TECNICATURA SUPERIOR EN

## Telecomunicaciones

# Proyecto Integrador I

Rest Ful APIs – parte II

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Dirección General de EDUCACIÓN TÉCNICA Y FORMACIÓN PROFESIONAL
Ministerio de EDUCACIÓN
CÓRDOBA entre todos

# Introducción a JSON

- JSON, que significa JavaScript Object Notation, es un formato de intercambio de datos que ha ganado una inmensa popularidad en los últimos años. Originalmente derivado de JavaScript, hoy en día se ha independizado y es ampliamente utilizado en muchos lenguajes de programación para la transmisión de datos.
- JSON se desarrolló con el objetivo de tener un formato de texto que sea fácil de leer y escribir para los humanos y fácil de analizar y generar para las máquinas. La simplicidad y la eficiencia de JSON han hecho que se convierta en el formato preferido para el intercambio de datos en la web, superando a otros formatos como XML en muchos aspectos.
- En las siguientes diapositivas, exploraremos la estructura y sintaxis de JSON, su uso en la web y cómo se compara con otros formatos de intercambio de datos. Además, veremos cómo podemos trabajar con JSON en varios lenguajes de programación y discutiremos algunas de las consideraciones de seguridad que deben tenerse en cuenta al usar JSON. Finalmente, realizaremos un ejemplo práctico con Python, donde crearemos, leeremos y modificaremos un archivo JSON.

- Pasemos ahora a la estructura de JSON. JSON es muy simple en términos de los tipos de datos que soporta: números, cadenas de texto, booleanos (verdadero o falso), arreglos (o listas), objetos (que son básicamente mapas o diccionarios), y el valor null.
- Un aspecto importante de JSON es su sintaxis, que es bastante simple y fácil de entender. Los datos en JSON se representan en pares de clave-valor, donde la clave es una cadena de texto y el valor puede ser cualquiera de los tipos de datos que acabamos de mencionar. Los pares de clave-valor se agrupan en objetos, que se delimitan por llaves { }. Por otro lado, los arreglos son una lista ordenada de valores, y se delimitan por corchetes [ ]. Un objeto puede contener otros objetos o arreglos, permitiendo la creación de estructuras de datos complejas.
- Por ejemplo, un objeto JSON que representa a una persona podría incluir claves para el nombre, la edad y los pasatiempos de la persona. Los pasatiempos podrían ser a su vez un arreglo de cadenas.

# persona.json

persona.json > ...

```json
{
    "nombre": "Juan",
    "edad": 30,
    "esEstudiante": false,
    "hobbies": [
        "fútbol",
        "lectura",
        "cocina"
    ],
    "direccion": {
        "calle": "Avenida Siempre Viva",
        "numero": 742,
        "ciudad": "Springfield"
    },
    "mascotas": null
}
```

# Comparación con otros formatos de intercambio de datos

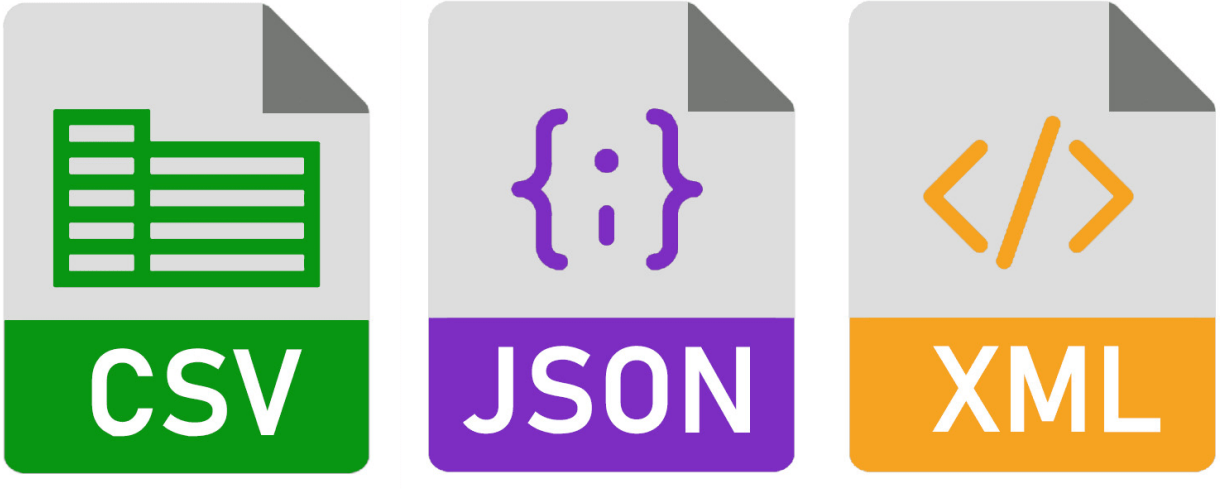
- **XML vs JSON:**

XML es un lenguaje de marcado que almacena información en una estructura anidada y etiquetada, lo que puede dar lugar a archivos voluminosos y más complejos de analizar. JSON es un formato de datos ligero que utiliza una sintaxis parecida a los objetos de JavaScript, siendo más sencillo de escribir y leer. No requiere el cierre de etiquetas, reduciendo así el tamaño del archivo.

- **CSV vs JSON:**

CSV es un formato simple que almacena datos tabulares. Es útil para grandes conjuntos de datos, pero carece de la capacidad de representar estructuras complejas o anidadas. JSON puede representar datos complejos con varias capas de anidación, siendo más versátil, pero puede resultar en archivos más grandes para conjuntos de datos de tamaño considerable.

Ambos XML y CSV tienen sus usos, dependiendo del contexto. XML es útil cuando se necesita un lenguaje de marcado fuerte y estructurado, mientras que CSV es una excelente opción para datos tabulares. Sin embargo, JSON ha ganado popularidad por su sencillez y la capacidad de representar estructuras de datos complejas de una manera fácilmente legible tanto para humanos como para máquinas. La elección del formato de intercambio de datos depende en gran medida de las necesidades y restricciones específicas del proyecto.



- Uso de **JSON en las APIs REST**: JSON se ha convertido en el formato estándar para el intercambio de datos en las APIs REST. Sus ventajas incluyen la facilidad de uso y la capacidad de representar estructuras de datos complejas.
- JSONP para evitar **la política del mismo origen**: JSONP o "JSON con padding" es una técnica utilizada para eludir la política del mismo origen en las aplicaciones web. Permite a un sitio web solicitar datos de otro sitio web en el mismo dominio.

- Las APIs REST son una arquitectura de software ampliamente utilizada en el desarrollo web. Utilizan HTTP y se basan en un conjunto de convenciones para crear, leer, actualizar y eliminar datos. JSON es ideal para este propósito debido a su simplicidad y flexibilidad.
- La política del mismo origen es una medida de seguridad implementada en los navegadores web para prevenir que los scripts de un sitio web accedan a datos de otro sitio web. JSONP es una solución a este problema. Aunque tiene sus riesgos de seguridad y se considera algo anticuado, todavía se usa en algunos casos.

- Cómo trabajar con **JSON en JavaScript**: El lenguaje de programación JavaScript tiene soporte nativo para JSON. Esto incluye la creación, lectura, modificación y eliminación de datos JSON.
- Los métodos JSON.stringify() y JSON.parse(): JavaScript proporciona dos métodos útiles para trabajar con JSON. JSON.stringify() convierte un objeto o valor de JavaScript en una cadena JSON. JSON.parse() convierte una cadena JSON en un objeto o valor de JavaScript.

- Trabajar con JSON en JavaScript es bastante directo gracias a las funciones incorporadas del lenguaje. Por ejemplo, podemos crear un objeto JSON simplemente declarando un objeto literal en nuestro código.
- Los métodos JSON.stringify() y JSON.parse() son esenciales para trabajar con JSON en JavaScript. JSON.stringify() es útil cuando necesitamos enviar datos JSON a un servidor o guardarlos en un archivo. Convierte un objeto JavaScript en una cadena JSON que puede ser transmitida o almacenada. Por otro lado, JSON.parse() es útil cuando recibimos datos JSON de un servidor o de un archivo. Toma una cadena JSON y la convierte en un objeto JavaScript que podemos usar en nuestro código.

- Aunque hemos hablado mucho sobre JSON en el contexto de JavaScript, la realidad es que JSON es un formato de intercambio de datos que se utiliza en muchos **otros lenguajes** de programación. De hecho, su nombre, que significa "JavaScript Object Notation", es un poco engañoso, ya que JSON se utiliza ampliamente fuera del contexto de JavaScript.
- Python, por ejemplo, tiene soporte nativo para JSON a través del módulo json. Esta biblioteca proporciona funciones como json.load() y json.dump() para leer y escribir datos JSON.
- Java, por otro lado, no tiene soporte nativo para JSON, pero existen varias bibliotecas populares que lo proporcionan, como la biblioteca Jackson. Esta biblioteca es muy poderosa y proporciona una amplia gama de funcionalidades para trabajar con JSON.
- Además de Python y Java, muchos otros lenguajes de programación tienen soporte para JSON, ya sea nativamente o a través de bibliotecas. Esto incluye lenguajes como Ruby, PHP, C#, y muchos más. Este amplio soporte para JSON en muchos lenguajes de programación es una de las razones por las que se ha convertido en un estándar para el intercambio de datos en la web.

- Aunque JSON es una herramienta muy útil, también es importante tener en cuenta sus posibles **vulnerabilidades de seguridad**. Al igual que con cualquier tecnología, debemos ser conscientes de estas vulnerabilidades y tomar medidas para mitigarlas.
- Una de las vulnerabilidades más notables asociadas con JSON es la inyección de JSON. Similar a la inyección SQL, la inyección de JSON ocurre cuando un atacante puede introducir datos maliciosos en una aplicación a través de la entrada de JSON. Para mitigar esta vulnerabilidad, es esencial que todas las entradas del usuario se validen y se desinfecten correctamente antes de ser procesadas.
- Otra vulnerabilidad potencial surge con las funciones de revivificación y replacer en JavaScript. Estas funciones pueden ser usadas para transformar los valores de un objeto JSON durante el proceso de serialización y deserialización. Si se utilizan incorrectamente, pueden presentar oportunidades para un ataque. Para mitigar esta vulnerabilidad, se deben seguir las prácticas recomendadas para estas funciones, como no usar la función eval() para revivificar datos JSON.
- En general, es importante recordar que la seguridad es un aspecto esencial del trabajo con cualquier tecnología, y JSON no es una excepción. Siempre debemos ser conscientes de las posibles vulnerabilidades y tomar medidas para mitigarlas.

Ahora vamos a ver cómo podemos trabajar con JSON en Python. Primero, vamos a crear un archivo JSON con algunos datos. Después, vamos a leer el archivo JSON y finalmente, vamos a modificar los datos del archivo JSON.

Para la creación del archivo JSON, utilizamos la función json.dump(), que toma dos argumentos: los datos a escribir y el archivo donde se escribirán los datos.

Para leer el archivo JSON, usamos la función json.load(), que toma como argumento el archivo a leer.

Finalmente, para modificar el archivo JSON, primero modificamos los datos en Python y luego escribimos los datos modificados de nuevo en el archivo utilizando json.dump().

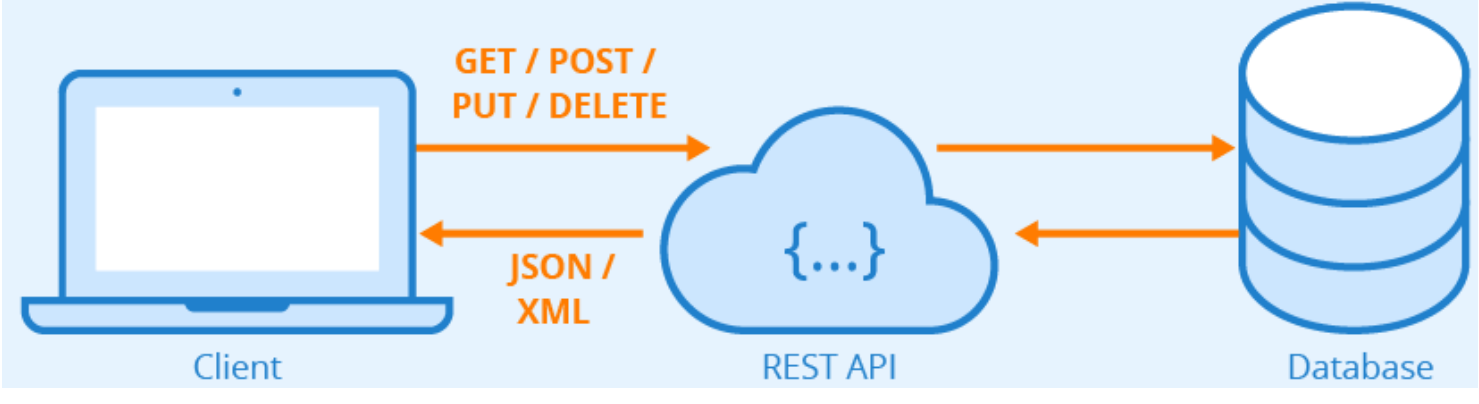
Este ejemplo nos da una idea de cómo podemos trabajar con JSON en Python. Como siempre, es importante recordar que debemos seguir las prácticas recomendadas de seguridad cuando trabajamos con archivos JSON.

datoJson.py > ...

```python
import json

# Creación de un archivo JSON
data = {
    "nombre": "Juan",
    "edad": 30,
    "ciudad": "Buenos Aires"
}
with open('datos.json', 'w') as file:
    json.dump(data, file)

# Lectura de un archivo JSON
with open('datos.json', 'r') as file:
    data = json.load(file)
print(data)

# Modificación de un archivo JSON
data["edad"] = 31
with open('datos.json', 'w') as file:
    json.dump(data, file)

# Verificamos la modificación
with open('datos.json', 'r') as file:
    data = json.load(file)
print(data)

```

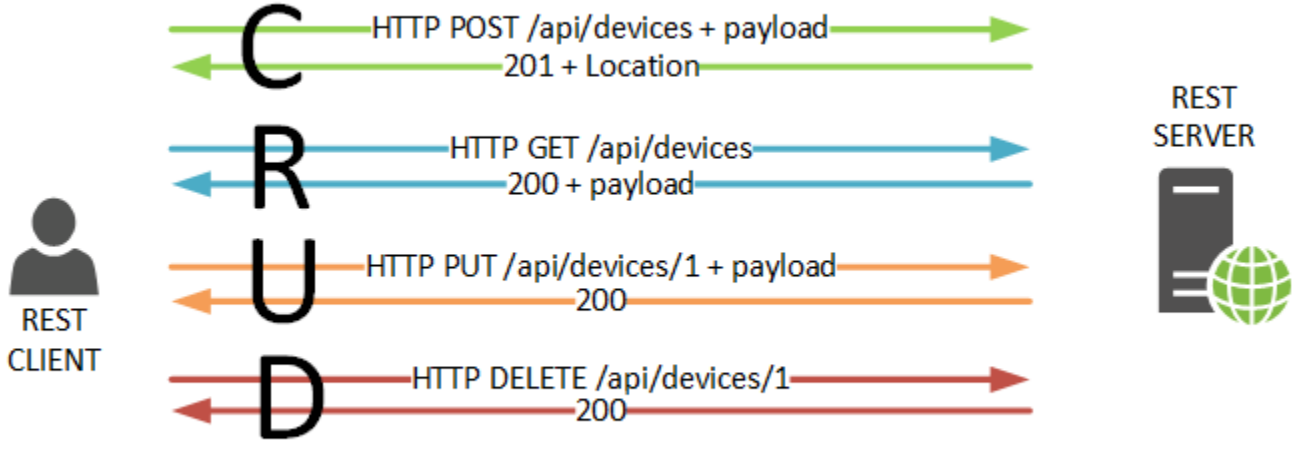
# Introducción a REST

REST (Representational State Transfer) es un estilo de arquitectura de software para sistemas hipermedia distribuidos, como la web. Fue propuesto por Roy Fielding en su tesis doctoral en el año 2000. REST no es un estándar o protocolo, sino un conjunto de restricciones o principios que, cuando se siguen, permiten construir sistemas que escalan a las necesidades de la web. Uno de los principios clave de REST es la "interfaz uniforme", lo que significa que todas las interacciones con recursos en un sistema RESTful deben ser realizadas de una manera consistente.



- REST es un estilo arquitectónico muy popular para diseñar servicios web debido a su simplicidad y eficiencia. Al proporcionar una interfaz uniforme para la interacción con los recursos, REST permite que los servicios web sean fácilmente utilizados por una amplia gama de clientes, incluyendo navegadores web, aplicaciones móviles, y otros servicios web.
- Los principios clave de REST incluyen la identificación de recursos, la manipulación de recursos a través de representaciones, autodescripción de mensajes, y la conexión hipermedia entre recursos. Cuando hablamos de "interfaz uniforme", nos referimos a un conjunto limitado de métodos bien definidos que se utilizan para manipular los recursos - típicamente los métodos HTTP como GET, POST, PUT, y DELETE.
- Es importante notar que aunque REST se basa en HTTP, no está limitado a él. Puedes implementar REST sobre cualquier protocolo siempre y cuando puedas satisfacer sus restricciones. Esto ha llevado a la aparición de sistemas RESTful sobre protocolos como MQTT en el espacio de IoT. Sin embargo, la mayoría de las APIs RESTful en uso hoy en día se implementan sobre HTTP.

- La ventaja de este enfoque es que permite aprovechar la infraestructura existente de la web - los servidores web, las cachés, los proxies, etc. Además, al ser basado en HTTP, es naturalmente compatible con los navegadores web, lo que facilita la creación de aplicaciones web que consumen APIs RESTful.



- CRUD es un acrónimo que proviene de las palabras en inglés: Create (crear), Read (leer), Update (actualizar) y Delete (eliminar). Estas son las cuatro operaciones básicas que se pueden realizar en la mayoría de las aplicaciones que manejan algún tipo de persistencia de datos, y son fundamentales en el diseño de APIs, especialmente las que siguen la arquitectura REST.

- Aquí se detalla cómo los verbos HTTP (métodos) se utilizan generalmente en una API REST para operaciones CRUD en los recursos:

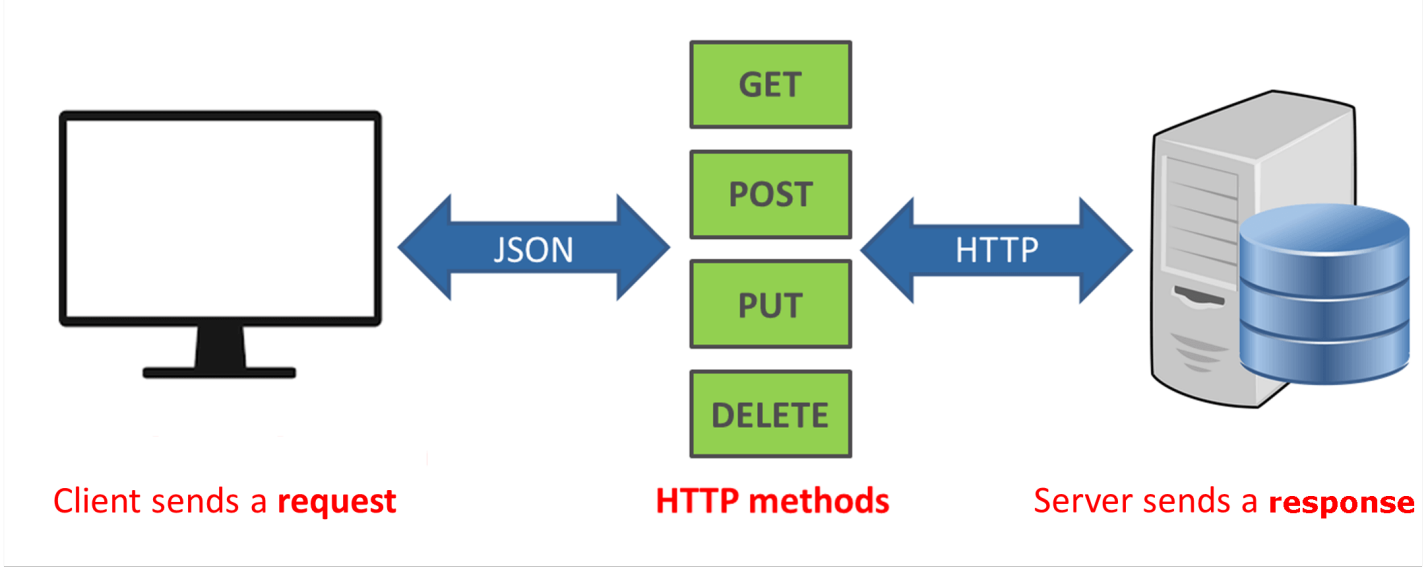
1. **Create** (Crear): Esta operación se realiza generalmente mediante una petición HTTP POST. En el caso de una API REST, enviaríamos una petición POST al endpoint que representa una colección de recursos para crear un nuevo recurso. El cuerpo de la petición contendría los detalles del nuevo recurso.

2. **Read** (Leer): Esta operación se realiza generalmente mediante una petición HTTP GET. Podemos obtener la representación de un recurso específico realizando una petición GET a la URI que representa a ese recurso. Del mismo modo, podemos obtener una lista de todos los recursos realizando una petición GET a la URI que representa a la colección de recursos.
3. **Update** (Actualizar): Esta operación se realiza generalmente mediante una petición HTTP PUT o PATCH. Enviando una petición PUT a la URI de un recurso específico, podemos actualizar ese recurso. La petición contendría la nueva representación del recurso. El método PATCH se utiliza cuando queremos hacer una actualización parcial del recurso.
4. **Delete** (Eliminar): Esta operación se realiza generalmente mediante una petición HTTP DELETE. Enviando una petición DELETE a la URI de un recurso específico, podemos eliminar ese recurso.

- Estas operaciones constituyen la esencia de una API RESTful y proporcionan los medios para interactuar con los recursos que expone la API.

# REST y HTTP: Implementación y Métodos

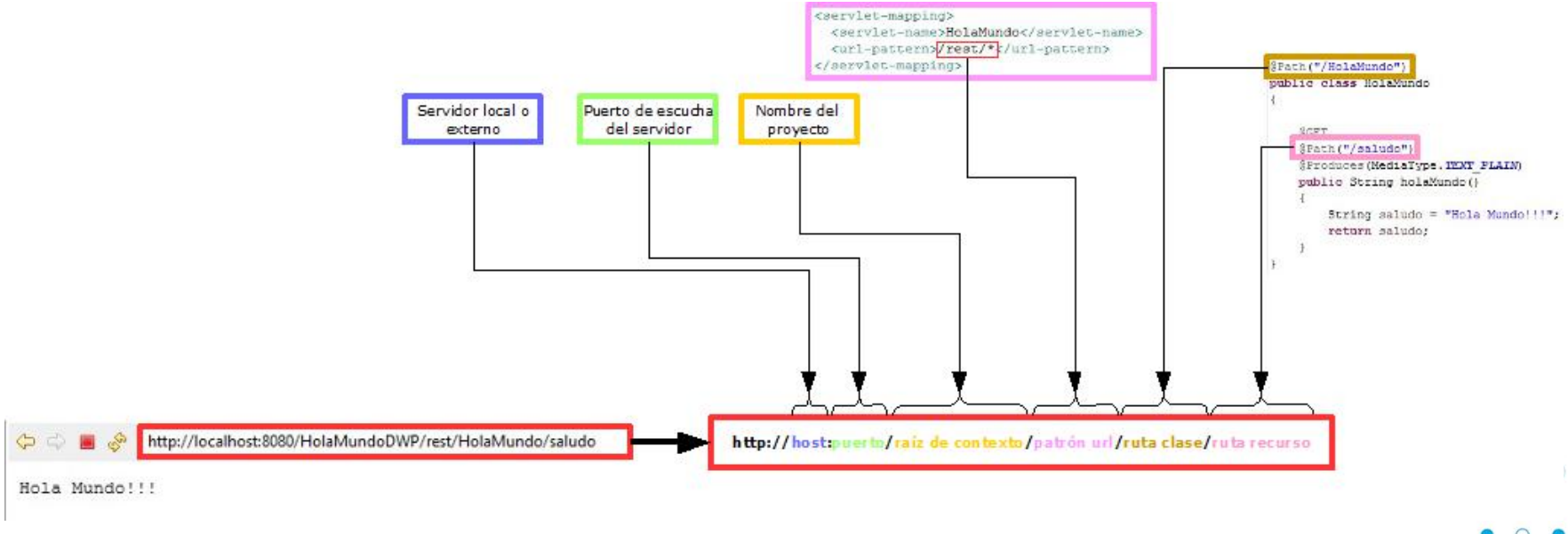
REST utiliza el protocolo HTTP para las operaciones de creación, lectura, actualización y eliminación de recursos. Los métodos HTTP se mapean a estas operaciones como sigue: **GET** para leer, **POST** para crear, **PUT** para actualizar y **DELETE** para eliminar. Los códigos de estado HTTP también se utilizan para comunicar el resultado de estas operaciones.



- REST se basa en la idea de que los recursos (las entidades en su aplicación a las que los usuarios acceden y modifican) pueden ser identificados y manipulados usando URLs y métodos HTTP estándar.
- En lugar de tener un conjunto personalizado de URLs y métodos para cada operación en su aplicación (por ejemplo, "/createUser", "/deleteUser", etc.), en un sistema RESTful, se tiene un conjunto fijo de URLs y métodos que se utilizan de manera consistente en toda la aplicación. Por ejemplo, para obtener un recurso, se haría una solicitud GET a la URL del recurso, para crear un recurso, se haría una solicitud POST a la URL de la colección de recursos, y así sucesivamente.
- Los códigos de estado HTTP se utilizan para indicar el éxito o fracaso de las operaciones. Por ejemplo, si una solicitud GET es exitosa, el servidor responderá con un código 200 OK. Si el recurso solicitado no existe, el servidor responderá con un código 404 Not Found. Estos códigos son una parte fundamental de la interfaz uniforme de REST, ya que proporcionan una manera estándar de indicar el resultado de una operación, independientemente del recurso específico que se esté manipulando.

# Identificación de Recursos: URIs en REST

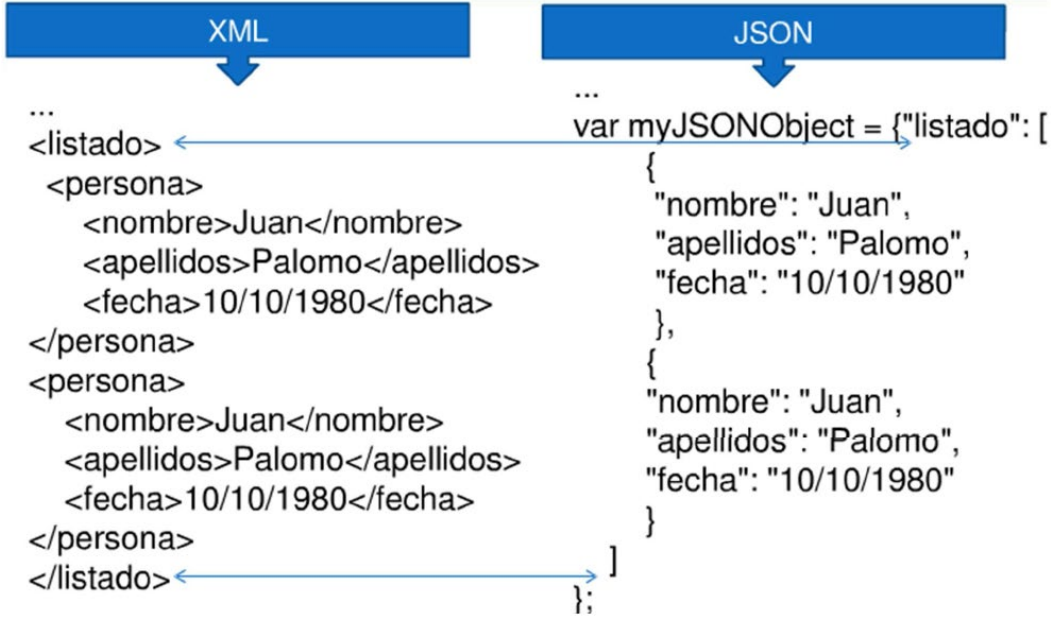
En REST, los recursos se identifican mediante URIs, que son la pieza fundamental de la interfaz uniforme. Las URIs proporcionan una forma estándar de localizar y acceder a los recursos. La estructura de las URIs en REST debe ser diseñada de tal manera que capture la información necesaria para entender y manejar la petición del cliente.



- Una URI (Uniform Resource Identifier) es un identificador único para un recurso en la web. En REST, cada recurso en el sistema debe tener su propia URI. Esto es parte de lo que se refiere con una "interfaz uniforme": todas las operaciones en el sistema se realizan a través de URIs, independientemente del recurso específico o la operación que se está realizando.
- La estructura de las URIs en un sistema RESTful es importante. Debe ser diseñada de tal manera que capture la información necesaria para entender y manejar la petición del cliente. Por ejemplo, si está manejando una colección de usuarios en su sistema, puede tener una URI como "/users" para la colección completa de usuarios, y "/users/{id}" para un usuario individual. Esto permite a los clientes de su API entender fácilmente cómo acceder y manipular los recursos en su sistema.
- Es importante tener en cuenta que las URIs en REST son solo identificadores. No contienen información sobre cómo se debe realizar una operación en el recurso, eso está determinado por el método HTTP de la solicitud.

# Formatos de Datos en REST: JSON y otras opciones

En REST, los datos se transmiten a través de representaciones de recursos, comúnmente en formato JSON debido a su sencillez y compatibilidad. Sin embargo, otros formatos como XML y HTML también pueden ser usados.



- REST no especifica un formato de datos particular que deba ser utilizado para las representaciones de recursos. En cambio, permite a los clientes y servidores negociar el formato de datos que se utilizará para cada intercambio de mensajes, lo que se conoce como "negociación de contenido".
- JSON se ha convertido en el formato de datos más comúnmente utilizado en las API RESTful debido a su sencillez y fácil compatibilidad con JavaScript, que es el lenguaje de programación más utilizado en el desarrollo web.
- El formato JSON es fácil de leer y escribir para los humanos, y fácil de analizar y generar para las máquinas, lo que lo convierte en un buen candidato para la transmisión de datos entre cliente y servidor.
- Además de JSON, también se pueden utilizar otros formatos de datos como XML y HTML. XML, aunque es más verboso que JSON, puede ser útil cuando se requiere un mayor grado de complejidad y detalle. HTML, por otro lado, puede ser útil para representaciones que se van a mostrar directamente en un navegador web.
- Independientemente del formato de datos que elija, es importante recordar que una de las ventajas clave de REST es su flexibilidad en este aspecto.

# API RESTful: Un puente entre cliente y servidor

- Una API RESTful, o Representational State Transfer API, es una interfaz que se adhiere a los principios de la arquitectura de REST. Se basa en el protocolo HTTP, el mismo protocolo subyacente que se utiliza para entregar páginas web.

- Estas APIs permiten la comunicación entre el cliente y el servidor de una manera que es coherente con la arquitectura de la web. Proporcionan una manera de interactuar con recursos web a través de operaciones estándar de HTTP.

- Por ejemplo, un cliente puede enviar una petición GET a una API RESTful para obtener una representación de un recurso, como los detalles de un usuario. Del mismo modo, el cliente puede enviar una petición POST para crear un nuevo recurso, o PUT para actualizar un recurso existente.

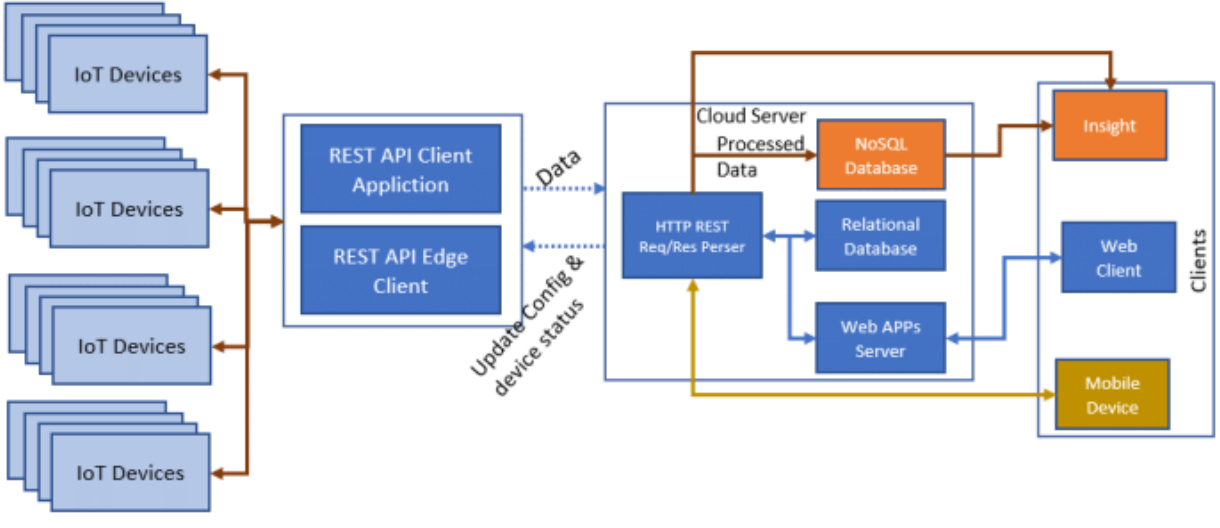
- Lo que hace que una API sea "RESTful" es que sigue los principios de REST, como la interfaz uniforme, la ausencia de estado y el uso de URIs para identificar recursos. Estos principios ayudan a asegurar que la API sea fácil de usar y compatible con la web en general.

- Además, una API RESTful generalmente implementará completamente el protocolo HTTP, utilizando todos sus métodos y códigos de estado para proporcionar una interfaz rica y expresiva.

- Finalmente, cabe mencionar que, a pesar de su popularidad, las APIs RESTful no son la única opción para la creación de servicios web. Hay otras arquitecturas y estilos de API que también se pueden utilizar, dependiendo de las necesidades particulares de tu aplicación.

# Uso de REST en el Internet de las Cosas (IoT)

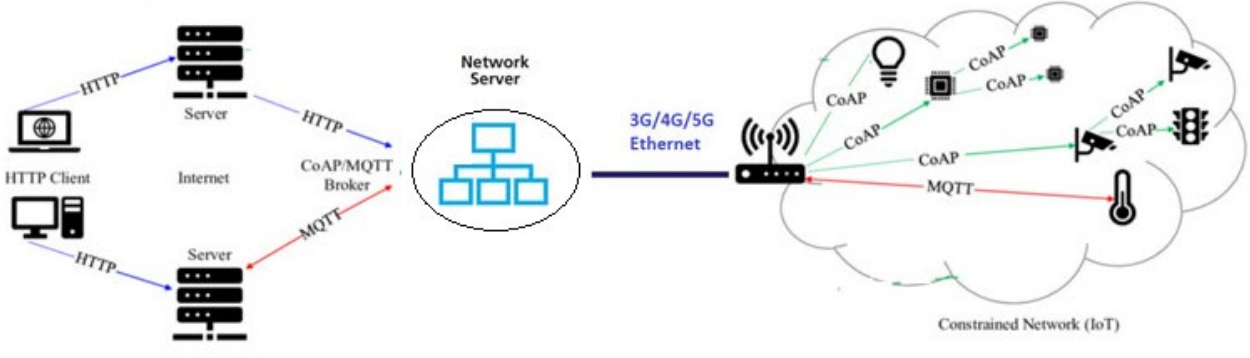
Internet de las Cosas, o IoT, se refiere a la creciente red de dispositivos físicos que están conectados a Internet y que pueden comunicarse entre sí. Este ámbito de la tecnología se ha vuelto cada vez más relevante y ha experimentado un rápido crecimiento en los últimos años. Exploraremos cómo REST se utiliza en el mundo de IoT, su integración con protocolos como MQTT y los desafíos y beneficios asociados con su uso en esta área.



- En el contexto de IoT, REST juega un papel importante ya que proporciona un método simple y estandarizado para permitir la comunicación entre dispositivos. Al adherirse a la arquitectura REST, los dispositivos IoT pueden comunicarse con los servidores y entre ellos de manera eficiente y coherente. Esto es particularmente útil en el contexto de IoT, donde puede haber una gran cantidad de dispositivos, cada uno de los cuales necesita comunicarse con los servidores y posiblemente con otros dispositivos.
- Además de REST, MQTT (Message Queuing Telemetry Transport) es otro protocolo comúnmente utilizado en el contexto de IoT. Aunque REST y MQTT son diferentes en varios aspectos y tienen diferentes fortalezas, pueden integrarse y utilizarse juntos para proporcionar una solución de comunicación completa en un entorno de IoT.
- Sin embargo, hay desafíos al usar REST en IoT. Dado que muchos dispositivos IoT son limitados en términos de potencia de procesamiento y capacidad de memoria, el uso de REST, que es un protocolo basado en texto, puede ser menos eficiente en términos de uso de la red y del dispositivo en comparación con protocolos más ligeros como MQTT.
- A pesar de estos desafíos, REST sigue siendo una opción popular para la comunicación en el ámbito de IoT debido a su simplicidad, facilidad de integración con la web existente y amplio soporte en una variedad de lenguajes y plataformas de programación.

# Practica: Creación y uso de un servicio web RESTful simple

1. Creación de un servicio web RESTful simple: Presentaremos el código fuente para un pequeño servicio web que sigue los principios REST, utilizando el lenguaje de programación Python y el micro-framework Flask.
2. Acceso al servicio web: Mostraremos cómo un cliente puede interactuar con este servicio a través de solicitudes HTTP.
3. Uso en un contexto de IoT: Hablaremos de cómo este servicio web RESTful podría ser utilizado en un contexto de IoT.



- Para ilustrar los conceptos que hemos discutido en esta presentación, pasaremos ahora a un ejemplo práctico. Utilizaremos Python, un lenguaje de programación común y ampliamente soportado, y Flask, un micro-framework para la creación de aplicaciones web.
- En este ejemplo, crearemos un servicio web simple que sigue los principios de REST. Este servicio tendrá un solo recurso, tal vez algo simple como una lista de tareas. Implementaremos métodos para obtener la lista de tareas (GET), agregar una nueva tarea (POST), modificar una tarea existente (PUT), y eliminar una tarea (DELETE).
- Después de implementar este servicio, mostraremos cómo un cliente puede interactuar con él. Utilizaremos solicitudes HTTP para acceder y manipular la lista de tareas.
- Finalmente, discutiremos cómo este tipo de servicio web RESTful podría ser utilizado en un contexto de IoT. Por ejemplo, los dispositivos IoT podrían interactuar con el servicio para reportar su estado, recibir instrucciones, o incluso interactuar con otros dispositivos.

**Herramientas y Recursos:**

- **Python:** Necesitarás tener Python instalado en tu sistema. Python es un lenguaje de programación que es conocido por su claridad y simplicidad, lo que lo convierte en una excelente opción para el desarrollo web. Puedes descargar Python desde la página oficial de Python.
- **Flask:** Flask es un microframework para Python que facilita el desarrollo de aplicaciones web. Es ligero, fácil de usar y muy adecuado para crear APIs RESTful. Puedes instalar Flask en tu entorno Python utilizando pip, que es el gestor de paquetes de Python. El comando para instalar Flask es pip install flask.
- **Postman:** Postman es una plataforma que te ayuda a diseñar, desarrollar y probar APIs. Puedes utilizar Postman para probar las operaciones CRUD de tu API RESTful enviando y recibiendo peticiones HTTP. Puedes descargar Postman desde la página oficial de Postman.

- **Visual Studio Code (VS Code):** VS Code es un editor de código fuente desarrollado por Microsoft. Soporta una variedad de lenguajes de programación, incluyendo Python. Es altamente personalizable y tiene una gran cantidad de extensiones disponibles para mejorar su funcionalidad. Puedes descargar VS Code desde la página oficial de Visual Studio Code.

- **CORS:** Debido a la política de mismo origen (Same Origin Policy) de los navegadores, es posible que necesites manejar CORS (Cross-Origin Resource Sharing) en tu aplicación Flask. Esto se puede hacer utilizando la extensión Flask-CORS. Puedes instalarla con pip install -U flask-cors.

Una vez que tengas estas herramientas instaladas y listas, estarás preparado para empezar a desarrollar tu API RESTful con Flask y Python.



- …
- Con Visual Studio Code (VSCode) como editor y Flask como framework, podemos crear un servicio RESTful simple.
- Vamos a crear una API básica para un blog que permitirá a los usuarios obtener publicaciones (GET), crear nuevas publicaciones (POST), actualizar publicaciones existentes (PUT) y eliminar publicaciones (DELETE). Cada publicación tendrá dos atributos: un "id" y un "contenido".
- En primer lugar, tendrás que crear un nuevo archivo Python (por ejemplo, app.py) en tu editor (VSCode).

Ahora puedes copiar el siguiente código en tu archivo app.py

- **Codigo de ejemplo:**

app.py > ...

```python
from flask import Flask, jsonify, request
from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # Esta línea permitirá las solicitudes de origen cruzado

posts = [
    {'id': 1, 'contenido': 'Mi primera publicación'},
    {'id': 2, 'contenido': 'Mi segunda publicación'}
]

@app.route('/blog', methods=['GET'])
def get_posts():
    return jsonify(posts)

@app.route('/blog', methods=['POST'])
def add_post():
    new_post = {'id': request.json['id'], 'contenido': request.json['contenido']}
    posts.append(new_post)
    return jsonify(new_post), 201
```

```python

@app.route('/blog/<int:post_id>', methods=['PUT'])
def update_post(post_id):
    post = [post for post in posts if post['id'] == post_id]
    if not post:
        return jsonify({'error': 'No se encontró la publicación'}), 404
    post[0]['contenido'] = request.json.get('contenido', post[0]['contenido'])
    return jsonify(post[0])

@app.route('/blog/<int:post_id>', methods=['DELETE'])
def delete_post(post_id):
    post = [post for post in posts if post['id'] == post_id]
    if not post:
        return jsonify({'error': 'No se encontró la publicación'}), 404
    posts.remove(post[0])
    return jsonify({'exito': 'Publicación eliminada correctamente'}), 200

if __name__ == '__main__':
    app.run(debug=True)

```

- Para este código, necesitamos el paquete flask_cors para manejar la política de mismo origen en Flask. Puedes instalarlo con pip install flask-cors.
- Este es un código muy básico y podría necesitar mejoras para un entorno de producción, pero nos dará una idea de cómo funcionan las APIs RESTful con Flask. Ahora, se puede usar Postman para probar los métodos GET, POST, PUT y DELETE en la API.
- Inicia el servidor Flask ejecutando $ python app.py en la terminal y abrir Postman.
- En la interfaz de Postman, verás un campo para ingresar una URL. Aquí es donde puedes poner la ruta a la que quieres enviar una solicitud. Como estás ejecutando el servidor en tu propia máquina, la URL base será http://localhost:5000/, que es la URL predeterminada donde Flask ejecuta tu aplicación.
- A la derecha del campo URL, hay un menú desplegable que te permite seleccionar el tipo de solicitud HTTP que quieres enviar (GET, POST, PUT o DELETE). Elige uno según la operación que quieras realizar.

Aquí están los detalles de cómo probar cada método:

- **GET:** Selecciona 'GET' en el menú desplegable y escribe http://localhost:5000/blog en la barra de URL.
- Luego, haz clic en el botón 'Send'. Deberías ver un arreglo JSON de todas las publicaciones en la respuesta.
- **POST:** Selecciona 'POST' en el menú desplegable y escribe http://localhost:5000/blog en la barra de URL.
- Haz clic en la pestaña 'Body' debajo de la URL, luego selecciona 'raw' y cambia el formato a 'JSON'. En el cuadro de texto que aparece, puedes escribir el cuerpo de tu solicitud en formato JSON. Por ejemplo, podrías escribir: {"id": 3, "contenido": "Mi tercera publicación"}.
- Luego, haz clic en el botón 'Send'. Deberías ver la nueva publicación en la respuesta.

- **PUT:**

- Selecciona 'PUT' en el menú desplegable y escribe http://localhost:5000/blog/1 en la barra de URL, donde '1' es el ID de la publicación que quieres actualizar.
- Al igual que con POST, necesitarás escribir el cuerpo de la solicitud en formato JSON. Por ejemplo, podrías escribir: {"contenido": "Contenido de la publicación actualizado"}.
- Luego, haz clic en el botón 'Send'. Deberías ver la publicación actualizada en la respuesta.

- **DELETE:**

- Selecciona 'DELETE' en el menú desplegable y escribe http://localhost:5000/blog/1 en la barra de URL, donde '1' es el ID de la publicación que quieres eliminar.
- Luego, haz clic en el botón 'Send'. Deberías ver un mensaje indicando que la publicación se eliminó con éxito.

# ¡Muchas gracias!