Airport Weather Queries

Juan Pablo Yamamoto

GitHub

Situation

We've been tasked to implement a service to gather information on the weather at several airports given the plane tickets of the passengers.

Features

- Parsing of non-standardized data
- Parallel queries
- Cache results

Solution

Solution

elixir

Highlights

Structs

```elixir
defmodule WebService.Airport do
  @type t() :: %__MODULE__{
    name: String.t() | nil,
    lat: number() | nil,
    lon: number() | nil,
    iata: String.t() | nil
  }

  defstruct [:name, :lon, :lat, :iata]
end
```

Structs

```elixir
defmodule WebService.Weather do
  @type t() :: %__MODULE__{
    temp: number(),
    temp_min: number(),
    temp_max: number(),
    humidity: number(),
    lat: number(),
    lon: number(),
    name: String.t()
  }

  @enforce_keys [:temp, :temp_min, :temp_max, :humidity, :lat, :lon, :name]
  defstruct [:temp, :temp_min, :temp_max, :humidity, :lat, :lon, :name]
end
```

Structs

```elixir
defmodule WebService.Ticket do
  @type t() :: %__MODULE__{
    origin: Airport.t(),
    destination: Airport.t()
  }

  @enforce_keys [:origin, :destination]
  defstruct [:origin, :destination]
end
```

Environment Variables

```
# config/runtime.exs
import Config
import Dotenvy

source!([".env", System.get_env()])

config :webservice,
  openweather_appid: env!("OPENWEATHER_APPID", :string!)
```

```
# lib/utils.ex
def get_appid(), do: Application.fetch_env!(:webservice, :openweather_appid)
def get_timeframe(), do: Application.fetch_env!(:webservice, :timeframe)
def get_max_requests(), do: Application.fetch_env!(:webservice, :max_requests)
```

Meta-Programming

(Not really the best decision, but something to flex nonetheless)

```
@external_resource
|> File.stream!()
|> CSV.decode(headers: true)
|> ...
|> Enum.each(fn %{"iata_code" => iata, "name" => name, "lat" => lat, "lon" => lon} ->
  def airport_from_iata(unquote(iata)) do
    %__MODULE__{name: unquote(name), iata: unquote(iata), lat: unquote(lat), lon: unquote(lon)}
  end
end)

def airport_from_iata(_), do: nil
```

Mock Testing

```
test "failed request: 400" do
  with_mock HTTPoison,
    [get: fn(_) ->
        {:ok, %HTTPoison.Response{status_code: 400, body: "Not found"}} end
    ] do
    assert WebService.Data.API.by_name("Invalid") == {:error, :not_found}
  end
end
```

Mock Testing

```
test "prints info correctly" do
  with_mocks([
    {Stream, [], [
      run: fn x -> Enum.to_list(x) |> Enum.join("\n") end,
      map: &Enum.map/2
      ]},
    {IO, [], [puts: fn(str) -> str end]},
    {WebService.Data, [], [
      fetch_city: fn (%Airport{name: name, lon: lon, lat: lat} = airport) ->
        {:ok, %Weather{@fake_weather | name: name, lon: lon, lat: lat}, airport} end
      ]}
    ]) do
    assert WebService.Ticket.process([{@airport1, @airport2}]) == expected_string
  end
end
```

What did I learn?

- A couple libraries:
  - Mock
  - Dotenvy
  - HTTPoison

What did I learn?

- Embrace modularity
  - Low coupling
  - High cohesion
  - Keep user interaction/effects only on the frontier modules

What did I learn?

- **Types!**
  - Use `struct`s to model your business components
  - Clarity
  - Get pattern matching for free
  - Use `dialyxir`:

```
> mix dialyzer
Total errors: 0, Skipped: 0, Unnecessary Skips: 0
done in 0m5.26s
done (passed successfully)
```

Future improvements

- A better architecture
- Consider using Erlang's `ets`
- A better rate-limiting mechanism for API queries
- Rethink the proper use of asynchrounous tasks

Airport Weather Queries

Juan Pablo Yamamoto

GitHub