WRITTEN BY
Mariusz Nosiński
Fullstack Developer @ Scalac.io

graphql-scala Tutorial - Introduction

Motivation

Scala is a very popular language nowadays and it's often chosen to deliver efficient and scalable systems. It leverages the Java VM, known for its reliability and robustness. Support for Functional Programming, rich ecosystem and stable foundation allow building fast applications, quickly.

In the next chapters you'll learn how to build your own GraphQL server using Scala and the following technologies:

Scala Scala language

Akka HTTP Web server to handle HTTP requests.

Sangria A library for GraphQL execution

Slick A Database query and access library.

H2 Database In-memory database.

Graphiql A simple GraphQL console to play with.

Giter8 A project templating tool for Scala.

I assume you're familiar with GraphQL concepts, but if not, you can visit GraphQL site to learn more about that.

What is a GraphQL Server?

A GraphQL server should be able to:

Receive requests following the GraphQL format, for example:

Connect to one or more data sources, like databases or other APIs and format obtained information.

Response with the requested data, such as this one:

```
{ "data": { "allLinks": { "url": "http://graphql.org/" } } }
```

Validate incoming requests accordingly to the schema definition and supported formats. For example, if a query is made with an unknown field, the response should be something like:

```
{
  "errors": [{
    "message": "Cannot query field \"unknown\" on type \"Link\"."
  }]
}
```

As you can see our server will be really simple, but real GraphQL implementation can do much more than this. (We will explore it more later on.)

Schema-Driven Development

Schema-first GraphQL development forces frontend and backend developers to agree on a strict contract up front, enabling them to work quickly and efficiently while staying on spec. It improves both your API's performance and the performance of your team in general.

Sensibly then, the experience of building a GraphQL server starts with working on its schema. You'll see in this chapter that the main steps you follow will be something like this:

1. Define your types and the appropriate queries and mutations for them.
2. Implement functions (called **resolvers**) to perform agreed upon queries in terms of defined types.
3. As new requirements arrive, go back to step 1 to update the schema and go through the other steps.

knowledge of the API from the start, using a simple mocking service (or even a full backend such as Graphcool) which can later be easily replaced with the final server.

Goal of the tutorial

Most of the HowToGraphQL tutorials are based on the same schema. In our tutorial we will try to run scala server which supports that schema. In this case you can take any frontend example and connect it to our server. The schema more or less looks like this:

```
type Query {
  allLinks(filter: LinkFilter, orderBy: LinkOrderBy, skip: Int, first: Int): [Li
  _allLinksMeta: _QueryMeta!
}

type Mutation {
  signinUser(email: AUTH_PROVIDER_EMAIL): SigninPayload!
  createUser(name: String!, authProvider: AuthProviderSignupData!): User
  createLink(description: String!, url: String!, postedById: ID): Link
  createVote(linkId: ID, userId: ID): Vote
}

type Subscription {
  Link(filter: LinkSubscriptionFilter): LinkSubscriptionPayload
  Vote(filter: VoteSubscriptionFilter): VoteSubscriptionPayload
}

interface Node {
  id: ID!
}

type User implements Node {
  id: ID! @isUnique
  createdAt: DateTime!
  name: String!
  links: [Link!]! @relation(name: "UsersLinks")
  votes: [Vote!]! @relation(name: "UsersVotes")
  email: String @isUnique
  password: String
}

type Link implements Node {
  id: ID! @isUnique
  createdAt: DateTime!
  url: String!
  description: String!
  postedBy: User! @relation(name: "UsersLinks")
  votes: [Vote!]! @relation(name: "VotesOnLink")
}
```

```
  link: Link! @relation(name: "VotesOnLink")
}

input AuthProviderSignupData {
  email: AUTH_PROVIDER_EMAIL
}

input AUTH_PROVIDER_EMAIL {
  email: String!
  password: String!
}

input LinkSubscriptionFilter {
  mutation_in: [_ModelMutationType!]
}

input VoteSubscriptionFilter {
  mutation_in: [_ModelMutationType!]
}

input LinkFilter {
  OR: [LinkFilter!]
  description_contains: String
  url_contains: String
}

type SigninPayload {
  token: String
  user: User
}

type LinkSubscriptionPayload {
  mutation: _ModelMutationType!
  node: Link
  updatedFields: [String!]
}

type VoteSubscriptionPayload {
  mutation: _ModelMutationType!
  node: Vote
  updatedFields: [String!]
}

enum LinkOrderBy {
  createdAt_ASC
  createdAt_DESC
}

enum _ModelMutationType {
  CREATED
  UPDATED
  DELETED
}
```

```
scalar DateTime
```

When we know what to do, we move on to the next chapter and begin the tutorial.

UNLOCK THE NEXT CHAPTER

What is a GraphQL?

- ○ Protocol
- ○ Library
- ○ Language

Specification

Well done, you unlocked the next chapter!

NEXT CHAPTER

Getting Started

In this chapter will be described how to setup the HTTP server, install all dependencies and setup the database.

Go to next chapter

HOW TO GRAPHQL