GraphQL

* Is new API standard that was invented & open sourced by Facebook
* Enables declarative data fetching
  + Allows API to specify exactly which data it needs
* GraphQL server exposes single endpoint and responds with exactly the data it needs

Why GraphQL

* Increased mobile usage creates need for efficient data loading
  + Low powered devices, sloppy network were the initial reasons graphQL was developed by Facebook
* Variety of different frontend frameworks and platforms on the client-side
* Fast development speed & expectation for rapid feature development

GraphQL is not only for React Developers

* Facebook uses GraphQL since 2012 in their native mobile apps
* GraphQL can be used with any programming language and framework
* GraphQL is used by other big companies such as Coursera yelp and twitter

GraphQL vs REST

* Great idea in REST: stateless servers and & structured access to resources
* REST is a strict specification
* Rapidly changing requirements on client-side don’t go well with REST

**Key: GraphQL was developed to cope with the need for more flexibility and efficiency in client server communication**

Example: Blogging App with REST

Graphical user interface, application, PowerPoint

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Traditional REST

* API endpoint ‘**/users/<id>**’ only needs **id** and **name**
  + Adding **Address** and **birthday** puts extra strain on user’s data plan
* We are actually downloading data that we don’t want to display
* To be conservative on usage 🡪 you need to refine API for that specific purpose
  + Not a good idea when the app needs quick iteration

GraphQL

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

* Fetches everything on single request
  + Allows to fetch only needed data 🡪 saving data on user
* Advantages
  + 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 requests problem)

**Rapid Product Iterations**

* REST: structure endpoints according to clients’ data needs
* No need to adjust API when product requirements and design change
* Faster feedback cycles and product iterations

**Insightful Analytics**

* Fine-grained info about what data is read by clients
* Enables evolving API and deprecating unneeded API features
* Great opportunities for instrumenting and performance monitoring

**Benefits of Schema & Types**

* GraphQL uses strong type system to define capabilities of an API
* Schema serves as contract between client and server
* Frontend and backend teams can work completely independent from each other

**Core Concepts**

* **Schema Definition Language**
  + Is it’s own type for defining it’s own schema

Rectangle

Description automatically generated with medium confidence

* + **!** following the type means that this field is ***required*.**

A picture containing graphical user interface

Description automatically generated

* Note that we just created a *one-to-many*-relationship between Person and Post since the posts field on Person is actually an *array* of posts

* **Fetching data with queries**
  + REST APIs, data is loaded from specific endpoints
    - Each endpoint has a clearly defined structure of the information that it returns.
    - This means that the data requirements of a client are effectively *encoded* in the URL that it connects to.
  + GraphQL APIs typically only expose *a* ***single endpoint*.**
    - it’s completely flexible and lets the client decide what data is actually needed.
* **Basic queries**

Example 1

Rectangle

Description automatically generated with low confidence

Response

Graphical user interface

Description automatically generated with low confidence