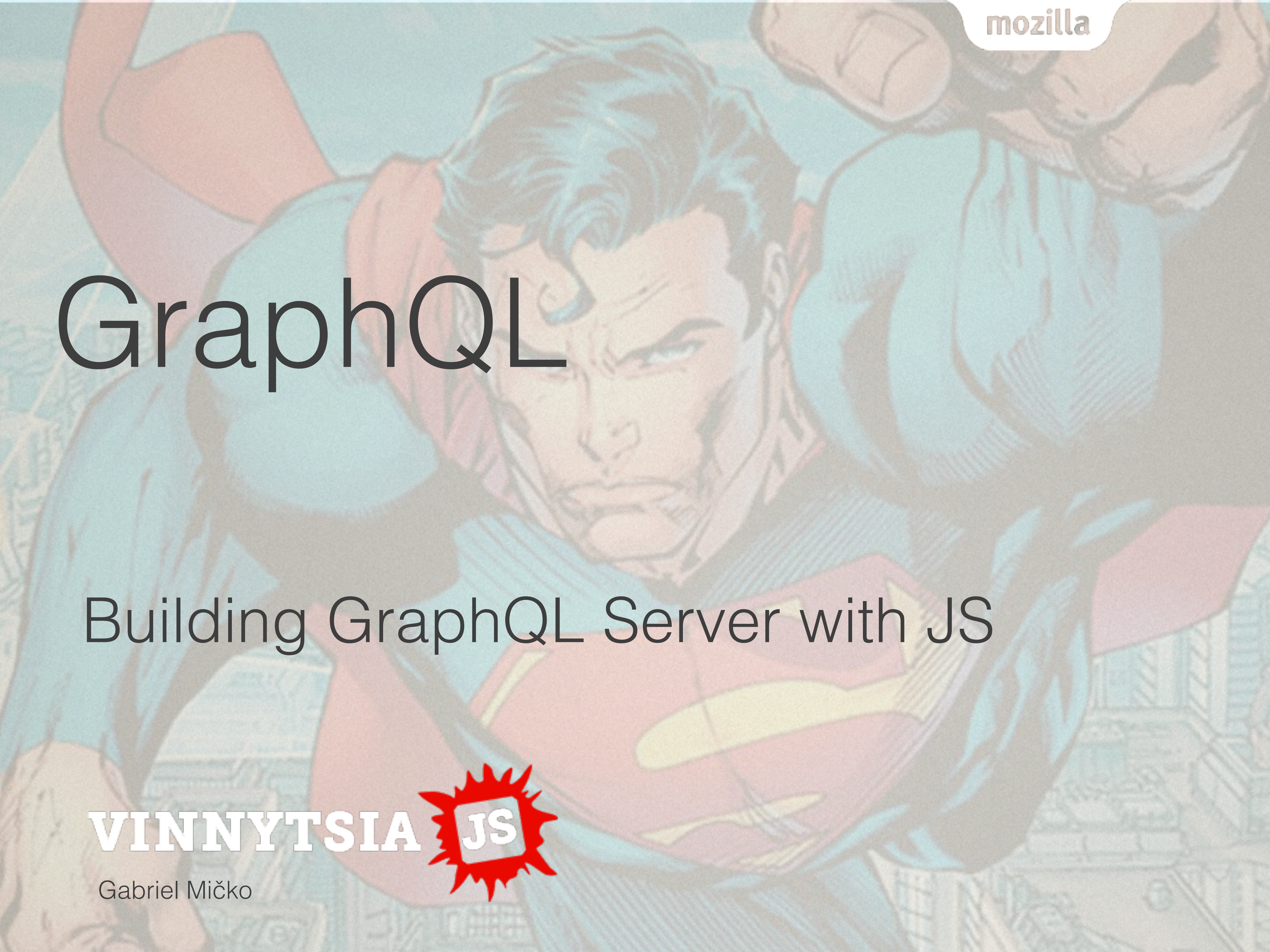


mozilla

GraphQL

Building GraphQL Server with JS

VINNYTSIA JS

Gabriel Mičko

About me 📖

Gabriel Mičko

Quid Freelance Software Engineer

moz://a Tech Speaker, Volunteer

@gabriel_micko

“We don't write client-server anymore.
We write self-sufficient applications.”

by Nikita Prokopov
@nikitonsky

ReactiveConf

Backbone

React

Elm

Knockout

Let’s choose one.

Cycle

Ember

Angular

Vue

Webpack

Meteor

Backbone

https://youtu.be/mVVNJKv9esE?t=21m

REST API

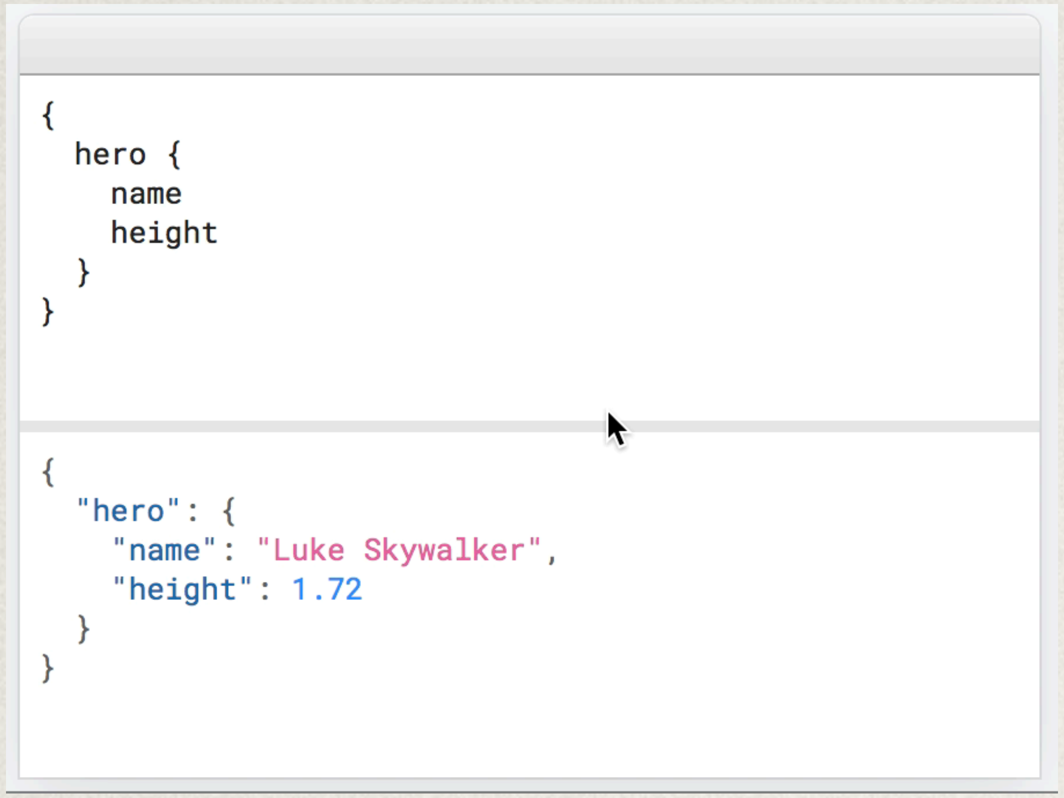
Let’s find out

GraphQL

Syntax

Query

Result

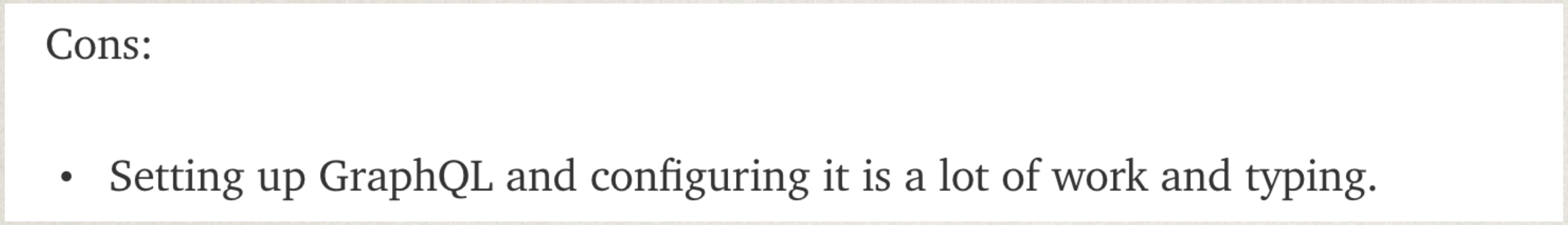


```
{
  hero {
    name
    height
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker",
    "height": 1.72
  }
}
```

Let’s find out

GraphQL cons ☔



Cons:

- Setting up GraphQL and configuring it is a lot of work and typing.

GraphQL pros ⚡

- Less requests
- More optimisable queries
- Less data trough network
- JS everywhere
- Graphiql
- Developer experience
- Knowing the data structure
- Type safety

Getting started 🐣

- http://graphql.org/community/
- https://graphqlweekly.com/
- https://twitter.com/GraphQL

Let’s work 🔨

- Develop Vinnytsia JS GraphQL API
- Pair up
- 7 steps in 14 branches
- Each step contains a task, the description is in the README.md file, search for “TASK X”
- Each step is covered by unit tests
- Task is ready when tests are passing

https://bit.ly/2Muu3zG

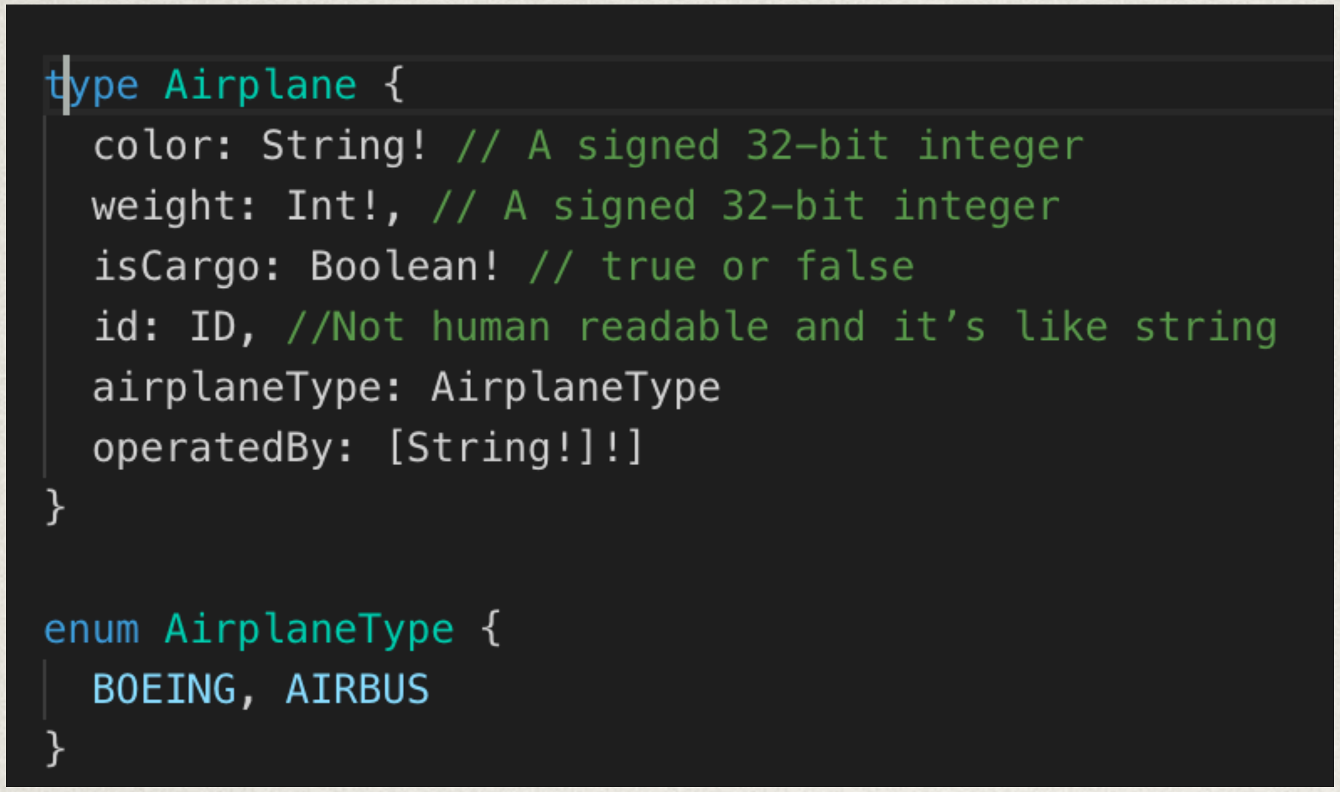
Step 1

Data resolvers

Step 2

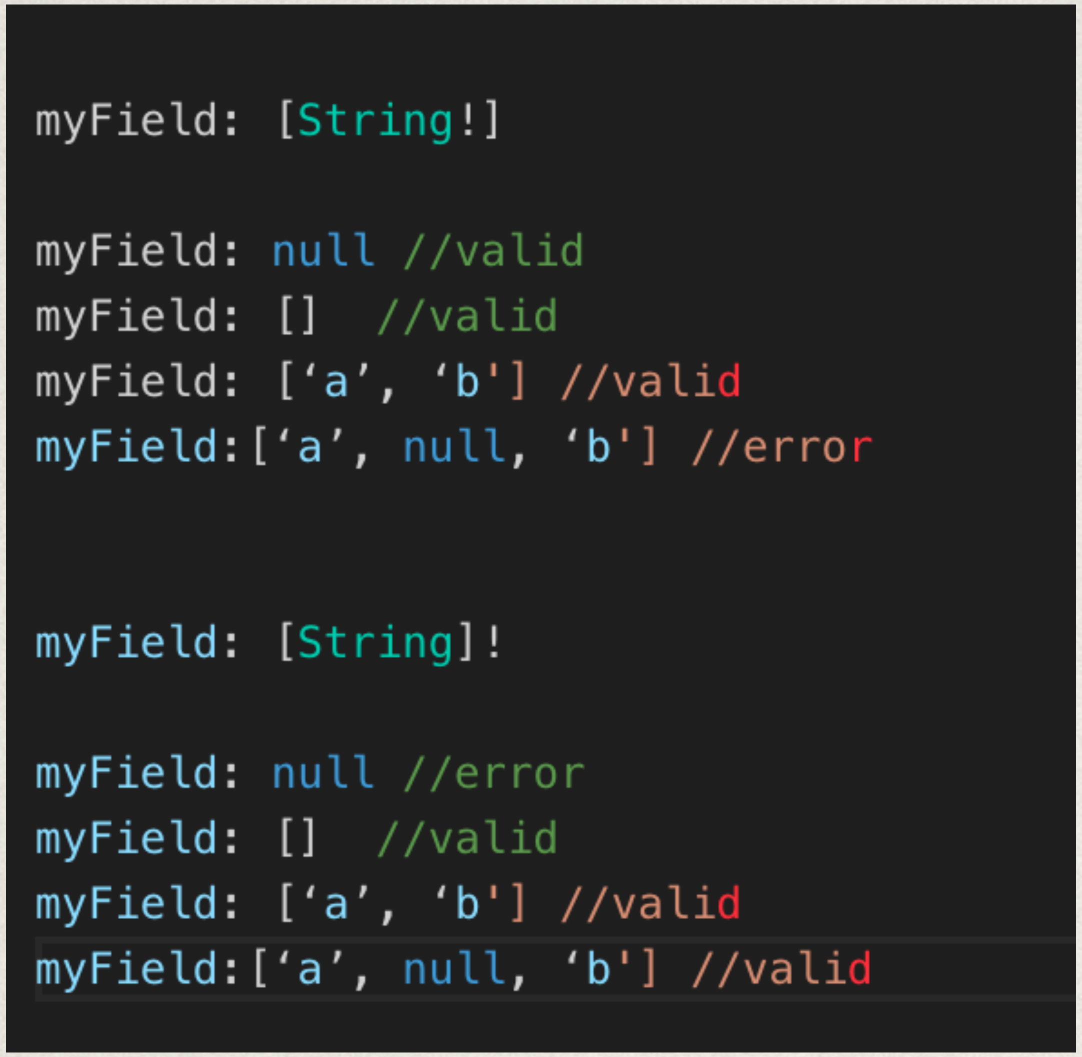
- Package: graphql-tools
- Package: express-graphql
- Schema
- Query
- Resolvers

Step 2 - Schema definition



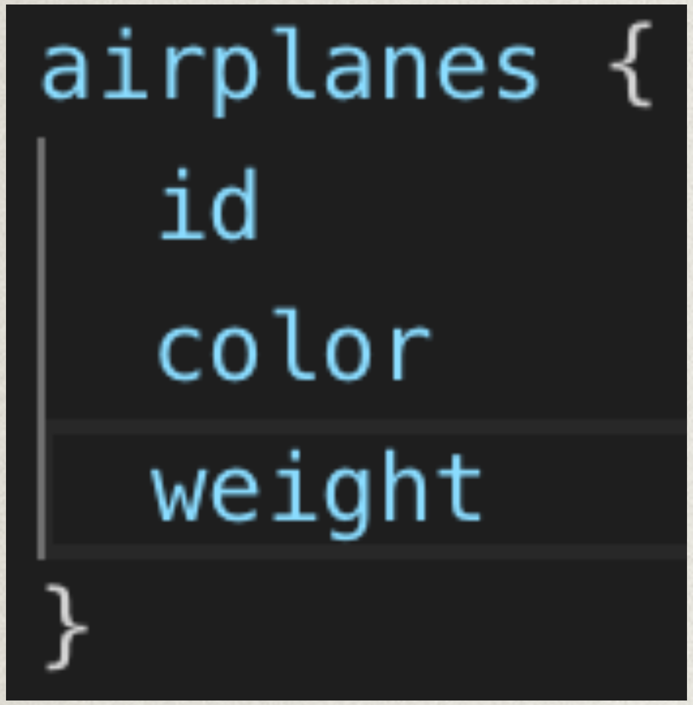
```
type Airplane {
  color: String! // A signed 32-bit integer
  weight: Int!, // A signed 32-bit integer
  isCargo: Boolean! // true or false
  id: ID, //Not human readable and it’s like string
  airplaneType: AirplaneType
  operatedBy: [String!]!]
}

enum AirplaneType {
  BOEING, AIRBUS
}
```

Step 2 - ! exclamation mark

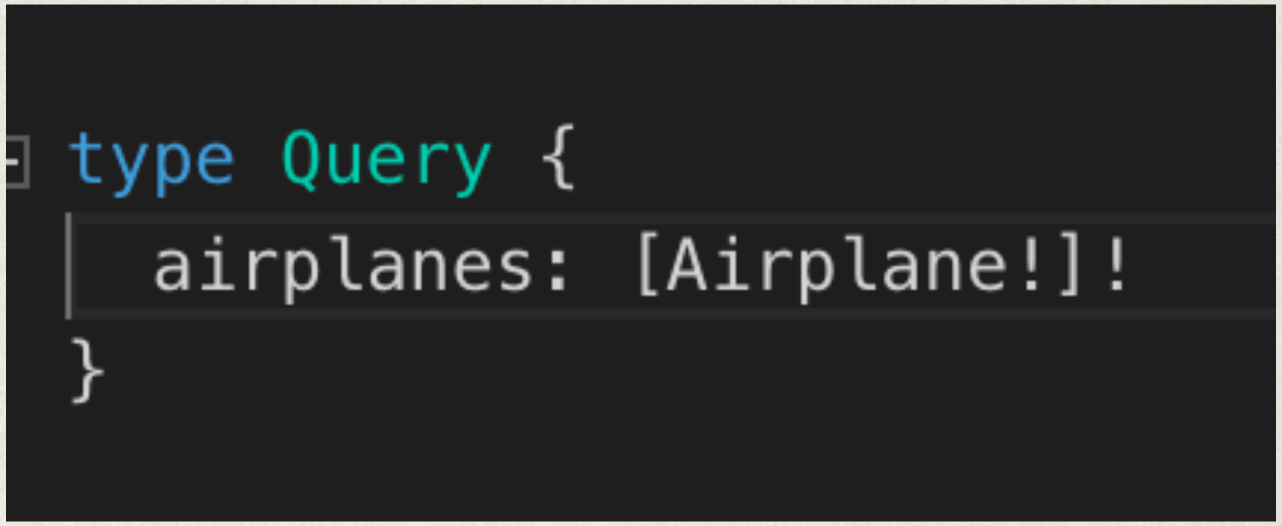


```
myField: [String!]

myField: null //valid
myField: []  //valid
myField: [‘a’, ‘b'] //valid
myField:[‘a’, null, ‘b'] //error


myField: [String]!

myField: null //error
myField: []  //valid
myField: [‘a’, ‘b'] //valid
myField:[‘a’, null, ‘b'] //valid
```

Step 2 - Query definition

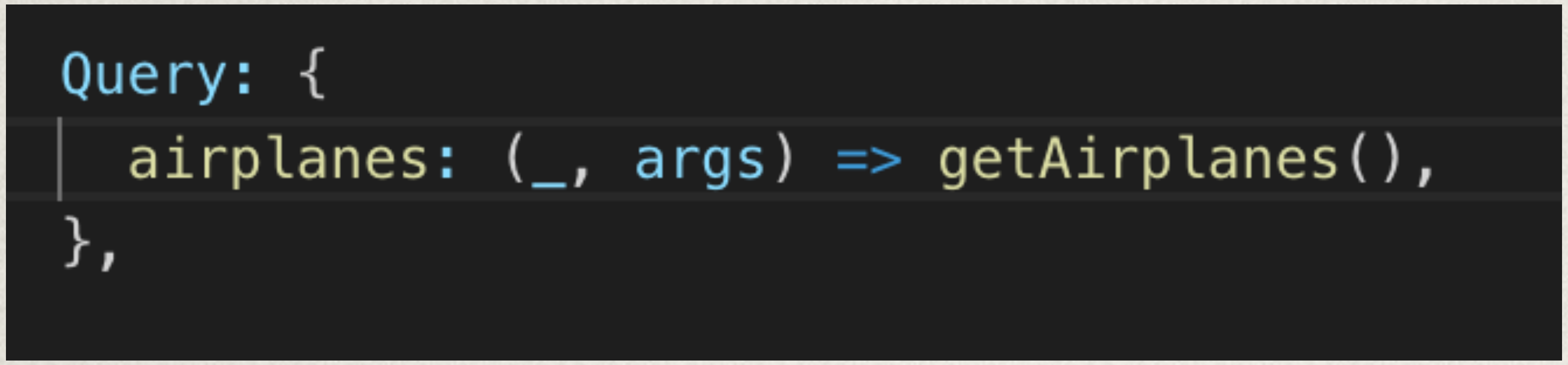


```
airplanes {
  id
  color
  weight
}
```



```
type Query {
  airplanes: [Airplane!]!
}
```

Step 2 - Resolvers

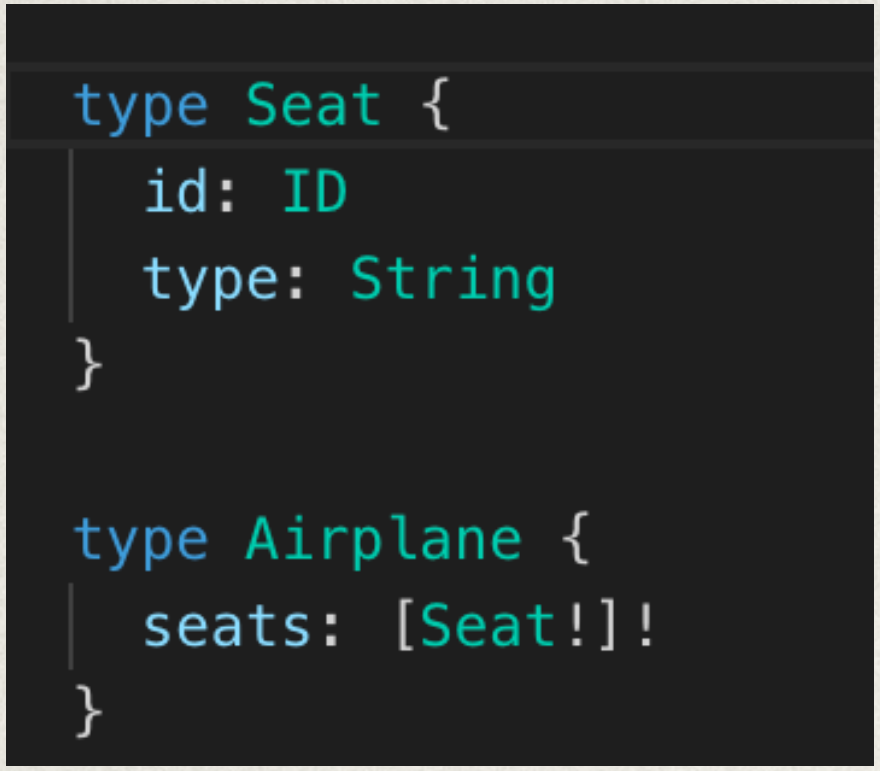


```
Query: {
  airplanes: (_, args) => getAirplanes(),
},
```

Step 3

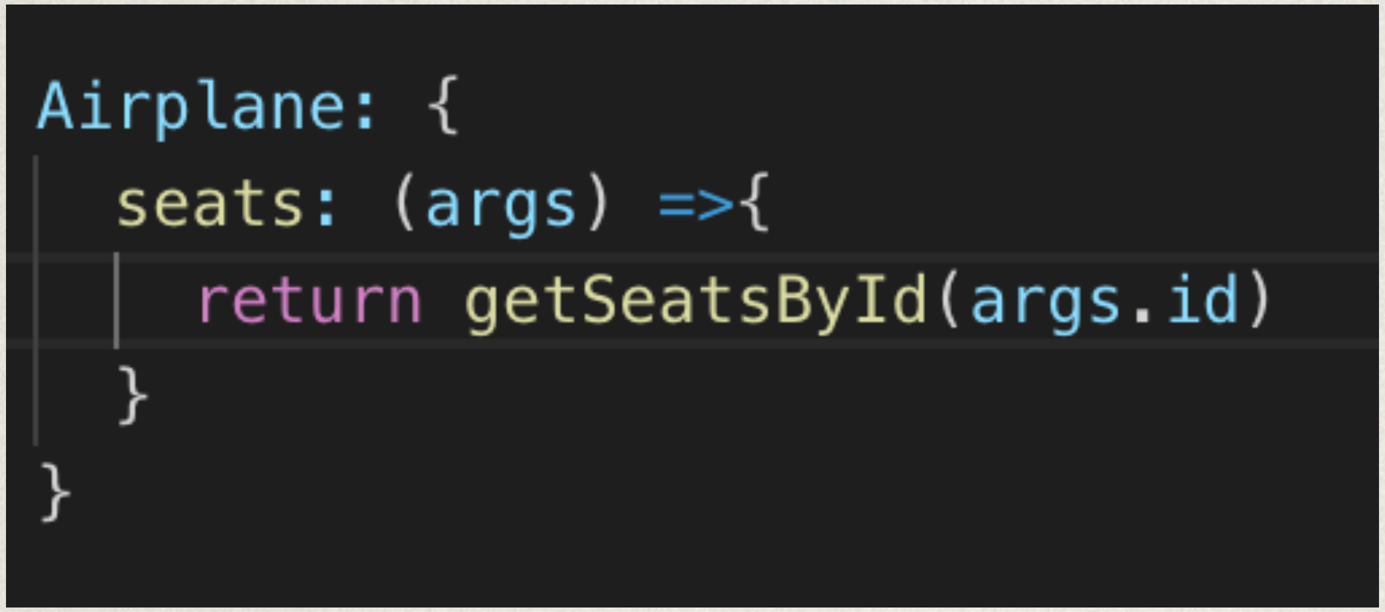
- Extending query
- Extending resolver

Step 3 - Extending query



```
type Seat {
  id: ID
  type: String
}

type Airplane {
  seats: [Seat!]!
}
```

Step 3 - Extending resolver

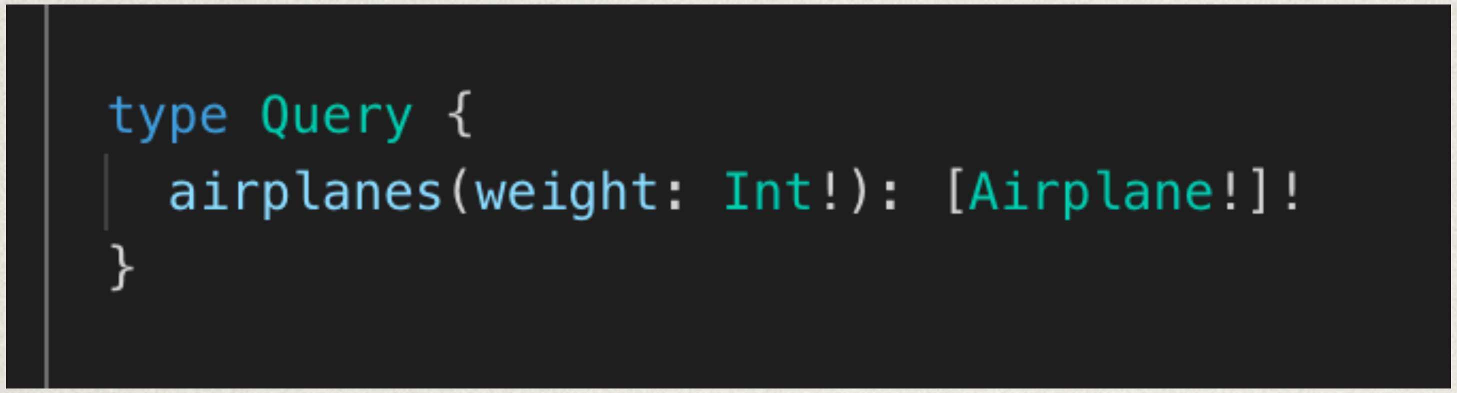


```
Airplane: {
  seats: (args) =>{
    return getSeatsById(args.id)
  }
}
```

Step 4

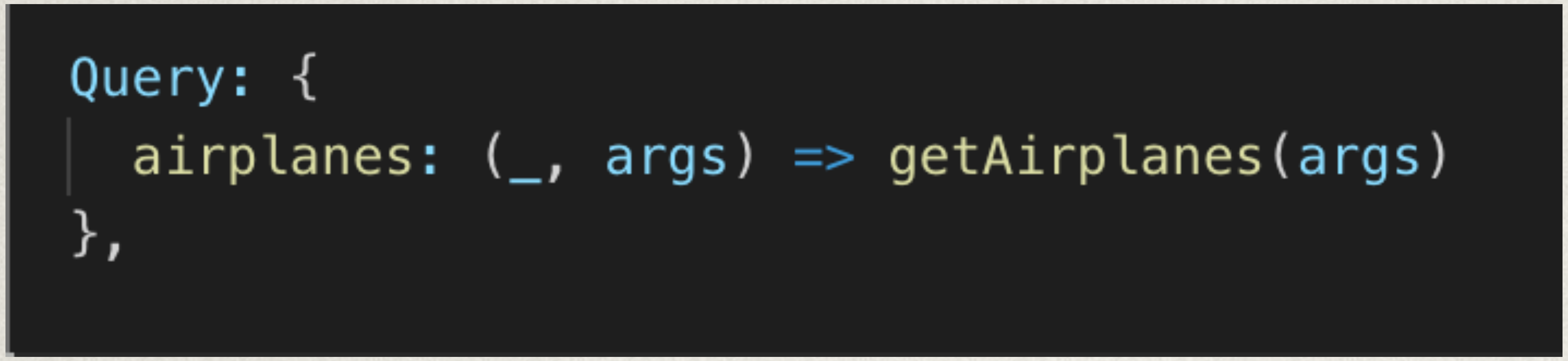
- Extending query
- Extending resolver

Step 4 - Extending query



```
type Query {
  airplanes(weight: Int!): [Airplane!]!
}
```

Step 4 - Extending resolver



```
Query: {
  airplanes: (_, args) => getAirplanes(args)
},
```

Step 5

Integrating **lowdb** and creating helper functions

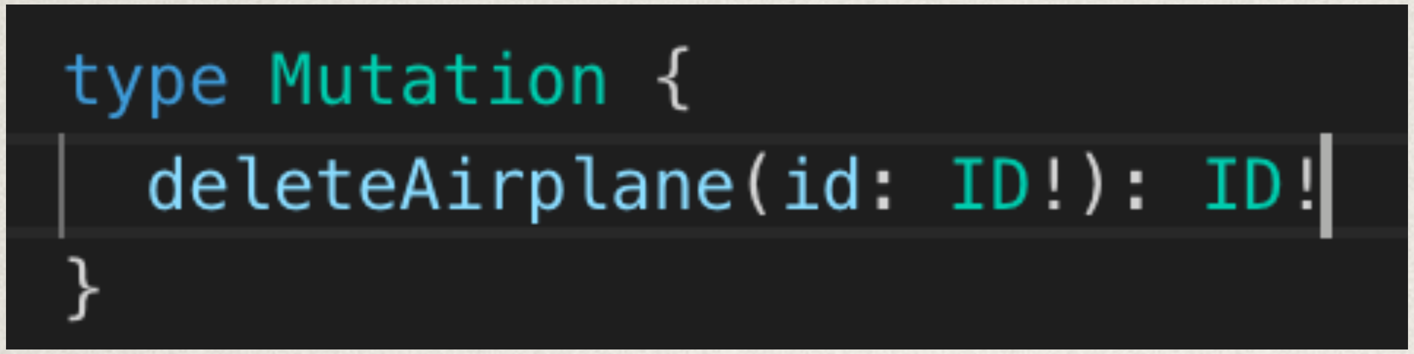
Step 6

Creating data model by using **lowdb** helper functions

Step 7

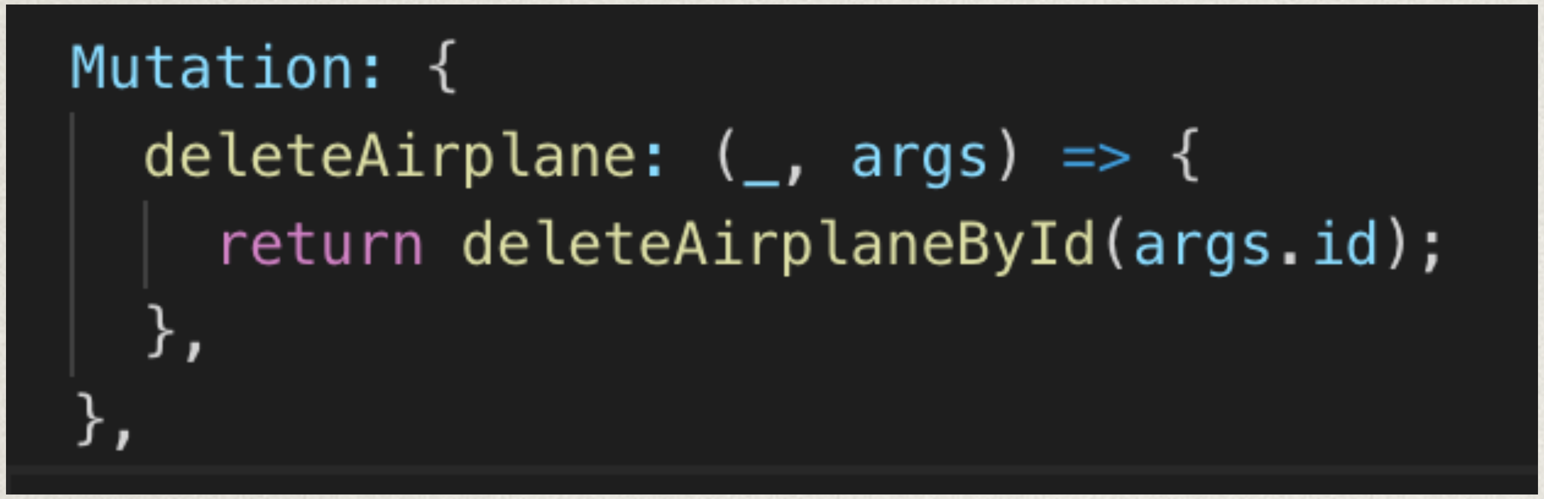
- Mutation type
- Mutation resolver

Step 7 - Mutation type



```
type Mutation {
  deleteAirplane(id: ID!): ID!
}
```

Step 7 - Mutation resolver



```
Mutation: {
  deleteAirplane: (_, args) => {
    return deleteAirplaneById(args.id);
  },
},
```

Questions? 🙋‍♀️

Gabriel Mičko

@gabriel_micko