Progress NEXT 2018

The API Debate: GraphQL vs. OData

Ken Beutler - Principal Product Manager

Brody Messmer - Senior Manager, Software Engineering

May 30, 2018

Agenda

- Introduction to GraphQL and OData
- Capabilities
  - Use Cases
  - Comparison
- Differences Between GraphQL and OData
  - Simplicity
  - Data Types
  - Metadata
  - Versioning / Schema Evolution
  - Pagination
  - Other / Advanced Features
  - Community
- Examples
- Q&A

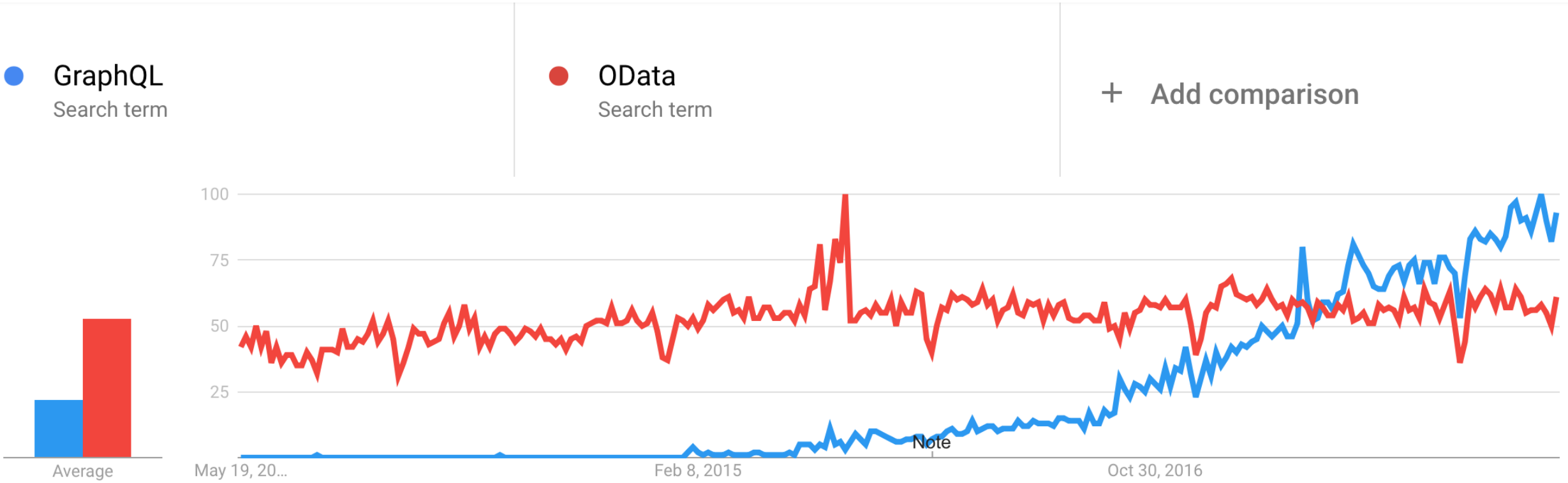
Progress NEXT 2018

Introduction to GraphQL and OData

Overview

OData	GraphQL
Initially developed by Microsoft in 2007, OData (Open Data Protocol) is an OASIS standard that defines the best practice for building and consuming RESTful APIs.	Developed by Facebook in 2012, GraphQL is a query language for APIs and a server-side runtime for fulfilling those queries by using a type system you define for your data.

Popularity



Progress NEXT 2018

Capabilities Use Case

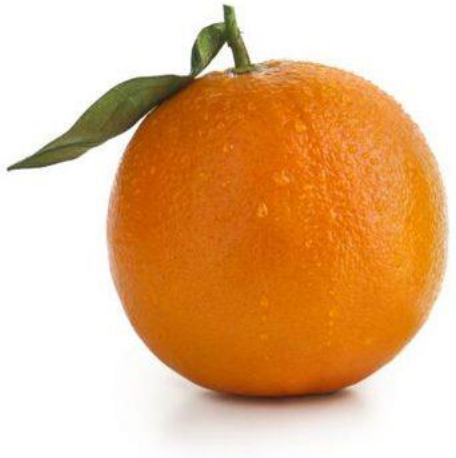
OData Use Cases

- Open Analytics
- Explosion of SaaS applications
- IT Modernization to the cloud
- Hybrid Data Pipeline

GraphQL Use Cases

- GitHub's v4 of their API was entirely refactored to leverage GraphQL to solve:
  - Scalability and performance
  - The collection of metadata about their endpoints

Hammers, Apples and Oranges



Progress NEXT 2018

Differences Between GraphQL and OData

Simplicity

Open Data Protocol

1 Introduction

The Open Data Protocol (OData) enables the creation of REST-based data services which allow resources, identified using Uniform Resource Locators (URLs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages. This specification defines the core semantics and the behavioral aspects of the protocol.

The **[OData-URL]** specification defines a set of rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query options.

The **[OData-CSDLJSON]** specification defines a JSON representation of the entity data model exposed by an OData service.

The **[OData-CSDLXML]** specification defines an XML representation of the entity data model exposed by an OData service.

The **[OData-JSON]** document specifies the JSON format of the resource representations that are exchanged using OData.

1.0 IPR Policy

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **[RFC2119]**.

1.2 Normative References

[OData-ABNF]	*OData ABNF Construction Rules Version 4.01.* See link in "Additional artifacts" section on cover page.
[OData-Aggregation]	*OData Extension for Data Aggregation Version 4.0.* See link in "Related work" section on cover page.
[OData-CSDLJSON]	*OData Common Schema Definition Language (CSDL) JSON Representation Version 4.01.* See link in "Related work" section on cover page.
[OData-CSDLXML]	*OData Common Schema Definition Language (CSDL) XML Representation Version 4.01.* See link in "Related work" section on cover page
[OData-JSON]	*OData JSON Format Version 4.01.* See link in "Related work" section on cover page.
[OData-URL]	*OData Version 4.01 Part 2: URL Conventions.* See link in "Additional artifacts" section on cover page.
[OData-VocCap]	*OData Vocabularies Version 4.0: Capabilities Vocabulary.* See link in "Related work" section on cover page.
[OData-VocCore]	*OData Vocabularies Version 4.0: Core Vocabulary.* See link in "Related work" section on cover page.
[RFC2046]	Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996. https://tools.ietf.org/html/rfc2046.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. https://tools.ietf.org/html/rfc2119.
[RFC3987]	Duerst, M. and, M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005. https://tools.ietf.org/html/rfc3987.
[RFC5789]	Dusseault, L., and J. Snell, "Patch Method for HTTP", RFC 5789, March 2010. http://tools.ietf.org/html/rfc5789.
[RFC7230]	Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014. https://tools.ietf.org/html/rfc7230.

GraphQL

GraphQL Learn Code Community Blog Spec Search docs...

Queries and Mutations

On this page, you'll learn in detail about how to query a GraphQL server.

Fields

At its simplest, GraphQL is about asking for specific fields on objects. Let's start by looking at a very simple query and the result we get when we run it:

```
{
  hero {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

You can see immediately that the query has exactly the same shape as the result. This is essential to GraphQL, because you always get back what you expect, and the server knows exactly what fields the client is asking for.

The field `name` returns a `String` type, in this case the name of the main hero of Star Wars, `"R2-D2"`.

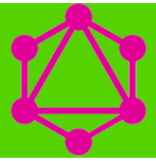
> Oh, one more thing - the query above is *interactive*. That means you can change it as you like and see the new result. Try adding an `appearsIn` field to the `hero` object in the query, and see the new result.

LEARN

- Introduction
- Queries and Mutations
  - Fields
  - Arguments
  - Aliases
  - Fragments
  - Operation Name
  - Variables
  - Directives
  - Mutations
  - Inline Fragments
- Schemas and Types
  - Type System
  - Type Language
  - Object Types and Fields
  - Arguments
  - The Query and Mutation Types
  - Scalar Types
  - Enumeration Types
  - Lists and Non-Null
  - Interfaces
  - Union Types
  - Input Types
- Validation
- Execution
- Introspection

Data Types

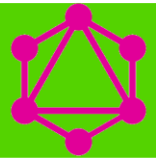
Many use cases don’t require a wide variety of types, so limiting the types supported keeps things simple…until you're forced to jump through hoops to support a non-native type.

 Open Data Protocol	 GraphQL
33 Simple Types Defined	String, Int, Float, Boolean, and custom • Lacks Decimal (Currency as Float/String) • Lacks Date/Timestamp
Supports complex types	Supports complex types
Weak data typing	Strong data typing

Metadata

Allows generic applications to be written by allowing them to:

- Discover Data Available (Catalog Metadata)
- Discover Capabilities and Expected Behavior

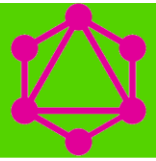
 Open Data Protocol	 GraphQL
Rich catalog metadata API • Everything returned in a single call	Introspection provides rich catalog metadata • Nested data requires iterative introspection
Rich capability reporting with clearly defined behaviors	Introspection provides insight into query capabilities, but naming convention and behaviors are not defined.

Subset of OData Query Options

System Query Option	SQL Construct	Description
$filter	WHERE clause	Restrict the entities returned when querying an entity set to those matching the filter criteria
$select	SELECT list	Specify the properties to be included in the returned entities
$orderby	ORDER BY clause	Specify the sorting of the returned entities
$expand	INNER JOIN	Include related entities and complex types nested in the returned entities
$count	COUNT(*)	Include the count of the number of entities returned in the result
$search	Full Text Search	Restrict the entities returned when querying an entity set to those matching the search expression
$top and $skip	TOP/SKIP and LIMIT/OFFSET	Enable client to page through results
$format	N/A	Specify the desired data format for the response

Versioning / Schema Evolution

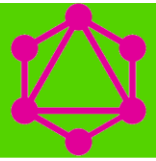
Schemas evolve rapidly in our world of Agile development. Our tools must allow for this evolution to occur while avoiding breaking changes from deprecating fields.

 Open Data Protocol	 GraphQL
Well defined protocol and model versioning within the OData spec.	Apps/Clients must request the exact data they want • Easy to identify who's using what (or what's not getting used) • Promotes efficient code @Deprecated annotation with comments returned via introspection. Deprecated fields can still be requested, thus avoiding breaking changes. Versioning available, but discouraged

Pagination

Most APIs require the ability to page data to avoid returning millions of rows of data.

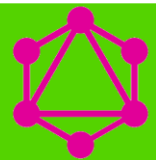
Most applications only have a need of displaying a subset of data at a given time.

 Open Data Protocol	 GraphQL
Consistent behavior that is either client or server driven: Next links with maxpagesize setting as well as $top and $skip.	Implementation specific and not easily discoverable

Other/Advanced Features

Feature	 Open Data Protocol	 GraphQL
Response Caching	HTTP, 3rd Party OData centric, Build Your Own	3rd Party GraphQL centric(GraphQL Client), Build Your Own
Transactions	Yes – “Change Sets”	No (some say “not needed”)
Delta Response (CDC)	Yes	No
Subscriptions / Callbacks	Yes – Callbacks	Yes – Subscriptions
Writing Data	Yes - HTTP POST, PUT, PATCH or DELETE operations utilized with data included in the payload.	Yes – Called “Mutations” Mutations can return data modified. All mutations utilize POST.
Primary Keys	Very Flexible (Composite Keys)	Single “ID” Field
Data Formats	JSON, XML	JSON

Community

	 Open Data Protocol	 GraphQL
Open Source Community	Yes	Yes
Server Libraries	.NET, Java, JavaScript, Python	.NET, Clojure, Elixir, Erlang, Go, Groovy, Java, JavaScript, PHP, Python, Scala, Ruby
Client Libraries	.NET, C++, Java, JavaScript, Python, Tcl/Tk	.NET, Go, Java, JavaScript, Python, Swift/Objective-C
	http://www.odata.org/libraries/	http://graphql.org/code/

Progress NEXT 2018

Examples

Retrieving Metadata (XML) – OData

http://localhost:5000/odata/$metadata

```
<edmx:Edmx
  xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:DataServices>
<edmx:Edmx
  xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:DataServices>
    <Schema
      xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="ProgressNext.Models">
      <EntityType Name="Book">
        <Schema
          xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Default">
          <Key>
            <PropertyRef Name="Id" />
          </Key>
          <EntityContainer Name="Container">
            <EntitySet Name="Book" EntityType="ProgressNext.Models.Book">
              <NavigationPropertyBinding Path="Author" Target="Author" />
              <NavigationPropertyBinding Path="Publisher" Target="Publisher" />
            </EntitySet>
            <EntitySet Name="Author" EntityType="ProgressNext.Models.Author">
            <EntitySet Name="Publisher" EntityType="ProgressNext.Models.Publisher">
              <NavigationPropertyBinding Path="Books" Target="Book" />
            </EntitySet>
          </EntityContainer>
        </Schema>
      </edmx:DataServices>
    </EntityType>
</edmx:Edmx>
        <EntityType Name="Publisher">
          <Key>
            <PropertyRef Name="Id" />
          </Key>
          <Property Name="Id" Type="Edm.Int32" Nullable="false" />
          <Property Name="Name" Type="Edm.String" Nullable="false" />
          <NavigationProperty Name="Books" Type="Collection(ProgressNext.Models.Book)" />
        </EntityType>
      </Schema>
```

Retrieving Metadata (JSON) – GraphQL

http://localhost:4000/graphql?query=

```
{
  __type(name: "Author") {
    name
    fields {
      name
      type {
        name
        kind
      }
    }
  }
}
```


```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Query",
          "fields": [
            {
              "name": "Books"
            },
            {
              "name": "Book"
            },
            {
              "name": "Authors"
            },
            {
              "name": "Publishers"
            }
          ]
        },
```



```
{
  "data": {
    "__type": {
      "name": "Author",
      "fields": [
        {
          "name": "id",
          "type": {
            "name": "Int",
            "kind": "SCALAR"
          }
        },
        {
          "name": "name",
          "type": {
            "name": "String",
            "kind": "SCALAR"
          }
        },
        {
          "name": "Books",
          "type": {
            "name": null,
            "kind": "LIST"
          }
        }
      ]
    }
  }
}
```



```
{
  "name": "Author",
  "fields": [
    {
      "name": "id"
    },
    {
      "name": "name"
    },
    {
      "name": "Books"
    }
  ]
},
{
  "name": "Publisher",
  "fields": [
    {
      "name": "id"
    },
    {
      "name": "name"
    }
  ]
},
```

Standard Queries – OData

Get the book collection

http://localhost:5000/odata/Book

Get a book entity using an ID

http://localhost:5000/odata/Book(1)

Get a book entity using an ID and restrict results

http://localhost:5000/odata/Book(1)?$select=Id,Name


```
{
  "@odata.context": "http://localhost:5000/odata/$metadata#Book(Id,Name)/Sentity",
  "Id": 1,
  "Name": "Into Thin Air",
}"AuthorId": 4,
  "PublisherId": 2
}
```



```
{
 "@odata.context": "http://localhost:5000/odata/$metadata#Book",
 "value": [
  {
   "Id": 1,
   "Name": "Into Thin Air",
   "AuthorId": 4,
   "PublisherId": 2
  },
  {
   "Id": 2,
   "Name": "Into the Wild",
   "AuthorId": 4,
   "PublisherId": 2
  },
  {
   "Id": 3,
   "Name": "Under the Banner of Heaven: A Story of Violent Faith",
   "AuthorId": 4,
   "PublisherId": 2
  },
  {
   "Id": 4,
   "Name": "The Lost Symbol",
   "AuthorId": 1,
   "PublisherId": 2
  },
```

Standard Queries – GraphQL

http://localhost:4000/graphql?query=

```
{
  Books {
    id,
    name,
    authorId,
    publisherId
  }
}
```

```
{
  Book(bookId: 1) {
    id,
    name,
    authorId,
    publisherId
  }
}
```

```
{
  Book(bookId: 1) {
    id,
    name
  }
}
```

```
{
  "data": {
    "Books": [
      {
        "id": 1,
        "name": "Into Thin Air",
        "authorId": 4,
        "publisherId": 2
      },
      {
        "id": 2,
        "name": "Into the Wild",
        "authorId": 4,
        "publisherId": 2
      },
      {
        "id": 3,
        "name": "Under the Banner of Heaven: A Story of Violent Faith",
        "authorId": 4,
        "publisherId": 2
      },
      {
        "id": 4,
        "name": "The Lost Symbol",
        "authorId": 4,
        "publisherId": 2
      },
```

```
{
  "data": {
    "Book": {
      "id": 1,
      "name": "Into Thin Air",
      "authorId": 4,
      "publisherId": 2
    }
  }
}
```

Advanced Queries – OData

Retrieving related resources (join) inline with requested resource(s)

http://localhost:5000/odata/Author(3)?$**expand**=Books($**select**=Name)


```
{
  "@odata.context": "http://localhost:5000/odata/$metadata#Author",
  "value": [
    {
      "Id": 3,
      "Name": "J.K. Rowling",
      "Books@odata.context": "http://localhost:5000/odata/$metadata#Author(3)/Books",
      "Books": [
        {
          "Name": "Harry Potter and the Sorcerer's Stone"
        },
        {
          "Name": "Harry Potter And The Chamber Of Secrets"
        },
        {
          "Name": "Harry Potter and the Prisoner of Azkaban"
        },
        {
          "Name": "Harry Potter And The Goblet Of Fire"
        }
      ]
    }
  ]
}
```

Advanced Queries – GraphQL

http://localhost:4000/graphql?query=

```
{
  Authors {
    id,
    name,
    Books {
      name
    }
  }
}
```


```
{
  "data": {
    "Authors": [
      {
        "id": 3,
        "name": "J.K. Rowling",
        "Books": [
          {
            "name": "Harry Potter and the Sorcerer's Stone"
          },
          {
            "name": "Harry Potter And The Chamber Of Secrets"
          },
          {
            "name": "Harry Potter and the Prisoner of Azkaban"
          },
          {
            "name": "Harry Potter And The Goblet Of Fire"
          }
        ]
      }
    ]
  }
}
```

Ordering and Filtering – OData

Retrieving a resource using selection, ordering and filtering criteria

http://localhost:5000/odata/Author?$orderby=Name&$select=Name,%20Id&$filter=Id%20ne%203


```
{
 "@odata.context": "http://localhost:5000/odata/$metadata#Author(Name,Id)",
 "value": [
  {
   "Name": "Dan Brown",
   "Id": 1
  },
  {
   "Name": "Jon Krakauer",
   "Id": 4
  },
  {
   "Name": "Robert C. Martin",
   "Id": 2
  }
 ]
}
```

Ordering and Filtering – GraphQL

Retrieving a resource using selection, ordering and filtering criteria will require GraphQL schema and resolver changes

```
type Query {
  Books: [Book],
  Author(authorId: Int, orderBy: AuthorOrderByInput, field:String, fieldValue: Int, filterOp: String): [Authors],
  Authors: [Author],
  Publishers: [Publisher]
}
```

Request:

```
{
  Author(
    orderBy: name_ASC,
    field: "id",
    fieldValue: 3,
    filterOp: "ne")
  {
    name
    id
  }
}
```

Response:

```
{
  "data": {
    "Authors": [
      {
        "name": "Dan Brown",
        "id":  1
      },
      {
        "name": "Jon Krakauer",
        "id": 4
      },
      {
        "name": "Robert C. Martin",
        "id": 2
      }
    ]
  }
}
```

Writing Data (Mutations) – GraphQL

Requires GraphQL schema changes:

```
input BookInput {
  name: String
  authorId: Int
  publisherId: Int
}

type Mutation {
  createBook(name: String!, authorId: Int!, publisherId: Int!): Book
  updateBook(input:BookInput!): Book
  deleteBook(id:Int!): Boolean
}
```

Request:

```
mutation {
  createBook(
    name: "Harry Potter and the Deathly Hallows",
    authorId: 3,
    publisherId: 3)
  {
    id,
    name,
    authorId,
    publisherId
  }
}
```

Response:

```
{
  "data": {
    "Books": [
      {
        "id": 14,
        "name": "Harry Potter and the Deathly Hallows ",
        "authorId": 3,
        "publisherId": 3
      }
    ]
  }
}
```

Progress NEXT 2018

Q&A


Progress®