Quarkus - GraphQL

Agenda

- What is GraphQL?
- Origin
- GraphQL vs REST
- Features
- Use Cases
- Core Concepts
- Quarkus and GraphQL

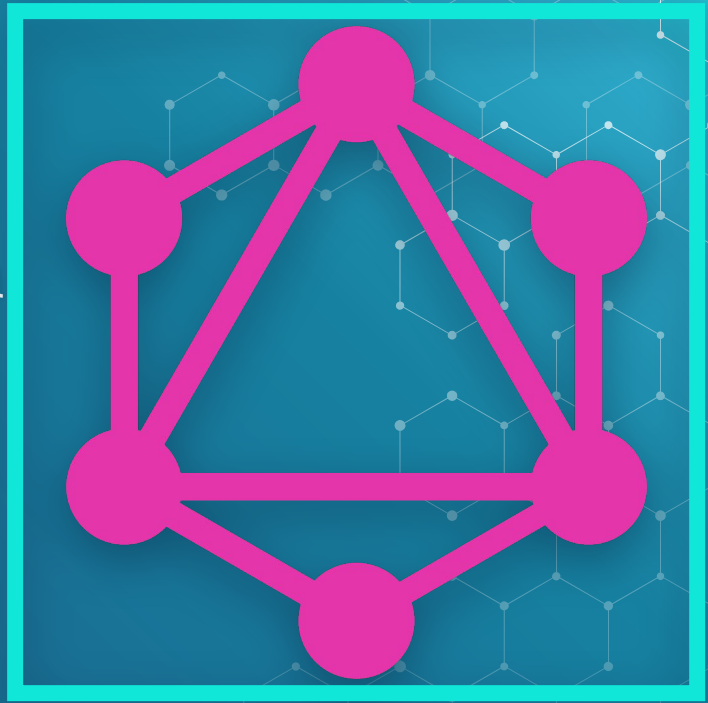
1 What is GraphQL?

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.

GraphQL

Provides:

- Complete and understandable data description
- Power for clients to ask for exactly what they need



Origin

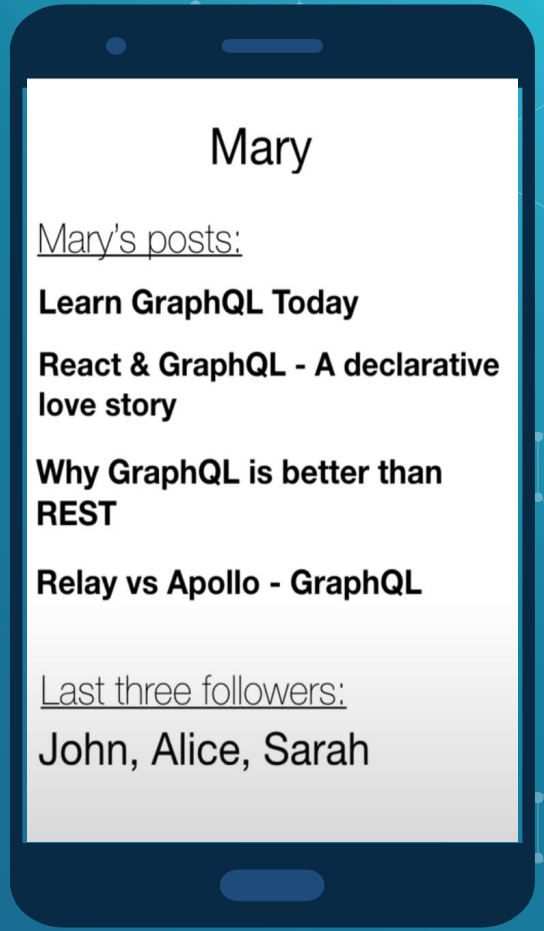
- Developed by Facebook in 2012
  - Combating a lack of modularity
- Goal was to allow for more dynamic development
- Open-sourced in 2015 together with React

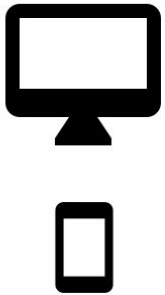
Used by:

2 GraphQL vs REST

Blog example

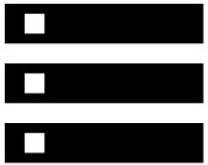
Drawing a view in a blogging app

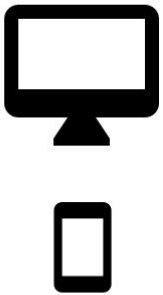




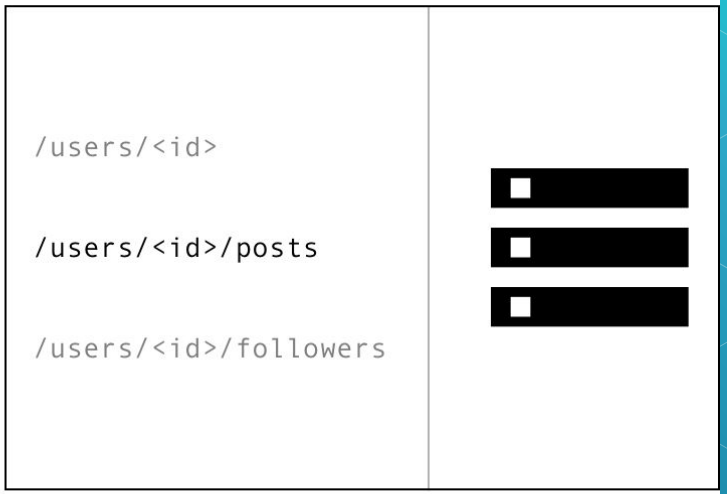

```
{
  “user”: {
    “id”: “er3tg439frjw”
    “name”: “Mary”,
    “address”: { … },
    “birthday”: “July 26, 1982”
  }
}
```

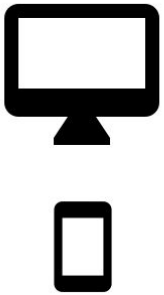
```
/users/<id>

/users/<id>/posts

/users/<id>/followers
```



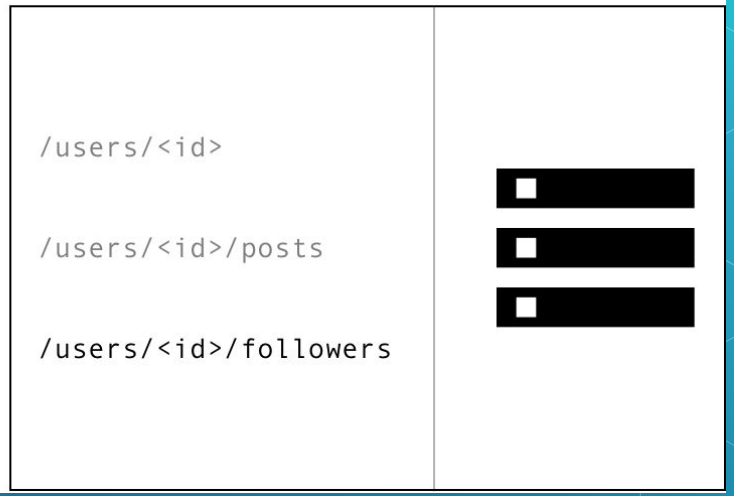



```
{
  “posts”: [{
    “id”: “ncwon3ce89hs”
    “title”: “Learn GraphQL today”,
    “content”: “Lorem ipsum … ”,
    “comments”: [ … ],
  }]
}
```



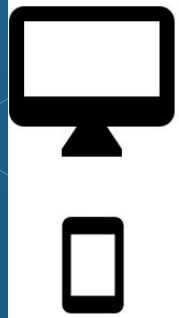



{
"followers": [{
"id": "leo83h2dojsu"
"name": "John",
"address": { … },
"birthday": "July 26, 1982"
},
…]
}
HTTP GET

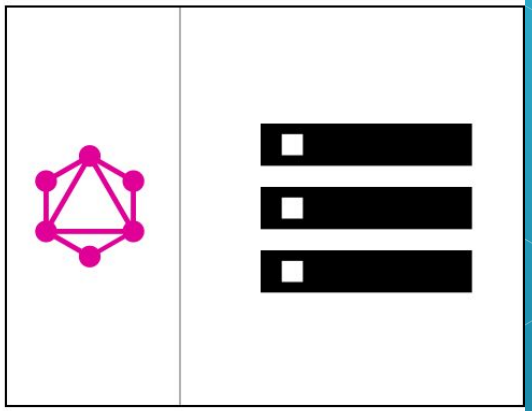

/users/<id>
/users/<id>/posts
/users/<id>/followers


1
HTTP GET
{
"user": {
"id": "er3tg439frjw"
"name": "Mary",
"address": { … },
"birthday": "July 26, 1982"
}
}
/users/<id>
/users/<id>/posts
/users/<id>/followers
2
HTTP GET
{
"posts": [{
"id": "ncwon3ce89hs"
"title": "Learn GraphQL today",
"content": "Lorem ipsum … ",
"comments": […],
}]
}
/users/<id>
/users/<id>/posts
/users/<id>/followers
3
{
"followers": [{
"id": "leo83h2dojsu"
"name": "John",
"address": { … },
"birthday": "July 26, 1982"
},
…]
}
HTTP GET
/users/<id>
/users/<id>/posts
/users/<id>/followers

```
query {
  User(id: “er3tg439frjw”) {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```



HTTP POST



```
{
  “data”: {
    “User”: {
      “name”: “Mary”,
      “posts”: [
        { title: “Learn GraphQL today” }
      ],
      “followers”: [
        { name: “John” },
        { name: “Alice” },
        { name: “Sarah” },
      ]
    }
  }
}
```

No more Over- and Under-fetching

- **Overfetching:** Downloading unnecessary data
- **Underfetching:** An endpoint doesn’t return enough of the right information; need to send multiple requests (n+1-request problem)

Rapid Product Iterations

- REST: structure endpoints according to client's data needs.
- GraphQL:
  - Often no need to adjust API when product requirements and design change.
  - Faster feedback cycles and product iterations.

Analytics

- Fine-grained info about what data is requested
- Enables evolving API and deprecating unneeded API features

Features

- Defines a data shape
- Hierarchical
- Strongly-typed
- Protocol, not storage
- Introspective
- Version-free / backwards compatible

Use Cases

- Apps for mobile devices, where bandwidth usage matters
- Applications where nested data needs to be fetched in a single call
- Composite pattern, where application retrieves data from multiple, different storage APIs.
- Proxy pattern on client side, GraphQL can be added as an abstraction on an existing API, so that each end-user can specify response structure based on their needs

Server-Side Components

Schema

Is at the center of any GraphQL server implementation and describes the functionality available to the clients which connect to it.

Query

Is the client application request to retrieve data from database or legacy API's.

Resolver

Provide the instructions for turning a GraphQL operation into data. They resolve the query to data by defining resolver functions.

Core Concepts

Client-Side Components

GraphiQL

Browser based interface for editing and testing GraphQL queries and mutations.

ApolloClient

is a comprehensive state management library for JavaScript that enables you to manage both local and remote data with GraphQL.

Query

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

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

Query

```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

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

Arguments

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

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

Operation Name

```
query HeroNameAndFriends {
  hero {
    name
    friends {
      name
    }
  }
}
```

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

Mutations

```
mutation CreateReviewForEpisode($ep: Episode!, $review: ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}
```

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

Schema & Types

- GraphQL uses strong type system to define capabilities of an API
- Schema provides clarity to client about the objects available
- Frontend and backend teams can work completely independent from each other

The basics

```
type Starship {
  id: ID!
  name: String!
  length(unit: LengthUnit = METER): Float
}
```

Query and Mutation

```
query {
  hero {
    name
  }
  droid(id: "2000") {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    },
    "droid": {
      "name": "C-3PO"
    }
  }
}
```

Query and Mutation

```
type Query {
  hero(episode: Episode): Character
  droid(id: ID!): Droid
}
```

Interfaces

```
interface Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
}
```

```
type Human implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  starships: [Starship]
  totalCredits: Int
}

type Droid implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction: String
}
```

SmallRye GraphQL

- Quarkus extension
- Implementation of Mircroprofile GraphQL Specification
- Added tools & features

THANK YOU FOR YOUR ATTENTION!

ANY QUESTIONS?