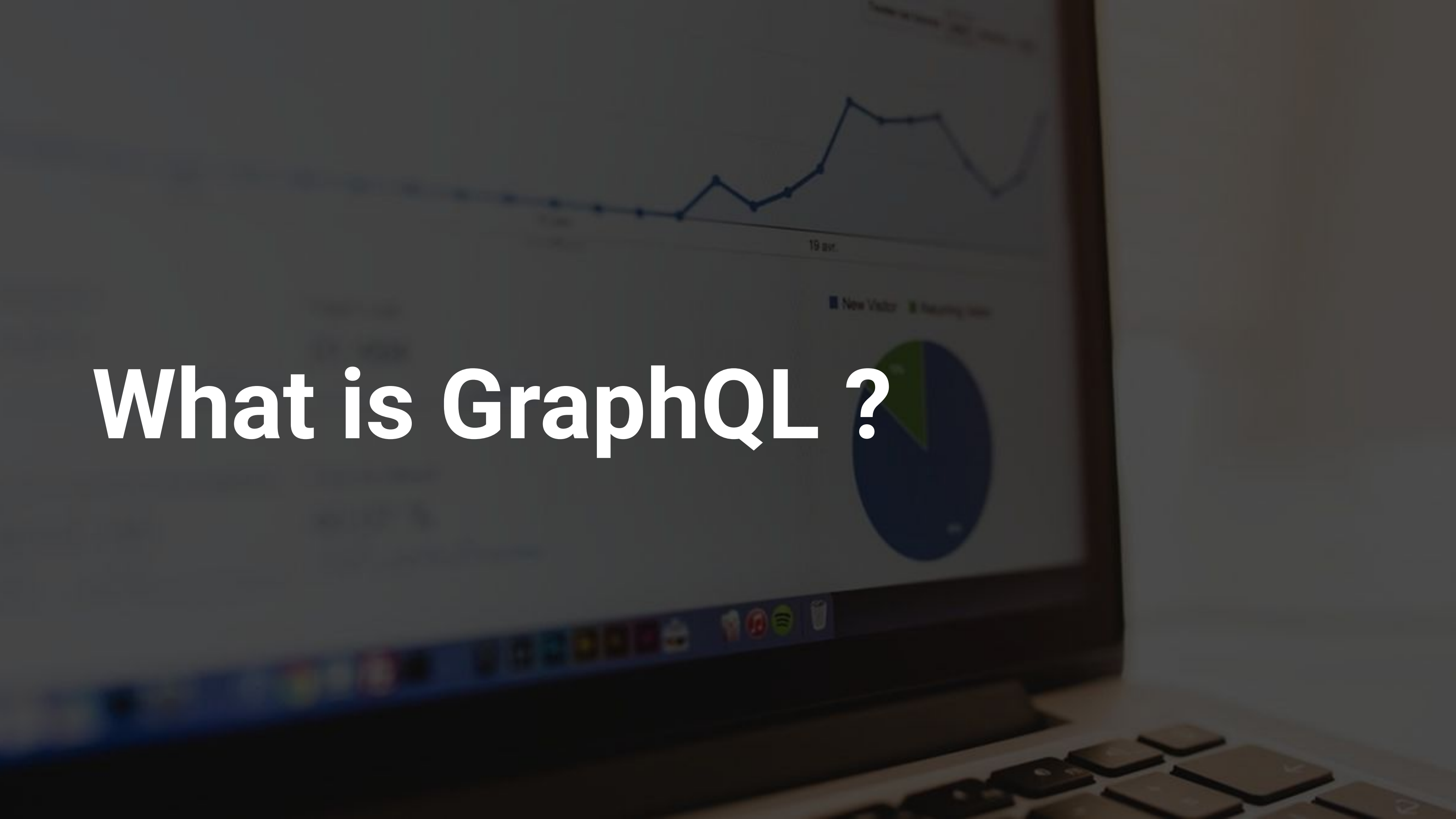
GraphQL

The smarter way to request data from your API



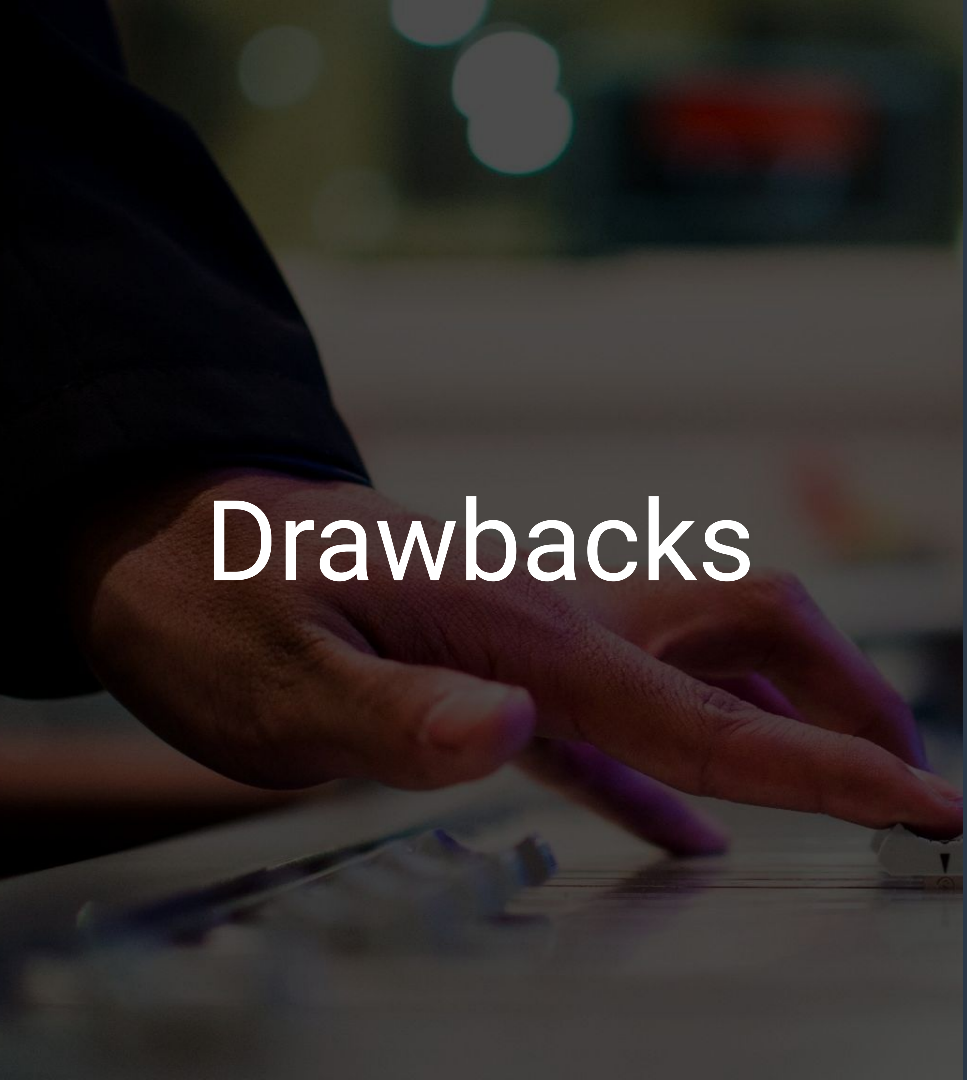
What is GraphQL ?

Introduction

- Created by facebook in 2012
- Open source since 2015
- Clients define data they want
- Single endpoint

Benefits

1. Ask for what you need, get exactly that
2. Get many resources in a single request
3. Describe what’s possible with a type system
4. Evolve your API without version
5. Improved performance



Drawbacks

1. Does not support file uploading
2. Caching mechanism to be optimized by developer
3. Security is harder to implement
4. Higher latency

Use cases

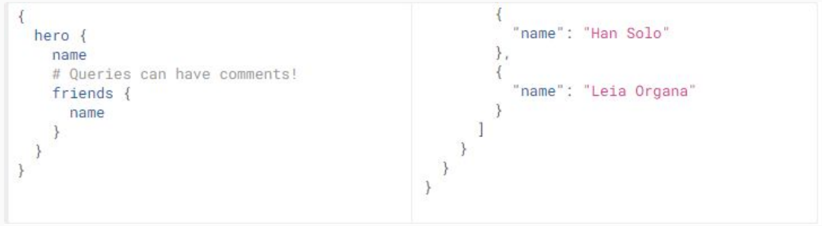
- Simplify complex systems
- No need for file upload
- Reducing app bandwidth usage (useful for mobile)
- Microservice based architectures

Schema definition language

Fields & Arguments

Fields

Specify only the fields you want

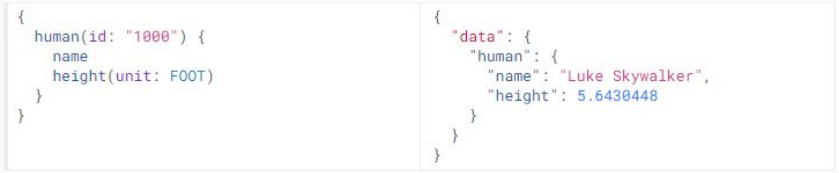


```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

```
            {
              "name": "Han Solo"
            },
            {
              "name": "Leia Organa"
            }
          ]
        }
      }
    }
```

Arguments

Every fields and object can have arguments. Can be used for search criteria and for data transformation



```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

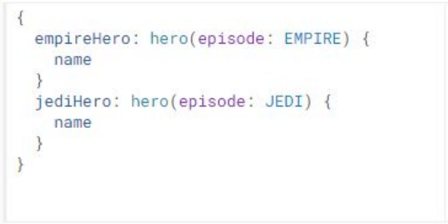
```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

Schema definition language

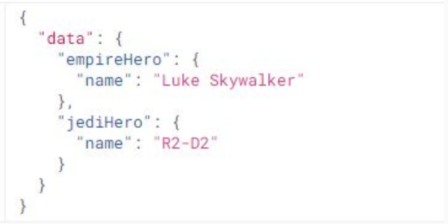
Alias & Fragments

Alias

Use it to make multiple request with different arguments



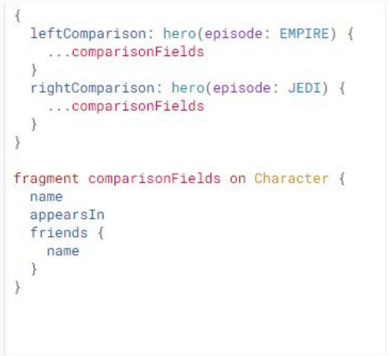
```
{
  empireHero: hero(episode: EMPIRE) {
    name
  }
  jediHero: hero(episode: JEDI) {
    name
  }
}
```



```
{
  "data": {
    "empireHero": {
      "name": "Luke Skywalker"
    },
    "jediHero": {
      "name": "R2-D2"
    }
  }
}
```

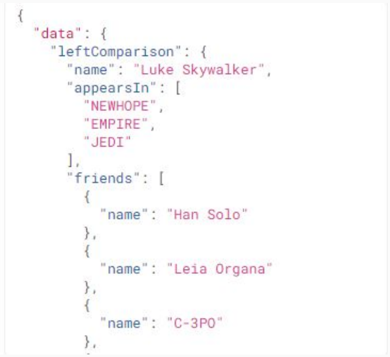
Fragments

Define fragment to be used in multiple queries



```
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```



```
{
  "data": {
    "leftComparison": {
      "name": "Luke Skywalker",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
        },
```

Schema definition language

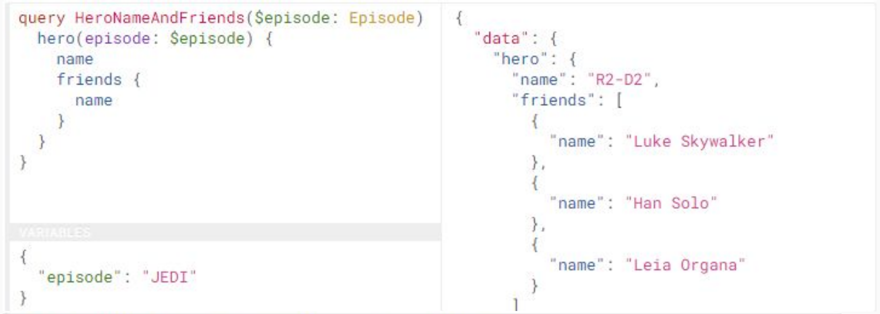
Operation names & Variables

Operation names

You can specify the operation: query, mutation or subscription. By default, it’s query

Variables

Replace static part of a query. Can have a default value

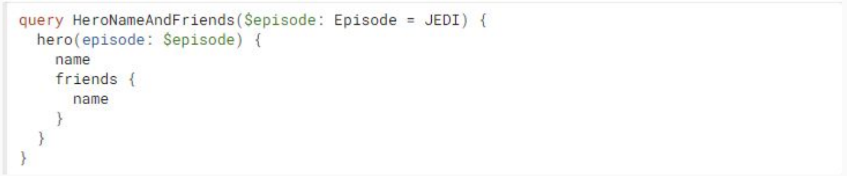


```
query HeroNameAndFriends($episode: Episode)
  hero(episode: $episode) {
    name
    friends {
      name
    }
  }
}
```

VARIABLES

```
{
  "episode": "JEDI"
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
```



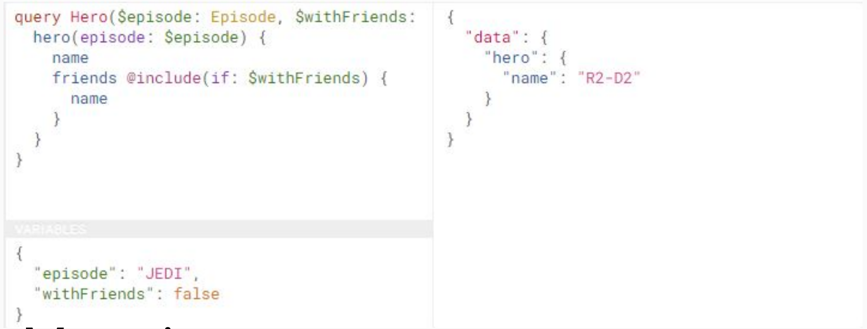
```
query HeroNameAndFriends($episode: Episode = JEDI) {
  hero(episode: $episode) {
    name
    friends {
      name
    }
  }
}
```

Schema definition language

Directives & Mutations

Directives

Enable more dynamic queries. Can be @include(if: boolean) or @skip(if: boolean)



```
query Hero($episode: Episode, $withFriends:
  hero(episode: $episode) {
    name
    friends @include(if: $withFriends) {
      name
    }
  }
}
```

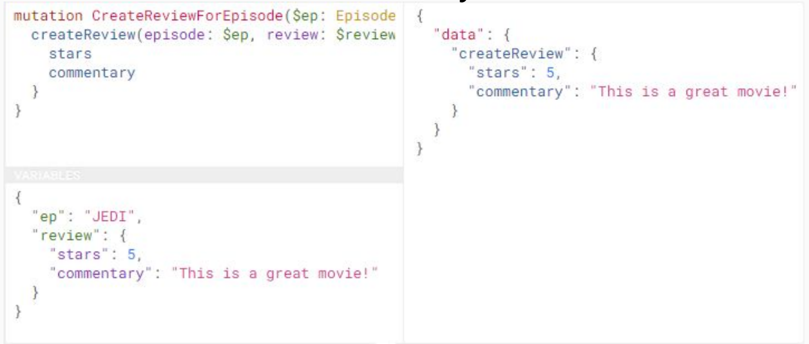
VARIABLES

```
{
  "episode": "JEDI",
  "withFriends": false
}
```

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

Mutations

Can modify data, will return only specified fields of modified data. Unlike Query, Mutation run in series



```
mutation CreateReviewForEpisode($ep: Episode
  createReview(episode: $ep, review: $review
    stars
    commentary
  }
}
```

VARIABLES

```
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

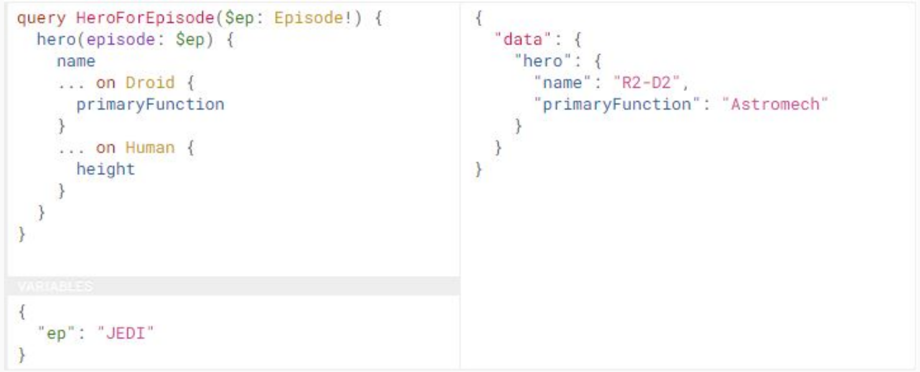
```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

Schema definition language

Inline fragments

Inline fragments

Can define interfaces and union types, you can access data from underlying concrete type



```
query HeroForEpisode($ep: Episode!) {
  hero(episode: $ep) {
    name
    ... on Droid {
      primaryFunction
    }
    ... on Human {
      height
    }
  }
}
```

```
{
  "ep": "JEDI"
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "primaryFunction": "Astromech"
    }
  }
}
```

Tools and libraries

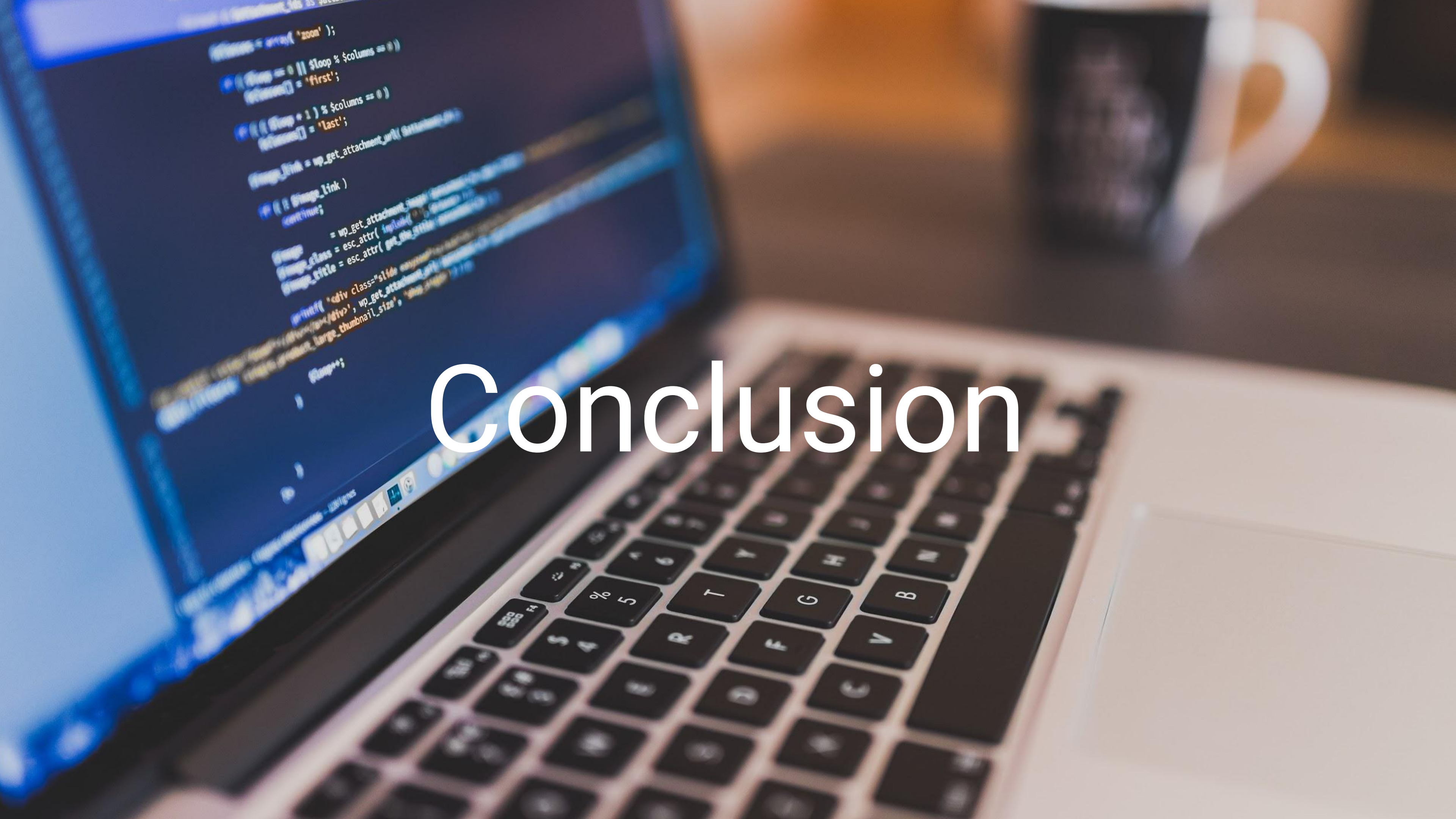
Graph*i*QL : A graphical interactive in-browser GraphQL IDE

Apollo : Library for both frontend and backend. Provide cloud solution for both

Relay : Frontend library to make scalable graphQL client

Prisma : Used with server library such as Apollo, Express, NestJS or Mercurius to connect directly to database

Questions?



Conclusion

References

GraphQL foundation: https://graphql.org/learn/queries

Apollo : https://www.apollographql.com

Prisma : https://www.prisma.io/graphql

Fauna : https://fauna.com/blog/what-is-graphql-use-cases-applications-and-databases