

Pruebas de Servicio Web

Módulo 6 – Evaluación

Equipo 4:

Felipe Lobos

Fabiola Díaz

Eduardo Arellano

Carlos Vasquez

Descripción

La empresa ficticia “TechMarket” es un Marketplace digital que permite a pequeños y medianos emprendedores publicar y vender sus productos de manera online. Este sistema dispone de una API REST que permite gestionar los productos, usuarios y pedidos. Actualmente, la empresa se encuentra en un proceso de mejora continua y ha solicitado al equipo de calidad implementar un plan de pruebas automatizadas que permita validar el correcto funcionamiento y seguridad de sus servicios web.

Como parte del equipo de QA, se te ha encomendado realizar una suite de pruebas que integre buenas prácticas de diseño, seguridad, validaciones mediante aserciones y automatización de pruebas utilizando las herramientas vistas en el módulo.

1. API REST utilizada: <https://jsonplaceholder.typicode.com>

2.

- ¿Que es una API REST?

Una API (Application Programming Interface o Interfaz de Programación de Aplicaciones) es la interfaz que nos permitirá conectar diferentes sistemas y aplicaciones para compartir información. Las API REST son una forma estándar de la industria para que los servicios web envíen y reciban datos. Utilizan métodos de solicitud HTTP para facilitar el ciclo de solicitud-respuesta y, por lo general, transfieren datos mediante JSON y, más raramente, HTML, XML u otros formatos. Las principales consultas usadas en una API REST son: **GET** (para obtener datos), **POST** (para crear datos), **PUT** (para actualizar datos), **PATCH** (para actualizar parcialmente datos) y **DELETE** (para eliminar datos)

- **Componentes principales de una API REST**

Las APIs REST siguen una serie de características o principios clave que aseguran simplicidad, escalabilidad y eficiencia en la comunicación entre sistemas

Recursos	Representan los datos o servicios que la API expone. Por ejemplo, /usuarios o /productos.
Puntos finales (Endpoints)	Son las URL que identifican y permiten acceder a los recursos. Actúan como la dirección del recurso en el servidor.
Métodos HTTP	Indican la acción que el cliente desea realizar sobre el recurso. Los más comunes son: <ul style="list-style-type: none">● GET: Para solicitar información o un recurso.● POST: Para crear un nuevo recurso.● PUT: Para actualizar un recurso existente o crearlo si no existe.● DELETE: Para eliminar un recurso● PATCH: Actualizar un determinado campo de un recursos existente.
Cabeceras (Headers)	Son metadatos que proporcionan información adicional sobre la solicitud y la respuesta. Incluyen detalles como el formato de los datos (JSON, XML) o información de autenticación
Cuerpo de la solicitud	Contiene los datos que se envían al servidor, especialmente para las operaciones de creación y actualización (POST, PUT

- **Buenas prácticas de diseño de API REST**

URIs limpias y orientadas a recursos	Las URIs deben representar sustantivos (recursos), no verbos (acciones). Usar nombres en plural para colecciones: /users, /products.
Uso correcto de métodos HTTP	Semántica clara: GET para obtener, POST para crear, PUT para reemplazar, PATCH para modificar parcialmente y DELETE para eliminar.
Estructura jerárquica de recursos	Reflejar relaciones entre recursos para mantener una estructura lógica y navegable.
Utilizar códigos de estado HTTP apropiados	Evitar respuestas genéricas como 200 para todo. Usar códigos específicos para cada situación.
Hacer uso del principio de "stateless"	Cada solicitud debe ser independiente, conteniendo toda la información necesaria para ser procesada.
Utilizar headers correctamente	Ejemplo: Content-Type, Accept, Authorization, etc. para enriquecer la comunicación.
Documentar la API	Preferiblemente con Swagger/OpenAPI, para facilitar el consumo por parte de otros equipos o aplicaciones

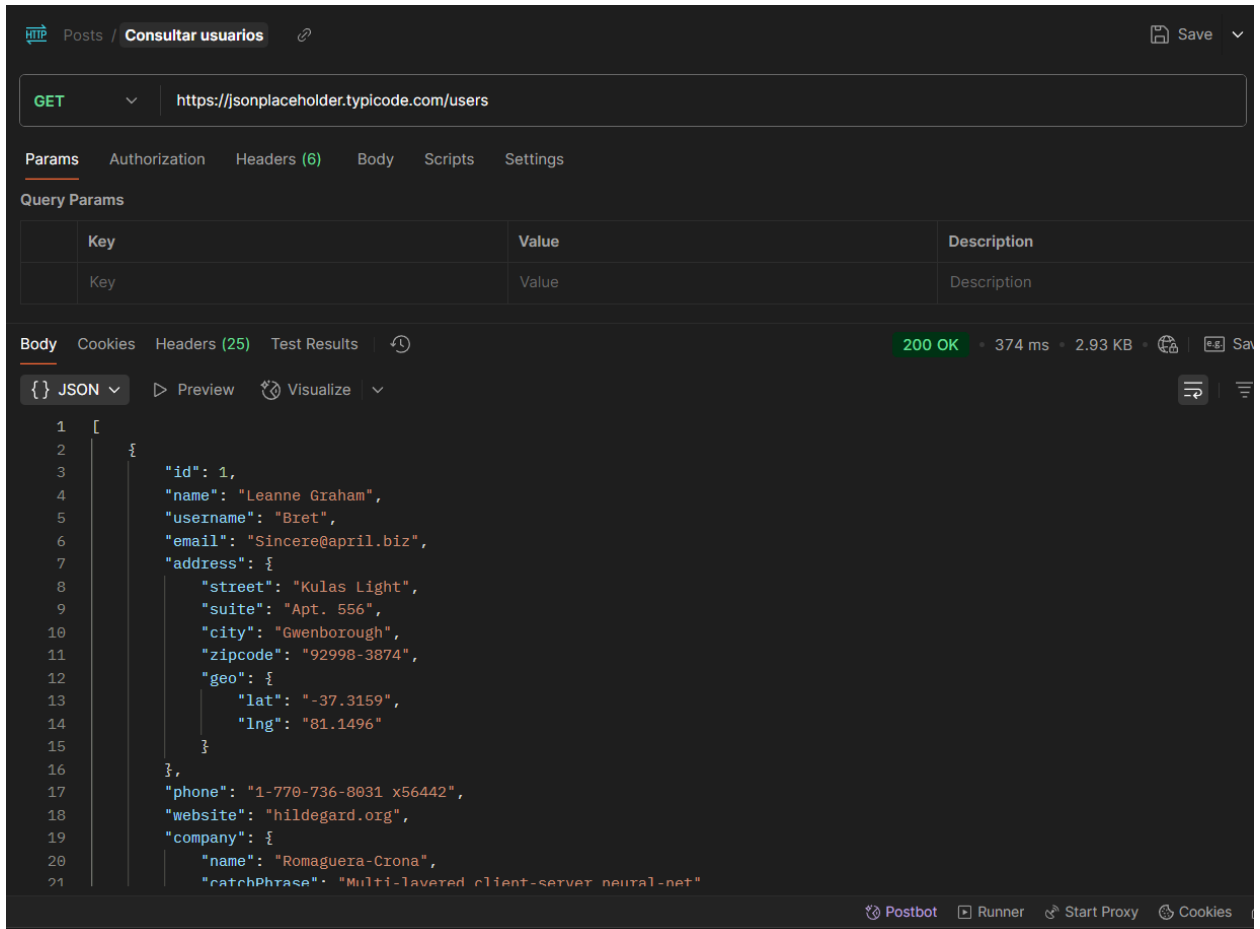
3. Identificar y describir las amenazas de seguridad comunes en APIs y las buenas prácticas para mitigar estos riesgos. Debes implementar al menos un mecanismo de autenticación en las pruebas (API Key, Token, JWT, o Autenticación Básica).

Amenaza	Descripción	Mitigación
Inyecciones de código	Cuando un atacante "engaña" a la API enviando código malicioso disfrazado de datos normales, con el fin de robar o modificar información.	<ul style="list-style-type: none">- Validar y filtrar los datos de entrada- Usar consultas seguras (prepared statements)- Nunca confiar en lo que escribe el usuario
Robo de credenciales o tokens	Cuando un atacante intercepta o adivina la clave o token que se usa para autenticar una sesión.	<ul style="list-style-type: none">- Usar HTTPS siempre- Expirar los tokens después de un tiempo corto- Evitar almacenar claves/token en lugares inseguros
Replay Attack	El atacante graba una solicitud válida y la repite más tarde para obtener los mismos beneficios.	<ul style="list-style-type: none">- Usar tokens de un solo uso (nonces)- Agregar marca de tiempo y verificar su validez- Firmar digitalmente las solicitudes
Falsificación de solicitudes (CSRF)	Un sitio malicioso engaña al usuario para que envíe sin querer una solicitud a otra API donde está autenticado.	<ul style="list-style-type: none">- Usar tokens CSRF únicos por sesión- Verificar origen de la solicitud- Implementar políticas de CORS correctamente
Suplantación de identidad (Spoofing)	Hacerse pasar por otro usuario o sistema para engañar a la API.	<ul style="list-style-type: none">- Múltiples factores de verificación de identidad- Validación del origen de la solicitud- Registrar y analizar comportamientos sospechosos (Logs)
Ataques de denegación de servicio (DoS o DDoS)	Un atacante envía miles de solicitudes en poco tiempo para saturar la API y hacer que deje de funcionar.	<ul style="list-style-type: none">- Limitar la cantidad de solicitudes por IP- Usar servicios de protección como Cloudflare- Monitorear y bloquear comportamientos sospechosos

4. Desarrollar una colección de pruebas manuales utilizando Postman que contemple: o Operaciones básicas (GET, POST, PUT, DELETE).

- Operaciones básicas (GET, POST, PUT, DELETE).

Ejemplo con GET



The screenshot shows the Postman interface with a GET request to `https://jsonplaceholder.typicode.com/users`. The request is successful, returning a 200 OK status with a response time of 374 ms and a body size of 2.93 KB. The response body is displayed in JSON format, showing an array of user objects.

```
1  [
2    {
3      "id": 1,
4      "name": "Leanne Graham",
5      "username": "Bret",
6      "email": "Sincere@april.biz",
7      "address": {
8        "street": "Kulas Light",
9        "suite": "Apt. 556",
10       "city": "Gwenborough",
11       "zipcode": "92998-3874",
12       "geo": {
13         "lat": "-37.3159",
14         "lng": "81.1496"
15       }
16     },
17     "phone": "1-770-736-8031 x56442",
18     "website": "hildegard.org",
19     "company": {
20       "name": "Romaguera-Crona",
21       "catchPhrase": "Multi-layered client-server neural-net"
```

Ejemplo con POST

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** `https://jsonplaceholder.typicode.com/users/1/posts`
- Body (raw):**

```
{  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit2",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum :"
```
- Response:** 201 Created, 149 ms, 1.51 KB
- Body (JSON):**

```
{  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit2",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum",  "userId": "1",  "id": 101}
```

Ejemplo con PUT

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** `https://jsonplaceholder.typicode.com/posts/100`
- Body (raw):**

```
{  "title": "otro titulo para esto",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"}
```
- Response:** 200 OK, 217 ms, 1.47 KB
- Body (JSON):**

```
{  "title": "otro titulo para esto",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum",  "id": 100}
```


Ejemplo con DELETE

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** https://jsonplaceholder.typicode.com/posts/100
- Body:**

```
{  "title": "otro titulo para esto",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"}
```
- Response:** 200 OK, 419 ms, 1.12 KB
- Body:**

```
{}
```

- Validación de estructura del cuerpo de respuesta (JSON).

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** https://jsonplaceholder.typicode.com/posts/1
- Scripts:**

```
1 // Validar que la respuesta sea JSON
2 pm.test("El Content-Type es JSON", function () {
3   pm.response.to.have.header("Content-Type");
4   pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
5 });
6
7 // Validar estructura JSON
8 pm.test("La respuesta tiene la estructura esperada", function () {
9   const jsonData = pm.response.json();
10
11   pm.expect(jsonData).to.have.property("userId");
12   pm.expect(jsonData).to.have.property("id");
13   pm.expect(jsonData).to.have.property("title");
14   pm.expect(jsonData).to.have.property("body");
15 });
16
```
- Response:** 200 OK, 121 ms, 1.31 KB
- Body:**

```
{  "userId": 1,  "id": 1,  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"}
```

- Ejemplo Validación de códigos de estado HTTP.



The screenshot shows the Postman interface for a GET request to the endpoint `{{BASE_URL}} /todos`. The 'Scripts' tab is selected, showing a pre-request script that asserts the response status is 200 OK. The interface includes tabs for Params, Authorization, Headers (6), Body, Scripts, and Settings. The left sidebar shows the Pre-request and Post-response sections.

```
1 pm.test('200 OK', () => pm.response.to.have.status(200));
2
3
4
```

- Ejemplo Aserciones de contenido específicas.



The screenshot shows the Postman interface for a POST request to the endpoint `{{BASE_URL}} /todos`. The 'Scripts' tab is selected, showing a pre-request script and a post-response script that asserts the response status is 201 Created, the title is 'Nuevo Todo en español', the userId is 3, and the completed status is false. The interface includes tabs for Params, Authorization, Headers (8), Body, Scripts, and Settings. The left sidebar shows the Pre-request and Post-response sections.

```
1 pm.test('201 Created', () => pm.response.to.have.status(201));
2
3 pm.test("Title igual a 'Nuevo Todo en español'", function () {
4   const responseData = pm.response.json();
5   pm.expect(responseData.title).to.eql('Nuevo Todo en español');
6 });
7 pm.test("userId igual a 3", function () {
8   const responseData = pm.response.json();
9   pm.expect(responseData.userId).to.eql(3);
10 });
11 pm.test("completed igual a false", function () {
12   const responseData = pm.response.json();
13   pm.expect(responseData.completed).to.eql(false);
14 });
```

5. Automatizar la ejecución de las pruebas desarrolladas en Postman mediante Newman, generando un reporte de resultados.

Newman Report

Collection	Modular 6
Description	Usando la url : https://jsonplaceholder.typicode.com/
Time	Mon Aug 25 2025 23:16:54 GMT-0400 (hora estándar de Chile)
Exported with	Newman v6.2.1

	Total	Failed
Iterations	1	0
Requests	5	0
Prerequisite Scripts	5	0
Test Scripts	7	0
Assertions	3	0

Total run duration	1262ms
Total data received	2.28KB (approx)
Average response time	158ms

Total Failures 0

6. Implementar una suite de pruebas automatizadas utilizando Rest Assured (Java)

```
INFO] -----
INFO] T E S T S
INFO] -----
INFO] Running cl.kibernet.TechMarket.ApiTest
INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.078 s -- in cl.kibernet.TechMarket.ApiTest
INFO] Running cl.kibernet.TechMarket.test.PostCrudTest
Request URI: https://jsonplaceholder.typicode.com/posts
Request URI: https://jsonplaceholder.typicode.com/posts/1
Request URI: https://jsonplaceholder.typicode.com/posts
Request URI: https://jsonplaceholder.typicode.com/posts/1
Request URI: https://jsonplaceholder.typicode.com/posts/101
INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.980 s -- in cl.kibernet.TechMarket.test.PostCrudTest
INFO] Running cl.kibernet.TechMarket.test.TODOCrudTest
Request URI: https://jsonplaceholder.typicode.com/todos
Request URI: https://jsonplaceholder.typicode.com/todos/1
Request URI: https://jsonplaceholder.typicode.com/todos
INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.674 s -- in cl.kibernet.TechMarket.test.TODOCrudTest
INFO]
INFO] Results:
INFO]
INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0
```

Análisis de Resultados (para API pública)

Pruebas de recursos públicos

- Todos los endpoints probados respondieron correctamente con código 200 OK.
- No se requirió autenticación, ya que la API es de acceso público.

Validaciones de cabeceras

- El Content-Type de las respuestas fue application/json, lo cual confirma que la API entrega datos en el formato esperado.

Validación del cuerpo de respuesta

- La estructura del JSON fue validada, comprobando que incluyera los campos esperados
- Los datos devueltos fueron consistentes con la documentación de la API.

Seguridad

- la API no cuenta con mecanismos de autenticación ni autorización
- Esto facilita el acceso, pero representa riesgos si la API maneja información sensible.
- Se recomienda, como buena práctica, implementar al menos un mecanismo de seguridad (ejemplo: API Key o JWT) en caso de exponer datos críticos.

Conclusión

Con este trabajo pudimos comprobar que la API responde bien a las operaciones principales (GET, POST, PUT, DELETE), entrega datos en JSON y los códigos de estado de manera correcta. También probamos cabeceras como Content-type y la autenticación en endpoints protegidos, lo que me permitió entender mejor cómo se maneja la seguridad en estos casos.

El uso de Postman, Rest Assured y las librerías de aserciones nos ayudó a automatizar pruebas y asegurarnos de que todo funcionara de forma confiable. Aprendimos que no basta con probar que una API funciona, también hay que considerar la seguridad, la integridad de los datos y que siga buenas prácticas REST.

En resumen, la automatización de pruebas aporta mucho valor porque permite detectar fallos temprano y da más confianza en el sistema.

Recomendaciones de mejora para el sistema probado

- Usar métodos de autenticación más seguros
- Definir respuestas de error claras y con códigos HTTP correctos.
- Implementar HTTPS siempre y considerar límites de peticiones para evitar abusos.
- Agregar monitoreo y logs para tener más control sobre el uso de la API.
- Mantener buenas prácticas REST, como versionar la API y nombrar bien los recursos.