«Talento Tech»

Testing QA

Clase 09

Buenos Aires Ciudad

Clase N°9 - Pruebas de API 1

Temario

- Introducción a formatos de datos: JSON vs XML
- Estructura y sintaxis de JSON
- Validación de esquemas JSON
- Manipulación y evaluación de respuestas JSON

Objetivos de la clase

En esta clase vamos a adentrarnos en el mundo de las APIs y entender por qué son tan importantes en el desarrollo de software actual. Nos enfocaremos en los dos formatos de datos más utilizados en las comunicaciones entre sistemas: JSON y XML. Veremos cómo se estructuran, cómo se validan y cómo podemos manipular sus respuestas para comprobar que una API está funcionando correctamente. Además, aprenderemos a interpretar y validar respuestas en formato JSON, algo clave en el día a día de un tester de calidad.

¿Qué es una API?

Una API (Application Programming Interface) es una interfaz que permite que dos aplicaciones se comuniquen entre sí. Es como un "mozo digital" que recibe pedidos, se los pasa a la cocina (el servidor), y luego trae la respuesta al cliente (la app o sistema que lo pidió).

Ejemplo simple:
Estás en una app para pedir comida. Vos tocás el botón de “ver menú” → eso genera un pedido que va por una API al servidor → el servidor responde con los platos disponibles → la app te muestra el menú.

Formatos de datos más comunes: JSON vs XML

Las respuestas de las APIs no vienen en lenguaje humano, sino en formatos estructurados que las computadoras entienden. Los dos más comunes son:

JSON

JSON (JavaScript Object Notation) es un formato de texto ligero para intercambiar datos entre sistemas. Se usa mucho en APIs y bases de datos porque es fácil de leer y escribir para los humanos, y fácil de procesar para las máquinas.

Características de JSON:

- Basado en texto: Los datos se guardan en un archivo de texto con extensión .json.
- Ligero y rápido: No tiene tanta sobrecarga como XML.
- Estructura sencilla: Utiliza pares clave-valor organizados en objetos y listas.
- Independiente de lenguaje: No depende de ningún lenguaje en particular (aunque proviene de JavaScript).

Es el formato más utilizado actualmente. Es liviano, fácil de leer y de procesar.

Ejemplo de respuesta JSON:

```
{
   "usuario": "juan123",
   "activo": true,
   "reservas": [
     {
       "hotel": "La Posada",
       "fecha": "2024-04-12",
       "precio": 150.75
     }
   ]
 }
```

Validación de JSON

Antes de usar un JSON, es importante verificar que su sintaxis sea correcta. Para ello, podemos usar herramientas como:

- JSONLint (https://jsonlint.com/)
- Postman (para probar APIs que devuelven JSON)

Si el JSON tiene errores, la validación nos dirá qué corregir.

JSONLint

JSONLint es una herramienta gratuita y en línea que permite **validar y formatear archivos JSON**. Su objetivo es detectar **errores de sintaxis** y mejorar la legibilidad del JSON, asegurando que la estructura sea correcta antes de utilizarlo en aplicaciones o pruebas de API.

¿Cómo se usa JSONLint?

1) **Acceder a JSONLint**
   Puedes usar JSONLint en línea desde https://jsonlint.com.
2) **Pegar o cargar un JSON**

- Copia tu JSON y pégalo en el cuadro de texto de JSONLint.
- También puedes subir un archivo `.json`.

3) **Hacer clic en "Validate JSON"**

   JSONLint analizará el JSON y mostrará:
   ✔ **"Valid JSON"** si la estructura es correcta.
   ❌ **Errores de sintaxis** si hay comas de más, llaves mal cerradas, etc.

Estructura y sintaxis de JSON

Un JSON se construye con:

- Llaves `{}` para objetos
- Corchetes `[]` para listas
- Pares clave-valor `"clave": valor`

Ejemplo completo:

```
{
 "usuario": {
   "nombre": "Juan",
   "edad": 30
 },
 "habilidades": ["testing", "python", "scrum"],
 "activo": true
}
```

Errores comunes al validar JSON:

- Olvidar una coma `,` entre elementos
- No usar comillas en las claves
- Formato de fechas o booleanos incorrecto

Validación de esquemas JSON

Validar un esquema JSON significa verificar que la estructura del dato que recibimos es la que esperábamos.

Ejemplo:

Esperamos recibir:

```
{
 "usuario": "string",
 "edad": integer,
 "activo": boolean
}
```

Si en cambio llega algo así:

```
{
 "usuario": "juan123",
 "edad": "treinta",
 "activo": "sí"
}
```

¿Qué está mal?

"edad": "treinta"

El valor "treinta" es un string (texto), no un número.

- Las APIs que esperan un número deben recibir algo como 30 (en números y sin comillas).
- Si se envía "treinta" (texto), puede causar errores al intentar hacer cálculos o validaciones con ese dato.

"activo": "sí"

"sí" es también un string, pero el campo "activo" espera un booleano, es decir:

- true si el usuario está activo
- false si no lo está
- Usar "sí" o "no" no es válido en un campo booleano.
- Algunos lenguajes ni siquiera entienden "sí" como booleano, lo cual puede hacer que la aplicación se rompa o no procesa bien el dato.

¿Por qué esto importa?

Cuando los datos no cumplen con el tipo esperado, pueden ocurrir varios problemas:

❌ Errores de validación
❌ Caídas en la app por conversiones incorrectas
❌ Filtros que no funcionan (por ejemplo, al buscar solo usuarios activos)
❌ Incompatibilidad entre servicios que usan esos datos

XML

XML (**Extensible Markup Language**) es un **formato de texto** que estructura datos de forma jerárquica usando **etiquetas**. Se utiliza para el **intercambio de información** entre sistemas, bases de datos y APIs.

Aunque hoy JSON es más popular en APIs, **XML sigue siendo usado en sistemas antiguos, configuraciones y servicios web**.

Características de XML

- **Autodescriptivo:** Usa etiquetas para definir los datos.
- **Estructurado y jerárquico:** Se organiza en nodos padre e hijo.
- **Extensible:** No tiene un conjunto fijo de etiquetas, puedes definir las tuyas.
- **Legible por humanos y máquinas:** Puede ser interpretado por distintos sistemas.
- **Compatible con múltiples tecnologías:** Soportado por bases de datos, APIs, y documentos de configuración.

Sintaxis de XML

XML utiliza una estructura basada en **etiquetas**, similar a HTML, pero con la diferencia de que las etiquetas en XML **no están predefinidas**.

Claves importantes:

- Todas las etiquetas deben cerrarse correctamente.
- No hay etiquetas predefinidas: Puedes definir las que necesites.
- **Es sensible a mayúsculas y minúsculas:** `<Nombre>` y `<nombre>` son diferentes.

Validación de XML

XMLLint que funcionan de manera similar a JSONLint.

- Disponible en línea en: https://xmllint.com
- Permite **validar XML** y verificar si cumple con su estructura.

Ejemplo de respuesta XML:

```
<usuario>
<nombre>juan123</nombre>
<activo>true</activo>
<reservas>
 <reserva>
   <hotel>La Posada</hotel>
   <fecha>2024-04-12</fecha>
   <precio>150.75</precio>
 </reserva>
</reservas>
</usuario>
```

¿Cuál es mejor?

- JSON: Más común, más simple, más rápido → ideal para APIs modernas.
- XML: Más robusto en validaciones complejas → usado en integraciones con sistemas antiguos o bancarios.

¡Otro día en Talento Lab!

Hoy Silvia y Matías te presentan una nueva etapa del proyecto: **la integración de servicios mediante APIs**. El equipo de desarrollo está avanzando en funcionalidades clave que requieren intercambiar datos entre diferentes módulos de Talento Lab, como el **registro de usuarios**, la **carga de CVs** y la **asignación automática de propuestas laborales**.

Silvia, como Product Owner, te explica que ahora es fundamental validar que las APIs estén respondiendo correctamente y con los datos esperados, ya que serán el puente entre distintos sistemas.

"Los servicios van a ser el corazón de la plataforma. Si fallan, el usuario no va a poder cargar su CV o ver sus postulaciones. Necesitamos que QA nos garantice que cada respuesta tenga la estructura correcta."

Matías, como Tester Senior, te muestra una colección de respuestas que el equipo backend está generando en formato **JSON**, y te desafía a empezar a validar si cumplen con el **esquema esperado**. También te comparte una herramienta de validación online, y te pide que verifiques si todos los campos están presentes y si los valores devueltos son correctos.

"Tu tarea ahora es revisar cada respuesta JSON y ver si cumple con el contrato que definimos. Pensá que si un campo falta o viene con un tipo incorrecto, puede romper toda la experiencia del usuario. Acá empieza el trabajo fino del tester con APIs."

A partir de ahora, como Analista de Calidad Trainee, tu nuevo desafío es entender el lenguaje de las APIs, analizar sus respuestas, y aprender a **verificar datos estructurados** como lo haría un detector de errores profesional.

Ejercicio práctico

Vamos a analizar la siguiente respuesta JSON que podría venir de la API de Talento Lab:

```
{
   "nombre": "María García",
   "documento": "33222444",
   "carrera": "QA Tester",
   "carga_cv": true
 }
```

Ejercicio 1: Verificá si cumple con este esquema:

```
{
 "nombre": "string",
 "documento": "string",
 "carrera": "string",
 "carga_cv": "boolean"
}
```

Ejercicio 2: Crear un JSON válido para representar un pedido de comida

Construye un JSON que represente una reserva en un hotel con los siguientes elementos:

- **Cliente:** Nombre, teléfono y correo electrónico.
- **Reserva:** Lista de habitaciones con número, tipo y precio por noche.
- **Duración de la estancia:** Fecha de entrada y salida.
- **Total:** Precio total de la estadía.

Próximos Pasos

En la próxima clase nos meteremos de lleno en el mundo de las **Pruebas de Servicios y Arquitectura de APIs**. Vamos a ver qué son los microservicios, qué tipos de APIs existen, y cómo los testers se enfrentan a estas pruebas en la práctica.

Preguntas Disparadoras

1. ¿Cuáles son las ventajas de JSON sobre XML en el contexto de APIs?
2. ¿Por qué es importante validar la estructura de una respuesta JSON?
3. ¿Cómo se pueden detectar errores en un JSON recibido de una API?
4. ¿Qué herramientas recomendarías para la validación de JSON y por qué?

Material Complementario

- Documentación oficial de JSON Schema: https://json-schema.org
- Introducción a JSON en MDN: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/JSON
- Guía de pruebas de API con Postman: https://learning.postman.com/docs/getting-started/introduction/


Buenos Aires
aprende
Agencia de Habilidades para el Futuro
Buenos
Aires
Ciudad