GraphQL Queries

Memi Lavi
www.memilavi.com

GraphQL Queries

- The second step in working with GraphQL is running the query
- While setting up the schema is usually a one-time step (with some modifications later), the whole purpose of it is to run queries against it
- This is the most important part of GraphQL

Query Basics

- We already played with queries when defining schema
- We saw we can use it to query object types, fields and related objects
- Let's have a quick refresher

Operation Type and Name

- So far, when querying, we didn't specify that we actually query the data
- We just typed in what we want to fetch, and let GraphQL do its thing
- GraphQL inferred that what we want to do is to query data and not, for example, create an entity

Operation Type

- It's a good practice to specify the Operation Type when calling GraphQL, so that it'll be clear what we're trying to do

Operation Name

- When calling GraphQL with multiple queries, it's a good practice to name the query
- This helps when debugging and logging the query on the server side
- Simple to implement – just add the name after the `query` keyword

Arguments

- So far, when querying, we always asked to return all the records
- This is nice for learning purposes, but not very practical
- We usually want to retrieve some of the data, and be able to specify how we want to filter it
- For this we have Query Arguments

Variables

- In the arguments that we defined, we included the value of the argument in the query
- Usually we want to specify the value separately from the query string
- For that we can use Variables
- Do not require any code changes

Aliases

- We can include multiple queries in a single query call
- We can also query the same object type multiple times in the same query call
- This can cause a conflict in the result, as the result is named after the object type queried

Aliases

- With aliases, we give a unique name to each query in the request
- The result is named after the alias, and not after the object type
- This way we avoid conflicts

Fragments

- In the previous example we saw two queries, where the only difference between them is the argument
- They both queried for the same fields
- This can be repetitive and error prone
- Fragments allow us to define the set of fields to retrieve and use it in queries

Directives

- So far, when defining a query, it always returned the same set of fields
- We often want the query to return different set of fields based on the specific scenario
- For example: For a list of entities we would need less fields than a detailed view of the entity

Directives

- This can be achieved using Directives
- With directives, we can dynamically change the structure and shape of the query based on variables passed to the query
- Each directive checks whether a given argument is `true` and then executes its instructions

Directives

- Currently two directives are part of GraphQL specification:
- `@include (if: Boolean)` – Include the field if argument is `true`
- `@skip (if: Boolean)` – Skip the field if argument is `true`

Inline Fragments

- We learned before that we can use Interfaces to retrieve multiple object types in a single query
- However, sometimes we might want to retrieve specific type fields even when querying interfaces
- For that we can use Inline Fragments

Inline Fragments

- With Inline Fragments we define which fields should be retrieved in addition to the fields included in the interface
- Use the …type syntax, similar to regular fragments
- Hence the name…

Meta Fields

- In the previous example we saw how we can retrieve multiple object types in a single result
- It can be useful if the query will explicitly tell us what is the object type of each item in the results
- For that we have the Meta Fields

Meta Fields

- Meta fields provide data about the types returned, and are part of the Introspection mechanism
- We'll learn about Introspection in the next lecture
- The most useful meta field is `__typename`, which indicates the name of the object type returned

Introspection

- When we start working with a schema of a GraphQL server, we don't always know what are the objects in the schema and what fields do they contain
- Introspection allows us to explore the schema and learn about its objects and fields
- Used as a query, with GraphQL keywords