GraphQL cheatsheet

This quick reference cheat sheet provides a brief overview of GraphQL.

Getting Started

Overview

- An alternative approach to RESTful APIs
- GraphQL is a query language for APIs
- Easily describe the shape of the GraphQL API using clear shared terms.
- Clients issue queries/mutations to read and update data
- GraphQL syntax can express complex entity relations
- Libraries to implement GraphQL in different languages

GraphQL

Schema

`schema`	GraphQL schema definition
`query`	Read and traverse data
`mutation`	Modify data or trigger an action
`subscription`	Run a query when an event occurs

Built-in Scalar Types

`Int`	Signed 32-bit integer
`Float`	Signed double-precision floating-point value
`String`	UTF-8 character sequence
`Boolean`	true or false
`ID`	A Unique identifier

Type Definitions

`scalar`	Scalar Type
`type`	Object Type
`interface`	Interface Type
`union`	Union Type
`enum`	Enum Type
`input`	Input Object Type

Type Modifiers

`String`	Nullable String
`String!`	Non-null String
`[String]`	List of nullable Strings
`[String]!`	Non-null list of nullable Strings
`[String!]!`	Non-null list of non-null Strings

Input Arguments

Basic Input

```
type Query {
    users(limit: Int): [User]
}
```

Input with default value

```
type Query {
    users(limit: Int = 10): [User]
}
```

Input with multiple arguments

```
type Query {
    users(limit: Int, sort: String): [User
}
```

Input with multiple arguments and default values

```
type Query {
    users(limit: Int = 10, sort: String):
[User]
}
type Query {
    users(limit: Int, sort: String =
"asc"): [User]
}
type Query {
    users(limit: Int = 10, sort: String =
"asc"): [User]
}
```

Input Types

```
input ListUsersInput {
    limit: Int
    since_id: ID
}

type Mutation {
    users(params: ListUsersInput): [User]!
}
```

Custom Scalars

```
scalar Url
type User {
    name: String
    homepage: Url
}
```

Interfaces

```
interface Foo {
    is_foo: Boolean
}
interface Goo {
    is_goo: Boolean
}
type Bar implements Foo {
    is_foo: Boolean
    is_bar: Boolean
}
type Baz implements Foo, Goo {
    is_foo: Boolean
    is_goo: Boolean
    is_baz: Boolean
}
```

Object implementing one or more Interfaces

Unions

```
type Foo {
    name: String
}
type Bar {
    is_bar: String
}
union SingleUnion = Foo
union MultipleUnion = Foo | Bar
type Root {
    single: SingleUnion
    multiple: MultipleUnion
}
```

Union of one or more Objects

Enums

```
enum USER_STATE {
    NOT_FOUND
    ACTIVE
    INACTIVE
    SUSPENDED
}
type Root {
    stateForUser(userID: ID!):
USER_STATE!
    users(state: USER_STATE, limit: Int =
10): [User]
}
```

Also see

GraphQL Schema Language Cheat Sheet (github.com)