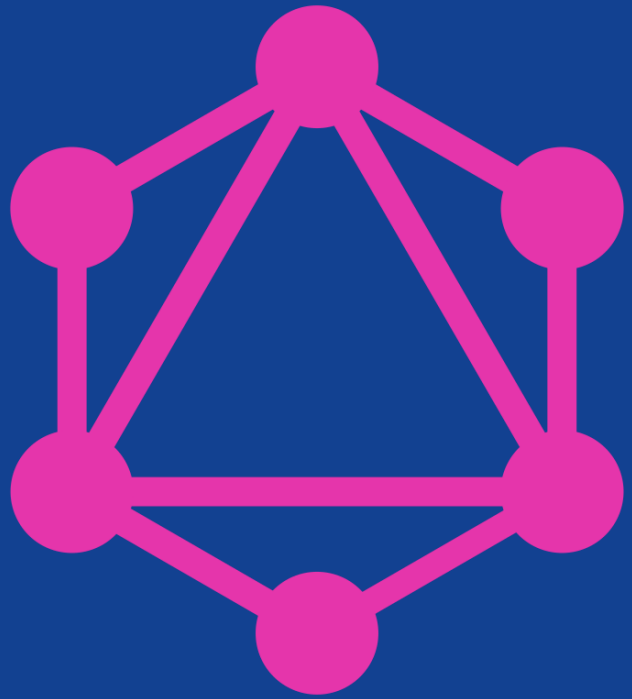
NOKIA

Coding-Corner



Introduction to GraphQL

- Pierre-Henri Symoneaux
- 28-03-2018

Introduction to GraphQL
A query language for your APIs

- A query language for APIs
- A runtime for fulfilling those queries with your existing data
- Already implemented in many languages
- Offers an aternative to REST APIs
- Developped by Facebook starting from 2012
- Published in 2015

Introduction to GraphQL
Why using graphQL

- Provides a complete and understandable description of the data in your API
- Gives clients the power to ask for exactly what they need and nothing more
- Makes it easier to evolve APIs over time
- Enables powerful developer tools
- Prevents under-fetching and over-fetching

Introduction to GraphQL
Ask for what you need

- The query contains the data you need
- Queries always return predictable results.
- Apps using GraphQL are fast and stable because they control the data they get, not the server.

```
query {
  Books {
    title
    year
  }
}
```


```
{
  "data": {
    "Books": [
      {
        "title": "A year at the wall",
        "year": 2018
      },
      {
        "title": "From basterd to king",
        "year": 2017
      },
      {
        "title": "Me and my dragons",
        "year": 2017
      },
      {
        "title": "I drink and I do things",
        "year": 2017
      },
      {
        "title": "How to get hated by the whole world",
        "year": 2015
      }
    ]
  }
}
```

Introduction to GraphQL

Get many ressources in a single request

- GraphQL queries smoothly follow references between resources
- Typical REST APIs require loading from multiple URLs
- GraphQL APIs get all the data your app needs in a single request
- Apps using GraphQL can be quick even on slow network connections.

```
query {
  BooksByTheme(theme: "war") {
    title
    year
    Author {
      fullname
      Friends {
        fullname
      }
    }
  }
}
```


```
{
  "data": {
    "BooksByTheme": [
      {
        "title": "From basterd to king",
        "year": 2017,
        "Author": {
          "fullname": "John Snow",
          "Friends": [
            {
              "fullname": "Daenerys Targaryen"
            },
            {
              "fullname": "Samwell Tarly"
            },
            {
              "fullname": "Tyrion Lannister"
            }
          ]
        }
      },
      {
        "title": "How to get hated by the whole world",
        "year": 2015,
        "Author": {
          "fullname": "Joffrey Baratheon",
          "Friends": []
        }
      }
    ]
  }
}
```

Introduction to GraphQL
A powerful type system

- APIs are organized in terms of types and fields, not endpoints
- Access the full capabilities of your data from a single endpoint
- Uses types to ensure Apps only ask for what's possible
- Provide clear and helpful errors

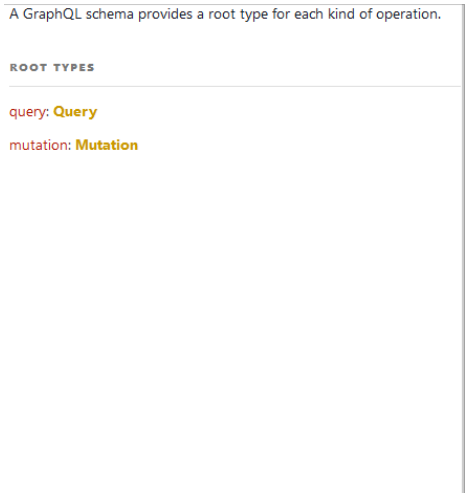
```
# A book
type Book {
  # Get the book author
  Author: Author!

  # The identification number
  id: Int!

  # The book's title
  title: String!

  # The year of publication
  year: Int!
}
```

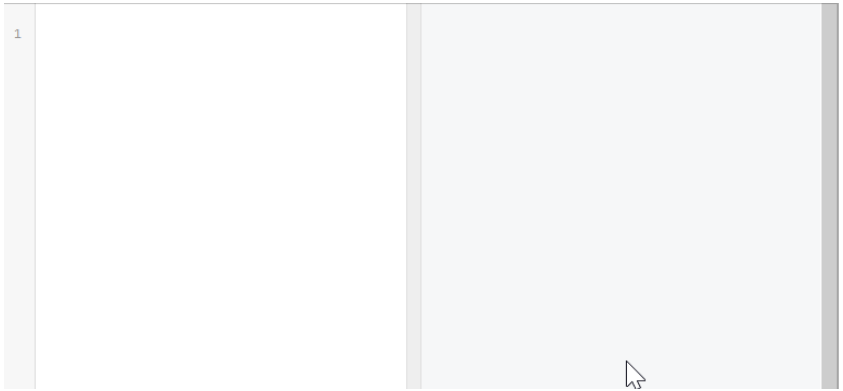
```
query {
  Books {
    name
  }
}
```

```
{
  "errors": [
    {
      "message": "Field 'name' doesn't exist on type 'Book'",
      "locations": [
        {
          "line": 3,
          "column": 5
        }
      ],
      "fields": [
        "query",
        "Books",
        "name"
      ]
    }
  ]
}
```

Introduction to GraphQL
Powerful developers tools

- GraphQL supports schema introspection
- Know exactly what data you can request from your API without leaving your editor
- Highlight potential issues before sending a query
- Take advantage of improved code intelligence





Introduction to GraphQL
Evolve your APIs

- Add new fields and types to your GraphQL API without impacting existing queries
- Aging fields can be deprecated and hidden from tools
- By using a single evolving version, GraphQL APIs give apps continuous access to new features and encourage cleaner, more maintainable server code.

Introduction to GraphQL
Create uniform APIs

- GraphQL creates a uniform API across your entire application without being limited by a specific storage engine

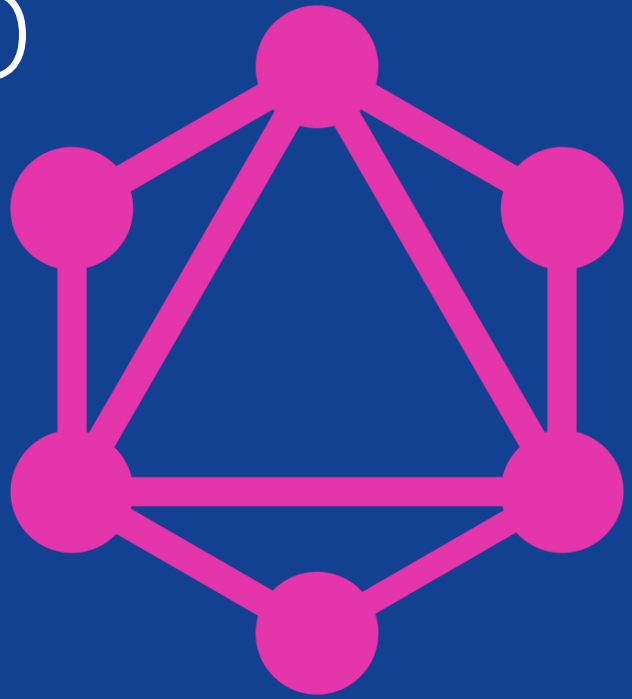
- Write GraphQL APIs that leverage your existing data and code with GraphQL engines available in many languages

- You provide functions for each field in the type system, and GraphQL calls them with optimal concurrency.

NOKIA

Introduction to GraphQL

Schema and types



Schema and types
Schema definition

- Interface is strongly typed
- Schema defined programmatically or in IDL

```
# A book writer
type Author {
  # All the books written
  Books: [Book!]!

  # List of related authors
  Friends: [Author!]!

  # The firstname
  firstname: String!

  # The fullname
  fullname: String!

  # The identification number
  id: Int!

  # The lastname
  lastname: String!
}
```

```
# A book
type Book {
  # Get the book author
  Author: Author!

  # The identification number
  id: Int!

  # The book's title
  title: String!

  # The year of publication
  year: Int!
}
```

```
Book = GraphQL::ObjectType.define do
       name "Book"
       description "A book"
       field :id, !types.Int, "The identification number“
       field :title, !types.String, "The book's title”
       field :year, !types.Int, "The year of publication“
       field :Author, !Author, "Get the book author“
end
```

Schema and types
Supported types

- Scalar types
  - Int
  - Float
  - String
  - Boolean
- Custom scalar types
- Lists
- Enumerations
- Unions
- Interfaces
- Objects
- Nullable & non-nullable fields

Schema and types
Schema introspection

- Schema can be discovered with graphQL queries

```
query {
  __schema {
    types {
      name
      kind
    }
  }
}
```

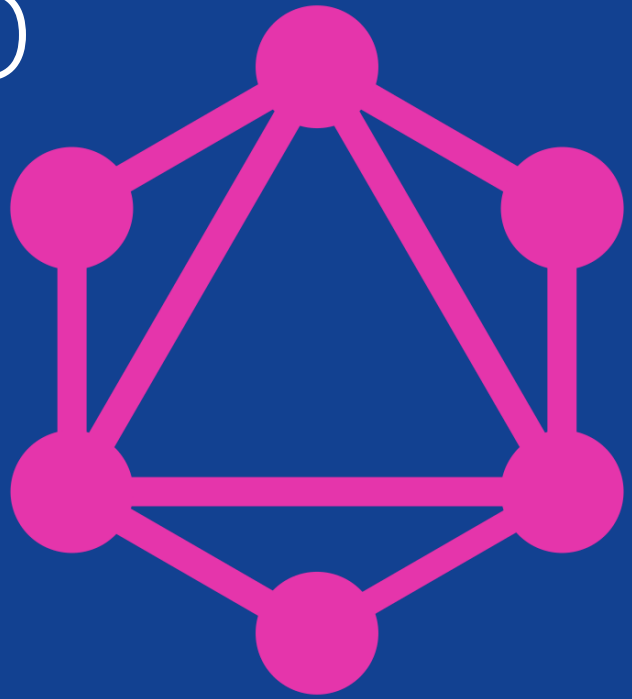
```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Boolean",
          "kind": "SCALAR"
        },
        {
          "name": "String",
          "kind": "SCALAR"
        },
```

- Mainly used by development tools

NOKIA

Introduction to GraphQL

Queries and mutations



Queries and mutations
Query or mutate your data

- Query: Read only requests

```
query {
  Books {
    title
    year
  }
}
```

- Mutation: Writes data

```
mutation {
  AddAuthor(firstname: "Brandon", lastname: "Stark") {
    id
    fullname
  }
}
```

```
{
  "data": {
    "AddAuthor": {
      "id": 5,
      "fullname": "Brandon Stark"
    }
  }
}
```

- Queries execution parrallelized
- Queries execution are sequential

Queries and mutations
Queries and Mutations

- Ask for object and fields

```
query {
  Books {
    title
    year
    Author {
     fullname
    }
  }
}
```

- Fields can accept arguments

```
query {
  BooksByTheme(theme: "war") {
    title
    year
  }
}
```

- Alias some fields

```
query {
  war: BooksByTheme(theme: "war") {
    title
    year
  }
  dragons: BooksByTheme(theme: "dragons") {
    title
    year
  }
}
```

```
{
  "data": {
    "war": [
      {
        "title": "From basterd to king",
        "year": 2017
      },
      {
        "title": "How to get hated by the whole world",
        "year": 2015
      }
    ],
    "dragons": [
      {
        "title": "Me and my dragons",
        "year": 2017
      }
    ]
  }
}
```


NOKIA

Queries and mutations
Queries and Mutations

- DRY, use fragments

```
fragment BookData on Book {
  title
  year
  Author {
    fullname
  }
}

query {
  war: BooksByTheme(theme: "war") {
    ...BookData
  }
  dragons: BooksByTheme(theme: "dragons") {
    ...BookData
  }
}
```

- Use variables

```
query($theme: String!) {
  BooksByTheme(theme: $theme) {
      title
      year
      Author {
        fullname
      }
  }
}
```

QUERY VARIABLES

```
{
  "theme": "war"
}
```

```
{
  "data": {
    "BooksByTheme": [
      {
        "title": "From basterd to king",
        "year": 2017,
        "Author": {
          "fullname": "John Snow"
        }
      },
      {
        "title": "How to get hated by the whole world",
        "year": 2015,
        "Author": {
          "fullname": "Joffrey Baratheon"
        }
      }
    ]
  }
}
```


NOKIA