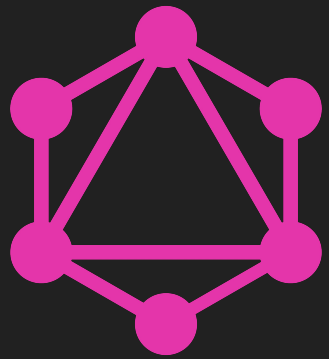


GraphQL Theory

by Roman Savchenko

Agenda

- What is GraphQL?
- GraphQL vs REST API
- Core concepts
- Schema
- Links

GraphQL - A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

What is GraphQL?

- new API standard that was invented and open-sourced by Facebook
- enables declarative data fetching
- GraphQL exposes single endpoint and responds to queries

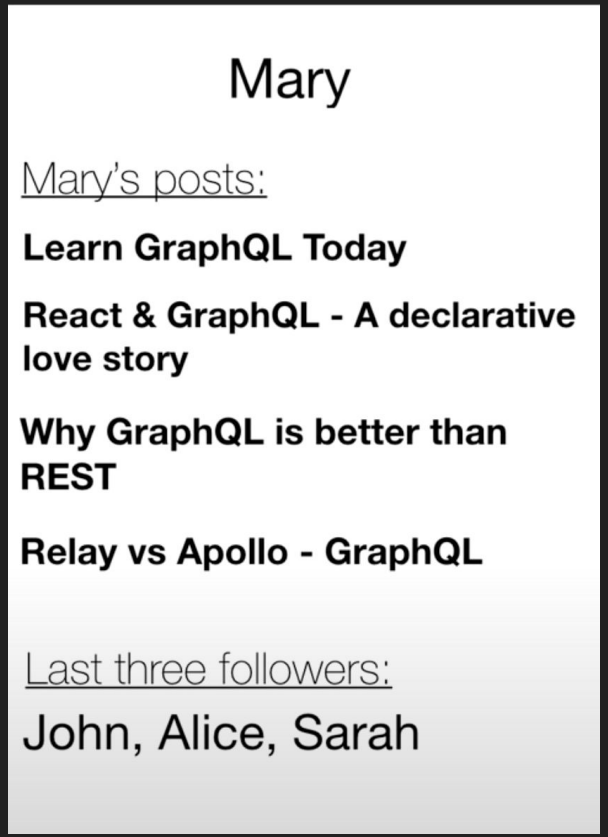
A more efficient alternative to REST

- Increased mobile usage creates need for efficient data loading
- Variety of different frontend frameworks and platforms
- Fast development & expectation for rapid feature development

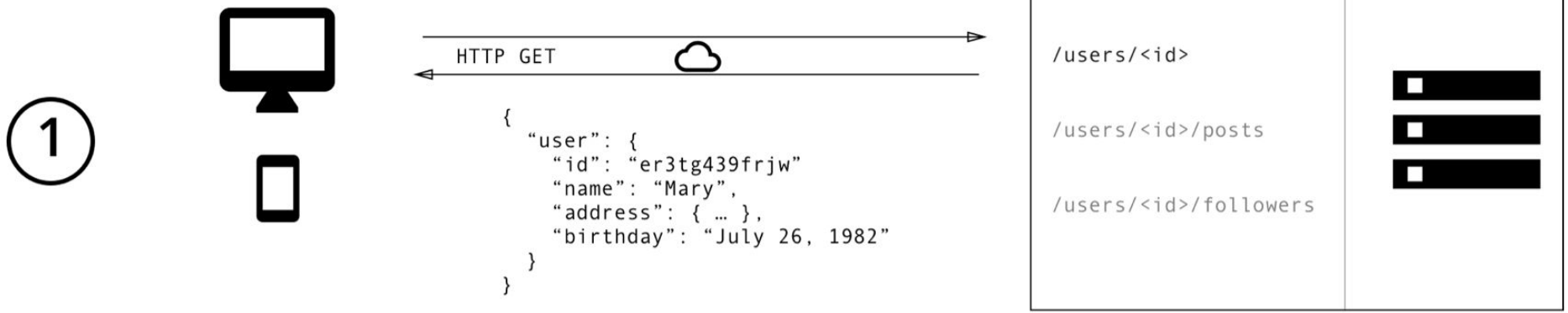
GraphQL vs REST API

Example task - show user's card

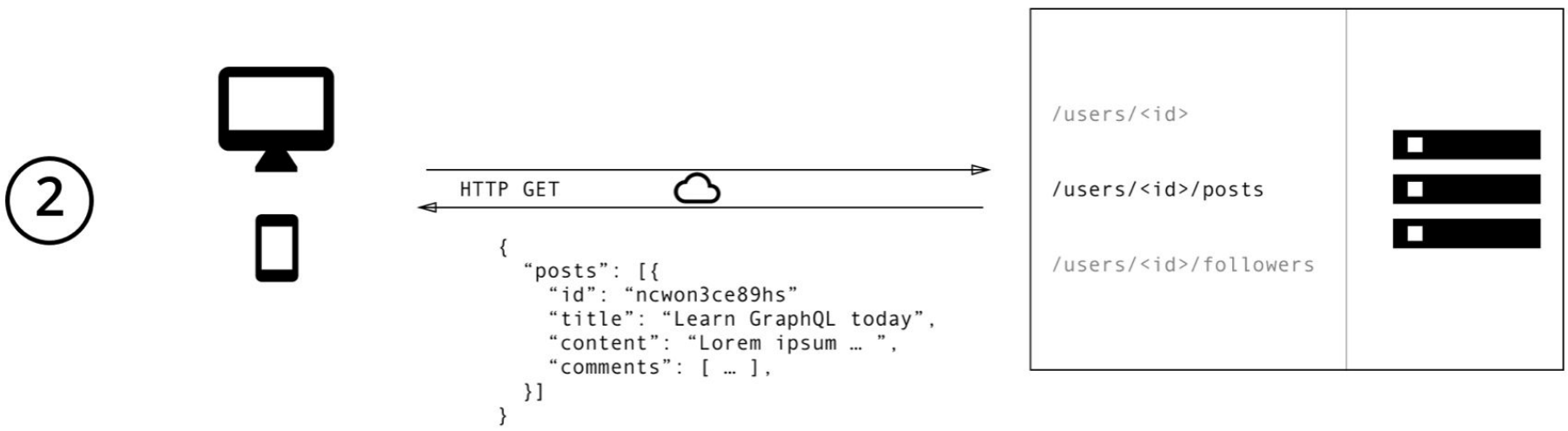
- show user's name
- show user's posts
- show user's last 3 followers



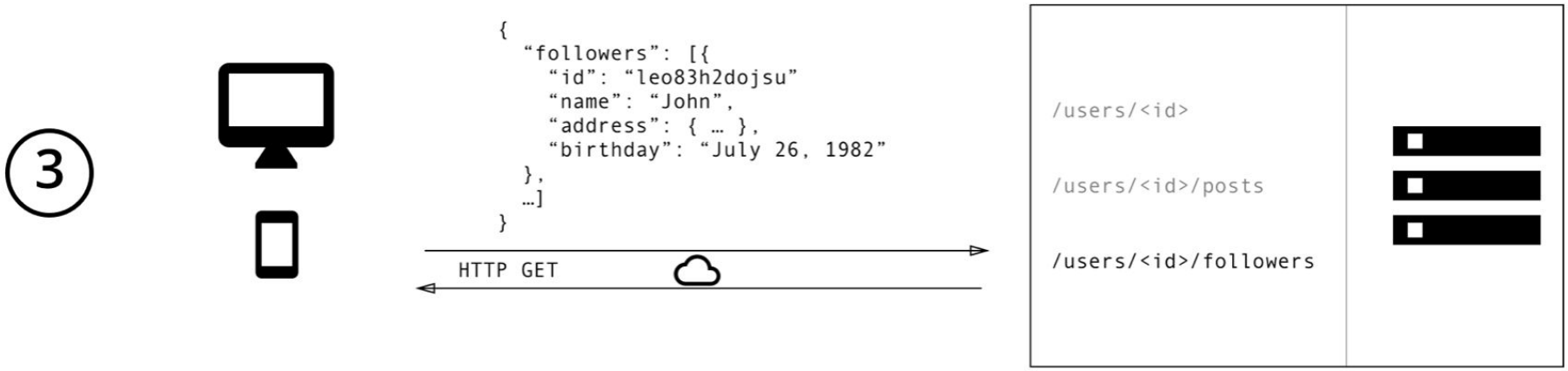
REST API way(step 1)



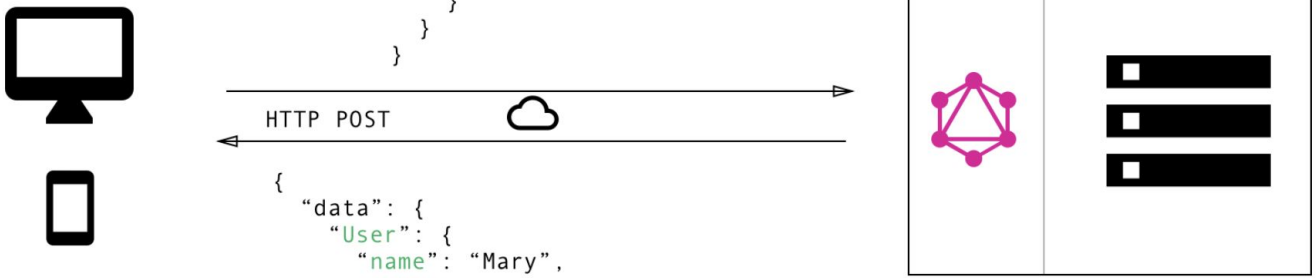
REST API way(step 2)



REST API way(step 3)



GraphQL way(single step)



No more Over- and Underfetching

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

Core Concepts:

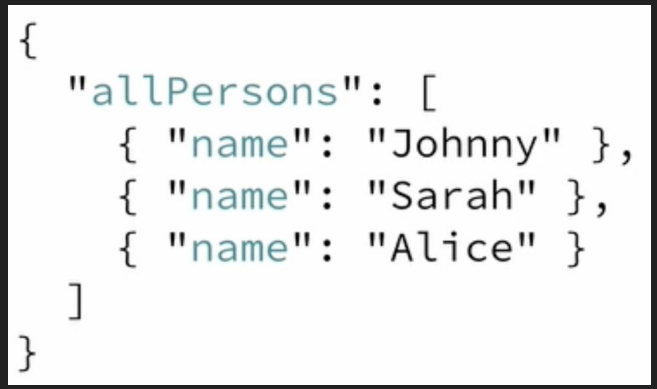
1. The Schema Definition Language (SDL)

```
type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}
```

```
type Post {
  title: String!
  author: Person!
}
```

2. Core Concepts: Queries

```
{
  allPersons {
    name
  }
}
```



2. Core Concepts: Queries

```
{
  allPersons {
    name
    age
  }
}
```

```
{
  "allPersons": [
    { "name": "Johnny", "age": 23 },
    { "name": "Sarah", "age": 20 },
    { "name": "Alice", "age": 20 }
  ]
}
```

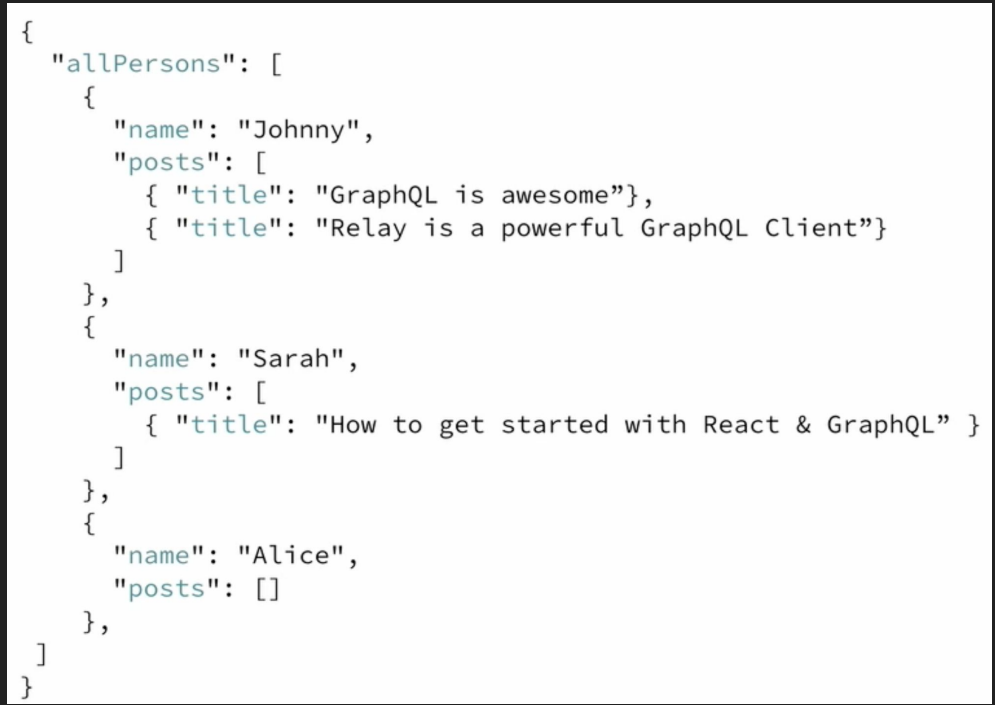
2. Core Concepts: Queries

```
{
  allPersons(last: 2) {
    name
    age
  }
}
```

```
{
  "allPersons": [
    { "name": "Sarah", "age": 20 },
    { "name": "Alice", "age": 20 }
  ]
}
```

2. Core Concepts: Queries

```
{
  allPersons(last: 2) {
    name
    posts {
      title
    }
  }
}
```



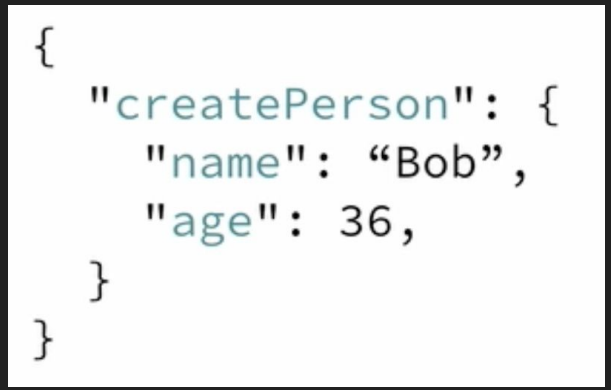
```
{
  "allPersons": [
    {
      "name": "Johnny",
      "posts": [
        { "title": "GraphQL is awesome”},
        { "title": "Relay is a powerful GraphQL Client”}
      ]
    },
    {
      "name": "Sarah",
      "posts": [
        { "title": "How to get started with React & GraphQL” }
      ]
    },
    {
      "name": "Alice",
      "posts": []
    },
 ]
}
```

3. Core Concepts: Mutation

- create new data
- update existing data
- deleting existing data

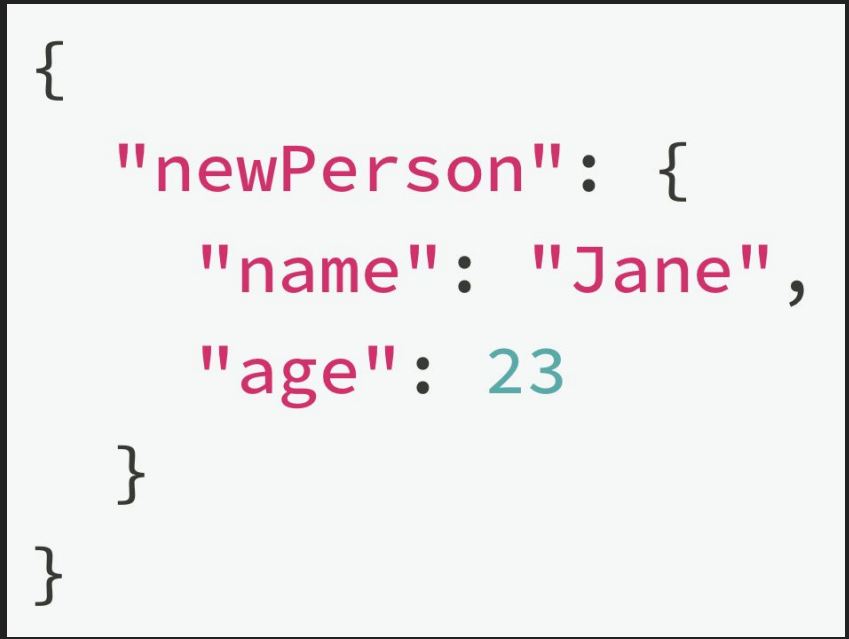
3. Core Concepts: Mutations

```
mutation {
  createPerson(name: "Bob", age: 36) {
    name
    age
  }
}
```



4. Core Concepts: Subscriptions

```
subscription {
  newPerson {
    name
    age
  }
}
```



Core Concepts: The GraphQL Schema

- defines capabilities of the API by specifying how a client can fetch and update data
- represents CONTRACT between client and server
- collection of GraphQL types with special root types

Core Concepts: Root Types

```
type Query {
  …
}
```

```
type Mutation {
  …
}
```

```
type Subscription {
  …
}
```

The Query Type

```
{
  allPersons {
    name
  }
}
```

```
type Query {
  allPersons(last: Int): [Person!]!
}
```

The Mutation Type

```
mutation {
  createPerson(name: "Bob", age: 36) {
    name
    age
  }
}
```

```
type Mutation {
  createPerson(name: String!, age: Int!): Person!
}
```

The Subscription Type

```
subscription {
  newPerson {
    name
    age
  }
}
```

```
type Subscription {
  newPerson: Person!
}
```

Full Schema

```
type Query {
  allPersons(last: Int): [Person!]!
}

type Mutation {
  createPerson(name: String!, age: Int!): Person!
}

type Subscription {
  newPerson: Person!
}
```

```
type Person {
  id: ID!
  name: String!
  age: Int!
  posts: [Post!]!
}

type Post {
  title: String!
  author: Person!
}
```

Full Schema(covering CRUD)

```
type Query {
  allPersons(last: Int): [Person!]!
  allPosts(last: Int): [Post]!
}
type Mutation {
  createPerson(name: String!, age: Int!): Person!
  updatePerson(id: ID!, name: String!, age: Int!): Person!
  deletePerson(id: ID!): Person!
  createPost(title: String!): Post!
  updatePost(id: ID!, title: String!): Post!
  deletePost(id: ID!): Post!
}
type Subscription {
  newPerson: Person!
  updatedPerson: Person!
  deletedPerson: Person!
  newPost: Post!
  updatedPost: Post!
  deletedPost: Post!
}
```

```
type Person {
  id: ID!
  name: String!
  age: Int!
  posts: [Post!]!
}

type Post {
  title: String!
  author: Person!
}
```

Useful links

https://graphql.org - Official website

https://github.com/graphql/graphql-spec - GraphQL specification

https://www.apollographql.com - Most popular GraphQL client for iOS

https://www.apollographql.com/blog/community/backend/8-free-to-use-graphql-apis-for-your-projects-and-demos/ - Free GraphQL APIs

Thank You