Transitioning from Ruby/Rails to Elixir/Phoenix

...

What I learned converting a Rails app to a Phoenix app.

Object-oriented vs. Functional

Classes vs. Modules

```ruby
# ruby
class User < Struct.new(:first, :last)
  def self.full_name(user)
    "#{user.first} #{user.last}"
  end
end

user = User.new("Bob", "Smith")
User.full_name(user)
# => "Bob Smith"
```

```elixir
# elixir
defmodule User do
  defstruct [:first, :last]

  def full_name(user) do
    "#{user.first} #{user.last}"
  end
end

user = %User{first: "Bob", last: "Smith"}
User.full_name(user)
# => "Bob Smith"
```

Embrace Pattern Matching

```elixir
# in function definitions
def changeset(:registration, params), do: # ...
def changeset(:password_update, params), do: # ...


# when handling flow of logic
case MyApp.do_something() do
  {:ok, value}      -> # ...
  {:error, errors} -> #...
end
```

rake VS. mix

Familiar commands

```
mix phoenix.new my_app
mix phoenix.server
mix ecto.migrate
mix phoenix.gen.html User users email:string
mix test
MIX_ENV=test mix ecto.migrate
```

`irb` **VS.** `iex`

Elixir's Interactive Shell

- `iex` is Elixir's interactive shell
- Run it within your Elixir project with `iex -S mix`
- Hitting Ctrl-C twice to exit takes a little getting used to
- Use `recompile()` when you've made changes to files and don't want to lose your current IEx session.
- Create an .iex.exs file in your project root to alias your project's modules for faster typing
- `User |> Repo.all |> List.first`

Minitest vs. ExUnit

ExUnit

```
setup do
  user = insert_user()
  conn = build_conn() |> with_current_user(user)
  {:ok, conn: conn, current_user: user}
end

describe "POST /comment" do
  test "when params are invalid", %{conn: conn} do
    # ...

    assert {:ok, comment} = result
  end
end
# Run with `mix test` or `mix test test/controller/...`
```

ExUnit - Test Failure Output

```
Comparison (using ==) failed in:
code: some_fun() == 10
lhs:  13
rhs:  10
```

ActiveRecord Models vs. Ecto Schemas

Ecto - Schema

```elixir
defmodule MyApp.User do
  use Ecto.Schema

  schema "users" do
    field :username, :string
    field :encrypted_password, :string
    field :email, :string
    field :confirmed, :boolean, default: false
    field :password, :string, virtual: true
    field :password_confirmation, :string, virtual: true

    timestamps
  end
end
```

Ecto - Changeset

```
defmodule MyApp.User do
  import Ecto.Changeset

  schema "users" do
    ...
  end

  @required_fields ~w(username encrypted_password email)
  @optional_fields ~w()

  def changeset(user, params \\ %{}) do
    user
    |> cast(params, @required_fields, @optional_fields)
    |> unique_constraint(:username)
  end
end
```

Ecto - Repo

```elixir
defmodule MyApp.Repo do
  use Ecto.Repo,
    otp_app: :example_app
end

Repo.insert(changeset)
Repo.all(User)
Repo.aggregate(User, :count, :id)

query = from u in "users",
          where: u.email == "user@example.com",
          select: u.name
Repo.one(query)
```

Controllers vs. . . . Controllers

Controllers

```
defmodule MyApp.UserController do
  use MyApp.Web, :controller

  alias MyApp.User

  def index(conn, _params) do
    users = Repo.all(User)
    render(conn, "index.html", users: users)
  end
end

# in web/router.ex
resources "/users", UserController
```

Views vs. Views + Templates

Views + Templates

```
# in web/views/user_view.ex
defmodule MyApp.UserView do
  use MyApp.Web, :view
end

# in web/templates/users/index.html.eex
<%= for user <- @users do %>
  <tr>
    <td><%= user.email %></td>
  </tr>
<% end %>
```

Rack vs. Plug

Plug

```
defmodule MyApp.Plugs.Auth do
  def init([]), do: false
  def call(conn, _opts), do: conn
end

defmodule MyApp.Router do
  # ...

  pipeline :auth do
    plug MyApp.Plugs.Auth
  end

  scope "/", MyApp do
    pipe_through [:auth]

    resources "/users", UserController
  end
end
```

Plug

```
defmodule MyApp.DashboardController do
  plug :authenticate

  defp authenticate(conn), do: #...
  defp authenticate(conn, _), do #...
end
```

RubyGems vs. Hex Packages

Hex Packages

```
# mix.exs
defp deps do
  [{:phoenix, "~> 1.2.1"},
   {:phoenix_pubsub, "~> 1.0"},
   {:phoenix_ecto, "~> 3.0"},
   {:postgrex, ">= 0.0.0"},
   {:phoenix_html, "~> 2.6"},
   # ...
  ]
end

# mix deps.get
```

Debugging

IEx.pry

- `IO.puts` and `IO.inspect`
- You *must* run `iex -S mix phoenix.server`
- In the file that you want to debug, you *must* `require IEx` *before* the `defmodule` line then you can add your `IEx.pry` anywhere in that file
- Refresh your browser, works just like pry in ruby
- Call `respawn()` to continue/finish the request

Final Tips

Final Tips

- Use `iex -S mix` generously.

```
iex(1)> Form.insert_changeset(%Form{}, %{name: "Contact Us"})
Ecto.Changeset<action: nil,
changes: %{name: "Contact Us"},
errors: [user_id: {"can't be blank", [validation: :required]}],
data: #MyApp.Form<>,
valid?: false>
```

- Don’t be afraid to make a mess.

Thank You

- www.grok-interactive.com
- @laurenfackler
- www.elformo.com