ERICH KADERKA

RUBY STORIES V

PRIMEHAMMER.COM

PHOENIX FRAMEWORK

WHAT IS PHOENIX?

- Phoenix is MVC web framework written in Elixir
- A productive web framework that does not compromise speed and maintainability
- Author Chris McCord (wanted to write a realtime web application)
- What does it mean to be productive?

WHAT IS ELIXIR?

- Dynamically typed functional language
- Compiled
- Erlang VM (BEAM)

HOW TO INSTALL && GENERATE NEW APP

- Prerequisites: Elixir, PostgreSQL (MySQL), node.js
- mix archive.install https://github.com/phoenixframework/archives/raw/master/phx_new.ez
- mix phx.new hello_ruby_stories
- cd hello_ruby_stories
- mix ecto.create
- mix phx.server

HOW TO WRITE A FEATURE TEST

- Add Wallaby (phantom.js)
- Hound

```
defmodule HelloRubyStories.BlogTest do
  use HelloRubyStories.IntegrationCase, async: true
  import Wallaby.Query, only: [css: 2, text_field: 1, button: 1]

  test "users create blog posts", %{session: session} do
    session
    |> visit("/blogs/new")
    |> assert_has(css("h2", text: "New Post"))
    |> fill_in(text_field("Title"), with: "First entry")
    |> fill_in(text_field("Body"), with: "Ruby stories 5")
    |> click(button("Save"))
    |> assert_has(css(".alert", text: "You created a blog"))
    |> assert_has(css(".blog-list > .blog", text: "Ruby stories 5"))
  end
end
```

FULFIL THE FEATURE TEST WITH GENERATORS

- mix phx.gen.html Blogs Post posts title:string body:text

```
lib/hello_ruby_stories/blogs/
lib/hello_ruby_stories_web/controllers/post_controller.e
lib/hello_ruby_stories_web/templates/post/
lib/hello_ruby_stories_web/views/post_view.ex
priv/repo/migrations/
test/hello_ruby_stories/
test/hello_ruby_stories_web/controllers/post_controller_
```

PLUG

- A specification for composable modules between web applications
- Connection adapters for different web servers in the Erlang VM

```elixir
# The Plug.Conn module gives us the main functions
# we will use to work with our connection, which is
# a %Plug.Conn{} struct, also defined in this module.
import Plug.Conn

def init(opts) do
  # Here we just add a new entry in the opts map, that we can use
  # in the call/2 function
  Map.put(opts, :my_option, "Hello")
end

def call(conn, opts) do
  # And we send a response back, with a status code and a body
  send_resp(conn, 200, "#{opts[:my_option]}, World!")
end
```

LAYERS OF PHOENIX

- Connection (conn - struct, port, host, headers etc)
- Endpoint
- Router
- Pipelines
- Controller

ENDPOINT

- The endpoint is the boundary where all requests to your web application start.
- It is also the interface your application provides to the underlying web servers
- List of plugs (Static Assets, Logger, Parser, Code reloading)

PIPELINES

- groups functions together to handle common tasks

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

pipeline :api do
  plug :accepts, ["json"]
end
```

2 pipelines, api, browser.

ROUTER

- Router

```
get "/", PageController, :index
resources "/blogs", PostController
```

CONTROLLER

```elixir
def create(conn, %{"post" => post_params}) do
  case Blogs.create_post(post_params) do
    {:ok, post} ->
      conn
      |> put_flash(:info, "Post created successfully.")
      |> redirect(to: post_path(conn, :show, post))
    {:error, %Ecto.Changeset{} = changeset} ->
      render(conn, "new.html", changeset: changeset)
  end
end

def show(conn, %{"id" => id}) do
  post = Blogs.get_post!(id)
  render(conn, "show.html", post: post)
end

def edit(conn, %{"id" => id}) do
  post = Blogs.get_post!(id)
  changeset = Blogs.change_post(post)
  render(conn, "edit.html", post: post, changeset: changeset)
end

def update(conn, %{"id" => id, "post" => post_params}) do
  post = Blogs.get_post!(id)
```

CONTEXT

- Module with public interface to your business logic, separated from the web interface
- Inspiration DDD (multiple roles in different contexts)
- Think a little bit about design upfront
- It's fine to use functions contexts in other contexts
- If you're not sure, just create a new one

EXAMPLE OF CONTEXT

```elixir
@doc """
Creates a post.

## Examples

    iex> create_post(%{field: value})
    {:ok, %Post{}}

    iex> create_post(%{field: bad_value})
    {:error, %Ecto.Changeset{}}

"""
def create_post(attrs \\ %{}) do
  %Post{}
  |> Post.changeset(attrs)
  |> Repo.insert()
end
```

ECTO

- Ecto is a database wrapper and integrated query language. Migrations, Insert/Update/Delete, Queries etc

```
schema "posts" do
  field :body, :string
  field :title, :string

  timestamps()
end

@doc false
def changeset(post, attrs) do
  post
  |> cast(attrs, [:body, :title])
  |> validate_required([:body, :title])
end
```

DEPLOY

- 2 free options. Gigalixir & Heroku
- Build packs
- The whole setup takes 40-50 minutes for the first time
- Limitations: Heroku - sleep, Gigalixir - database
- Other options: docker, distillery (builds the release binary including BEAM), edeliver (deploys the release to VM(s) using SSH)

SECURITY

- Sobelow - security static analysis tool (configuration, vulnerable dependencies, SQL injection etc)

FUTURE

- Phoenix 1.4
- WebPack instead of Brunch
- HTTP2 with cowboy 2.0
- Remove Bootstrap
- Faster development compilation
- Book from Pragmatic Programmers

HOW TO PERSUADE OTHERS?

- Book: Adopting Elixir: From Concept to Production
- Productive
- Secure
- Easy to understand source code (Plug, Controller)
- Zero costs for first MVP deploy
- Reddex

SOURCES

- https://hexdocs.pm/phoenix/Phoenix.html
- https://pragprog.com/book/phoenix14/programming-phoenix-1-4
- https://michal.muskala.eu/2017/05/16/putting-contexts-in-context.html
- https://www.youtube.com/watch?v=MTT1Jl4Fs-E&feature=youtu.be
- https://github.com/keathley/wallaby