Web Development and API Design

Lesson 11: GraphQL APIs

Prof. Andrea Arcuri

Goals

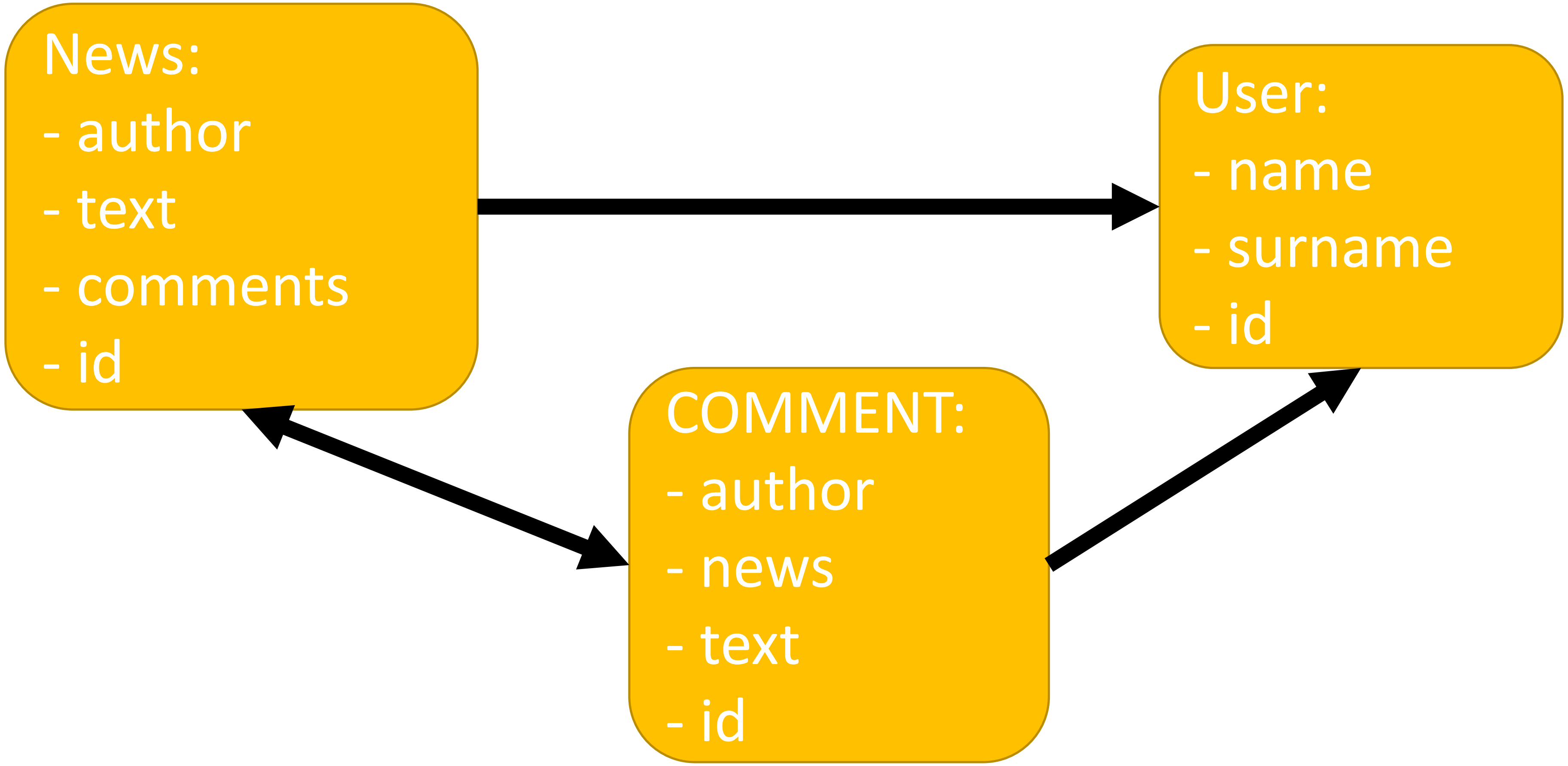
- Understand how to use and develop *GraphQL* web services
- Understand the differences between *REST* and *GraphQL*

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

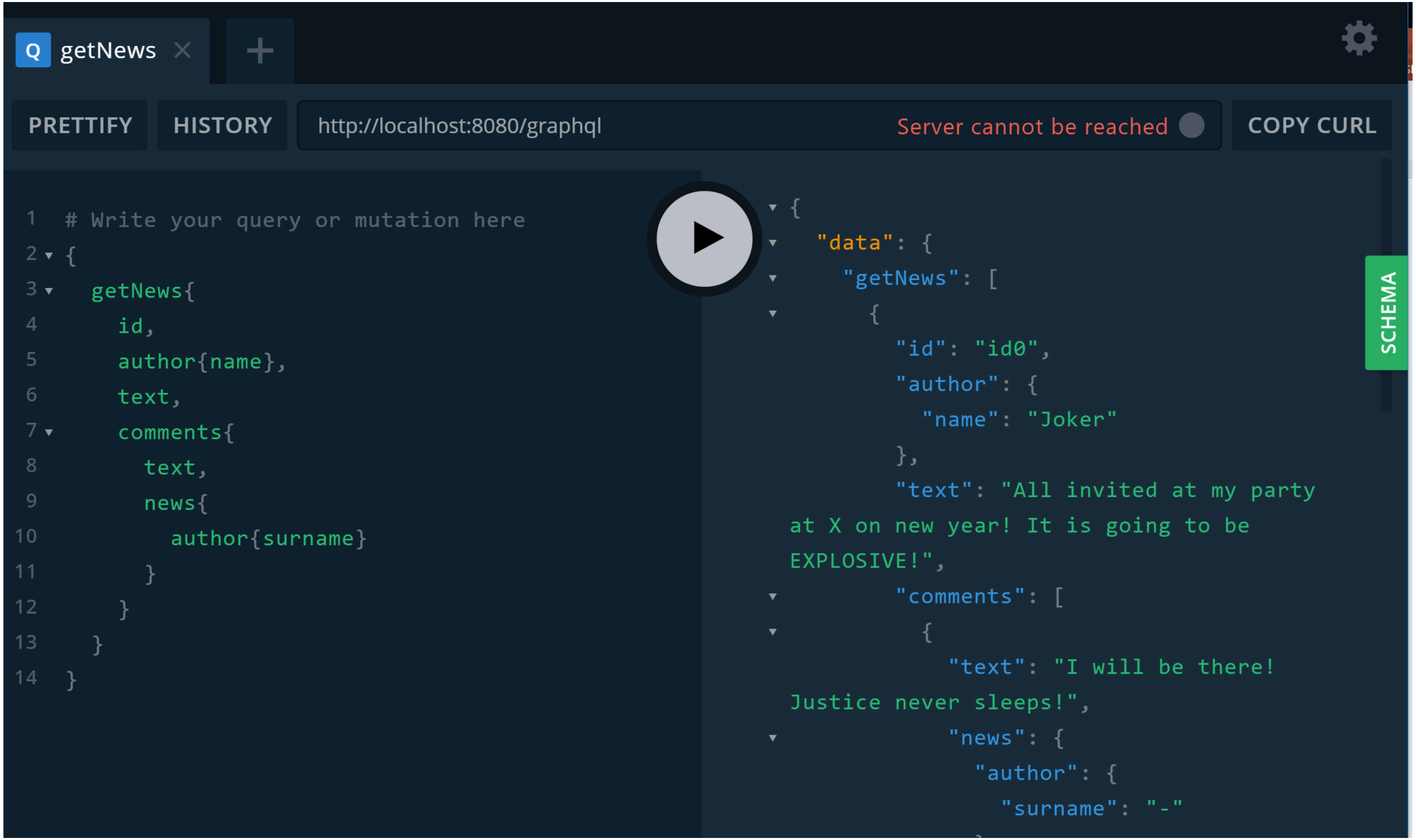
- Eg, forum posts with comments, and author info
- On backend, could be saved in a SQL database



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

Can use tools to visualize and debug queries



```
# Write your query or mutation here
{
  getNews{
    id,
    author{name},
    text,
    comments{
      text,
      news{
        author{surname}
      }
    }
  }
}
```

```
{
  "data": {
    "getNews": [
      {
        "id": "id0",
        "author": {
          "name": "Joker"
        },
        "text": "All invited at my party at X on new year! It is going to be EXPLOSIVE!",
        "comments": [
          {
            "text": "I will be there! Justice never sleeps!",
            "news": {
              "author": {
                "surname": "-"
```

Structure of a Query

```
{
 getNews{
  id,
  author{name},
  text,
  comments{
   text,
   news{ author{surname}}
  }
 }
}
```

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

Cont.

```
{
 getNews{
  id,
  author{name},
  text,
  comments{
   text,
   news{ 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*
  - *news.author === news.comments[i].news.author*
- Working on a graph, a query could be arbitrarily deep when following links between nodes

Response

```
{
 "data": {
  "getNews": [
   {
    "id": "id0",
    "author": {
     "name": "Joker"
    },
    "text": "All invited at my party at X on new year! It is going to be EXPLOSIVE!",
    "comments": [
     {
      "text": "I will be there! Justice never sleeps!",
      "news": {
       "author": {
        "surname": "-"
       }
  // other posts…
```

- What we get back is a JSON object
- Payload is under a “*data*” field
- Payload will have same *shape* of the query
- 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 when dealing with databases
  - 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

Cont.

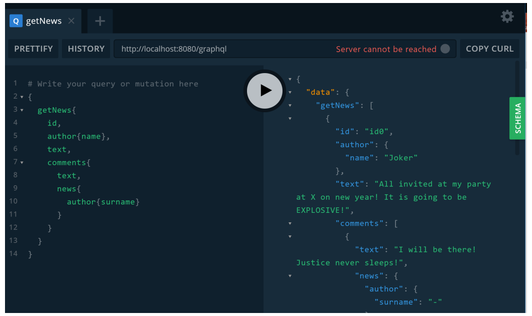
- Relatively new technology, so tooling still needs improvement
  - fine for JavaScript, but not so well supported yet in other languages
  - but this will get better with passing of time…
- No unbound recursive relationships
  - eg, assume you can have Comments on Comments… in *GraphQL*, you cannot specify to retrieve *all* comments in a tree regardless of its depth… whereas it would be simple with REST

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, and not so widespread yet

GraphQL in NodeJS

- Going to use library *Apollo* to add a "*/graphql*" endpoint to Express
- When making requests with "*Accept:*/**" or in JSON, we will get back the JSON response of *GraphQL*
- However, if "*Accept:text/html*", we will get a web app in which we can test the *GraphQL* API
    - this is what happens when using address bar in a browser



```
# Write your query or mutation here
{
  getNews{
    id,
    author{name},
    text,
    comments{
      text,
      news{
        author{surname}
      }
    }
  }
}
```

```
{
  "data": {
    "getNews": [
      {
        "id": "id0",
        "author": {
          "name": "Joker"
        },
        "text": "All invited at my party
at X on new year! It is going to be
EXPLOSIVE!",
        "comments": [
          {
            "text": "I will be there!
Justice never sleeps!",
            "news": {
              "author": {
                "surname": "-"
```