Enterprise Programming 2

Lesson 07: SOAP and GraphQL

Bogdan Marculescu

Goals

- Get a high level overview of *SOAP* web services
- Understand how to use and develop *GraphQL* web services
- Understand the differences between *REST*, *SOAP* and *GraphQL*

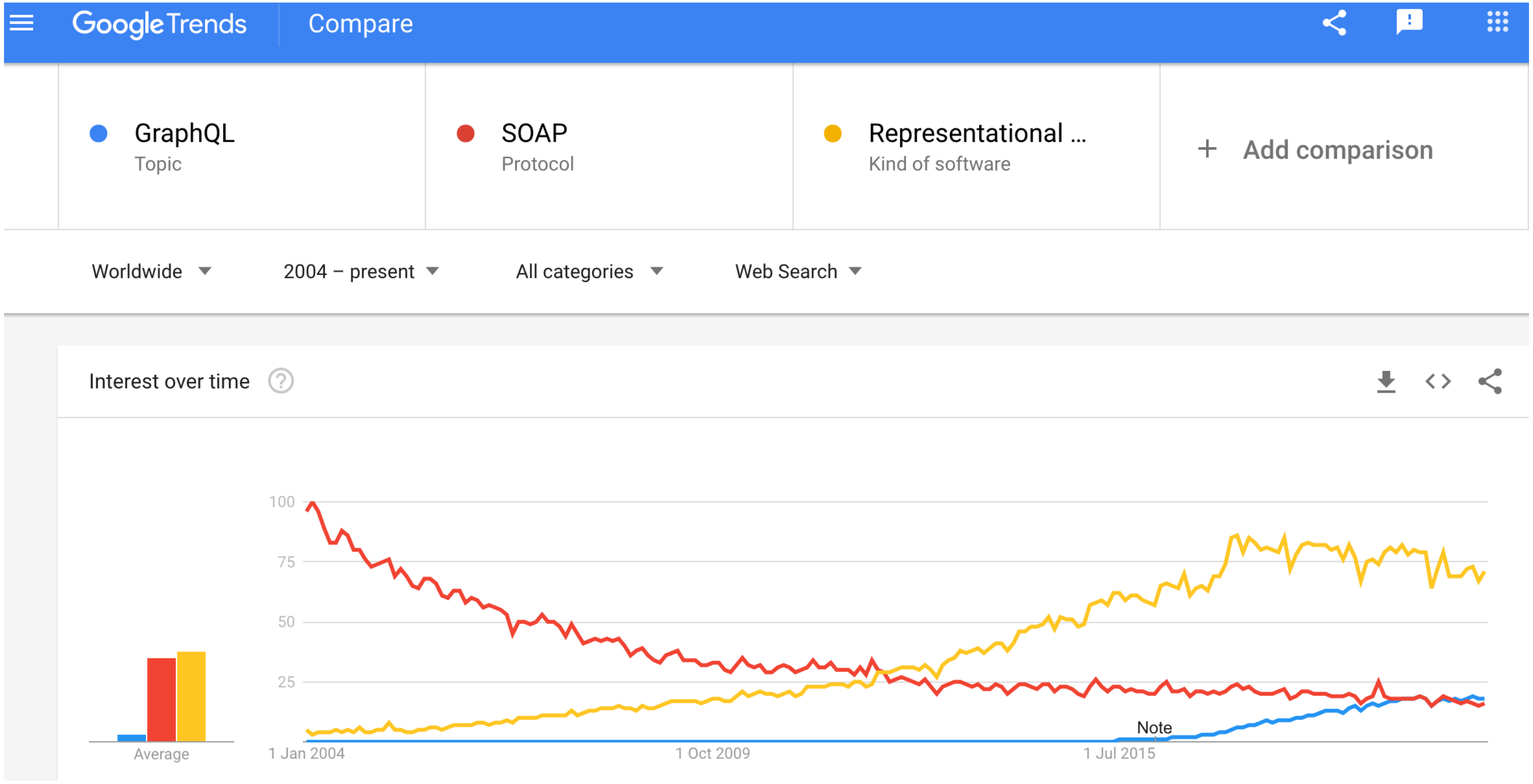
SOAP

Simple Object Access Protocol (SOAP)

- An application-level communication **protocol**
  - REST is not a protocol
- Typically used for Web Services communicating over HTTP
- In contrast to REST, SOAP can be used outside of HTTP
- SOAP is XML based
- Started in 1998, by Microsoft

SOAP, REST or GraphQL?

Google Trends, late 2020



● GraphQL
Topic

● SOAP
Protocol

● Representational …
Kind of software

+ Add comparison

Worldwide
2004 – present
All categories
Web Search

Interest over time

Status of SOAP

- Major way of developing web services in 2000s decade
- But now mainly substituted by REST
  - With GraphQL (2015) being the new kid on the block
- But there are still *a lot* of SOAP Web Services out there
- Important to understand how to use existing SOAP services, but not so much how to implement a new one
- Also important to understand *why* REST won over SOAP

SOAP Over HTTP

- One *single* HTTP endpoint, handling POST
- HTTP XML payload not only contains data, but also the instructions of what to do with it
- Eg, in following, asking NASDAQ to list current market centers

```
<?xml version="1.0" ?>
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ListMarketCenters
         xmlns="http://ws.nasdaqdod.com/services/v1/">
      </ListMarketCenters>
   </S:Body>
</S:Envelope>
```

SOAP XML

- SOAP messages are in XML
- Specific tags, like <Envelope> and <Body>
- Actual payload of SOAP will be in XML inside <Body>
- Server needs to analyze what defined in the XML to determine how to respond, and which actions to take
  - eg create a new resource, delete it, or just fetch it

SOAP Problems

- Quite verbose protocol
  - And so very tedious to use manually
  - Need to use client-libraries specific for the target server
- Stuck with XML
- Coupling between server and client
  - Not necessarily a bad thing inside a distributed enterprise application, but it hurts maintainability
  - REST+JSON is more flexible

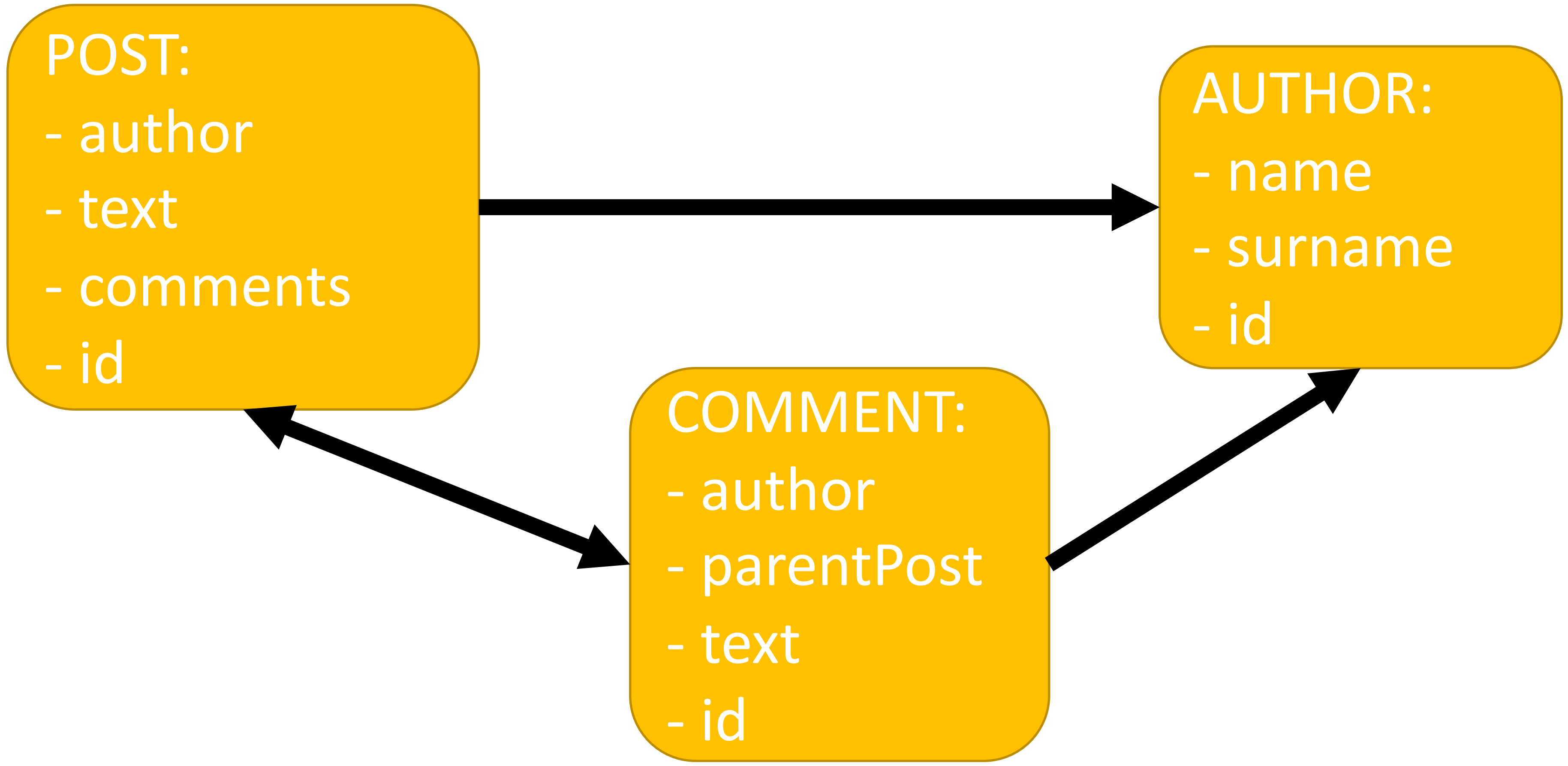
GraphQL

Graph Query Language (GraphQL)

- Made by Facebook
  - in 2012, but publically released in 2015
- Actual *protocol* used to define how an API can be queried with a specific query language
- A GraphQL Web Service will typically run on HTTP, where the GraphQL queries are sent as part of the HTTP messages
- GraphQL can be used outside of HTTP

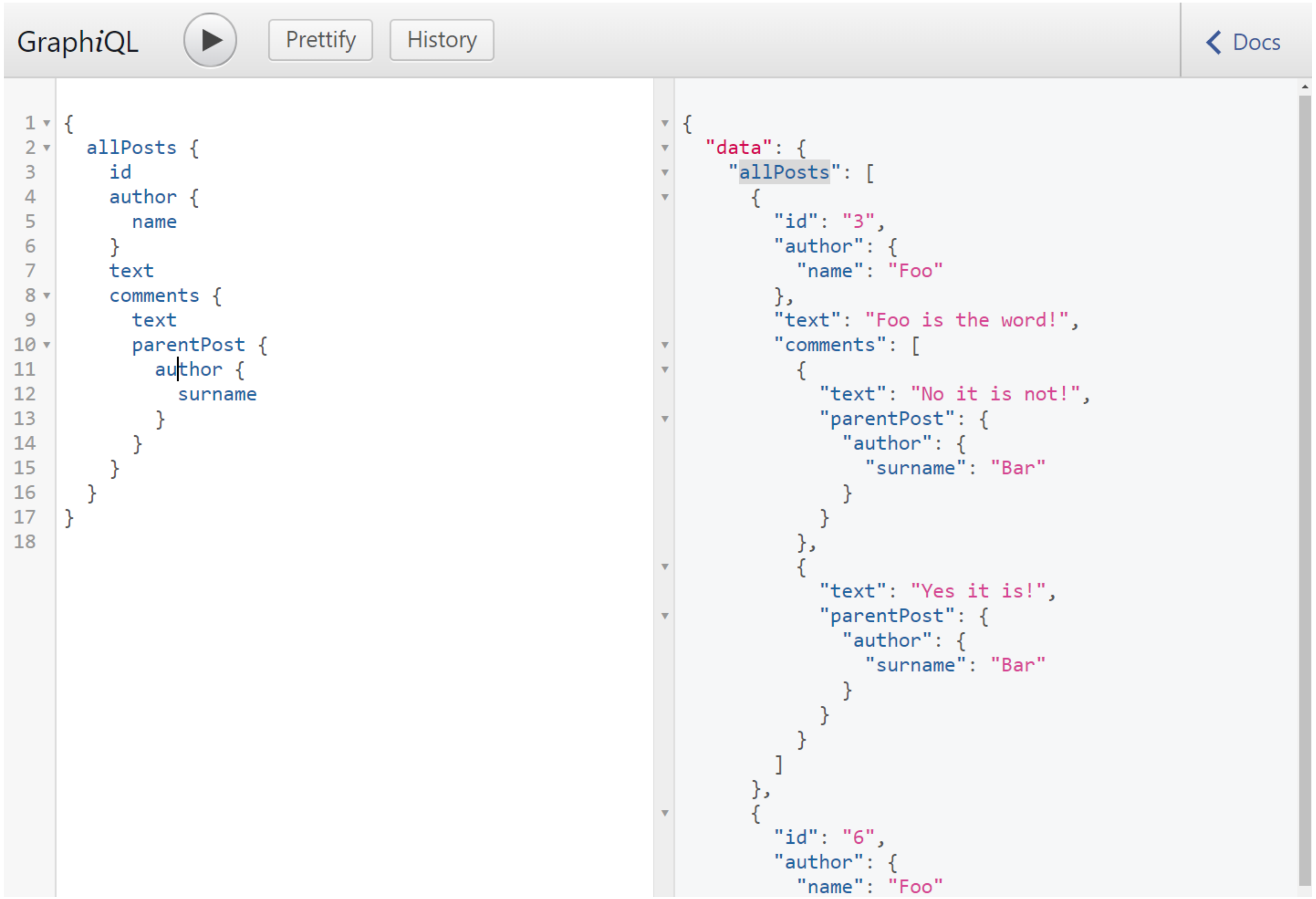
API Data as a Directed Graph

- Eg, forum posts with comments, and author info
- On backend, could be saved in a SQL database



GraphQL Queries

- Start from a method that returns elements of one of the nodes in the data graph
- Exactly specify which fields in the node to retrieve
  - eg just surname and no name in Author
- Can follow links on the graph to retrieve other connected data
- On such links, still need to specify the fields to retrieve
- From links can follow other links
  - being a graph and not a tree, same data could be accessed several times



```
{
  allPosts {
    id
    author {
      name
    }
    text
    comments {
      text
      parentPost {
        author {
          surname
        }
      }
    }
  }
}
```

```
{
  "data": {
    "allPosts": [
      {
        "id": "3",
        "author": {
          "name": "Foo"
        },
        "text": "Foo is the word!",
        "comments": [
          {
            "text": "No it is not!",
            "parentPost": {
              "author": {
                "surname": "Bar"
              }
            }
          },
          {
            "text": "Yes it is!",
            "parentPost": {
              "author": {
                "surname": "Bar"
              }
            }
          }
        ]
      },
      {
        "id": "6",
        "author": {
          "name": "Foo"
```

GraphiQL is a tool we can use to visualize and debug queries

Structure of a Query

```
{
 allPosts {
  id
  author { name }
  text
  comments {
       text
       parentPost { author { surname} }
  }
 }
}
```

- Need an entry point
  - eg, *allPosts*
- Specify which fields to retrieve from that type
  - e.g., *id, author, text, comments*
- When field is reference to another type in graph, need to specify its fields
  - eg, *name* for *author*

Cont.

```
{
 allPosts {
   id
   author { name }
   text
   comments {
         text
         parentPost { author { surname} }
   }
 }
}
```

- *comments* here does retrieve a list of comments
- Note the use of *author*: the same instances are accessed twice, but retrieving different fields
  - ie, *name* and *surname*
  - *post.author == post.comments[i].parentPost.author*
- Working on a graph, a query could be arbitrarily deep when following links between nodes

Response

```
{
 "data": {
  "allPosts": [
   { "id": "3",
    "author": {"name": "Foo"},
    "text": "Foo is the word!",
    "comments": [
     { "text": "No it is not!",
      "parentPost": {"author": {"surname": "Bar"} }
     },
     { "text": "Yes it is!",
      "parentPost": { "author": {"surname": "Bar"} }
     }
    ]
   },
   // other posts…
```

- What we get back is a JSON object
- Payload is under a “*data*” field
- Payload will have same *shape* of the query
- Similar to a Wrapped Response in REST, where in case of errors we have “*data*” being *null* and a “*errors*” field with info on the error(s)

Change Operators

- To modify data, GraphQL defines “*mutation*” operators
- These are Remote Procedure Calls (RPC)
- In other words, a GraphQL server can define a set of methods that can be invoked remotely
- Input/output data should be basic types
- *Benefits*: high flexibility, can do whatever you want
- *Downsides*: high flexibility, each API will behave differently

GraphQL Over HTTP

- Either via a POST or a GET
- Eg, **POST localhost/graphql**
  - JSON payload: **{ "query" : "{all{id}}" }**
  - Here, the actual query is a string stored in the variable called “query”
- Eg, **GET localhost/graphql?query=%7Ball%7Bname%7D%7D**
  - Here the query is passed as a URL query parameter called “query”, and not in a JSON object
  - Note that symbols **{** and **}** need to be escaped with **%7B** and **%7D**

HTTP Idempotency

- Need to remember that GET is idempotent, whereas POST is not
- So, a "*mutation*" operation that changes the server state must not be sent via a GET
  - GraphQL HTTP Services will likely throw an exception in those cases
- So, "*mutations*" must go via a POST, whereas read operations could go either way, POST or GET

GraphQL Benefits

- Why did Facebook need to create a yet another type of web service instead of just using REST???
- *Client has full control on what retrieved*
  - Do not retrieve fields that are not needed
  - Can retrieve all needed data in a **SINGLE** HTTP call
  - Very important for *mobiles*, to reduce bandwidth and energy consumption
- Can have drastic changes in what called from clients without the need to change the server
  - ie, GraphQL is very flexible
- Note: could achieve same things in REST, but it will end up in *manually* re-implementing GraphQL on top of a REST service

GraphQL Downsides

- More difficult to implement the server (at least on the JVM)
- Can use existing libraries, but still it is more difficult to achieve high *server-side* performance
  - eg, think about how to create optimized SQL queries on databases which could be based on GraphQL queries of *any* shape on the graph
  - eg., in REST, could provide high performant, optimized endpoints for widely used operations
- No common semantics of "*mutations*" among different services
  - so, for each new service, need to study its docs/code to have an idea of what they do… which is quite different from typical POST/PUT in REST APIs
- No native handling of authentication, versioning and caching
  - eg, have to rely on transport protocol like HTTP
  - eg, more complex HTTP caches, as here there is only one single endpoint
- Relatively new technology, so tooling still needs improvement
  - but this will get better with passing of time…

REST or GraphQL???

- Will GraphQL replace REST???
- Maybe… maybe not… *too early to tell*
- Better for clients, but can be worse for servers
- RPC for mutations has quite a few downsides
- Personally, I quite like GraphQL, but current tooling still has many rough edges
  - especially on the JVM
  - support in JavaScript is better

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **advanced/graphql/base**
- **advanced/graphql/resolver**
- **advanced/graphql/database**
- **advanced/graphql/graphql-dto**
- **advanced/graphql/mutation**
- **advanced/graphql/news-graphql**