Structs for order

Andrey Chernykh

Me: quick facts

C# 5 years |> Ruby 4 years |> Elixir since 2016

Number of services (not all of them are micro) in production.

 https://github.com/madeinussr

- exop
- express

 https://medium.com/@andreichernykh

Currently: Elixir developer at Coingaming Group 

“Container” data types

List, Tuple, Keywords list, Map, Struct

“Structs are extensions built on top of maps that provide compile-time checks and default values”

“Container” data types

“Structs are data structures built on top of maps that conform predefined public contract”

Contract

Contract is about expectations

No contract — no confidence

Maps are for “here and now”

Structs are for reading code months (days?) later

Structs are for long-term maintenance

Contract

```elixir
# 1
%{} = %{a: 1, b: 2, c: 3}
%{a: 1} = %{a: 1, b: 2, c: 3}

# 2
defmodule Triangle do
  defstruct [:a, b: 2, c: 3]
end

%{} = %Triangle{}
%{a: 1, b: 2} = %Triangle{a: 1}
```

```elixir
# what else the map contains?
def do_smth(map), do: :smth
def do_smth(%{a: _, b: _, c: _} = map), do: :smth

# I know Triangle contains :a, :b, :c keys in any case
def do_smth(%Triangle{} = struct), do: :smth
```

Responsibility

A Struct module a self-consistent unit

Takes care of its own data

Self-documented: attributes/schema, doctests,
types, typespecs

Responsibility

```elixir
defmodule Triangle do
  defstruct [:a, :b, :c, :area, :perimeter]

  @doc """
  Constructs a triangle structure. Expects a map as an argument.
  Calculates area & perimeter if they aren't provided.
  """
  @spec new(map()) :: %__MODULE__{}
  def new(params) when is_map(params) do
    params = params |> coerce() |> validate() |> perimeter() |> area()

    case params do
      {:ok, params} -> struct(__MODULE__, params)
      error -> error
    end
  end
  # ...
end
```

```elixir
triangle = Triangle.new(%{a: 3, b: 4, c: 5})
#> %Triangle{a: 3, b: 4, c: 5, area: 6, perimeter: 12}
triangle.area
#> 6

triangle = Triangle.new(%{a: 3, b: -4, c: 5})
#> {:error, :validation_failed}
```

Ecto's embedded schema

Known schema declaration

Arguments casting (schema mapping)

Types check + custom types

Built-in and custom validations

Relations + complex nested structs

And all this you get for out-of-the-box, tested and proved in production*

* but with some cost

```elixir
defmodule Triangle do
  use Ecto.Schema
  import Ecto.Changeset
  @primary_key false

  embedded_schema do
    field(:a, :integer)
    field(:b, :integer)
    # ...
  end

  def new(params) do
    %__MODULE__{}
    |> cast(params, [:a, :b, :c, :area, :perimeter])
    |> validate_required([:a, :b, :c])
    |> validate_number(:a, greater_than_or_equal_to: 1)
    |> # ... more validations here
    |> try_to_apply()
  end

  defp try_to_apply(%{valid?: true} = changeset), do: apply_changes(changeset)
  defp try_to_apply(%{valid?: false, errors: errors}), do: {:error, errors}
end
```

More practical example

```elixir
def create(%Conn{} = conn, %{"user" => customer_id,
                             "timestamp" => date_time,
                             "campaign" => discount,
                             "currency" => currency,
                             "quantity" => amount,
                             "taxes" => tax,
                             "taxCode" => tax_code} = params)
    when is_integer(customer_id) and is_binary(date_time) and
         is_float(discount) and is_binary(currency) and
         is_float(amount) and is_float(tax) and is_binary(tax_code)
do
  # further validation, errors handling and creation itself
end
```

```elixir
def update(%Conn{} = conn, %{"campaign" => discount,
                             "currency" => currency,
                             "quantity" => amount,
                             "taxes" => tax,
                             "taxCode" => tax_code} = params)
    when is_float(discount) and is_binary(currency) and
         is_float(amount) and is_float(tax) and is_binary(tax_code)
do
  # further validation, errors handling and update itself
end
```

Later…

How it could be

0
```elixir
defmodule SuperApp.Invoice.Create.Request do
  # defstruct or embedded_schema
end
defmodule SuperApp.Invoice.Create.Response do
  # defstruct or embedded_schema
end
```

1
```elixir
scope "/invoice" do
  alias SuperApp.Invoice.{Create, Update}

  post("/create", Invoice, :create, assigns: %{
    req_module: Create.Request,
    resp_module: Create.Response
  })
  post("/update", Invoice, :update, assigns: %{
    req_module: Update.Request,
    resp_module: Update.Response
  })
end
```

2
```elixir
defmodule SuperApp.Controllers.RequestBuilder do
  # ...
  def call(%Conn{params: params, assigns: %{req_module: req_module}} = conn, _opts) do
    case req_module.new(params) do
      %{} = request_struct -> assign(conn, :request_struct, request_struct)
      {:error, error} -> {:error, error}
      _ -> {:error, :unhandled_error}
    end
  end
  # ...
end
```

How it could be

3
```elixir
action_fallback(FallbackController)

plug(RequestBuilder)

def create(%Conn{assigns: %{request_struct: req_struct, resp_module: resp_mod}} = conn, params) do
  with {:ok, action_result} ← CreateOperation.run(req_struct) do
    render(conn, "create.json", action_result: action_result, resp_mod: resp_mod)
  end
end
```

4
```elixir
def render("create.json", %{action_result: action_result, resp_mod: resp_mod}) do
  action_result ▷ resp_mod.new() ▷ resp_mod.to_map()
end
```

Before you ask

No, not only for controllers

Do not delegate everything to a struct module

Ecto brings some overhead

Protobuf can be an alternative, but…

Ecto’s schema vs plain struct

Requirements:
  coerce (atomify) given params map keys if needed
  check required params presence
  check given params types

```
available memory: "16 GB", cpu_speed: "2.70GHz",
elixir: "1.7.3", erlang: "20.3.2", num_cores: 8

tool: Benchee

Name                   ips        average  deviation         median         99th %
Plain Struct       424.44 K        2.36 µs   ±940.79%        2 µs           7 µs
Ecto Schema        156.11 K        6.41 µs   ±277.71%        6 µs          16 µs

Comparison:
Plain Struct       424.44 K
Ecto Schema        156.11 K - 2.72x slower
```

https://github.com/madeinussr/struct_vs_ecto

Thank |> you