

Elixir: Scalable and Efficient Application Development

João Gonçalves

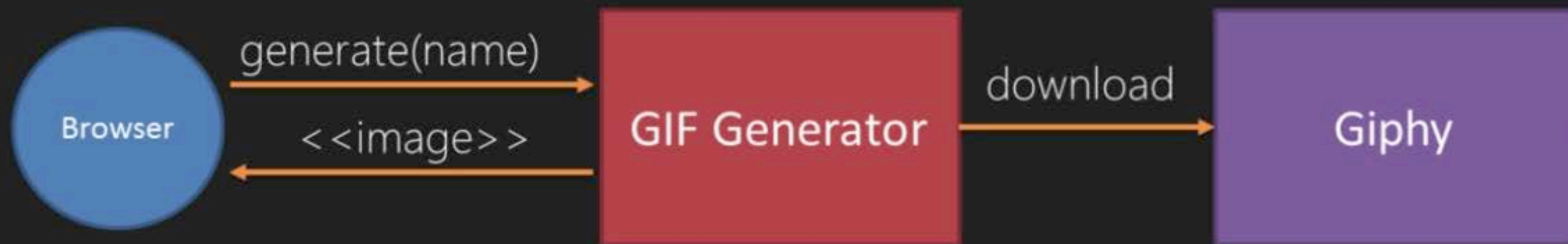
Creating an Application



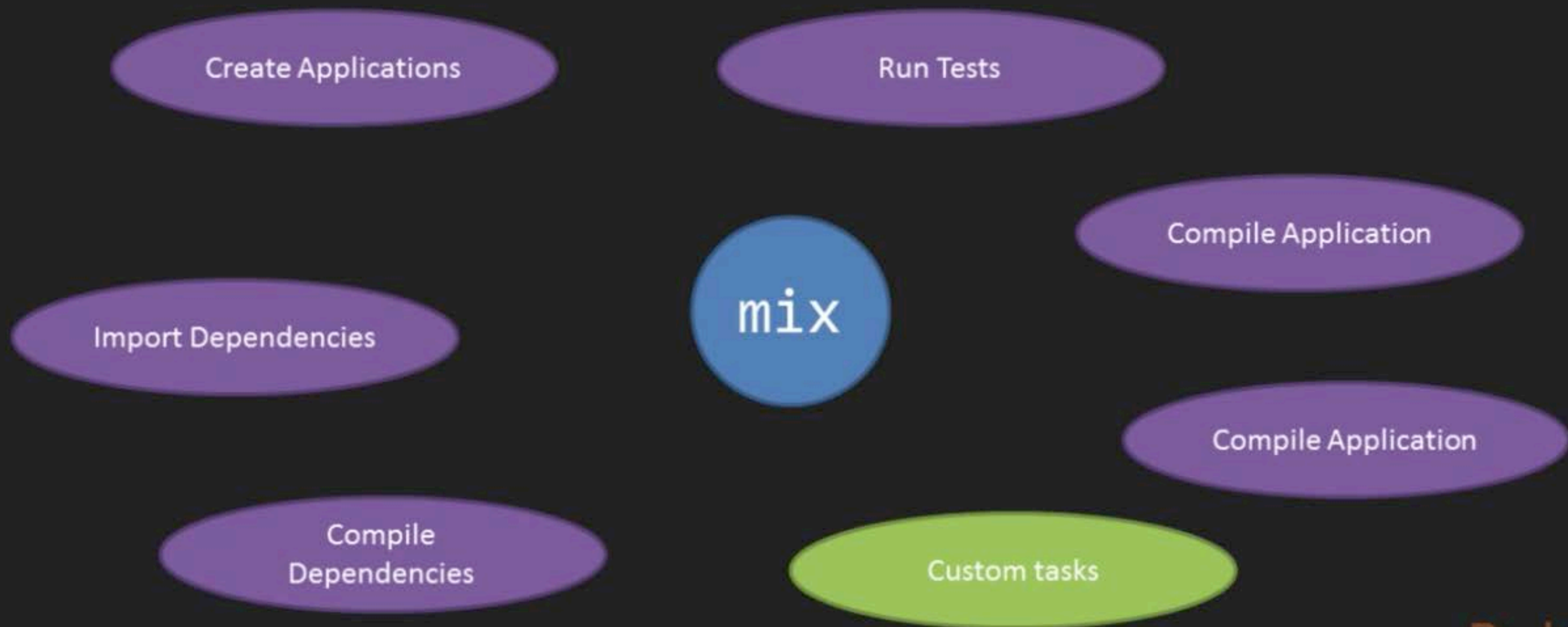
In this video, we are going to take a look at...

- Using mix to create an application
- Structure of an application
- Running interactively

The Application



Mix

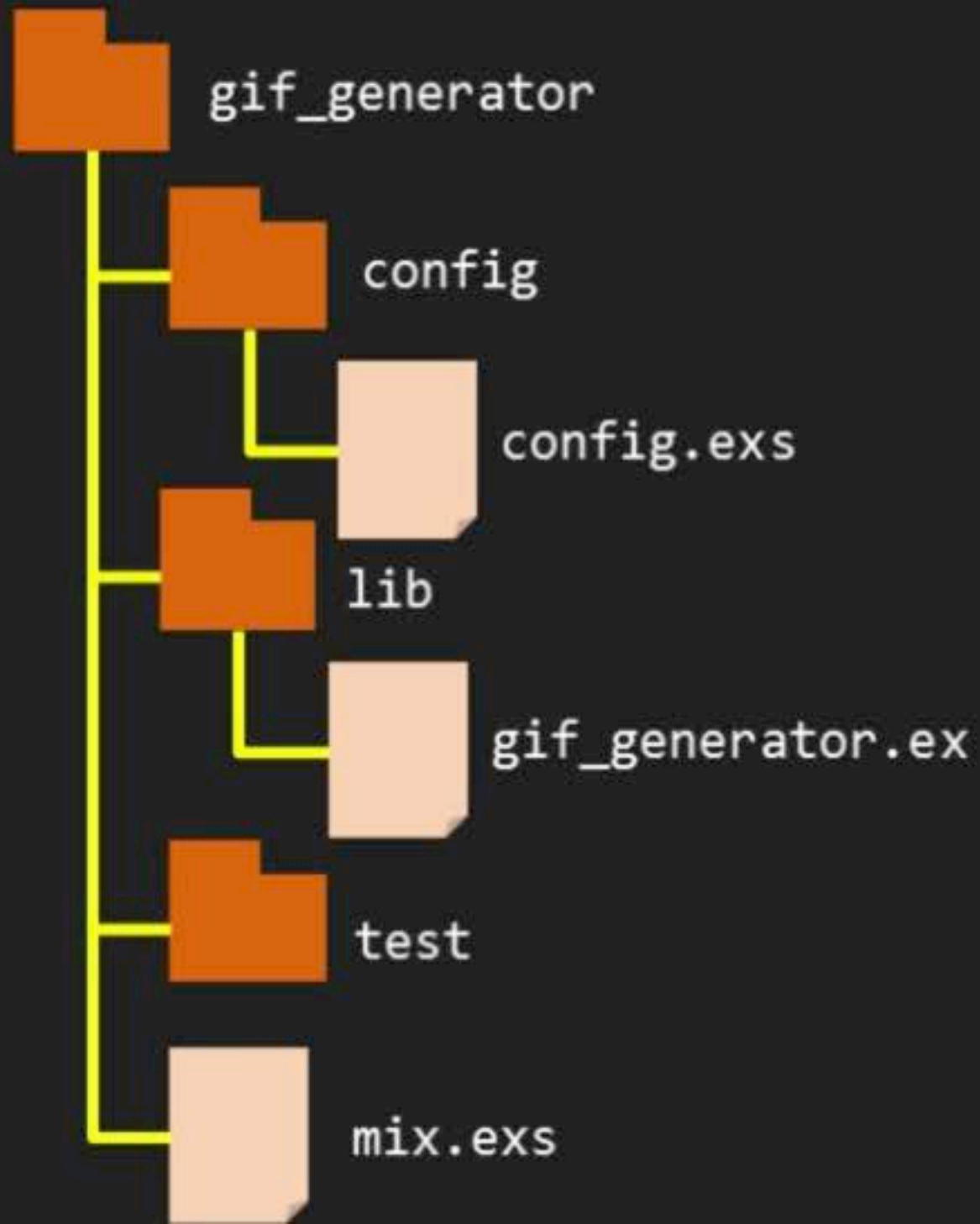


Creating an Application

Terminal

```
$projects> mix new gif_generator  
...  
$projects> cd gif_generator  
$gif_generator>
```

Anatomy of an Application



mix.exs

```
defmodule Generator.Mixfile do
  use Mix.Project

  def project do
    [
      ...
    ]
  end

  def application do
    [extra_applications: [:logger]]
  end

  defp deps do
    []
  end
end
```

Entering the Application Interactively

Terminal

```
$gif_generator> iex -S mix
```


Entering the Application Interactively

Terminal

```
$gif_generator> iex -S mix
```

```
iex(1)> GifGenerator.hello
```

```
:world
```

```
iex(2)>
```

João Gonçalves

Importing Dependencies



GIF Generator

GifGenerator

generate/1

GIF Generator

```
defmodule GifGenerator do  
  def generate(theme) do  
    {:ok, <<1,2,3,4,5>>}  
  end  
end
```

GIF Generator

```
defmodule GifGenerator do  
  def generate(theme) do  
    {:ok, <<1,2,3,4,5>>>}  
  end  
end
```

Get Image URL from Giphy

Download the image using the URL

Using HEX

[Packages](#)[Documentation ▾](#)[Log in](#)

The package manager for the Erlang ecosystem

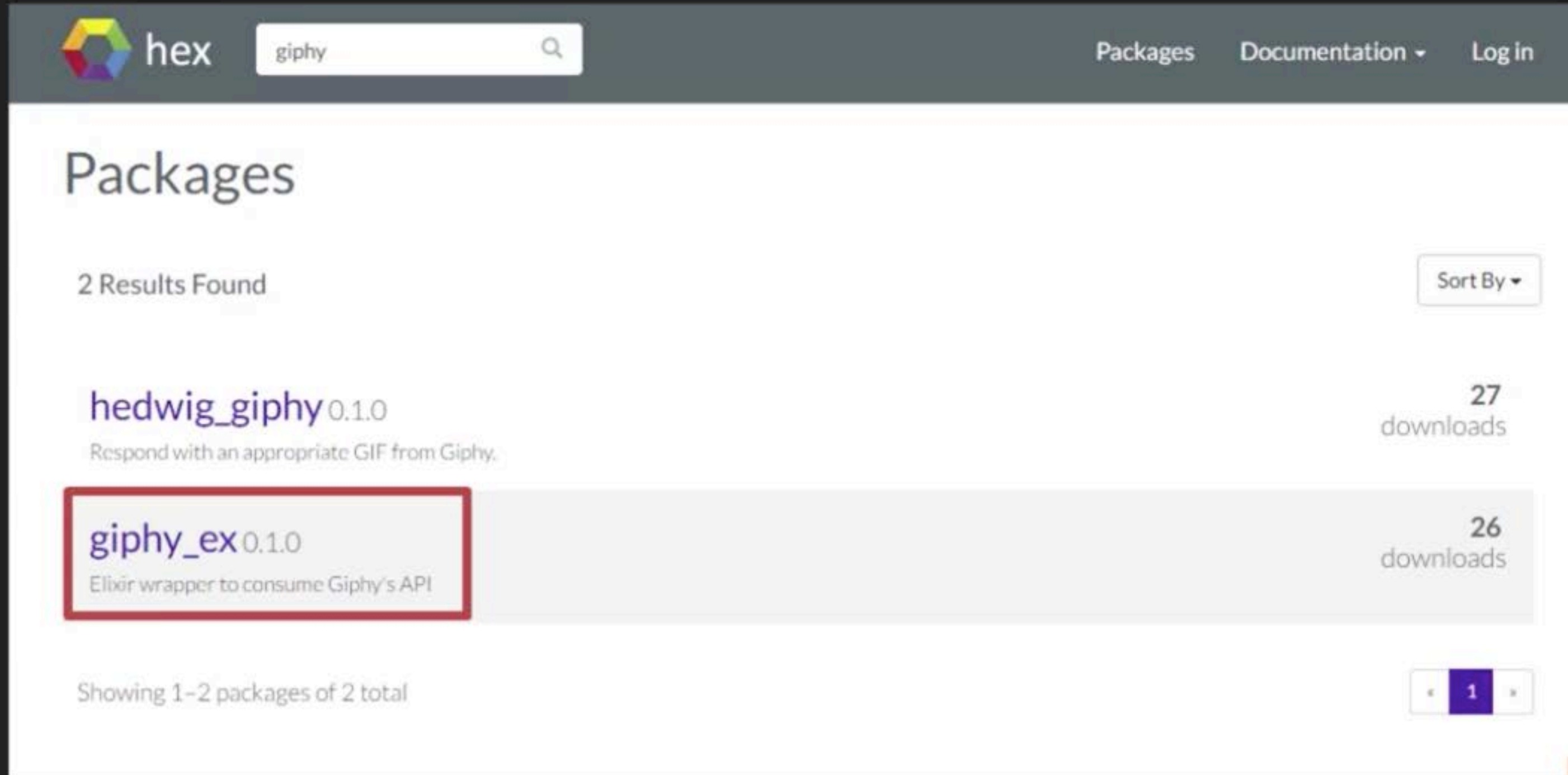


Using with Elixir



Simply specify your Mix dependencies as two-item tuples like `{:plug, "~> 1.1.0"}` and Elixir will ask if you want to install Hex if you haven't already. After installed, you can run `$ mix local` to see all available Hex tasks and `$ mix help TASK` for more information about a specific task.

Using HEX



The screenshot shows the Hex.pm website interface. At the top, there is a navigation bar with the Hex logo, a search bar containing the text 'giphy', and links for 'Packages', 'Documentation', and 'Log in'. Below the navigation bar, the main heading 'Packages' is displayed. Underneath, it says '2 Results Found'. To the right of this text is a 'Sort By' dropdown menu. The search results are listed in two rows. The first row shows the package 'hedwig_giphy' version 0.1.0, with a description 'Respond with an appropriate GIF from Giphy.' and '27 downloads'. The second row shows the package 'giphy_ex' version 0.1.0, with a description 'Elixir wrapper to consume Giphy's API' and '26 downloads'. The 'giphy_ex' package name and version are highlighted with a red rectangular border. At the bottom left, it says 'Showing 1-2 packages of 2 total'. At the bottom right, there is a pagination control showing '1' in a blue box, indicating the current page.

hex giphy

Packages Documentation Log in

Packages

2 Results Found Sort By

hedwig_giphy 0.1.0 27 downloads
Respond with an appropriate GIF from Giphy.

giphy_ex 0.1.0 26 downloads
Elixir wrapper to consume Giphy's API

Showing 1-2 packages of 2 total

Using HEX



hex

[Packages](#)[Documentation ▾](#)[Log in](#)

giphy_ex 0.1.0

Elixir wrapper to consume Giphy's API

Maintainers

dreamingechoes

Links

[Github](#)

[Online documentation \(download\)](#)

License

MIT



26

downloads



5

downloads



13

downloads



26

downloads

Config

mix.exs

```
{:giphy_ex, "~> 0.1.0"}
```



Build Tools

mix

Owners



dreamingechoes

Adding GiphyEX

```
defmodule GifGenerator.Mixfile do
  use Mix.Project

  def project do
    [
      ...
    ]
  end

  def application do
    [extra_applications: [:logger]]
  end

  defp deps do
    [{:giphy_ex, "~> 0.1.0"}]
  end
end
```

Adding GiphyEX

```
use Mix.Config
```

```
...
```

```
config :giphy_ex, :api_key, "dc6zaTOxFJmzC"
```

Getting the Dependency

Terminal

```
$gif_generator> mix deps.get
```

Getting the Dependency

Terminal

```
$gif_generator> mix deps.get  
...  
$gif_generator> mix deps.compile
```


Getting the Dependency

Terminal

```
$gif_generator> mix deps.get  
...  
$gif_generator> mix deps.compile  
...  
$gif_generator> iex -S mix
```

Testing the Dependency

Terminal

```
iex(1)> GiphyEx.Gifs.random("banana")
```

Testing the Dependency

Terminal

```
iex(1)> GiphyEx.Gifs.random("banana")
%{"data" =>
  %{"caption" => "",
    ...
    ...
    "image_url" => "...."
  }
}
```

GIF Generator 2.0

```
defmodule GifGenerator do

  def generate(theme) do
    {:ok, <<1,2,3,4,5>>}
  end

  defp image_url(theme) do
    %{"data" => %{"image_url" => image_url}} =
      GiphyEx.Gifs.random(theme)

    {:ok, image_url}
  end
end
```

GIF Generator 2.0

```
defmodule GifGenerator do

  def generate(theme) do
    {:ok, <<1,2,3,4,5>>}
  end

  defp image_url(theme) do
    %{"data" => %{"image_url" => image_url}} =
      GiphyEx.Gifs.random(theme)

    {:ok, image_url}
  end

  defp download_image(url) do
    {:ok, <<1,2,3,4,5>>}
  end
end
```


GIF Generator 2.0

```
defmodule GifGenerator do

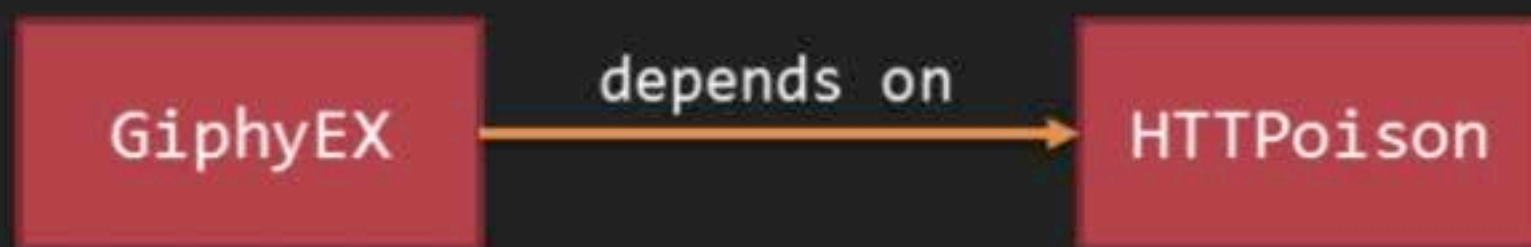
  def generate(theme) do
    with {:ok, url} <- image_url(theme),
         {:ok, binary} <- download_image(url) do
      {:ok, binary}
    end
  end

  defp image_url(theme) do
    %{“data” => %{“image_url” => image_url}} =
      GiphyEx.Gifs.random(theme)

    {:ok, image_url}
  end

  defp download_image(url) do
    {:ok, <<1,2,3,4,5>>}
  end
end
```


Downloading



GIF Generator 2.0

```
defmodule GifGenerator do

  def generate(theme) do
    with {:ok, url} <- image_url(theme),
         {:ok, binary} <- download_image(url) do
      {:ok, binary}
    end
  end

  defp image_url(theme) do
    %{“data” => %{“image_url” => image_url}} =
      GiphyEx.Gifs.random(theme)

    {:ok, image_url}
  end

  defp download_image(url) do
    {:ok, %HTTPoison.Response{body: binary}} =
      HTTPoison.get(url)

    {:ok, binary}
  end
end
```

Testing it All

Terminal

```
iex(1)> GifGenerator.generate("hello")
```

Testing it All

Terminal

```
iex(1)> GifGenerator.generate("hello")  
{:ok, <<342, 324, 21, 43, ... >>}
```

João Gonçalves

Running Tests



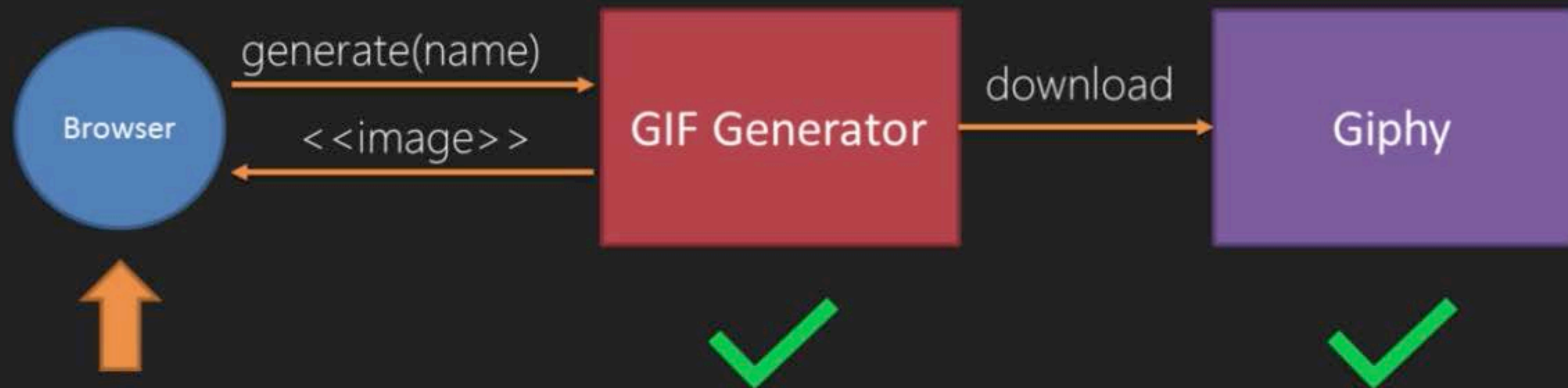
In this video, we are going to take a look at...

- The complete GIF generator application
- Running automated tests on our application

The Application



The Application



Creating the Browser Endpoint



hex

Find packages



Packages

Documentation ▾

Log in

maru 0.11.3

REST-like API micro-framework for elixir inspired by grape.

Maintainers

Xiangrong Hao

Links

[Github](#)

[Online documentation \(download\)](#)

License

BSD 3-Clause



1 663

downloads
this version



130

downloads
yesterday



856

downloads
last 7 days



42 632

downloads
all time

Config

mix.exs

`{:maru, "~> 0.11.3"}`



Build Tools

[mix](#)

Owners



[falood](#)

Creating the Browser Endpoint

```
defmodule GifGenerator.Mixfile do
  use Mix.Project

  ...

  def application do
    [extra_applications: [:logger, :giphy_ex, :maru]]
  end

  defp deps do
    [
      {:giphy_ex, "~> 0.1.0"},
      {:maru, "~> 0.11.3"}
    ]
  end
end
```

Creating the Browser Endpoint

```
use Mix.Config
```

```
...
```

```
config :giphy_ex, :api_key, "dc6zaTOxFJmzC"
```

```
→ config :maru, GifGenerator.API,  
    http: [port: 9830]
```


Creating the Browser Endpoint

```
defmodule GifGenerator.API do
  use Maru.Router

  route_param :theme do
    def get do
      {:ok, image} = params[:theme]
      |> GifGenerator.generate

      conn
      |> put_resp_content_type("image/gif")
      |> send_resp(200, image)
      |> halt
    end
  end
end
```


Wrapping All Up

Terminal

```
$gif_generator> mix deps.get  
...  
$gif_generator> iex -S mix
```

Wrapping All Up

- <http://localhost:9830/elixir>

Running Tests

“Manual”



Unit



Running Tests

Terminal

```
$gif_generator> MIX_ENV=test mix test
```

```
Finished in 0.01 seconds
```

```
1 test, 0 failures
```

```
Randomized with seed 432445
```

```
$gif_generator>
```

Running Tests



Running Tests



Running Tests

- Use different modules during testing

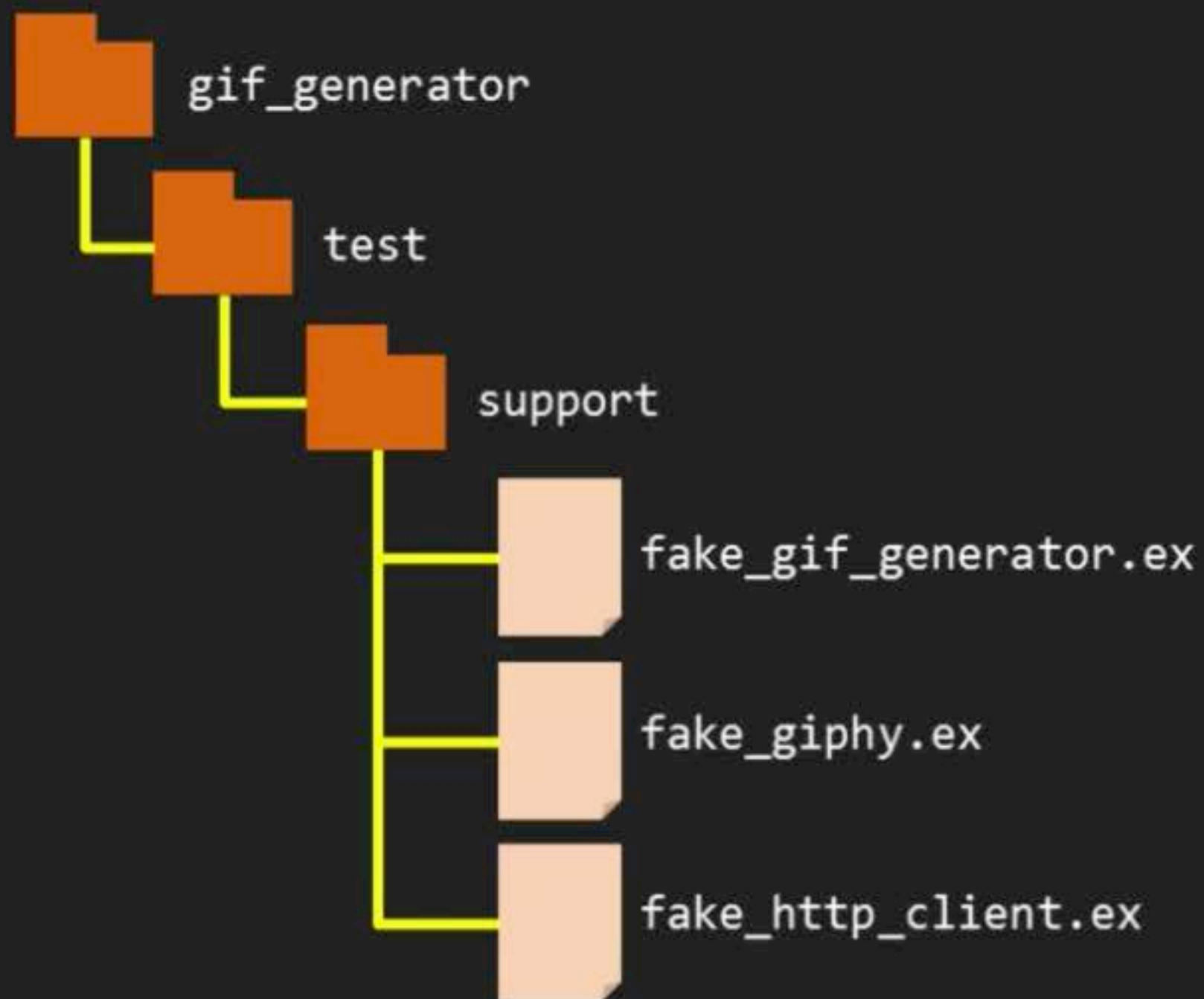


Using Test Doubles

- Use different modules during testing



Using Test Doubles




Using Test Doubles

 fake_gif_generator.ex

```
defmodule FakeGifGenerator do
  def generate(theme) do
    {:ok, <<1,2,3,4,5,6>>}
  end
end
```

 fake_giphy.ex

```
defmodule FakeGiphy do
  def random(theme) do
    %{"data" => %{"image_url" => "a_url"}}
  end
end
```

 fake_http_client.ex

```
defmodule FakeHttpClient do
  def get(url) do
    {:ok, %HTTPoison.Response{body: <<1,2,3,4,5,6>>}}
  end
end
```


Using Test Doubles

```
defmodule GifGenerator do
  → @giphy Application.get_env(:gif_generator, :giphy)
    @http_client Application.get_env(:gif_generator, :http_client)

    def generate(theme) do
      ...
    end

    defp image_url(theme) do
      → %{“data” => %{“image_url” => image_url}} =
        @giphy.random(theme)

        { :ok, image_url }
      end

    defp download_image(url) do
      → { :ok, %HTTPoison.Response{body: binary} } =
        @http_client.get(url)

        { :ok, binary }
      end
    end
end
```

Using Test Doubles

```
use Mix.Config

...

config :giphy_ex, :api_key, "dc6zaTOxFJmzC"

config :maru, GifGenerator.API,
  http: [port: 9830]

config :gif_generator,
  generator: GifGenerator,
  giphy: GiphyEx.Gifs,
  http_client: HTTPoison
```



Using Test Doubles

```
use Mix.Config

...

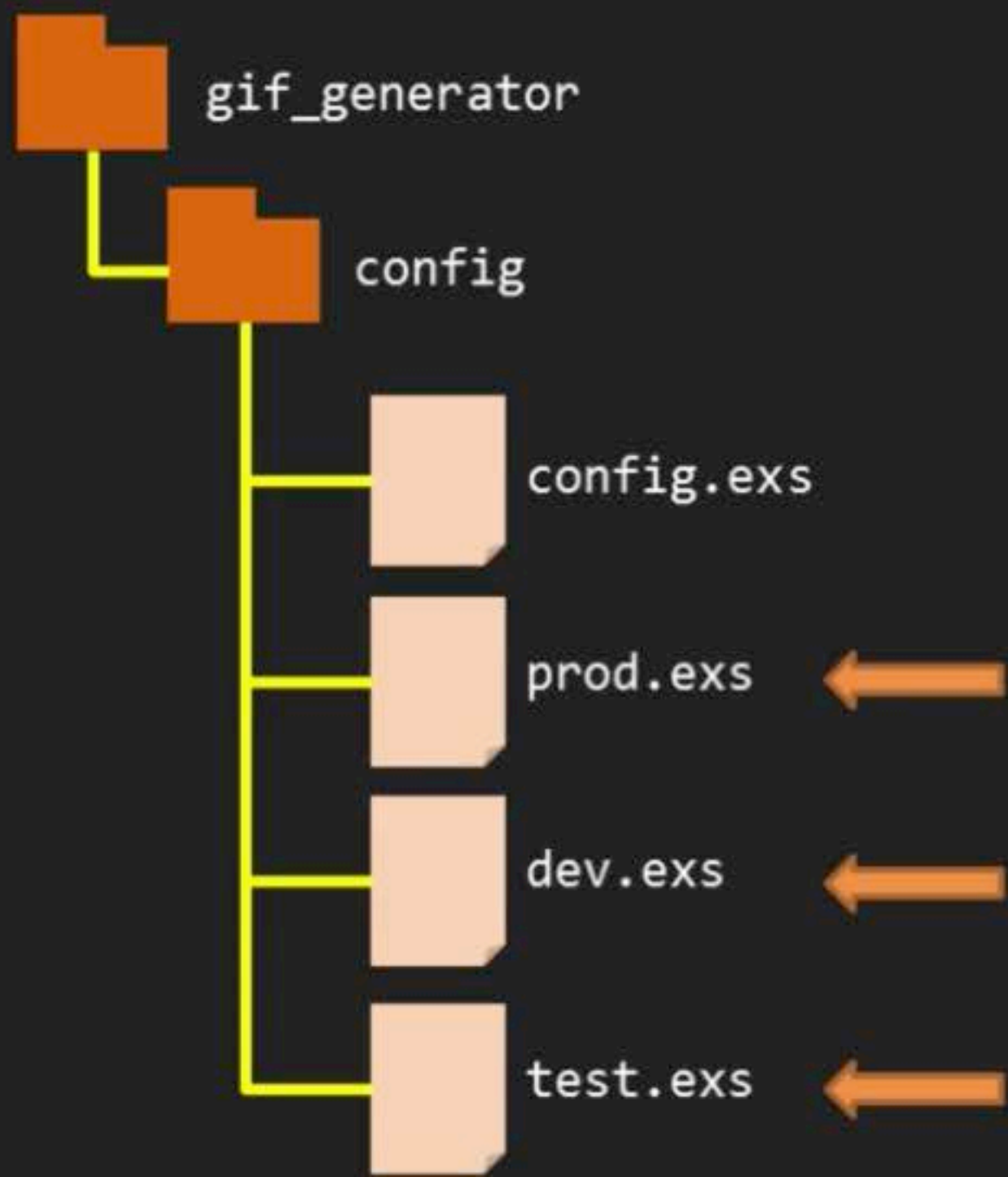
config :giphy_ex, :api_key, "dc6zaTOxFJmzC"

config :maru, GifGenerator.API,
  http: [port: 9830]

config :gif_generator,
  generator: GifGenerator,
  giphy: GiphyEx.Gifs,
  http_client: HTTPoison

➡ import_config "#{Mix.env}.exs"
```

Test-Specific Configurations



Test-Specific Configurations



test.exs

```
use Mix.Config
```

```
config :gif_generator,  
  generator: FakeGifGenerator,  
  giphy: FakeGiphy,  
  http_client: FakeHttpClient
```



Test-Specific Configurations

```
defmodule GifGenerator.Mixfile do
  use Mix.Project

  def project do
    [
      ...
      elixirc_paths: elixirc_paths(Mix.env) ←
    ]
  end

  ...

  defp elixirc_paths(:test), do: ["lib", "test/support"] ←
  defp elixirc_paths(_), do: ["lib"]
end
```

Testing GifGenerator



test/gif_generator_test.exs

```
defmodule GifGeneratorTest do
  use ExUnit.Case
  doctest GifGenerator

  test "generate" do
    assert GifGenerator.generate("foo") == {:ok, <<1,2,3,4,5,6>>}
  end
end
```

Quiz Time!

Quiz 9 | 2 Questions

Start Quiz

Skip Quiz >

Question 1:

Which of the following function is included in the IO module?

☐ `delete`

☐ `view`

☐ `execute`

☐ `write`

Question 2:

Which of the following functions is used for accepting user input?

☐ write

☐ read

☐ gets

☐ puts