Preguntas generales sobre HTTP/HTTPS

¿Qué es HTTP y cuál es su función principal?

HTTP es un protocolo de comunicación (conjunto de reglas y normas) de la capa de aplicación del modelo TCP/IP que generalmente opera sobre TCP, otro protocolo de la capa de transporte. Originalmente su función principal era lograr la comunicación entre servidores web y clientes, aunque en la actualidad es más parecido a un protocolo de la capa de transporte porque provee los medios para que procesos ejecutándose en dos puntos de una red se transmitan información.

¿Cual es la diferencia entre HTTP y HTTPS?

La diferencia entre HTTP y HTTPS es que el segundo se utiliza sobre TLS (Seguridad de la capa de transporte), el cual es un paquete de seguridad implementado como una capa intermedia entre la de aplicación y transporte. TLS genera una conexión segura entre dos procesos que se comunican a través de sockets o puntos de comunicación (su nombre es una analogía a los enchufes eléctricos).
Utilizarlo HTTP junto a TLS logramos que las comunicaciones sean seguras porque garantiza el cifrado de los datos enviados.

¿Cómo funciona el proceso de cifrado en HTTPS?

El proceso de cifrado en HTTPS lo podemos descomponer en dos partes:

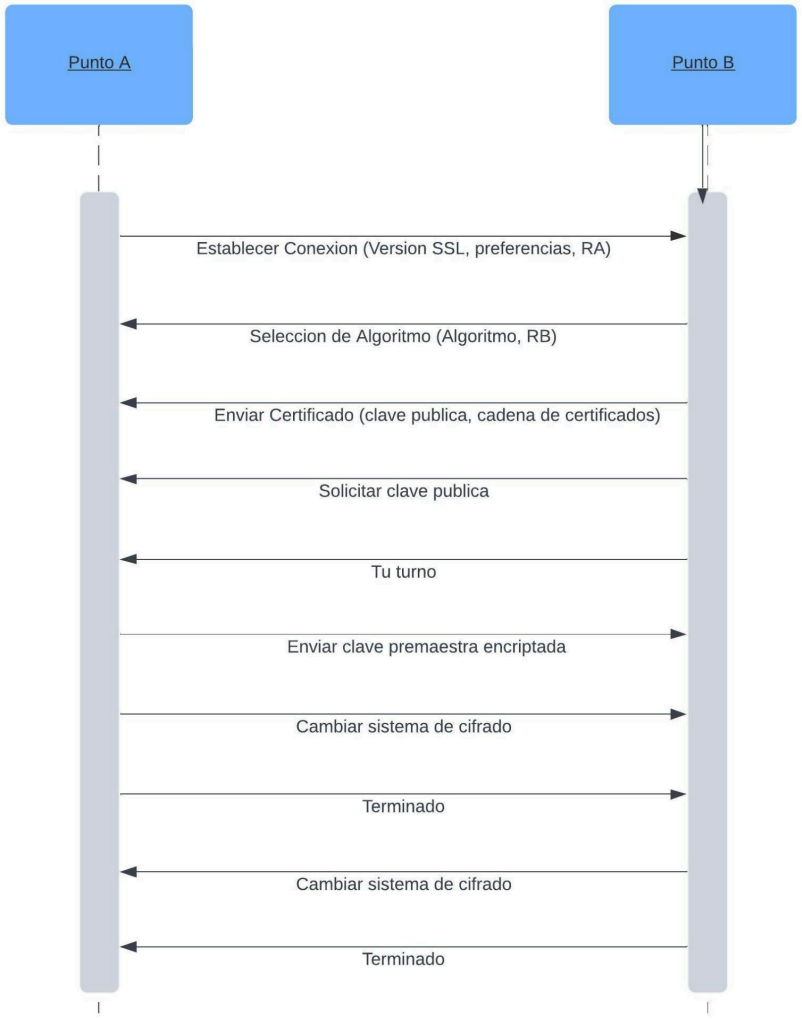
- Establecimiento de la conexión segura (handshake o apretón de manos)
- Comunicación cifrada

Paso a explicar el subproceso de establecimiento de la conexión segura entre un punto A y un punto B porque considero que es fundamental:

- A envía un mensaje para establecer una conexión especificando la versión de SSL que posee, sus preferencias con respecto a algoritmos criptográficos y de compresión, y un valor aleatorio (llamemoslo RA) que garantiza que la sesión sea única.
- B selecciona algún algoritmo que A soporte y envía su propio valor aleatorio (llamemoslo RB)
- B envía un certificado que contiene su clave pública, y si no está firmada por una entidad bien conocida, también envía una cadena de certificados que pueden seguirse hasta encontrar uno.
- A verifica la clave pública de B, a su vez, B puede enviar algunos mensajes más como solicitar la clave pública de A
- B envía un mensaje indicando que es el turno de A
- A selecciona una clave premaestra aleatoria de 384 bits, la encripta con la clave pública de B y se la envía.
- En este punto tanto A como B pueden calcular la clave de sesión, por lo que A indica a B que cambie al nuevo sistema de cifrado y que finalizó el subproceso de establecimiento. La clave de sesión utilizada para cifrar los datos se calcula combinando la clave premaestra con los valores RA y RB de manera compleja

- B confirma el recibimiento enviando indicando también cambiar al nuevo sistema de cifrado y que finalizó su comunicación.

Diagrama de secuencia del subproceso de establecimiento de conexión:



Link al diagrama

El proceso de comunicación cifrada comienza cuando finaliza el handshake y se realiza utilizando la clave de sesión acordada. Todo mensaje de comunicación se cifra con un algoritmo de cifrado simétrico.
Finalmente se cierra la conexión de forma segura y se elimina la clave de sesión para que no se reutilice.

¿Que es un certificado SSL/TLS y cual es su importancia en HTTPS?

Un certificado SSL/TLS es un archivo que autentica la identidad de quien lo posee para permitir la conexión cifrada. Estos certificados son emitidos por entidades autorizadas.
Su importancia es fundamental en HTTPS ya que como se mencionó en la pregunta anterior, el envío de los certificados es parte del subproceso de establecimiento de una conexión segura.

En mi caso utilice certificados gratuitos emitidos por let’s encrypt para certificar sitios realizados en wordpress

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

Aunque HTTP se diseñó para la web, se hizo intencionalmente más general para adaptarse a la orientación a objetos. Por esta razón se definieron operaciones de distintos tipos que se pueden realizar sobre recursos web, por ejemplo, creación, eliminación, obtención, etc.
Estas operaciones son representadas por métodos HTTP que indican el tipo de operación a realizar.
Algunos de los métodos más utilizados son:

1. GET
2. POST
3. PUT
4. DELETE

Aunque existen otros como TRACE, CONNECT y OPTIONS.

Explica las diferencias entre los métodos HTTP GET y POST

Las diferencias entre los métodos HTTP GET y POST son:

- GET sirve para obtener un recurso web como una página, mientras que POST sirve para enviar datos al servidor como un formulario
- GET puede enviar datos en la solicitud y esos datos se cargan en la ruta de la solicitud en forma de query params (no es recomendable si se envían datos sensibles porque quedan visibles en la ruta). POST envía datos en la solicitud pero estos se cargan en el cuerpo de la solicitud, lo cual es recomendado para el envío de datos sensibles. Aclaración: que se utilice POST no garantiza la seguridad en la comunicación, como se aclaró en los puntos anteriores, esto se realiza utilizando HTTPS
- GET es idempotente, es decir cada vez que hago la misma solicitud GET, espero obtener el mismo resultado. En cambio POST no es idempotente por que el envío de múltiples POST no generarían el mismo resultado.
- Existen otras diferencias entre estos métodos como que las solicitudes de GET son cacheables y el límite del tamaño de los datos es distinto, pero las tres primeras son las más importantes.

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

El código de estado HTTP es un número que viene incluido en la primera línea de una respuesta HTTP e indica si la petición fue atendida o si no lo fue y su razón. Los códigos de estado se clasifican en 5 grupos: 1xx, 2xx, 3xx, 4xx, 5xx.
Los más comunes son los del grupo de 2xx, 4xx y 5xx, por ejemplo:

- 200 indica que la solicitud fue exitosa
- 201 indica que la solicitud fue exitosa y creó un recurso.
- 401 indica que el cliente no está autenticado para el recurso que solicitó.

- 403 indica que el cliente no tiene los permisos para el recurso que solicitó, es similar al 401, pero no se soluciona con una autenticación. Por ejemplo un usuario que está autenticado pero que no tiene permisos de administrador
- 404 indica que el recurso no fue encontrado
- 500 indica que ocurrió un error interno al servidor por el cual no se pudo cumplir la solicitud

¿Que es una cabecera HTTP? Da ejemplos de cabeceras comunes

Una cabecera HTTP es una línea adicional de información que se puede añadir a una solicitud o respuesta, por lo que existen cabeceras de solicitud y cabeceras de respuesta. Las cabeceras se podrían comparar con los parámetros que se pasan a una función para que se configure internamente y funcione de manera correcta.
Si bien la lista de cabeceras es extensa, considero que algunas de las más importantes son:

- Authorization: el cliente puede listar sus credenciales con esta cabecera.
- Set-Cookie: el servidor indica la cookie que debe guardar el cliente
- Cookie: Es un fragmento de datos que el servidor envía al cliente y pueda almacenar información del mismo. La cookie es enviada al servidor en cada petición siguiente.
- Content-Type: indica el formato en el que se enviará el cuerpo de la petición o respuesta. Por ejemplo application/json para cuerpos en formato json.
- Accept: indica los formatos de contenido aceptados como por ejemplo el contenido multimedia

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

El concepto de idempotencia en los métodos HTTP se refiere a que hay métodos que al ejecutarlos múltiples veces, siempre generan el mismo resultado.
Los métodos que cumplen esta característica son:

- GET ya que la obtención de un recurso no debería alterar el estado del servidor
- PUT ya que al aplicar la misma modificación a un recurso generaría el mismo resultado.
- DELETE ya que al eliminar varias veces el mismo recurso genera el mismo resultado, el recurso fue eliminado la primera vez.
- OPTIONS ya que solo obtiene información sobre el servidor
- HEAD ya que obtiene información de las cabeceras

¿Que es un redirect (redirección) HTTP y cuando es utilizado?

Una redirección HTTP es una respuesta que envía un servidor al cliente para indicarle que el recurso que solicita está en otra ubicación. Se utiliza en varios casos como cuando hay cambio permanente o temporal de una URL, forzar a la utilización de la versión segura HTTPS o para indicar que debe autenticarse.
La redirección se realiza indicando la ubicación del recurso en la cabecera 'Location' y con algún código de estado del rango 3xx.
Al recibir este tipo de respuesta un cliente, por ejemplo un navegador, realizará una nueva petición a la url indicada en Location.

Preguntas técnicas y de seguridad en HTTP/HTTPS

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

La integridad de los datos en una conexión HTTPS se asegura gracias a SSL el cual realiza los siguientes pasos en la transmisión de los mensajes:

- Se fragmentan los mensajes a transmitir, se comprimen y se les concatena una clave.
- A la concatenación se le aplica una función hash obteniendo un código llamado MAC (Código de autenticación de mensajes)
- El fragmento comprimido y el MAC se encriptan con el algoritmo de encriptación simétrico acordado.
- Por último el fragmento es transmitido por la conexión.

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

La diferencia que hay entre un ataque “man-in-the-middle” y uno de “replay” es que el primero además de interceptar los mensajes transmitidos, los modifica o redirige. Por otro lado el segundo no los modifica, solo los reutiliza para suplantar al remitente original.

Explica el concepto de “handshake” en HTTPS

Lo explique con más detalle en la pregunta sobre el proceso de cifrado en HTTPS, pero para resumir podemos decir que handshake es el subproceso de establecer la conexión segura. Para lograrlo el cliente y el servidor se envían entre sí una serie de mensajes para determinar las claves públicas, el algoritmo de encriptación a utilizar y otros detalles.

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

HSTS es un estándar para asegurar que los navegadores siempre se conecten a un sitio utilizando el protocolo HTTPS.
El sitio puede indicar al cliente a través de la cabecera Strict-Transport-Security que habilitó HSTS. Al saber esto, el cliente enviará las peticiones utilizando HTTPS aún aunque el usuario escriba manualmente http en la url o clickee un enlace que dirija a una dirección http
Mejora la seguridad en una aplicación web porque se utilizará un protocolo seguro de comunicación y evita que el cliente acepte advertencias sobre certificados no válidos.

¿Que es un ataque “downgrade” y como HTTPS lo previene?

Un ataque downgrade es un tipo de ataque donde se logra que la conexión se realice utilizando algoritmos criptográficos débiles o versiones antiguas de protocolos con fallas de seguridad. Al lograrlo el atacante puede explotar las vulnerabilidades de los algoritmos o los protocolos.
HTTPS lo previene utilizando protocolos como HSTS y mecanismos de seguridad como SCSV el cual previene que se utilicen algoritmos o protocolos vulnerables.

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

CORS (Uso compartido de recurso entre orígenes) es un mecanismo para integrar aplicaciones que están en dominios distintos. En este mecanismo el cliente comprueba con el servidor si la solicitud a realizar está autorizada antes del envío de datos.
Se puede implementar en una aplicación web a través de los mecanismos del lenguaje o framework utilizado. Por ejemplo en el entorno de nodejs se puede implementar a través de la biblioteca “cors” y a través de anotaciones en el framework de java spring.
Se puede configurar cors para permitir ciertos orígenes, métodos o cabeceras. Estas configuraciones se envían en cabeceras de tipo “Access-Control”

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

La diferencia principal entre una cabecera Authorization y una cabecera Cookie es que la primera es utiliza para que un cliente se autentique con el servidor, por ejemplo enviando en ella el usuario y la contraseña o un token, y la segunda para almacenar información sobre el cliente como la sesión.
Existen otras diferencias como que la Authorization se debe indicar en cada solicitud y la cookie se envía automáticamente.

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

La cabecera Context-Security-Policy sirve para indicar que tipos de recursos se pueden cargar en un sitio (como scripts) y desde que dominios. Por otro lado X-Frame-Options sirve para indicar al navegador si puede representar la página en una etiqueta como <frame> o <embed>.
Ayudan a mitigar ataques comunes como XSS al no permitir que se inserten scripts o recursos maliciosos en la página y clickjacking al inhabilitar la inserción de las páginas en etiquetas como <embed> para engañar al usuario.

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

Las tres versiones de HTTP utilizan una unica conexion para todas las solicitudes, pero varían en cómo se procesan esas solicitudes y las cabeceras:

- HTTP/1.1 maneja las solicitudes de manera secuencial, por lo que es bloqueante. Si se descargan muchos recursos se realizaría una solicitud por recurso y sería muy lenta.
- HTTP/2 maneja múltiples solicitudes a la vez, por lo que no es bloqueante. Además prioriza las solicitudes y comprime las cabeceras.
- HTTP/3 a diferencia de las dos anteriores no usa TCP sino QUIC que está basado en UDP por lo que tiene mejor latencia y permite conexiones más rápidas. Además tiene cifrado nativo a diferencia de las versiones anteriores que se apoyan en TLS

¿Que es un “keep-alive” en HTTP y cómo mejora el rendimiento de las aplicaciones?

“keep-alive” en HTTP representa que la conexión establecida a través del protocolo TCP permanecerá abierta aun después de enviada la respuesta. Es útil en HTTP/1.0 porque en esta versión las conexiones no son persistentes por defecto. En HTTP/1.1 las conexiones son persistentes por defecto, pero aun así se pueden configurar para establecer el tiempo de duración de la conexión y el límite de solicitudes por conexión. En las versiones HTTP/2 y HTTP/3 no es útil ya que estas versiones manejan las conexiones de manera eficiente por defecto.
La forma de configurar “keep-alive” es configurando las cabeceras Keep-Alive y Connection de la siguiente manera

Connection: keep-alive
Keep-Alive: timeout=5, max=1000

Indicamos que la conexión permanecerá abierta con la cabecera Connection, configuramos la conexión para que permanezca abierta durante 5 segundos de inactividad y permitimos un máximo de 1000 solicitudes en esa conexión.

Preguntas de implementacion practica

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

Depende del caso de uso a resolver. Propongo casos de uso específicos donde aplicaría cada tipo de autenticación en una API:

- Si el caso de uso es autenticar usuarios internos dentro de una organización bajo un entorno controlado usaría la autenticación Basic con usuario y contraseña.
- Si el caso de uso es autenticar usuarios externos, que no requiere renovación frecuente de tokens ni acceso a recursos protegidos del usuario en otro servicio usaría JWT
- Si el caso de uso es autenticar usuarios externos y acceder a información del usuario protegida por otro servicio, usaría OAuth, porque me permite el control granular de permisos y accesos específicos sin que el usuario brinde su usuario y contraseña

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

Un proxy inverso es un servidor que se coloca delante de un grupo de servidores para actuar de intermediario entre los clientes y los servidores. La idea del proxy inverso es que los clientes se comuniquen con él y no directamente con los servidores detrás de él.
Un proxy inverso se puede utilizar para múltiples tareas como balanceo de carga, protección contra ataques, brindar respuestas más rápidas a los clientes utilizando caché en el proxy inverso o delegación de tareas como encriptar/desencriptar comunicaciones.

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

Usaría un servidor que actúa de proxy inverso como Nginx y lo configuraria para que dirija todas las peticiones HTTP (puerto 80) a HTTPS (puerto 443) utilizando el código de estado

301 (movido permanentemente) para indicar a los clientes que las futuras solicitudes se realicen utilizando HTTPS. Además configuraría el certificado SSL para recibir las peticiones HTTPS

¿Como mitigarias un ataque de denegación de servicio (DDoS) en un servidor HTTP?

Utilizaria un CDN como Cloudflare que es una red de servidores distribuidos geográficamente que brindan el contenido de manera eficiente. Al mismo le configuraría características de seguridad como firewalls, rate limit y protecciones contra ataques DDoS para que monitorice el tráfico.

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

Podría enfrentar problemas de rendimiento como los siguientes
Sobrecarga en el servidor, para solucionarlo haría un escalamiento horizontal agregando más servidores y anteponiendo un balanceador de carga para que gestione el tráfico.
Sobrecarga del servidor de base de datos, lo solucionaria utilizando memoria caché como Redis, optimizando consultas y utilizando sharding para particionar la base de datos en porciones más pequeñas llamadas shards.
Además utilizaria una cola de mensajes para delegar las tareas de procesamiento exhaustivo a un grupo de workers.
También podría enfrentar problemas de seguridad de distinto tipo. Para mitigarlos utilizaria HTTPS, mecanismos de autenticación como OAuth2 o JWT y CDNs como cloudflare para prevenir ataques DDoS.

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

Un cliente HTTP es un software como un navegador o aplicación que realiza peticiones a un servidor utilizando el protocolo HTTP.
La diferencia es que POSTMAN es un cliente HTTP con interfaz GUI que permite guardar colecciones de solicitudes, generar documentación e integrarlo con herramientas de CI/CD.
A diferencia de CURL que es un cliente HTTP que se ejecuta sobre una terminal y permite utilizarlo en scripts con mayor flexibilidad, aunque no posee las características mencionadas de POSTMAN.

Preguntas de GIT

¿Que es GIT y para que se utiliza en desarrollo de software?

Git es un sistema de control de versiones de software que se utiliza para llevar un registro de los cambios que se fueron realizando en el tiempo en los distintos archivos que componen la implementación del software.

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

La diferencia es que el repositorio local se encuentra ubicado en la computadora del desarrollador y el repositorio remoto se encuentra ubicado en un servicio de alojamiento de repositorios GIT como GitHub o GitLab

¿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

Un repositorio local de git se crea a través de una terminal de línea de comandos con el comando “git init”. También se puede crear utilizando alguna aplicación gráfica como Git Kraken pero por detras realiza el mismo comando
La diferencia entre“git commit” y “git push” es que el primero confirma los cambios realizados en la base de datos del repositorio local y el segundo envía los cambios confirmados desde el repositorio local al remoto.

¿Que es un “branch” en GIT y para que se utilizan las ramas en el desarrollo de software?

Un branch en GIT es una bifurcación del proyecto en un punto específico del tiempo que crea una línea de desarrollo independiente. Se utiliza en el desarrollo de software para distintas tareas como desarrollo de funcionalidades aún no integradas, solución de errores o pruebas que no afecten la versión estable del proyecto.

¿Que significa hacer un “merge” en GIT y cuales son los posibles conflictos que pueden surgir durante un merge?

Hacer un merge en GIT significa fusionar dos ramas utilizando una de ellas como destino, esto implica la integración de los cambios realizados en ambas ramas. Los posibles conflictos que pueden surgir en un merge son que en ambas ramas existan los mismos archivos cuyo contenido sea distinto en una misma línea. Esto se debe solucionar indicando cual de estos contenidos distintos se va a conservar y cual se va a descartar.

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

Branching model describe la estrategia de ramas que se utilizará en un proyecto. Según el modelo elegido se crearan, nombraran y fusionaran ramas siguiendo las reglas del modelo.
Git Flow utiliza las ramas master, develop, feature, release, hotfix, bugfix y support. Como indican sus nombres se utilizan para tareas específicas como arreglar un bug de último momento, una funcionalidad nueva, etc.
GitHub Flow es más simple ya que se crean ramas a partir de main para crear nuevas funcionalidades o arreglar bugs, se fusionan con main y se eliminan después de fusionarlas.

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

Se puede deshacer un cambio después de hacer el commit y antes de hacer push utilizando git reset. Hay tres tipos de reset:

- soft: deshace el commit pero mantiene los cambios en el area de preparacion

- mixed: deshace el commit y saca los cambios del área de preparación, pero los mantiene el directorio de trabajo
- hard: deshace el commit y elimina los cambios del área de preparación y del directorio de trabajo.

¿Que es un “pull request” y como contribuye a la revisión de código en un equipo?

Un pull request es una solicitud para fusionar cambios de una rama a otra en un repositorio. Esta solicitud puede venir del mismo proyecto o de un repositorio bifurcado (fork). Contribuye a la revisión de código porque permite al equipo revisar, comentar y sugerir cambios antes de integrar los cambios propuestos.

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

Un repositorio de GIT se puede clonar con el comando git clone, este comando crea una copia nueva local del repositorio remoto. En cambio git pull trae los cambios que hay en el repositorio remoto y actualiza el repositorio local

Preguntas de Node

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

NodeJS es un entorno de ejecución para javascript. Esto significa que permite que el código javascript se pueda ejecutar fuera del navegador, permitiendo la creación de aplicaciones de servidor como por ejemplo APIs.
Es una opción popular para el desarrollo backend por características como la utilización de un mismo lenguaje para backend y frontend. Además es asíncrono y basado en eventos lo que permite manejar múltiples conexiones de forma eficiente, lo que es ideal para aplicaciones en tiempo real.

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

El modelo de I/O no bloqueante en NodeJS funciona de la siguiente manera: cuando se solicita una operación de I/O se registra una función callback asociada a la operación. A continuación se continúa con la ejecución del código y cuando finaliza la operación I/O se ejecuta el callback.
Esta forma de funcionamiento beneficia el rendimiento porque permite manejar múltiples operaciones de manera no bloqueante.

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

En javascript las operaciones a ejecutar se colocan en una pila de llamadas. Cuando el intérprete lee una operación síncrona la coloca directamente en esta pila, pero cuando la operación es asíncrona la coloca en una cola de tareas.

El Event Loop es un ciclo que cada vez que se completa observa la pila de llamadas y la cola de tareas, si detecta que la pila está vacía y la cola no está vacía, saca la primera operación de la cola y la coloca en la pila para su ejecución.
Este proceso se repite hasta que no haya operaciones que realizar.

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

Ambas son instrucciones para la inclusión de un módulo. La diferencia es que require() se utiliza en módulos commonjs (tradicional en nodejs) y se ejecuta de manera síncrona. En cambio import se utiliza en módulos siguiendo la especificación de ecmascript, es asíncrono lo que permite importaciones estáticas y dinámicas y para su utilización se debe indicar "type: module" en el package.json o la extensión del archivo debe ser mjs.

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

npm (node package manager) es el gestor de bibliotecas por defecto de NodeJS que permite la instalación, actualización y eliminación de bibliotecas de terceros en proyectos propios. También permite la publicación de bibliotecas propias en el registro de npm para compartir código con otros desarrolladores.

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

Un proyecto de NodeJS se inicia ejecutando el comando init del gestor npm de la siguiente manera: "npm init".
Este comando crea un archivo nombrado package.json que contiene las configuraciones del proyecto de NodeJS.
Algunas de las configuraciones más importantes son los scripts, las dependencias de desarrollo y las dependencias de producción entre otras.

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

Las dependencias en npm son bibliotecas que se pueden incluir en nuestro proyecto de NodeJS. Se pueden instalar utilizando el comando "npm install" seguido del nombre de la dependencia para una dependencia en específico. También se pueden indicar en el package.json y ejecutar el comando "npm install" que instalara todas las dependencias incluidas en este archivo.
La diferencia entre dependencias y dependencias de desarrollo es que el código de las primeras está incluido en el empaquetado que se distribuye para el correcto funcionamiento en un entorno productivo. En cambio en las segundas su código no es incluido en el empaquetado que se distribuye porque solo es necesario para el funcionamiento en un entorno de desarrollo.

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

Las versiones específicas de los paquetes se pueden gestionar de dos maneras:

- Indicando la versión del paquete a instalar cuando se ejecuta el comando “npm install”, por ejemplo: “npm install express@4.21.1”
- Indicando en el package.json la versión de la dependencia

Además hay dos operaciones, caret (^) y tilde(~), que se utilizan para hacer más flexible la versión de dependencia a instalar.
El archivo package-lock.json se crea automáticamente al ejecutar “npm install” y se utiliza para fijar las dependencias y subdependencias para que los distintos entornos que instalen el proyecto utilicen las mismas versiones.

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

NestJS es un framework de NodeJS para construir aplicaciones backend. Está escrito en typescript, es modular, escalable, posee un sistema de inyección de dependencias, enrutadores, controladores, está basado en clases y utiliza el servidor express por defecto.
Para construir un proyecto NestJS primero se debe instalar su CLI de forma global utilizando npm con el comando “npm i -g @nestjs/cli”. Para crear un proyecto se ejecuta el comando “nest new nombre-proyecto” el cual creará toda la estructura.
Se utilizan distintos componentes de NestJS como controladores manejan las peticiones del cliente, los servicios que tienen la lógica de negocio y los módulos que engloban a estos dos en una unidad cohesiva

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

Los errores en NodeJS se pueden manejar encerrando el código que puede lanzar un error en una estructura try catch, pasando un callback de manejo de error si el método que puede lanzar el error lo permite o con promesas.
Al momento de manejar código asincrónico se pueden utilizar dos enfoques: callbacks y promesas. Los callbacks son funciones que se pasan como parámetros a funciones asíncronas y son ejecutados internamente por esta. Ya que una función asíncrona puede finalizar exitosamente o con un error, comúnmente se pasan dos callback, uno para manejar el caso exitoso y otro para manejar el error.
Por otro lado las promesas son objetos que devuelven algunas operaciones asíncronas para representar que devolverán un valor al finalizar. A diferencia de las operaciones asíncronas que reciben callbacks, las promesas pueden ser procesadas de manera más elegante utilizando métodos de su api como then() para procesar una respuesta exitosa o catch() para procesar un error.
Hay ocasiones donde necesitamos obtener primero el resultado de una promesa porque la operación siguiente depende de este resultado, en estos casos se utiliza la palabra reservada “await” delante de la ejecución de la operación que devuelve la promesa y además se coloca la palabra reservada “async” a la función que contiene el await para convertirla en una función asíncrona. Al realizar esto se pausa la ejecución de la función en la línea que contiene await pero no bloquea el hilo principal, por lo que el código externo a la función asíncrona continua ejecutando.

Práctica

Actividad Practica Numero 1

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

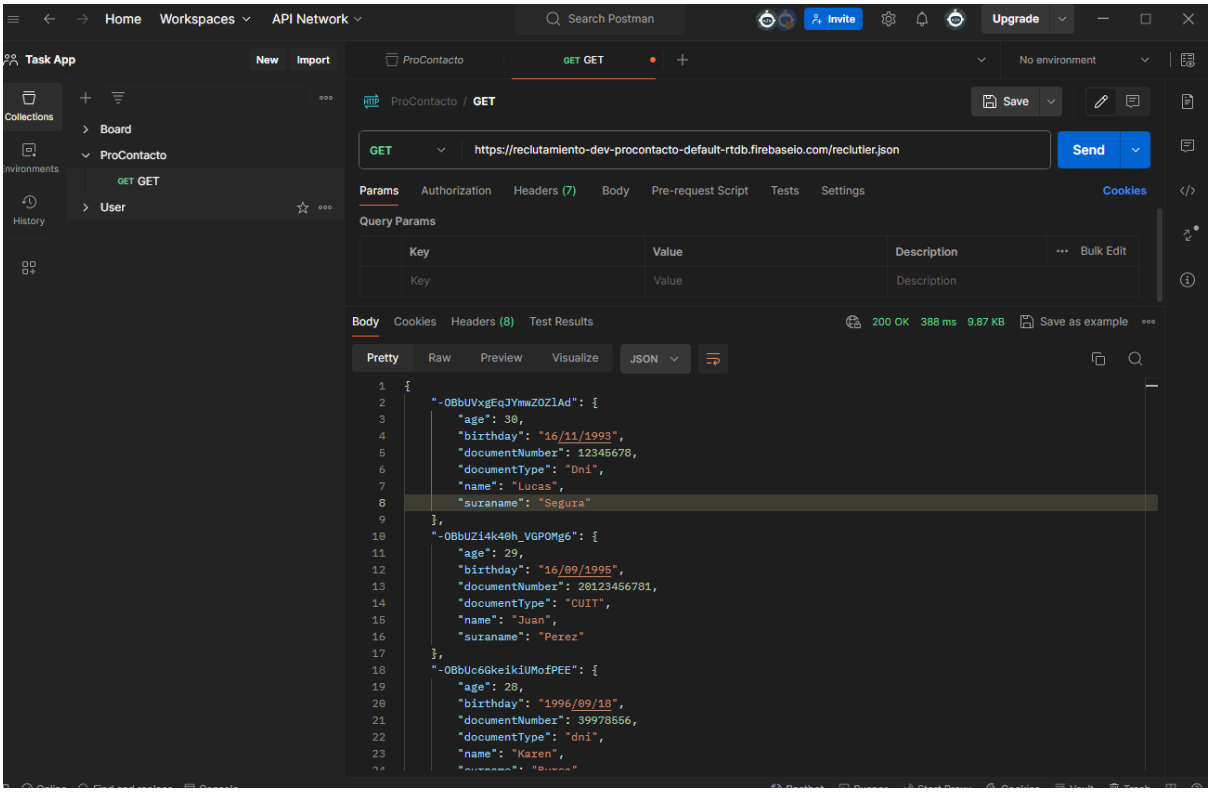
2) Realizar una peticion POST a la siguiente URL a traves de POSTMAN
https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json
con el siguiente body:

```
{
        "name":"TuNombre",
        "suraname":"TuApellido",
        "birthday":"1995/11/16/",
        "age":29,
        "documentType":"CUIT",
        "documentNumber":20123456781
}
```

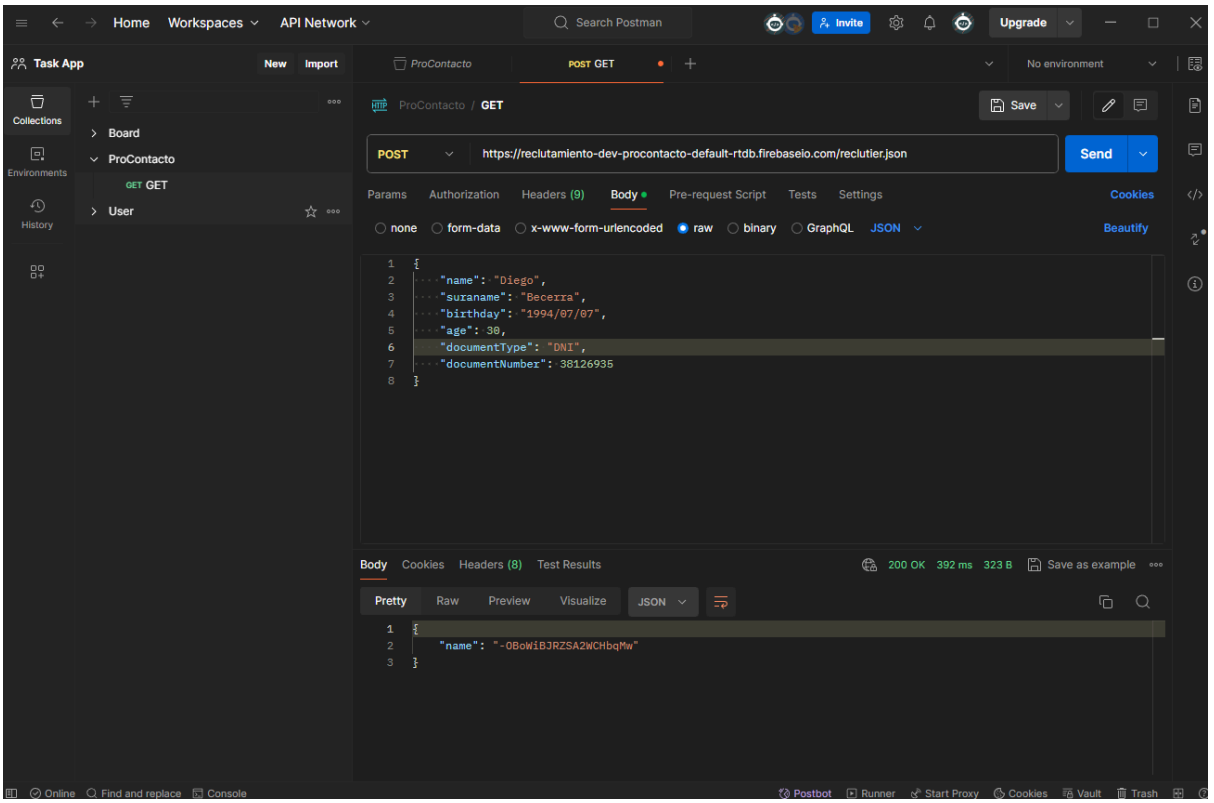
Preguntas

1) Adjuntar imágenes del response de un GET y de un post de cada punto

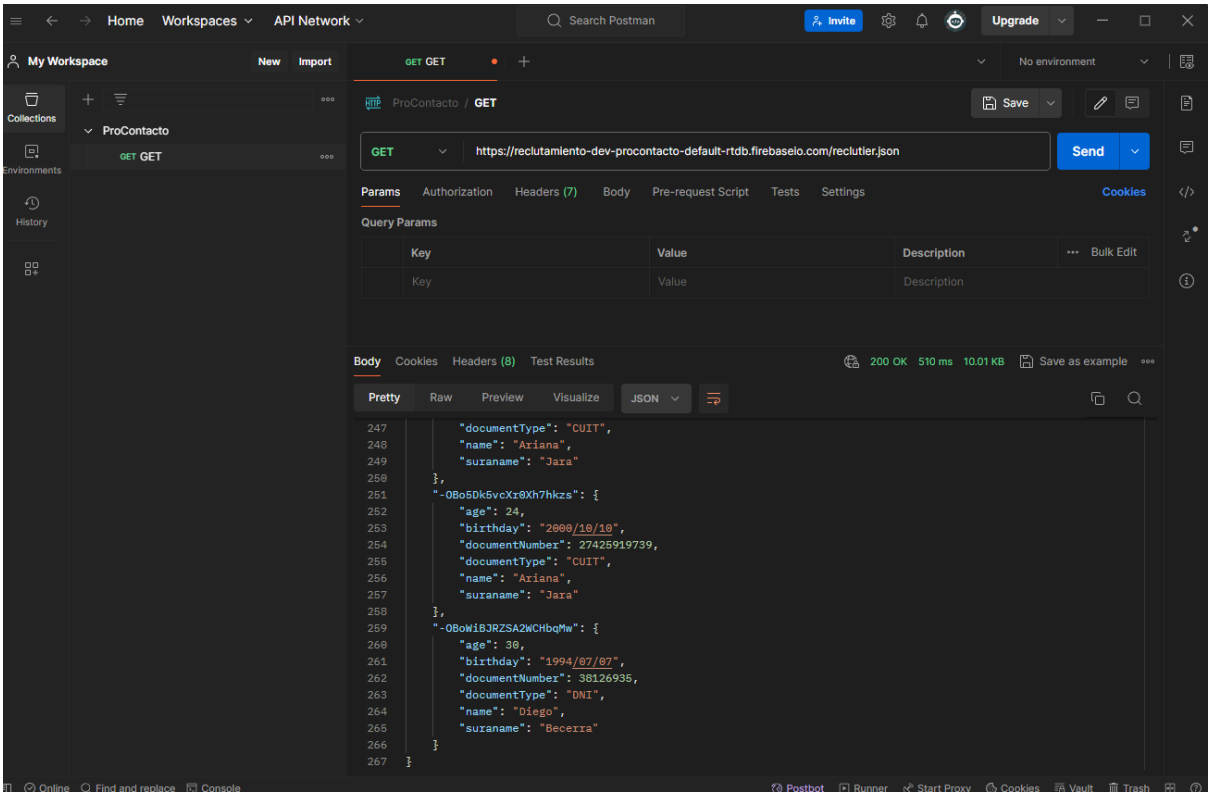
Captura del GET antes de hacer el POST



Captura del POST



Captura al volver a hacer GET, se puede observar que quedaron registrados mis datos



2) ¿Que sucede cuando hacemos el GET por segunda vez, luego de haber ejecutado el POST?

Al hacer GET por segunda vez, podemos observar que se guardaron los datos personales que envíe por POST.

Actividad Practica Numero 2 y 3

Link al repositorio:
https://github.com/becerrawebd/practica_procontacto