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?

Es un protocolo de comunicación, el cual, su función principal es permitir la conexión entre un cliente (navegador de un usuario final y aplicaciones) y un servidor web. El cliente y el servidor se transfieren datos entre sí gracias a este sistema.

¿Cuál es la diferencia entre HTTP y HTTPS?

La diferencia radica en la seguridad. Mientras HTTP envía los datos en texto plano, HTTPS aplica un cifrado a los datos, por lo cual, estos no pueden ser interceptados con facilidad, ofreciendo una capa adicional de seguridad.

¿Cómo funciona el proceso de cifrado en HTTPS?

El cliente solicita la comunicación al servidor, luego el servidor envia su certificado SSL/TLS al navegador. El cliente usa la clave pública del servidor y se crea una clave secreta que es enviada al servidor. Con esa información, cliente y servidor generan una llave, que posteriormente se utilizará para cifrar y descifrar los datos.

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

Es un archivo emitido por una autoridad certificadora. Su valor está en la certificación de la identidad del sitio web.

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

También denominados verbos HTTP, son acciones que se realizan en los datos.

- GET: se obtiene información.
- POST: enviar datos (generalmente nuevos).
- PUT: actualizar datos.
- DELETE: eliminar datos

Explica las diferencias entre los métodos HTTP GET y POST.

Las diferencias más comunes son:

- GET transporta los datos en la URL y POST en la sección body.
- GET no aplica ningún cambio, POST si.

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

Es la respuesta del servidor para describir en que condición se encuentra la petición. Los códigos son numéricos y pertenecen a distintas categorías de estado. Algunos de estos son:

- 200 OK → Éxito
- 202 OK → Creado
- 400 OK → Error del Cliente
- 401 OK → Sin autorización
- 500 OK → Error del Servidor

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

Son metadatos (datos adicionales) que viajan en las peticiones y respuestas. Suelen describir la forma de tratar los datos, información sobre el origen y otros detalles útiles para utilizar los datos del mensaje de forma apropiada. Algunos ejemplos son:

- Content-Type: application/json
- Authorization: Bearer <token>

¿En qué consiste el concepto de "idempotencia" en los métodos HTTP? ¿Qué métodos cumplen con esta característica?

Son aquellos métodos a los cuales, si realizamos multiples veces la misma petición, se producirá el mismo resultado.

Algunas peticiones idempotentes son: GET, PUT, DELETE

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

Es cuando el servidor le dice al cliente que el recurso no está en la URL pedida, sino en otra. Esto se hace con códigos 3XX y la cabecera Location. Se usa mucho para pasar de HTTP a HTTPS, para mover recursos a nuevas rutas, o en SEO para unificar URLs.

Preguntas técnicas y de seguridad en HTTP/HTTPS:

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

TLS cifra los datos con algoritmos simétricos, pero no es suficiente para asegurar la integridad de los datos. Por lo cual, se agrega una firma o un código de autenticación para detectar cambios en el mensaje durante el camino. La información no solo va encriptada, también protegida contra alteraciones.

¿Qué diferencia hay entre un ataque de "man-in-the-middle" y un ataque de "replay" en un contexto HTTPs?

Man-in-the-middle es cuando el atacante intercepta el mensaje y puede leer e incluso modificar los datos que viaja por el método HTTP.
Replay es el caso en el que el atacante intercepta una petición valida y la reenvía para intentar obtener un acceso no autorizado o realizar una transacción fraudulenta.

Explica el concepto de "handshake" en HTTPS.

Es el proceso inicial entre el cliente y el servidor, donde establecen la conexión segura. Durante el proceso, ambas parte acuerdan el protocolo a utilizar, validan los certificados y concuerdan la clave para el cifrado de la información.

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

Es una política que le dice al navegador: “con este sitio **solo** usá HTTPS”.

Es un encabezado de seguridad:

```
Strict-Transport-Security: max-age=31536000
includeSubDomains; preload
```

Eso indica al navegador que solo debe usar HTTPS con ese dominio, incluso si el usuario escribe http://, para evitar que alguien te fuerce a bajar a una conexión insegura.

¿Qué es un ataque "downgrade" y cómo HTTPS lo previene?

Es cuando el atacante intenta que la comunicación entre el cliente y el servidor sea a través de una versión vieja o insegura de TLS. HTTPS lo previene en el handshake ya que se establece una versión mínima valida. Ahora los navegadores modernos bloquean los cifrados antiguos

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

Es un mecanimos de seguridad que controla que dominios pueden hacer peticiones a un servidor distinto al del origen. Por ejemplo: un servidor frontend cuando quiere comunicarse con la API creada por el servidor backend percibe un bloqueo CORS por seguridad.
Para habilitar el acceso de origen cruzado, se añaden cabeceras al servidor, ej:

```
Access-Control-Allow-Origin: https://midominio.com
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Headers:Authorization
```

Usualmente, es para evitar que scripts maliciosos de otro domino accedan a nuestros recursos.

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

Authorization: en cada petición, debe ingresarse el token en la cabecera de forma explícita.

Cookie: el navegador automáticamente la manda sin necesidad de que el desarrollador la ingrese en cada petición. Pero el peligro que tiene es que la cookie puede ser obtenida por un atacante.

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

Son normas o directivas que el servidor envía al navegador para indicarle como se deben manejar ciertos contenidos. Ayudan a proteger a la aplicación web de ataques comunes.

1- **Content-Security-Policy (CSP)**

- Permite definir de dónde se puede cargar contenido (scripts, imágenes, estilos, iframes, fuentes, etc.).

  Ejemplo:

  - Content-Security-Policy: default-src 'self'; script-src 'self' https://apis.google.com

Esto obliga a que solo se ejecuten scripts del propio sitio o de Google APIs.

- **Mitiga**:
  - **XSS (Cross-Site Scripting)**: bloquea scripts maliciosos inyectados.
  - **Data injection**: evita que se carguen recursos de orígenes no autorizados.

2- **X-Frame-Options**

- Indica si una página puede ser cargada dentro de un <iframe> en otro sitio.
- Valores típicos:
  - DENY → nunca permite ser cargado en iframes.
  - SAMEORIGIN → solo permite iframes en el mismo dominio.
  - ALLOW-FROM https://ejemplo.com → solo permite un dominio específico.
- **Mitiga**:
  - **Clickjacking**: ataques donde un atacante “superpone” la página dentro de un iframe invisible para engañar al usuario y hacer clic en elementos ocultos.

¿Cuáles son las diferencias entre HTTP/1.1, HTTP/2 y HTTP/3?

Versión	**Características principales**

HTTP/1.1	- Texto plano. - Una conexión por recurso (aunque con *keep-alive*). - Lento para sitios con muchos recursos.
HTTP/2	- Binario, multiplexado (varias peticiones/respuestas en una sola conexión). - Compresión de cabeceras (HPACK). - Más eficiente en latencia.
HTTP/3	- Basado en QUIC (sobre UDP, no TCP). - Mejor en conexiones inestables (móviles). - Cifrado TLS 1.3 obligatorio desde el inicio.

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

Es un header que mantiene la conexión TCP abierta entre el cliente y el servidor para posteriores peticiones. Evita la sobrecarga de crear una conexión para cada recurso. Es mas eficiente, mejora la latencia y el consumo del CPU.

Preguntas de implementación práctica:

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

La autenticación en una API se puede manejar de distintas formas, y la elección depende de cuánta seguridad y complejidad necesite el sistema.

Basic Auth: es simple, un usuario y contraseña en cada request. Hoy casi no se usa porque es inseguro si no está sobre HTTPS, y no es lo ideal para APIs modernas.

OAuth 2.0: se usa mucho cuando hay que dar acceso delegado, por ejemplo, que una app de terceros pueda entrar a datos de Google o Facebook en nombre del usuario.

JWT (JSON Web Token): es el que más conozco. El servidor genera un token firmado con datos como el ID del usuario o sus roles, y ese token viaja en cada request dentro del header Authorization: Bearer <token>. La ventaja es que no hace falta guardar sesiones en el servidor, solo validar la firma. Además, podés ponerle expiración y usar refresh tokens para mayor seguridad. La contra es que, si un token se roba, es difícil invalidarlo antes de que caduque.

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

Un reverse proxy es un servidor que se ubica entre el cliente y los servidores de aplicación. Los clientes no se conectan directamente al servidor real, sino al proxy, que reenvía las solicitudes al backend adecuado y luego devuelve la respuesta.

Administra certificados, balancea carga, puede hacer caché, mejorar la seguridad ocultando la infraestructura y optimizar el rendimiento de la aplicación.

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

Lo común es configurar el servidor web (Apache y Nginx) para que cualquier request en puerto 80 (HTTP) haga un **redirect 301** a la versión HTTPS.
Ejemplo en Nginx:

```
server {
    listen 80;
    server_name midominio.com;
    return 301 https://$host$request_uri;
}
```

¿Cómo mitigarías un ataque de denegación de servicio (DDoS) en un servidor HTTP?

Mitigar ese tipo de ataque no suele resolverse con una sola estrategia, algunas pueden ser:

- Usar una Web Application Firewall o una reverse proxy que filtre los datos.
- Limitar el rate de request por IP.
- **CAPTCHAs / JavaScript challenges**: evita tráfico automatizado.
- Usar un servicio en la nube que ofrezca protección contra ataques de este estilo.

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

Seguridad: Las APIs que usan HTTP son vunerables a SQL injection, XSS o MITM. Siempre hay que usar HTTPS y TSL. También es necesario utilizar tokens de autenticación para accesos legitimos.

Rendimiento: Suelen ser lentas debido a la distancia entre los extremos. Usar un CDN para almacenar en caché las respuestas en nodos mas cercanos al usuario. Reducir el tamaño de la respuesta por medio de compresores mejora la latencia.

Versionado: La API, podría necesitar cambios que pueden generar problemas de compatibilidad con clientes antiguos. Lo ideal, sería implementar un sistema de transición

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

Es un programa o una herramienta que realiza peticiones HTTP a un servidor. Algunos pueden ser: navegadores web, aplicaciones móviles o de escritorio, línea de comandos, entre otros.

Característica	**Postman**	**cURL**
Interfaz	Gráfica de usuario (GUI)	Línea de comandos (CLI)
Uso Principal	Desarrollo y pruebas de APIs	Automatización y scripting
Curva de Aprendizaje	Baja, muy intuitivo	Más alta, requiere conocer sintaxis
Flexibilidad	Alta, con colecciones, ambientes, y pruebas	Muy alta, se puede integrar en scripts

Preguntas de GIT

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

Git es un sistema de control de versiones que registra los cambios en los archivos de un proyecto. Permite a los desarrolladores colaborar sin conflictos, volver a versiones anteriores y gestionar el historial del código de manera eficiente.

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

Repositorio Local: Es la copia completa del proyecto en tu computadora. Los cambios que haces **(commit)** se guardan aquí.

Repositorio Remoto: Es la versión del proyecto que está en un servidor (como GitHub), compartida con todo el equipo. Los cambios se sincronizan entre el repositorio local y el remoto.

¿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 una carpeta, usa el comando **git init**.

git commit: guarda tus cambios en el repositorio local.
git push: envía esos cambios del repositorio local al repositorio remoto, haciéndolos visibles para otros.

¿Qué es un "branch" en GIT y para qué se utilizan las ramas en el desarrollo de software?

Una rama **(branch)** es una línea de desarrollo independiente. Se usan para:

- **Aislar el trabajo:** Permiten a los desarrolladores trabajar en nuevas funciones o arreglos sin afectar el código principal.
- **Colaboración en paralelo:** Varios desarrolladores pueden trabajar en sus propias ramas al mismo tiempo.

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

Un **merge** en Git básicamente significa juntar los cambios de una rama con otra. Se usa, por ejemplo, cuando querés integrar una nueva funcionalidad a la rama principal después de que ya fue revisada y aprobada en la rama de desarrollo.

Ahora, a veces el merge no es tan automático. Los conflictos aparecen cuando dos ramas

cambian las mismas líneas de un archivo y Git no sabe cuál elegir. En ese caso te va a pedir que lo resuelvas vos de manera manual, decidiendo qué parte del código conservar o cómo combinarlas.

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

Un **modelo de ramificación** (*branching model*) no es más que un conjunto de reglas que un equipo define para decidir cómo crear, manejar y fusionar ramas en Git. La idea es evitar el caos y mantener el trabajo ordenado.

Algunos de los modelos más usados son:

- **Git Flow**: bastante completo y estructurado. Tiene ramas con roles bien definidos: main (producción), develop (desarrollo en curso) y otras para *features*, *releases* y *hotfixes*. Es muy útil en proyectos grandes porque te da un marco claro de cómo avanzar.
- **GitHub Flow**: mucho más sencillo. Se trabaja con una sola rama principal (main), que siempre debería estar lista para desplegar. Cada cambio (sea una nueva funcionalidad o un bugfix) se hace en una rama aparte y, cuando está terminado, se integra a main mediante un pull request.

¿Cómo se deshace un cambio en GIT después de hacer un commit pero antes de hacer push?

Para deshacer un *commit* después de haberlo hecho localmente, pero antes de enviarlo al repositorio remoto (push), puedes usar el comando git reset.

- **git reset --soft HEAD~1**: Deshace el último commit pero mantiene los cambios en tu área de staging.
- **git reset --mixed HEAD~1:** Deshace el último commit y quita los cambios del área de staging, pero los mantiene como archivos modificados. Es el comportamiento predeterminado de git reset.
- **git reset --hard HEAD~1:** Deshace el último *commit* y elimina permanentemente todos los cambios que contenía. **No recomendable**

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

Un **pull request** (PR) es básicamente una solicitud para que los cambios de una rama se unan con otra dentro de un repositorio remoto (por ejemplo, en GitHub). Es la forma más común que tienen los equipos para colaborar y revisar el código antes de que llegue a la rama principal.

En la práctica, un PR ayuda mucho en la revisión de código:

Notificación: avisa al resto del equipo que hay cambios listos para mirar.

Feedback: permite que otros desarrolladores comenten, sugieran mejoras o marquen errores directamente en las líneas del código.

Control de calidad: asegura que lo que se está subiendo cumpla con los estándares del proyecto y no meta bugs innecesarios.

¿Cómo puedes clonar un repositorio de GIT y cuál es la diferencia entre git clone y git pull?

Para clonar un repositorio en Git se usa el comando:

git clone <URL>

Esto baja una copia completa del repositorio remoto (todas las ramas y el historial de commits) y lo guarda en tu máquina como un nuevo repositorio local.

Diferencia entre git clone y git pull:

- **git clone**: lo usás una sola vez, cuando necesitás traer por primera vez un repositorio que no tenías en tu computadora.
- **git pull**: lo usás después, de forma habitual, para actualizar tu copia local con los cambios que se hicieron en el repositorio remoto. Básicamente es como hacer un git fetch (traer) seguido de un git merge (integrar).

Preguntas de Node.js

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

Node.js es un entorno de ejecución que permite correr JavaScript fuera del navegador, es decir, en el servidor. Está construido sobre el motor V8 de Chrome, lo que lo hace muy rápido a la hora de procesar código.

Es popular en backend por varias razones:

- Permite usar **JavaScript en todo el stack** (frontend y backend).
- Cuenta con un **ecosistema enorme de librerías** a través de npm, que facilita agregar funcionalidades sin tener que reinventar la rueda.
- Es **ligero y escalable**, lo que lo hace muy usado en aplicaciones modernas, APIs y microservicios.

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

En Node.js, las operaciones de E/S (como leer un archivo o consultar una base de datos) no bloquean el programa. Es decir, Node puede seguir haciendo cosas mientras espera que esas tareas terminen.

Funciona así:

1. Pides algo, como leer un archivo.
2. Node le dice al sistema operativo “ocúpate de esto” y sigue con lo siguiente, sin esperar.
3. Cuando termina, el sistema operativo avisa a Node.
4. Node pone la función que manejará el resultado en una cola y la ejecuta cuando puede.

Esto es gracias al **Event Loop**, que siempre está revisando qué tareas pendientes hay. Por eso Node es genial para apps con muchas conexiones al mismo tiempo, como servidores web o chats en tiempo real.

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

Cuando una tarea es **asincrónica** (como leer un archivo o hacer una petición web), el Event Loop la envía a un segundo plano para que no bloquee al programa. Mientras la tarea se procesa, el Event Loop sigue ejecutando el resto del código. Cuando la tarea asincrónica termina, su función de *callback* (el código a ejecutar después) se coloca en una "cola de eventos".

El Event Loop está constantemente vigilando esta cola. En cuanto la pila de tareas principales está vacía, toma la siguiente función de la cola y la ejecuta. Esto permite a Node.js ser muy eficiente y manejar muchas operaciones a la vez sin detenerse a esperar.

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

La diferencia principal es la sintaxis y el contexto en el que se usan:

- **require()** es la forma original de Node.js para cargar módulos. Es una función que carga módulos de forma **sincrónica** (es decir, el código se detiene hasta que el módulo se carga por completo).
- **import** es la sintaxis más moderna, parte de los módulos de ES6 (ECMAScript 2015). Es **asincrónica** y es la forma estándar de trabajar con módulos en JavaScript moderno. Necesitas configurar tu proyecto para usarlo.

¿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, y es la herramienta más usada en su ecosistema. Su función principal es permitir a los desarrolladores instalar, gestionar y compartir módulos (librerías de código) para sus proyectos. Funciona como una especie de tienda de aplicaciones para desarrolladores, donde puedes descargar fácilmente herramientas y funcionalidades creadas por otros, lo que acelera el desarrollo y evita reinventar la rueda.

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

Para arrancar un proyecto de Node.js con npm, primero abre la terminal y ve a la carpeta donde quieres crear tu proyecto. Luego, ejecuta: *npm init*

Esto te llevará por un proceso paso a paso donde te pedirá información sobre tu proyecto: nombre, versión, descripción, autor, entre otras cosas.

Al terminar, se creará un archivo llamado **package.json**, que es como la “identidad” de tu proyecto. Básicamente, sirve para tres cosas principales:

1. **Identificar tu proyecto:** Guarda datos como el nombre, la versión y el autor.
2. **Gestionar dependencias:** Aquí se listan todas las librerías que tu proyecto necesita (tanto para desarrollo como para producción). Esto hace que otros desarrolladores puedan instalar todo fácilmente con un solo comando: *npm install*
3. **Automatizar tareas:** Puedes definir scripts para arrancar el proyecto, probarlo o desplegarlo. Por ejemplo, con npm start puedes iniciar tu aplicación sin tener que escribir todo el comando cada vez.

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

Las **dependencias** son librerías o paquetes que el proyecto necesita para funcionar. Por ejemplo, si tengo una app que usa **Express** para crear un servidor, Express sería una dependencia.

Ejemplo de instalación: *npm install nombre-del-paquete*

El paquete luego se carga en la carpeta **node_modules** y se agrega en el package.json, en la sección **dependencias**

Las dependencias se necesitan para el proyecto en producción, pero las dependencias de desarrollo solo se usan durante el proceso de creación del producto, el usuario final no las necesita.

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

El package-lock.json es como un índice de todas las librerías que tenés instaladas y sus versiones, incluyendo las dependencias de tus dependencias.

Sirve para:

- Que todos los que trabajen en el proyecto tengan **exactamente las mismas versiones**.
- Evitar sorpresas cuando alguien más instala los paquetes con npm install.

En pocas palabras: package.json dice **qué paquetes querés**, y package-lock.json dice **exactamente qué versiones se instalaron**.

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

Básicamente, NestJS es un framework que da una estructura clara: módulos, controladores y servicios, lo que hace que el código sea más fácil de mantener y escalar. Con NestJS podemos crear **APIs REST o GraphQL**, manejar rutas, validar datos y usar middlewares de manera muy intuitiva gracias a los decoradores.

NestJS crea un entorno de carpetas y archivos que nos permite arrancar con la lógica de negocio

Nos ayuda a escribir código más limpio, modular y profesional en Node.js**,** sin tener que preocuparnos demasiado por la estructura básica de la app.

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

El manejo de errores en node se divide en si son sincronicos o asincrónicos.
Para sincronico, usamos **try-catch.** En caso de ser errores asincrónicos, podemos usar callbacks, promesas o async/await.

Un **callback** es una función que se ejecuta cuando termina la tarea. En Node.js, por convención el primer parámetro del callback es el error (si hay alguno).

- **Pros**: Funciona en todas las funciones de Node actuales.
- **Contras:** su mayor problema es el “callback hell” si anidamos muchas funciones de calback, es difícil de mantener.

Las **promesas** son objetos que representan una operación asincrónica que **podrá completarse o fallar** en el futuro

- **Pros:** evita la anidación excesiva de callbacks y se puede encadenar (.then().then()).
- **Contras:** aún puede ser menos legible si hay muchas operaciones consecutivas.

Async/await es la forma más moderna y parece código sincrónico, pero sigue siendo asincrónico:

- **Pros:** el código es más limpio y fácil de leer.
- **Contras:** hay que usarlo dentro de funciones marcadas con async, puede tomar tiempo acostumbrarse.

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":20123456781
}
```

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?

> Podemos ver el JSON actualizado con los datos que ingresamos en el método 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.

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":20123456781
}
```

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