Server Side GraphQL

Using Elixir, Phoenix and Absinthe

Robert Boone

ChaiOne

June 27, 2018

Table of Contents

What is GraphQL?

GraphQL is a data query language developed by Facebook in 2012 and release in 2015.

Query Language

- A language for the specification of procedures for the retrieval (and sometimes also modification) of information from a database.

Query Language

- A language for the specification of procedures for the retrieval (and sometimes also modification) of information from a database.
- The fact that GraphQL can be used as a web API is an implementation detail.

Query Language

- A language for the specification of procedures for the retrieval (and sometimes also modification) of information from a database.
- The fact that GraphQL can be used as a web API is an implementation detail.
- It's has more in common with SQL than it does with REST.

Types

- APIs in GraphQL are organized by types and fields.

Builtin Types

- boolean
- float
- id
- integer
- string

Types

- You can also create custom Scalar Types.

```
1   scalar :date do
2     parse(fn input ->
3       case Timex.parse!(input.value, "{YYYY}-{0M}-{D}")
            |> DateTime.from_naive("Etc/UTC") do
4         {:ok, date} -> {:ok, date}
5         _ -> :error
6       end
7     end)
8
9     serialize(fn date -> Date.to_iso8601(date) end)
10  end
```

Listing 1: Custom Scaler Type

Types

... Enum Types

```
1       enum :priority_level, description: "Todo priority levels
            " do
2         value :high, as: :true, description: "High Priority"
3         value :low, as: :false, description: "Low Priority"
4       end
```

Listing 2: Enum Type

```
Request to pry #PID<0.623.0> at FariWeb.Resolvers.Todos.create/3 (lib/fari_web/resolvers/todos.ex:5)

    3:
    4:   def create(_obj, args, %{context: %{current_user: user}}) do
    5:     require IEx; IEx.pry
    6:     user
    7:     |> Ecto.build_assoc(:todos)

Allow? [Yn] y

Interactive Elixir (1.6.6) - press Ctrl+C to exit (type h() ENTER for help)
pry(1)> args
%{priority: true}
pry(2)> true == :true
true
```

Types

- The type used most often is the Object Type

Types

- The type used most often is the Object Type

```
1   object :todo do
2     field(:id, :id, description: "Todo id")
3     field(:title, :string, description: "Todo Title")
4     field(:priority, :priority_level, description: "Priority
          ?")
5     field(:due_at, :date, description: "Due date")
6     field(:complete, :boolean, description: "Todo complete")
7   end
```

Listing 3: Object Type

Query

- The Query Root object is where all queries are found.

Query

- The Query Root object is where all queries are found.

```
1     query do
2       field :users, list_of(:user), description: "List users
            in my groups" do
3         resolve(&FariWeb.Resolvers.Users.list_users/3)
4       end
5     end
```

Listing 4: User Root Object Type

Resolvers

- The resolve function tells the object how to get it's data.

```
1     resolve(&FariWeb.Resolvers.Users.list_users/3)
2
3     def list_users(_obj, _args, %{context: %{current_user:
          user}}) do
4       users =
5         user
6         |> Fari.Repo.preload(:groups)
7         |> get\_groups()
8         |> Enum.map(fn group -> Fari.Repo.preload(group, :
              users) end)
9         |> Enum.map(fn g -> g.users end)
10        |> List.flatten()
11
12      {:ok, users}
13  end
```

Listing 5: Resolve function

Resolvers Arguments

- Resolve functions take 3 arguments.

Resolvers Arguments

- Resolve functions take 3 arguments.
- parent
  - This is the object above the current object.
  - At the root level this is normally nil.

Resolvers Arguments

- Resolve functions take 3 arguments.
- parent
  - This is the object above the current object.
  - At the root level this is normally nil.
- arguments

Resolvers Arguments

- Resolve functions take 3 arguments.
- parent
  - This is the object above the current object.
  - At the root level this is normally nil.
- arguments
  - Field arguments

Resolvers Arguments

- Resolve functions take 3 arguments.
- parent
  - This is the object above the current object.
  - At the root level this is normally nil.
- arguments
  - Field arguments
- Absinthe.Resolution

Resolvers Arguments

- Resolve functions take 3 arguments.
- parent
  - This is the object above the current object.
  - At the root level this is normally nil.
- arguments
  - Field arguments
- Absinthe.Resolution
  - This contains the **context** and other execution data

Resolvers Return Value

- The resolve function returns successful result with a tuple in the form of `{:ok, %{}}`
- The resolve function returns an error with a tuple in the form of `{:error, reason}`