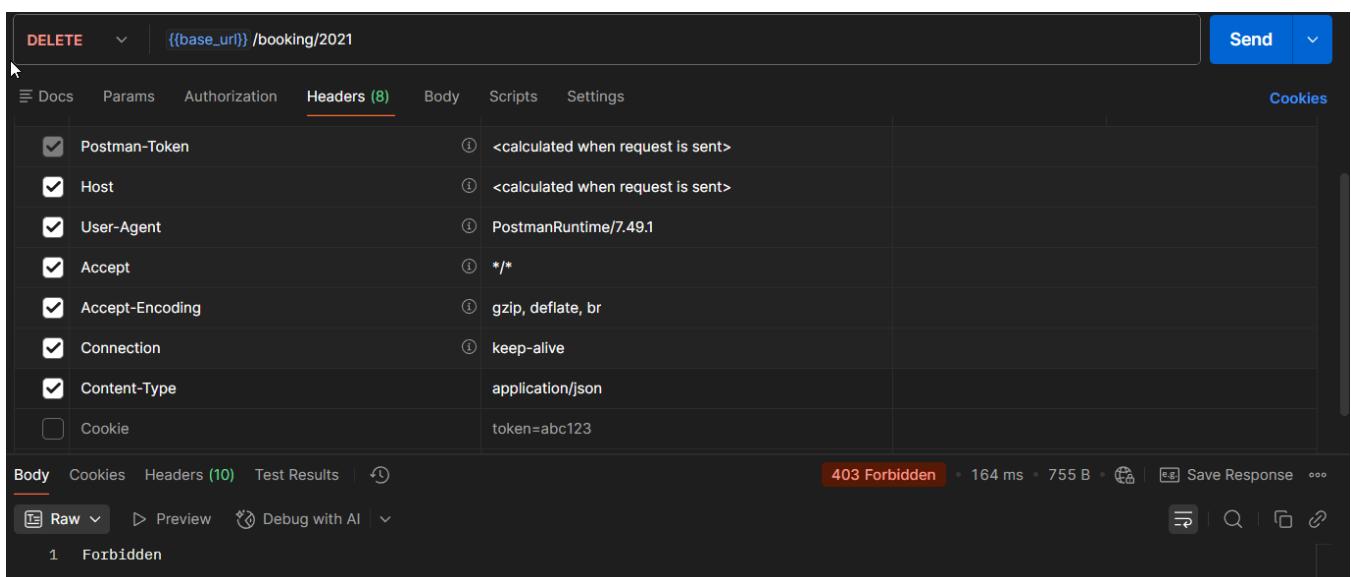
This request does not use any authorization. ⓘ

403 Forbidden | 164 ms | 755 B | Save Response | ...

Body Cookies Headers (10) Test Results | ↻

Raw | Preview | Debug with AI | ↻

1 Forbidden



DELETE `DELETE {{base_url}}/booking/2021`

Headers (8) | Body | Scripts | Settings

Header	Value
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.49.1
Accept	/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Content-Type	application/json
Cookie	token=abc123

403 Forbidden | 164 ms | 755 B | Save Response | ...

Body Cookies Headers (10) Test Results | ↻

Raw | Preview | Debug with AI | ↻

1 Forbidden

3.3.2 Acceso a recurso protegido con token inválido

IDENTIFICADOR	A-02
DESCRIPCIÓN	<p>Esta prueba valida que la API bloquee el acceso a endpoints protegidos cuando la solicitud incluye tokens de autorización inválidos.</p> <p>El objetivo es verificar que operaciones sensibles (modificación o eliminación de recursos) no puedan ejecutarse por usuarios no autenticados.</p>
PRECONDICIONES	<p>Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com</p> <p>API en funcionamiento</p>
SET UP DE DATOS	<p>Body de POST:</p> <pre>{ "firstname" : "Marco", "lastname" : "Polo", "totalprice" : 222, "depositpaid" : false, "bookingdates" : { "checkin" : "2018-01-01", "checkout" : "2019-01-01" }, "additionalneeds" : "Breakfast" }</pre> <p>Cookie en DELETE request: token=invalid123456fake</p>
PASOS A SEGUIR	<ul style="list-style-type: none"> Enviar request POST URL: {{base_url}}/booking Enviar request DELETE URL: {{base_url}}/booking/(id del POST) con cookie token inválido
RESULTADO ESPERADO	<ul style="list-style-type: none"> Status code 401 o 403
RESULTADO OBTENIDO	<ul style="list-style-type: none"> Status code 401 o 403
ESTADO	Pass

Evidencias de la prueba:

The screenshot shows a POST request to `https://{{base_url}}/booking`. The 'Post-response' script contains the following code:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("verifying booking id", function () {
6     var jsonData = pm.response.json();
7     var bookingId = jsonData.bookingid; // Assuming the bookingid is in the response body
8     pm.environment.set("BookingId", bookingId);
9
10    pm.expect(bookingId).to.not.be.null;
11});
```

The response body is displayed as JSON:

```
1 {
2     "bookingid": 4059,
3     "booking": {
4         "firstname": "Marco",
5         "lastname": "Polo",
6         "totalprice": 222,
7         "depositpaid": false,
8         "bookingdates": {
9             "checkin": "2018-01-01",
10            "checkout": "2019-01-01"
11        },
12        "additionalneeds": "Breakfast"
13    }
14}
```

Test results show 2/2 passed.

The screenshot shows a DELETE request to `https://restful-booker.herokuapp.com/booking/{{BookingId}}`. The 'Headers' tab shows the following configuration:

Key	Value	Description
Accept		
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Content-Type	application/json	
Cookie	token=invalid123456fake	

The response status is 403 Forbidden.

Conclusión Ejecutiva

El presente documento detalla el resultado de las pruebas de testing funcional y de seguridad realizadas sobre la API Restful Booker, con foco en vulnerabilidades comunes de aplicaciones web (OWASP Top 10), particularmente inyecciones SQL (SQLi), validación y codificación de entradas/salidas (Input & Output Handling) y control de accesos.

Durante la ejecución de los casos de prueba relacionados con SQL Injection, no se detectaron vulnerabilidades explotables. En todos los escenarios evaluados (boolean-based, error-based y time-based), la API respondió de manera consistente, sin exponer errores internos, sin variaciones significativas en las respuestas y sin retrasos anómalos en los tiempos de ejecución. Estos resultados indican que, al menos desde el punto de vista observado, el backend no es vulnerable a inyecciones SQL inferenciales ni directas, o bien no ejecuta consultas SQL dinámicas sobre los campos evaluados.

En cuanto al control de accesos, las pruebas confirmaron que los endpoints críticos se encuentran correctamente protegidos. Los intentos de eliminación de recursos sin autenticación o con tokens inválidos fueron bloqueados adecuadamente, devolviendo códigos de estado HTTP coherentes (401/403). Esto demuestra una correcta implementación de los mecanismos básicos de autorización para operaciones sensibles.

No obstante, se identificaron vulnerabilidades críticas en la validación y sanitización de datos de entrada, específicamente relacionadas con HTML Injection y Stored Cross-Site Scripting (XSS). La API permite almacenar y devolver contenido HTML y JavaScript sin ningún tipo de escape o validación, lo cual representa un riesgo significativo para cualquier aplicación cliente que consuma estos datos y los renderice en un contexto web. Estas vulnerabilidades no afectan directamente a la API en aislamiento, pero sí introducen un vector de ataque severo en escenarios reales de integración con frontends o paneles administrativos.

En conclusión, si bien la API presenta un buen comportamiento frente a ataques de inyección SQL y controles de acceso, requiere mejoras urgentes en la validación y codificación de entradas y salidas. Se recomienda implementar sanitización del lado del servidor, validaciones estrictas de modelos de datos, codificado seguro de salida y pruebas automatizadas de seguridad como parte del pipeline de desarrollo. Estas acciones permitirán reducir significativamente la superficie de ataque y alinear el sistema con buenas prácticas de seguridad aplicables a entornos productivos.