# **¿Qué es GraphQL?**

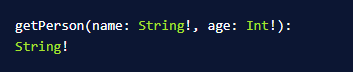
GraphQL es creado por facebook (actualmente pertenece a GraphQL Fundation) y nace de la necesidad de trasmitir datos desde las aplicaciones móviles que tiene facebook, necesitaban un protocolo mucho más solido para poder comunicarse con mucha más flexibilidad.

# **Sistema de tipado**

* En GraphQL todas las peticiones van por POST.
* En GraphQL todas las peticiones tienen como respuestas 201.

Tipos de datos

* **Query**: Para consultas de datos, es decir, obtener datos. Ej en REST: GET
  + **Por defecto** los parametros de la query son opcionales. Si queremos que sean obligatorios sus types deben tener un signo **!**



* **Mutations**: Para modificación de datos. Ej en REST: PUT, PATCH, POST y DELETE.
* **Subscriptions**: Consultas de datos en tiempo real.

Resolver los tipos de datos

* **Resolvers**: Se conectan a la base de datos y responden la información.

# **Scalar types**

GraphQL nos ofrece los siguientes tipos de datos:

* **Int**
* **Float**
* **String**
* **Boolean**
* **ID**: Este tipo de datos es especial y se escribe así, todo en mayúsculas. Es un String, pero normalmente representa un número único o a veces se suele utilizar como sistema de cache.

El servidor de GraphQL tiene validaciones automáticas de tipos, esto quiere decir que, valida los datos de la respuesta y los datos de las solicitudes del cliente.

# **Listas y non-null types**

## **Non-null**

Para validar que un dato no sea null, hacemos lo siguiente:

* **Non-null:** Type\_value**!**

El signo **!** le indica al servidor GraphQL que la respuesta no puede ser null. Lo mismo sucede con las solicitudes del cliente.

## **Listas**

Para devolver una lista en la respuesta, hacemos:

* [String]
* [Int]
* [Float]
* ... Y así con todos los tipos.

Para no permitir que el cliente mande un null como lista, hacemos:



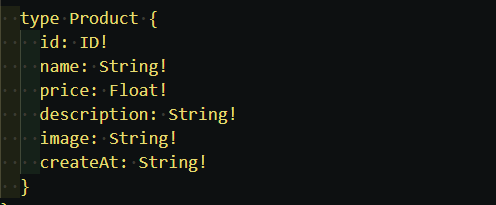
Para no permitir que el cliente envié valores en null dentro del array o null directamente.



# **Object types and Fields**

Para crear nuestros propios tipos de objetos, dentro de nuestros **typeDefs** hacemos:

Ejemplo de creación de tipo **Product**.



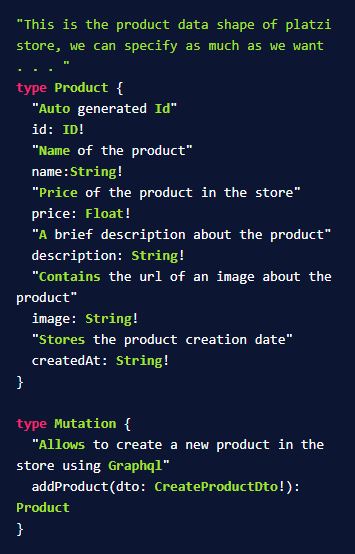
Dentro de los **resolvers**, hacemos nuestra función:



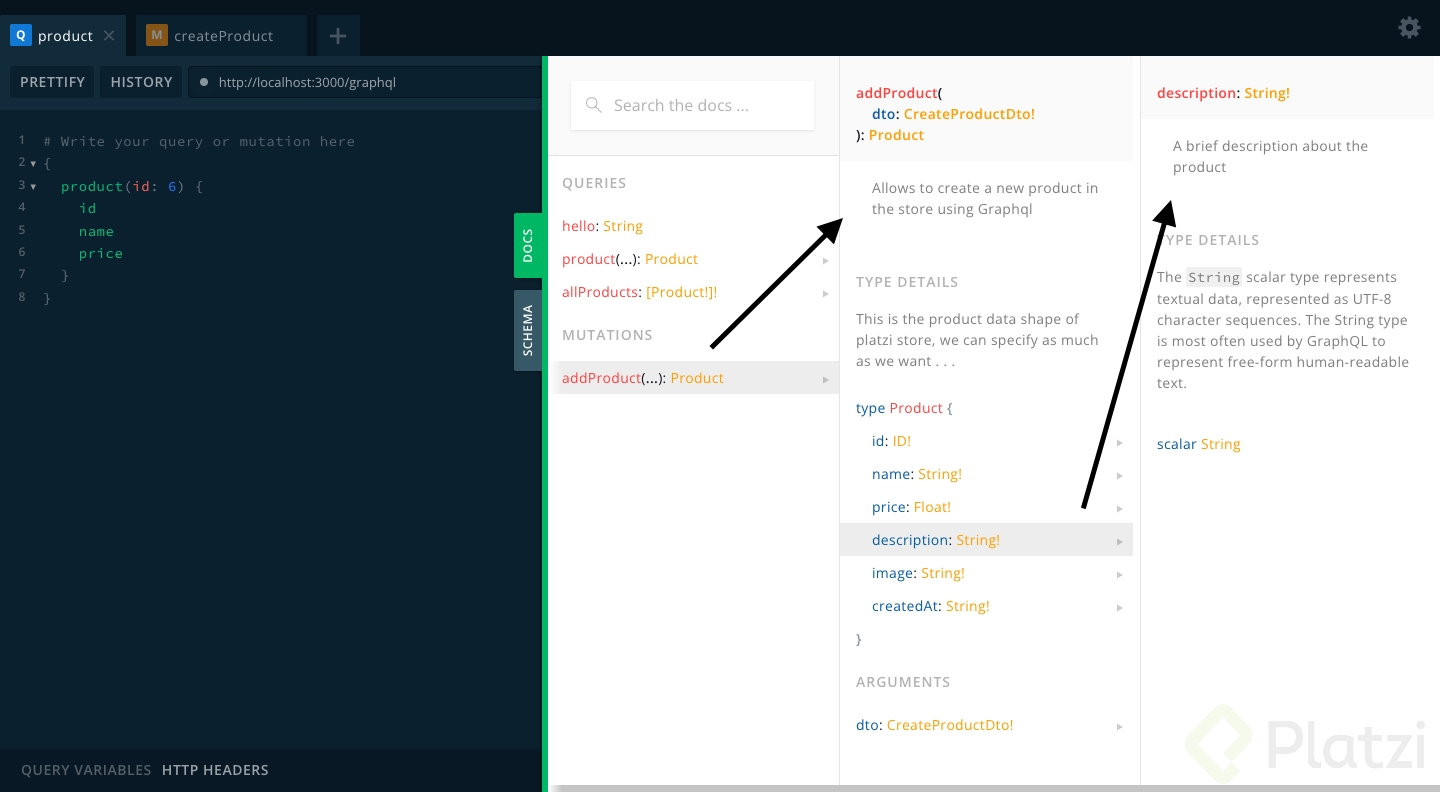
# **Archivos .graphql (mejor developer experience)**

Graphql además de auto generar la documentación de cada tipo y de tipar sus datos, nos permite agregar una descripcion de cada campo e incluso una descripcion para cada tipo o mutación!

Basta con solo sobre el campo, el tipo o mutación entre comillas dobles escribir nuestras descripciones.



Y esto se vería reflejado de esta manera en nuestro playground de Graphql:



# **Diferencias entre GraphQL vs. API REST**

|  |  |
| --- | --- |
| Rest API | GraphQL |
| El control de los datos: Lo tiene únicamente el Server. | **El control de los datos**: Lo tiene **Cliente**. Es decir, el que el cliente es quién decide que datos obtener y cuales no. |
| Retorna datos innecesarios: Esto puede ser un problema en app móviles. | El cliente sólo solicita lo que realmente requiera para su funcionamiento. |
| El número de solicitudes es mucho mayor, por ej: Sí se quiere consumir 2 estados de información al mismo tiempo, se deberá realizar 2 request distintos. | **Con una sola solicitud** se puede obtener todo lo que el cliente desee. |
| Existen muchos endpoints. | Todo se envía a un solo endpoint. |
| Hay dependencias de equipos, es decir, si el equipo de frontend necesita un endpoint en específico, entonces el equipo de backend debe desarrollarlo de forma personalizada generando esto, perdidas de tiempos. | **Hay independencias de equipos**, es decir, por ej: Si el equipo móvil necesita un endpoint muy especifico con ciertos datos, pero el equipo web necesita otro endpoint con otros datos, esto es irrelevante, porque GraphQL le das la facilidad a todos los equipos por igual.  Existe un costo, el cual es, los desarrolladores de frontend les toca aprender como consumir una API GraphQL, pero una vez tomada esa curva de aprendizaje, la independencia de equipos ya es más alta. |