**GraphQL Subscription**

[GraphQL](https://graphql.org/) offers the concept of *subscriptions*.

When a client *subscribes* to an event, it will initiate and hold a steady connection to the server. Whenever that particular event then actually happens, the server pushes the corresponding data to the client. Unlike queries and mutations that follow a typical “*request-response*-cycle”, subscriptions represent a *stream* of data sent over to the client.

Subscriptions are written using the same syntax as queries and mutations. Here’s an example where we subscribe on events happening on the Person type:

subscription {

newPerson {

name

age

}

}

After a client sent this subscription to a server, a connection is opened between them. Then, whenever a new mutation is performed that creates a new Person, the server sends the information about this person over to the client:

{

"newPerson": {

"name": "Jane",

"age": 23

}

}

**Defining subscriptions in the server side (.NET Core)**

Subscriptions are supported through the use of the [IObservalbe<T>](https://docs.microsoft.com/en-us/dotnet/api/system.iobservable-1?view=netframework-4.7.2). We will need a server that supports a Subscription protocol. The GraphQL Server project provides a .NET Core server that implements the Apollo GraphQL subscription protocol.

In order to implement graphql subscriptions, we need to use graphql web sockets and it should be initialized in the *Startup.cs* class

In *ConfigureService* method –



In *Configure* method –



Now we have to define the subscription schema.

Here’s an example for schema definition for subscription



The graphql subscriptions are specified by using *AddField* like *Field* in queries and mutations.

*Type* – Specifies the return type of the subscription

*Resolver* – Specifies the resolver for the subscription. Whenever a new message is added (corresponding mutation takes place), the resolver method will execute.

*Subscriber* – Specifies the subscriber method. When the client subscribes, this method will execute. It will open the connection between server and client. This method will return the [IObservalbe<T>](https://docs.microsoft.com/en-us/dotnet/api/system.iobservable-1?view=netframework-4.7.2) (i.e. the data stream).

**The service class**

In the service we maintain a [ReplaySubject<T>](https://docs.microsoft.com/en-us/previous-versions/dotnet/reactive-extensions/hh211810(v%3Dvs.103)) which is a stream to store data (This stream is returned in the subscribe method mentioned above) and also a [ConcurrentStack<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentstack-1?view=netframework-4.7.2) to store all events.

We define a method(*AddEvent()* in this example) which pushes the data to these streams and call this method when a mutation has taken place.

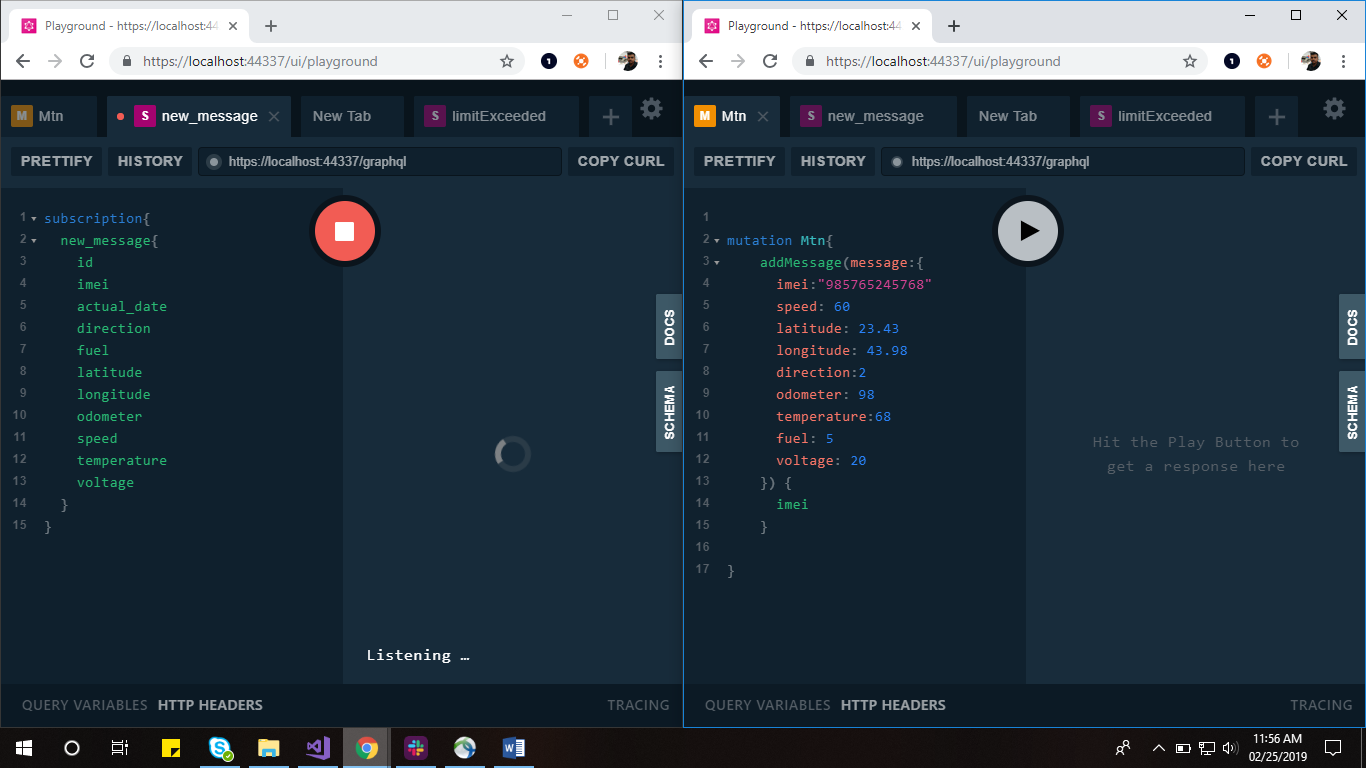
An example for implementation of these are shown below

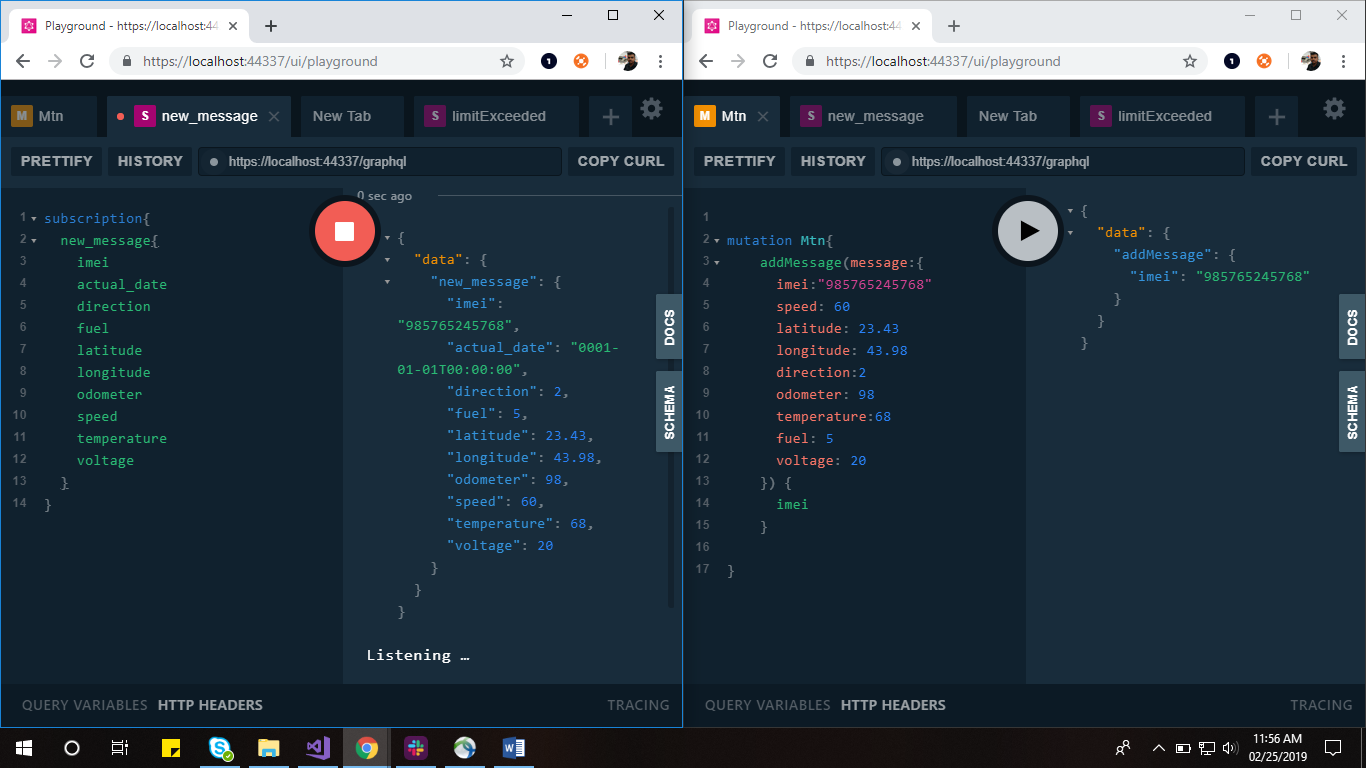


As mentioned above, the AddEvent method is called when the mutation has taken place.

Like this –

The working for the above schema will look like this.





**Using GraphQL from Client side (Angular)**

To use GraphQL API from an Angular application, we use Apollo-angular.

Apollo Client is the ultra-flexible, community driven GraphQL client for Angular, JavaScript, and native platforms. It is designed from the ground up to make it easy to build UI components that fetch data with GraphQL.

To start using Apollo angular, we need to install some packages.



apollo-client: Where the magic happens

apollo-angular: Bridge between Angular and Apollo Client

apollo-cache-inmemory: Our recommended cache

apollo-angular-link-http: An Apollo Link for remote data fetching

graphql: Second most important package

graphql-tag: Parses your strings to GraphQL documents

After installing the packages, we need to add the web socket Uri in the *graphql.module.ts* file to use GraphQL subscriptions.

The format for the Uri is





Since we may use Query, Mutation and subscriptions in the client side, we have to specify the normal http Uri for Query and Mutation.



So we should use the web socket Uri only if the operation is ‘Subscription’. Else the graphql operations should be carried out using the http Uri.

The logic will be something like this –



**Using subscription in the code**

The subscription is carried out using the Apollo object as in the case with queries and mutations.

First we need to inject the Apollo object in the component class



We use the *subscribe* method to call a subscription

The syntax is –





An example is -

