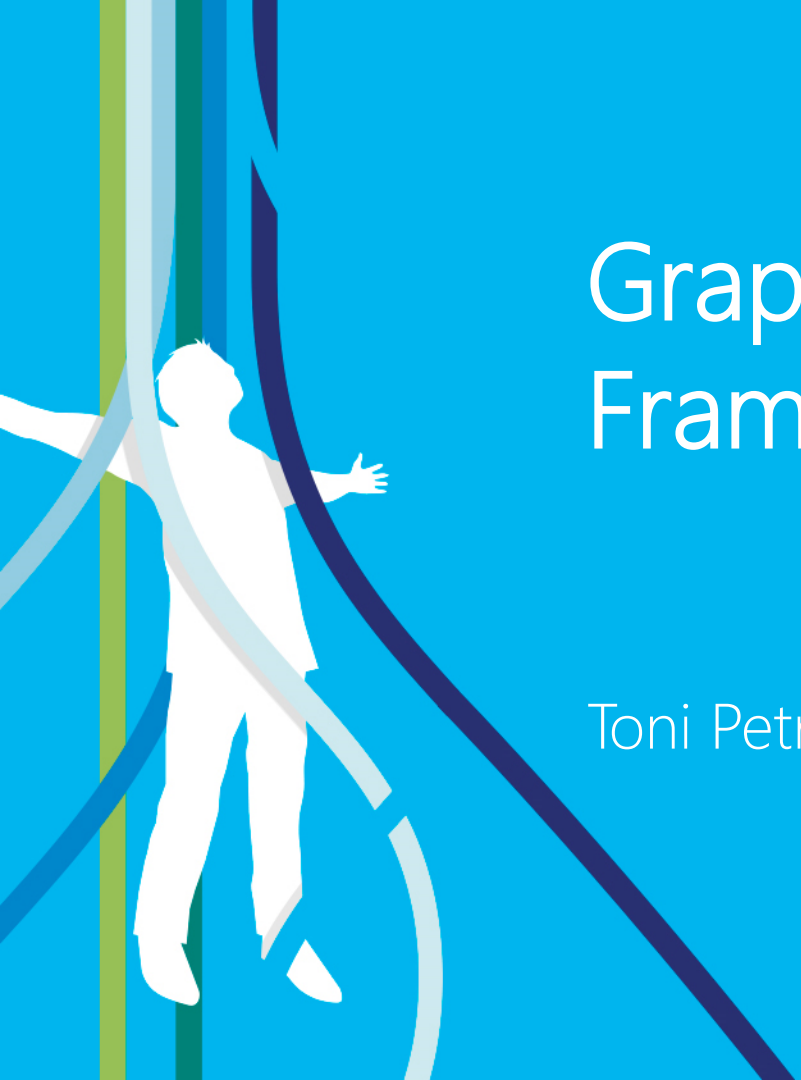

GraphQL with Entity Framework Core

Toni Petrina

Experts Live Netherlands

Agenda

- What is GraphQL
- GraphQL in action
  - **Features**
- Conjuction with EF
- Future work

Toni Petrina

Full stack dev, Microsoft MVP

Visma e-conomic a/s

@to_pe

https://github.com/tpetrina

What is GraphQL?

- A new way of building APIs with strong emphasis on graph-like querying
- Single endpoint (POST)
- Requires data modeling

AZURE

Model –> Ask –> Results

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

COMPAREX A SoftwareONE Company

KPN ICT Consulting Sterk in ICT-advies

Microsoft

AZURE

Model –> Ask –> Results

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Model

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Query

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

Result

COMPAREX
A SoftwareONE Company

KPN ICT Consulting
Sterk in ICT-advies

Microsoft

Connect to SQL

- We will use EF Core to resolve data
- Mapping of EF models to GraphQL
- Data loader
- Pitfalls
- Why EF? Because it supports SQL and NoSQL

Demo

Declaring global schema

```
public class Demo1Schema : Schema
{
  public Demo1Schema(Demo1Query query)
  {
    Query = query;
  }
}
```

How do we *get* data?

- Every defined type needs data resolvers
- Data resolvers can be anything

Fetching products

```
public class Demo1Query : ObjectGraphType {
 public Demo1Query(ApplicationDbContext db) {

  Field<ListGraphType<DemoProductType>>(
   "products",
   resolve: context =>
   {
    return db.Product.ToList();
   }
  );

 }
}
```

Fetching products

```
public class DemoProductType : ObjectGraphType<DemoProduct>
{
  public DemoProductType()
  {
    Field(i => i.Id);
    Field(i => i.Name);
  }
}

[Table("Product")]
public class DemoProduct
{
  [Key]
  public int Id { get; set; }

  public string Name { get; set; }
}
```

Fetching one product

```
public class Demo1Query : ObjectGraphType {
  public Demo1Query(ApplicationDbContext db) {
    Field<DemoProductType>(
      "product",
      arguments: new QueryArguments(new QueryArgument<NonNullGraphType<IntGraphType>> { Name =
"id" }),
      resolve: context => {
        var id = context.GetArgument<int>("id");
        return db.Product.FirstOrDefault(x => x.Id == id);
      }
    );
  }
}
```

Fetching products

```
Field<ListGraphType<DemoProductType>>("products",
 arguments: new QueryArguments(new QueryArgument<StringGraphType> { Name = "name" }),
 resolve: context =>
 {
  var nameFilter = context.GetArgument<string>("name");

  if (string.IsNullOrEmpty(nameFilter))
   return db.Product.ToList();

  return db.Product
    .Where(x => EF.Functions.Like(x.Name, nameFilter))
    .ToList();
 }
);
```

Powerful introspection

- Inspectable by default
- Allows tooling on top
  - Generate TypeScript files
  - GraphiQL

Mutations

- Another graph object
- Can return object – reuse graph query

Mutation

```
public class ProductMutation : ObjectGraphType
{
 public ProductMutation(ApplicationDbContext db)
 {
  Field<DemoProductType>(
   "createProduct",
   arguments: new QueryArguments(
    new QueryArgument<NonNullGraphType<CreateProductType>> {Name = "product" }
   ),
   resolve: context =>
   {
    var product = context.GetArgument<DemoProduct>("product");
    db.Product.Add(product);
    db.SaveChanges();
    return product;
   }
  );
 }
}
```

Mutation param

```
public class CreateProductType : InputObjectGraphType
{
  public CreateProductType()
  {
    Name = "CreateProduct";
    Field<NonNullGraphType<StringGraphType>>("name");
  }
}
```

Features

- Ask what you need
- One request for multiple resources
- Typed responses
- Evolving API and metrics
- GraphiQL

https://graphql.org

Demo - Northwind

What we have seen

- Mappings, simple and complex
- Deprecating fields
- Complexity
- One to many, many to many
- DataLoader pattern for optimizing N+1 queries

When do we want to use this?

- Unknown consumer patterns (public API)
  - **Underfetching (N+1), overfetching**
- Quickly evolving backend/frontend
- Microservices

GraphQL limitations

- Standardized endpoint (only 1)
- POST vs PUT vs PATCH
- Naming discussions
- Proliferation of special types
- Waiting for endpoint to be extended…

Demo – consume from frontend

Using Apollo

```
import ApolloClient, { gql } from 'apollo-boost';

const client = new ApolloClient({
  uri: 'https://localhost:5001/api/graphql',
});
```

```
client
  .query({ query: gql`
  {
    products {
      productName
    }
  }`})
  .then(result => setData(result.data));
```

Demo – Relay connections

...further features

- Authorization on field level
- Learn type system under the hood
  - **Union types**
  - **Enums**
  - **Split schema in multiple files**
- Generate simple GraphQL for CRUD
- Subscriptions
- Variables, fragments, more...

Pitfalls

- Authorization – can you access this?
- Errors – How to handle special cases?
- Overfetching on server side
  - DDOS waiting to happen?
  - Efficient translation?
  - Max depth?

Conclusion

Should you use it?

It depends!

See more

- Hasura https://blog.hasura.io/architecture-of-a-high-performance-graphql-to-sql-server-58d9944b8a87/
- https://graphql-dotnet.github.io
- https://graphql.org
- https://www.apollographql.com/docs/react/
- https://github.com/Dotnet-Boxed/Templates/blob/master/Docs/GraphQL.md

Questions?

Thank you!

Please review my session in the Yellenge App!

Experts Live Netherlands