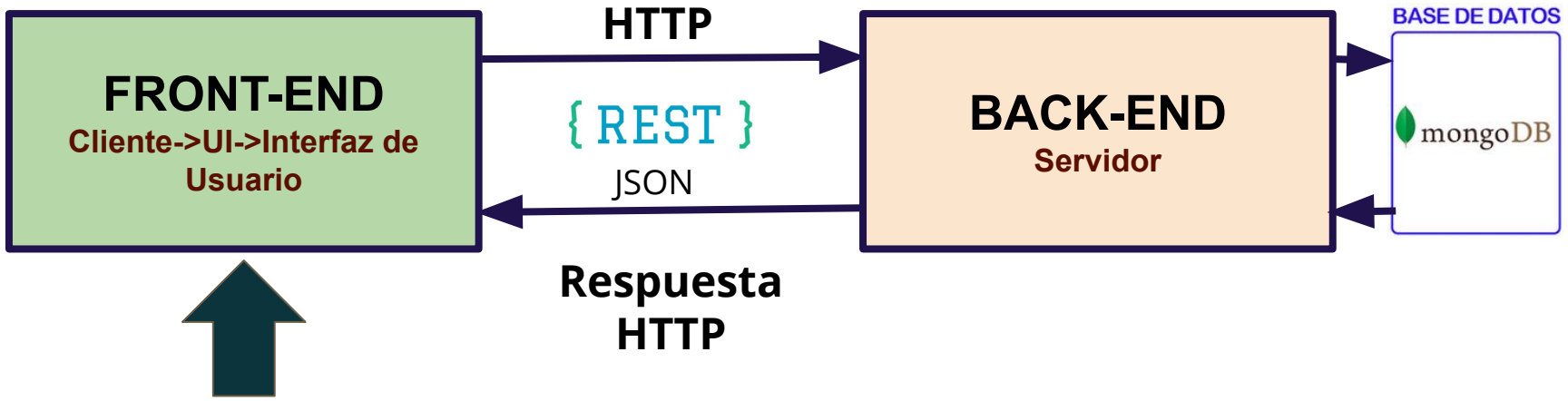
Componentes

Angular, React, Vue, Flutter, App Android o IOS

Java (Spring Boot, Micronaut, Quarkus), Node.Js (Express), Python (Flask), Dart, Golang, Serverless



Flutter

Es la parte de una aplicación que interactúa con los usuarios, es conocida como el lado del cliente

Lado del Servidor. El back end del sitio web consiste en un servidor, una aplicación y una base de datos. Se toman los datos, se procesa la información y se envía al usuario.

Introducción a la Programación Multiplataforma para Web y Mobile

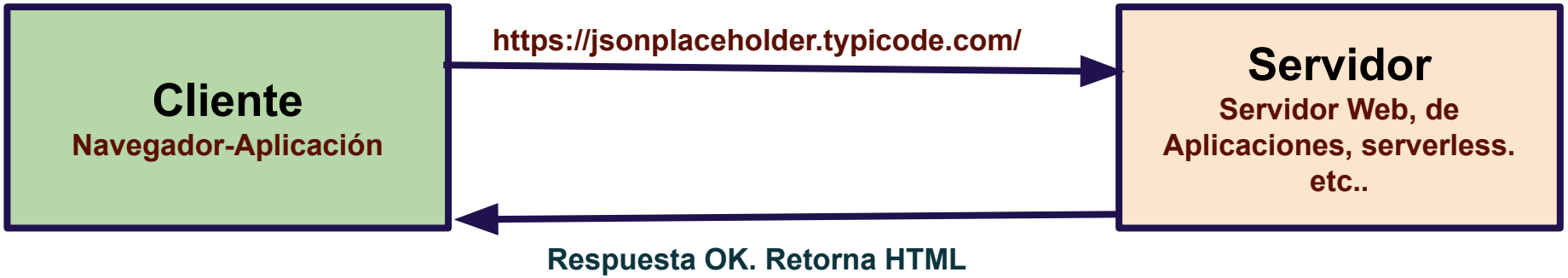
HTTP

Mgter. Ing. Carlos Alejandro Martinez

HTTP

Un protocolo define las reglas y la estructura que se va a intercambiar entre clientes y servidores

Protocolo de transferencia de hipertexto (en inglés, Hypertext Transfer Protocol, abreviado HTTP)



Ubicación en la pila de protocolos	
Aplicación	HTTP/2
Transporte	TCP
Red	IP
Estándares	
RFC 1945 (HTTP/1.0, 1996)	
RFC 2616 (HTTP/1.1, 1999)	
RFC 7540 (HTTP/2.0, 2015)	

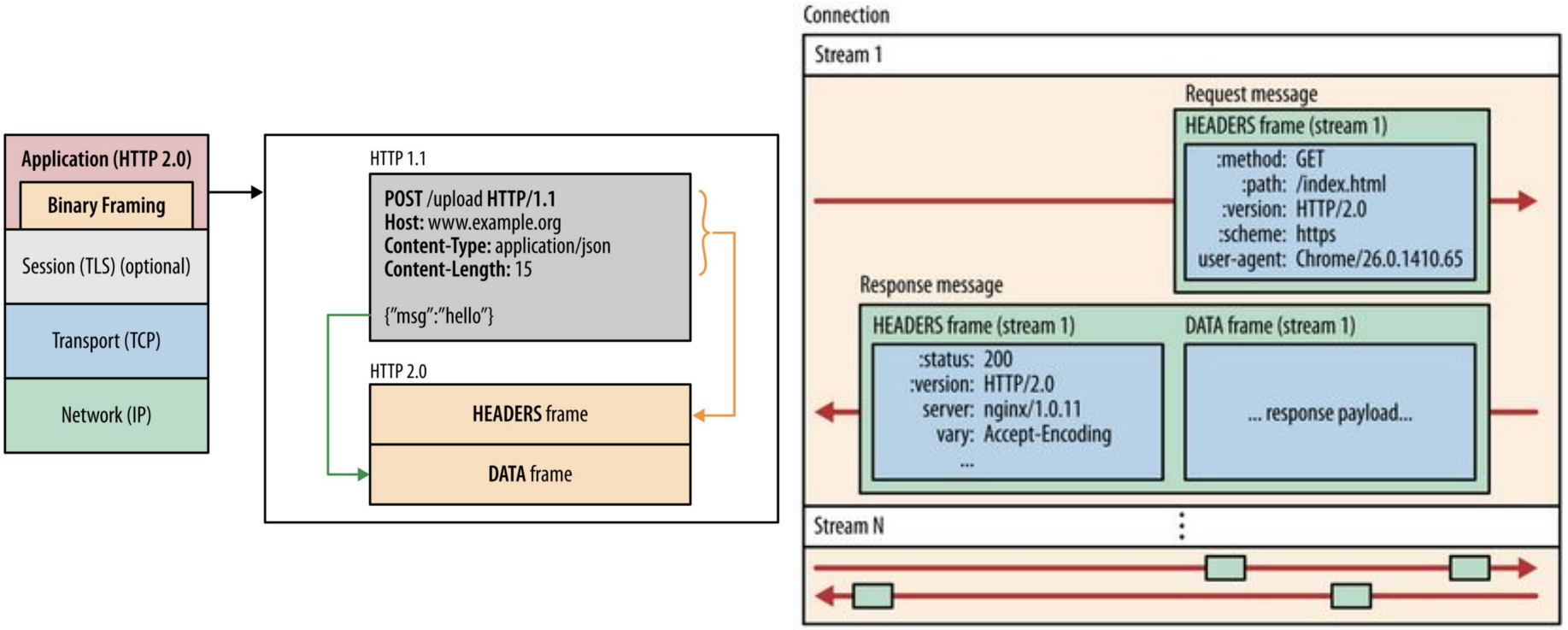
HTTP es un protocolo cliente-servidor, lo que significa que el cliente envía una petición al servidor y espera un mensaje de respuesta del servidor.

Es un protocolo sin estado, lo que significa que el servidor no guarda información del cliente, cada petición es independiente de las demás.

HTTP


GET
Scheme: https
Host: jsonplaceholder.typicode.com
Filename: /
Cliente
Navegador-Aplicación
GET https://jsonplaceholder.typicode.com/
Servidor
Servidor Web, de Aplicaciones, serverless. etc..
Respuesta OK. Retorna HTML
Estado 200 OK
Versión HTTP/2
Transferido 2,15 KB (tamaño 6,86 KB)
{JSON} Placeholder
Response payload
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
<link rel="stylesheet" href="/style.css" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/prism/1.21.0/themes/p
<title>JSONPlaceholder - Fake online REST API for testing and prototyping</title>

HTTP/2.0



http v2=>https://developers.google.com/web/fundamentals/performance/http2?hl=es-419

Protocolo HTTP/2

HTTP (Hiper Text Transfer Protocol). Protocolo de la capa de Aplicación. Cliente Servidor.

1 Petición(request): Envío por parte del cliente de una petición al servidor

- **URI-URL**
  - http://servidor:80/recurso?params=uno&params=dos
- **Encabezado(Header)**
  - Metódo: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD, TRACE
  - Parámetros
- **Cuerpo (Body)**

2 Respuesta (response) Envío por parte del servidor de una respuesta al cliente

- **Encabezado**
  - Estado: 1xx: Respuesta Informativa 2xx peticiones correctas 3xx Redirecciones 4xx Errores del cliente y 5xx Errores internos
  - Parámetros
- **Cuerpo (Body)**

HTTP es un protocolo sin estado, es decir, que no se guarda ninguna información sobre conexiones anteriores

Introducción a la Programación Multiplataforma para Web y Mobile

REST y JSON

Mgter. Ing. Carlos Alejandro Martinez

¿Qué es REST?

REST: Transferencia de estado representacional (en inglés representational state transfer) o REST es un estilo de arquitectura software que se apoya totalmente en el estándar HTTP

Podemos crear APIs que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP. No existe un estándar oficial de REST.

¿Qué es API?

API es el acrónimo inglés "Application Programming Interface", es decir, "Interfaz de Programación de Aplicaciones".

> **Una "interfaz" es la forma en que dos aplicaciones o servicios se comunicar entre sí. Lo hacen exponiendo al resto de aplicaciones el conjunto de servicios y cómo se deben acceder.**

A la serie de estos servicios, se le denomina API. Por eso, las API sirven para que una aplicación pueda interactuar con otra. Ejemplo de Apis:https://jsonplaceholder.typicode.com/

URL

Un recurso es la información a la que queremos acceder o que queremos modificar o borrar. Ejemplo: Productos

Las URL, Uniform Resource Locator , son un tipo de URI, Uniform Resource Identifier

Las URL además de permitir identificar de forma única un recurso, nos permite localizarlo para poder acceder a él o compartir su ubicación.

Una URL se estructura de la siguiente forma:

{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

URL

Forma de identificar de forma unívoca un recurso

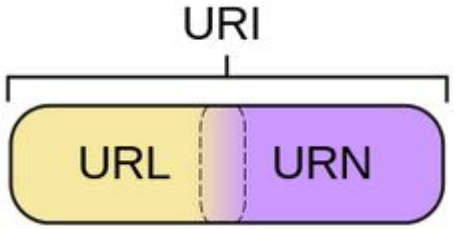
{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

https://jsonplaceholder.typicode.com:443/todos

https://jsonplaceholder.typicode.com/todos?id=1

https://jsonplaceholder.typicode.com/todos?id=1&completed=false

URL es un tipo de URI



Un **identificador de recursos uniforme** o **URI** —del inglés *uniform resource identifier*— es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

El localizador de recursos uniforme (URL) hace referencia a recursos.

Normalmente estos recursos son accesibles en una red o sistema. **Los URI pueden ser localizador uniforme de recursos (URL)**, uniform resource name (URN), o ambos.

https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme
https://es.wikipedia.org/wiki/Localizador_de_recursos_uniforme

Reglas

Las URIs no deben implicar acciones y deben ser únicas

Por ejemplo, la URI *_facturas/234/editar_* sería incorrecta ya que tenemos el verbo editar en la misma.

Para el recurso factura con el identificador 234, la siguiente URI sería la correcta, independientemente de que vayamos a editarla, borrarla, consultarla o leer sólo uno de de sus conceptos: /facturas/234

Las URIs deben ser independientes del formato

Por ejemplo, la URI /facturas/234.pdf no sería una URI correcta, ya que estamos indicando la extensión pdf en la misma.

Filtrados y otras operaciones.

Para filtrar, ordenar, paginar o buscar información en un recurso, debemos hacer una consulta sobre la URI, utilizando parámetros HTTP en lugar de incluirlos en la misma.

Por ejemplo, la URI /facturas/orden/desc/fecha-desde/2007/pagina/2 sería incorrecta ya que el recurso de listado de facturas sería el mismo pero utilizaríamos una URI distinta para filtrarlo, ordenarlo o paginarlo.

La URI correcta en este caso sería:

/facturas?fecha-desde=2007&orden=DESC&pagina=2

Operaciones

Para manipular los recursos, HTTP nos brinda los siguientes métodos con los cuales debemos operar:

- **GET: Para consultar y leer recursos**
- **POST: Para crear recursos**
- **PUT: Para editar recursos**
- **DELETE: Para eliminar recursos.**
- **PATCH: Para editar partes concretas de un recurso.**

Por ejemplo para un recurso productos.

GET /productos Nos permite acceder al listado de facturas. headers.

POST /productos Nos permite crear una factura nueva. headers. Json en body.

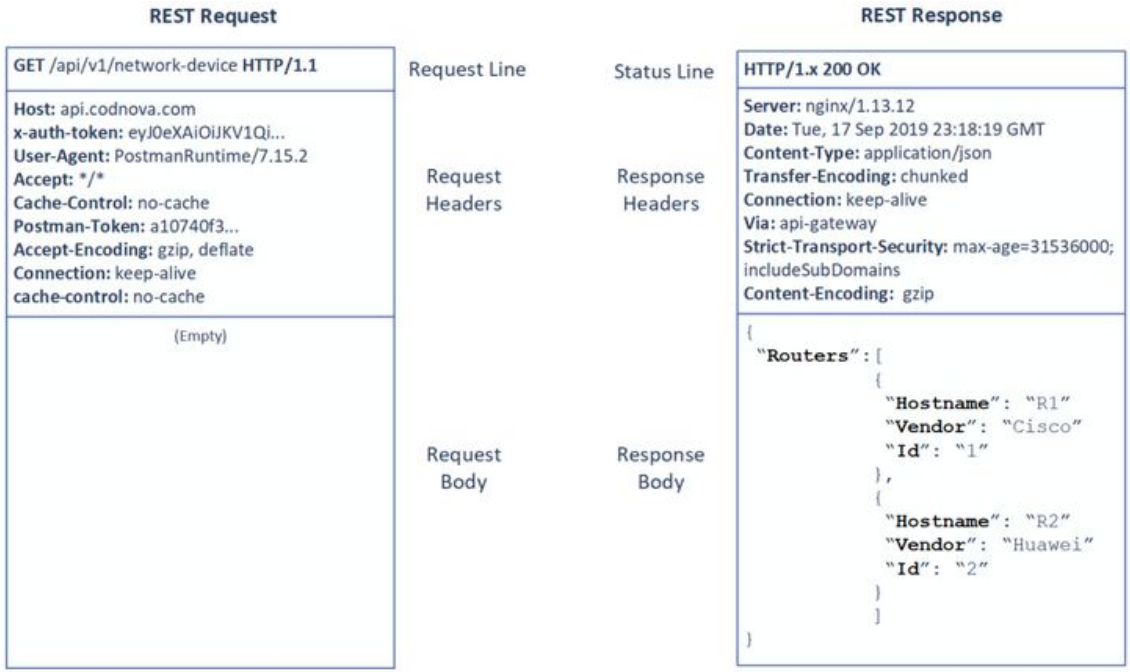
GET /productos/123 Nos permite acceder al detalle de una factura. headers.

PUT /productos/123 Nos permite editar la factura, sustituyendo la totalidad de la información anterior por la nueva.headers. Json en body.

DELETE /productos/123 Nos permite eliminar la factura. headers.

PATCH /productos/123 Nos permite modificar cierta información de la factura, como el número o la fecha de la misma. headers. Json en body.

Código de Error y de Estado



1. Información 1XX:
2. Éxito 2XX: 200 OK
3. Redirección 3XX:
4. Error del Cliente 4XX: 404 Not Found (page does not exist)
5. Error del Servidor 5XX: 500 Internal Server Error (generic error)

Detalle de Códigos de Error y de Estado:
https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

Código de Error y de Estado

Códigos de Estado de Éxito(2xx) : La solicitud fue procesada exitosamente.

Código de Estado	Mensaje del Estado	Significado
200	OK	La Solicitud fue procesada exitosamente
201	Created	Indica que uno o mas recursos fueron creados
202	Accepted	Indica que la solicitud ha sido aceptada para procesarse pero el proceso no ha terminado.
203	Non-Authoritative Information	Indica que la solicitud fue exitosa, pero que el Payload ha sido modificado por un Proxy.
204	No Content	Indica que la solicitud fue exitosa pero que el Cuerpo de HTTP no contiene una respuesta.

Códigos de Estado de Error en el Cliente (4xx): Un error ha ocurrido. El cliente es responsable del problema.

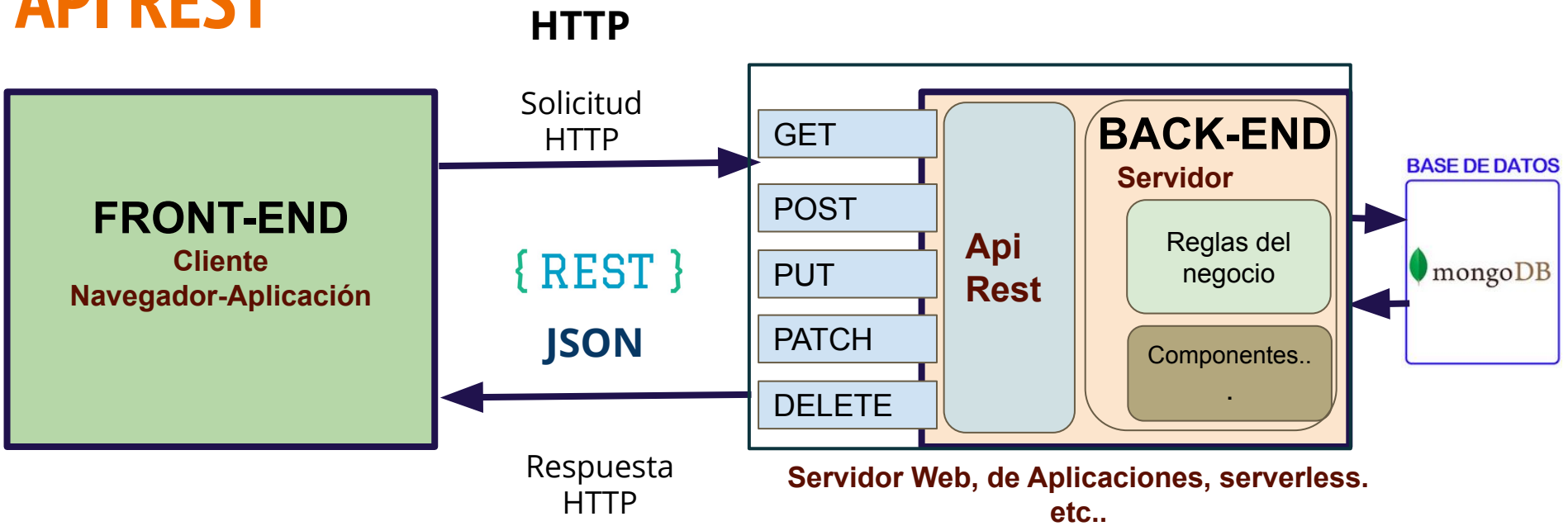
Código de Estado	Mensaje del Estado	Significado
400	Bad Request	Sintaxis Incorrecta en la Solicitud.
401	Unauthorized	Solicitud fallida, usuario no autenticado.
402	Insufficients Funds	Fondos insuficientes. Para APIs Monetizados.
403	Forbidden	Solicitud fallida, usuario no tiene autorización para acceder al recurso.
404	Not Found	El recurso no fue encontrado.
405	Method not Allowed	Método HTTP especificado en la solicitud no está permido.
406	Not Acceptable	El API no puede producir una respuesta con un MEdia-Type que el cliente pueda aceptar.
408	Request Timeout	Indica que el servidor no recibió una solicitud completa.
409	Conflict	Indica que hay un conflicto con el estado actual de l recurso.

Código de Error y de Estado

Códigos de Estado de Error en el Servidor(5xx) : Un error ha ocurrido. El Servidor o API es responsable del problema.

Código de Estado	Mensaje del Estado	Significado
500	Internal Server Error	Una condición inesperada ocurrió en el API y una excepción fue arrojada.
501	Not Implemented	La funcionalidad solicitada por el cliente no esta implementada todavía.
502	Bad Gateway	El servidor que actua como Gateway o Proxy obtuvo una respuesta inbalida del Backend
503	Service Unavailable	El servidor no puede procesar la solicitud o fue rechazada la conexión.

API REST



Solicitud HTTP. GET:
https://jsonplaceholder.typicode.com/albums

Respuesta HTTP. GET:


```
[
  {
    "userId": 1,
    "id": 1,
    "title": "quidem molestiae enim"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "sunt qui excepturi placeat culpa"
  },
```

Json

JSON (acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos.

JSON es solo un formato de datos. Una colección de pares de nombre/valor

Requiere usar comillas dobles para las cadenas y los nombres de propiedades.

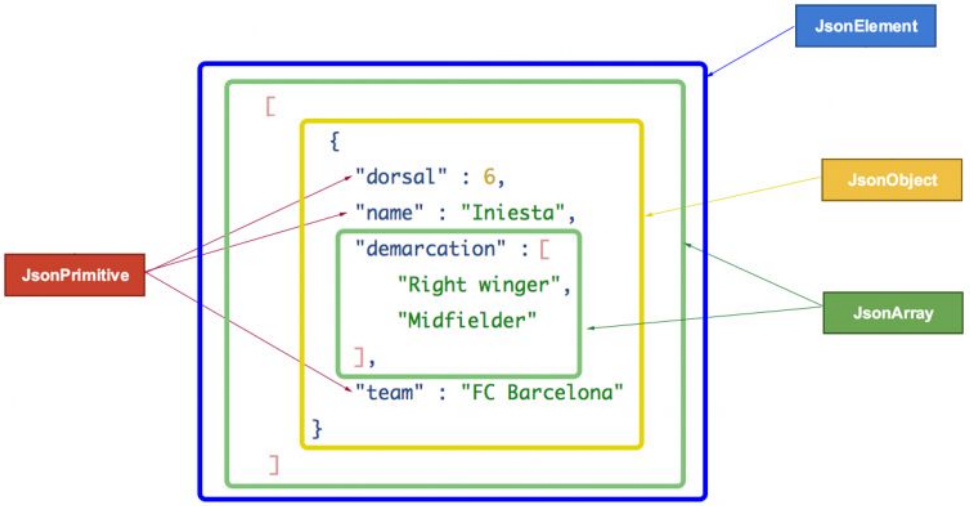
La sintaxis de JSON se resumen en:

Cada dato es un par nombre/valor.

Los datos se separan entre sí por comas.

Un objeto es un conjunto de datos, que se define entre llaves { }

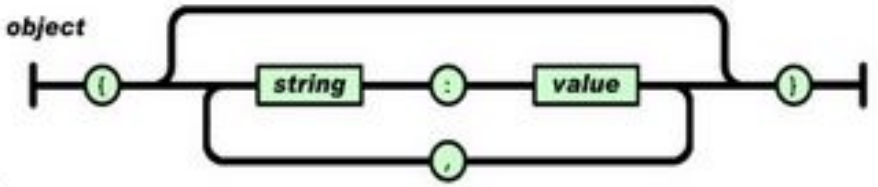
Los valores de los datos puede contener arrays, que se definen entre corchetes [ ]

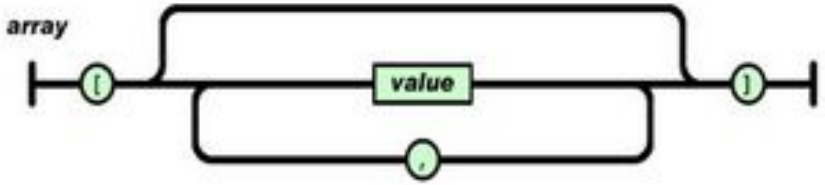


JSON

Sintaxis

- Objeto: Conjunto desordenado de pares nombre/valor
- Arreglo: colección de






```
{
  "resourceType": "Organization",
  "id": "1704632",
  "identifier": [ {
    "system": "www.nationalorgidentifier.gov",
    "value": "456789"
  } ],
  "name": "LITTLE CLINIC",
  "address": [ {
    "line": [ "2000 CLINIC DRIVE" ],
    "city": "ANN ARBOR",
    "state": "MI",
    "country": "US"
  } ]
}
```

Un dato es un par '*nombre : valor*' separado por los dos puntos ':'

http://hapi.fhir.org/baseR4/Organization/1704632/_history/1

Se puede usar:
https://jsoneditoronline.org

Fuente: http://www.json.org/json-es.html

API

Las APIs son interfaces que permiten la comunicación entre dos aplicaciones de software siguiendo cierto conjunto de reglas.

REST

Es una lógica de restricciones y recomendaciones bajo la cual se puede construir una api. Un estilo de arquitectura

REST ful API

Es una API que implementa la lógica de restricciones y recomendaciones de REST

Funcionan estrictamente bajo una arquitectura cliente servidor utilizando http como protocolo de comunicación

El Cliente (app Flutter) envía solicitudes. El servidor las recibe, realiza acciones y retorna respuesta

El Cliente envía solicitud con un método http (GET, POST, PUT, PATCH, DELETE) con una URL única
{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}
y la información necesaria que requiere servidor para satisfacer el requerimiento.

Trabaja con recursos representado en un formato, por ejemplo JSON o XML.

Se intercambian JSON. También otros formatos como XML

El servidor recibe la solicitud y procesa la URL. Realiza una acción y devuelve una respuesta. Por ejemplo retorna un código:200, con un body y header
Comunicaciones sin estado. Cada petición al servidor es tratada de forma independiente.

Introducción a la Programación Multiplataforma para Web y Mobile

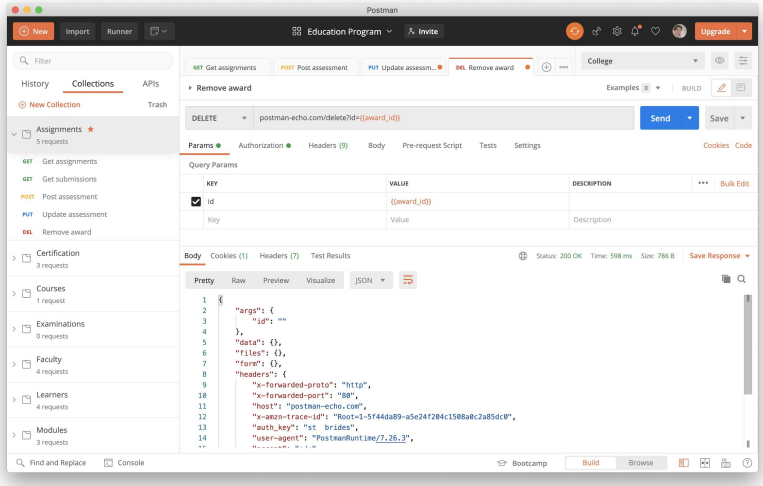
Firebase Realtime Database

Mgter. Ing. Carlos Alejandro Martinez

Postman

Postman es un cliente para poder probar y aprender sobre la interacción con APIs REST.

Descargar e Instalar Postman: https://www.postman.com/downloads/



Firebase. Realtime Database

- **Realtime Database es un servicio de base de datos en la nube**
- **Base de datos no relacional (NoSQL) totalmente administrada**
- **Los datos se almacenan en formato JSON.**
- **Está construida bajo la infraestructura de Google**

Los desarrolladores pueden construir aplicaciones rápidamente sin preocuparse demasiado por la escalabilidad y sólo se centran en la construcción de las mejores aplicaciones.

Fuente: https://firebase.google.com/docs/database?hl=es-419

Realtime Database. Estructura de Datos

Todos los datos de Firebase Realtime Database se almacenan como objetos JSON.

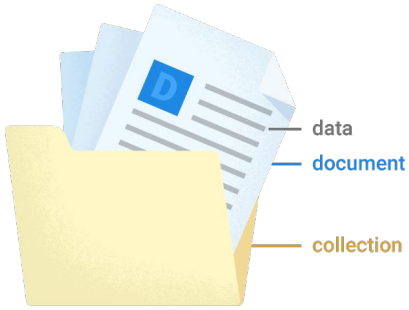
La base de datos se puede conceptualizar como un árbol JSON alojado en la nube.

A diferencia de una base de datos de SQL, no hay tablas ni registros.

Cuando se agregan datos al árbol JSON, estos se convierten en un nodo de la estructura JSON existente con una clave asociada.

Fuente: https://firebase.google.com/docs/database?hl=es-419

Modelo de Datos



Cada documento contiene un conjunto de pares clave-valor.

Los documentos pertenecen a colecciones, que simplemente son contenedores de documentos.

Fuente: https://firebase.google.com/docs/firestore/data-model?hl=es-419

Realtime Database. Documentación

Indexa tus datos y Filtrar Datos

https://firebase.google.com/docs/database/security/indexing-data?hl=es

https://firebase.google.com/docs/database/rest/retrieve-data#section-rest-filtering

Estructura la base de datos

https://firebase.google.com/docs/database/web/structure-data?hl=es-419

API de REST de Firebase Database

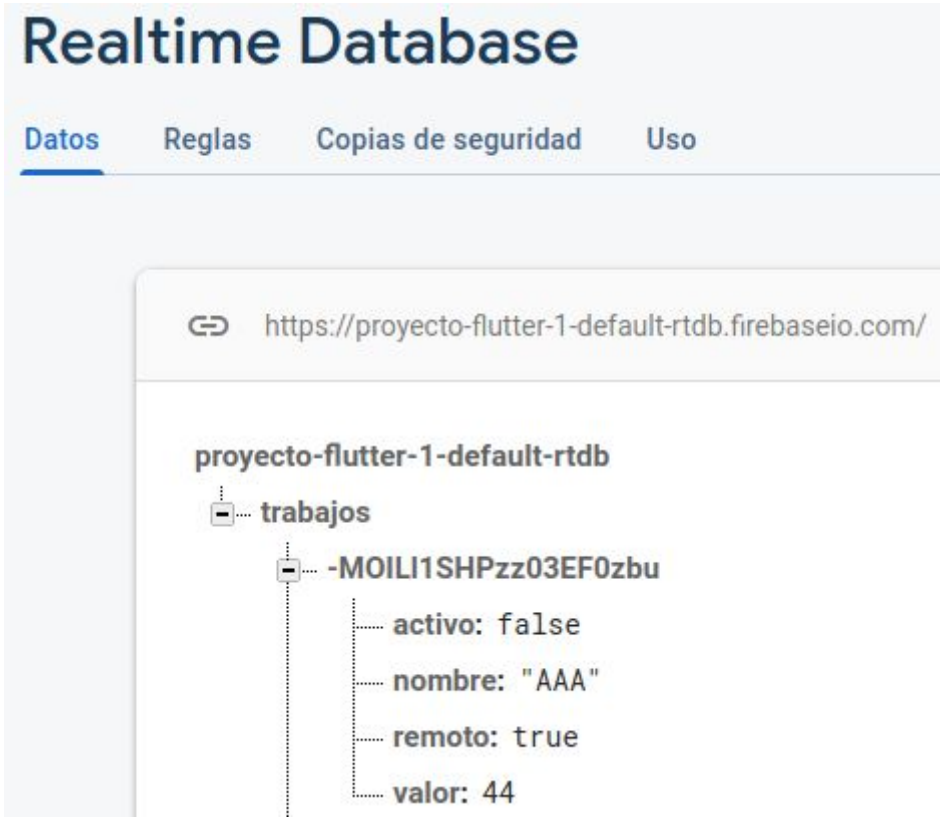
https://firebase.google.com/docs/reference/rest/database

Ejercicio

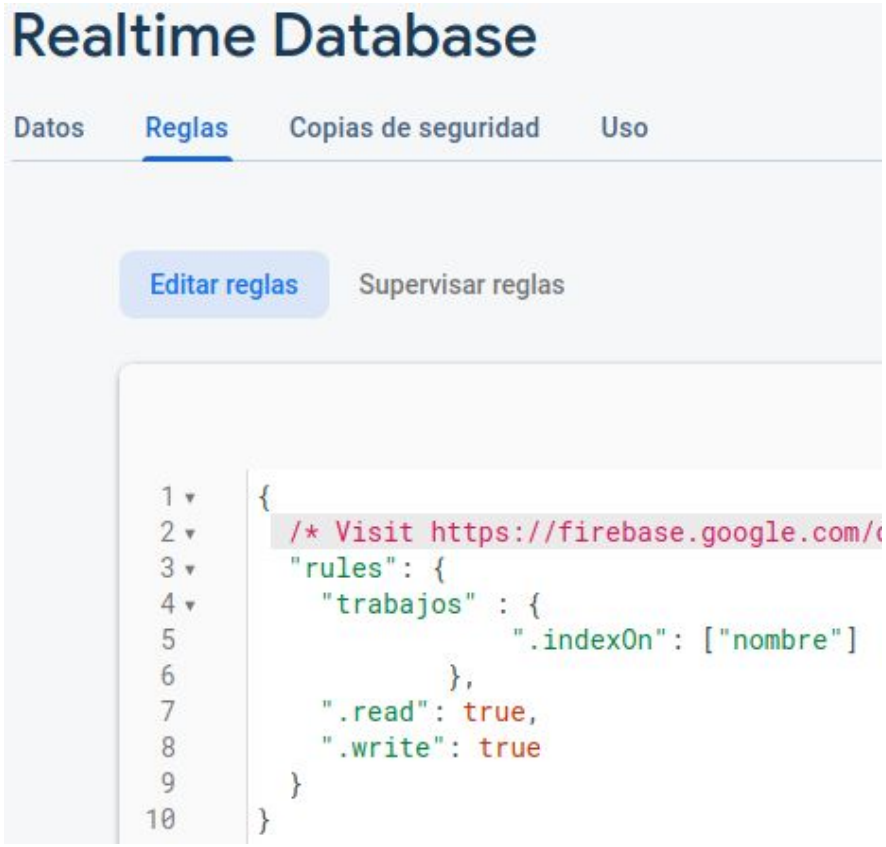
Crear una Firebase Realtime Database

Probar con Postman la API de Realtime Database

API Rest de Firebase Database Realtime



API Rest de Firebase Database Realtime




```
{
  "rules": {
      "trabajos" : {
          ".indexOn": ["nombre"]
      },
      ".read": true,
      ".write": true
  }
}
```

¿Cómo crear una clave auto-incrementada en la Realtime Database(Firebase)?

Realtime Database no tiene claves de autoincremento, como:1,2,35., Ya que éstas no funcionan bien en sistemas masivos multiusuario donde los clientes pueden estar desconectados por períodos prolongados.

Es decir, las claves de autoincremento, como:1,2,3,5, limitan la escalabilidad de un sistema distribuido multiusuario que también necesita lidiar con la falta de conectividad.

En cambio, la Firebase tiene su propio tipo de claves autogeneradas. Estas claves están ordenadas y son secuenciales. Pero además, pueden ser calculadas del lado del cliente, incluso cuando el cliente no está conectado a los servidores de Realtime Database