Introducción a las preguntas teóricas:

Como sabemos, en la actualidad existen grandes modelos de inteligencia artificial con la capacidad de responder la mayoría de preguntas que podamos realizar como reclutadores. Si bien las respuestas van a ser evaluadas, la finalidad de la parte teórica es ser una guía para que el postulante pueda entender la orientación de la posición y afianzarse con los conocimientos necesarios para su desarrollo laboral.

Posteriormente a la evaluación del trabajo práctico, podrá existir una instancia de conversación donde validemos los conocimientos y el entendimiento de los conceptos.

Teoría:

Preguntas generales sobre HTTP/HTTPS:

¿Qué es HTTP y cuál es su función principal?

HTTP (HyperText Transfer Protocol) es un protocolo de comunicación que permite la transferencia de información entre un cliente (generalmente un navegador) y un servidor web. Su función principal es definir cómo se solicitan y entregan recursos como páginas HTML, imágenes, videos o datos en formato JSON.

¿Cuál es la diferencia entre HTTP y HTTPS?

Por un lado, HTTP transmite los datos en texto plano, lo que lo hace vulnerable a ataques de interceptación. En cambio, HTTPS (HTTP Secure) utiliza cifrado mediante SSL/TLS, garantizando confidencialidad, integridad y autenticidad en la comunicación.

¿Cómo funciona el proceso de cifrado en HTTPS?

- Se inicia un handshake TLS, donde cliente y servidor acuerdan un algoritmo de cifrado.
- El servidor envía su certificado digital, que contiene su clave pública.
- El cliente valida el certificado con una autoridad certificadora (CA).
- Se genera una clave de sesión compartida mediante criptografía asimétrica.
- Toda la comunicación posterior se cifra con esa clave (criptografía simétrica).

¿Qué es un certificado SSL/TLS y cuál es su importancia en HTTPS?

Es un archivo digital emitido por una Autoridad Certificadora (CA) que verifica la identidad del servidor, permite el cifrado de la comunicación mediante claves públicas y privadas y da confianza al usuario al mostrar el candado en el navegador.

¿Qué es un método HTTP? ¿Podrías enumerar algunos de los más utilizados?

Un método HTTP indica la acción que el cliente quiere realizar sobre un recurso en el servidor. Algunos de los más utilizados son:

- **GET**: obtener datos.
- **POST**: enviar datos.
- **PUT**: actualizar un recurso existente.
- **DELETE**: eliminar un recurso.
- **PATCH**: actualizar parcialmente un recurso.

Explica las diferencias entre los métodos HTTP GET y POST.

GET envía los datos en la URL (parámetros de consulta). Es idempotente y seguro. Se usa para obtener información. En cambio, POST envía los datos en el cuerpo de la petición. Se usa para crear recursos o enviar información sensible. No es idempotente.
NOTA: operación idempotente es aquella que produce el mismo resultado o estado final sin importar cuántas veces se ejecute.

¿Qué es un código de estado HTTP? ¿Podrías mencionar algunos de los más comunes y lo que significan?

Es un número que indica el resultado de una petición. Los mas comunes son:

- 200 OK → Petición exitosa.
- 201 Created → Recurso creado correctamente.
- 400 Bad Request → Error en la petición del cliente.
- 401 Unauthorized → Falta autenticación.
- 403 Forbidden → No tiene permisos.
- 404 Not Found → Recurso no encontrado.
- 500 Internal Server Error → Error en el servidor.

¿Qué es una cabecera HTTP? Da ejemplos de cabeceras comunes.

Son metadatos enviados en las peticiones/respuestas HTTP.
Ejemplos:

- Content-Type: tipo de dato (ej. application/json).
- Authorization: credenciales de autenticación.
- User-Agent: información del cliente.
- Accept: formatos aceptados por el cliente.
- Cache-Control: políticas de caché.

¿En qué consiste el concepto de "idempotencia" en los métodos HTTP? ¿Qué métodos cumplen con esta característica?
Un método es idempotente si ejecutarlo una o varias veces produce el mismo resultado en el servidor.
Métodos idempotentes: GET, PUT, DELETE, HEAD, OPTIONS.
No idempotentes: POST, PATCH.

¿Qué es un redirect (redirección) HTTP y cuándo es utilizado?

Es una respuesta del servidor que indica que el recurso solicitado se encuentra en otra ubicación. Se usa para:

- Redirigir a una nueva URL (ej. cambio de dominio).
- Forzar HTTPS desde HTTP.
- Enviar al usuario después de un login.

Códigos típicos: 301 (permanente), 302 (temporal).

Preguntas técnicas y de seguridad en HTTP/HTTPS:

¿Cómo se asegura la integridad de los datos en una conexión HTTPS?

La integridad se garantiza gracias a:

- Firmas digitales y certificados TLS → evitan que los datos sean modificados en tránsito.
- Algoritmos de hash (SHA-256, SHA-3) → verifican que el mensaje no cambió.
- MAC (Message Authentication Code) → permite validar la autenticidad del emisor.

¿Qué diferencia hay entre un ataque de "man-in-the-middle" y un ataque de "replay" en un contexto HTTPs?
Man-in-the-middle (MITM): un atacante intercepta y modifica la comunicación entre cliente y servidor.

Replay attack: un atacante captura un mensaje válido y lo reenvía más tarde para engañar al servidor (ej. reutilizar credenciales).

Explica el concepto de "handshake" en HTTPS

Es el proceso inicial de conexión TLS en el que:

Cliente y servidor negocian algoritmos de cifrado -> El servidor envía su certificado digital ->El cliente valida el certificado ->Se genera una clave de sesión compartida.

Este proceso asegura autenticidad y confidencialidad antes de transmitir datos.

¿Qué es HSTS (HTTP Strict Transport Security) y cómo mejora la seguridad de una aplicación web?

HSTS es un encabezado (Strict-Transport-Security) que indica al navegador que solo debe comunicarse con el servidor mediante HTTPS.
Algunos beneficios que obtenemos mediante este encabezado son:

- Evita ataques de downgrade (forzar HTTP).
- Evita ataques MITM con redirecciones a HTTP.

¿Qué es un ataque "downgrade" y cómo HTTPS lo previene?
Un ataque downgrade ocurre cuando un atacante obliga al cliente y servidor a usar una versión antigua e insegura de un protocolo (ej. SSL 2.0).

¿Qué es el CORS (Cross-Origin Resource Sharing) y cómo se implementa en una aplicación web?

CORS es un mecanismo que permite a un servidor autorizar solicitudes desde orígenes distintos al suyo.
Se implementa con cabeceras HTTP, por ejemplo:

- Access-Control-Allow-Origin: * → permite cualquier origen.
- Access-Control-Allow-Origin: https://midominio.com → permite solo un dominio.
- Access-Control-Allow-Methods: GET, POST, PUT.

¿Qué diferencia hay entre una cabecera Authorization y una cabecera Cookie?

La cabecera Authorization se usa para enviar credenciales en cada petición (ej. Bearer <token>).Y por otro lado la cabecera cookie almacena información en el navegador que se envía automáticamente en cada petición.

¿Qué son las cabeceras de seguridad como Content-Security-Policy o X-Frame-Options? ¿Cómo ayudan a mitigar ataques comunes?

La cabecera Content-Security-Policy (CSP): limita qué recursos (scripts, imágenes, estilos) puede cargar el navegador mitigando asi ataques XSS. Luego la cabecera X-Frame-Options evita que el sitio se cargue en un <iframe> y de esta forma previniendo clickjacking.

¿Cuáles son las diferencias entre HTTP/1.1, HTTP/2 y HTTP/3?
HTTP/1.1: conexiones secuenciales, un request por conexión.

HTTP/2: multiplexación (múltiples requests/responses en una misma conexión), compresión de cabeceras.

HTTP/3: usa QUIC (sobre UDP), mejora velocidad y confiabilidad en redes inestables.

¿Qué es un "keep-alive" en HTTP y cómo mejora el rendimiento de las aplicaciones?

Es una cabecera (Connection: keep-alive) que mantiene abierta la conexión TCP entre cliente y servidor para múltiples requests.
Mejora el rendimiento porque evita crear una conexión nueva por cada petición y reduce latencia y carga en el servidor.

Preguntas de implementación práctica:

¿Cómo manejarías la autenticación en una API basada en HTTP/HTTPS? ¿Qué métodos conoces (Basic, OAuth, JWT, etc.)?

Para manejar la autenticación en una API HTTP/HTTPS, utilizaria el encabezado Authorization con el método Bearer Token (JWT), siempre preferiblemente mediante HTTPS para cifrar la comunicación.

¿Qué es un proxy inverso (reverse proxy) y cómo se utiliza en entornos HTTP/HTTPS?

Un reverse proxy es un servidor que recibe las peticiones del cliente y las reenvía a uno o varios servidores backend.
Formas de utilizacion:

- Balanceo de carga.
- Terminar conexiones TLS (SSL offloading).
- Cachear contenido estático.
- Proteger servidores internos.

¿Cómo implementarías una redirección automática de HTTP a HTTPS en un servidor?

Yo lo implementaria en Apache y lo haria de la siguiente forma:

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

¿Cómo mitigarías un ataque de denegación de servicio (DDoS) en un servidor HTTP?
Aplicaria las siguiente estrategia multicapa:

- Firewall y rate limiting (ej. limitar requests por IP).
- CDN (Cloudflare, Akamai) para absorber tráfico.
- Load balancing entre varios servidores.
- CAPTCHAs para distinguir bots de humanos.
- Escalado automático en la nube para manejar picos de tráfico.

¿Qué problemas podrías enfrentar al trabajar con APIs que dependen de HTTP, y cómo los resolverías?

- Latencia alta → usar caché y compresión (GZIP).
- Falta de seguridad (HTTP plano) → migrar a HTTPS.
- Errores de CORS → configurar correctamente Access-Control-Allow-Origin.
- Versionado de la API → mantener versiones (/api/v1/).

¿Qué es un cliente HTTP? ¿Mencionar la diferencia entre los clientes POSTMAN y CURL?

Un cliente HTTP es una herramienta que permite enviar peticiones a un servidor web.

- cURL: herramienta de línea de comandos, ligera y scriptable.
- Postman: aplicación con interfaz gráfica, útil para pruebas y documentación de APIs.

Preguntas de GIT

¿Qué es GIT y para qué se utiliza en desarrollo de software?

GIT es un sistema de control de versiones distribuido. Se usa para:

- Guardar el historial de cambios en el código.
- Colaborar en equipo sin sobrescribir trabajo.
- Volver a versiones anteriores del proyecto.

¿Cuál es la diferencia entre un repositorio local y un repositorio remoto en GIT?

Un repositorio local se encuentra en un solo dispositivo como podria ser una computadora personal y en cambio un repositorio remoto es el cual se encuentra alojado en un servidor ya sea github, gitlab, etc.

¿Cómo se crea un nuevo repositorio en GIT y cuál es el comando para inicializarlo?
Explica la diferencia entre los comandos git commit y git push.

Para crear un nuevo repositorio en GIT debo hacer click derecho en la carpeta deseada, iniciar GIT BASH y luego poner el comando git init.
La diferencia entre commit y push es que al realizar git commit lo que estamos haciendo es solamente guardar los cambios en el repositorio local y en cambio con git push lo que

hacemos es enviar estos cambios al repositorio remoto.
¿Qué es un "branch" en GIT y para qué se utilizan las ramas en el desarrollo de software?

Un branch (rama) es una línea de desarrollo independiente. Se usa para:

- Trabajar en nuevas funcionalidades.
- Corregir errores sin afectar la rama principal (main o master).

¿Qué significa hacer un "merge" en GIT y cuáles son los posibles conflictos que pueden surgir durante un merge?

Merge significa unir los cambios de una rama con otra. Los conflictos ocurren cuando dos ramas modifican la misma línea o archivo de forma distinta. GIT pide que el desarrollador elija qué versión conservar. Esto puede ser un grave problema si no hay una buena comunicacion de desarrollo.

Describe el concepto de "branching model" en GIT y menciona algunos modelos comunes (por ejemplo, Git Flow, GitHub Flow).

Un branching model es la estrategia de cómo se organizan y usan las ramas.
Ejemplos:

- Git Flow: ramas para features, releases, hotfixes.
- GitHub Flow: ramas cortas desde main, pull request y merge.

¿Cómo se deshace un cambio en GIT después de hacer un commit pero antes de hacer push?
Si sucede esto hay dos formas de deshacer este cambio.

1) git reset --soft HEAD~1 Esto deshace el último commit pero deja cambios en staging (osea, los cambios permanecen pero ya no están en el historial.)
2) git reset --hard HEAD~1 Esto deshace commit y borra los cambios

¿Qué es un "pull request" y cómo contribuye a la revisión de código en un equipo?

Un Pull Request (PR) es una propuesta de cambios desde una rama hacia otra (ej. de feature-x a main).
Sirve para:

- Revisar el código entre compañeros.
- Discutir mejoras antes de integrarlo.
- Que las actualizaciones no se integren al main o prod sin una revision por parte de otro integrante.

¿Cómo puedes clonar un repositorio de GIT y cuál es la diferencia entre git clone y git pull?
Para clonar un repositorio lo primero que tenes que hacer es conseguir el URL y luego mediante el bash realizas el git clone con la URL.

- git clone: copia un repositorio remoto en tu computadora la primera vez.
- git pull: actualiza tu repositorio local trayendo los cambios más recientes del remoto

Preguntas de GIT

¿Qué es Node.js y por qué es una opción popular para el desarrollo backend?

Node.js es un entorno de ejecución de JavaScript del lado del servidor, basado en el motor V8 de Google Chrome.

Es popular porque:

- Permite usar JavaScript en el backend y frontend → mismo lenguaje en toda la app.
- Tiene alto rendimiento gracias a su arquitectura asincrónica y no bloqueante.
- Cuenta con un ecosistema enorme de librerías gracias a npm.
- Escala muy bien para aplicaciones con muchas conexiones concurrentes.

¿Cómo funciona el modelo de I/O no bloqueante en Node.js y cómo beneficia el rendimiento de una aplicación backend?

En Node.js, cuando se ejecuta una operación de entrada/salida (leer archivos, consultar una BD, llamar a una API), esta no bloquea la ejecución del programa:

- La tarea se delega al sistema.
- Node sigue ejecutando otras instrucciones.
- Cuando la tarea termina, notifica mediante un callback, promesa o async/await.

El beneficio que nos da es que permite manejar muchas solicitudes al mismo tiempo sin necesidad de crear un hilo por cada una, siendo asi más eficiente y escalable.

¿Qué es el Event Loop en Node.js y cuál es su papel en la ejecución de código asincrónico?

El Event Loop es el mecanismo central de Node.js que gestiona la ejecución de operaciones asincrónicas.

- Revisa constantemente una cola de eventos/tareas pendientes.
- Cuando una operación (ej. acceso a BD) finaliza, coloca su callback/promesa en la cola.
- El Event Loop los va ejecutando en orden cuando el call stack queda libre.

Esto permite que Node.js sea monohilo pero altamente concurrente.

¿Cuál es la diferencia entre require() y import en Node.js?

- require(): forma clásica de importar módulos en Node.js (sistema CommonJS).
- import: sintaxis moderna de ES Modules (ESM), estándar de JavaScript.

¿Qué es npm y cuál es su función en el ecosistema de Node.js?

npm (Node Package Manager) es el gestor de paquetes de Node.js.

Sirve para:

- Instalar librerías y frameworks.
- Gestionar versiones y dependencias del proyecto.
- Ejecutar scripts de automatización definidos en package.json.

¿Cómo se inicializa un proyecto de Node.js usando npm y cuál es el propósito del archivo package.json?

Para crear un proyecto de Node.js:

npm init -y

Esto genera un archivo package.json que:

- Describe el proyecto (nombre, versión, autor).

- Registra las dependencias necesarias.
- Define scripts de ejecución (ej. npm start).

¿Qué son las dependencias en npm y cómo se instalan? Explica la diferencia entre dependencias y dependencias de desarrollo.

- Dependencias normales (dependencies): necesarias para que la app funcione en producción (ej. express).

```
npm install express
```

- Dependencias de desarrollo (devDependencies): solo se usan en el entorno de desarrollo (ej. Jest para testing).

```
npm install jest -D
```

¿Cómo puedes gestionar versiones específicas de paquetes en npm y para qué sirve el archivo package-lock.json?

- Puedes instalar una versión exacta:

```
npm install express@4.17.1
```

- package-lock.json: guarda las versiones exactas instaladas para que todo el equipo use las mismas → evita incompatibilidades.

¿Qué es nest.js cómo se usa en Node.js para construir aplicaciones backend?

Nest.js es un framework para construir aplicaciones backend sobre Node.js usando TypeScript.

Se basa en conceptos como:

- Módulos para organizar el código.
- Controladores para manejar rutas.
- Servicios para la lógica de negocio.

Ideal para proyectos grandes, escalables y con arquitectura limpia (inspirada en Angular).

¿Cómo se manejan errores en Node.js y cuál es la diferencia entre callbacks, promesas y async/await para manejar código asincrónico?

Existen tres enfoques:

- Callbacks: patrón clásico. El error va como primer parámetro.

```
fs.readFile('file.txt', (err, data) => {
  if (err) return console.error(err);
  console.log(data.toString());
});
```

- Promesas: más moderno, usando .then() y .catch().

```
fetch(url)
  .then(res => res.json())
  .catch(err => console.error(err));
```

- async/await: sintaxis más clara, permite usar try...catch.

```
try {
  const res = await fetch(url);
  const data = await res.json();
```

```
} catch (err) {
  console.error(err);
}
```

Práctica

Para realizar los ejercicios prácticos deberás contar en tu ambiente de trabajo con las siguientes herramientas:

- Postman
- GIT
- Node.js instalado

La entrega deberá realizarse en un repositorio de código público que se deberá compartir al equipo de reclutamiento. El repositorio deberá contener tanto la parte teórica como la práctica

Actividad práctica número 1
Pasos:

1) Realizar una petición GET a la siguiente URL a través de Postman: https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json

Ejemplo con CURL:

```
curl --location --request GET
'https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json' \
--header 'Content-Type: application/json'
```

2) Realizar una petición POST a la siguiente URL a través de Postman: https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json y con el siguiente body:

```
{
   "name":"TuNombre",
   "suraname":"TuApellido",
   "birthday":"1995/11/16/",
   "age":29,
   "documentType":"CUIT",
   "documentNumber":2012345678
   1
}
```

Reemplazar los campos por los valores personales tuyos.

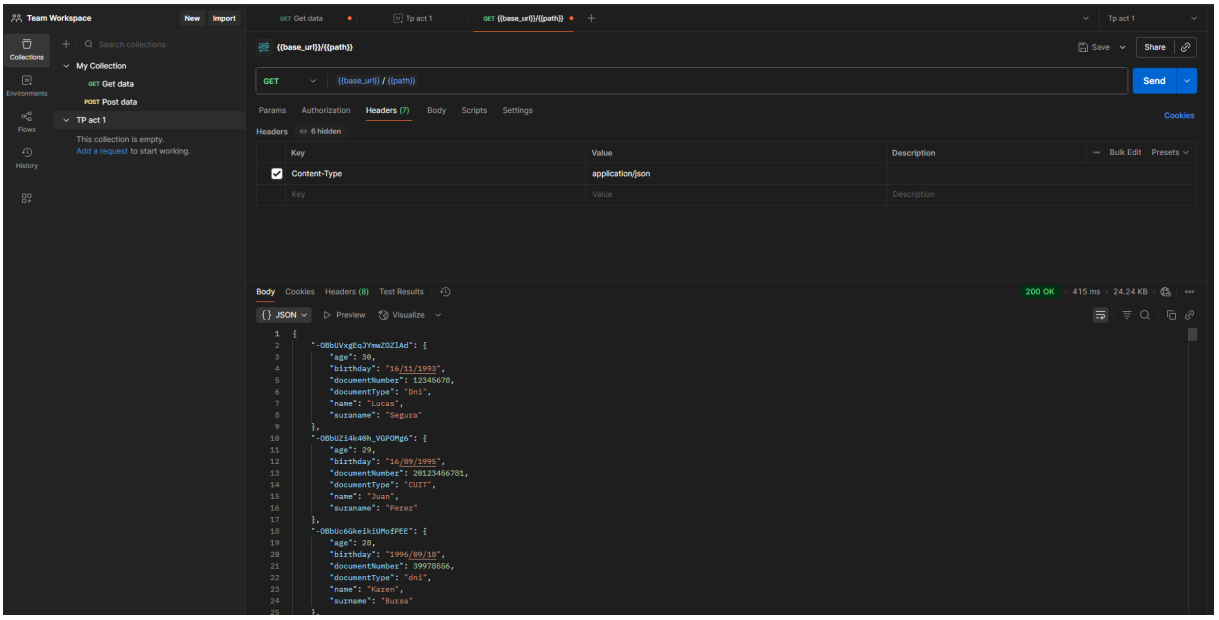
3) Volver a realizar el GET del punto número 1.

Preguntas/Ejercicios:

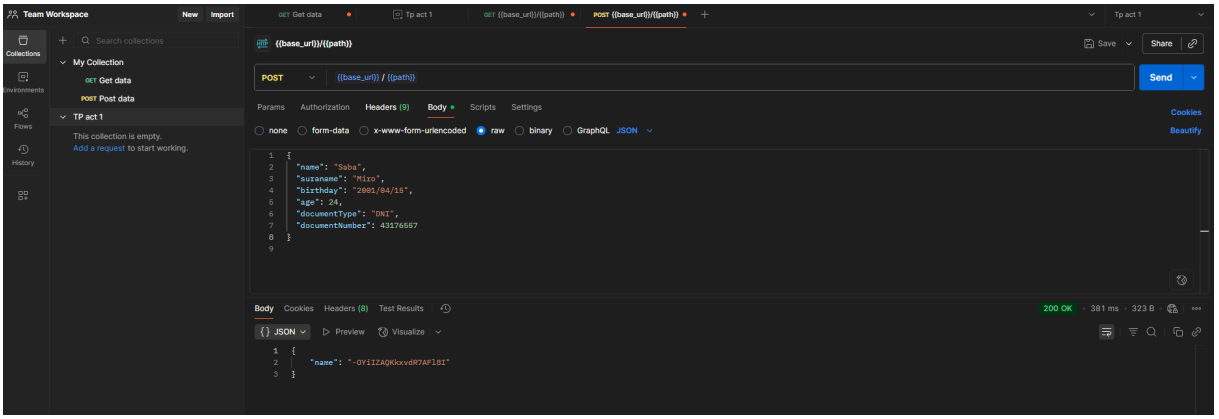
1. Adjuntar imagenes del response de un GET y de un POST de cada punto
2. ¿Qué sucede cuando hacemos el GET por segunda vez, luego de haber ejecutado el POST?

1)

Response del primer GET



Response del POST



2)

Lo que sucede cuando hacemos el GET por segunda vez luego de haber realizado el POST (que se puede observar en la imagen de arriba) es que se agrega al archivo JSON mis datos que envie anteriormente mediante el POST.

Actividad práctica número 2

Realizar un script en Node.JS que realice un GET a la URL: https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json y muestre por pantalla todos los registros.

RESUELTO EN TP-NODE/ACT2/get.js

Actividad práctica número 3:

Realizar un Servicio Web en nestjs que permita recibir una petición post con el formato:

```
{
   "name":"TuNombre",
   "suraname":"TuApellido",
   "birthday":"1995/11/16/",
   "age":29,
   "documentType":"CUIT",
   "documentNumber":2012345678
   1
}
```


una vez recibida la petición deberá ejecutar otra petición hacia el servicio web de reclutamiento:
https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json

Adicionalmente el servicio web de nestjs deberá tener las siguientes características:

1. El campo Name y Surename deberán empezar siempre con la primer letra de cada palabra en mayúscula, si se recibe una petición con otro formato deberá normalizarse al formato esperado
2. Validar que el campo birthday tenga un formato válido en el formato YYYY/MM/DD. La fecha proporcionada no podrá ser posterior al día de hoy ni anterior al 1900/01/01. En caso que no se cumpla alguna petición se deberá rechazar la petición
3. El campo age deberá ser un número entero. En caso que no se cumpla alguna petición se deberá rechazar la petición
4. Los valores posibles de documentType son: “CUIT” o “DNI” si se envía otros valores se deberá rechazar la petición

RESUELTO EN TP-NODE/tp-nest