kargo

Backend Elixir: Absinthe Graphql API

Kargo Engineering Academy

kargo

Excellerate

Excellent & Accelerate - A Kargo Tech Hiring Event

Agenda

Day 7

09.30	Opening
09.45	Elixir: Graphql
10.15	Workshop II
11.45	Quiz
12.00	Closing

What is GraphQL ?

What is Absinthe ?

Absinthe: Core Concepts and Installation

Core Concepts

Absinthe Installation

Setting Up GraphQL

Schema Definition

- Query
- Mutation
- Object Type
- Input Object Type

Middleware

Resolver Function

Absinthe Installation

```
#Put dependency below to mix.exs
defp deps do
   [
      ...
      {:absinthe, "~> 1.6"},
      {:absinthe_plug, "~> 1.5"}
      ...
   ]
end
#Run get dependency and compilation
> mix deps.get
> mix deps.compile
```

Absinthe: Get Started

Step 1: GraphQL Module

```elixir
defmodule CommunityWeb.Schema do
  # required to make a module graphQL Schema
  use Absinthe.Schema

  # this is the resolver that will be used
  alias CommunityWeb.NewsResolver

  query do
    # this is the query entry point to our app
  end

  mutation do
    # this is the mutation entry point to our app
  end
end
```

Step 2: GraphQL Endpoint

```elixir
defmodule CommunityWeb.Router do
  use CommunityWeb, :router

  pipeline :api do
    plug :accepts, ["json"]
  end

  scope "/" do
    pipe_through :api

    forward "/graphiql", Absinthe.Plug.GraphiQL,
      schema: CommunityWeb.Schema, # GraphQL Schema Module
      interface: :simple,
      context: %{pubsub: CommunityWeb.Endpoint}
  end
end
```

Step 3: Resolver Module

```
defmodule CommunityWeb.NewsResolver do
  alias Community.News

  def all_links(_root, _args, _info) do
    {:ok, News.list_links()}
  end
end
```

Absinthe: Schema Definition

Query

```
query do
  @desc "Get all links"
  field :all_links, non_null(list_of(non_null(:link))) do
    arg(:id, :string)
    arg(:user_id, non_null(:string))

    middleware(Middleware.Authorize, Role.all())
    resolve(&NewsResolver.all_links/3)
  end
end
```

Mutation

```
mutation do
  field :create_link, :link do
    arg :url, non_null(:string)
    arg :user, non_null(:user_type)
    arg :description, non_null(:string)

    resolve &NewsResolver.create_link/3
  end
end
```

Types (object and input_object)

```
object :user do
 field :id, :integer
 field :name, :string
end

input_object :user do
 field :id, non_null(:integer)
 field :name, :string
end
```

Schema Module

```
defmodule Loads.JobQueries do

     use Absinthe.Schema.Notation

     use Absinthe.Relay.Schema.Notation, :classic


     object :job_queries do

       @desc "Get all links"
       field :all_links, non_null(list_of(non_null(:link))) do
         arg(:id, :string)
          arg(:user_id, non_null(:string))

          resolve(&NewsResolver.all_links/3)
       end

     end

end
```

Types Module

```
defmodule User.UserTypes do
 @moduledoc false
 use Absinthe.Schema.Notation
 use Absinthe.Ecto, repo: LoadService.Repo
 use Absinthe.Relay.Schema.Notation, :classic

 object :user do
  field :id, :integer
  field :name, :string
 end
 input_object :user do
  field :id, non_null(:integer)
  field :name, :string
 end

end
```

Import Fields

```elixir
defmodule CommunityWeb.Schema do
  # required to make a module graphQL Schema
  use Absinthe.Schema

  # this is the resolver that will be used
  alias CommunityWeb.NewsResolver
  # Import object
  import_types(Loads.JobQueries)

  import_types(User.UserTypes)


  query do
    # Import field inside object
    import_fields(:job_queries)
  end

  mutation do
    # this is the mutation entry point to our app
  end

end
```

Elixir Syntax: Middleware

Middleware

Middleware is custom module that is called to process resolution. It’s like plug for absinthe

```
defmodule Middleware.Authorize do
  @behaviour Absinthe.Middleware

  def call(resolution, _config) do
    case resolution.context do
      %{current_user: _} ->
        resolution
      _ ->
        resolution
        |> Absinthe.Resolution.put_result({:error, "unauthenticated"})
    end
  end
end
```

Implementation

```
defmodule Loads.JobQueries do

     use Absinthe.Schema.Notation

     use Absinthe.Relay.Schema.Notation, :classic


     object :job_queries do

       @desc "Get all links"
       field :all_links, non_null(list_of(non_null(:link))) do
         arg(:id, :string)
          arg(:user_id, non_null(:string))

          middleware(Middleware.Authorize, Role.all())
          resolve(&NewsResolver.all_links/3)
       end

     end

end
```

Elixir Syntax: Resolver Function

Resolver Functions

Resolver is the same as controller functions. Resolver function can have 2 or 3 arity. Resolver function can only accept {:ok, result} as the valid output of the function. Example as below:

```
def all_links(_parent, args, resolution) do
   {:ok, News.list_links()}
end

def all_links(args, resolution) do
   {:ok, News.list_links()}
end
```

Error Message

Resolver function can also return error message using {:error, _} tuple. Example as below:

```
#Simple Error Message
{:error, "Something bad happened"}
#Multiple Error message
{:error, ["Something bad", "Even worse"]}
#Custom error message, need to have item with name ‘message’
{:error, message: "Unknown user", code: 21}
{:error, %{message: "A database error occurred", details:
format_db_error(some_value)}}
#Mixed
{:error, ["Simple message", [message: "A keyword list error", code: 1],
%{message: "A map error"}]}
```

Workshop Continue
(1 Hour 30 Minutes)

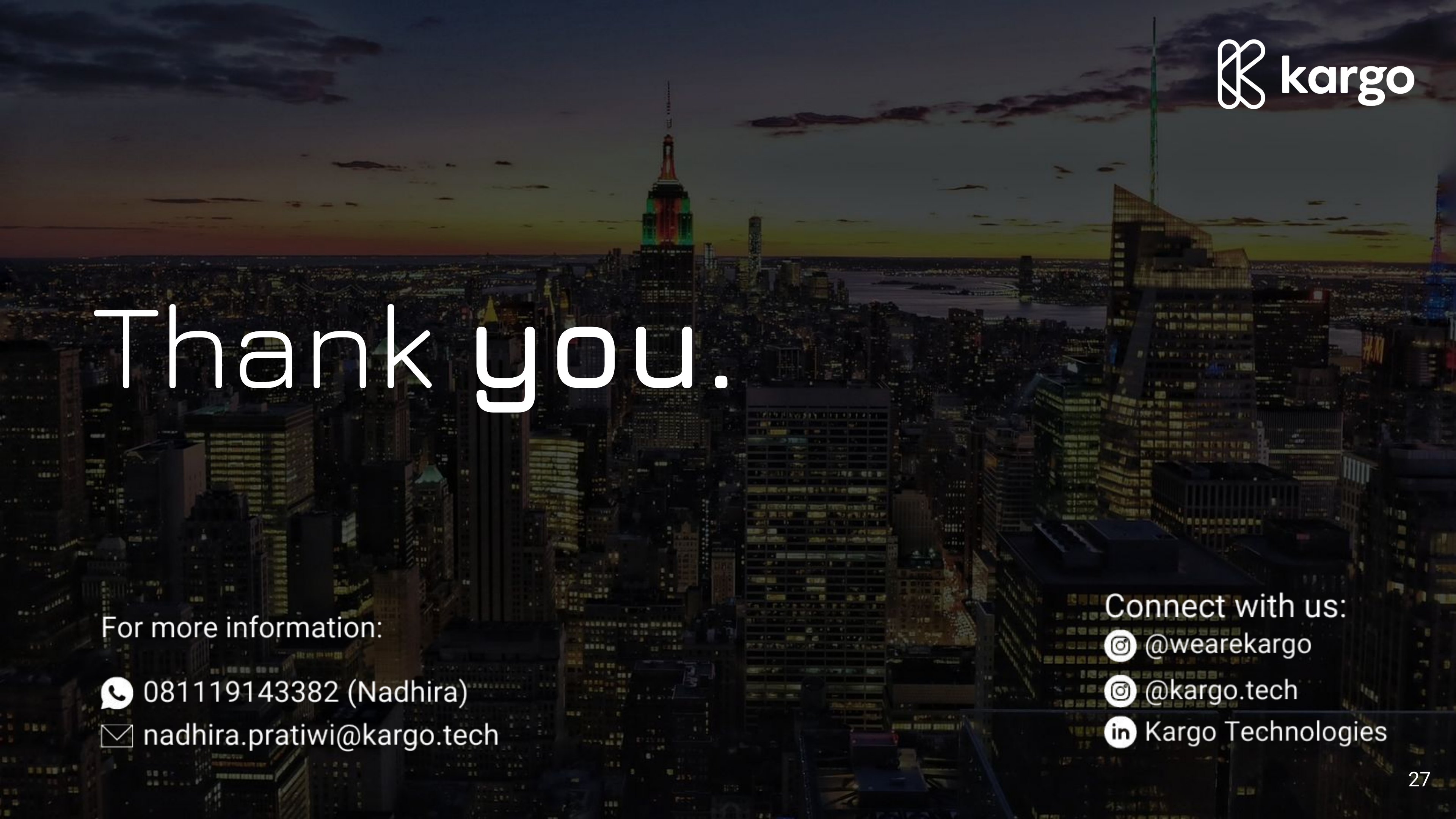
Quiz

https://bit.ly/3NO05mN

Sources

[1] https://www.howtographql.com/graphql-elixir/0-introduction/

[2] https://hexdocs.pm/absinthe/overview.html



Thank you.

For more information:

081119143382 (Nadhira)

nadhira.pratiwi@kargo.tech

Connect with us:

@wearekargo

@kargo.tech

Kargo Technologies