TESTS UNITAIRES EN ELIXIR

Trucs et astuces

Nicolas Talfer | meetup #24

Le projet

Tous les matins du lundi au vendredi

Récupérer un gif aléatoire sur Les Joies du code

Et le poster dans un channel Slack

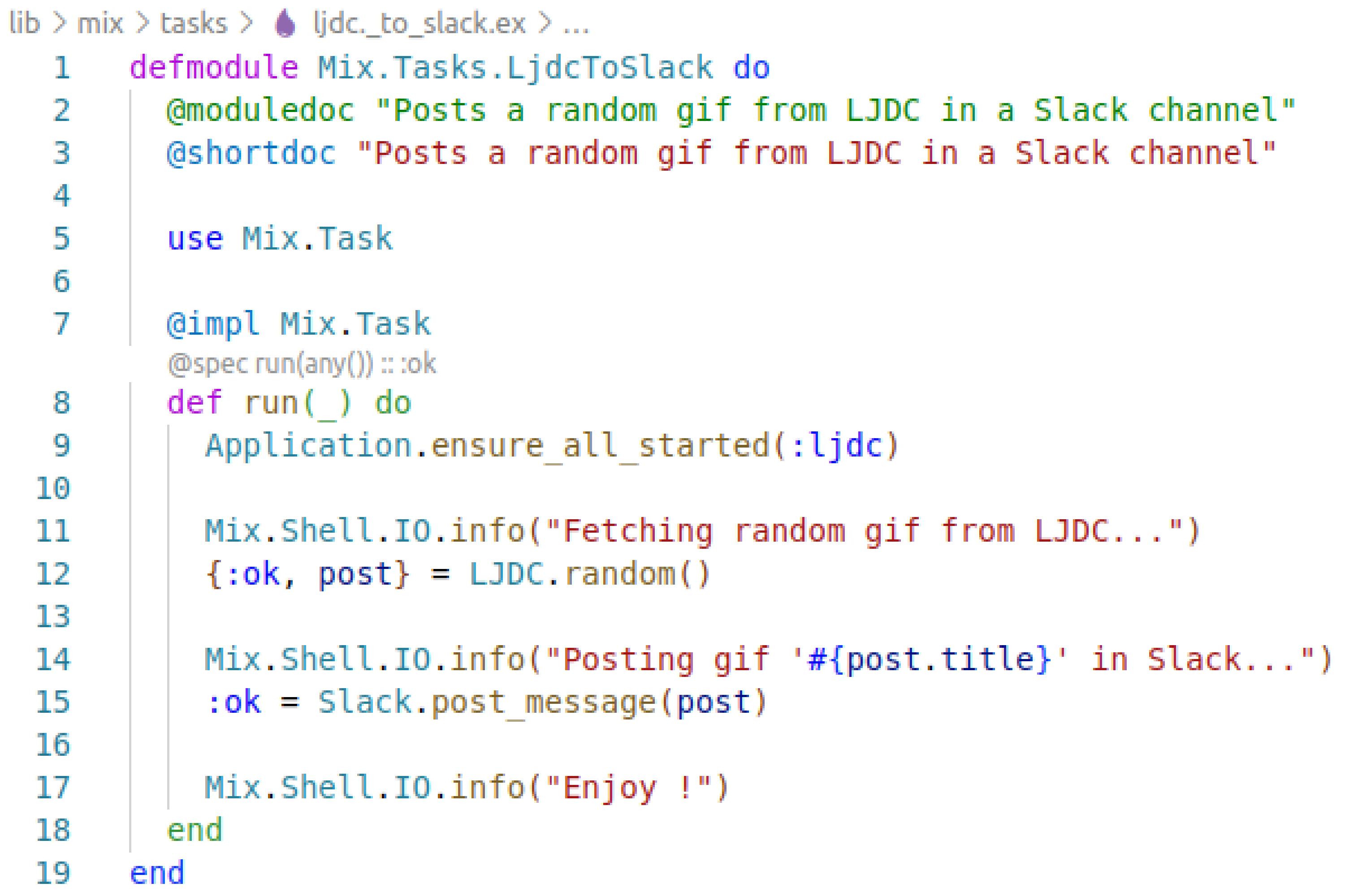
Pour commencer sa journée de travail de bonne humeur

Le cron

```
nico@otter:~$ crontab -l
SHELL=/bin/bash
```

$$\underbrace{\texttt{0 9 * * 1,2,3,4,5}}_{\text{du lundi au vendredi à 9h}}\;\underbrace{\texttt{. /home/nico/.asdf/asdf.sh;}}_{\text{script asdf pour mettre Elixir dans PATH}}\;\underbrace{\texttt{cd \textasciitilde/products/ljdc \&\& mix ljdc\_to\_slack}}_{\text{exécution d'une tâche mix}}$$

La tâche mix


lib > mix > tasks > ljdc_to_slack.ex > ...

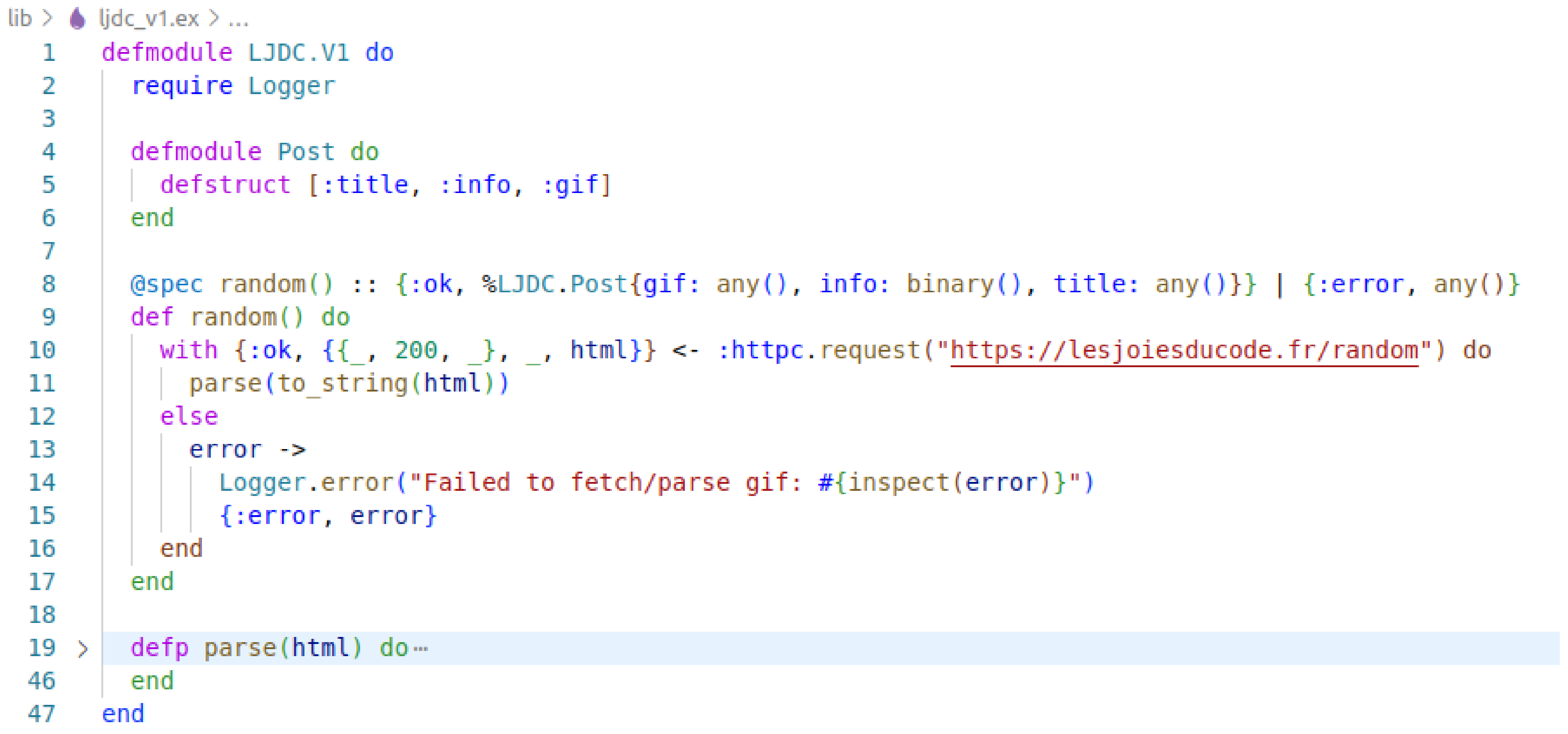
```elixir
defmodule Mix.Tasks.LjdcToSlack do
  @moduledoc "Posts a random gif from LJDC in a Slack channel"
  @shortdoc "Posts a random gif from LJDC in a Slack channel"

  use Mix.Task

  @impl Mix.Task
  @spec run(any()) :: :ok
  def run(_) do
    Application.ensure_all_started(:ljdc)

    Mix.Shell.IO.info("Fetching random gif from LJDC...")
    {:ok, post} = LJDC.random()

    Mix.Shell.IO.info("Posting gif '#{post.title}' in Slack...")
    :ok = Slack.post_message(post)

    Mix.Shell.IO.info("Enjoy !")
  end
end
```

Récupération d'un gif sur les joies du code

Pas d'API -> scraping de la page web quand on clique sur le bouton "RANDOM" avec la librarie floki

```
nico@otter:~$ curl -iLs https://lesjoiesducode.fr/random
HTTP/2 302
location: https://lesjoiesducode.fr/quand-mon-code-ne-fonctionne-pas-comme-prvu

HTTP/2 200
content-type: text/html; charset=UTF-8
<!doctype html>
...
```

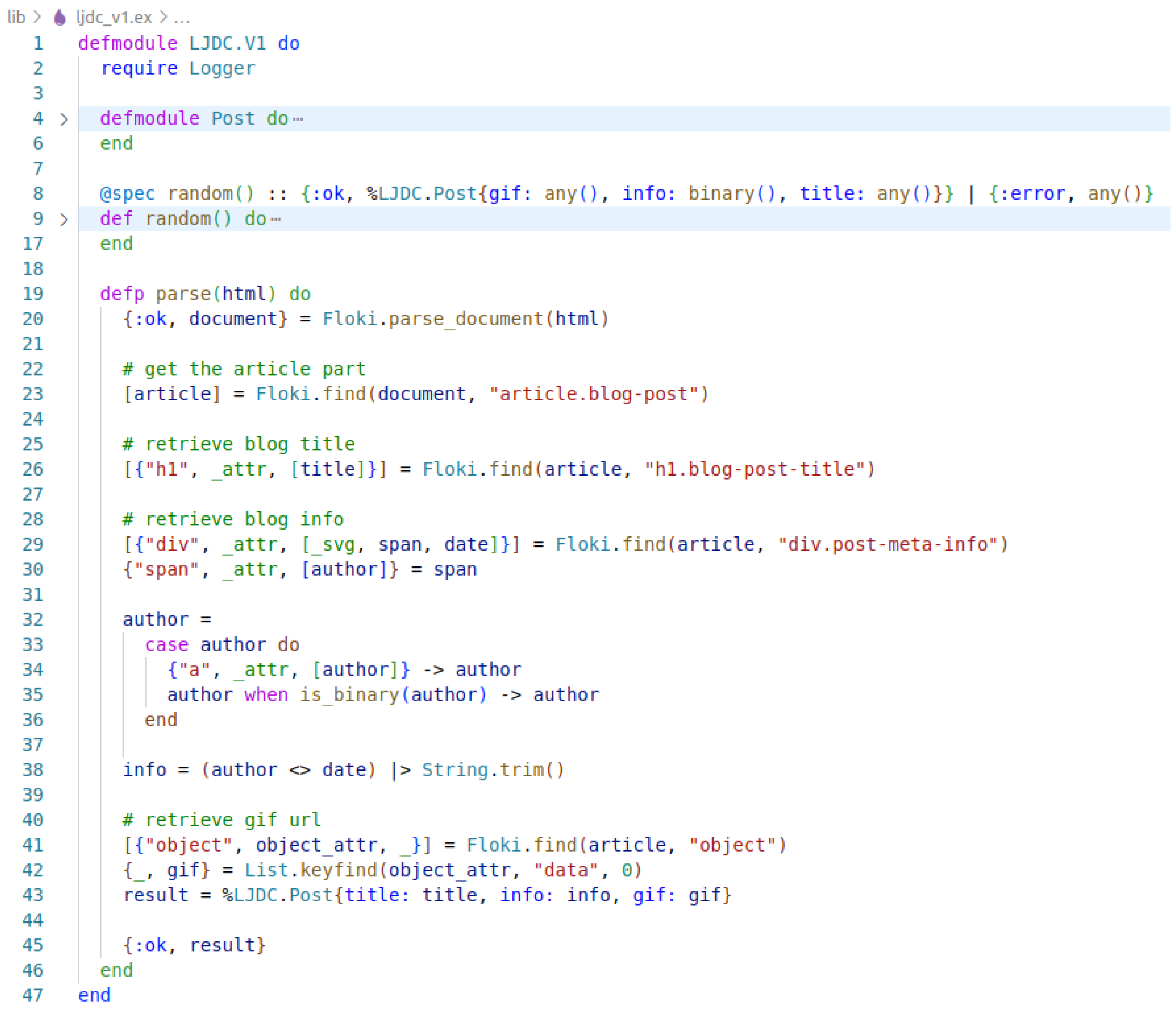
Une 1ère implémentation...



lib > ljdc_v1.ex > ...

```elixir
defmodule LJDC.V1 do
  require Logger

  defmodule Post do
    defstruct [:title, :info, :gif]
  end

  @spec random() :: {:ok, %LJDC.Post{gif: any(), info: binary(), title: any()}} | {:error, any()}
  def random() do
    with {:ok, {{_, 200, _}, _, html}} <- :httpc.request("https://lesjoiesducode.fr/random") do
      parse(to_string(html))
    else
      error ->
        Logger.error("Failed to fetch/parse gif: #{inspect(error)}")
        {:error, error}
    end
  end

  defp parse(html) do⋯
  end
end
```

Une 1ère implémentation…

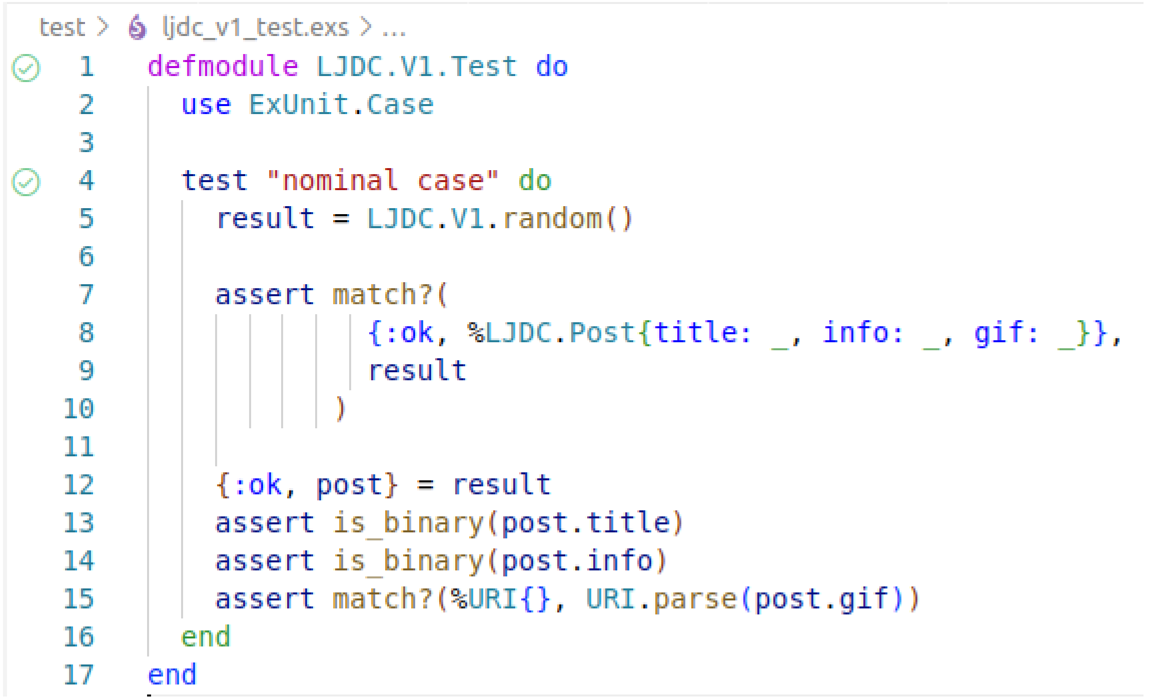


```elixir
defmodule LJDC.V1 do
  require Logger

  defmodule Post do⋯
  end

  @spec random() :: {:ok, %LJDC.Post{gif: any(), info: binary(), title: any()}} | {:error, any()}
  def random() do⋯
  end

  defp parse(html) do
    {:ok, document} = Floki.parse_document(html)

    # get the article part
    [article] = Floki.find(document, "article.blog-post")

    # retrieve blog title
    [{"h1", _attr, [title]}] = Floki.find(article, "h1.blog-post-title")

    # retrieve blog info
    [{"div", _attr, [_svg, span, date]}] = Floki.find(article, "div.post-meta-info")
    {"span", _attr, [author]} = span

    author =
      case author do
        {"a", _attr, [author]} -> author
        author when is_binary(author) -> author
      end

    info = (author <> date) |> String.trim()

    # retrieve gif url
    [{"object", object_attr, _}] = Floki.find(article, "object")
    {_, gif} = List.keyfind(object_attr, "data", 0)
    result = %LJDC.Post{title: title, info: info, gif: gif}

    {:ok, result}
  end
end
```

Quand mon code fonctionne du premier coup et que je me rappelle que j'ai encore rien testé

Les Joies du Code, commit du 24 Mai 2024

Testons cette 1ère implementation...

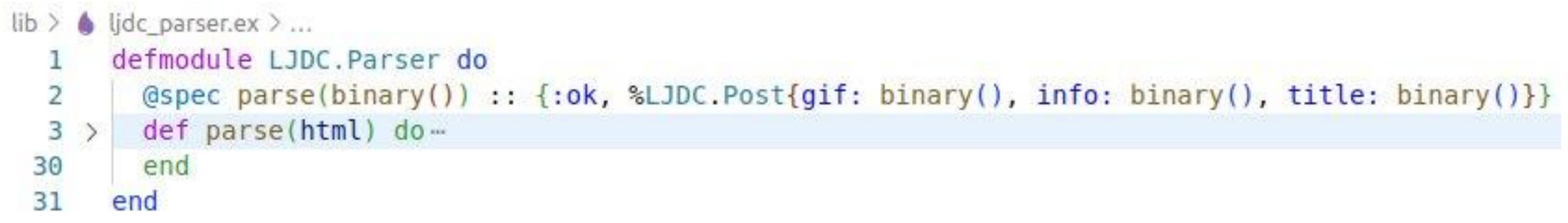

test > ljdc_v1_test.exs > ...

```elixir
defmodule LJDC.V1.Test do
  use ExUnit.Case

  test "nominal case" do
    result = LJDC.V1.random()

    assert match?(
             {:ok, %LJDC.Post{title: _, info: _, gif: _}},
             result
           )

    {:ok, post} = result
    assert is_binary(post.title)
    assert is_binary(post.info)
    assert match?(%URI{}, URI.parse(post.gif))
  end
end
```

✅ un 1er test du cas nominal
...mais c'est tout !

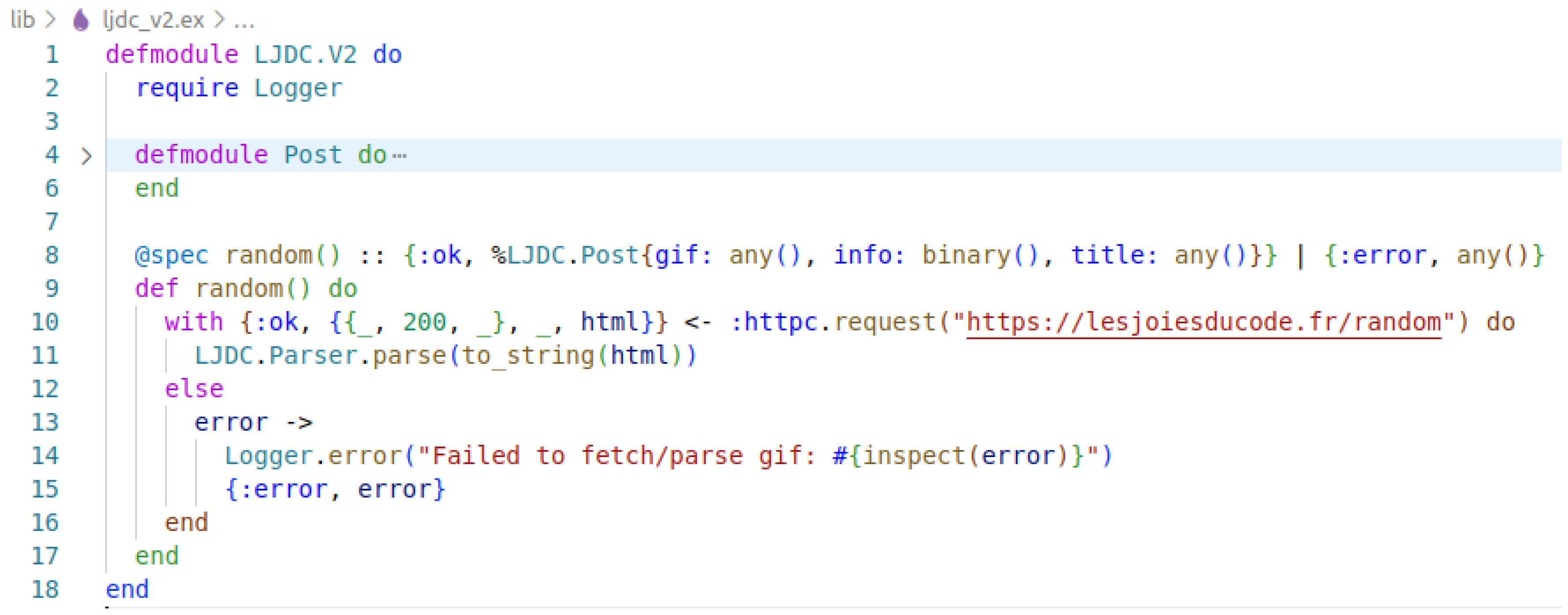
❌ pas sûr à 100% qu'on parse bien
❌ pas de test des cas d'erreur
❌ vraie connection HTTP à l'exécution du test
(accès internet nécessaire, le site doit être dispo,
une lenteur de la connection entraîne une lenteur du test
heureusement ce n'est pas une API HTTP payante)
❌ impossible de tester unitairement la fonction de parsing

V2 : LJDC.Parser



lib > ljdc_parser.ex > ...

```elixir
defmodule LJDC.Parser do
  @spec parse(binary()) :: {:ok, %LJDC.Post{gif: binary(), info: binary(), title: binary()}}
  def parse(html) do…
  end
end
```



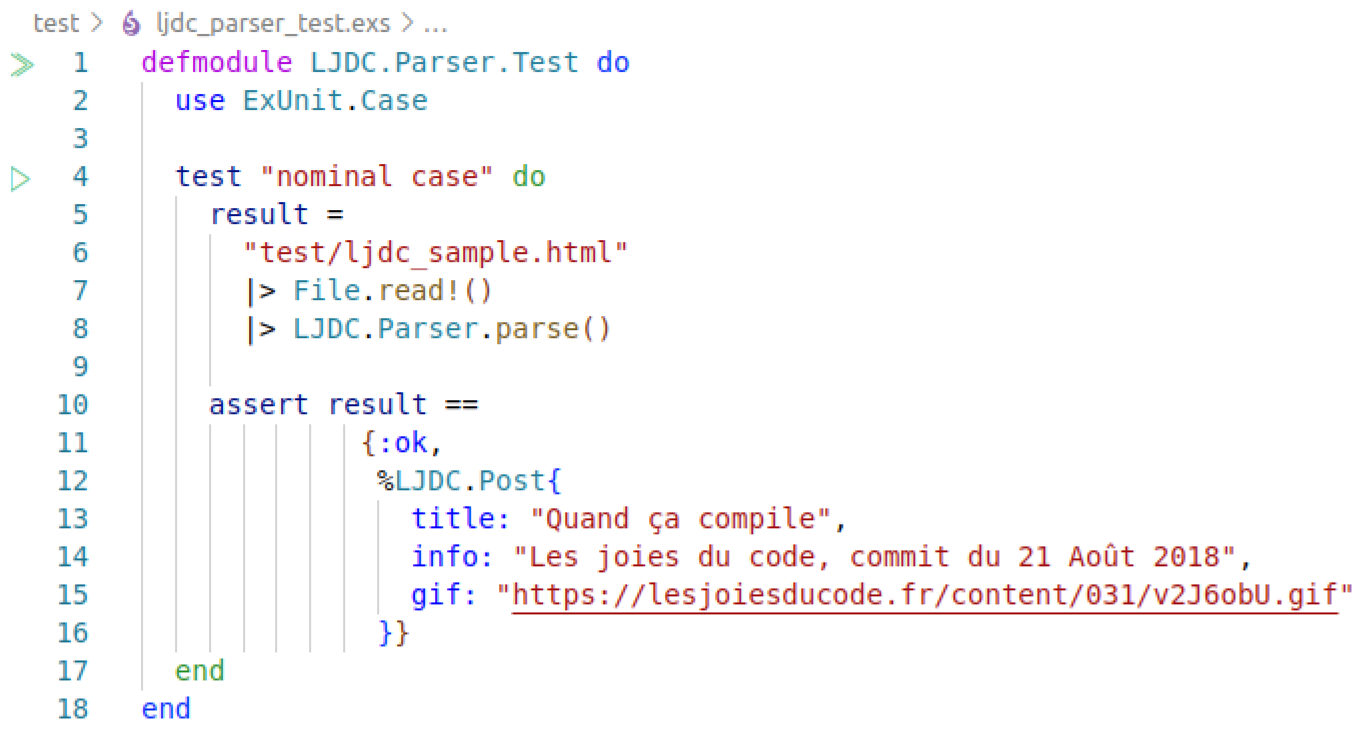
lib > ljdc_v2.ex > ...

```elixir
defmodule LJDC.V2 do
  require Logger

  defmodule Post do…
  end

  @spec random() :: {:ok, %LJDC.Post{gif: any(), info: binary(), title: any()}} | {:error, any()}
  def random() do
    with {:ok, {{_, 200, _}, _, html}} <- :httpc.request("https://lesjoiesducode.fr/random") do
      LJDC.Parser.parse(to_string(html))
    else
      error ->
        Logger.error("Failed to fetch/parse gif: #{inspect(error)}")
        {:error, error}
    end
  end
end
```

Test u de LJDC.Parser

Récupération un example pour les tests :

curl -sL "http://lesjoiesducode.fr/random" > ljdc_sample.html



test > ljdc_parser_test.exs > ...

```elixir
defmodule LJDC.Parser.Test do
  use ExUnit.Case

  test "nominal case" do
    result =
      "test/ljdc_sample.html"
      |> File.read!()
      |> LJDC.Parser.parse()

    assert result ==
                {:ok,
                 %LJDC.Post{
                   title: "Quand ça compile",
                   info: "Les joies du code, commit du 21 Août 2018",
                   gif: "https://lesjoiesducode.fr/content/031/v2J6obU.gif"
                 }}
  end
end
```

✅ test exhaustif et offline

Tester c'est douter.

Les Joies du Code, commit du 14 Mai 2020

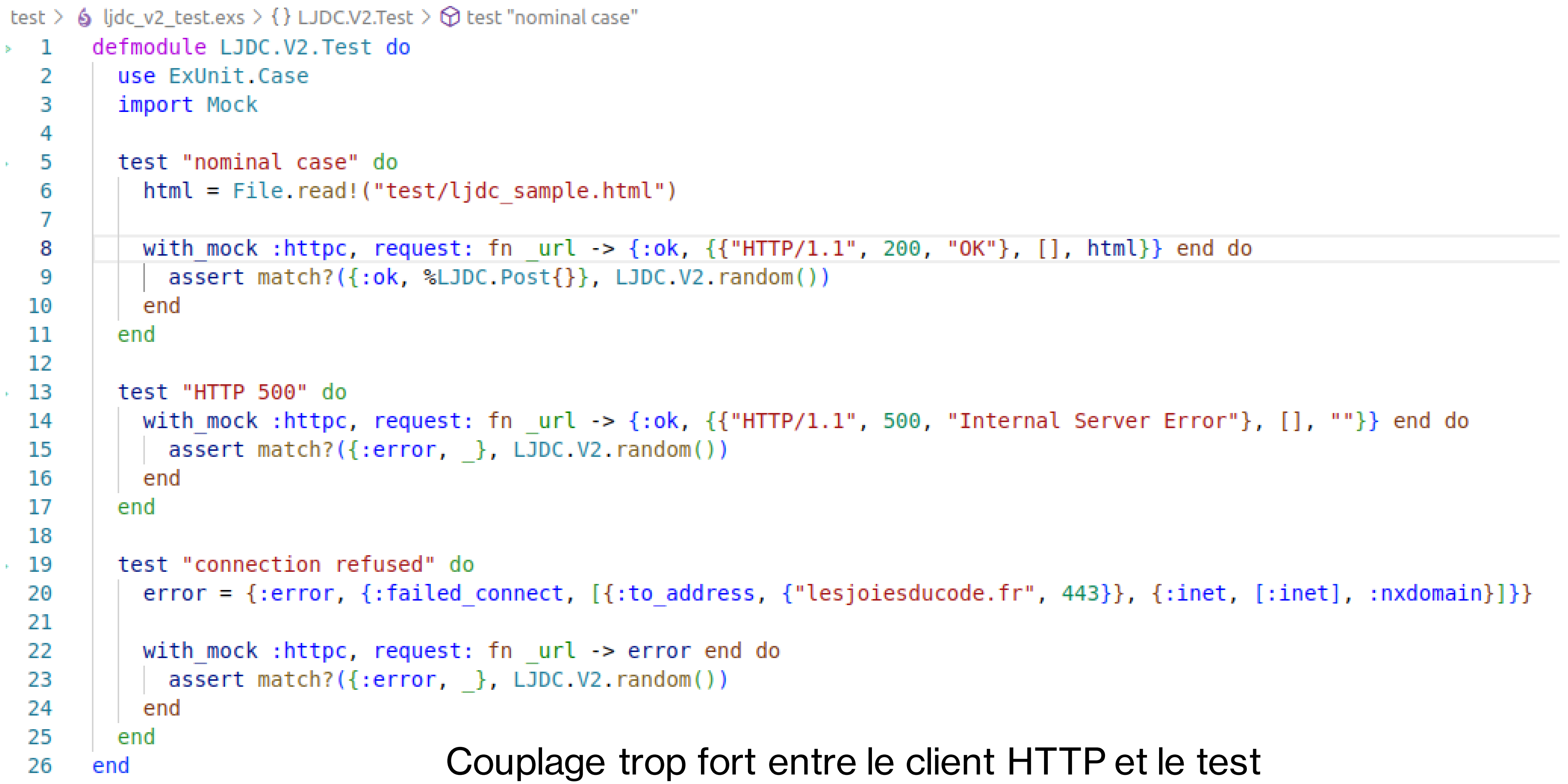
J'ai
J'ai
testé
testé
mon code
mon code
J'ai testé mon code
J'ai livré en prod
> lesjoiesducode.fr

Quand on me demande si j'ai bien tout testé

LSteen, commit du 3 Déc 2020

Test u de LJDC.V2

Utilisation de mocks pour tester le cas nominal et des cas d'erreur offline (librairie mock)

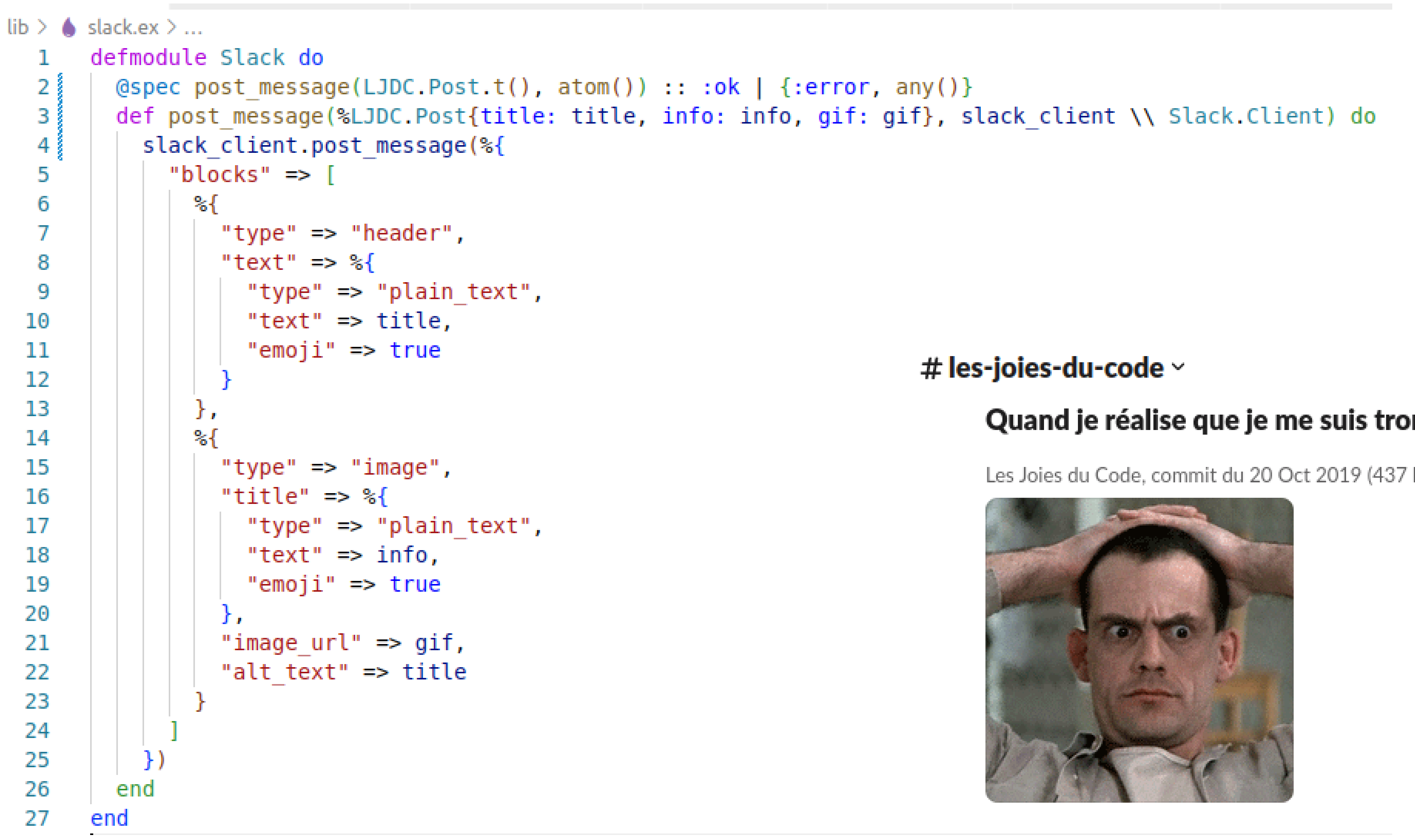


test > ljdc_v2_test.exs > {} LJDC.V2.Test > test "nominal case"

```elixir
defmodule LJDC.V2.Test do
  use ExUnit.Case
  import Mock

  test "nominal case" do
    html = File.read!("test/ljdc_sample.html")

    with_mock :httpc, request: fn _url -> {:ok, {{"HTTP/1.1", 200, "OK"}, [], html}} end do
      assert match?({:ok, %LJDC.Post{}}, LJDC.V2.random())
    end
  end

  test "HTTP 500" do
    with_mock :httpc, request: fn _url -> {:ok, {{"HTTP/1.1", 500, "Internal Server Error"}, [], ""}} end do
      assert match?({:error, _}, LJDC.V2.random())
    end
  end

  test "connection refused" do
    error = {:error, {:failed_connect, [{:to_address, {"lesjoiesducode.fr", 443}}, {:inet, [:inet], :nxdomain}]}}

    with_mock :httpc, request: fn _url -> error end do
      assert match?({:error, _}, LJDC.V2.random())
    end
  end
end
```

Couplage trop fort entre le client HTTP et le test
Si on modifie le client HTTP, on doit modifier le test !

Passons à Slack et supprimons le couplage...



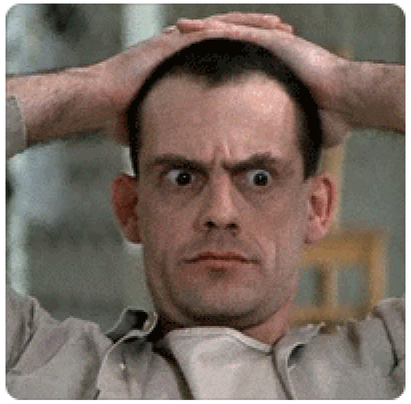
lib > slack.ex > ...

```elixir
defmodule Slack do
  @spec post_message(LJDC.Post.t(), atom()) :: :ok | {:error, any()}
  def post_message(%LJDC.Post{title: title, info: info, gif: gif}, slack_client \\ Slack.Client) do
    slack_client.post_message(%{
      "blocks" => [
        %{
          "type" => "header",
          "text" => %{
            "type" => "plain_text",
            "text" => title,
            "emoji" => true
          }
        },
        %{
          "type" => "image",
          "title" => %{
            "type" => "plain_text",
            "text" => info,
            "emoji" => true
          },
          "image_url" => gif,
          "alt_text" => title
        }
      ]
    })
  end
end
```

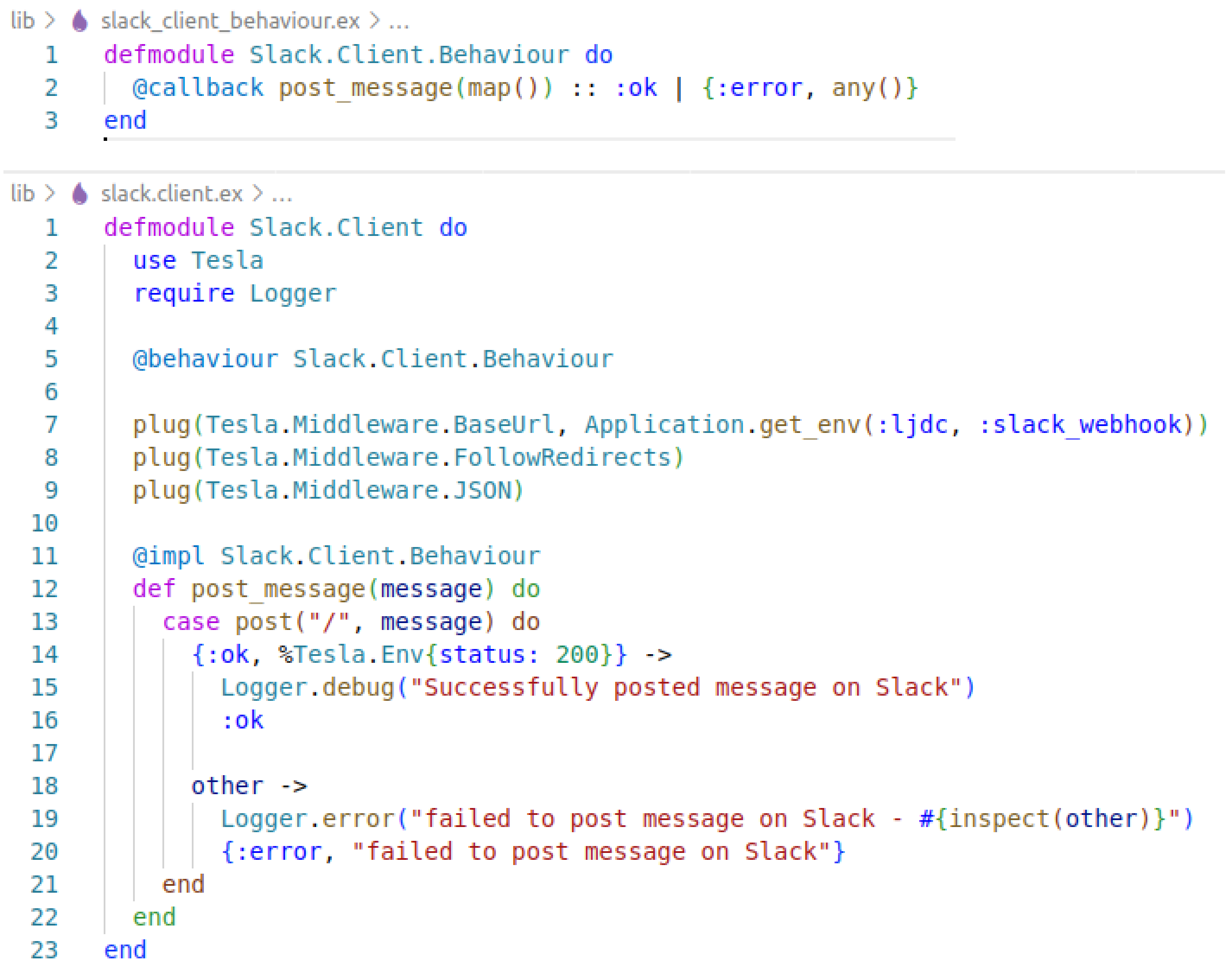
les-joies-du-code

Quand je réalise que je me suis trompé de base pour tous mes delete

Les Joies du Code, commit du 20 Oct 2019 (437 kB)



S'affranchir du couplage : contrat d'interface (behaviour) et module client



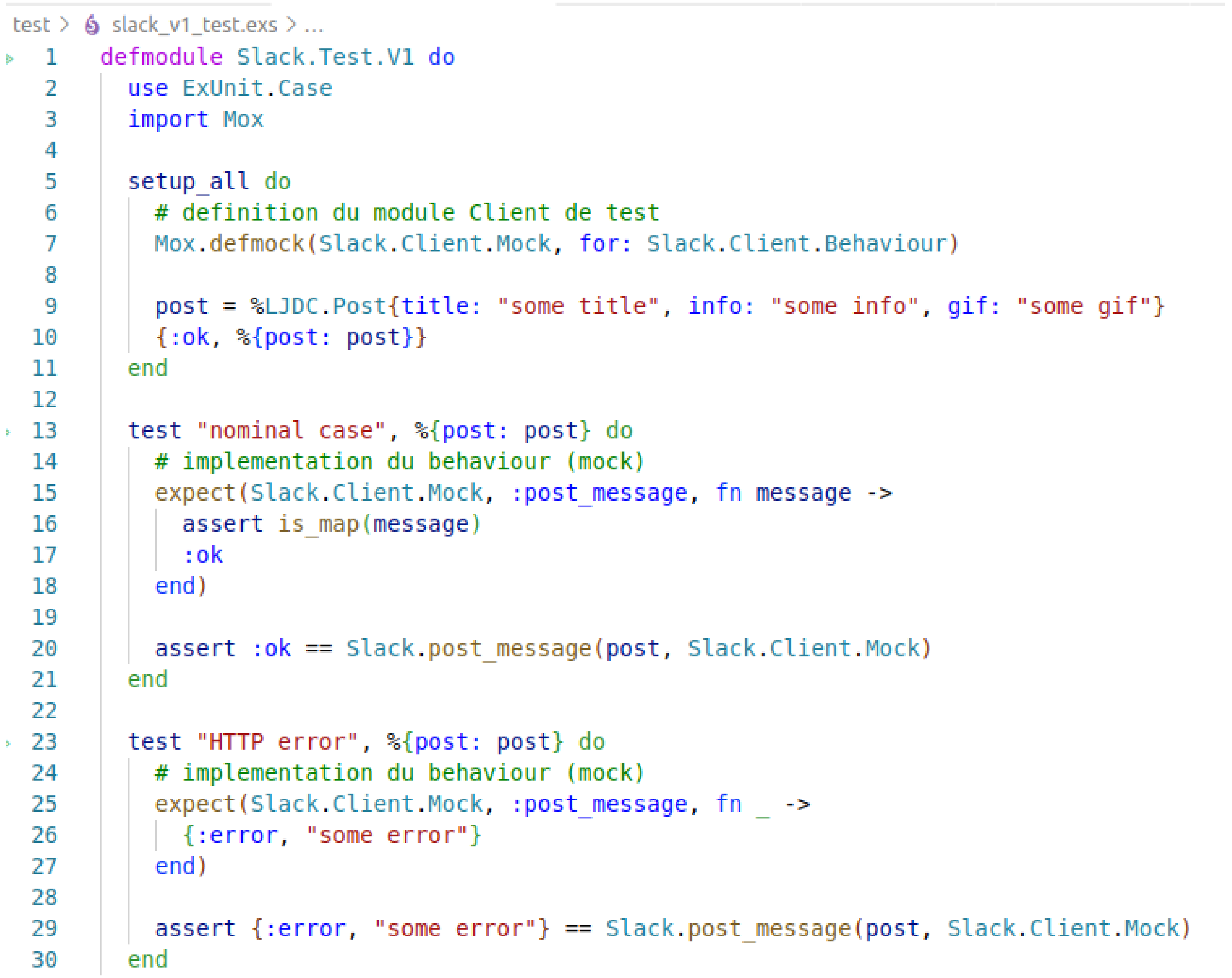
lib > slack_client_behaviour.ex > ...

```elixir
defmodule Slack.Client.Behaviour do
  @callback post_message(map()) :: :ok | {:error, any()}
end
```

lib > slack.client.ex > ...

```elixir
defmodule Slack.Client do
  use Tesla
  require Logger

  @behaviour Slack.Client.Behaviour

  plug(Tesla.Middleware.BaseUrl, Application.get_env(:ljdc, :slack_webhook))
  plug(Tesla.Middleware.FollowRedirects)
  plug(Tesla.Middleware.JSON)

  @impl Slack.Client.Behaviour
  def post_message(message) do
    case post("/", message) do
      {:ok, %Tesla.Env{status: 200}} ->
        Logger.debug("Successfully posted message on Slack")
        :ok

      other ->
        Logger.error("failed to post message on Slack - #{inspect(other)}")
        {:error, "failed to post message on Slack"}
    end
  end
end
```

S'affranchir du couplage : module client de test et lib mox

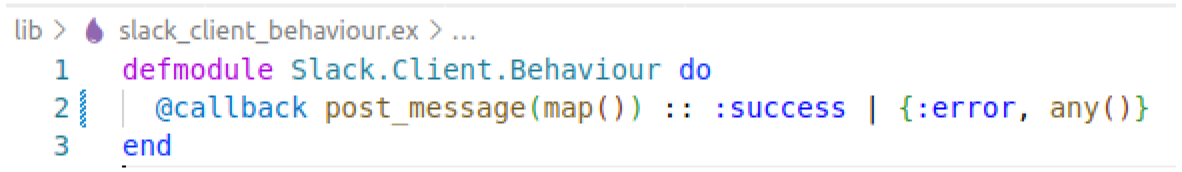

test > slack_v1_test.exs > ...

```elixir
defmodule Slack.Test.V1 do
  use ExUnit.Case
  import Mox

  setup_all do
    # definition du module Client de test
    Mox.defmock(Slack.Client.Mock, for: Slack.Client.Behaviour)

    post = %LJDC.Post{title: "some title", info: "some info", gif: "some gif"}
    {:ok, %{post: post}}
  end

  test "nominal case", %{post: post} do
    # implementation du behaviour (mock)
    expect(Slack.Client.Mock, :post_message, fn message ->
      assert is_map(message)
      :ok
    end)

    assert :ok == Slack.post_message(post, Slack.Client.Mock)
  end

  test "HTTP error", %{post: post} do
    # implementation du behaviour (mock)
    expect(Slack.Client.Mock, :post_message, fn _ ->
      {:error, "some error"}
    end)

    assert {:error, "some error"} == Slack.post_message(post, Slack.Client.Mock)
  end
```

- Pas de référence à Tesla
- Nos modules clients implémentent le behaviour :
  pas besoin d'écrire 1 test pour chaque code d'erreur HTTP : 1 test qui retourne {:error, xxx} suffit !

OK mais...

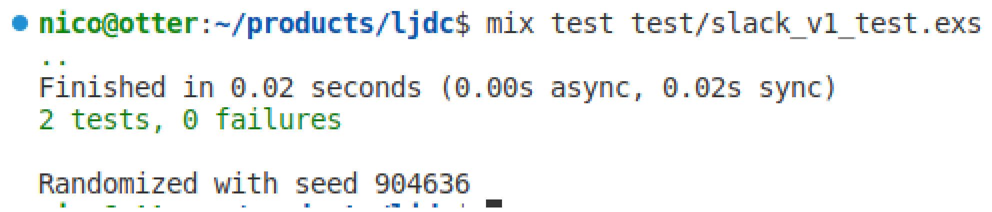
Supposons qu'on change le behaviour (on retourne :success au lieu de :ok) et qu'on modifie Slack.Client en conséquence.



lib > slack_client_behaviour.ex > ...

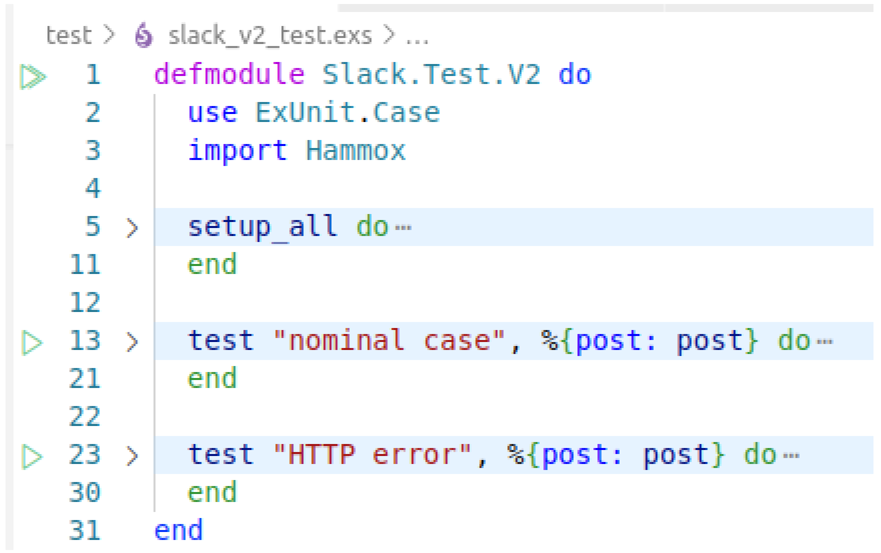
```elixir
defmodule Slack.Client.Behaviour do
  @callback post_message(map()) :: :success | {:error, any()}
end
```

Sans changer le test, celui-ci passe toujours !



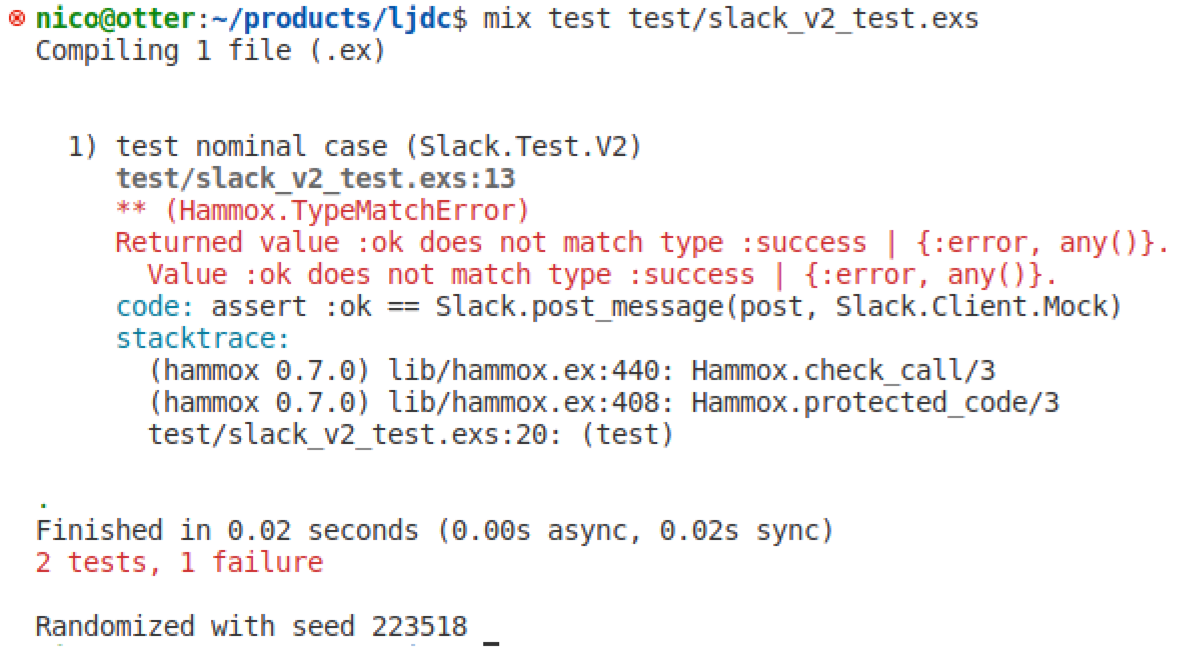
```
● nico@otter:~/products/ljdc$ mix test test/slack_v1_test.exs
  ..
  Finished in 0.02 seconds (0.00s async, 0.02s sync)
  2 tests, 0 failures

  Randomized with seed 904636
```

Utilisation de Hammox au lieu de Mox pour vérifier les typespec du behaviour



test > slack_v2_test.exs > ...

```elixir
defmodule Slack.Test.V2 do
  use ExUnit.Case
  import Hammox

  setup_all do…
  end

  test "nominal case", %{post: post} do…
  end

  test "HTTP error", %{post: post} do…
  end
end
```



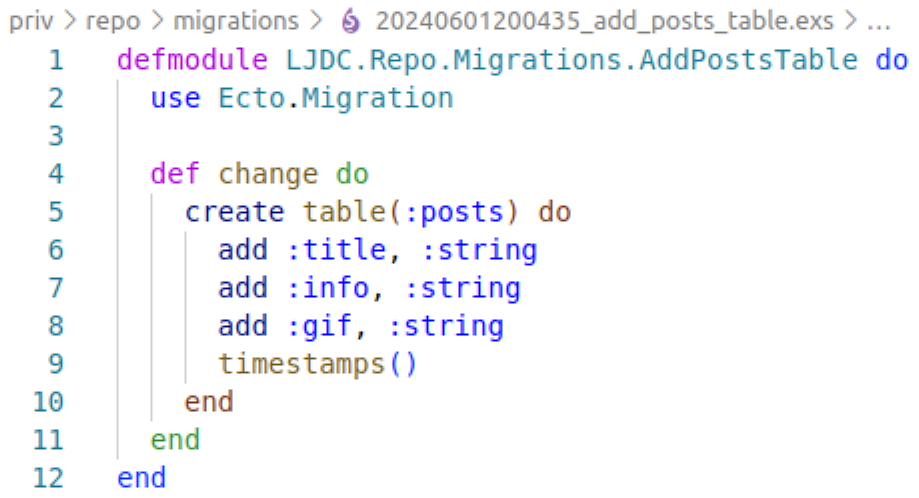
```
nico@otter:~/products/ljdc$ mix test test/slack_v2_test.exs
Compiling 1 file (.ex)


  1) test nominal case (Slack.Test.V2)
     test/slack_v2_test.exs:13
     ** (Hammox.TypeMatchError)
     Returned value :ok does not match type :success | {:error, any()}.
       Value :ok does not match type :success | {:error, any()}.
     code: assert :ok == Slack.post_message(post, Slack.Client.Mock)
     stacktrace:
       (hammox 0.7.0) lib/hammox.ex:440: Hammox.check_call/3
       (hammox 0.7.0) lib/hammox.ex:408: Hammox.protected_code/3
       test/slack_v2_test.exs:20: (test)

.
Finished in 0.02 seconds (0.00s async, 0.02s sync)
2 tests, 1 failure

Randomized with seed 223518 _
```

On détecte que notre test doit être modifié   :)

Bonus : tester les rollbacks ecto

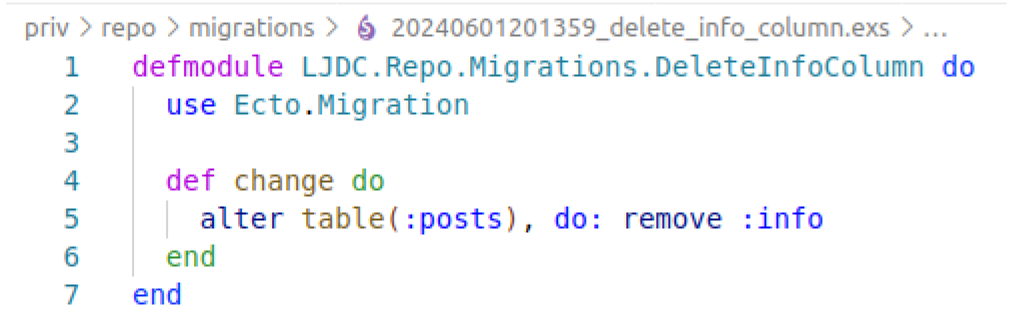
Par défaut les scripts de migration ecto contiennent une callback "change/0".

Lors d'un rollback, les opérations de rollback à effectuer sont déduites de cette callback.



priv > repo > migrations > 20240601200435_add_posts_table.exs > ...

```elixir
defmodule LJDC.Repo.Migrations.AddPostsTable do
  use Ecto.Migration

  def change do
    create table(:posts) do
      add :title, :string
      add :info, :string
      add :gif, :string
      timestamps()
    end
  end
end
```

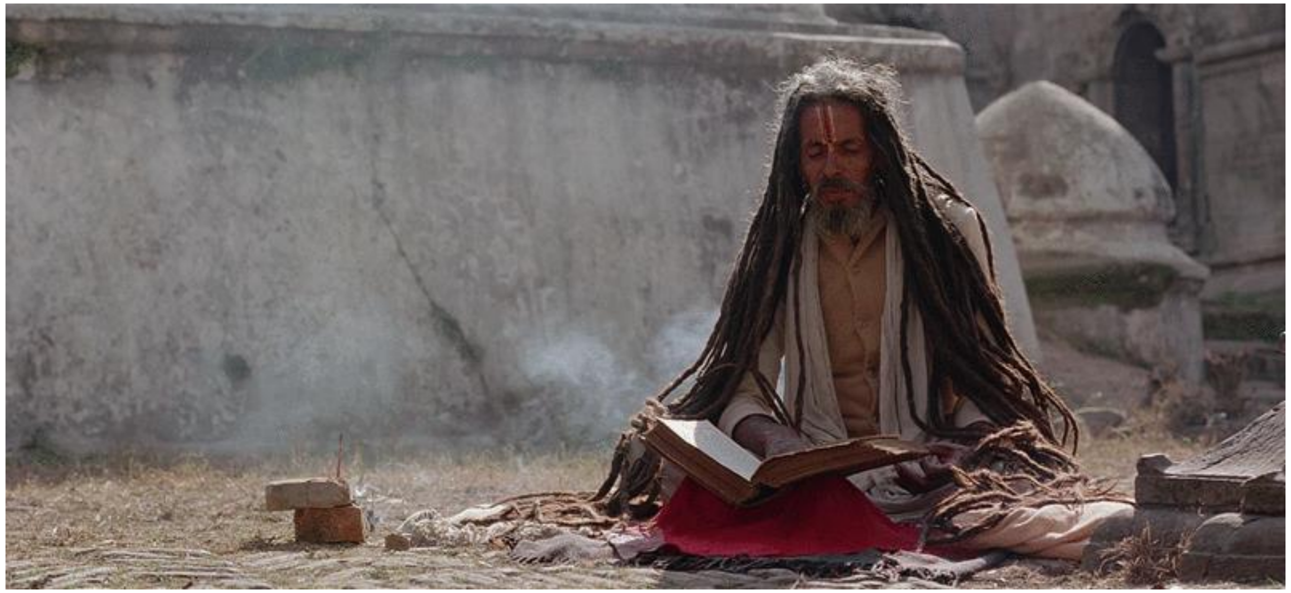


priv > repo > migrations > 20240601201359_delete_info_column.exs > ...

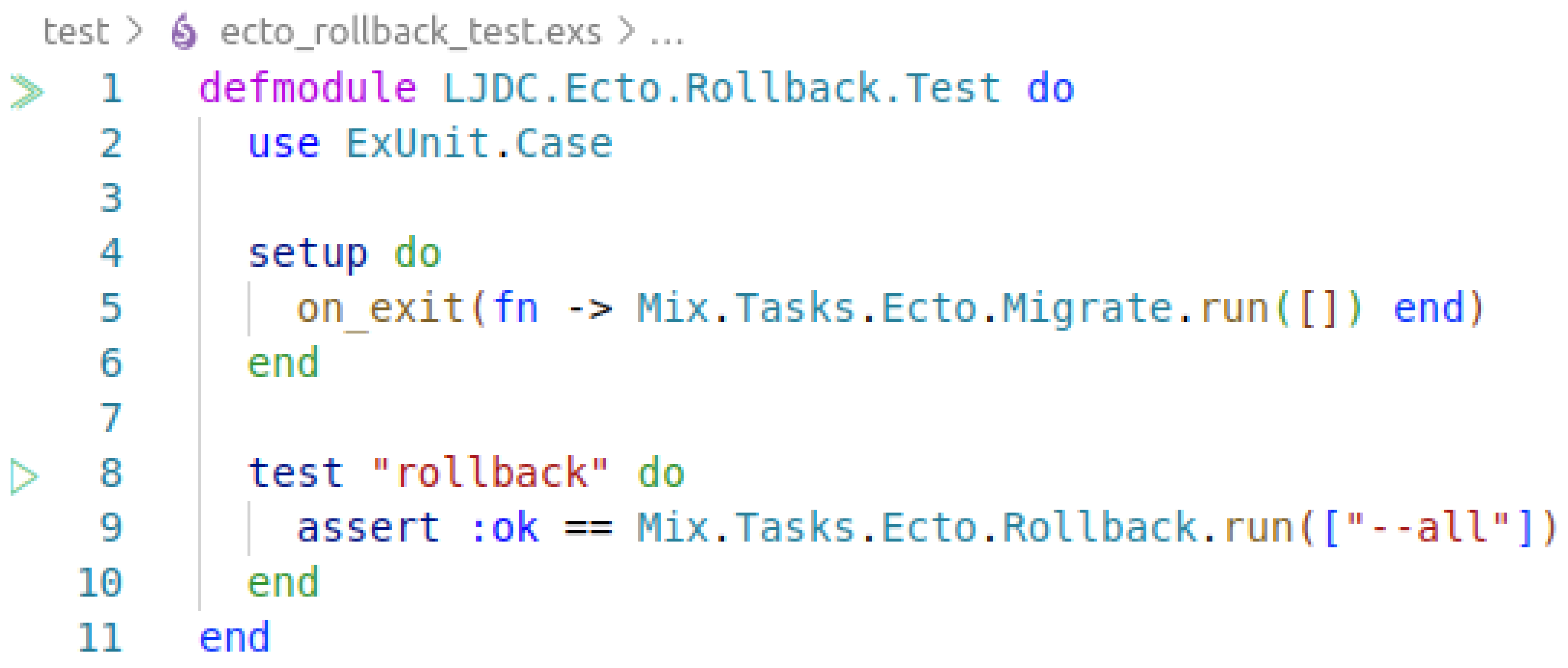
```elixir
defmodule LJDC.Repo.Migrations.DeleteInfoColumn do
  use Ecto.Migration

  def change do
    alter table(:posts), do: remove :info
  end
end
```

Avant de lancer les scripts de migration de la base de données

Les joies du code, commit du 11 Mar 2013

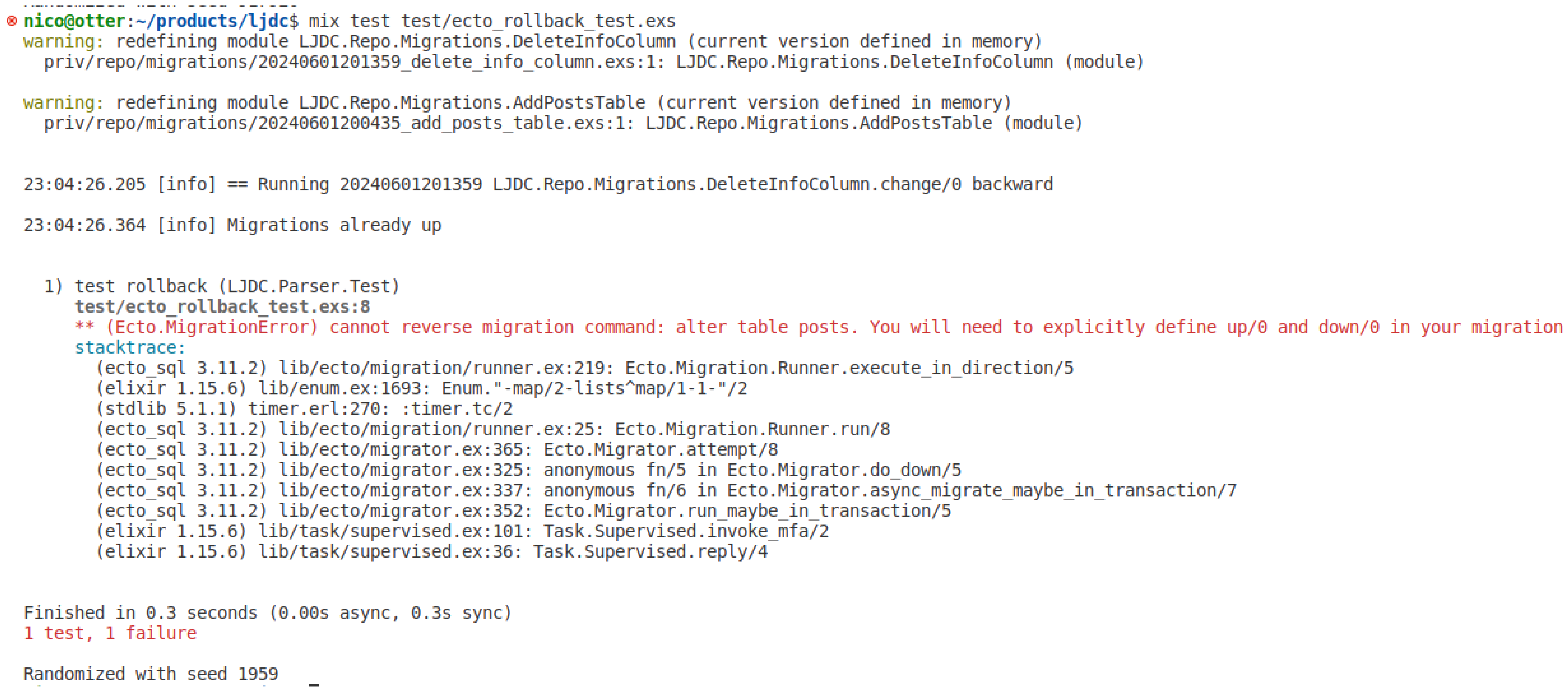


Bonus : tester les rollbacks ecto


test > ecto_rollback_test.exs > ...

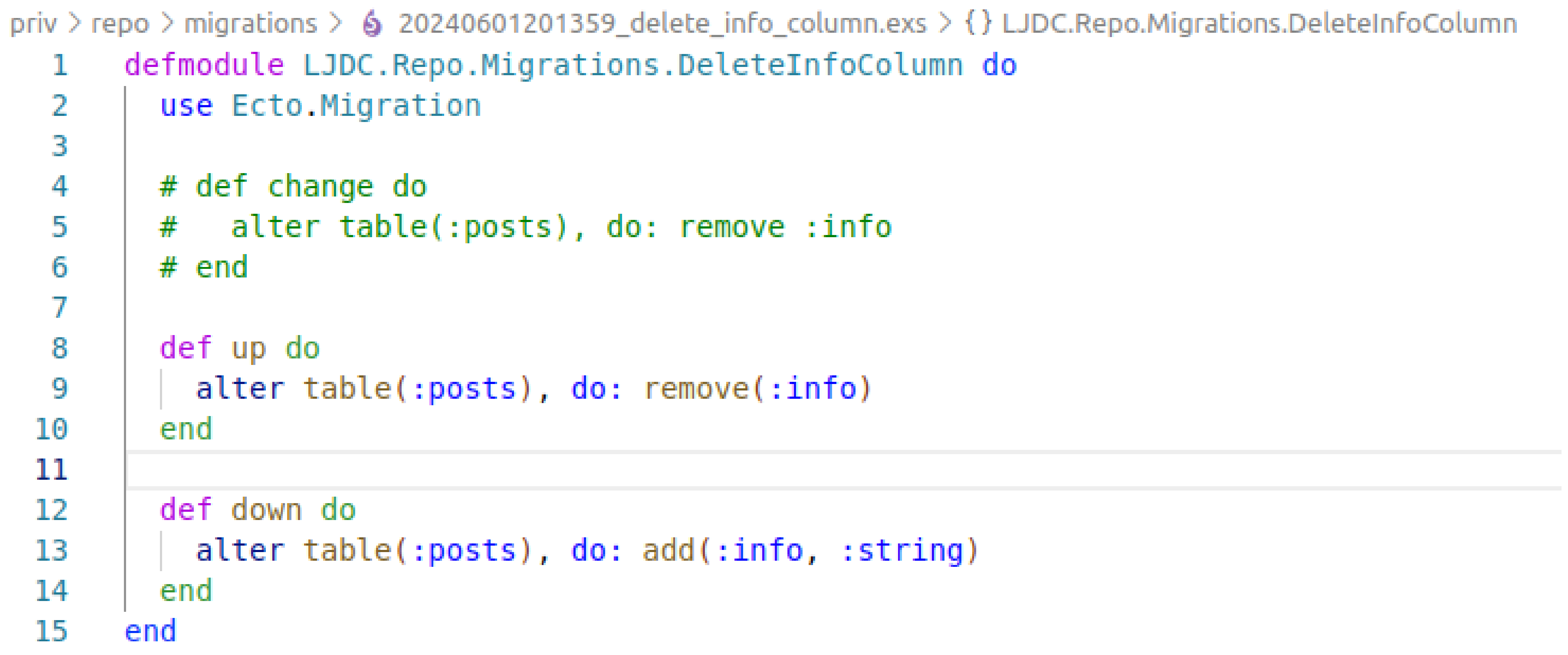
```elixir
defmodule LJDC.Ecto.Rollback.Test do
  use ExUnit.Case

  setup do
    on_exit(fn -> Mix.Tasks.Ecto.Migrate.run([]) end)
  end

  test "rollback" do
    assert :ok == Mix.Tasks.Ecto.Rollback.run(["--all"])
  end
end
```

Bonus : tester les rollbacks ecto



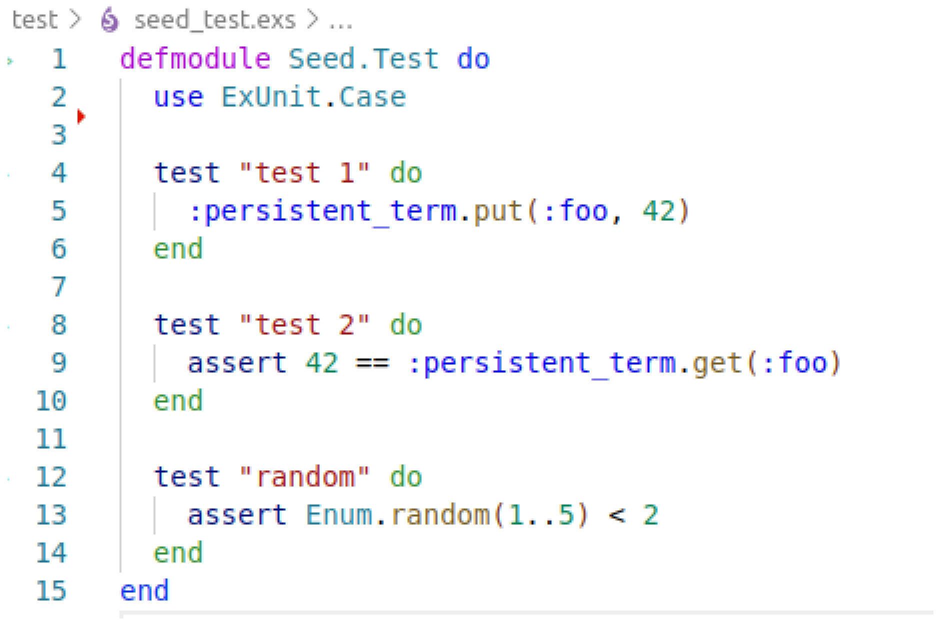
```
nico@otter:~/products/ljdc$ mix test test/ecto_rollback_test.exs
warning: redefining module LJDC.Repo.Migrations.DeleteInfoColumn (current version defined in memory)
  priv/repo/migrations/20240601201359_delete_info_column.exs:1: LJDC.Repo.Migrations.DeleteInfoColumn (module)

warning: redefining module LJDC.Repo.Migrations.AddPostsTable (current version defined in memory)
  priv/repo/migrations/20240601200435_add_posts_table.exs:1: LJDC.Repo.Migrations.AddPostsTable (module)


23:04:26.205 [info] == Running 20240601201359 LJDC.Repo.Migrations.DeleteInfoColumn.change/0 backward

23:04:26.364 [info] Migrations already up


  1) test rollback (LJDC.Parser.Test)
     test/ecto_rollback_test.exs:8
     ** (Ecto.MigrationError) cannot reverse migration command: alter table posts. You will need to explicitly define up/0 and down/0 in your migration
     stacktrace:
       (ecto_sql 3.11.2) lib/ecto/migration/runner.ex:219: Ecto.Migration.Runner.execute_in_direction/5
       (elixir 1.15.6) lib/enum.ex:1693: Enum."-map/2-lists^map/1-1-"/2
       (stdlib 5.1.1) timer.erl:270: :timer.tc/2
       (ecto_sql 3.11.2) lib/ecto/migration/runner.ex:25: Ecto.Migration.Runner.run/8
       (ecto_sql 3.11.2) lib/ecto/migrator.ex:365: Ecto.Migrator.attempt/8
       (ecto_sql 3.11.2) lib/ecto/migrator.ex:325: anonymous fn/5 in Ecto.Migrator.do_down/5
       (ecto_sql 3.11.2) lib/ecto/migrator.ex:337: anonymous fn/6 in Ecto.Migrator.async_migrate_maybe_in_transaction/7
       (ecto_sql 3.11.2) lib/ecto/migrator.ex:352: Ecto.Migrator.run_maybe_in_transaction/5
       (elixir 1.15.6) lib/task/supervised.ex:101: Task.Supervised.invoke_mfa/2
       (elixir 1.15.6) lib/task/supervised.ex:36: Task.Supervised.reply/4


Finished in 0.3 seconds (0.00s async, 0.3s sync)
1 test, 1 failure

Randomized with seed 1959
```

Bonus : tester les rollbacks ecto


priv > repo > migrations > 20240601201359_delete_info_column.exs > {} LJDC.Repo.Migrations.DeleteInfoColumn

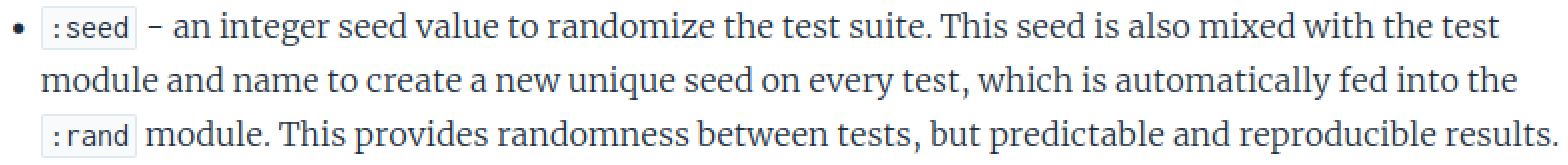
```elixir
defmodule LJDC.Repo.Migrations.DeleteInfoColumn do
  use Ecto.Migration

  # def change do
  #   alter table(:posts), do: remove :info
  # end

  def up do
    alter table(:posts), do: remove(:info)
  end

  def down do
    alter table(:posts), do: add(:info, :string)
  end
end
```

Bonus : investiguer les tests random ou inter-dépendants avec mix test –seed xxx

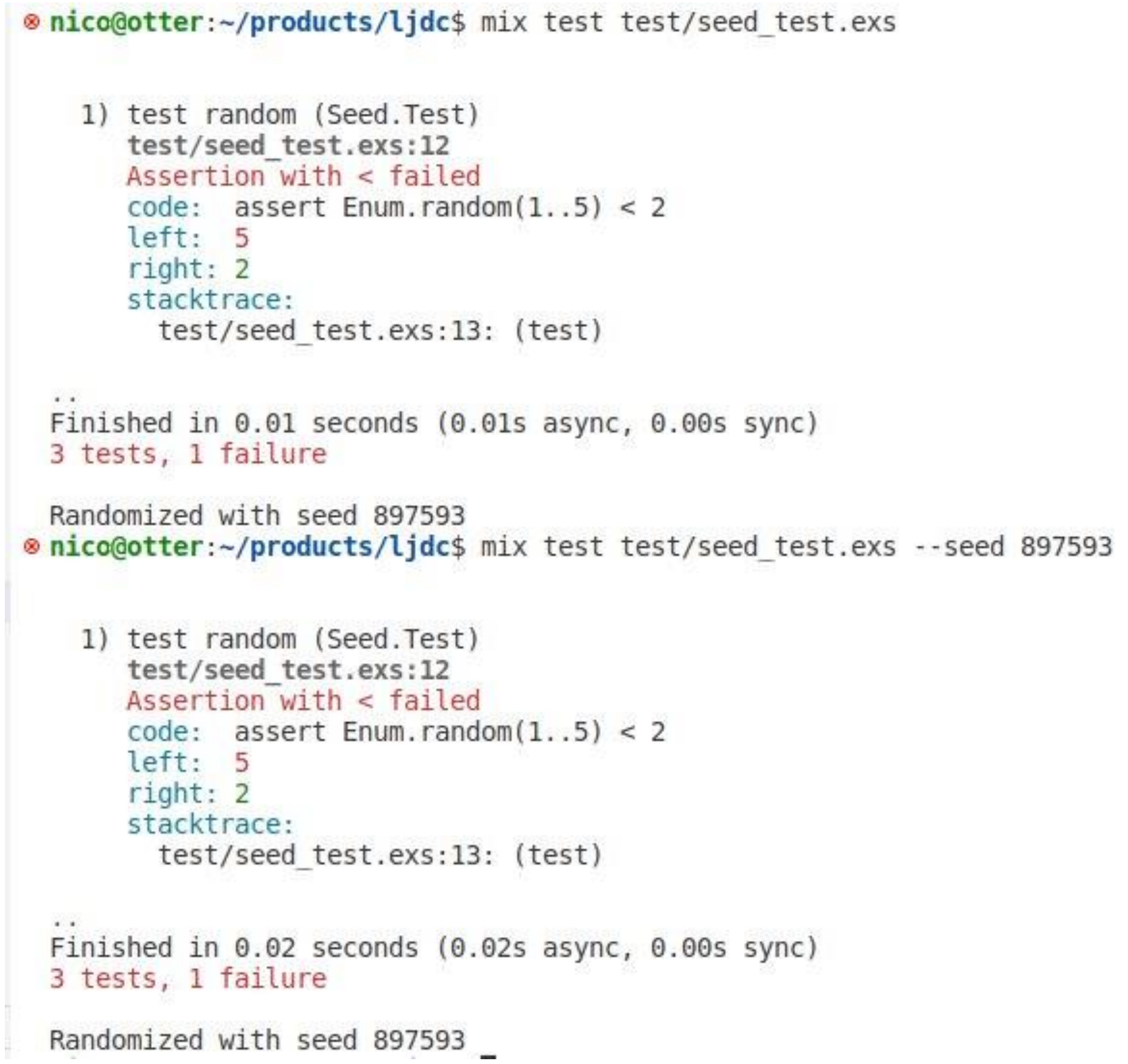

test > seed_test.exs > ...

```elixir
defmodule Seed.Test do
  use ExUnit.Case

  test "test 1" do
    :persistent_term.put(:foo, 42)
  end

  test "test 2" do
    assert 42 == :persistent_term.get(:foo)
  end

  test "random" do
    assert Enum.random(1..5) < 2
  end
end
```

- "test 2" échoue s'il est joué avant "test 1"
- "test random" échoue 4 fois sur 5


- `:seed` - an integer seed value to randomize the test suite. This seed is also mixed with the test module and name to create a new unique seed on every test, which is automatically fed into the `:rand` module. This provides randomness between tests, but predictable and reproducible results.

Bonus : investiguer les tests random ou inter-dépendants avec mix test –seed xxx



```
⊗ nico@otter:~/products/ljdc$ mix test test/seed_test.exs


  1) test random (Seed.Test)
     test/seed_test.exs:12
     Assertion with < failed
     code:  assert Enum.random(1..5) < 2
     left:  5
     right: 2
     stacktrace:
       test/seed_test.exs:13: (test)

..
Finished in 0.01 seconds (0.01s async, 0.00s sync)
3 tests, 1 failure

Randomized with seed 897593
⊗ nico@otter:~/products/ljdc$ mix test test/seed_test.exs --seed 897593


  1) test random (Seed.Test)
     test/seed_test.exs:12
     Assertion with < failed
     code:  assert Enum.random(1..5) < 2
     left:  5
     right: 2
     stacktrace:
       test/seed_test.exs:13: (test)

..
Finished in 0.02 seconds (0.02s async, 0.00s sync)
3 tests, 1 failure

Randomized with seed 897593 _
```

En conclusion, à retenir :

- Organiser son code pour pouvoir tester les cas nominaux et les cas d'erreurs facilement
- Mock d'une fonction interne : librarie Mock
- Mock d'un appel d'API : behaviour + librarie Hammox
- Tester les rollbacks ecto
- Savoir utiliser le seed ExUnit