

{desafío}
latam_

API REST y acceso a recursos remotos _

Parte I



Cliente - Servidor

Introducción

Internet ha traído un cambio revolucionario en el ámbito de las tecnologías, interconectando a el mundo entero. En este contexto, las tecnologías de comunicación en internet siguen una arquitectura y una estructura específica para conectarse entre sí. La más popular es la arquitectura cliente-servidor, la cual profundizaremos a continuación.

Es fundamental entender estos conceptos básicos de programación moderna, dado que hoy por hoy la gran mayoría de los sistemas y aplicaciones que se construyen siguen un standard de arquitectura basada en el concepto cliente-servidor.

Objetivos

- Identificar que es un cliente
- Identificar que es un servidor
- Comprender la integración entre un cliente y un servidor



Arquitectura Cliente - Servidor

La arquitectura cliente-servidor se denomina estructura de computación de red porque cada solicitud y sus servicios asociados se distribuyen a través de una red privada o pública en internet.

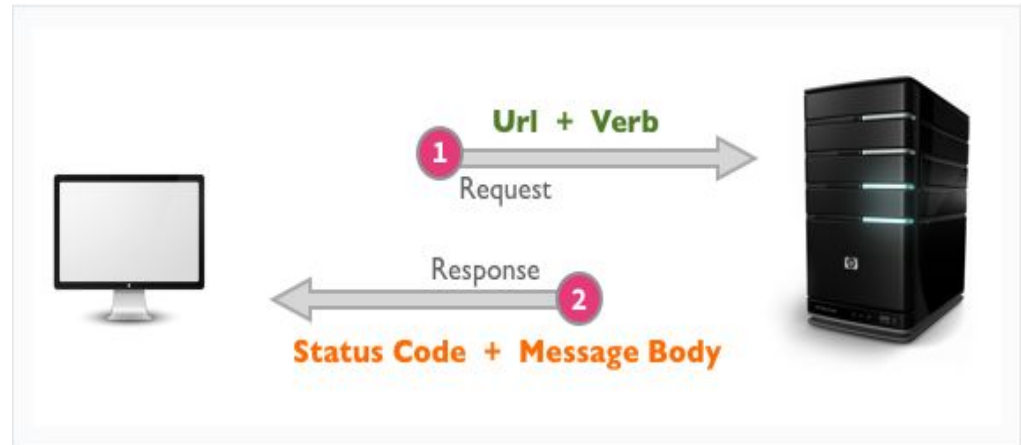


Arquitectura Cliente - Servidor

En la arquitectura cliente-servidor, cuando la computadora cliente envía una solicitud de datos al servidor a través de Internet, el servidor acepta la solicitud, la procesa y entrega los paquetes de datos solicitados al cliente. Una característica especial es que la computadora servidor tiene el potencial de administrar numerosos clientes al mismo tiempo. Además, un solo cliente puede conectarse a numerosos servidores en una sola marca de tiempo, donde cada servidor proporciona un conjunto diferente de servicios a ese cliente específico.

Arquitectura Cliente - Servidor

Las computadoras cliente proporcionan una interfaz para permitir que un usuario de la computadora solicite servicios del servidor y muestre los resultados que el servidor devuelve. Los servidores esperan que lleguen las solicitudes de los clientes y luego responden a ellos. Idealmente, un servidor proporciona una interfaz transparente estandarizada a los clientes para que los clientes no necesiten conocer los detalles del sistema (es decir, el hardware y el software) que proporciona el servicio.



Cliente

En una arquitectura cliente-servidor de una red computacional el cliente se puede definir como un procesador remoto que recibe y envía peticiones de un servidor. Cada teléfono móvil android desde su ubicación en el planeta es un cliente para cada una de las empresas o servicios que prestan diferentes aplicaciones desde sus servidores principales.



Principales Funciones de un Cliente

- Iniciar el request o solicitud
- Esperar y recibir respuestas
- Poder conectarse con un servidor o múltiples servidores al mismo tiempo
- Interactuar con los usuarios finales usando una Graphical User Interface (GUI)

Servidor

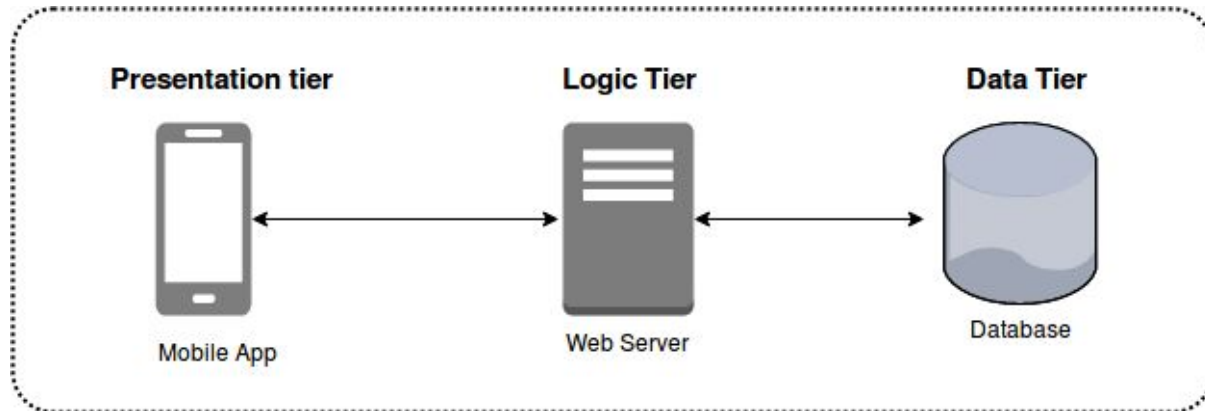
Un servidor es un computador de red, programa o dispositivo que procesa solicitudes de un cliente. En la internet hoy por hoy un servidor web es una computadora o una red de computadoras que utiliza el protocolo HTTP para enviar datos y archivos a la computadora de un cliente cuando el este último lo solicita.



Principales Funciones de un Servidor

- Esperar por los requerimientos de un cliente
- Recibido un requerimiento, procesar, preparar y enviar las respuestas
- Aceptar múltiples conexiones de una largo número de clientes
- Normalmente, no interactuar directamente con los usuarios finales

Cliente - Servidor en la actualidad



Cliente - Servidor en la actualidad

En una arquitectura cliente-servidor web moderna, se consideran los siguientes factores específicos:

- Un conjunto específico de idiomas junto con un estándar de comunicación, exclusivamente un protocolo para la interacción de dos sistemas. Los más populares son HTTP y HTTPS (Protocolo seguro de transferencia de hipertexto)
- Mecanismo y protocolo para solicitar los aspectos requeridos del servidor. Eso podría estar en cualquier estructura de datos formateados. Los formatos más implementados y populares se realizan en XML y JSON
- Respuesta del servidor enviando en una estructura de datos formateados (generalmente XML o JSON)

API REST

Introducción

Al consumir servicios web REST desde nuestras aplicaciones somos capaces de desarrollar sistemas móviles multi-usuario, masivos y de largo alcance, entendiendo que desde el lugar en que se encuentre un cliente android, este se puede conectar con servicios centralizados expuestos con la tecnología rest.

En la actualidad casi no existen proyectos o aplicaciones que no disponga de una API REST para la creación de servicios profesionales. WhatsApp, YouTube, Twitter, Facebook y cientos de miles de empresas generan negocio gracias a REST y las APIs REST.

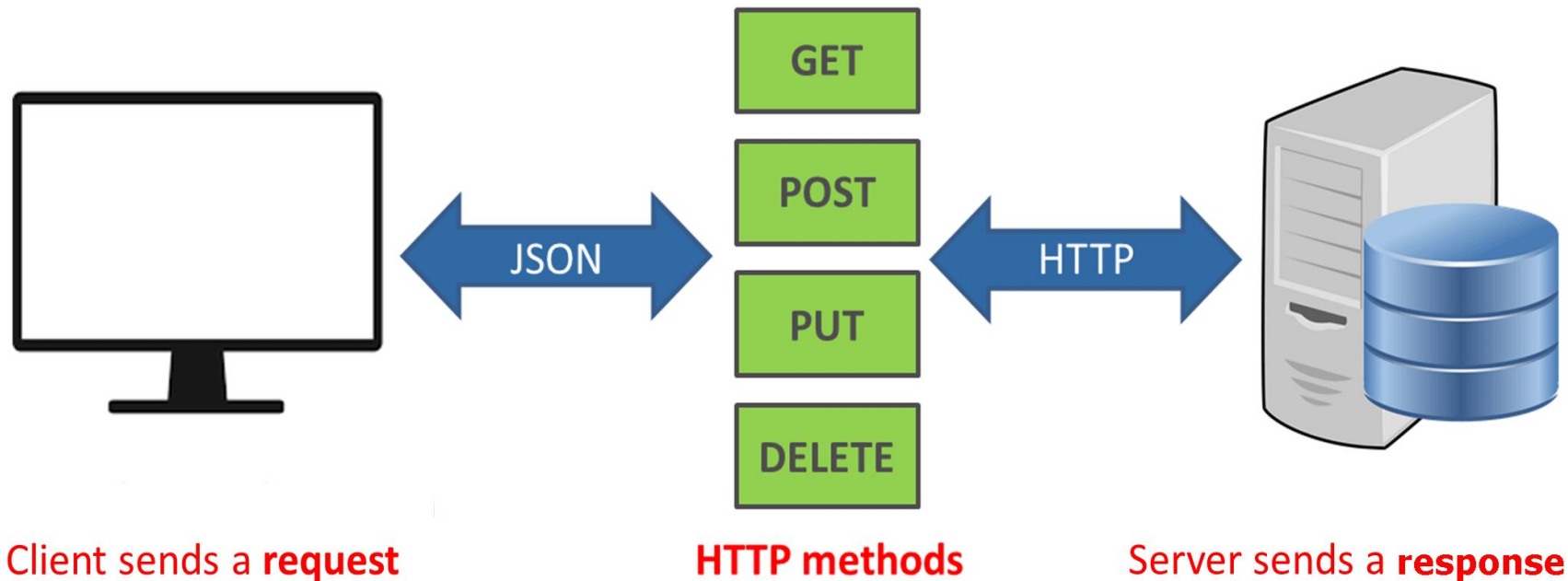
Conocer cómo funcionan estos servicios proporcionará al alumno herramientas para trabajar en cualquier empresa que ofrezca servicios rest, o necesite que las aplicaciones se conecten a un tercero.

Objetivos

- Conocer qué es un api rest
- Comprender cómo interactúa un api rest con las aplicaciones android
- Diferenciar los métodos para consumir e interactuar con una API Rest



Api REST



Api REST

Una api REST simplifica el desarrollo al admitir métodos HTTP estándar, manejo de errores y otras convenciones RESTful. Un api REST expone recursos a través de una o distintas direcciones web (URL) utilizando arquitectura basada en HTTP, exponiendo servicios web para el procesamiento de distintas tareas y operaciones.

REST le pide a los desarrolladores que usen métodos HTTP explícitamente y de manera consistente con la definición del protocolo. Este principio de diseño REST básico establece un mapeo uno a uno entre las operaciones de creación, lectura, actualización y eliminación (CRUD) y los métodos HTTP.

Principios de REST

- Cliente - Servidor
- Sin estado (Stateless)
- Uso de Caché
- Interfaz Uniforme
- Sistema en Capas
- Código bajo demanda (opcional)

Principios de REST

Si intentamos acceder a una página que no existe, el servidor te devolverá la página 404 No encontrada o error 404, si arruinamos nuestro código rest del servidor, obtendremos un error 500 de sistemas. Estos son códigos de respuesta que el servidor envía para que su navegador sepa lo que está sucediendo.

Tipos de Códigos de respuesta:

- Entre 200-299 significa que la solicitud fue exitosa.
- 300-399 significa que la solicitud estuvo bien, pero debe hacer otra cosa.
- 400-499 es un error dónde no se encuentra algún objeto o recurso, o no se ha invocado correctamente.
- 500-599 es un error realmente malo

Operaciones o métodos REST

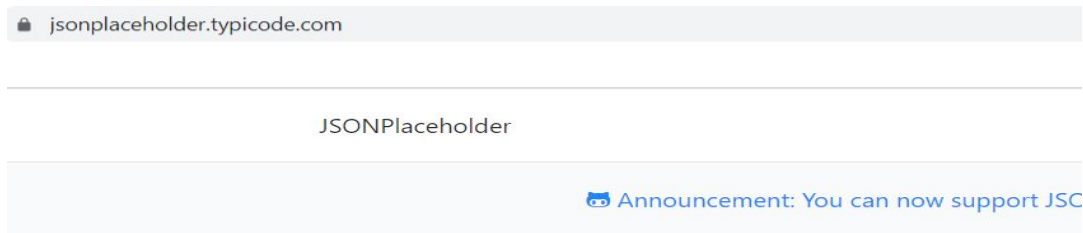
- **GET:** Es utilizado únicamente para consultar información al servidor, muy parecidos a realizar un SELECT a la base de datos. No soporta el envío del payload
- **POST:** Es utilizado para solicitar la creación de un nuevo registro, es decir, algo que no existía previamente, es decir, es equivalente a realizar un INSERT en la base de datos. Soporta el envío del payload.
- **PUT:** Se utiliza para actualizar por completo un registro existente, es decir, es parecido a realizar un UPDATE a la base de datos. Soporta el envío del payload.

Operaciones o métodos REST

- **PATCH:** Este método es similar al método PUT, pues permite actualizar un registro existente, sin embargo, este se utiliza cuando actualizar solo un fragmento del registro y no en su totalidad, es equivalente a realizar un UPDATE a la base de datos. Soporta el envío del payload.
- **DELETE:** Este método se utiliza para eliminar un registro existente, es similar a DELETE a la base de datos. No soporta el envío del payload.
- **HEAD:** Este método se utilizar para obtener información sobre un determinado recurso sin retornar el registro. Este método se utiliza a menudo para probar la validez de los enlaces de hipertexto, la accesibilidad y las modificaciones recientes.

Ejercicio 1

Descargar e instalar postman y utilizarlo para el consumo de métodos REST del api público fake JsonPlacesHolder. Postman: <https://www.getpostman.com/downloads/>.



Resources

JSONPlaceholder comes with a set of 6 common resources:

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

Ejercicio 1 - Solución

GET Get posts JsonPlaceholder

▶ Get posts JsonPlaceholder

GET https://jsonplaceholder.typicode.com/posts

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

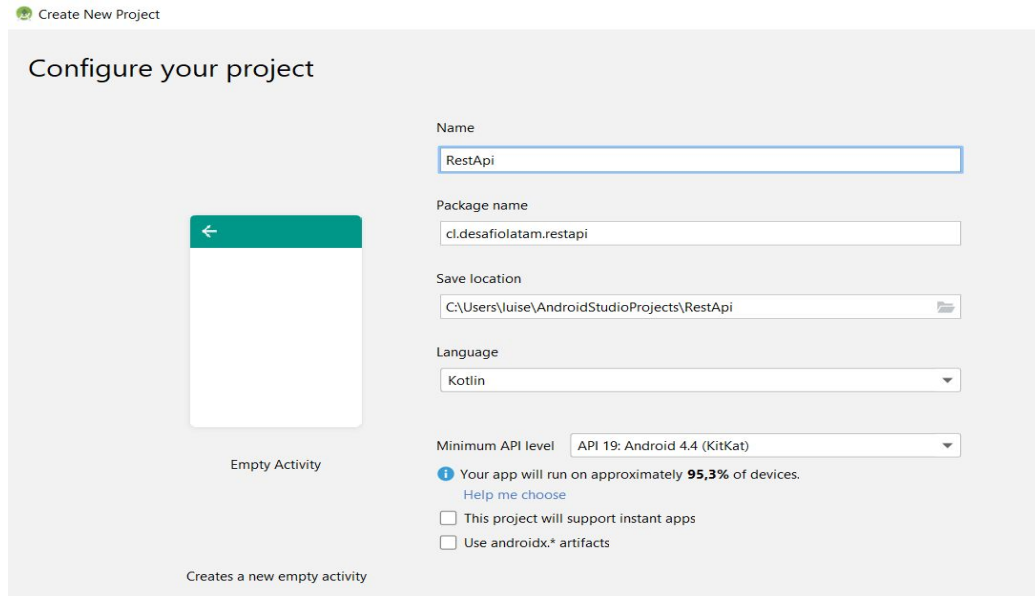
Body Cookies (1) Headers (20) Test Results Status: 200 OK

Pretty Raw Preview Visualize BETA JSON

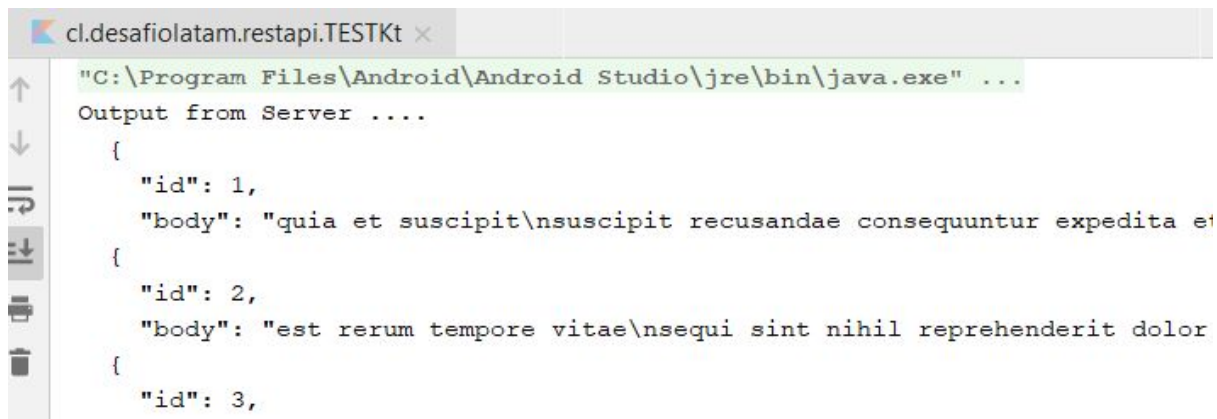
```
1 [
2   {
3     "userId": 1,
4     "id": 1,
5     "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
6     "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum architecto"
7   },
8   {
9     "userId": 1,
10    "id": 2,
11    "title": "qui est esse",
12    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate pc
```


Ejercicio 2

Consultar el método `get posts` del api `JsonPlaceholder` desde android studio en un proyecto llamado “RestApi”, con la utilización de los métodos nativos de `URLConnection`, para visualizar por consola del IDE los resultados una vez inicie la aplicación android.



Ejercicio 2 - Solución



```
cl.desafiolatam.restapi.TESTKt x
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
Output from Server ....
[
  {
    "id": 1,
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita e
  },
  {
    "id": 2,
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor
  },
  {
    "id": 3,
```

Ventajas de usar REST

- Separación entre el cliente y el servidor
- Visibilidad, fiabilidad y escalabilidad
- La API REST siempre es independiente del tipo de plataformas o lenguajes

HTTP Client con Retrofit

Introducción

A continuación utilizaremos retrofit, el cual es una librería que encapsula los métodos de httpCLient. Es necesario aprender a implementar retrofit para la comunicación con apis externas a nuestras aplicaciones móviles, por su amplio uso y características que lo han posicionado como casi un estándar en la industria.

Retrofit es ampliamente utilizada en la comunidad de desarrollo android, convirtiéndose prácticamente en un estándar de la industria, razón que nos compromete estudiar y conocer sus métodos e implementación en nuestros proyectos.

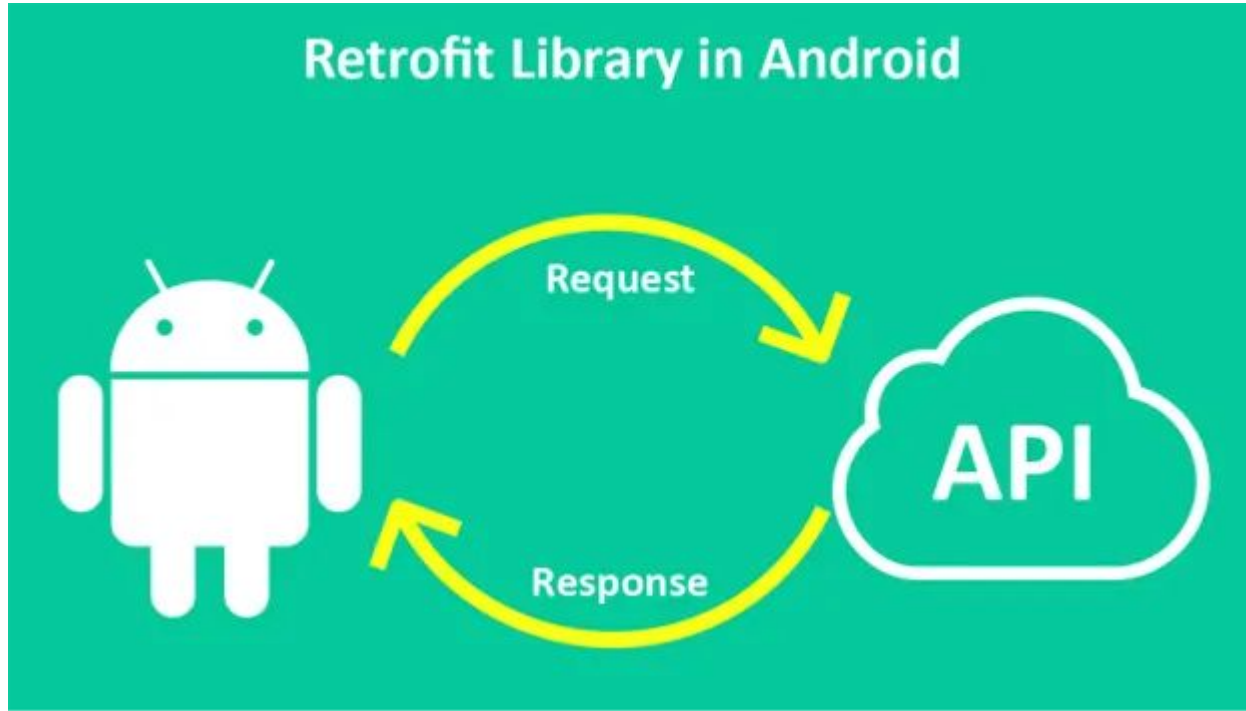
Realizaremos ejercicios con distintas operaciones de creación, actualización, consulta y eliminación de datos.

Objetivos

- Conocer los conceptos del cliente http retrofit
- Implementar Retrofit en una aplicación Android
- Conocer alternativas a Retrofit



Retrofit



Retrofit

Es una librería Http client para específicamente realizar peticiones http tipo rest para android y java, permitiendo el envío y recepción de respuestas de datos entre nuestras aplicaciones y distintas plataformas de software a través de internet. Retrofit simplifica la codificación para lograr esta comunicación, gestionando y transformando nuestros datos para ser traspasados a nuestras clases pojo (plain old java object).

Para realizar una petición con Retrofit se necesitará lo siguiente:

- Una clase Retrofit
- Una Interfaz de operaciones
- Clases de modelo de datos o pojo

Ejercicio 3

Agregar dependencias retrofit y gson al proyecto “RestApi”.



Ejercicio 3 - Solución

```
implementation 'com.squareup.retrofit2:retrofit:2.6.0'
```

```
implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
```

Características de Retrofit como cliente HTTP

- Declaración de API
- Método Request
- Manipulación de URL
- Request Body
- Form Encode, Multipart
- Manipulación de Header

Ejercicio 4

Crear una aplicación que se conecte a un api pública que disponga de varios métodos HTTP para utilizar servicios REST. En este caso utilizaremos la api pública de JsonPlaceholder.com la cuál nos brinda distintos métodos HTTP Rest, con data de prueba que nos ayuda a emular este tipo de desarrollos con comodidad.

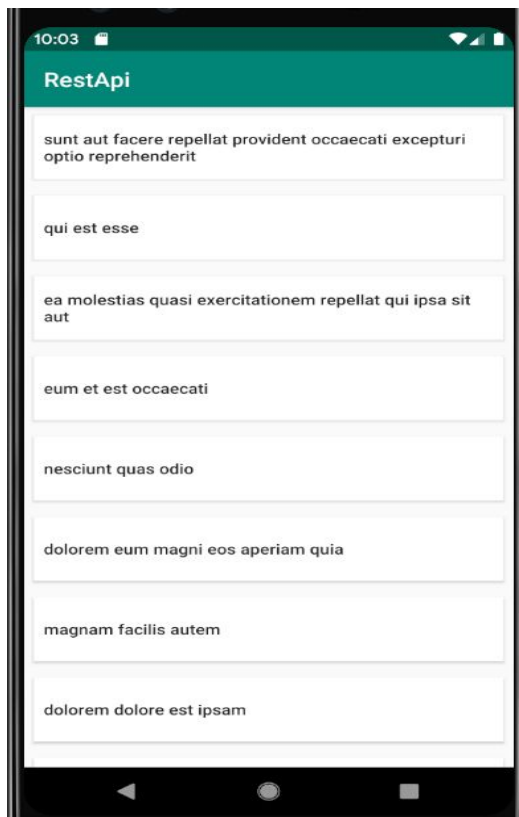
1. <https://jsonplaceholder.typicode.com/>
2. <https://jsonplaceholder.typicode.com/posts>



A screenshot of a web browser displaying the JSONPlaceholder API endpoint for posts. The address bar shows the URL `jsonplaceholder.typicode.com/posts`. The browser content area displays a JSON array of two post objects. The first object has `userId: 1`, `id: 1`, `title: "sunt aut facere repellat provident occaecati excepturi"`, and `body: "quia et suscipit suscipit recusandae consequuntur expedit"`. The second object has `userId: 1`, `id: 2`, `title: "qui est esse"`, and `body: "est rerum tempore vitae sequi sint nihil reprehenderit d neque nisi nulla"`.

```
[
  - {
    userId: 1,
    id: 1,
    title: "sunt aut facere repellat provident occaecati excepturi",
    body: "quia et suscipit suscipit recusandae consequuntur expedit",
  },
  - {
    userId: 1,
    id: 2,
    title: "qui est esse",
    body: "est rerum tempore vitae sequi sint nihil reprehenderit d neque nisi nulla"
  },
]
```

Ejercicio 4 - Solución



Volley

Volley es una biblioteca HTTP-Client al igual que Retrofit, que hace que la creación de redes para aplicaciones de Android sea más fácil y, lo que es más importante, más rápida. Volley está disponible en [GitHub](#).

Volley ofrece los siguientes beneficios:

- Programación automática de solicitudes de red.
- Múltiples conexiones de red concurrentes.
- Disco transparente y memoria caché de respuesta con coherencia de caché HTTP estándar .
- Soporte para solicitud de priorización.
- Solicitud de cancelación API. Puede cancelar una sola solicitud o puede establecer bloques o ámbitos de solicitudes para cancelar.
- Facilidad de personalización, por ejemplo, para reintentos y retrocesos.
- Ordenación sólida que facilita el llenado correcto de la interfaz de usuario con los datos obtenidos de forma asincrónica de la red.
- Herramientas de depuración y rastreo.

Ejercicio 5

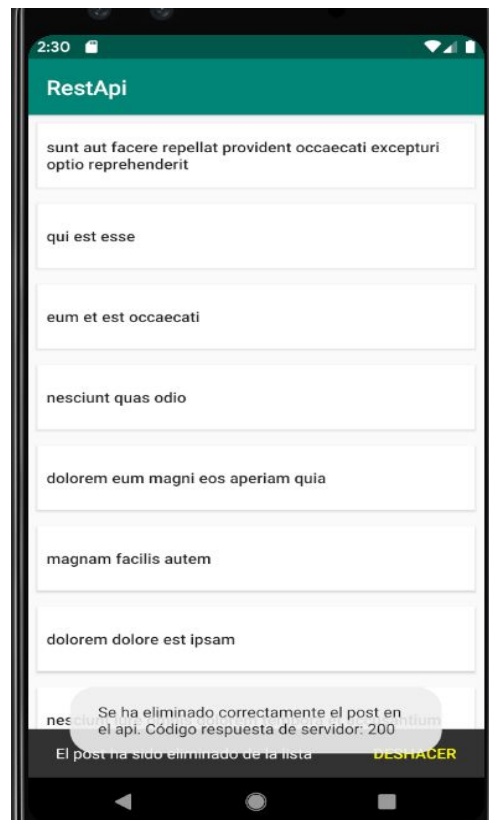
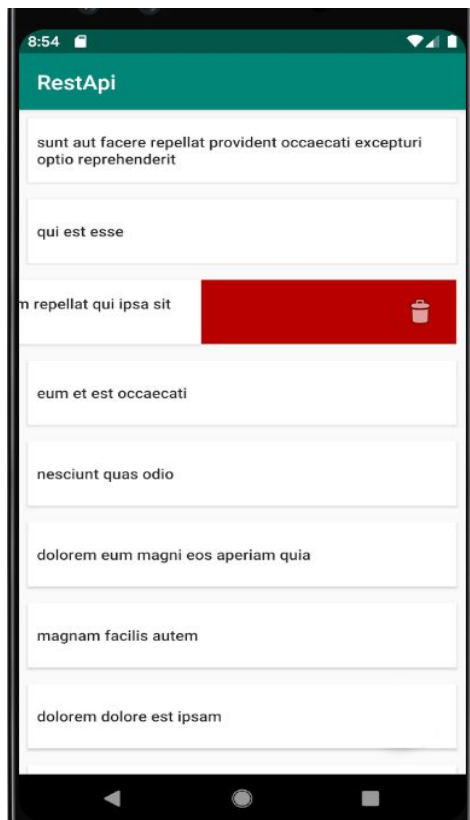
Continuando con el ejercicio anterior, profundizaremos la funcionalidad de nuestra app “Rest Api”, agregando en esta oportunidad la opción de eliminar un post de nuestra vista y emulando su eliminación en el api invocando el método http DELETE como lo indica la página del api, además realizaremos esta tarea aprendiendo a trabajar con una técnica denominada “SwipeToDelete”, de la misma forma que gmail envía a archivo los correos de tu bandeja.

Routes

All HTTP methods are supported.

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

Ejercicio 5 - Solución



Intercambio de datos con JSON

Introducción

JSON ha ido desplazando desde hace algunos años las antiguas formas de intercambio de datos que por ejemplo se hacían utilizando XML, llegando a tal punto que las bases de datos modernas han comenzado a evolucionar hacia el almacenamiento de datos JSON en su forma compilada BSON, como lo son por ejemplo, MongoDB y Oracle 12c+.

Actualmente el estándar de comunicación es a través de Json, Comprender la sintaxis y cómo se ordenan los datos en este formato será de vital importancia para nuestro desarrollador como creadores de apps.

Objetivos

- Conocer el formato de intercambio de datos JSON
- Comprender la estructura que conforma un JSON
- Identificar la cabecera de una consulta api rest
- Identificar el cuerpo de una consulta api rest



JSON

```
{
  "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard Generalized Markup Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language, used to create markup
languages such as DocBook.",
            "GlossSeeAlso": ["GML", "XML"]
          },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```

JSON

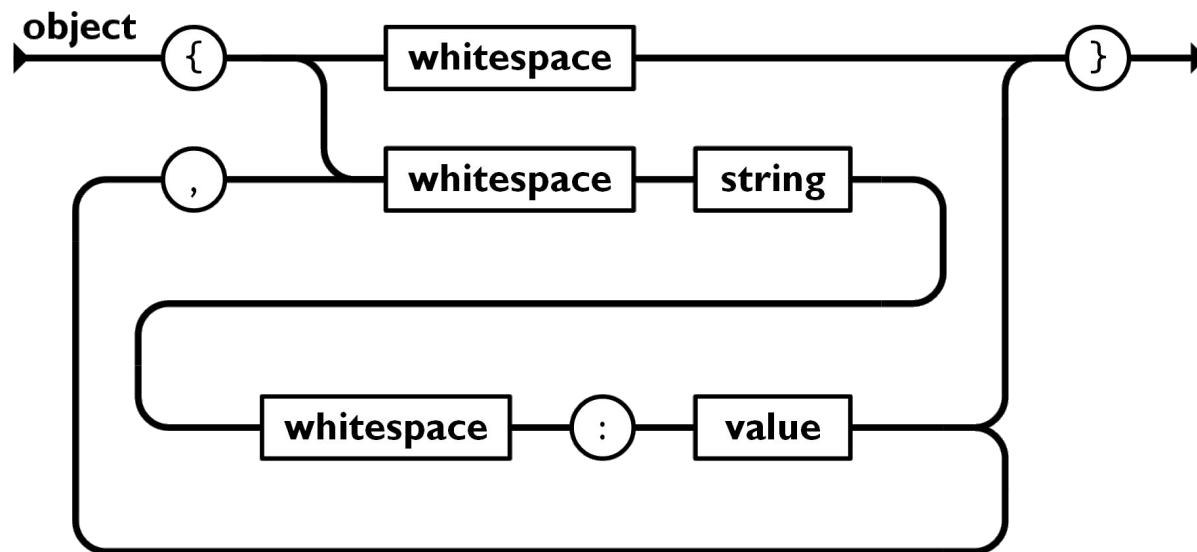
JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Es fácil para los humanos leer y escribir. Es fácil para las máquinas analizar y generar. Se basa en un subconjunto del lenguaje de programación denominados JavaScript Standard ECMA-262.

JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluido Kotlin, C, C ++, C#, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

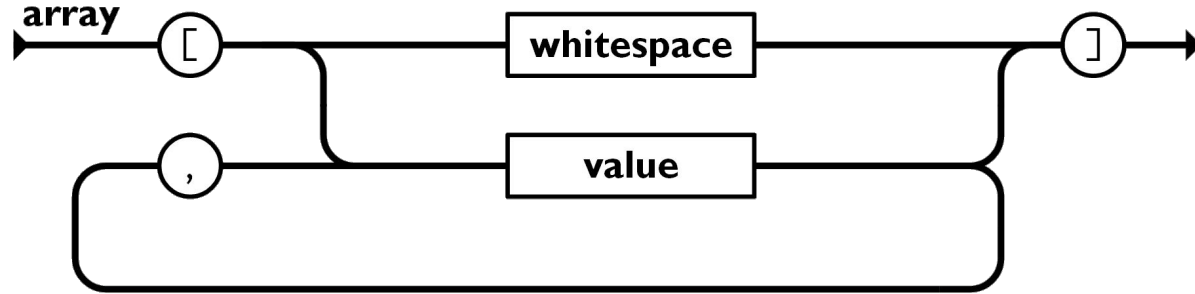
Estructura de un JSON

- Una colección de pares de nombre / valor, en distintos idiomas y lenguajes de programación; esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista con clave o arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes de programación, esto se implementa como arreglos, vectores, listas o secuencias.

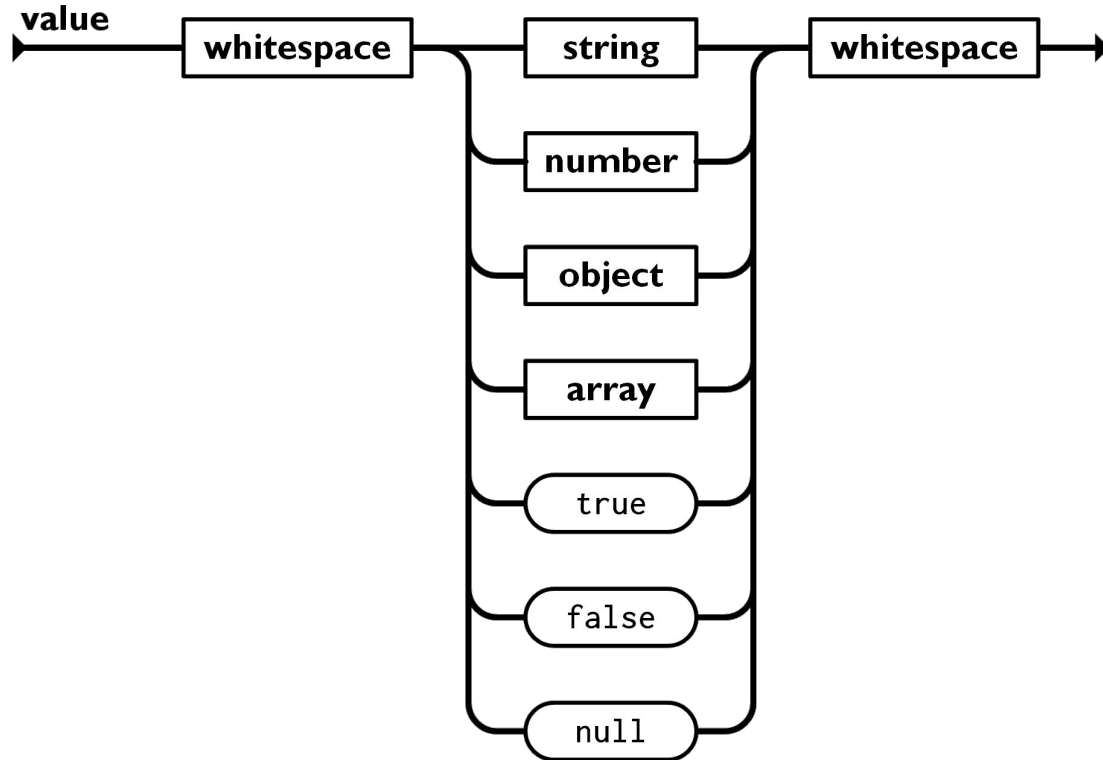
Estructura de un objeto JSON



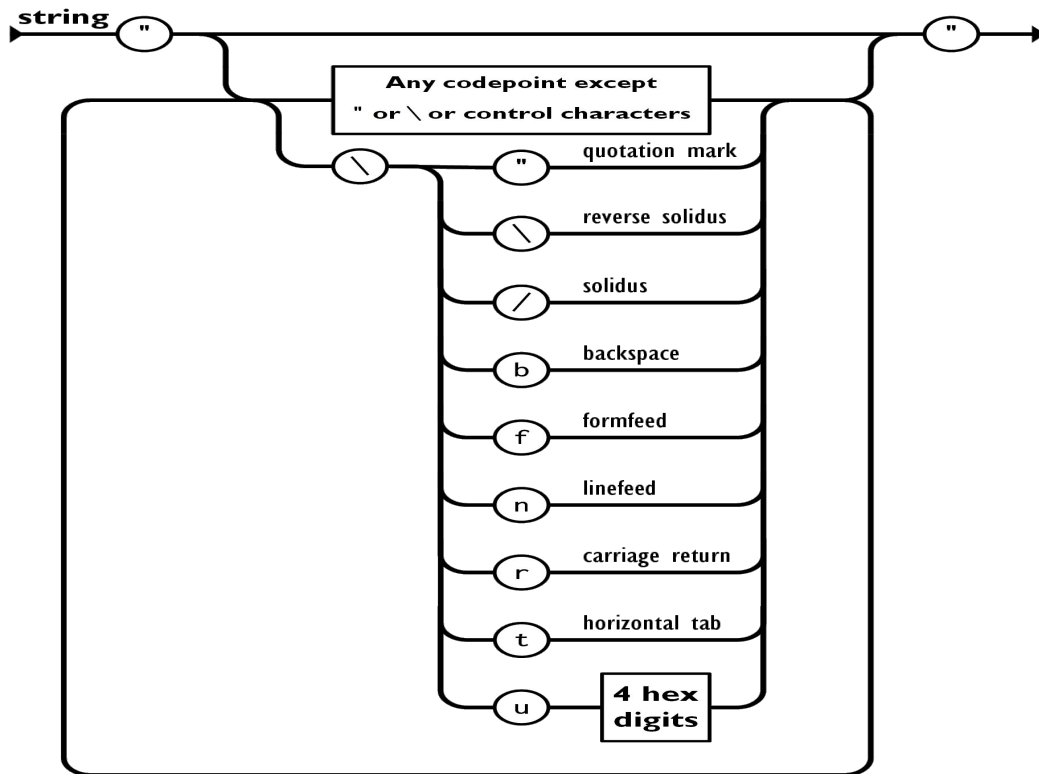
Estructura de un Array JSON



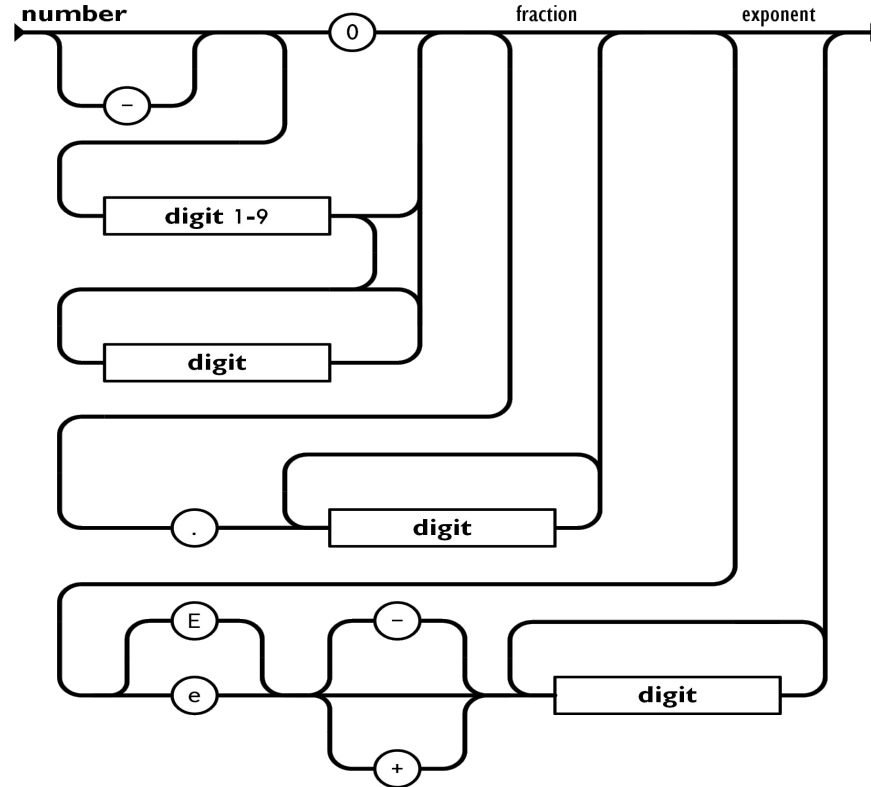
Estructura de un valor JSON



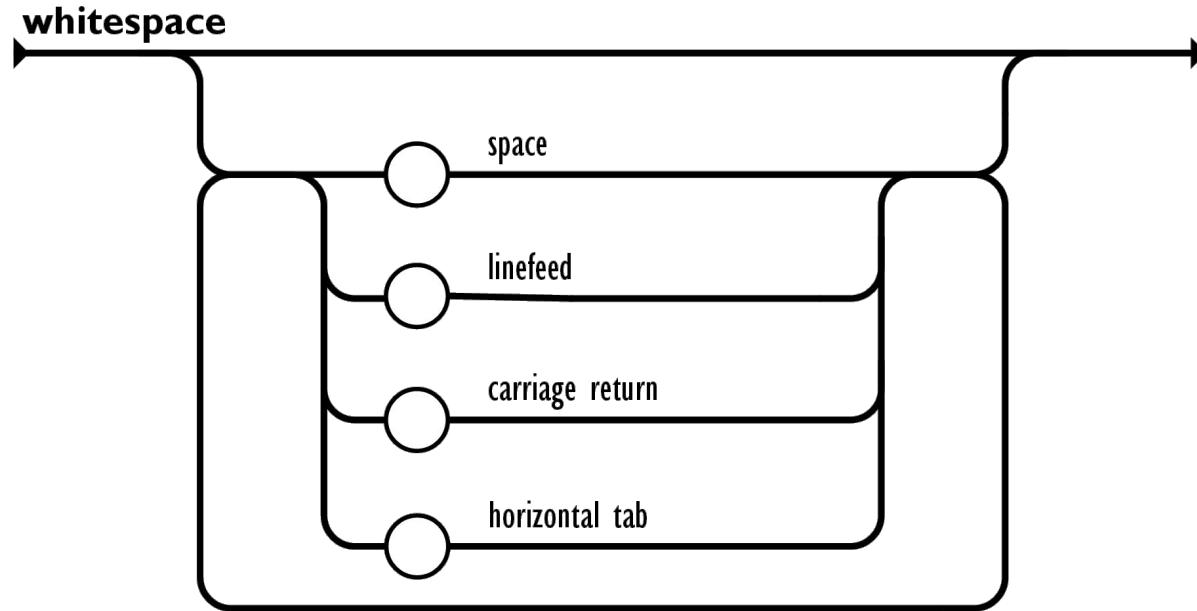
Estructura de un String JSON



Estructura de un Number JSON



Estructura de un espacios en blanco en JSON



JSON - Retrofit Header

Los encabezados con los que nos encontraremos más durante el trabajo con un API son los siguientes:

- Authorization
- WWW-Authenticate
- Accept-Charset
- Content-Type
- Cache-Control

JSON - Retrofit Header

En retrofit para asignar parámetros a nuestros métodos http de las interfaces, basta con agregar la anotación @Headers sobre la anotación del tipo de método http, ya sea get, post, put, delete.

Ejemplo:

```
@Headers("Content-Type: application/json; charset=UTF-8")
```

JSON - Retrofit Body

El cuerpo de un objeto json, no requiere mayor explicación de la que ya hemos dado, el cuerpo json (body) es ese objeto json que puede anexarse a una solicitud (request) o recibirse en la respuesta del servicio.

```
{
  "customers":
  {
    "firstName": "Joe",
    "lastName": "Bloggs",
    "fullAddress":
    {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": 10021
    }
  }
}
```

JSON - Retrofit Body

En kotlin utilizando retrofit podemos disponer de una anotación `@Body` que se utiliza para asignar un objeto tipo pojo (clase que contiene instancias de los datos similares a los que contiene el objeto json) a una respuesta o solicitud. Por ejemplo, anteponiendo los siguientes ejercicios, así sería una asignación de un método Post en la interfaz de retrofit para enviar un nuevo post al servidor:

```
@Headers("Content-Type: application/json; charset=UTF-8")
@POST("/posts")
fun createNewPost(@Body post: Post): Call<Post>
```

Dónde "Post" es nuestra clase pojo que contiene los objetos del json body.

Ejercicio 6

Crear un post nuevo usando Postman con los datos indicados en la página del api de JsonPlaceholder, enviando cabeceras y cuerpos (headers, body) como se indican en su guía:

<https://jsonplaceholder.typicode.com/guide.html>

Ejercicio 6 - Solución


POST ▼ https://jsonplaceholder.typicode.com/posts

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL BETA **JSON** ▼

```
1 {  
2   "title": "Este es el titulo del nuevo post",  
3   "body": "Descripcion del post",  
4   "userId": 2  
5 }
```

Body Cookies (1) Headers (19) Test Results Status: 201 Created

Pretty Raw Preview Visualize BETA JSON ▼ 

```
1 {  
2   "title": "Este es el titulo del nuevo post",  
3   "body": "Descripcion del post",  
4   "userId": 2,  
5   "id": 101  
6 }
```

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com