INTRODUCCIÓN AL DESARROLLO BACKEND

CON NODEJS

2023

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

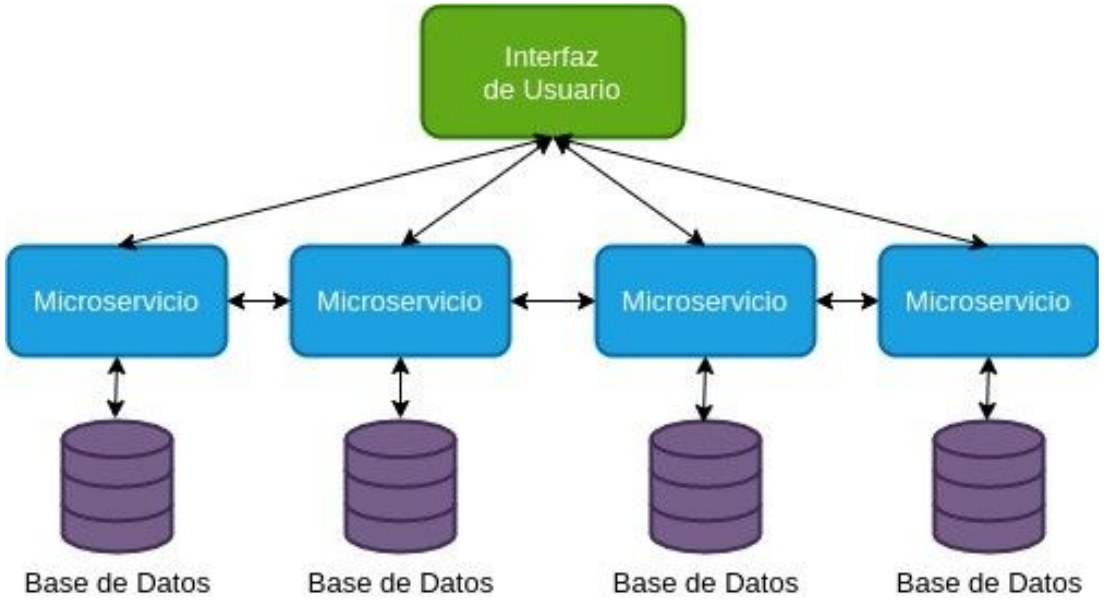
Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos

Introducción a las APIs y API RestFul

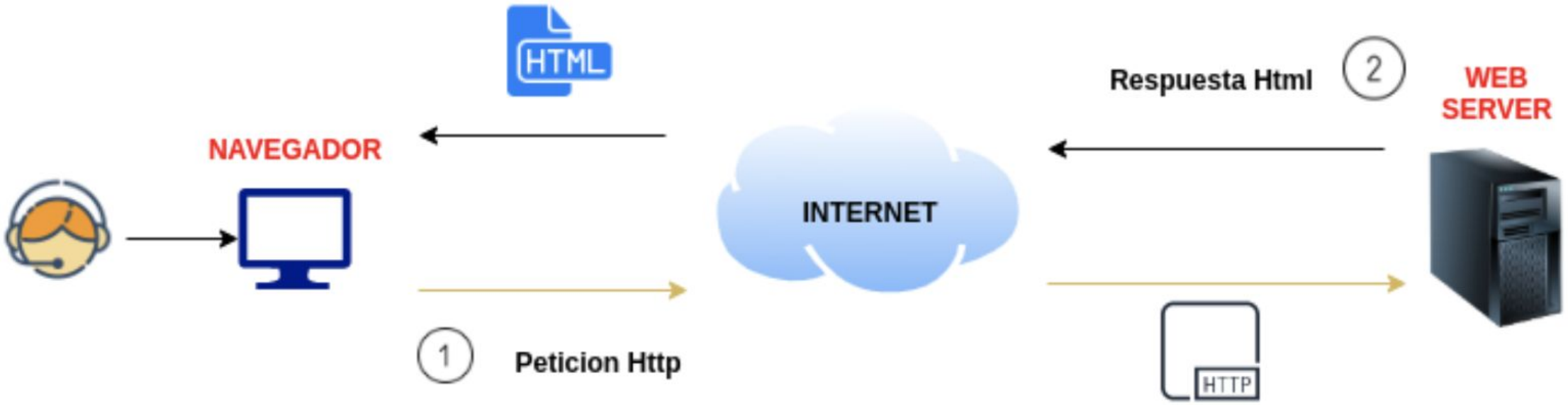
- Aplicaciones Web
  - Arquitectura de Microservicios
  - ¿Cómo funciona una página Web?
  - ¿Qué es HTTP?
    - HTTP: Urls aplicaciones Web
    - HTTP: Componentes
- API Restful
  - Ventajas de API RestFul
  - Principios API RestFul
  - Documentar API RestFul
- Introducción a Postman
- ¿Qué es Express?

Arquitectura de Microservicios



- Las aplicaciones se distribuyen en múltiples partes, más pequeñas e independientes llamados microservicios.
- Estos microservicios están completamente desacoplados entre sí y funcionan de forma totalmente independiente, lo que permite que cualquiera de ellos pueda ser reemplazado en tiempo real sin afectar el funcionamiento de la aplicación.

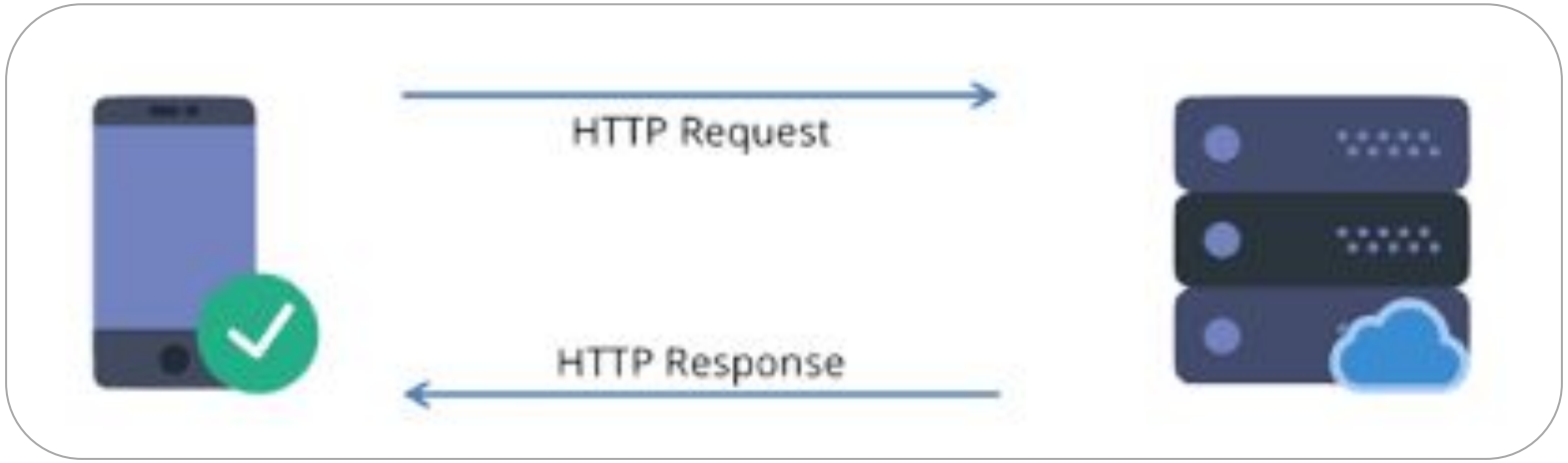
¿Cómo funciona una página Web?



¿Qué es HTTP?

HTTP es un protocolo de comunicación utilizado en la web para la transferencia de datos entre diferentes dispositivos y servidores.

- Se basa en un **modelo cliente-servidor**,
- Utiliza **diferentes métodos** para realizar operaciones en los datos, y
- Utiliza un formato de **mensaje** simple que consta de una **cabecera y un cuerpo**.



HTTP: Urls de aplicaciones Web

1) Protocolo

http://www.example.com:80/path

Protocol

2) Autoridad

http://www.example.com:80/path/to/my

Domain Name

3) Puerto

com:80/path/to/myfile.html?key1=valu

Port

4) Ruta

n:80/path/to/myfile.html?key1=value1

Path to the file

5) Parámetros

html?key1=value1&key2=value2#Some

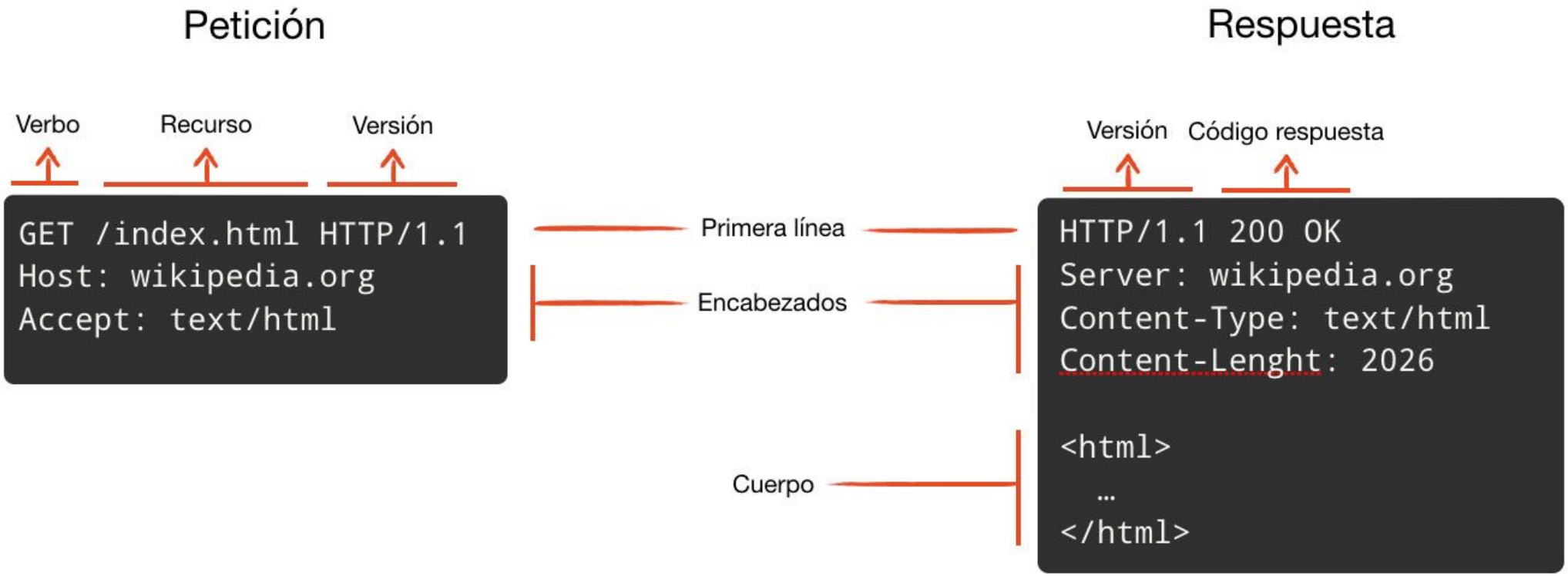
Parameters

6) Fragmento

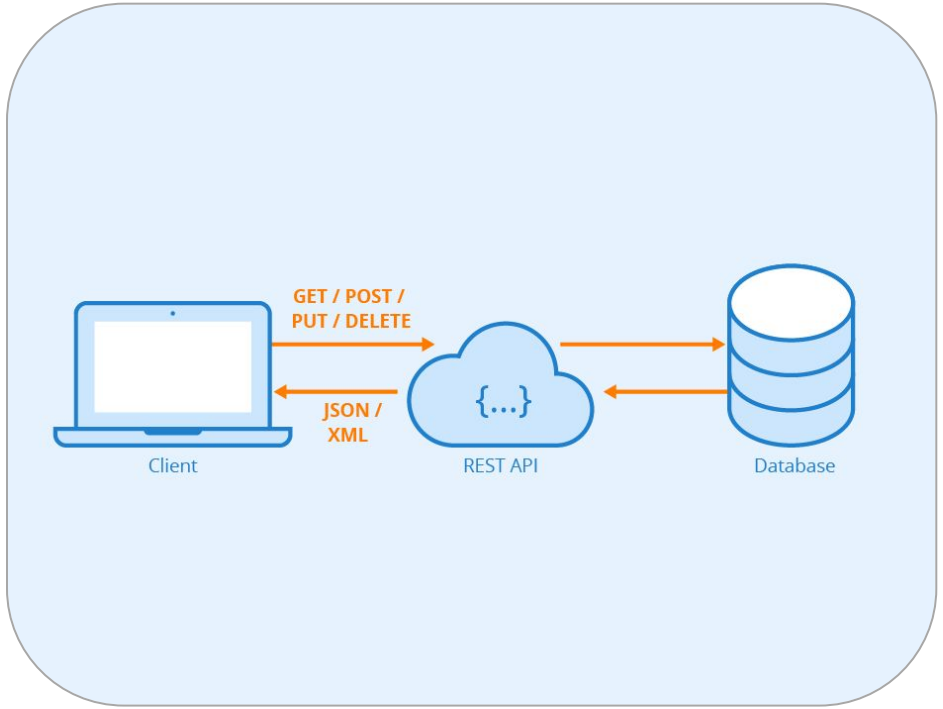
lue2#SomewhereInTheDocument

Anchor

HTTP: Componentes



API Restful



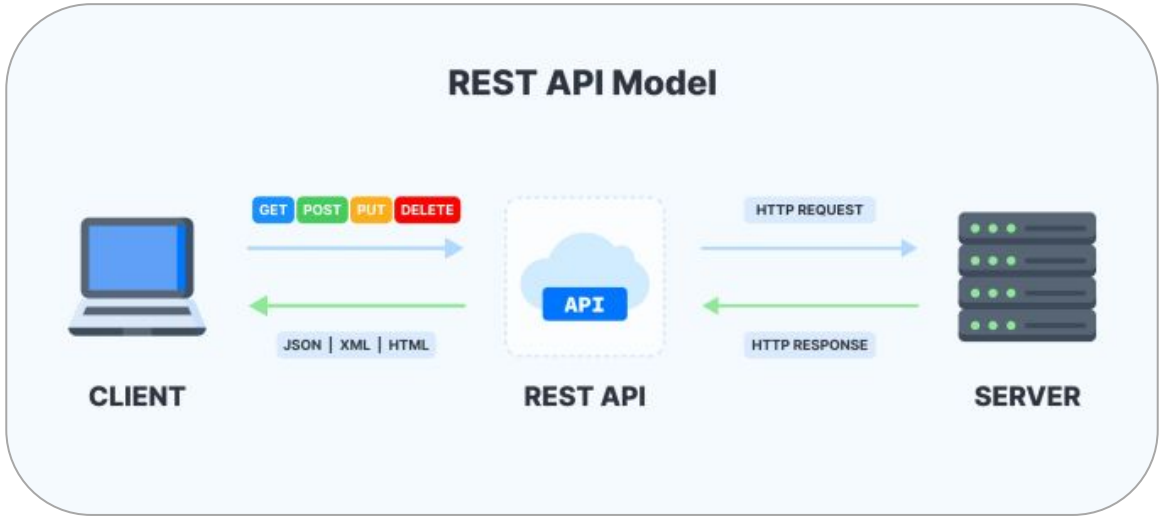
- **API** es un conjunto de reglas y especificaciones que las **aplicaciones** pueden seguir para **comunicarse** entre ellas. Es el mecanismo más útil para conectar dos softwares entre sí, de esta manera, podemos garantizar el intercambio de mensajes o datos en formato estándar.
- **Rest** (Representational State Transfer) es una arquitectura que se ejecuta sobre HTTP.
- **API Rest:** hace referencia a un servicio web que implementa la arquitectura REST.
- **Web APIs:** Sinónimo de API Rest.
- **API Restful:** API Rest que aplica buenas prácticas de diseño.

Ventajas de API RestFul

- **Flexibilidad:** Las WebAPI permiten que diferentes aplicaciones cliente se comuniquen con el servidor a través de protocolos de comunicación estándar.
- **Escalabilidad:** Las WebAPI permiten que el servidor proporcione servicios a un gran número de aplicaciones cliente simultáneamente.
- **Reutilización:** Las WebAPI permiten que la funcionalidad del servidor sea reutilizada en diferentes aplicaciones cliente.

API Restful

- Manténgalo simple (Keep it simple)
- Usar sustantivos y no verbos
- Uso de los métodos HTTP adecuados
- Usar plurales
- Utilizar parámetros
- Utilice códigos HTTP adecuados
- Versiones
- Usar paginación
- Formatos soportados
- Utilizar mensajes de error adecuados
- Uso de las especificaciones de OpenAPI



Manténgalo simple (Keep it simple)

Necesitamos asegurarnos de que la **URLs** de la API RestFul sea **simple**. Por ejemplo, si queremos diseñar APIs para productos, debería diseñarse como:

```
http://api.ejemplo.com/productos

http://api.ejemplo.com/productos/{id_producto}
```

La primera API es para **obtener todos** los productos y la segunda es para **obtener un producto específico**.

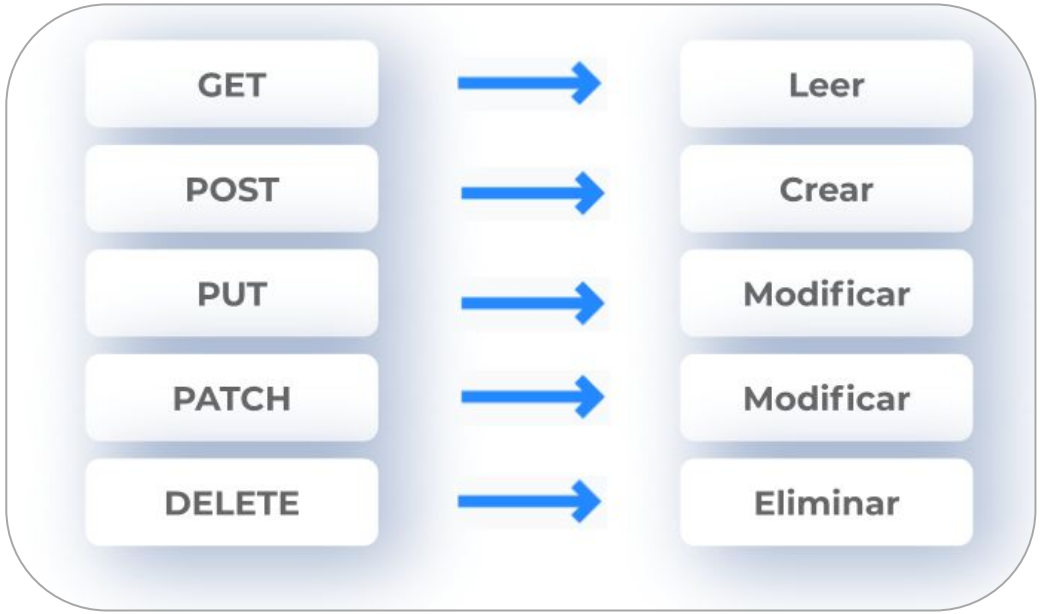
Usar sustantivos y no verbos

Muchos desarrolladores cometen este error. Generalmente olvidan que tenemos métodos HTTP para describir mejor las APIs y terminan usando verbos en las URLs de las APIs. Por ejemplo, API para obtener todos los productos debe ser:

Uso de los métodos HTTP adecuados

Las APIs de RESTful tienen varios **métodos** para indicar el tipo de operación que vamos a realizar con esta API. Debemos asegurarnos de que usamos el método HTTP correcto para una operación determinada.

- **GET**: Para obtener un recurso o colección de recursos.
- **POST:** Para crear un recurso o colección de recursos.
- **PUT/PATCH:** Para actualizar el recurso o conjunto de recursos existentes.
- **DELETE:** Para borrar el recurso existente o el conjunto de recursos.



Usar plurales

Mantenerlo en plural evita confusiones sobre si estamos hablando de obtener un solo recurso o una colección.

Utilizar parámetros

Los parámetros se utilizan para enviar datos adicionales junto con las solicitudes HTTP. Estos parámetros pueden clasificarse en diferentes tipos, dependiendo de cómo se envíen y se utilicen en las solicitudes:

- **Parámetros de consulta (Query Parameters):**
  - Se especifican en la URL después del signo de interrogación (?).
  - Se utilizan para filtrar, ordenar o paginar los resultados de una solicitud.
  - Ejemplo: https://api.ejemplo.com**/productos**?categoria=electronics&max_precio=100
- **Parámetros de ruta (Path Parameters):**
  - Se incluyen directamente en la URL y forman parte de la ruta.
  - Se utilizan para identificar un recurso específico en una solicitud.
  - Ejemplo: https://api.ejemplo.com**/productos/{id_producto}**
- **Parámetros de encabezado (Header Parameters):**
  - Se incluyen en el encabezado de la solicitud HTTP.
  - Proporcionan información adicional sobre la solicitud o el cliente.
  - Ejemplo: Authorization: Bearer <token>
- **Parámetros de cuerpo (Body Parameters):**
  - Se incluyen en el cuerpo de la solicitud HTTP.
  - Se utilizan para enviar datos más complejos, como JSON o XML.
  - Comúnmente utilizados en solicitudes POST, PUT o PATCH.
  - Ejemplo (JSON): { "name": "John", "age": 25 }

Utilice códigos HTTP adecuados

Algunos códigos de estado son:

- **2xx** - Successful response
  - 200 OK
  - 201 Created
- **3xx** - Redirect response
  - 301 Moved permanently
- **4xx** - Client error Response
  - 401 Unauthorized
  - 403 Forbidden
  - 404 File not found
- **5xx** - Service error response
  - 500 Server Error
  - 503 Service unavailable

Versiones

La creación de versiones de las API es muy importante. Muchas empresas diferentes utilizan las versiones de diferentes maneras. Algunas utilizan versiones como fechas, mientras que otras utilizan versiones como parámetros de consulta.

Lo que mejor funciona es mantenerlo prefijado al recurso. Por ejemplo:

```
http://api.ejemplo.com/v1/productos

http://api.ejemplo.com/v2/productos
```

Siempre es una buena práctica mantener la **compatibilidad hacia atrás** para que si cambia la versión de la API, los **consumidores** tengan suficiente tiempo para pasar a la siguiente versión.

Usar paginación

El uso de la paginación es imprescindible cuando se expone una API que puede devolver **grandes cantidades de datos**, y si no se hace un balanceo de carga adecuado, el consumidor puede terminar por derribar el servicio.

Es buena práctica usar **limit** y **offset**. Por ejemplo:

https://api.ejemplo.com/productos?**limit=25&offset=50.**

También se aconseja mantener un valor predeterminado de limit y offset.

Parámetro "limit":

- El parámetro "limit" establece el número máximo de resultados que deseas recibir en una sola solicitud.
- Por ejemplo, si estableces "limit=10", la API te devolverá como máximo 10 resultados en la respuesta.
- Este parámetro es útil cuando trabajas con conjuntos de datos grandes y deseas limitar la cantidad de información que recibes en una sola solicitud.

Parámetro "offset":

- El parámetro "offset" indica el punto de inicio para los resultados que deseas obtener.
- Por ejemplo, si estableces "offset=20", la API omitirá los primeros 20 resultados y comenzará a devolver resultados a partir del número 21.
- Este parámetro es útil cuando necesitas recuperar resultados en lotes o páginas y deseas especificar desde qué posición comenzar.

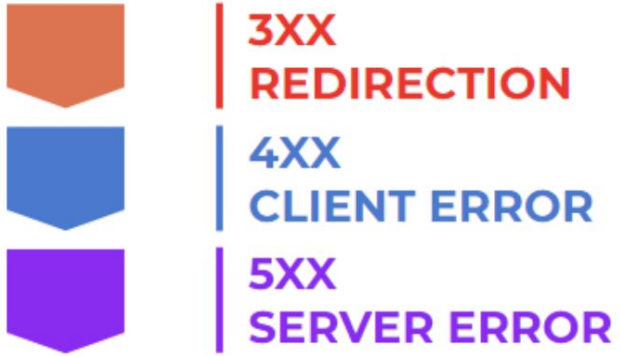
Formatos soportados

También es importante elegir **cómo responderá su API**.
La mayoría de las aplicaciones de hoy en día deberían devolver las respuestas de **JSON**:


```
{
  "clientes": [
    {
      "nombre": "Juan",
      "apellido": "Perez"
    },
    {
      "nombre": "Pedro",
      "apellido": "Garcia"
    },
    {
      "nombre": "Maria",
      "apellido": "Sanchez"
    }
  ]
}
```

Utilizar mensajes de error adecuados

Siempre es una buena práctica mantener un conjunto de **mensajes de error** que la aplicación envía y responder a ellos con la identificación adecuada. Por ejemplo:




```
{
    "error":{
        "mensaje":"Error General",
        "tipo":"general",
        "codigo":"1234",

    }
}
```

Introducción a Postman

- Postman es una herramienta utilizada por los desarrolladores para probar y realizar solicitudes a API (Application Programming Interface) de manera sencilla.
- No es necesario conocer programación para utilizar Postman y realizar pruebas de API.
- Es una herramienta visual e intuitiva que permite interactuar con servicios web de manera eficiente.

POSTMAN

Instalar Postman

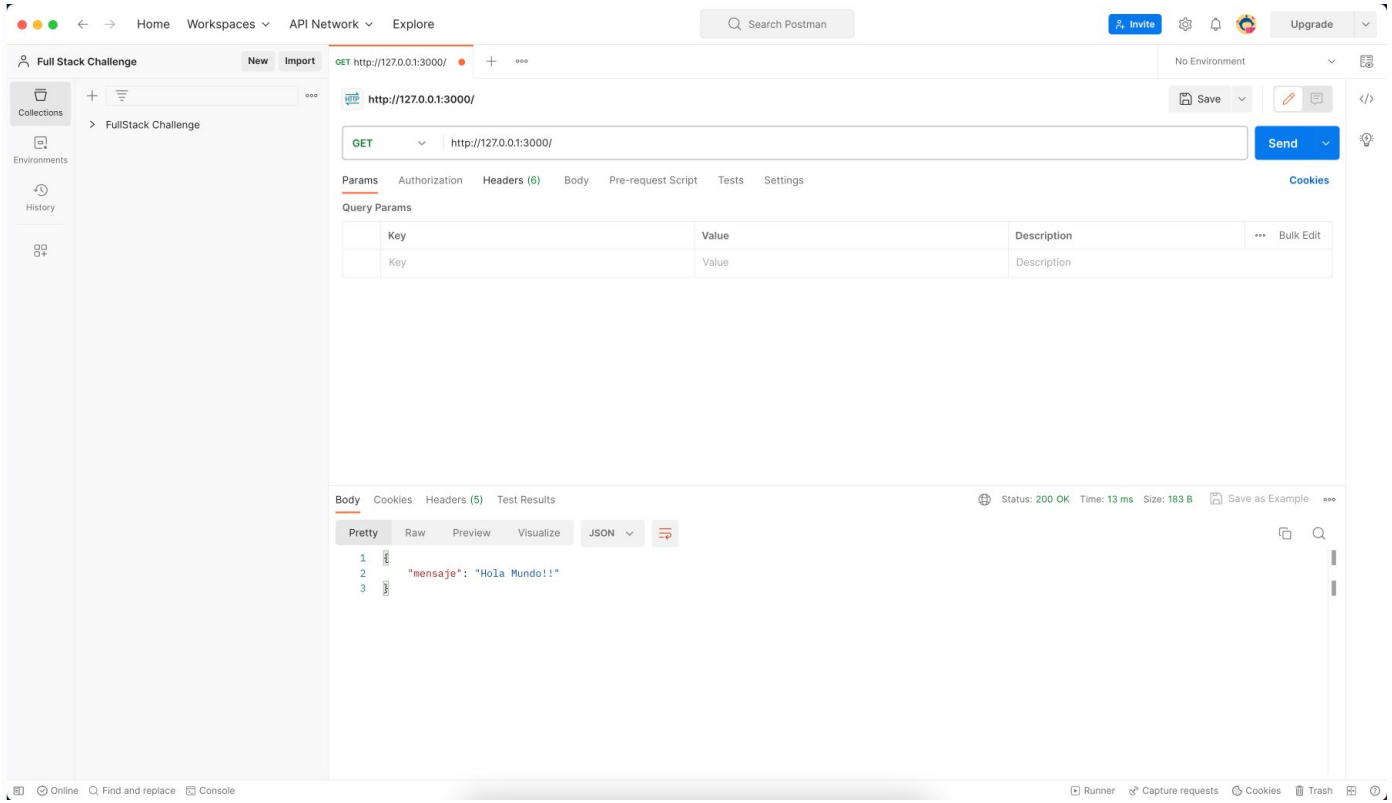
- Descargar archivo de instalación usando el siguiente link:
  - https://www.postman.com/downloads/
- Seguir las instrucciones de instalación.

Características principales de Postman

- **Interfaz de usuario amigable:** Mostrar una captura de pantalla de la interfaz de Postman resaltando su diseño intuitivo y fácil de usar.
- **Creación de solicitudes HTTP:** Explicar que se pueden crear diferentes tipos de solicitudes HTTP, como GET, POST, PUT, DELETE, para interactuar con las API.
- **Gestión de colecciones:** Explicar que las colecciones permiten organizar y guardar las solicitudes en grupos lógicos para facilitar su reutilización.
- **Pruebas y validación:** Mencionar que Postman ofrece la posibilidad de realizar pruebas y validaciones de respuestas de API, verificando si cumplen con los requisitos esperados.
- **Documentación:** Destacar que Postman permite generar automáticamente documentación de API a partir de las solicitudes y respuestas creadas.
- **Colaboración:** Mencionar que Postman facilita la colaboración en equipo, permitiendo compartir colecciones y colaborar en el desarrollo y prueba de API.

POSTMAN

Postman en vivo!



Para probar una APIs vamos a usar **JsonPlaceholder**, una API pública que podemos utilizar para practicar:

https://jsonplaceholder.typicode.com/

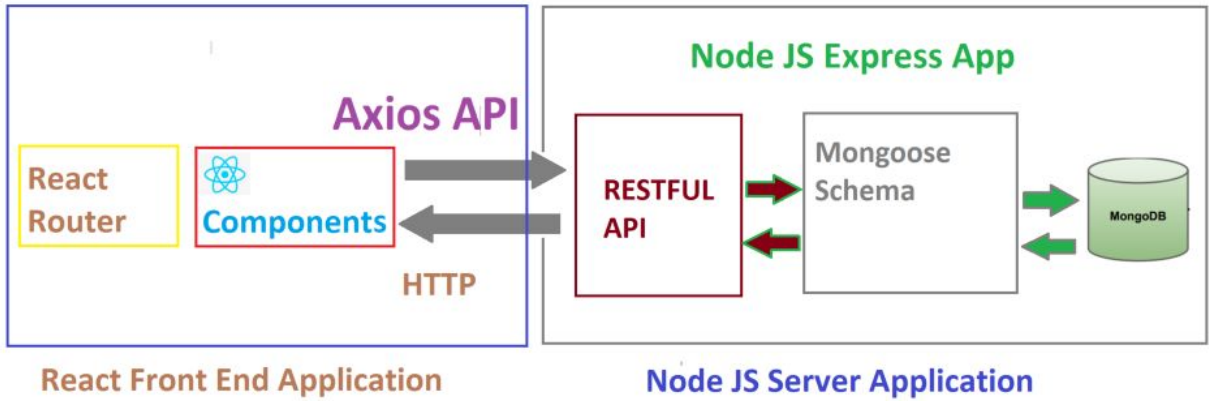
Documentar API RestFul

Swagger nos ofrece una serie de reglas, herramientas y especificaciones que nos ayudan a documentar APIs Restful. Para la especificación se puede trabajar con Json o Yaml.

Herramientas:

- **Swagger UI:** transforma una especificación Swagger / OpenApi en una página web más amigable
- **Swagger Editor:** Editor web de especificaciones Swagger / OpenApi. Tiene una vista de la especificación a la izquierda y la vista UI a la derecha. (https://editor.swagger.io/)

Frontend vs Backend



Bueno, Vamo a Codea!!!

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World!!!');
});

server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Módulo http

- El módulo http se utiliza en node para gestionar el protocolo http
  - `const http = require('http');`
- La constante http ahora puede gestionar peticiones HTTP.
- La función (anónima) que recibe http.createServer se va a ejecutar cada vez que hay una petición al puerto de escucha:

```
const server = http.createServer((req, res) => {
res.statusCode = 200;
res.setHeader('Content-Type', 'text/plain');res.end('Hello World
DDS!!');});
```

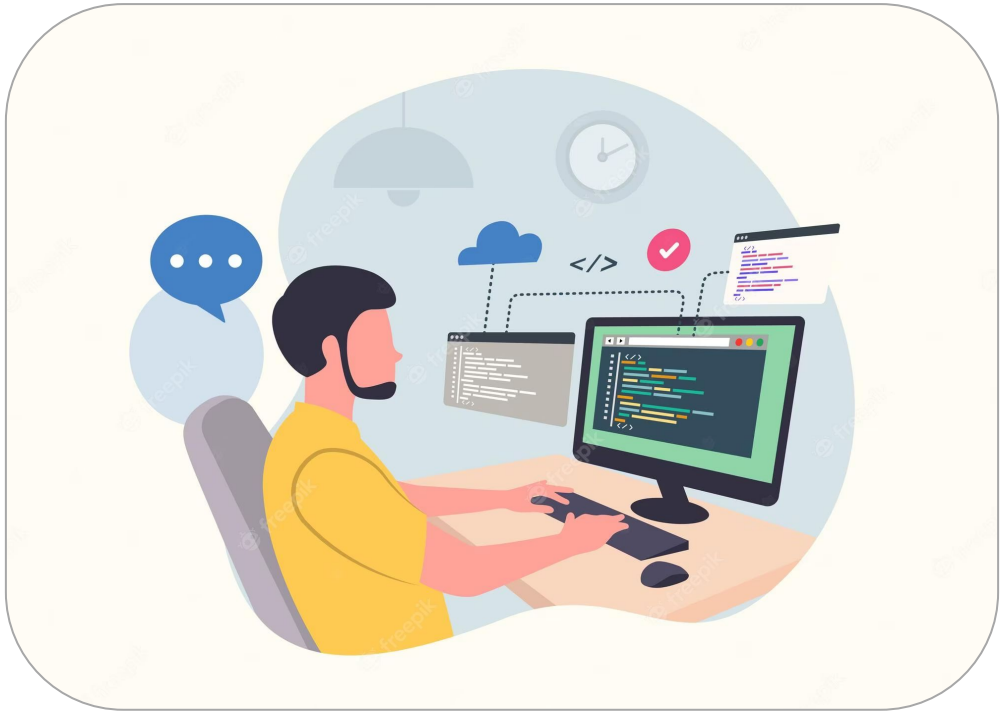
- El servicio http comienza a escuchar en el puerto y hostname configurado con `server.listen(port, hostname, () => {.....});`

¿Qué es Express?

- Express es una infraestructura de aplicaciones web Node.js **mínima y flexible** que proporciona un conjunto sólido de características para las aplicaciones web y móviles.
- Se desarrolló en el año **2010**.
- Permite **desarrollo simple y rápido** de api REST
- **Comunidad** de soporte y desarrollo muy amplia y madura.
- No depende de patrones MVC
- Hay otros frameworks para aplicaciones web como Restify, Hapi, Sails, Meteor, Loopback, etc.

Actividad 3: Paso a Paso API con NodeJs

- Seguir las instrucciones de la actividad publicada en la UVE.



MUCHAS GRACIAS

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos