Antonio Prieto
pryet2@gmail.com

Tarea 5 DWES

24 de febrero del 2022

1. Servicios web

a) ¿Qué es un servicio web (diagrama)? Lee, investiga, mira todo este vídeo y explícalo con tus palabras.

Un servicio web es una funcionalidad que puede consumir cualquier tipo de cliente que pueda comunicarse con el servidor donde está alojado generalmente usando Json

b) Indica qué ventajas e inconvenientes presentan los servicios web y para qué se usan principalmente.

Ventajas:

Aportan flexibilidad a la hora de consumir estos servicios ya que no importa en que lenguaje estan hechos o cómo están implementados, solo importa que podamos comunicarnos con él, normalmente usando Json

Modularidad, permitiendo dividir nuestra aplicación en diferentes servicios web que podamos reutilizarlos en otras aplicaciones posteriores como por ejemplo un servicio de login único para diferentes aplicaciones como hace google, facebook, microsoft amazon ....

Inconvenientes: rendimiento algo menor

c) Enumera algunos de todos los protocolos y especificaciones existentes en los servicios web.

BEEP - Blocks Extensible Exchange Protocol

CTS - Canonical Text Services Protocol

E-Business XML

Hessian

Internet Open Trading Protocol

JSON-RPC

JSON-WSP

SOAP - outgrowth of XML-RPC, originally an acronym for Simple Object Access Protocol

d) Describe la arquitectura (diagrama) de un servicio web y qué tres roles y tres operaciones intervienen.

Proveedor : el proveedor crea el servicio web y lo pone a disposición de la aplicación cliente que desea utilizarlo.

Solicitante : un solicitante no es más que la aplicación cliente que necesita ponerse en contacto con un servicio web. La aplicación cliente puede ser .Net, Java o cualquier otra aplicación basada en lenguaje que busque algún tipo de funcionalidad a través de un servicio web.

Intermediario : el intermediario no es más que la aplicación que proporciona acceso a la UDDI. El UDDI, como se discutió en el tema anterior, permite que la aplicación cliente localice el servicio web.

e) Indica los dos tipos de servicios web más populares en la actualidad.

SOAP:

SOAP se conoce como un protocolo de mensajería independiente del transporte. SOAP se basa en la transferencia de datos XML como mensajes SOAP. Cada mensaje tiene algo que se conoce como un documento XML. Solo la estructura del documento XML sigue un patrón específico, pero no el contenido. La mejor parte de los servicios web y SOAP es que todo se envía a través de HTTP, que es el protocolo web estándar.

REST:

Restful Web Services es un servicio ligero, fácil de mantener y escalable que se basa en la arquitectura REST. Restful Web Service, exponga la API de su aplicación de manera segura, uniforme y sin estado al cliente que llama. El cliente que llama puede realizar operaciones predefinidas utilizando el servicio Restful. El protocolo subyacente para REST es HTTP. REST significa Transferencia de estado representacional.

2. SOAP

a) Lee, investiga y explica con tus propias palabras qué es el protocolo de mensajes SOAP (diagrama).

Es un protocolo basado en XML que consta de tres partes:

- Un sobre, que define la estructura del mensaje y cómo procesarlo
- Un conjunto de reglas de codificación para expresar instancias de tipos de datos definidos por la aplicación
- Una convención para representar llamadas y respuestas a procedimientos

Básicamente SOAP es un protocolo que define como tiene que ir el mensaje en xml

b) Indica qué ventajas e inconvenientes presenta SOAP y cuál es la estructura de un mensaje SOAP.

Ventajas:

- La característica de neutralidad de SOAP lo hace explícitamente adecuado para su uso con cualquier protocolo de transporte. Las implementaciones a menudo usan HTTP
- SOAP, cuando se combina con los intercambios de publicación/respuesta HTTP, se canaliza fácilmente a través de servidores de seguridad y servidores proxy existentes y, en consecuencia, no requiere modificar las infraestructuras informáticas y de comunicación generalizadas que existen para procesar los intercambios de publicación/respuesta HTTP.
- SOAP tiene a su disposición todas las facilidades de XML, incluyendo fácil internacionalización y extensibilidad con XML Namespaces.

Inconvenientes:

- Cuando se utiliza la implementación estándar y el enlace SOAP/HTTP predeterminado, el conjunto de información XML se serializa como XML. Para mejorar el rendimiento en el caso especial de XML con objetos binarios incrustados, se introdujo el mecanismo de optimización de transmisión de mensajes .
- Cuando se confía en HTTP como protocolo de transporte y no se utiliza el direccionamiento de servicios web o un bus de servicios empresariales , las funciones de las partes que interactúan son fijas. Solo una de las partes (el cliente) puede utilizar los servicios de la otra.
- SOAP es menos "simple" de lo que sugiere el nombre. La verbosidad del protocolo, la lenta velocidad de análisis de XML y la falta de un modelo de interacción estandarizado

llevaron al dominio de los servicios que utilizan el protocolo HTTP de manera más directa. Véase, por ejemplo, REST .

- Al ser independiente del protocolo, SOAP no puede aprovechar las funciones y optimizaciones específicas del protocolo, como la interfaz uniforme de REST o el almacenamiento en caché , sino que tiene que volver a implementarlas (como con WS-Addressing).

Estructura del mensaje SOAP:

Sobre: Identifica el documento XML como un mensaje SOAP.

Encabezado: Contiene información de encabezado.

Cuerpo: Contiene información de llamadas y respuestas.

Fallo: Proporciona información sobre los errores que ocurrieron al procesar el mensaje.

c) Muestra un ejemplo de petición SOAP así como una respuesta del servidor a dicha petición.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

d) Explica qué es WSDL y UDDI y explica detalladamente el funcionamiento de un servicio web SOAP

WSDL:

es un lenguaje de descripción de interfaz basado en XML que se utiliza para describir la funcionalidad que ofrece un servicio web

UDDI:

es un protocolo de lenguaje de marcado extensible independiente de la plataforma que incluye un registro (basado en XML) mediante el cual las empresas de todo el mundo pueden incluirse en Internet y un mecanismo para registrar y localizar aplicaciones de servicios web

Funcionamiento:

El Service Provider genera el WSDL describiendo el Web Service y registra el WSDL en el directorio UDDI o Service Registry.

El Service Requestor o la aplicación del cliente requiere un Web Service y se pone en contacto con el UDDI para localizar el Web Service.

El cliente, basándose en la descripción descrita por el WSDL, envía un request para un servicio particular al Web Service Listener, que se encarga de recibir y enviar los mensajes en formato SOAP.

El Web Service analiza el mensaje SOAP del request e invoca una operación particular en la aplicación para procesar el request. El resultado se escribe de nuevo en SOAP en forma de respuesta y se envía al cliente.

El cliente analiza el mensaje de respuesta SOAP y lo interpreta o genera un error si ha habido alguno.

3. REST y RESTful

a) Lee, investiga y explica el estilo de arquitectura de software para sistemas hipermedia REST.

- Rendimiento en las interacciones de los componentes, que puede ser el factor dominante en el rendimiento percibido por el usuario y la eficiencia de la red; [7]
- Escalabilidad que permite soportar un gran número de componentes e interacciones entre componentes;
- Simplicidad de una interfaz uniforme;
- Modificabilidad de los componentes para satisfacer las necesidades cambiantes (incluso mientras se ejecuta la aplicación);
- Visibilidad de la comunicación entre los componentes por parte de los agentes de servicio;
- Portabilidad de los componentes moviendo el código del programa con los datos;
- Confiabilidad en la resistencia a la falla a nivel del sistema en presencia de fallas dentro de los componentes, conectores o datos.

b) Esclarece la diferencia entre REST y RESTful.

La transferencia de estado representacional (REST) es un estilo de arquitectura de software

RESTful se usa típicamente para referirse a los servicios web que implementan dicha arquitectura.

c) Explica las seis restricciones de arquitectura REST que tiene que cumplir un servicio web RESTful.

Restricciones REST:

- **Arquitectura cliente-servidor**

  El patrón de diseño cliente-servidor hace cumplir el principio de **separación de responsabilidades:** separar las responsabilidades de la interfaz de usuario de las responsabilidades de almacenamiento de datos. De este modo se mejora la **portabilidad de la interfaz de usuario**. En el caso de la Web, se ha desarrollado una plétora de navegadores web para la mayoría de las plataformas **sin necesidad de conocer ninguna implementación de servidor.** La separación también simplifica los componentes del servidor, mejorando la escalabilidad, pero lo que es más importante, permite que los componentes evolucionen de forma independiente (escalabilidad anárquica), lo cual es necesario en un entorno a escala de Internet que involucra múltiples dominios organizacionales.

- **Sin estado**

  En informática, un protocolo sin estado es un protocolo de comunicaciones en el que el receptor, generalmente un servidor, no retiene información de la sesión. El cliente envía los datos de sesión relevantes al receptor de tal manera que cada paquete de información transferido se puede entender de forma aislada, sin información de contexto de los paquetes anteriores en la sesión. Esta propiedad de los protocolos sin estado los hace ideales en aplicaciones de alto volumen, ya que **aumenta el rendimiento al eliminar** la carga del servidor causada por **la retención de la información de la sesión.**

- **Caché**

  Al igual que en la World Wide Web, **los clientes e intermediarios pueden almacenar en caché las respuestas**. Las respuestas deben, implícita o explícitamente, definirse a sí mismas como almacenables en caché o no almacenables en caché para evitar que los clientes proporcionen datos obsoletos o inapropiados en respuesta a solicitudes posteriores. **El almacenamiento en caché bien administrado elimina parcial o**

completamente algunas interacciones cliente-servidor, lo que mejora aún más la escalabilidad y el rendimiento.

- **Sistema de capas**

  Normalmente, un cliente no puede saber si está conectado directamente al **servidor final o a un intermediario** en el camino. Si se coloca un proxy o un balanceador de carga entre el cliente y el servidor, no afectará sus comunicaciones y no será necesario actualizar el código del cliente o del servidor. Los **servidores intermediarios pueden mejorar la escalabilidad** del sistema al permitir el equilibrio de carga y al proporcionar cachés compartidos. Además, **la seguridad se puede agregar como una capa sobre los servicios web**, separando la lógica comercial de la lógica de seguridad. [9] Agregar seguridad como una capa separada hace cumplir las políticas de seguridad . Finalmente, los servidores intermediarios pueden llamar a muchos otros servidores para generar una respuesta al cliente.

- **Código bajo demanda (opcional)**

  Los servidores pueden **ampliar o personalizar temporalmente la funcionalidad de un cliente** mediante la transferencia de código ejecutable: por ejemplo, componentes compilados como **applets de Java** o secuencias de comandos del lado del cliente como **JavaScript** .

- **Interfaz uniforme**

  La restricción de interfaz uniforme es fundamental para el diseño de cualquier sistema RESTful. **Simplifica y desacopla la arquitectura, lo que permite que cada parte evolucione de forma independiente**. Las cuatro restricciones para esta interfaz uniforme son:

  Identificación de recursos en solicitudes: los recursos individuales se identifican en solicitudes, por ejemplo, utilizando **URI** en servicios web RESTful. Los recursos en sí mismos están conceptualmente separados de las representaciones que se devuelven al cliente. Por ejemplo, el servidor podría enviar datos desde su base de datos como HTML , XML o JSON , ninguno de los cuales es una representación interna del servidor.

  Manipulación de recursos a través de representaciones: cuando un cliente tiene una representación de un recurso, incluidos los metadatos adjuntos, tiene suficiente información para modificar o eliminar el estado del recurso.

Mensajes autodescriptivos: cada mensaje incluye suficiente información para describir cómo procesar el mensaje. Por ejemplo, qué analizador invocar se puede especificar mediante un tipo de medio .

Hipermedia como el motor del estado de la aplicación (HATEOAS) - Habiendo accedido a un URI inicial para la aplicación REST (análogo a un usuario web humano que accede a la página de inicio de un sitio web), **un cliente REST debería poder usar enlaces proporcionados por el servidor dinámicamente para descubrir todos los recursos disponibles que necesita.** A medida que avanza el acceso, el servidor responde con un texto que incluye hipervínculos a otros recursos que están disponibles actualmente. No es necesario que el cliente esté codificado con información sobre la estructura o la dinámica de la aplicación.

d) Señala las diferencias entre REST y SOAP, cuándo es mejor uno que otro y qué retos afrontan sus API.

SOAP significa Protocolo simple de acceso a objetos, mientras que REST significa Transferencia de estado representacional.

SOAP es un protocolo mientras que REST es un patrón arquitectónico.

SOAP usa interfaces de servicio para exponer su funcionalidad a las aplicaciones cliente, mientras que REST usa localizadores de servicios uniformes para acceder a los componentes en el dispositivo de hardware.

SOAP necesita más ancho de banda para su uso, mientras que REST no necesita mucho ancho de banda.

Comparando SOAP vs REST API, SOAP solo funciona con formatos XML, mientras que REST funciona con texto sin formato, XML, HTML y JSON.

SOAP no puede hacer uso de REST mientras que REST puede hacer uso de SOAP.

4. HTTP RESTful API

a) Define qué es una RESTful API.

Las API de servicios web que se adhieren a las restricciones arquitectónicas REST se denominan API RESTful. Las API RESTful basadas en HTTP se definen con los siguientes aspectos:

- Un URI base , como http://api.example.com/;
- Métodos HTTP estándar (p. ej., GET, POST, PUT y DELETE);
- Un tipo de medio que define elementos de datos de transición de estado (p. ej., Atom, microformatos, application/vnd.collection+json, [13] : 91–99, etc.). La representación actual le dice al cliente cómo redactar solicitudes de transiciones a todos los siguientes estados de aplicación disponibles. Esto podría ser tan simple como un URI o tan complejo como un applet de Java. [14]

b) Estudia detenidamente e indica los tres aspectos que definen una RESTful API basada en HTTP.

Explicado en el punto anterior

c) Define qué es una URI y detalla cómo se usan para nombrar recursos en una HTTP RESTful API.

Un recurso puede ser un singleton o una colección .

Por ejemplo, “ customers” es un recurso de cobro y “ customer” es un recurso singleton (en un dominio bancario).

Podemos identificar customers el recurso de colección “ ” usando el URI “ **/customers**“. Podemos identificar un único customer recurso “ ” usando el URI “ **/customers/{customerId}**“.

Un recurso también puede contener recursos de subcolección .

Por ejemplo, el recurso de subcolección “ accounts” de un “ customer” en particular se puede identificar utilizando el URN “ **/customers/{customerId}/accounts**” (en un dominio bancario).

De manera similar, un recurso singleton “ account” dentro del recurso de subcolección “ accounts” se puede identificar de la siguiente manera: “ **/customers/{customerId}/accounts/{accountId}**“.

RESTful URI debe referirse a un recurso que es una cosa (sustantivo) en lugar de referirse a una acción (verbo)

http://api.example.com/user-management/users

http://api.example.com/user-management/users/{id}

Use "verbo" para denotar el arquetipo del controlador.

http://api.example.com/cart-management/users/{id}/cart/checkout
http://api.example.com/song-management/users/{id}/playlist/play

d) Define HTTP y explica el uso de los métodos HTTP GET, POST, PUT y DELETE en una HTTP RESTful API.

El Protocolo de transferencia de hipertexto (HTTP) es un protocolo de capa de aplicación en el modelo de conjunto de protocolos de Internet para sistemas de información hipermedia distribuidos y colaborativos . HTTP es la base de la comunicación de datos para la World Wide Web , donde los documentos de hipertexto incluyen hipervínculos a otros recursos a los que el usuario puede acceder fácilmente, por ejemplo, con un clic

- El método GET solicita que el recurso de destino transfiera una representación de su estado. Las solicitudes GET solo deben recuperar datos y no deben tener ningún otro efecto.
- El método POST solicita que el recurso de destino procese la representación incluida en la solicitud de acuerdo con la semántica del recurso de destino. Por ejemplo, se utiliza para publicar un mensaje en un foro de Internet
- El método PUT solicita que el recurso de destino cree o actualice su estado con el estado definido por la representación incluida en la solicitud
- El método DELETE solicita que el recurso de destino elimine su estado.

e) Define media type, enumera algunos e indica qué media type se usa en este y en este HTTP RESTful API.

Es un identificador de dos partes para formatos de archivo y contenidos de formato transmitidos en Internet (htm, html xml,xhtml,js,css,json...)

En el primer ejemplo se usa Json y en el segundo html

f) Define JSON, cuáles son sus tipos de datos, qué diferencias hay con XML y pon tres ejemplos de JSON.

Es un formato de archivo estándar abierto y un formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos de datos que consisten en pares atributo-valor y arreglos (u otros valores serializables).

- String
- numérico
- booleano
- nulo/vacío
- objeto
- array

JSON	XML
El objeto JSON tiene un tipo	Los datos XML no tienen tipo
Tipos JSON: cadena, número, matriz, booleano	Todos los datos XML deben ser cadenas
Los datos son fácilmente accesibles como objetos JSON	Los datos XML deben analizarse.
JSON es compatible con la mayoría de los navegadores.	El análisis XML entre navegadores puede ser complicado
JSON no tiene capacidades de visualización.	XML ofrece la capacidad de mostrar datos porque es un lenguaje de marcado.
JSON solo admite tipos de datos de texto y números.	XML admite varios tipos de datos, como números, texto, imágenes, tablas, gráficos, etc. También brinda opciones para transferir la estructura o el formato de los datos con datos reales.
Recuperar valor es fácil	Recuperar valor es difícil
Compatible con muchas herramientas de Ajax	No es totalmente compatible con el kit de herramientas de Ajax
Una forma totalmente automatizada de deserializar/serializar JavaScript.	Los desarrolladores tienen que escribir código JavaScript para serializar/deserializar desde XML
Soporte nativo para objeto.	El objeto tiene que expresarse mediante convenciones, principalmente el uso perdido de atributos y elementos.
Solo admite la codificación UTF-8.	Admite varias codificaciones.

No admite comentarios.	Admite comentarios.
Los archivos JSON son fáciles de leer en comparación con XML.	Los documentos XML son relativamente más difíciles de leer e interpretar.
No proporciona ningún soporte para espacios de nombres.	Admite espacios de nombres.
Es menos seguro.	Es más seguro que JSON.

Ejemplo Json:

```
{
        "visibility" : true,
        "popularity" : "immense",
        "id" : 210
}
```

g) Explica con tus propias palabras y detalladamente los ejemplos de esta tabla para el recurso Order.

http://example.com/api/orders (GET)

Devuelve la lista de pedidos

http://example.com/api/orders/123 (GET)

Devuelve el pedidor con identificador 123

http://example.com/api/orders (POST)

Crea un nuevo pedido con los datos que le envía el cliente

http://example.com/api/orders/123 (PUT)

Actualiza el pedido con identificador 123 con los datos que le envía el cliente

http://example.com/api/orders/123 (DELETE)

Elimina el pedido con identificador 123

h) Explica con tus propias palabras y detalladamente todas las combinaciones de URI y métodos HTTP para el recurso User.

	/users	/user/123
GET	Obtiene todo los usuarios	Obtiene el usuario con id=123
POST	Crea un usuario	URI no válida
PUT	Actualización de todos los usuarios	Actualiza el usuario con id=123
DELETE	Elimina todos los usuarios	Elimina el usuario con id=123

5. Uso de HTTP RESTful API en servicios web

a) Di en qué consiste el servicio web JSONPlaceholder, enumera sus recursos, y pon ejemplos de rutas

Genera datos en formato Json, sus recursos son:

/posts	100 posts
/comments	500 comments
/albums	100 albums
/photos	5000 photos
/todos	200 todos
/users	10 users

Rutas:

GET /posts

GET /posts/1

GET /posts/1/comments

GET	/comments?postId=1
POST	/posts
PUT	/posts/1
PATCH	/posts/1
DELETE	/posts/1

b) Explica qué es Fetch API y ejecuta los ejemplos que vienen en la guía del servicio web JSONPlaceholder.

La API Fetch proporciona una interfaz para recuperar recursos (incluso a través de la red)

```
> fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then((response) => response.json())
    .then((json) => console.log(json));
◂ ▸ Promise {<pending>}
                                                                    VM79:3
  ▸ {userId: 1, id: 1, title: 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit', body: 'quia et s
    uscipit\nsuscipit recusandae consequuntur …strum rerum est autem sunt rem eveniet architecto'}
>
```

```
> fetch('https://jsonplaceholder.typicode.com/posts')
    .then((response) => response.json())
    .then((json) => console.log(json));
◂ ▸ Promise {<pending>}
                                                                    VM83:3
   (100) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
   {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  ▸{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
   {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
   {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
>
```

```
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  body: JSON.stringify({
    title: 'foo',
    body: 'bar',
    userId: 1,
  }),
  headers: {
    'Content-type': 'application/json; charset=UTF-8',
  },
})
  .then((response) => response.json())
  .then((json) => console.log(json));
▸ Promise {<pending>}
▸ {title: 'foo', body: 'bar', userId: 1, id: 101}
```

```
> fetch('https://jsonplaceholder.typicode.com/posts/1', {
    method: 'PUT',
    body: JSON.stringify({
      id: 1,
      title: 'foo',
      body: 'bar',
      userId: 1,
    }),
    headers: {
      'Content-type': 'application/json; charset=UTF-8',
    },
  })
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
  ▶ {id: 1, title: 'foo', body: 'bar', userId: 1}
>
```

```
> fetch('https://jsonplaceholder.typicode.com/posts/1', {
    method: 'PATCH',
    body: JSON.stringify({
      title: 'foo',
    }),
    headers: {
      'Content-type': 'application/json; charset=UTF-8',
    },
  })
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
                                                                                    VM95:11
  ▶ {userId: 1, id: 1, title: 'foo', body: 'quia et suscipit\nsuscipit recusandae consequuntur …strum rerum est autem sunt
  rem eveniet architecto'}
>
```

```
> fetch('https://jsonplaceholder.typicode.com/posts/1', {
    method: 'DELETE',
  });
< ▶ Promise {<pending>}
>
```

```
> fetch('https://jsonplaceholder.typicode.com/posts?userId=1')
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
  ▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
```

```
> fetch('https://jsonplaceholder.typicode.com/posts/1/comments')
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
  ▶ (5) [{…}, {…}, {…}, {…}, {…}]
>
```

c) Usando Fetch API, haz una petición con método HTTP GET con URI /posts/10 a JSONPlaceholder.

```
> fetch('https://jsonplaceholder.typicode.com/posts/10')
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
                                                                    VM163:3
  ▶ {userId: 1, id: 10, title: 'optio molestias id quia eum', body: 'quo et expedita modi cum officia vel magni\ndolorib…i
    t\nquos veniam quod sed accusamus veritatis error'}
```

d) Usando Fetch API, obtén todos los comentarios de la publicación 8 de JSONPlaceholder.

```
> fetch('https://jsonplaceholder.typicode.com/post/8/comments')
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
  ▶ (5) [{…}, {…}, {…}, {…}, {…}]
>
```

e) Usando Fetch API, obtén todas las publicaciones del usuario 1 de JSONPlaceholder.

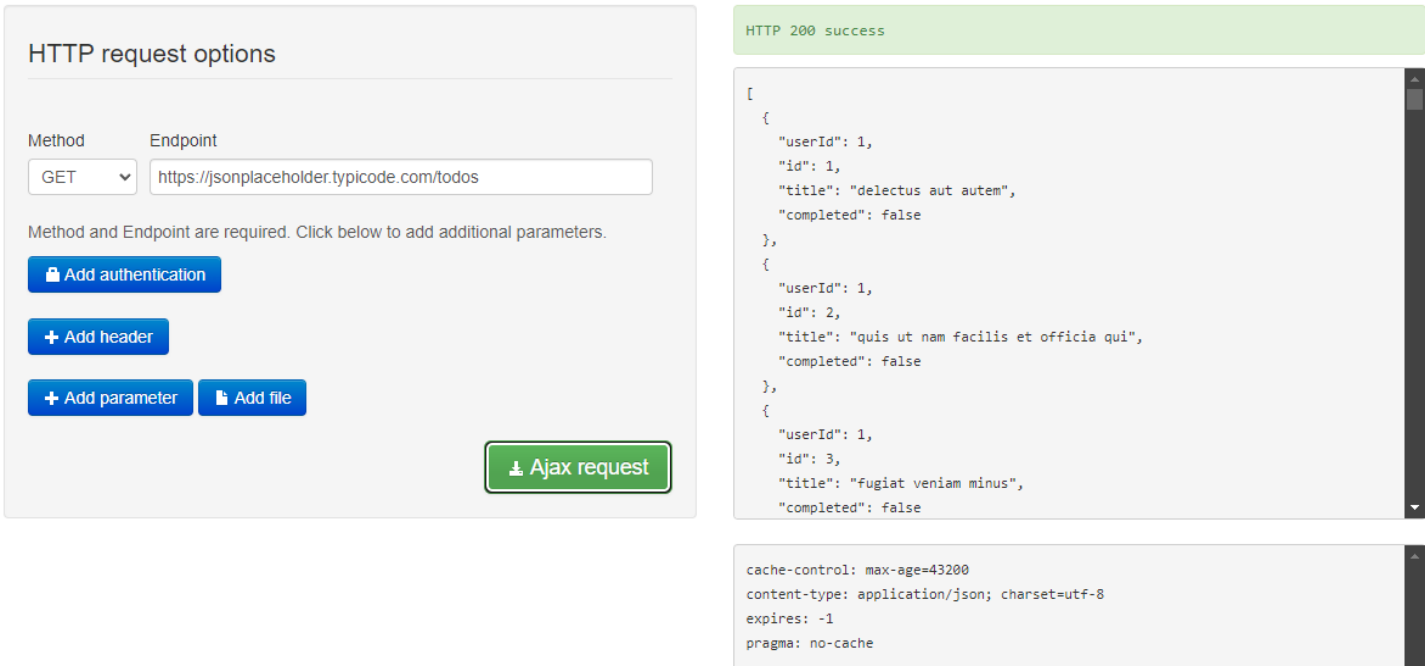
```
> fetch('https://jsonplaceholder.typicode.com/users/1/posts')
    .then((response) => response.json())
    .then((json) => console.log(json));
< ▶ Promise {<pending>}
  ▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
>
```

f) Busca tres clientes REST online, lístalos y usa uno para obtener todos las TODO's de JSONPlaceholder.

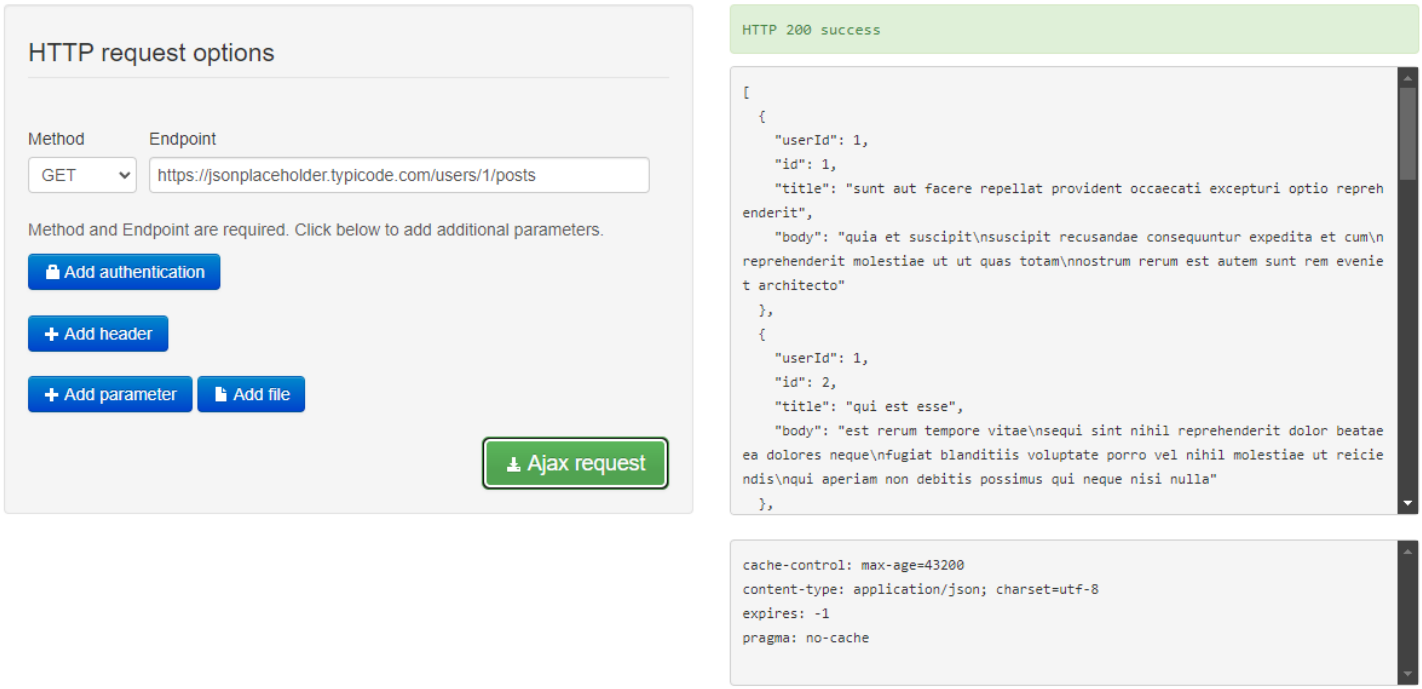
https://extendsclass.com/rest-client-online.html

https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo?hl=es (extensión para chrome)

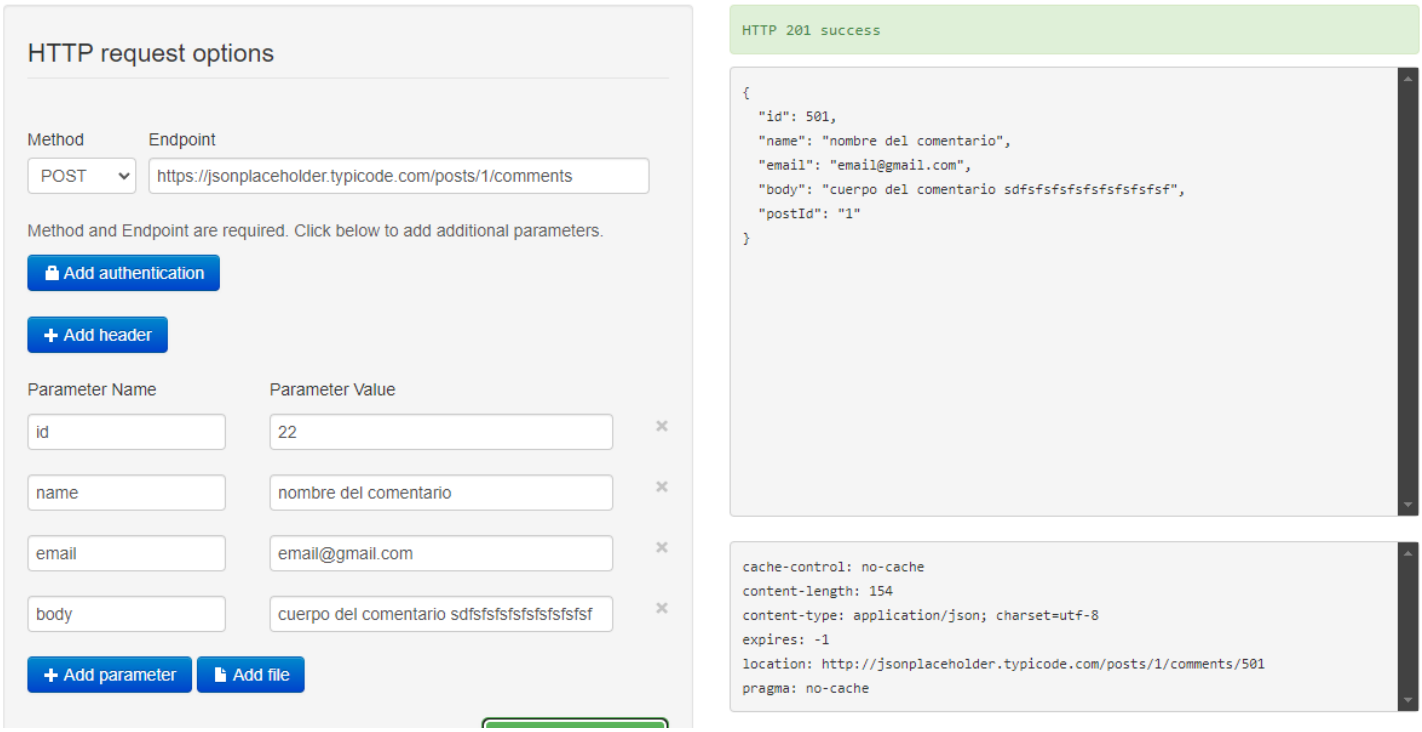
https://resttesttest.com/



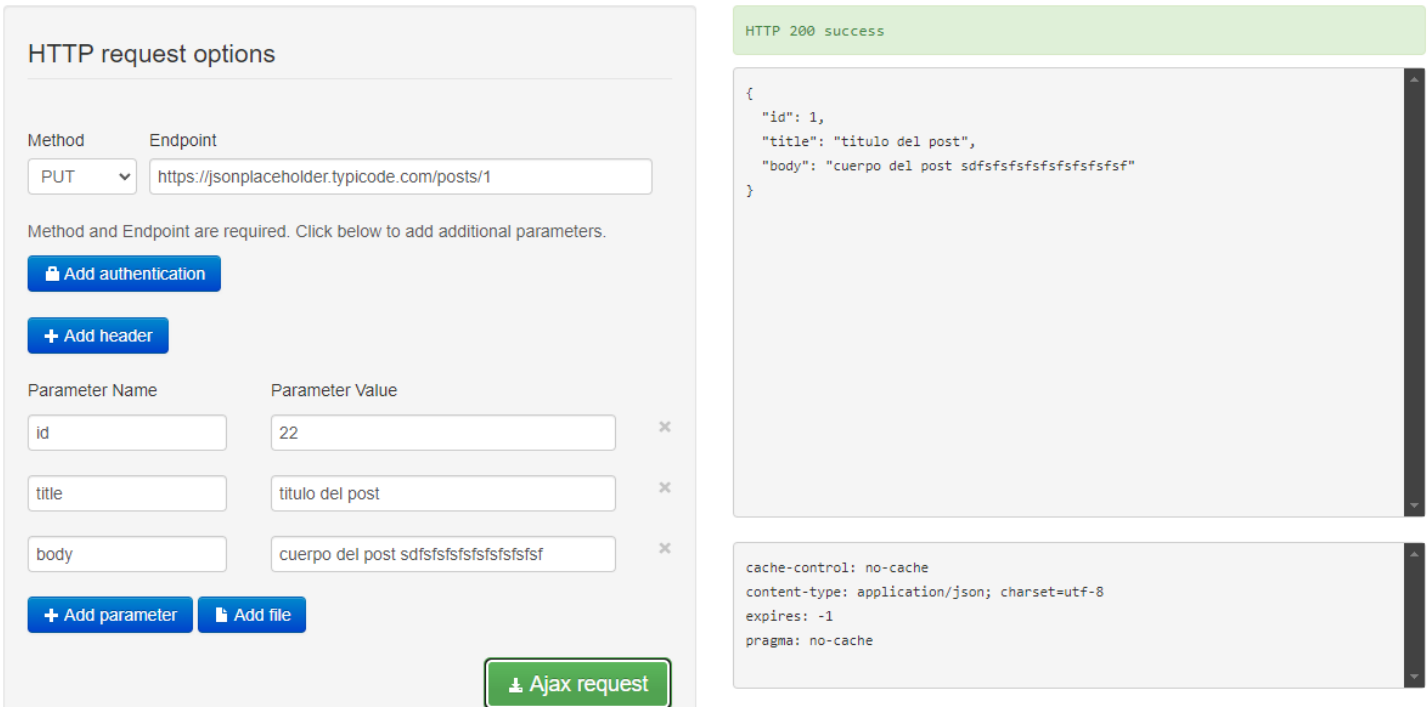
g) Usando Fetch API o un cliente REST online, obtén todas los posts del usuario 1 de JSONPlaceholder



h) Usando Fetch API o un cliente REST online, crea un nuevo comentario a un post de JSONPlaceholder..



i) Usando Fetch API o un cliente REST online, actualiza o modifica un post de JSONPlaceholder.



j) Usando Fetch API o un cliente REST online, borra el usuario 5 de JSONPlaceholder.