Absinthe
GraphQL Basics

Ruben Amortegui
@ramortegui
http://rubenamortegui.com
https://github.com/ramortegui

GraphQL

- A query language for your API
  - Ask for what you need, get exactly that.
  - Get many resources in a single request
  - Describe what is possible with a type system
  - Move faster with powerful developer tools
  - Bring your own data an code
  - Evolve your API without versions

http://graphql.org/

GraphQL as Specification

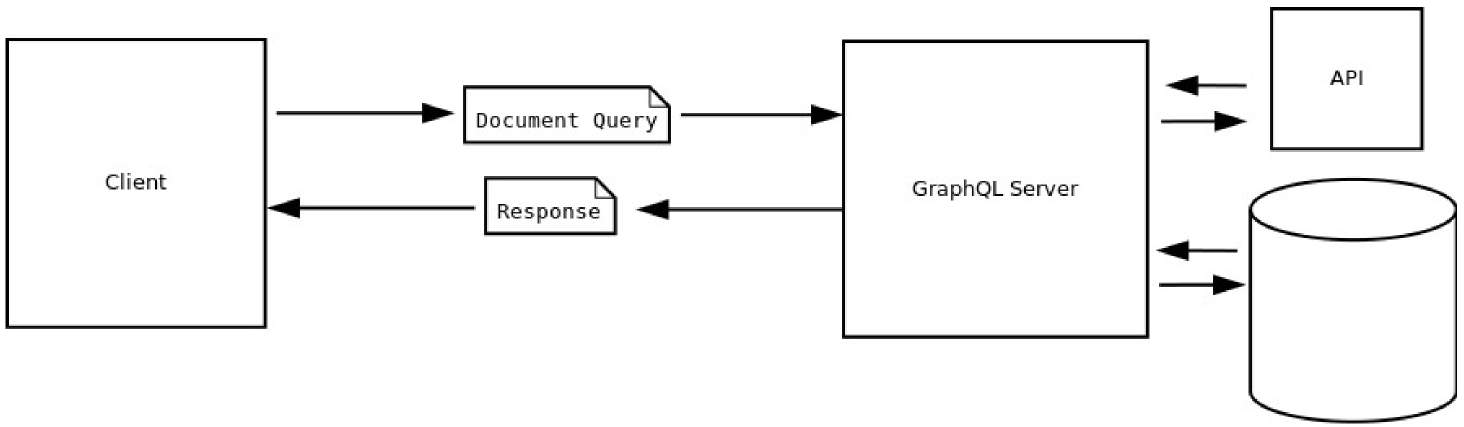
- Language
- Type System
- Introspection
- Validation
- Execution
- Response

http://facebook.github.io/graphql/October2016/

GraphQL

- Why?
  - REST (REpresentational State Transfer)
    - What is on the response?
    - What if we need more info
    - What if we don’t need all the info
    - Validations
  - GraphQL
    - Give the developer a query language to interact with the server
    - Provides validation of data by default
    - Handle relationships
    - Give proper error messages
    - What you ask is what you expect
    - Single point of entrance

Basic architecture



GraphQL

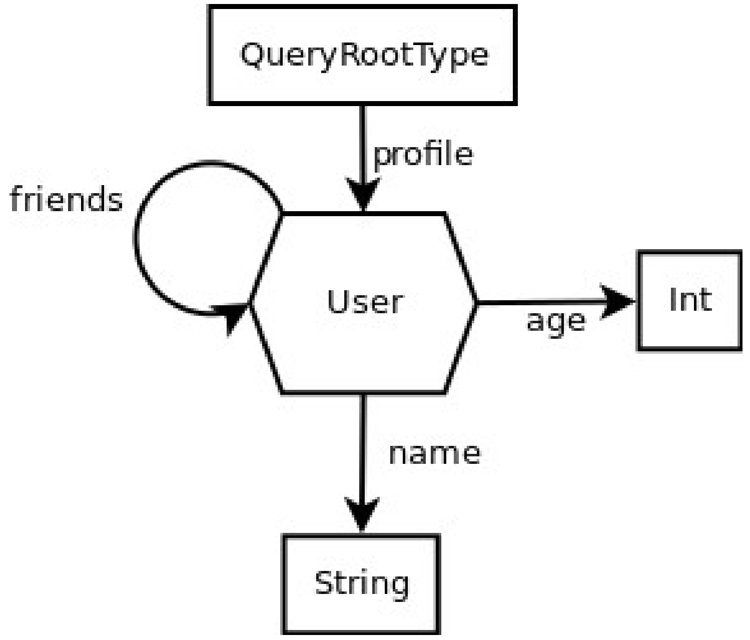
- It’s not magic
- You need to build the representation of your data.

Schema sample

Built-in types

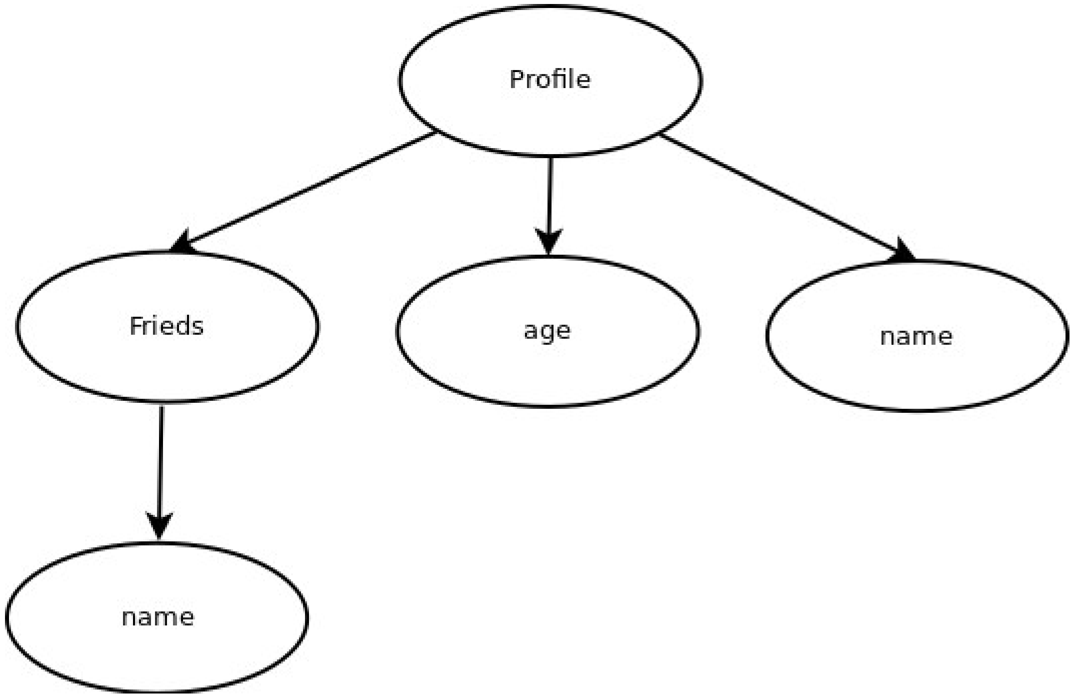
Relations

Custom Types



Query representation sample

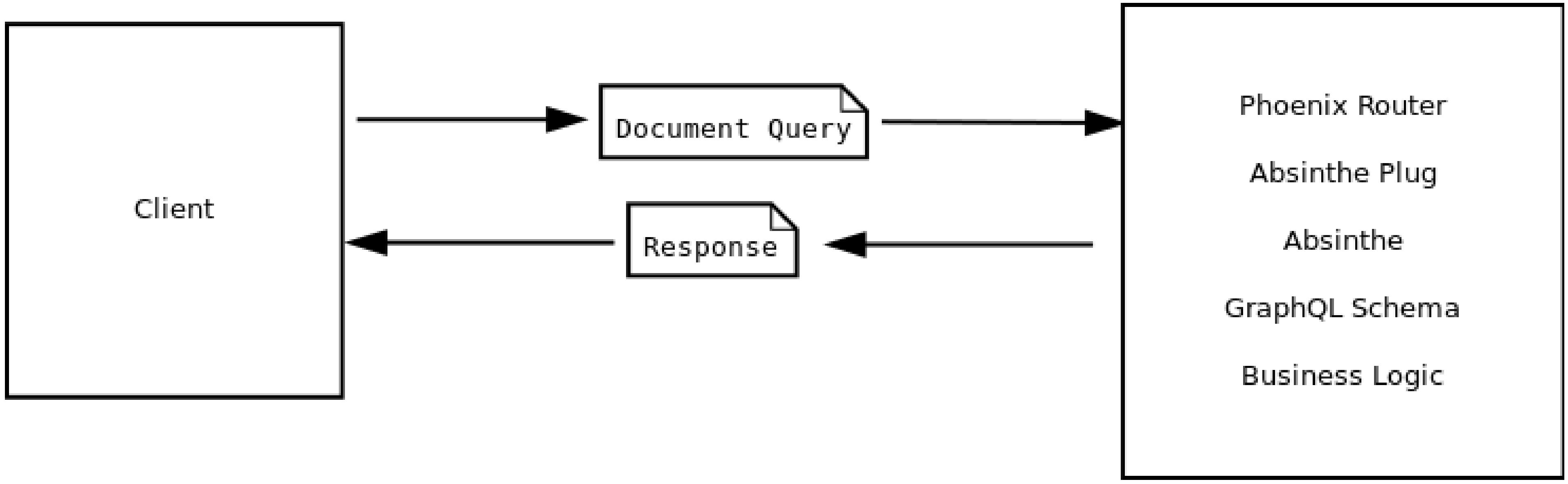
```
{
  profile{
    name
    age
    friends{
      name
    }
  }
}
```



What is Absinthe

- GraphQL toolkit for elixir
  - absinthe
  - absinthe_plug
  - absinthe_phoenix

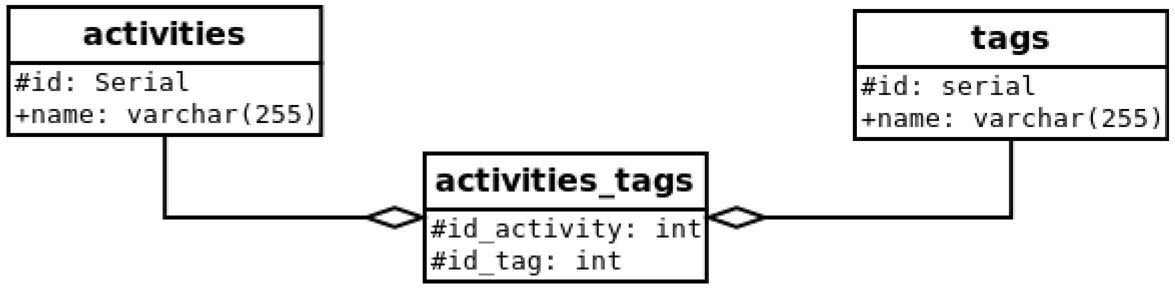
Absinthe


Client
Document Query
Response
Phoenix Router
Absinthe Plug
Absinthe
GraphQL Schema
Business Logic

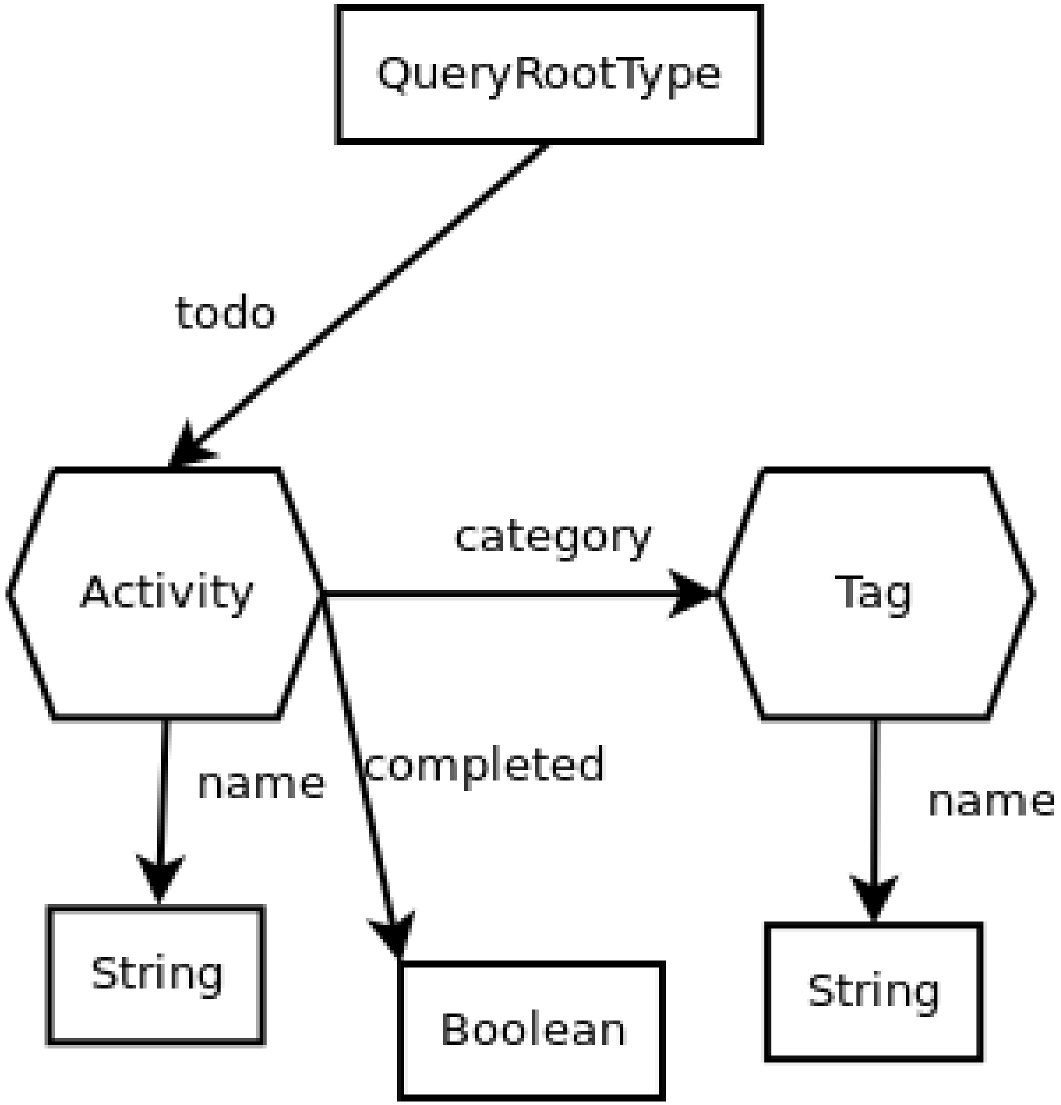
Basic Sample

- Create a API to manage a Todo application.
  - Design a Schema
  - Query
  - Mutation
  - Test

“Todo” DB


activities
#id: Serial
+name: varchar(255)
activities_tags
#id_activity: int
#id_tag: int
tags
#id: serial
+name: varchar(255)

“Todo” GraphQL Schema

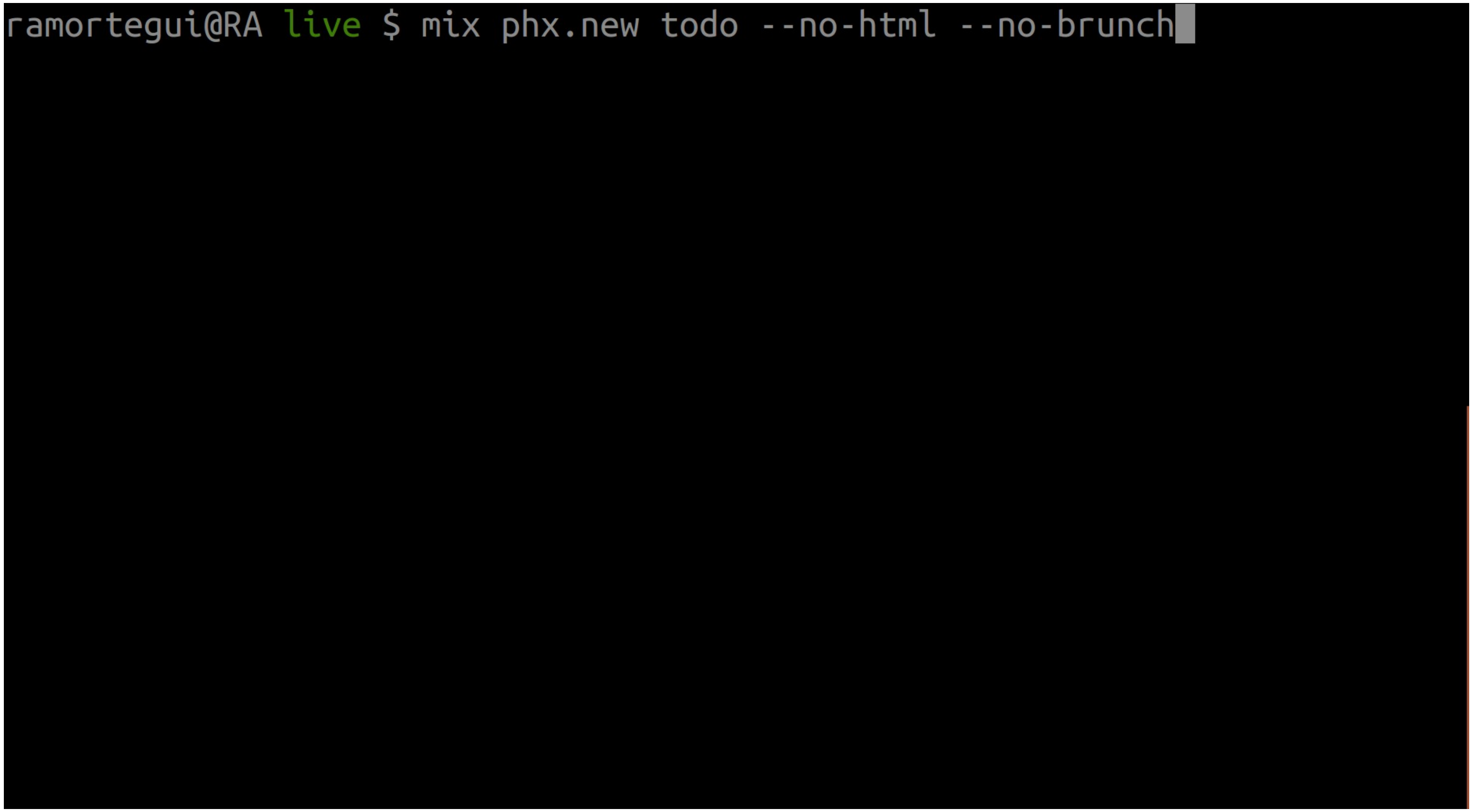


“Todo” GraphQL Schema

- Queries

```
{
  todos {
    name
    categories {
      name
    }
  }
}
```

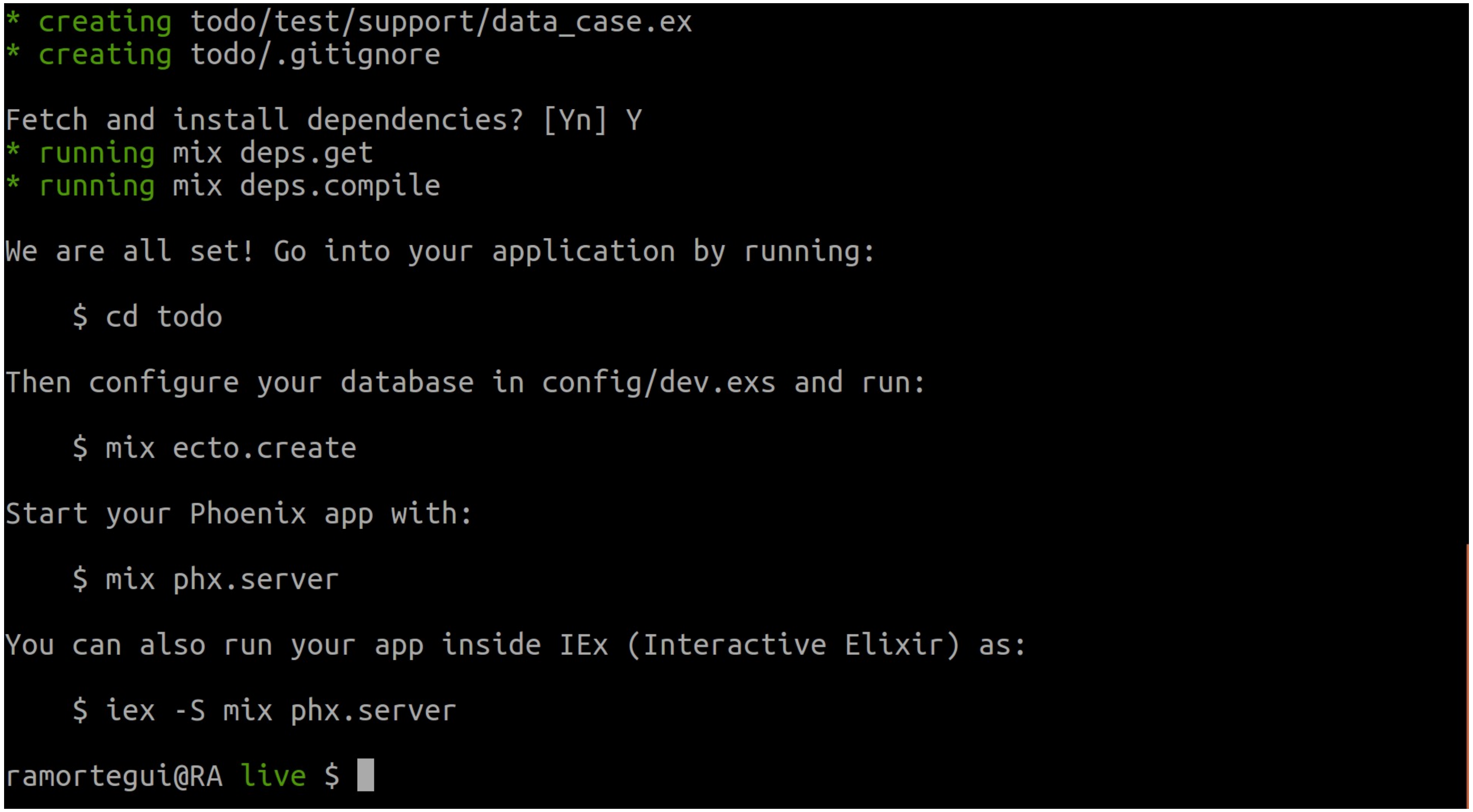
TODO GraphQL



```
ramortegui@RA live $ mix phx.new todo --no-html --no-brunch
```

TODO GraphQL

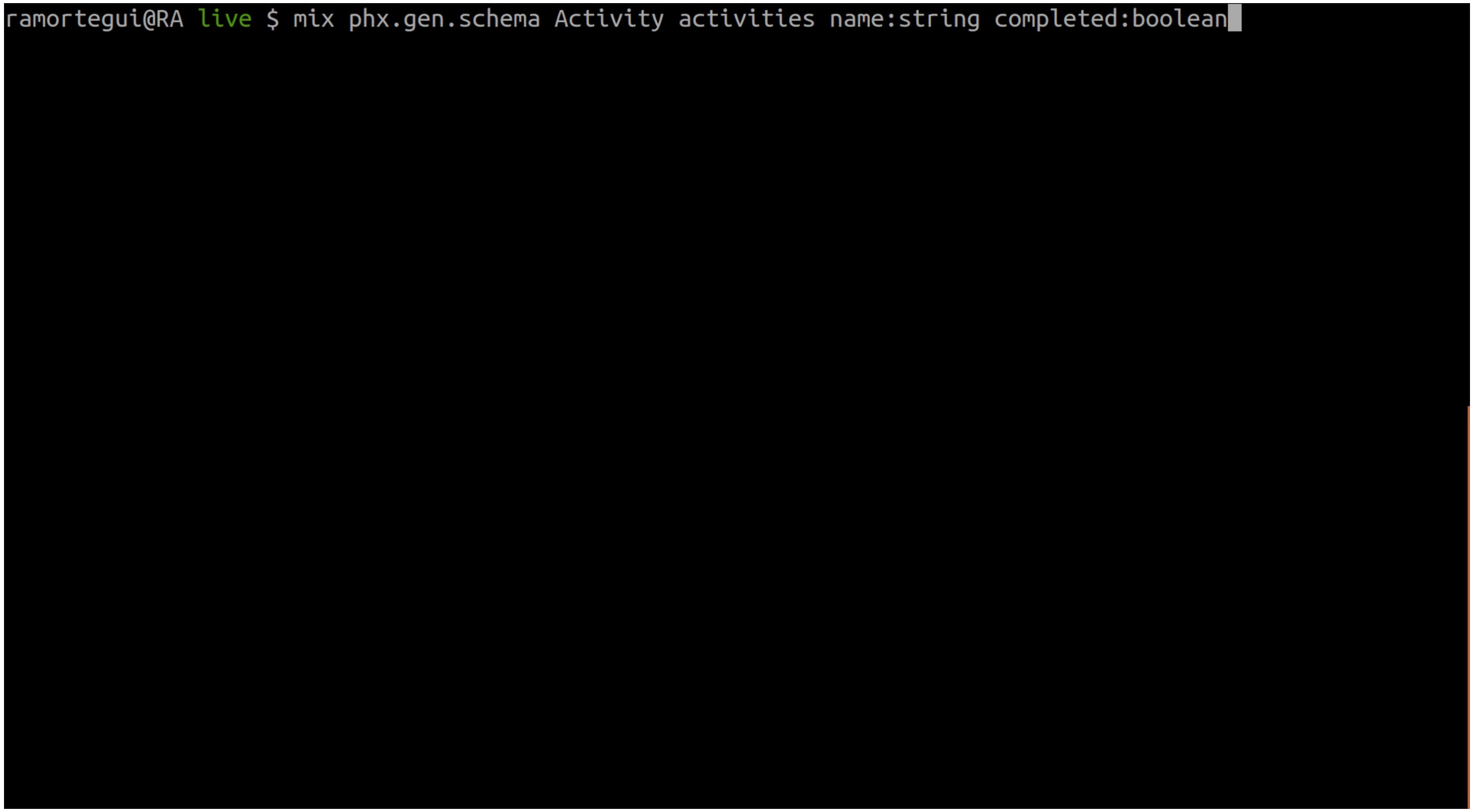
Create a phx app



```
* creating todo/test/support/data_case.ex
* creating todo/.gitignore

Fetch and install dependencies? [Yn] Y
* running mix deps.get
* running mix deps.compile

We are all set! Go into your application by running:

    $ cd todo

Then configure your database in config/dev.exs and run:

    $ mix ecto.create

Start your Phoenix app with:

    $ mix phx.server

You can also run your app inside IEx (Interactive Elixir) as:

    $ iex -S mix phx.server

ramortegui@RA live $ 
```

TODO GraphQL

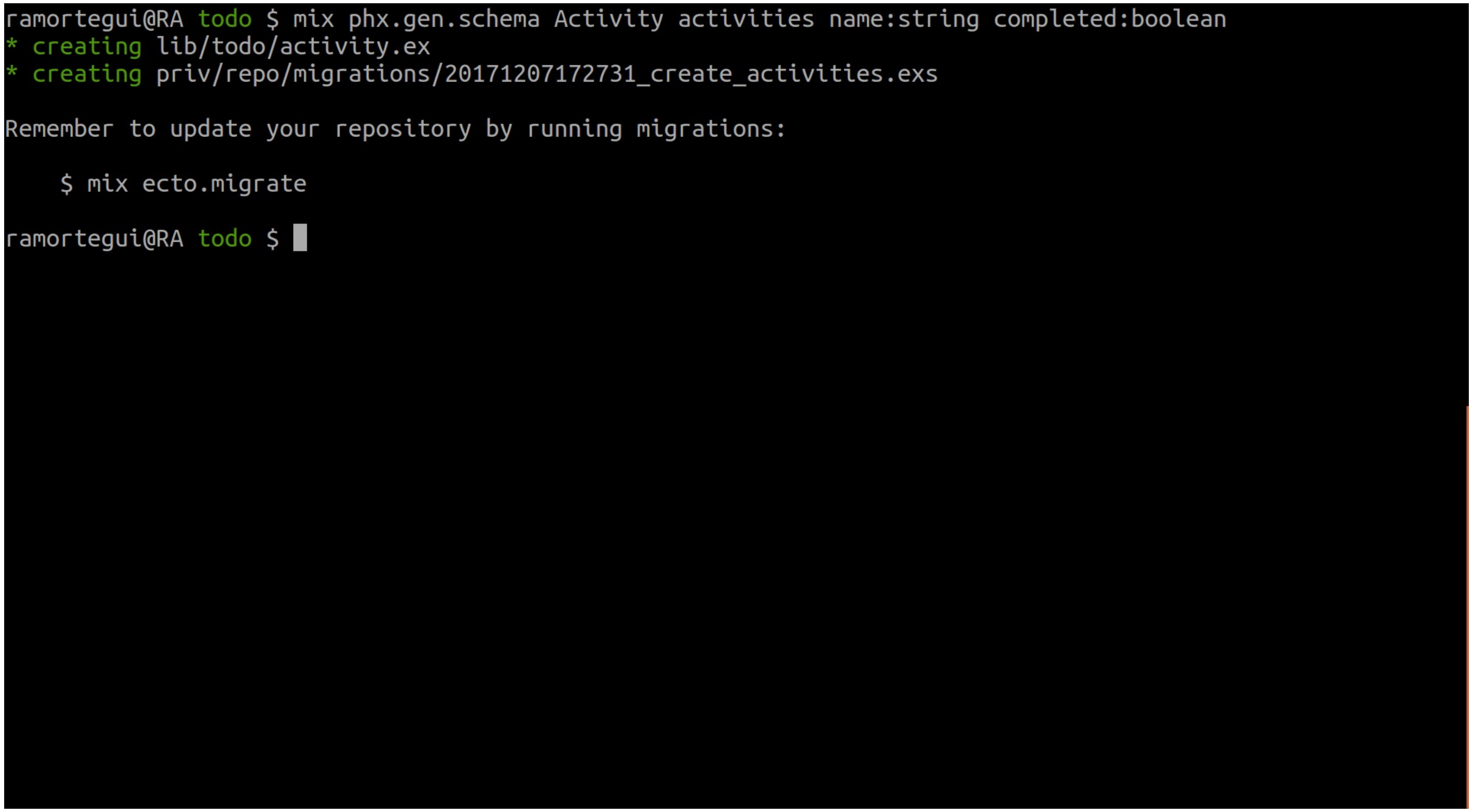
Add tables and run migration



```
ramortegui@RA live $ mix phx.gen.schema Activity activities name:string completed:boolean
```

TODO GraphQL

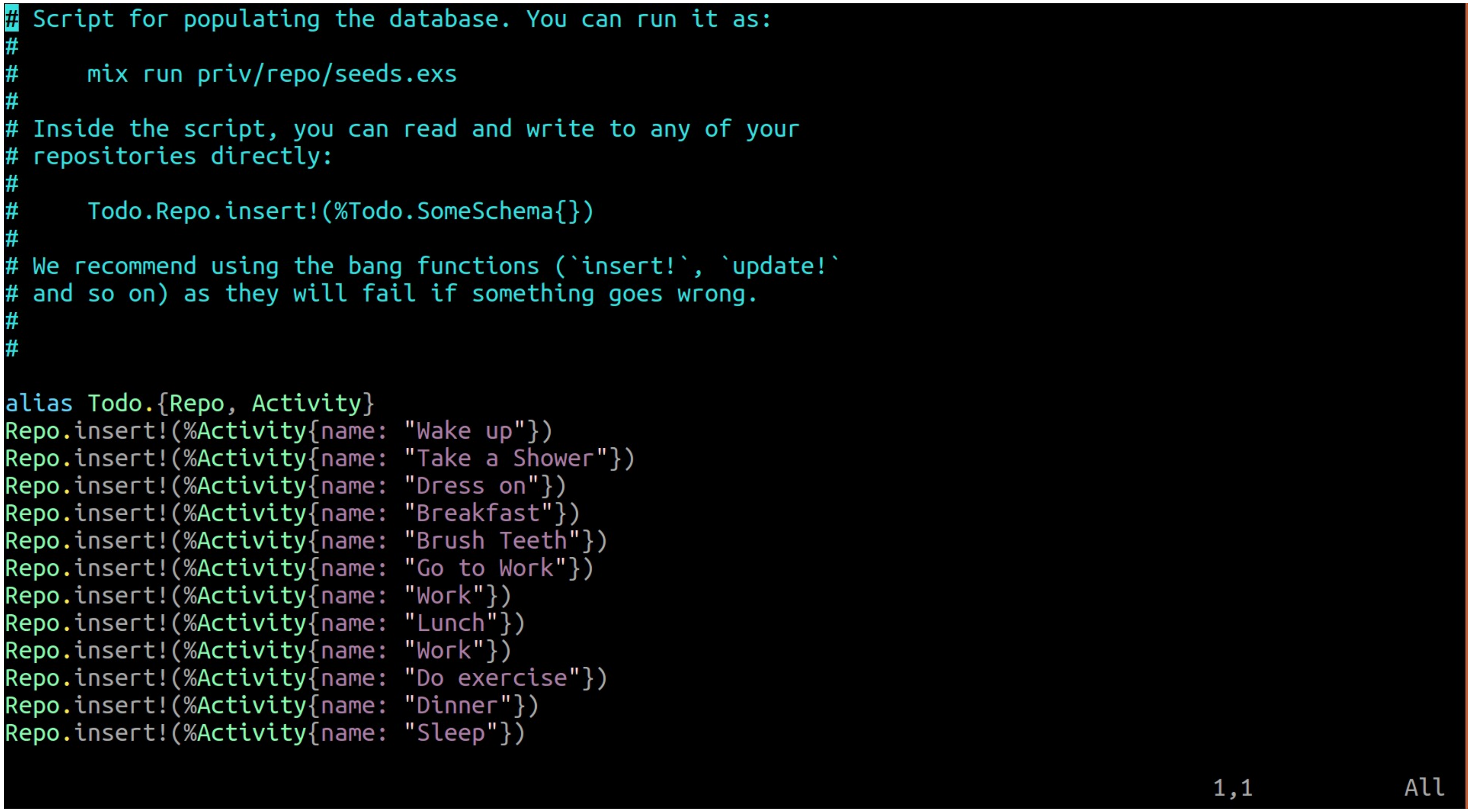
Add tables and run migration



```
ramortegui@RA todo $ mix phx.gen.schema Activity activities name:string completed:boolean
* creating lib/todo/activity.ex
* creating priv/repo/migrations/20171207172731_create_activities.exs

Remember to update your repository by running migrations:

    $ mix ecto.migrate

ramortegui@RA todo $ 
```

TODO GraphQL

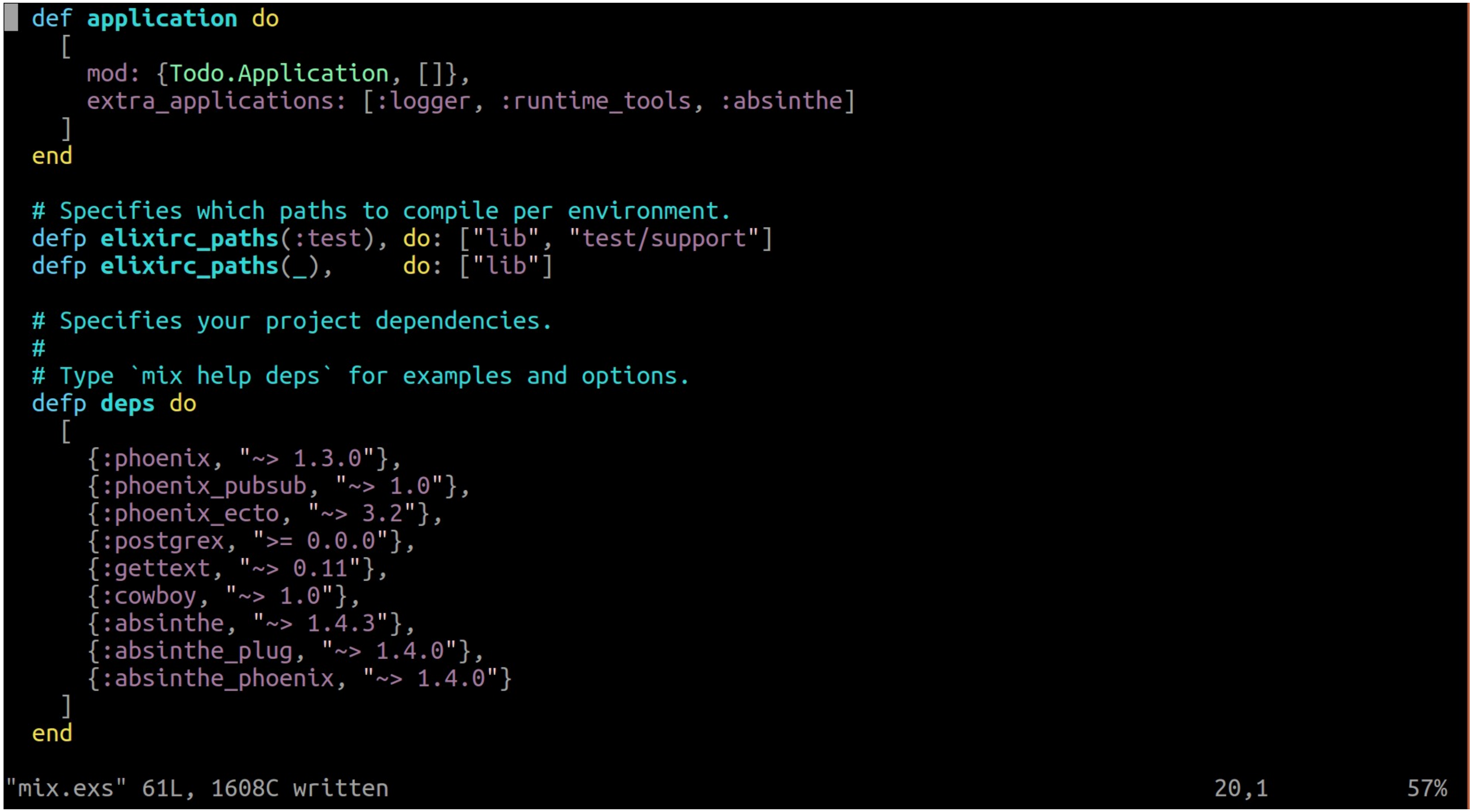
Create seed data



```
# Script for populating the database. You can run it as:
#
#     mix run priv/repo/seeds.exs
#
# Inside the script, you can read and write to any of your
# repositories directly:
#
#     Todo.Repo.insert!(%Todo.SomeSchema{})
#
# We recommend using the bang functions (`insert!`, `update!`
# and so on) as they will fail if something goes wrong.
#
#

alias Todo.{Repo, Activity}
Repo.insert!(%Activity{name: "Wake up"})
Repo.insert!(%Activity{name: "Take a Shower"})
Repo.insert!(%Activity{name: "Dress on"})
Repo.insert!(%Activity{name: "Breakfast"})
Repo.insert!(%Activity{name: "Brush Teeth"})
Repo.insert!(%Activity{name: "Go to Work"})
Repo.insert!(%Activity{name: "Work"})
Repo.insert!(%Activity{name: "Lunch"})
Repo.insert!(%Activity{name: "Work"})
Repo.insert!(%Activity{name: "Do exercise"})
Repo.insert!(%Activity{name: "Dinner"})
Repo.insert!(%Activity{name: "Sleep"})

                                                                                                    1,1           All
```

TODO GraphQL

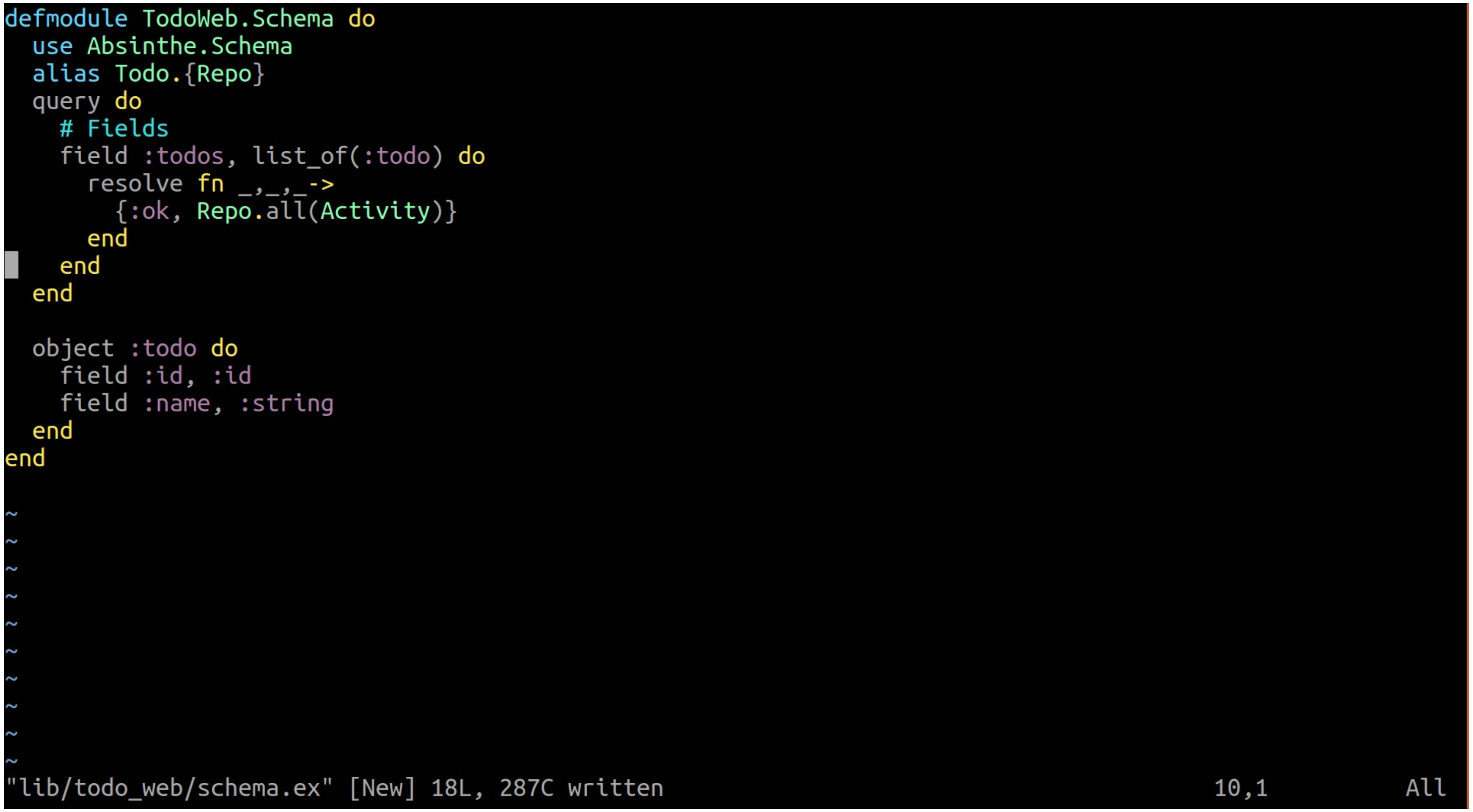
Add absinthe library



```
  def application do
    [
      mod: {Todo.Application, []},
      extra_applications: [:logger, :runtime_tools, :absinthe]
    ]
  end

  # Specifies which paths to compile per environment.
  defp elixirc_paths(:test), do: ["lib", "test/support"]
  defp elixirc_paths(_),     do: ["lib"]

  # Specifies your project dependencies.
  #
  # Type `mix help deps` for examples and options.
  defp deps do
    [
      {:phoenix, "~> 1.3.0"},
      {:phoenix_pubsub, "~> 1.0"},
      {:phoenix_ecto, "~> 3.2"},
      {:postgrex, ">= 0.0.0"},
      {:gettext, "~> 0.11"},
      {:cowboy, "~> 1.0"},
      {:absinthe, "~> 1.4.3"},
      {:absinthe_plug, "~> 1.4.0"},
      {:absinthe_phoenix, "~> 1.4.0"}
    ]
  end

"mix.exs" 61L, 1608C written                                   20,1          57%
```

TODO GraphQL

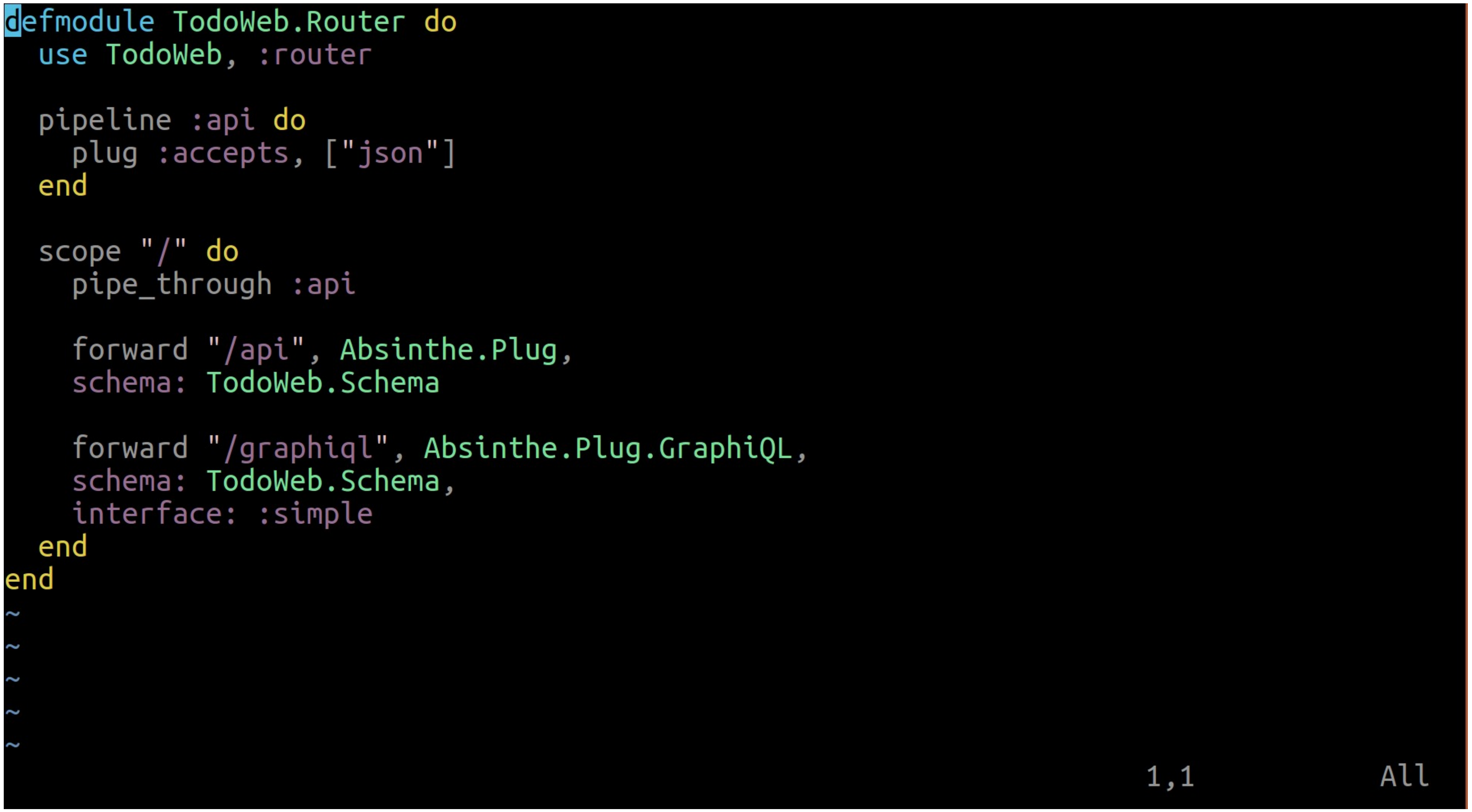
Add schema



```
defmodule TodoWeb.Schema do
  use Absinthe.Schema
  alias Todo.{Repo}
  query do
    # Fields
    field :todos, list_of(:todo) do
      resolve fn _,_,_->
        {:ok, Repo.all(Activity)}
      end
    end
  end

  object :todo do
    field :id, :id
    field :name, :string
  end
end

~
~
~
~
~
~
~
~
~
~
"lib/todo_web/schema.ex" [New] 18L, 287C written                                        10,1          All
```

TODO GraphQL

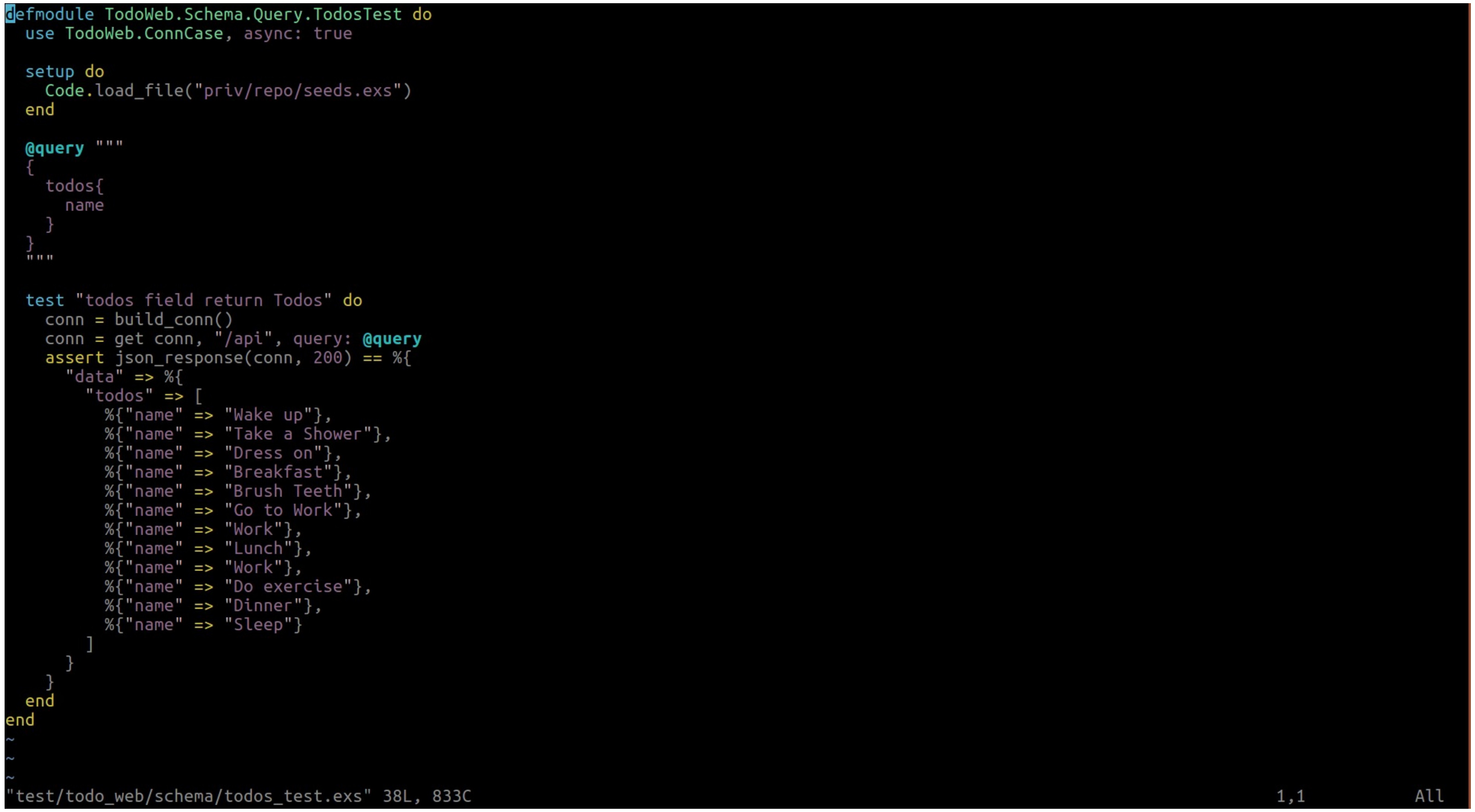
Add routes



```
defmodule TodoWeb.Router do
  use TodoWeb, :router

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

  scope "/" do
    pipe_through :api

    forward "/api", Absinthe.Plug,
    schema: TodoWeb.Schema

    forward "/graphiql", Absinthe.Plug.GraphiQL,
    schema: TodoWeb.Schema,
    interface: :simple
  end
end
~
~
~
~
~
                                                                         1,1           All
```

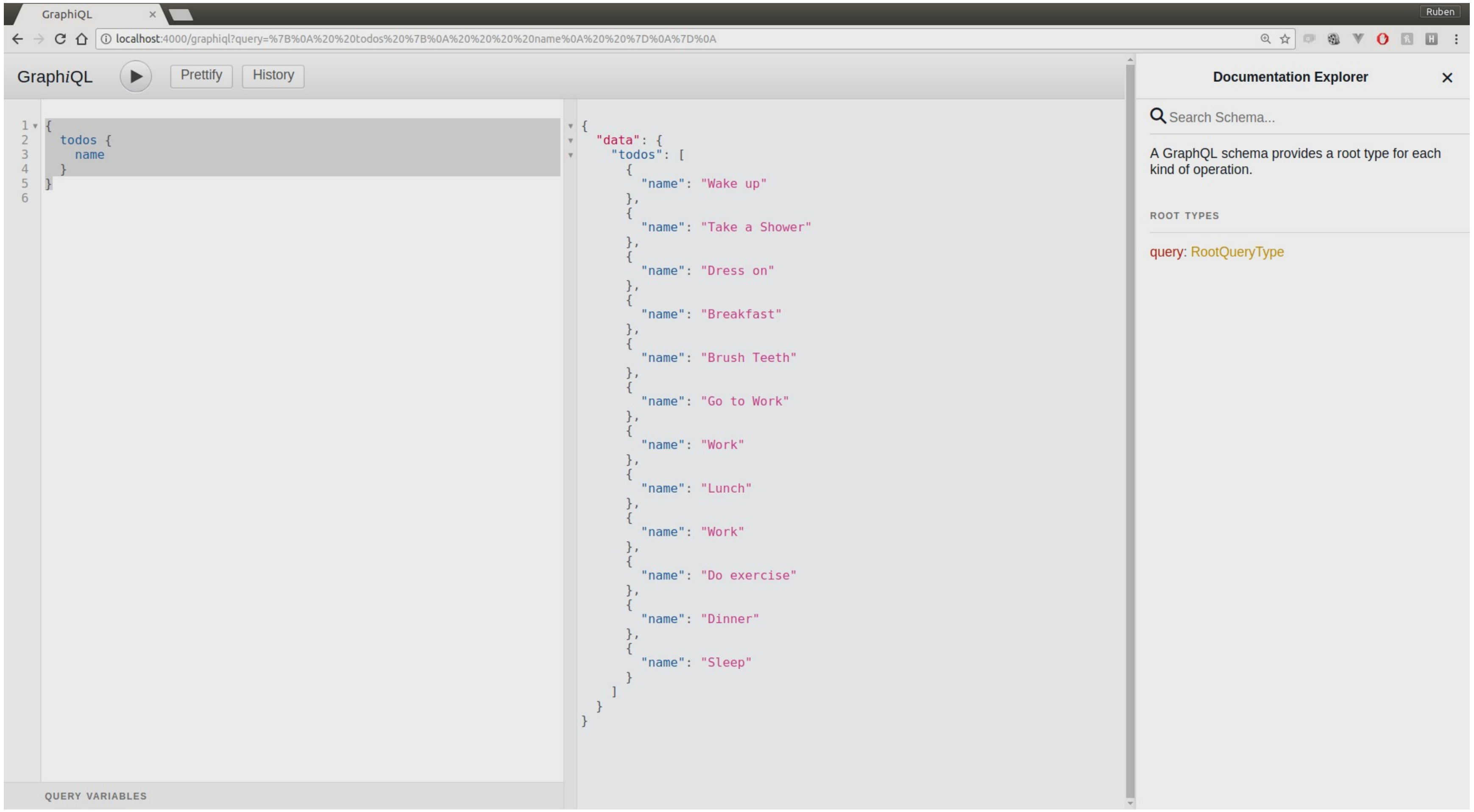
TODO GraphQL

Test



```
defmodule TodoWeb.Schema.Query.TodosTest do
  use TodoWeb.ConnCase, async: true

  setup do
    Code.load_file("priv/repo/seeds.exs")
  end

  @query """
  {
    todos{
      name
    }
  }
  """

  test "todos field return Todos" do
    conn = build_conn()
    conn = get conn, "/api", query: @query
    assert json_response(conn, 200) == %{
      "data" => %{
        "todos" => [
          %{"name" => "Wake up"},
          %{"name" => "Take a Shower"},
          %{"name" => "Dress on"},
          %{"name" => "Breakfast"},
          %{"name" => "Brush Teeth"},
          %{"name" => "Go to Work"},
          %{"name" => "Work"},
          %{"name" => "Lunch"},
          %{"name" => "Work"},
          %{"name" => "Do exercise"},
          %{"name" => "Dinner"},
          %{"name" => "Sleep"}
        ]
      }
    }
  end
end
~
~
~
"test/todo_web/schema/todos_test.exs" 38L, 833C                                                                 1,1           All
```

Query Sample


GraphiQL
localhost:4000/graphiql?query=%7B%0A%20%20todos%20%7B%0A%20%20%20%20name%0A%20%20%7D%0A%7D%0A
Ruben
GraphiQL
Prettify
History
Documentation Explorer
Search Schema...
QUERY VARIABLES

```
{
  todos {
    name
  }
}
```

```
{
  "data": {
    "todos": [
      {
        "name": "Wake up"
      },
      {
        "name": "Take a Shower"
      },
      {
        "name": "Dress on"
      },
      {
        "name": "Breakfast"
      },
      {
        "name": "Brush Teeth"
      },
      {
        "name": "Go to Work"
      },
      {
        "name": "Work"
      },
      {
        "name": "Lunch"
      },
      {
        "name": "Work"
      },
      {
        "name": "Do exercise"
      },
      {
        "name": "Dinner"
      },
      {
        "name": "Sleep"
      }
    ]
  }
}
```

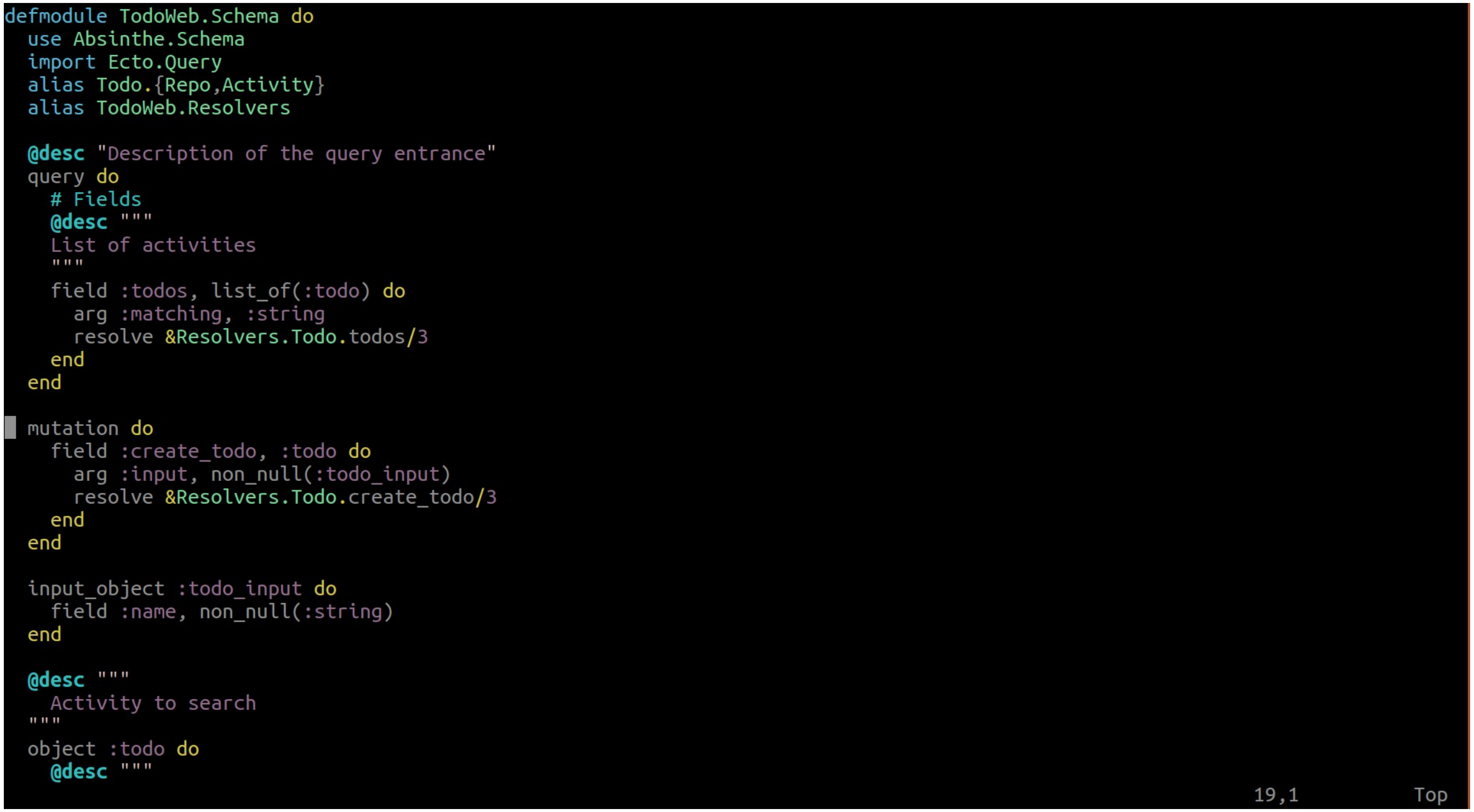
A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: RootQueryType

TODO GraphQL

Mutation



```
defmodule TodoWeb.Schema do
  use Absinthe.Schema
  import Ecto.Query
  alias Todo.{Repo,Activity}
  alias TodoWeb.Resolvers

  @desc "Description of the query entrance"
  query do
    # Fields
    @desc """
    List of activities
    """
    field :todos, list_of(:todo) do
      arg :matching, :string
      resolve &Resolvers.Todo.todos/3
    end
  end

  mutation do
    field :create_todo, :todo do
      arg :input, non_null(:todo_input)
      resolve &Resolvers.Todo.create_todo/3
    end
  end

  input_object :todo_input do
    field :name, non_null(:string)
  end

  @desc """
    Activity to search
  """
  object :todo do
    @desc """
                                                                                                              19,1          Top
```

Other Fun Stuff

- Use variables to do queries
  - Unions
  - Interfaces
  - Fragments
- Create your own Scalar types
  - Parse
  - Serialize
- Organize your code
  - Import fields
  - Import types
- Subscriptions
- Publishing your code “middleware”
- Authentication and Authorization
- Tunning Performance

References

- http://graphql.org/
- http://facebook.github.io/graphql/October2016/
- Williams, B. Willson, B. (2017). Craft GraphQL APIs in Elixir with Absinthe, Beta (Nov. 2017): The pragmatic programmers.

Next Meetups

- Elixir
  - Ecto
  - Test
  - OTP

Thanks!

Q & A?

@ramortegui

http://rubenamortegui.com

https://github.com/ramortegui