2.1 - Tecnologías de Servicios Web

Tema 7 – Tecnologías de comunicación

Universidad Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

code URJC

Tecnologías de comunicación

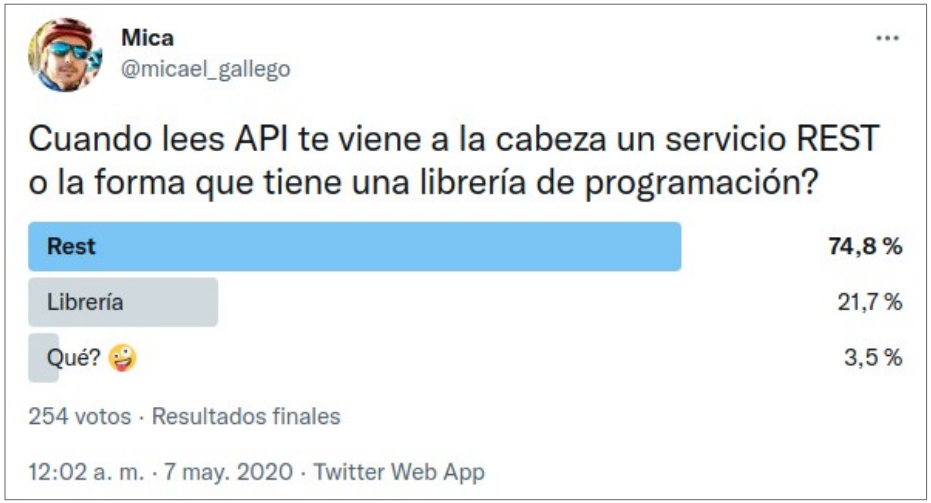
- Los **servicios web** son aplicaciones que ofrecen sus funcionalidades de forma **remota (vía red)**
- **Se comunican con:**
  - **Clientes:** Aplicaciones ejecutadas en navegadores web, aplicaciones en dispositivos móviles, aplicaciones de escritorio
  - **Otros servicios web:** Que forman parte de la misma organización o de otras organizaciones

Tecnologías de comunicación

- Existen muchas **tecnologías de comunicación diferentes**
- Cada una es más adecuada para un **contexto** determinado
  - Cuando la aplicación cliente está en una **navegador web** hay que tener en cuenta sus **limitaciones**
  - Cuando hay que ofrecer funcionalidades a servicios de otras organizaciones hay que ser **interoperable**
  - Cuando se comunican servicios web de la misma organización se puede tener un mayor control y preferir la **eficiencia**

Tecnologías de comunicación

- **API (de un servicios web)**
  - *Application Programming Interface*
  - Antes se usaba para librerías, ahora representa el interfaz de red (generalmente de tipo REST)



https://twitter.com/micael_gallego/status/1258155255071678466

Tecnologías de comunicación

- **API (de un servicios web)**
  - Conjunto de **operaciones** y **tipos de datos** que ofrece un servicio web
  - Dependiendo de la tecnología existen formatos estándar de **definición de esa API**: OpenAPI, AsyncAPI, Proto, GraphQL Schema…
  - A los lenguajes de definición de APIs se les conocía como **IDL** (*Interface Definition Language*)

Tecnologías de comunicación

- **API (de un servicios web)**
  - Enfoque de **API primero** *(API-first approach)*
  - Cuando se implementa un servicio web se suele recomendar **primero diseñar su API** en colaboración con los clientes y luego implementar esa API
  - Es más fácil identificar requisitos y casos de uso

https://swagger.io/resources/articles/adopting-an-api-first-approach/

Tecnologías de comunicación

- **Evolución de las APIs**
  - No hay que hacer cambios incompatibles sin seguir ofreciendo la API previa
  - Los clientes deben ser conscientes de la versión de la API que usan y su compatibilidad (*semantic versioning)*
    - URL, headers, mensajes…
  - Procura hacer cambios compatibles
    - Añadir atributos a una respuesta
    - Añadir parámetros opcionales a una petición
    - Añadir operaciones nuevas

Tecnologías de comunicación

- **Formatos de mensajes**
  - Para que una tecnología de comunicación sea **interoperable** (que permita comunicación entre diferentes lenguajes de programación y sistemas) es necesario que se pueda intercambiar **información estructurada**
  - Muchas tecnologías permiten elegir el **formato de mensaje**
    - Formato **textual**: JSON, XML…
    - Formato **binario**: Protocol Buffers, Avro, Thrift…

Tecnologías de comunicación

- **Tipos de comunicación**
  - **Interacciones *one-to-one***
    - *Petición-respuesta (request-response)*
    - Notificaciones sin respuesta (*one-way*)
  - **Interacciones *one-to-many***
    - *Publish-subscribe:* Se publica información, no se espera respuesta. Puede no haber suscriptores
    - *Publish-async responses* : Se espera un tiempo por las respuestas

Tecnologías de comunicación

- Tecnologías más usadas en la actualidad

WebSockets

Tecnologías de comunicación

- *Remote Procedure Invocation* (RPI)

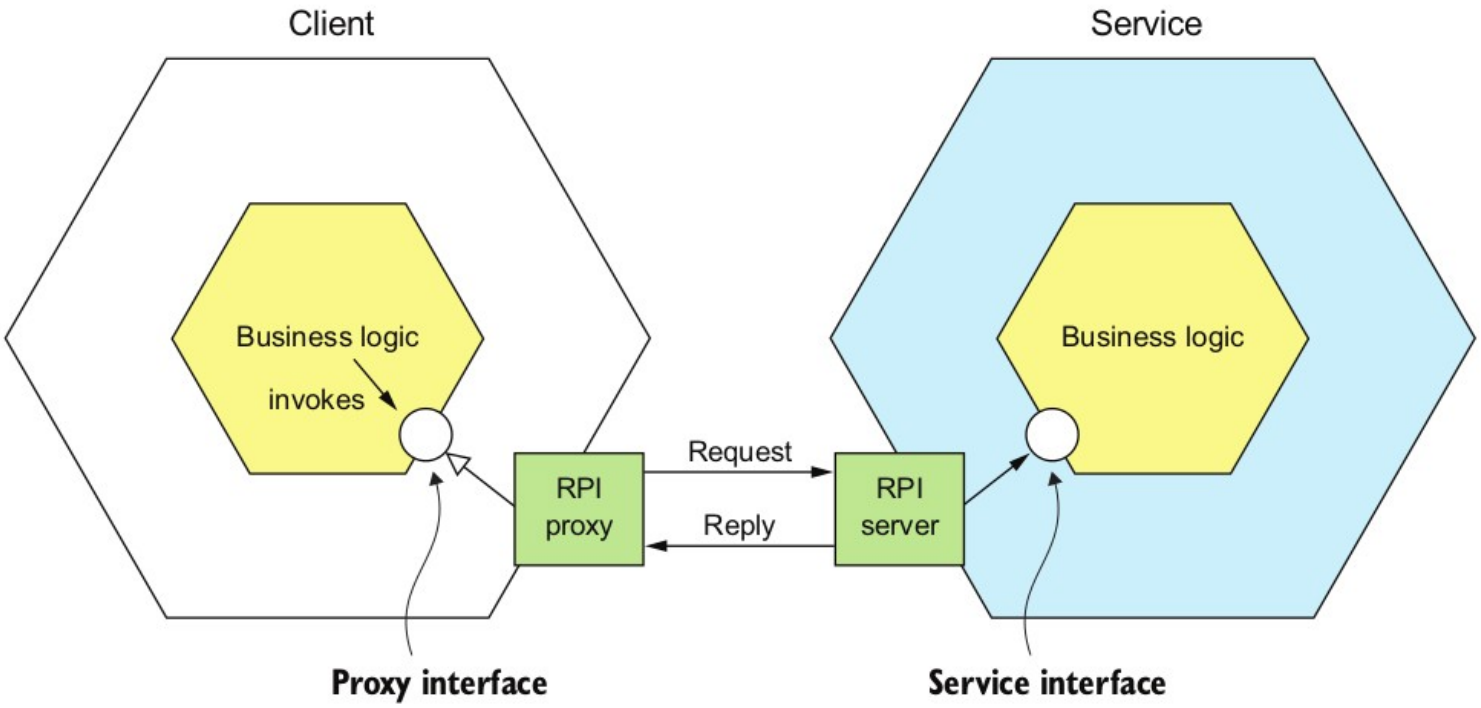


Figure 3.1 The client's business logic invokes an interface that is implemented by an *RPI proxy* adapter class. The *RPI proxy class* makes a request to the service. The *RPI server* adapter class handles the request by invoking the service's business logic.

Tecnologías de comunicación

- *Remote Procedure Invocation* (RPI)

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet.

https://en.wikipedia.org/wiki/Representational_state_transfer

REST API

- **Características**
  - Basado en http 1.1
  - Comunicación
    - Entre servidores
    - Navegador web / servidor

REST API

- **Características**
  - Tipo de comunicación:
    - Petición / Respuesta de cliente a servidor
    - No soporta streaming / eventos
    - Longpolling ha sido una técnica para sortear estas limitaciones

REST API

- **Definición de la API**
  - Inicialmente no se definía formalmente
  - Ahora se usa **OpenAPI** con **JSON-Schema**

https://www.openapis.org/

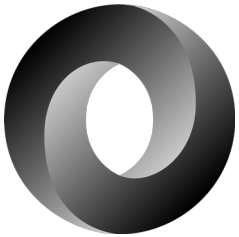
https://json-schema.org/

REST API

- **Definición de la API**
  - Se puede usar con diferentes niveles de madurez
  - En los más bajos, se convierte en un **protocolo de transporte**
  - En los más alto, es un protocolo de alto nivel de **CRUD para recursos**

REST API

- **Formato de mensajes**
  - Inicialmente se usaba **XML**
  - Actualmente se usa bastante **JSON**
  - Se pueden usar otros formatos binarios, por ejemplo **protobuf** de Google



https://www.json.org/

https://www.w3.org/XML/

https://developers.google.com/protocol-buffers/

REST API

- **Interoperabilidad**
  - **Muy interoperable**
  - Basado en protocolo estándar (http 1.1)
  - Formato mayoritario basado en texto "estándar" (JSON)
  - No es necesaria la especificación formal ni la generación de código para que se pueda usar

REST API

- **Clientes Interactivos**
  - Comando curl

https://insomnia.rest/

https://www.postman.com/

REST API

- **Ejemplos Java con SpringBoot**
  - Servicio expone API REST (**rest-ejem1-server**)
  - Cliente API REST con JsonObject (**rest-ejem2-client-json**)
  - Cliente API REST con POJOs (**rest-ejem3-client-pojo**)
  - Cliente API REST con Feign (**rest-ejem4-client-feign**)

https://github.com/MasterCloudApps/2.1.Tecnologias-de-servicios-web/tree/master/tema7-comunicacion

code URJC

A high performance, open-source universal RPC framework

https://gRPC.io/

gRPC

- **Características**
  - Basado en **http 2 (Cambio importante)**
  - Comunicación
    - **Entre servidores**
    - Existen adaptadores para comunicación navegador / servidor, pero no es habitual

gRPC

- **Características**
  - Tipo de comunicación:
    - Petición / Respuesta
    - **Streaming / Eventos** (como respuesta)
    - Cliente a servidor

gRPC

- **Definición de la API**
  - Obligatoria con ficheros **.proto**
  - En lenguajes compilados (Java) es obligatoria la **generación de código** tanto para clientes como servidor
  - En Node el código se puede generar de forma dinámica

gRPC

- **Definición de la API**
  - Las operaciones se pueden definir en la API (en REST están predefinidas a los métodos de http)
  - Se recomienda seguir el mismo enfoque que REST (**gestión de recursos**)
  - Eso permite la conversión automática entre gRPC y REST

https://cloud.google.com/apis/design/

gRPC

- **Formato de mensajes**
  - Binario con **protobuf** de Google

Protocol Buffers

https://developers.google.com/protocol-buffers/

gRPC

- **Objetivos**
  - gRPC nace para mejorar las APIs REST
    - Más eficiencia (binario)
    - Eventos / Streams
    - Http 2 (mejor protocolo de bajo nivel)

gRPC

- **Interoperabilidad**
  - Google ha creado herramientas y librerías para que se pueda usar en muchos lenguajes
    - Librerías cliente/servidor
    - Generación de código en algunos lenguajes
    - Librerías nativas si no hay buen soporte http 2
    - Adaptadores para clientes en browser

REST API

- **Clientes Interactivos**
  - Grpcurl: https://github.com/fullstorydev/grpcurl

https://insomnia.rest/

gRPC

- **Implementación en Java**

  - gRPC Nativo (Sin SpringBoot)

    https://gRPC.io/docs/tutorials/basic/java/

  - Integrado en SpringBoot

    https://github.com/LogNet/gRPC-spring-boot-starter

    (1.300 *)

    https://github.com/yidongnan/gRPC-spring-boot-starter

    (1.200 *)

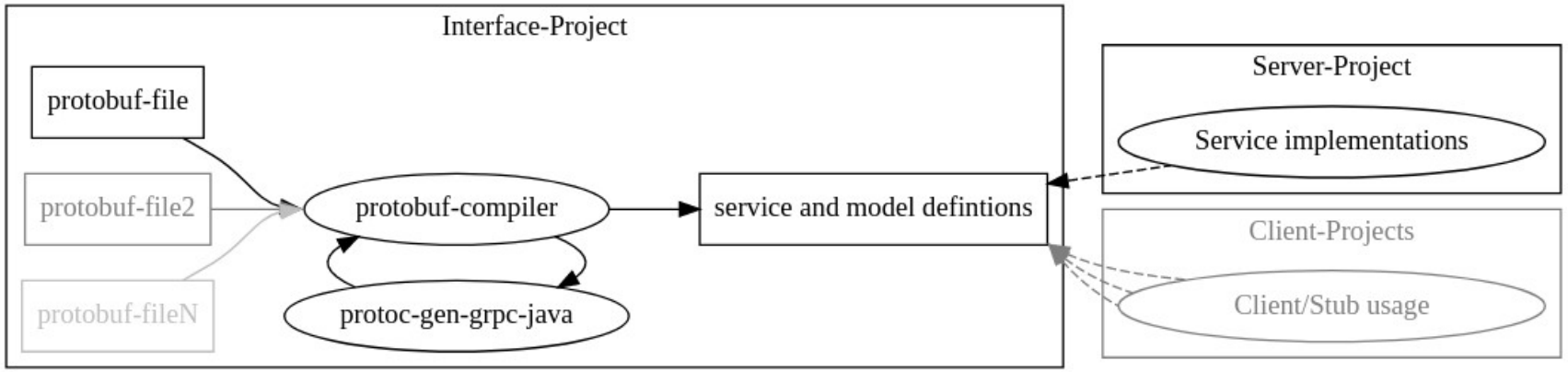
gRPC

- **Ejemplos Java**
  - Java plano (sin Spring)
    - grpc-ejem1
  - SpringBoot
    - grpc-ejem2-client
    - grpc-ejem2-server
    - grpc-ejem2-interface (código generado desde .proto)

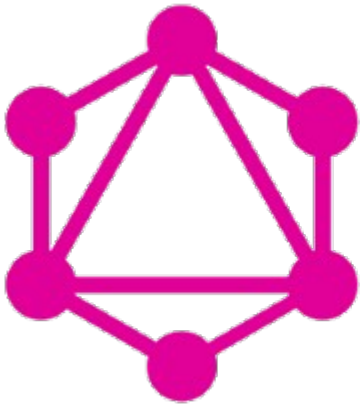
https://github.com/MasterCloudApps/2.1.Tecnologias-de-servicios-web/tree/master/tema7-comunicacion

gRPC

- Ejemplos Java
  - SpringBoot



GraphQL



GraphQL

https://graphql.org/

GraphQL

- Es una tecnología de comunicación que se ha diseñado con el objetivo de **mejorar algunas limitaciones de REST**
- Su principal característica es que en las consultas se especifican los atributos de las entidades (en REST los atributos los decide el servidor) **(query lenguage)**
- Incluso se especifican los atributos de las entidades relacionadas **(graph)**

GraphQL

- Petición

```
{
  human(id: "1000") {
    name
    height
  }
}
```

- Respuesta

```
{
    "human": {
      "name": "Luke Skywalker",
      "height": 1.72
    }
}
```

GraphQL

Petición

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

Respuesta

```
{
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
}
```

GraphQL

Petición

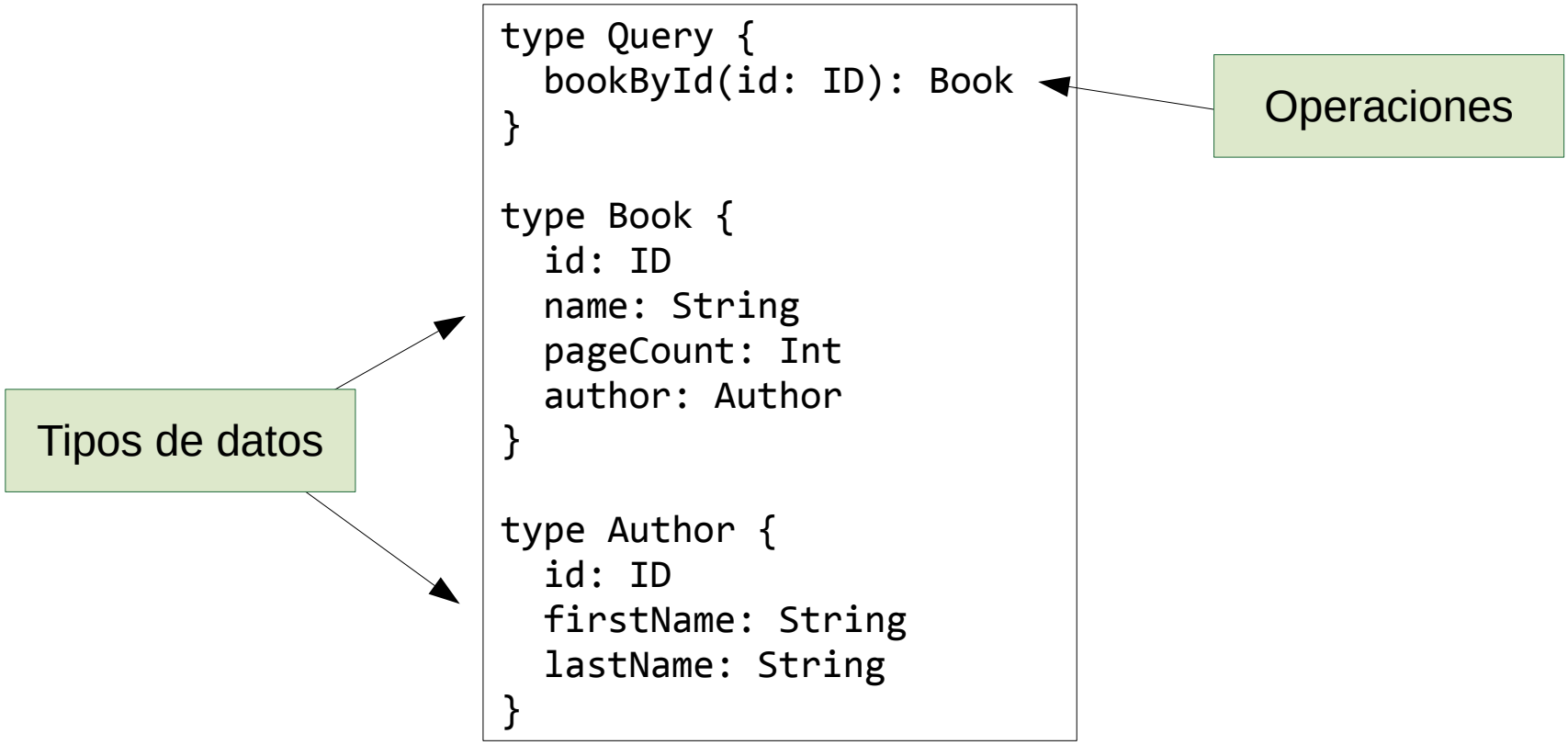
```
{
  bookById(id: "book-1"){
    id
    name
    pageCount
    author {
      firstName
      lastName
    }
  }
}
```

Respuesta

```
{
  "bookById":
  {
    "id":"book-1",
    "name":"Harry Potter...",
    "pageCount":223,
    "author": {
      "firstName":"Joanne",
      "lastName":"Rowling"
    }
  }
}
```

GraphQL

Definición de la API (esquema)



GraphQL

- **Características**
  - **Similitudes con REST**
    - Basado en http
    - Respuesta en JSON
    - Permite hacer consultas de recursos

GraphQL

- **Características**
  - **Diferencias con REST**
    - Todas las peticiones son por POST a una misma URL (http://server/graphql)
      - Algunos servidores permiten consultas por GET
    - Los recursos no se especifican en la URL, se especifican en el body de la petición POST con un lenguaje propio
    - No se usan los códigos de estado http (los errores van en la respuesta de la petición)

GraphQL

- **Características**
  - **Diferencias con REST**
    - Se pueden definir operaciones propias (como gRPC)
    - Es obligatorio definir un esquema de la API (como gRPC)
    - Soporte nativo de eventos del servidor (suscripciones) implementadas con websockets

GraphQL

- **GraphQL en GitHub**
  - GitHub cambió de su API REST a una API GraphQL en 2016
  - Motivación

    https://github.blog/2016-09-14-the-github-graphql-api/

  - Documentación

    https://docs.github.com/en/graphql

GraphQL

- **Clientes Interactivos**
  - Comando Curl

```
curl -X POST \
-H "Content-Type: application/json" \
-d '{"query": "{ hello }"}' \
http://localhost:4000/graphql
```

```
{"data":{"hello":"Hello world!"}}
```

GraphQL

- Clientes Interactivos

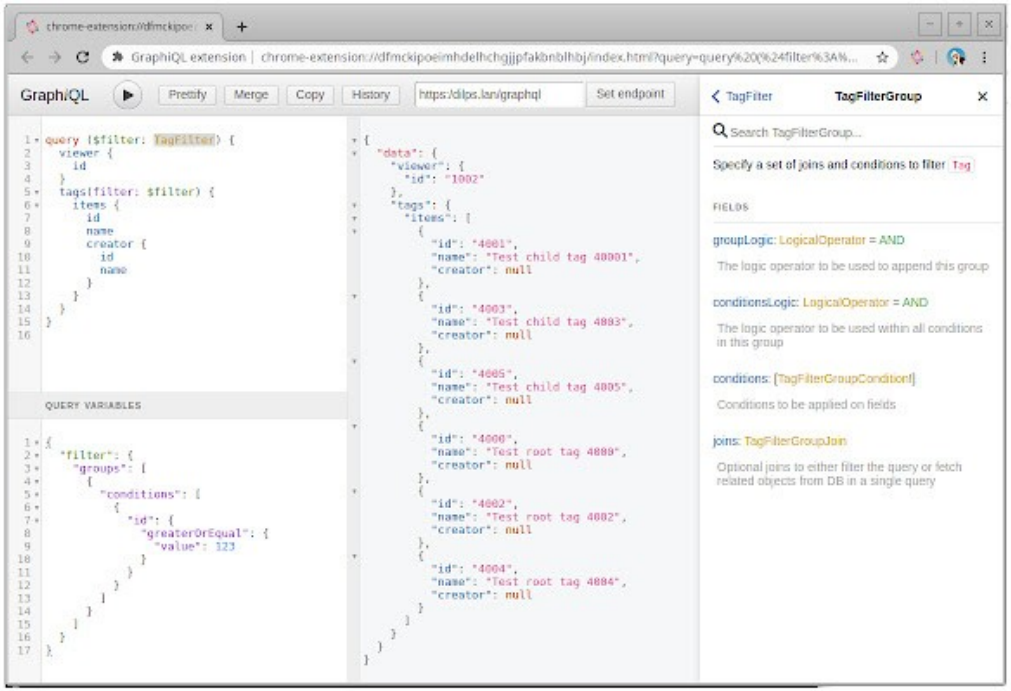
https://insomnia.rest/

https://www.postman.com/

GraphQL

- **Clientes Interactivos**

  Graph*i*QL integrado en el servidor



http://server/graphiql http://server/graphql

GraphQL

- **Implementación en Java**
  - Graphql-java
    - https://www.graphql-java.com/
    - Independiente del framework (no integrado con SpringBoot)
  - Graphql-java-spring
    - https://github.com/graphql-java/graphql-java-spring
    - Integración no oficial de graphql en Spring
  - GraphQL Java Kickstart
    - https://www.graphql-java-kickstart.com/
    - Starters no oficiales basados en graphql-java

GraphQL

- **Implementación en Java**
  - Spring-graphql
    - https://spring.io/projects/spring-graphql
    - Integración oficial de GraphQL en Spring basada en graphql-java (todavía en desarrollo)
  - DSG Framework
    - https://netflix.github.io/dgs/
    - Integración de GraphQL en Spring por Netflix

GraphQL

- **Ejemplos en Java**
  - Servidor basado en Spring-graphql (oficial)
    - graphql-ejem1-server
  - Cliente basado en GraphQL Java Kickstart
    - graphql-ejem1-client

Comparación REST vs gRPC vs GraphQL

- Estudio comparativo con ejemplo en Node.js

https://github.com/MasterCloudApps-Projects/REST-gRPC-GraphQL

Comparación REST vs gRPC vs GraphQL

Usage	REST	GraphQL	gRPC
Contract	HATEOAS or OpenAPI	GraphQL Schema Language: operations	Protocol Buffers: rpc
Schema definition	Resource oriented. HTTP response headers, Media Type and JSON Schema	Graph oriented. GraphQL Schema Language	Resource and Action oriented. Protocol Buffers: messages
Standard methods	GET, POST, PUT, PATCH, DELETE	query and mutation	Through rpc operations
Get	GET	query	Get rpc operation
Get (representation)	Content Negotiation	✘ Only JSON	✘ only one. Default: Protocol Buffers
Get (custom)	Sparse fieldsets. Embedded resources	Native support	FieldMask

Comparación REST vs gRPC vs GraphQL

Usage	REST	GraphQL	gRPC
List	GET. Custom pagination, sorting and filtering	query. Standard pagination and sorting	List and Search rpc operations
Create	POST or PUT	mutation	Create rpc operation
Update	PUT	mutation	Update rpc operation (unrecommended)
Partial update	PATCH	✘ Workarounds	Update rpc operation with FieldMask
Delete	DELETE	mutation	Delete rpc operation
Custom methods	HATEOAS or POST	pure functions: query, other: mutation	Custom rpc operation

Comparación REST vs gRPC vs GraphQL

Usage	REST	GraphQL	gRPC
Long-requests	Resource operation		Interface Operation
Error handling	Native in HTTP. Extensible	errors property. Extensible	Standard errors. Google Error Model
Security	HTTP: Bearer, OAuth, CORS, API Keys	HTTP: Bearer, OAuth, CORS, API Keys	TLS, ALTS, token-based (Google), custom
Subscriptions	Unsupported. WebHook and HTTP streaming	subscription	HTTP/2 streaming
Caching	HTTP, application and local cache	GET, application and local cache	Application and local cache
Discoverability	HATEOAS and OPTIONS or OpenAPI	Native introspection	✘ autogenerated client code