Topics

(References: https://graphql.org/learn & https://www.howtographql.com)

- Introduction
- GraphQL is the better REST
- Core Concepts - The Schema Definition Language (SDL)
- Queries & Mutations
- Schemas and Types
- GraphQL client and server
- Connecting with Database via Prisma
- GraphQL Tools and Ecosystem
- Security

GraphQL

A query language for your API

Introduction

- 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.

Facebook GraphQL Draft

- GraphQL, a query language and execution engine originally created at Facebook in 2012 for describing the capabilities and requirements of data models for client-server applications.
- The development of this open standard started in 2015.
- **Source: https://facebook.github.io/graphql/draft/**

Who's using GraphQL

- Facebook
- GitHub
- Coursera
- Dailymotion
- Myntra
- PayPal
- Pinterest
- Shopify
- Twitter

https://graphql.org/users/

More on GraphQL

- GraphQL is not a programming language capable of arbitrary computation, but is instead a language used to query application servers that have capabilities defined in this specification.
- GraphQL does not mandate a particular programming language or storage system for application servers that implement it.
- Instead, application servers take their capabilities and map them to a uniform language, type system, and philosophy that GraphQL encodes.
- This provides a unified interface friendly to product development and a powerful platform for tool-building.

GraphQL design principles:

- **Hierarchical**:
  - Most product development today involves the creation and manipulation of view hierarchies.
  - To achieve congruence with the structure of these applications, a GraphQL query itself is structured hierarchically.
  - The query is shaped just like the data it returns.
  - It is a natural way for clients to describe data requirements.
- **Product-centric**:
  - GraphQL is unapologetically driven by the requirements of views and the front-end engineers that write them.
  - GraphQL starts with their way of thinking and requirements and builds the language and runtime necessary to enable that.

GraphQL design principles:

- **Strong-typing**:
  - Every GraphQL server defines an application-specific type system.
  - Queries are executed within the context of that type system.
  - Given a query, tools can ensure that the query is both syntactically correct and valid within the GraphQL type system before execution,
  - i.e. at development time, and the server can make certain guarantees about the shape and nature of the response.

GraphQL design principles:

- **Client-specified queries**:
  - Through its type system, a GraphQL server publishes the capabilities that its clients are allowed to consume.
  - It is the client that is responsible for specifying exactly how it will consume those published capabilities.
  - These queries are specified at field-level granularity. In the majority of client-server applications written without GraphQL, the server determines the data returned in its various scripted endpoints.
  - A GraphQL query, on the other hand, returns exactly what a client asks for and no more.

GraphQL design principles:

- **Introspective**:
- GraphQL is introspective.
- A GraphQL server's type system must be queryable by the GraphQL language itself, as will be described in this specification.
- GraphQL introspection serves as a powerful platform for building common tools and client software libraries.

Code - https://graphql.org/code/

- Many different programming languages support GraphQL.
- popular server-side frameworks,
- client libraries,
- services,
- and other useful stuff.