GraphQL Cheatsheet

A concise reference for GraphQL syntax, schema definition, queries, mutations, and common directives, designed for quick lookup and efficient development.

GraphQL Basics

Core Concepts

GraphQL: A query language for your API and a server-side runtime for executing those queries. It provides a complete and understandable description of the data in your API, giving clients the power to ask for exactly what they need and nothing more.

Schema: The foundation of a GraphQL API. It defines the data types (objects) available and the operations (queries and mutations) that can be performed.

Queries: Used to fetch data. Clients specify the data they need in a structured format.

Mutations: Used to modify data (create, update, delete). Similar in structure to queries but indicate an intent to change data.

Resolvers: Functions that fetch the data for a particular field in the schema.

Basic Syntax

Query:

```
query {
  user(id: "123") {
    name
    email
  }
}
```

Mutation:

```
mutation {
  createUser(name: "John Doe", email: "john@example.com") {
    id
    name
    email
  }
}
```

Fragment:

```
fragment UserInfo on User {
  id
  name
  email
}

query {
  user(id: "123") {
    ...UserInfo
  }
}
```

Schema Definition Language (SDL)

Object Types

Defines the structure of data objects.

```
type User {
  id: ID!
  name: String!
  email: String
  posts: [Post]
}
```

- `ID`: A unique identifier, often a string.
- `String`: A text string.
- `[Post]`: A list of `Post` objects.
- `!`: Indicates a non-nullable field (required).

Queries and Mutations

Query Example:

```
type Query {
  user(id: ID!): User
  posts: [Post]
}
```

Mutation Example:

```
type Mutation {
  createUser(name: String!, email: String): User
  updateUser(id: ID!, name: String, email: String): User
}
```

Input Types

Used to define the structure of input arguments for mutations.

```
input CreateUserInput {
  name: String!
  email: String
}

type Mutation {
  createUser(input: CreateUserInput!): User
}
```

Queries in Detail

Basic Queries

A simple query to fetch a user's name and email.

```
query {
  user(id: "123") {
    name
    email
  }
}
```

Arguments

Passing arguments to filter or specify the data being fetched.

```
query {
  posts(limit: 10, sortBy: "date") {
    title
    content
  }
}
```

Aliases

Rename the fields in the response.

```
query {
  user1: user(id: "123") {
    name
  }
  user2: user(id: "456") {
    name
  }
}
```

Fragments

Reusable units of query logic.

```
fragment UserInfo on User {
  id
  name
  email
}

query {
  user(id: "123") {
    ...UserInfo
  }
}
```

Inline Fragments

Used when dealing with interfaces or unions.

```
query {
  node(id: "123") {
    id
    ... on User {
      name
      email
    }
    ... on Post {
      title
      content
    }
  }
}
```

Mutations in Detail

Basic Mutations

A simple mutation to create a user.

```
mutation {
  createUser(name: "John Doe", email:
"john@example.com") {
    id
    name
    email
  }
}
```

Arguments and Input Types

Using input types for complex arguments.

```
mutation {
  createUser(input: {name: "John Doe", email:
"john@example.com"}) {
    id
    name
    email
  }
}
```

Returning Updated Data

Mutations typically return the updated data.

```
mutation {
  updateUser(id: "123", name: "Jane Doe") {
    id
    name
    email
  }
}
```