05

API Gateway - GraphQL

Ciclo 4a:

Desarrollo de aplicaciones web

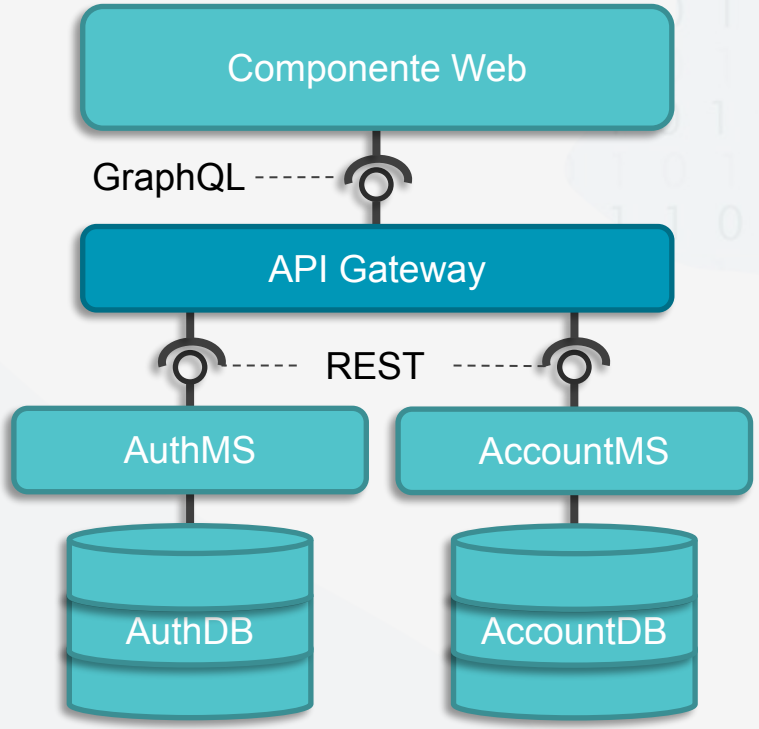
Objetivo de Aprendizaje

Identificar las principales características de un **API Gateway,** dentro de una arquitectura de **Microservicios**.

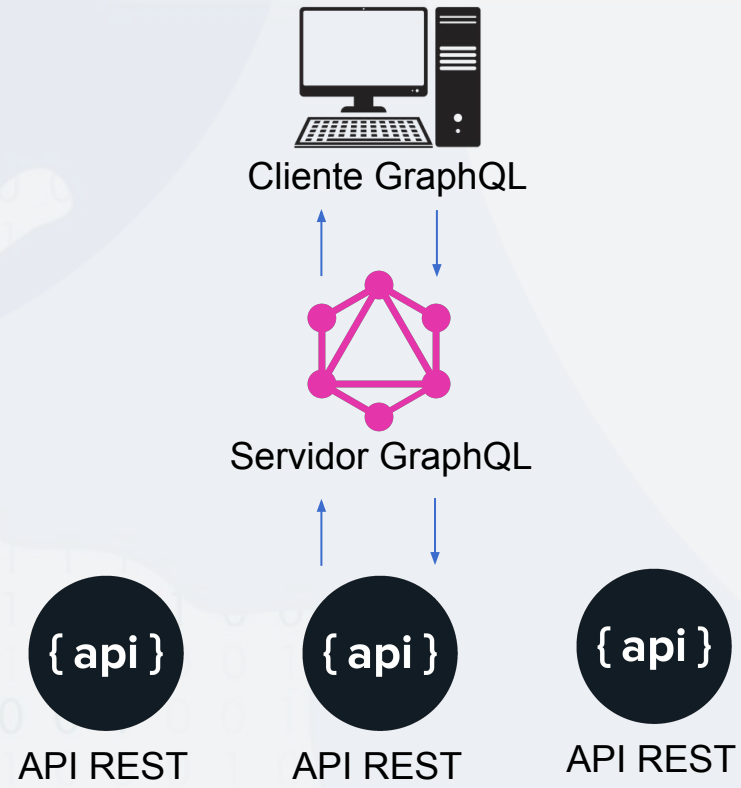
Parte 1

API Gateway

El **API Gateway** es el componente encargado de **recibir** y **responder** las solicitudes del **Componente Web**. Por esta razón, es necesario que el API Gateway se **comunique** con el **Componente Web** y con los **microservicios** que le proveen las funcionalidades de la aplicación. Para realizar la conexión con los microservicios se utilizarán conectores de tipo **REST**, y para la conexión con el componente web se utilizará un conector de tipo **GraphQL**, el cual se introducirá posteriormente.



GraphQL: Definición

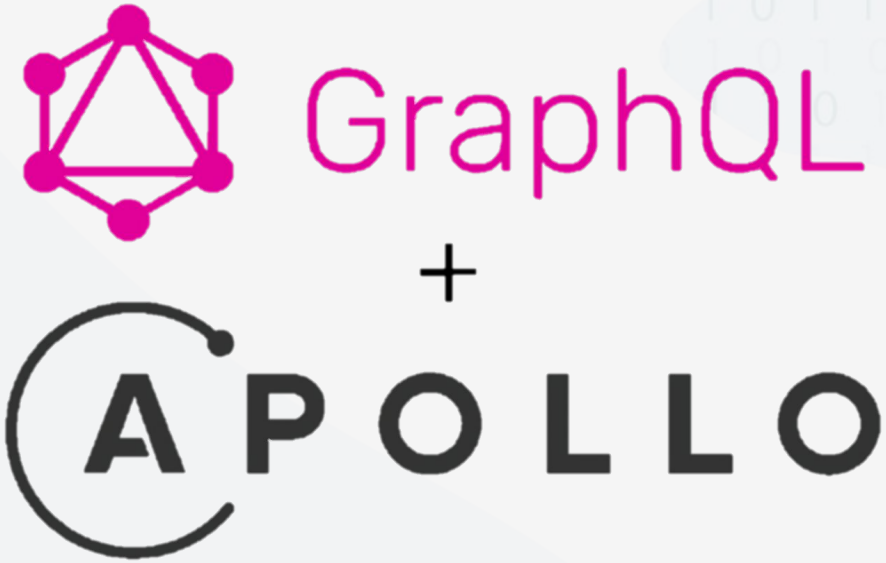


GraphQL es un **entorno de ejecución** para la **consulta** y **manipulación** de datos para APIs, que cuenta con su propio **lenguaje de consulta**. GraphQL es una **alternativa** a la exposición de datos por medio de **REST**, que ofrece algunas **ventajas**, como la consulta **únicamente** de los datos que el cliente requiere, la **descripción** completa de los datos de la API, y la **fácil evolución** de las APIs sobre el tiempo.

GraphQL: Apollo Server

Para **implementar** una API de tipo GraphQL, se debe emplear un **servidor** que utilice el entorno de ejecución de **GraphQL** para recibir y responder peticiones. Existen **múltiples** servidores para cada **lenguaje de programación**. Por ejemplo, para JavaScript existen: Apollo Server, GraphQL.js o Express GraphQL.

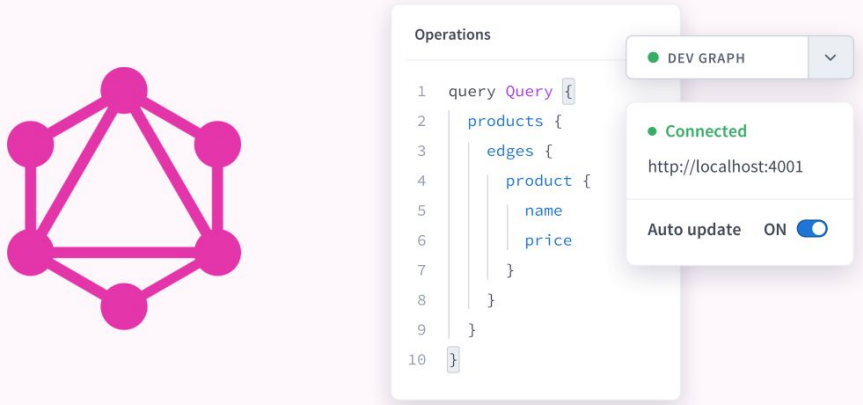
En el sistema de software planeado, el **API Gateway** se desarrollará en **JavaScript** empleando **Apollo Server** para la recepción y respuesta de peticiones.



[Imagen] GraphQL & Apollo. (s. f.). [PNG]. Medium. https://miro.medium.com/max/1400/1*o9Fp5squKFlA6qlM1Yp10A.png

GraphQL: Peticiones

Para realizar **consultas** y **modificaciones** sobre los datos del **API Gateway**, existen 2 tipos de peticiones: **Queries** y **Mutations**. Las **Queries** sirven para **consultar datos**, como una operación GET en REST, mientras que las **Mutations** sirven para **modificar los datos**, como las operaciones POST, PUT y DELETE en REST.

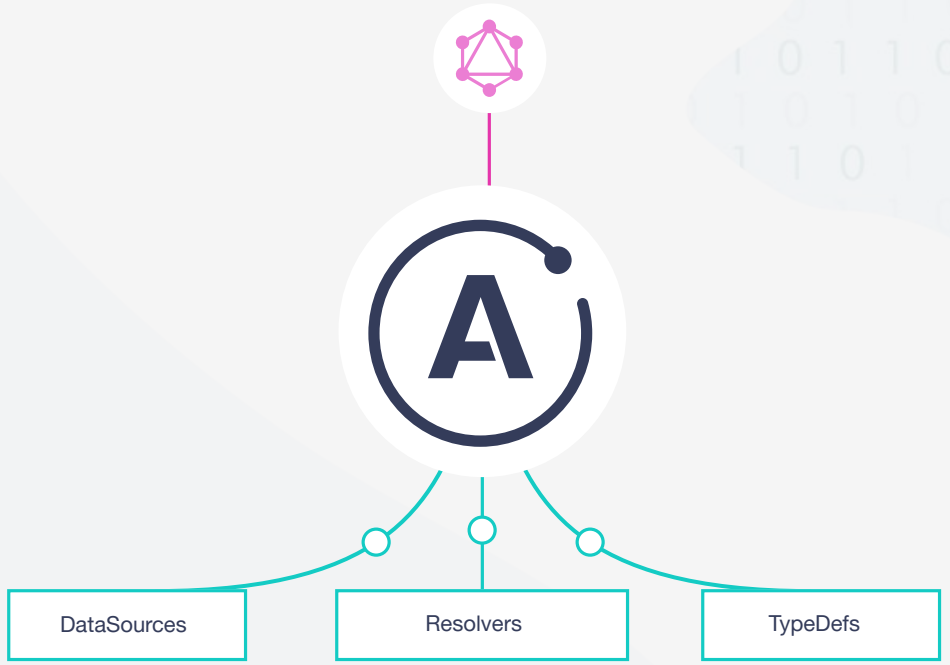


[Imagen] Queries. (s. f.). [PNG]. ApolloGraphQL. https://www.apollographql.com/blog/static/graphql-query-79196ebc9ef66116c562969e686a6cf5.png

Apollo Server: Estructura

Para lograr que un API Gateway pueda llevar a cabo la **orquestación de funcionalidades** entre microservicios, Apollo Server plantea una **estructura compuesta** por 3 pilares principales:

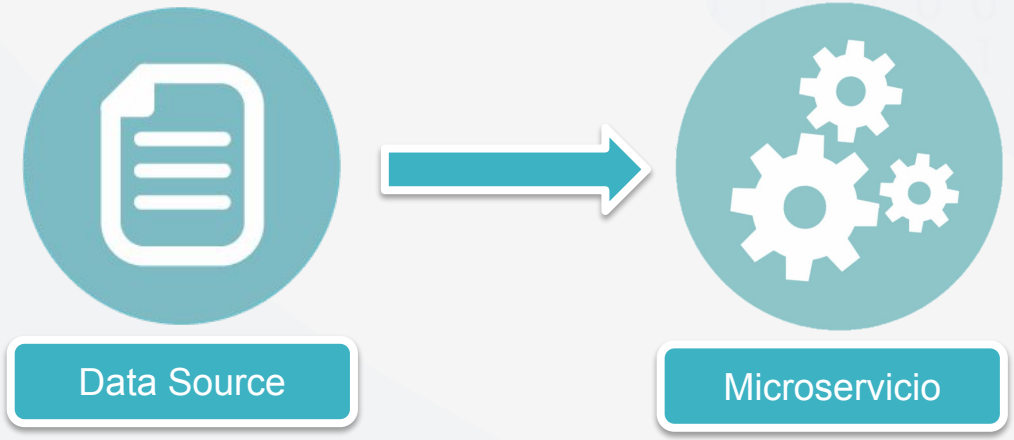
- **DataSources:** encargados de establecer la **comunicación** con los microservicios.
- **TypeDefs:** encargados definir las **operaciones** y los datos que se manejarán.
- **Resolvers:** encargados de **procesar** las operaciones.



Apollo Server: DataSource

Una de la principales tareas de un API Gateway es **realizar peticiones** a distintos microservicios con el fin de **consumir sus servicios**. Dado que estos poseen una API de tipo REST, la comunicación se realiza a través de **peticiones HTTP**.

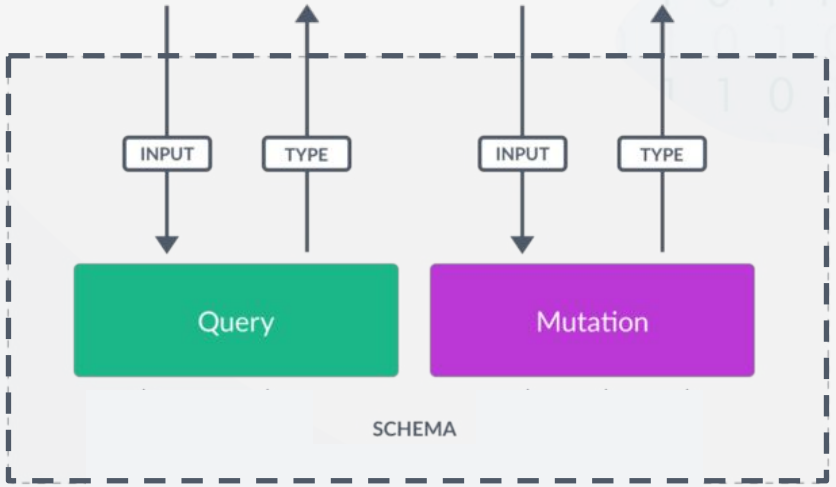
Apollo Server plantea el concepto de **DataSource**, el cual es una abstracción de una **fuente de datos y servicios** (como un microservicio). Para comunicarse con los microservicios, solamente se deberán definir algunos elementos y Apollo Server se encargará del proceso.



Apollo Server: TypeDefs

Una de las ventajas de la interfaz de GraphQL, es la **robustez de su esquema de datos y peticiones**. GraphQL define de manera **clara y concisa** las **peticiones** (Queries y Mutations) que se pueden realizar al API Gateway, así como los **datos de entrada** y de **salida** de estas.

El proceso de **definición de las peticiones y los datos** se realiza en los **TypeDefs**. Un TypeDef es un objeto compuestos por los **encabezados** de las Queries y Mutations , y los **datos** usados en las peticiones.



[imagen] Schema Apollo Server. (s. f.). [Ilustración]. https://miro.medium.com/max/1400/1*O8mWd4Hq4_3c9nM9VoRAlg.png

Apollo Server: TypeDefs

Los **tipos de datos** aceptados como **entradas** y **salidas** de las operaciones pueden ser **escalares** como Int, Float, String o Boolean. Sin embargo, también se aceptan **objetos** construidos a partir de escalares u otros objetos. TypeDef también soporta el concepto de **arreglos** utilizando corchetes cuadrados, de esta forma, el tipo [objetoA] representará un arreglo de elementos del tipo objetoA.

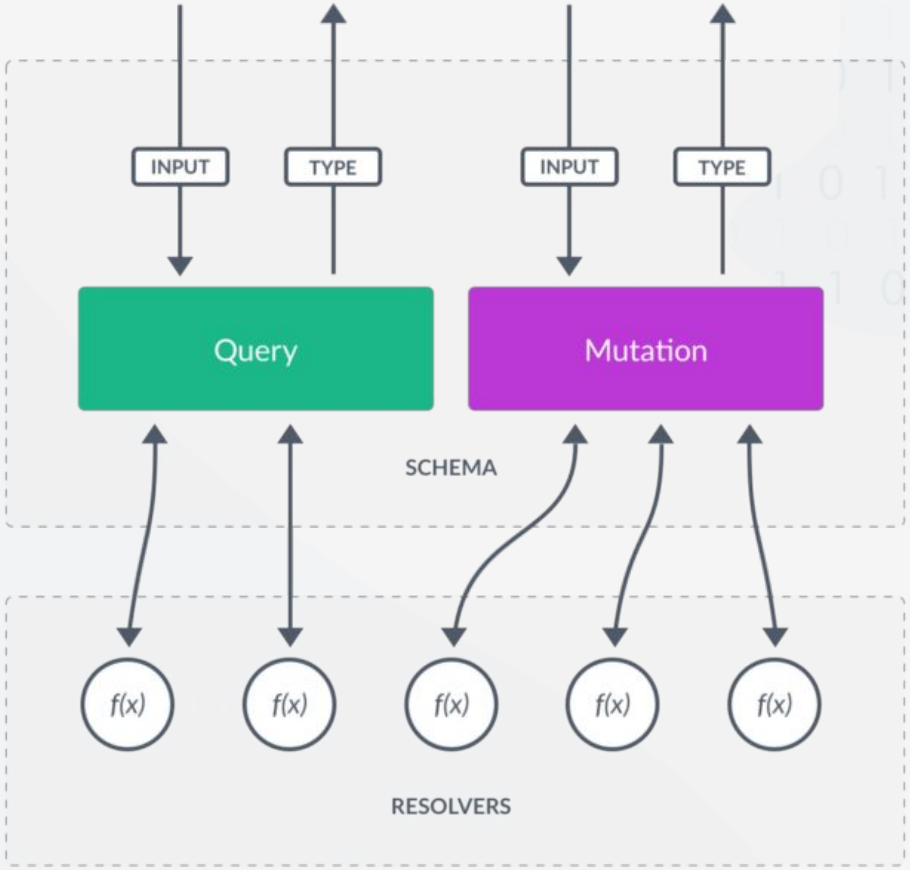
Para indicar si un dato u objeto es **obligatorio** se usa el indicador **!**, si no se utiliza, el dato será **opcional**.

Copy

```
type Query {
  upcomingEvents: [Event!]!
}

type Event {
  name: String!
  date: String!
  location: Location
}

type Location {
  name: String!
  weather: WeatherInfo
}

type WeatherInfo {
  temperature: Float
  description: String
}
```

Apollo Server: Resolvers

Cada una de las peticiones que llegan al API Gateway contienen una **operación**, ya sea una **Query** o una **Mutation**, cada una de estas operaciones debe tener un **mecanismo que permite procesarla**. A este mecanismo, Apollo Server lo llama **Resolver**. En la práctica, un resolver es una **función que procesa una operación**.

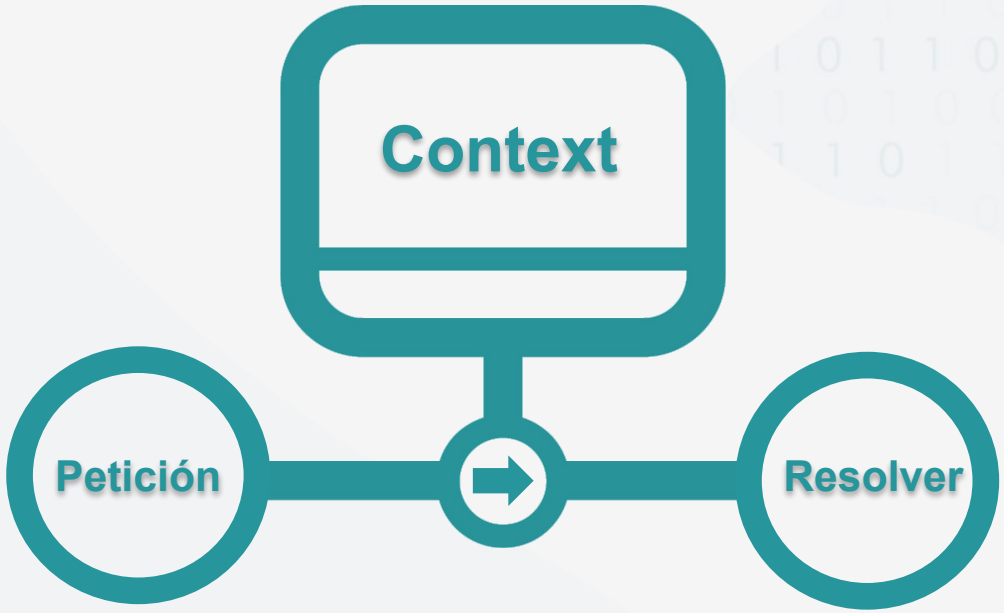


Schema Apollo Server. (s. f.). [Ilustración]. https://miro.medium.com/max/1400/1*O8mWd4Hq4_3c9nM9VoRAIg.png

Apollo Server: Context

Una **herramienta** que no forma parte de los pilares de Apollo Server, pero que resulta **útil** en muchos escenarios, son los **Context**.

Un context es un **preprocesamiento** realizado sobre una **petición**, es decir, dota a la petición de un contexto adicional. En la práctica, un context es una **función** que **recibe** como parámetro **una petición** y **realiza operaciones sobre esta**, la función es **ejecutada justo cuando una petición llega** al API Gateway.



Apollo Server: Context

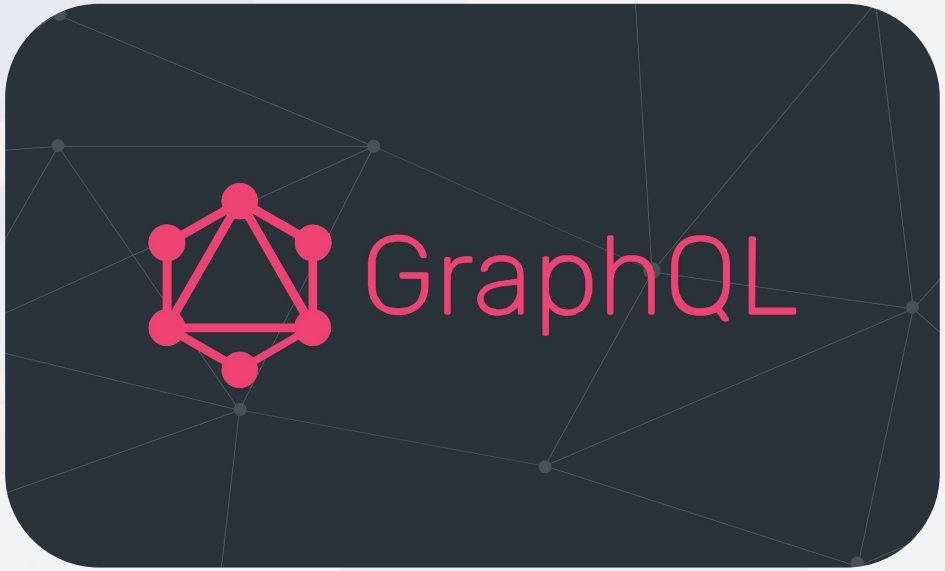
Los **context** son útiles cuando se quiere **expandir** o **revisar** la **información** que contiene una **petición**. Algunos usos comunes son la **autenticación**, donde un context revisa si una petición está autorizada para ejecutarse o no.

Otro caso de uso es cuando la **petición** contiene un **identificador** y el API Gateway **requiere información adicional** de ese identificador, proveniente de **recursos externos**.

[imagen] Autenticación. (s. f.). [Icon]. http://cdn.onlinewebfonts.com/svg/img_354628.png

Parte 2

Peticiones GraphQL: Sintaxis



Para realizar **peticiones** a un servidor GraphQL se deben construir **sentencias** que utilicen la **sintaxis** adecuada. Dichas sintaxis está conformada por **múltiples elementos** que se explicarán a continuación.

[imagen] GraphQL Reasons. (s. f.). [PNG]. Coder Society. https://cdn.codersociety.com/uploads/graphql-reasons.png

Sintaxis: Tipo de petición

Inicialmente, en el objeto externo de la **sentencia** se define el **tipo de petición** a realizar (**Query** o **Mutation**), junto con el **nombre** de la operación, seguido por unos corchetes ({}), donde se define el cuerpo de la sentencia.

El nombre de la operación solo se utiliza para describir el propósito de la petición, por lo cual, si se desea **se puede omitir**.

```
query User {

}
```

El futuro digital es de todos

MinTIC

Sintaxis: Peticiones

Posteriormente, en el **cuerpo** de la sentencia se debe incluir al menos una **petición definida** en el **servidor** GraphQL, que **coincida** con el **tipo de operación** indicada en el objeto externo.

Toda petición se especifica utilizando su **nombre**, seguido de la definición de sus **argumentos**, y finalizando por la descripción de los **campos** que se desean obtener de la respuesta del servidor, encerrados en corchetes ({ }).

```
query User {
    userDetailById (userId: 1) {
        id
        username
        password
        name
        email
    }
}
```

UNIVERSIDAD NACIONAL DE COLOMBIA

MisiónTIC 2022

Sintaxis: Variables

En un cliente GraphQL es posible definir los **argumentos** de las peticiones utilizando **variables**. Dónde y cómo se definen las variables, **depende del cliente GraphQL**, sin embargo, la sintaxis para la asignación de variables no varía. Se **recibe** la variable como un **argumento** del objeto externo de la sentencia, definiendo el **tipo de dato** de la variable, y luego se **asigna** la variable al parámetro de la petición.

```
query User ($userDetailByIdUserId: Int!) {
    userDetailById (userId: $userDetailByIdUserId) {
        id
        username
        password
        name
        email
    }
}
```