From Ruby to Elixir:

Developing Web Applications

Mario Alberto Chávez
@mario_chavez

Funcional

Moderno

Dinámico

Distribuido

Tolerante a fallas

Basado en Erlang VM

MICHELADA.IO

"A web server is a natural problem for a functional language to solve"

"Programming Phoenix"

MICHELADA.IO

Phoenix Framework

Productivo, Confiable y Rápido

MICHELADA.IO

Implementa MVC

Aunque la parte M cambia radicalmente

MICHELADA.IO

MIX

Herramienta base para compilar y ejecutar tareas

```
mix phoenix.new

mix ecto.migrate

mix deps.get

mix compile
```

MICHELADA.IO

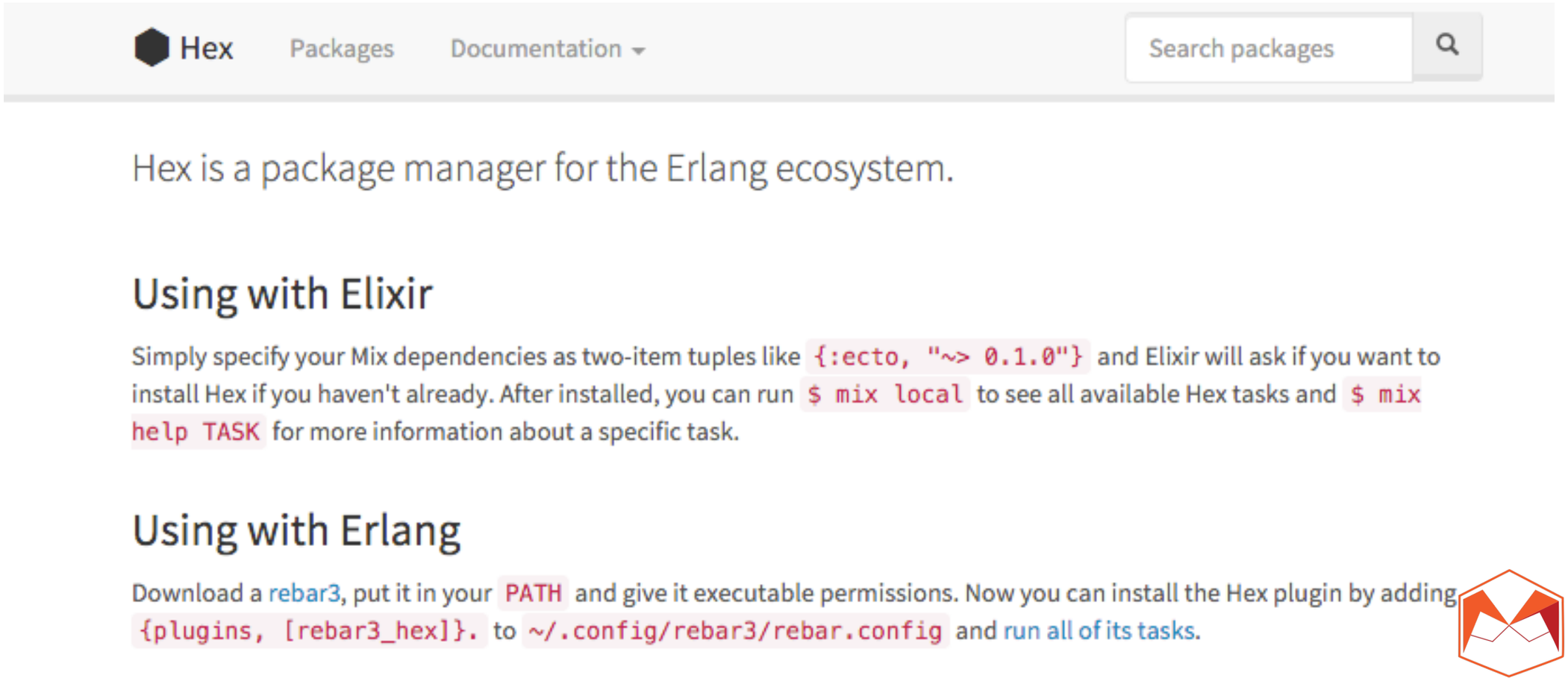
DEPENDENCIAS

Declaradas en `mix.ex` y manejadas con comandos de mix

```
[{:phoenix, "~> 1.0.3"},
     {:phoenix_ecto, "~> 1.1"},
     {:postgrex, ">= 0.0.0"},
     {:phoenix_html, "~> 2.1"},
     {:scrivener, "~> 1.0"},
     {:phoenix_live_reload, "~> 1.0", only: :dev},
     {:cowboy, "~> 1.0"}]
```

DEPENDENCIAS

Directorio en http://hex.pm



MICHELADA.IO

VIDA DE UN REQUEST

```
connection
 |> endpoint
 |> router
 |> controller
 |> common_services
 |> action
 |> process_data
 |> view
 |> template
```

MICHELADA.IO

PLUGS

Cada paso es una función que se conecta a otra, toma un `conn` y regresa una versión modificada de `conn`

```
f(conn)
```

MICHELADA.IO

PLUGS

`conn` es un struct que contiene información del `request` y `response`

```
%Plug.Conn{adapter: {Plug.Adapters.Cowboy.Conn, :...}, assigns: %{},
 before_send: [#Function<1.23820878/1 in Plug.Logger.call/2>,
  #Function<0.109114241/1 in
Phoenix.LiveReloader.before_send_inject_reloader/1>],
 body_params: %{}, cookies: %Plug.Conn.Unfetched{aspect: :cookies},
 halted: false, host: "localhost", method: "GET", owner: #PID<0.509.0>,
 params: %{}, path_info: ["api", "conferences"], peer: {{127, 0, 0, 1},
57547},
 port: 4000,
…}
```

PLUGS

Si un `Plug` es solamente una función, entonces la aplicación Web es un pipeline de plugs.

ESTRUCTURA

```
priv
├── repo
│   └── migrations
└── static
    ├── css
    ├── images
    └── js
web
├── channels
├── controllers
├── models
├── static
│   ├── assets
│   │   └── images
│   ├── css
│   ├── js
│   └── vendor
├── templates
│   ├── layout
│   └── page
└── views
config
lib
└── present
test
├── channels
├── controllers
├── fixtures
├── models
├── support
└── views
```

AMBIENTES

Diferentes configuraciones

```
production
  development
    test
```

MICHELADA.IO

AMBIENTES

Existe una configuración general y otra complementaria por ambiente

`config.exs`

```
config :present, Present.Endpoint,
  url: [host: "localhost"],
  root: Path.dirname(__DIR__),
  render_errors: [accepts: ~w(html json)]
```

`dev.exs`

```
config :present, Present.Endpoint,
  http: [port: 4000],
  debug_errors: true,
  code_reloader: true,
  cache_static_lookup: false
```

MICHELADA.IO

ENDPOINT

Es la frontera entre el Web Server y nuestra aplicación.

```
defmodule Present.Endpoint do
use Phoenix.Endpoint, otp_app: :present

plug Plug.RequestId
plug Plug.Logger

plug Plug.Parsers,
  parsers: [:urlencoded, :multipart, :json],
  pass: ["*/*"],
  json_decoder: Poison

plug Plug.MethodOverride
plug Plug.Head
```

ROUTER

Contiene la tabla de rutas y … más plugs

```
defmodule Present.Router do
  use Present.Web, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :protect_from_forgery
  end

  scope "/", Present do
    pipe_through :browser # Use the default browser stack

    get "/", PageController, :index
    resources "/events", EventController, except: [:index, :show]
  end
```

ROUTER

De las rutas se generan funciones *_path y *_url

```
mix phoenix.routes
    atom_path  GET     /atom.xml          Dash.AtomController :index
sitemap_path  GET     /sitemap.xml       Dash.SitemapController :index
    post_path  GET     /admin             Dash.PostController :index
    post_path  GET     /admin/:id/edit    Dash.PostController :edit
    post_path  GET     /admin/new         Dash.PostController :new
    post_path  GET     /admin/:id         Dash.PostController :show
    post_path  POST    /admin             Dash.PostController :create
```

CONTROLLER

Ejecuta las acciones y genera la respuesta

```
defmodule Dash.PageController do
  use Dash.Web, :controller

  import Ecto.Query, only: [from: 2]

  alias Dash.Post

  plug :scrub_params, "post" when action in [:create]

  def index(conn, _params) do
    posts = Repo.all(Post)
    render(conn, "index.html", posts: posts)
  end
end
```

MICHELADA.IO

MIGRATION

DSL para modificar la base de datos. Versionadas y ordenadas.

```
defmodule Dash.Repo.Migrations.CreatePost do
  use Ecto.Migration

  def change do
    create table(:posts) do
      add :title, :string
      add :body, :text
      add :permalink, :string
      add :tags, {:array, :string}
      timestamps
    end
  end
end
```

MICHELADA.IO

MODEL

El modelo está desconectado de la base de datos

- Schema
- Changeset
- Asociaciones
- Query composition

MICHELADA.IO

MODEL

No hay introspección

```
schema "posts" do
  field :title, :string
  field :body, :string
  field :permalink, :string
  field :tags, {:array, :string}
  field :published, :boolean, default: false
  field :published_at, Ecto.DateTime
  field :tags_text, :string, virtual: true

  belongs_to :user, Dash.User
  timestamps
end

@required_fields ~w(title)
@optional_fields ~w(body permalink tags published published_at user_id)
```

MODEL

Changeset nos ayuda a generar los cambios en el modelo y aplicar validaciones

```
  def changeset(model, params \\ :empty) do
    model
    |> cast(params, @required_fields, @optional_fields)
    |> inflate_tags
    |> update_published
  end

iex> changeset = Post.changeset(%Post{}, post_params)
```

MICHELADA.IO

MODEL

Changeset es autocontenido

```
id = 1
query = from u in User, where: u.id == ^id

user_params = %{nickname: “mario.chavez"}
changeset = User.changeset(user, user_params)

%Ecto.Changeset{action: nil, changes: %{nickname: "mario.chavez"}, constraints: [],
errors: [], filters: %{},
model: %Dash.User{__meta__: #Ecto.Schema.Metadata<:loaded>, bio: "Rubyst",
 id: 1, inserted_at: #Ecto.DateTime<2015-09-17T19:08:21Z>,
 name: "Mario Alberto Chavez", nickname: "mario_chavez",
 posts: #Ecto.Association.NotLoaded<association :posts is not loaded>,
 social: %{"github" => "http://github.com/mariochavez",
   "twitter" => "http://twitter.com/mario_chavez"},
 updated_at: #Ecto.DateTime<2015-09-17T19:08:21Z>}, optional: [:bio, :social],
opts: nil, params: %{"nickname" => "mario.chavez"}, repo: nil,
required: [:name, :nickname],
types: %{bio: :string, id: :id, inserted_at: Ecto.DateTime, name: :string,
  nickname: :string,
  posts: {:assoc,
   %Ecto.Association.Has{cardinality: :many, defaults: [], field: :posts,
    on_cast: :changeset, on_delete: :nothing, on_replace: :raise,
    owner: Dash.User, owner_key: :id, queryable: Dash.Post, related: Dash.Post,
    related_key: :user_id}}, social: :map, updated_at: Ecto.DateTime},
valid?: true, validations: []}
```

MODEL

Podemos generar "fragmentos" de queries para hacer "query composition"

```
defmodule Present.Event do
  def conference_events(query, conference) do
    from e in query,
    where: e.conference_id == ^conference.id
  end
end

iex> query = Event.conference_events(Event, conference)
iex> query2 = from e in query, order_by: [asc: e.name]

iex> Event.conference_events(Event, conference)
     |> order_by([e], [asc: e.name])
```

MICHELADA.IO

MODEL

No hace preload de asociaciones, explícitamente tenemos que llamar `preload`

MICHELADA.IO

MODEL

Ecto entiende de constraints en la base de datos y puede registrar los errores

MICHELADA.IO

REPO

Repo maneja la conexión a la base de datos.

all(query)

insert(changeset)

get(query, params)

update(changeset)

one(query, params)

get_by(query, params)

delete(keyword)

MICHELADA.IO

TEMPLATES

Son plantillas EEx, código HTML y código Elixir

```
  <%= for post <- @posts do %>
    <article id='<%= "post_#{post.id}" %>' class='content post'>
      <h2><a href='<%= page_path(@conn, :show, post.permalink) %>'><%=
post.title %></a></h2>
      <span class='author'><%= "By #{author_name(post.user)}" %></span>
      <span class='time'><%= human_date(post.published_at) %></span>
      <p><%= post.summary %></p>
    </article>
  <% end %>
```

MICHELADA.IO

TEMPLATES

Existen diversos "helpers"

```
<%= form_for @changeset, @action, fn f -> %>

<div class="form-group">
  <%= label f, :title, "Title", class: "control-label" %>
  <%= text_input f, :title, class: "form-control" %>
</div>

<div class="form-group">
  <%= label f, :summary, "Summary", class: "control-label" %>
  <%= textarea f, :summary, class: "form-control" %>
</div>
```

MICHELADA.IO

TEMPLATES

Se compilan como métodos en las vistas.

TEMPLATES

Todo controlador se acompaña de una vista.

MICHELADA.IO

TEMPLATES

Podemos usar “Layouts” o parciales

MICHELADA.IO

VIEWS

Contienen helpers y se encargan de hacer el render de las templates

```
defmodule Dash.PageView do
  use Dash.Web, :view

  def human_date(date) do
    {:ok, date} = Ecto.DateTime.dump(date)
    Chronos.Formatter.strftime(date, "%B %d, %Y")
  end

  def render_author(conn, author) do
    render_existing(Dash.PageView, "author.html",
      %{conn: conn, author: author})
  end
end
```

MICHELADA.IO

CHANNELS

Comunicación en tiempo real

Channel Socket

Channel Route

Channel

PubSub

Message

Transport

Transport Adapter

Client libraries

MICHELADA.IO

NAMING CONVENTION

Simple: Todos los nombres son en singular.

MICHELADA.IO

ASSETS

Brunch y Nodejs procesan los assets

Brunch

Home Docs Guide Plugins Skeletons Examples Compare

Brunch

Will you be coming to brunch?

Brunch...

compiles your scripts, templates, styles, **lints** them,
wraps the scripts and templates in common.js / AMD modules., **concatenates** scripts and styles,
generates source maps for concatenated files, **copies** assets and static files,
shrinks the output by minifying code and optimizing images, **watches** your files for changes,
and **notifies** you about errors via console and system notifications.

Brunch is an ultra-fast HTML5 build tool

Installation is one-line, once you have node.js. In your console, run:

```
npm install -g brunch
```

Star 4,643

Follow @brunch 1,618 followers

Tweet 1,142

MICHELADA.IO

Why Brunch instead of

TESTING

ExUnit es un framework básico de unit testing

```
defmodule Present.EventTest do
  use Present.ModelCase
  use EventFixture

  alias Present.Event

  @valid_attrs base_attrs
  @invalid_attrs %{}

  test "changeset with valid attributes" do
    changeset = Event.changeset(%Event{}, @valid_attrs)
    assert changeset.valid?
  end
```

MICHELADA.IO

Elixir y Phoenix han sido una experiencia divertida

REFERENCES

- http://elixir-lang.org/
- https://pragprog.com/book/elixir/programming-elixir
- http://www.phoenixframework.org/
- https://pragprog.com/book/phoenix/programming-phoenix
- Slack: https://elixir-lang.slack.com/

MICHELADA.IO

Gracias

Mario Alberto Chávez
@mario_chavez