GraphQL

## Introduction

This document covers

Introduction

## DotNet Examples

### Simple Examples

Consider the following POCO class that we would want to expose in a query

public class Person

{

public int Age { get; set; }

public string FirstName { get; set; }

}

We now create a graph type that describes objects of type Person.

public class PersonType : ObjectGraphType<Person>

{

public PersonType()

{

Field(x=>x.Age);

Field(x=>x.FirstName);

}

}

We add a query that returns a person

public class PersonQuery : ObjectGraphType

{

public PersonQuery()

{

Field<PersonType>( "getPerson",

resolve: ctx => new Person() {

Age = 21,

FirstName = "Dave"

});

}

}

Now we need a schema that has a single top-level query.

var schema = new Schema {Query = new PersonQuery()};

Now we execute a query on the schema

var json = await schema.ExecuteAsync(\_ =>

{

\_.Query = @"

{

getPerson {

age,

firstName,

}

}";

\_.Root = schema;

});

The result is then given by

{  
  "data": {  
    "getPerson": {  
      "age": 21,  
      "firstName": "Dave"  
    }  
  }  
}

### List Examples

Let us extend our example from the previous section to include a second query that return multiple people.

public class PersonQuery : ObjectGraphType

{

public PersonQuery()

{

Field<PersonType>( "getPerson",

resolve: ctx => new Person() {

Age = 21,

FirstName = "Dave"

});

Field<ListGraphType<PersonType>>( "getPeople",

resolve: ctx => new List<Person>

{

new Person {Age=21, FirstName="John"},

new Person {Age=45, FirstName="Ken"},

});

}

}

Our query becomes

getPeople {

age,

firstName

}

Our result is then

{  
  "data": {  
    "getPeople": [  
      {  
        "age": 21,  
        "firstName": "John"  
      },  
      {  
        "age": 45,  
        "firstName": "Ken"  
      }  
    ]  
  }  
}

### Arguments

We can add arguments to queries

public class PersonQuery : ObjectGraphType

{

private List<Person> \_people = new List<UserQuery.Person>()

{

new Person {Age=21, FirstName="John"},

new Person {Age=45, FirstName="Ken"},

};

public PersonQuery()

{

Field<PersonType>( "getPerson",

arguments: new QueryArguments(

new QueryArgument<StringGraphType> {Name="name"}),

resolve: ctx =>

\_people.FirstOrDefault(x =>

x.FirstName == (string)ctx.Arguments["name"])

);

Field<ListGraphType<PersonType>>( "getPeople",

resolve: ctx => new List<Person>

{

new Person {Age=21, FirstName="John"},

new Person {Age=45, FirstName="Ken"},

});

}

}

Our query becomes

getPerson(name:""John"") {

age,

firstName,

}

And our result becomes

{  
  "data": {  
    "getPerson": {  
      "age": 21,  
      "firstName": "John"  
    }  
  }  
}