Phoenix is not your API

LilleFP #7

$ whoami

- Thomas Gautier
- Co Founder & CTO at Fewlines
- We don't have a website, but we have meaningful Open Source contributions : bamboo_smtp, kaur
- Teacher, Architect, Ops, Developer
- API Lover

$ man fewlines

- We provide Tech Education 🤓
- We build ⚡ API-first Software ⚡ for the Retail industry
- We 😍 Elixir and Elm!
- We're not a Web Agency or a Consulting Company

Why this talk?

- Because our main activity is to develop web applications,
- that we decouple at least in two parts:
  - Back: API Elixir
  - Front: SPA Elm (S for Several)
  - or React (but we prefer Elm)

Your application should not be part of a Web Framework

The Web Framework should be a part of your application

Web Frameworks

- Data
    - Data Manipulation
    - Data Persistence
- HTTP
    - HTTP Requests Handling
    - JSON Serialization/Deserialization
    - Serving Requests
- Web (HTML/Website)
    - Templating
    - Assets Management
    - Internationalization

Phoenix Web Framework

elixir web - Google Search

https://www.google.com/search?client=safari&rls=en&q=elixir+web&ie=UTF-8&oe=UT

GOOGLE

elixir web

Sign in

All Maps Shopping Images Videos More Settings Tools

About 31,600,000 results (0.38 seconds)

Phoenix
phoenixframework.org/
A productive web framework that does not compromise speed and maintainability. ... of connections alongside Elixir's beautiful syntax and productive tooling for ...

Elixir
elixir-lang.github.io/
Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development ...
Getting Started guide · Installing Elixir · Elixir v1.5 released · A Crash Course

Elixir and Phoenix: The Future of Web APIs and Apps?
https://blog.carbonfive.com/.../elixir-and-phoenix-the-future-of-web-apis-and-apps/
Apr 19, 2016 - We decided to kick the tires by rewriting one of our in-house web applications using Elixir and Phoenix so that we could answer the questions ...

Elixir - The next big language for the web - Creative Deletion
www.creativedeletion.com/2015/04/19/elixir_next_language.html
Apr 19, 2015 - In this article I will explain whyI thinkthe Elixir language will make a bigimpact in the world of web development.

elixir-web · GitHub
https://github.com/elixir-web
[WIP] Web framework for Elixir inspired by Rails [#WeberMVC at freenode]. Elixir 376 38 Updated on May 20, 2016. weber-contrib. Extension for Weber ...

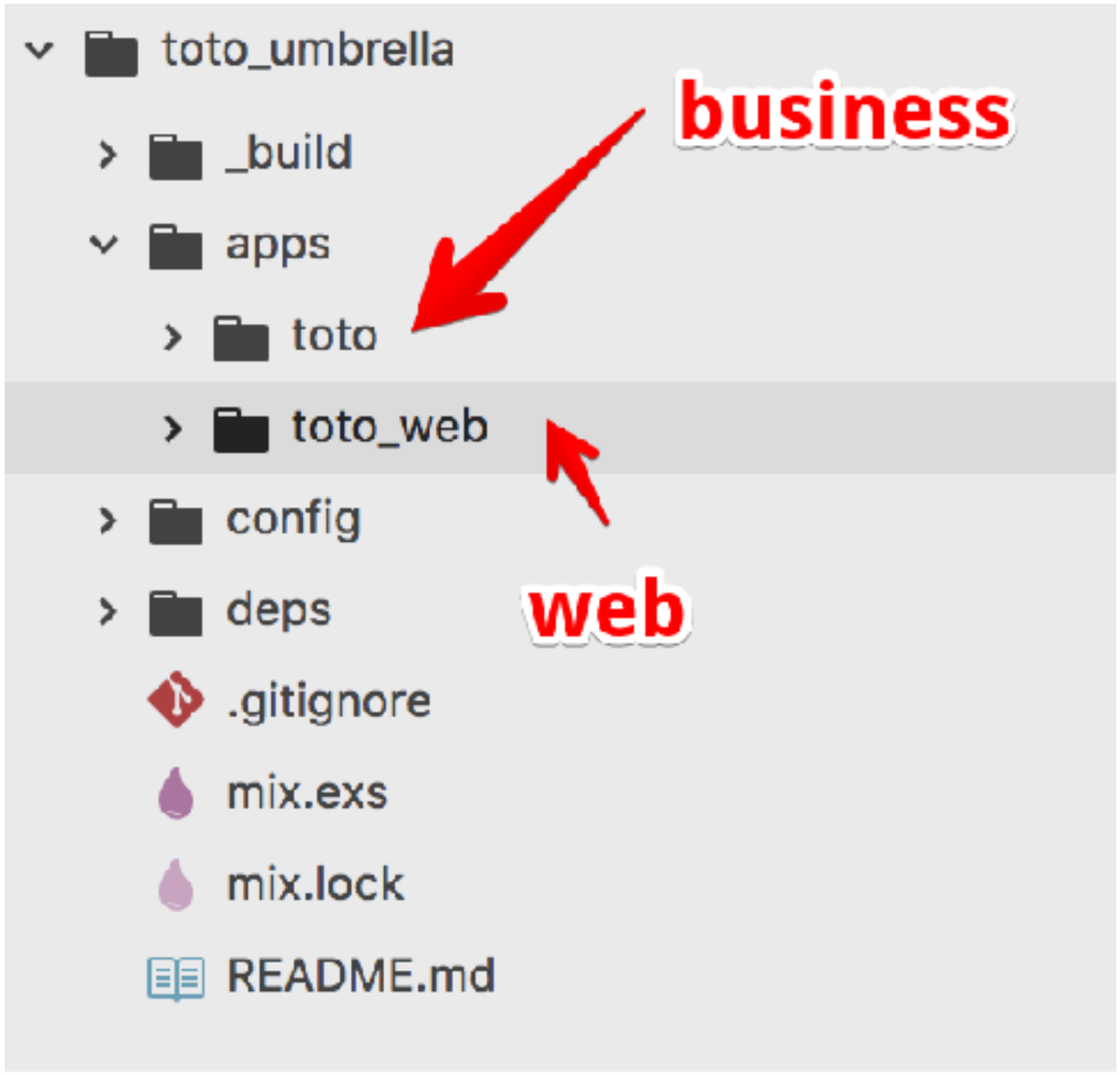
GitHub - phoenixframework/phoenix: Productive. Reliable. Fast.

Phoenix

- Elixir Web Framework
- Smart assembly of composable libraries:
- Ecto - Data Manipulation & Validation
- Plug - HTTP Requests Composition & Routing
- Cowboy - Server (does the same as nginx)

Phoenix

- And some more specific, entangled, components:
  - Channels (Protocol over WebSockets)
  - Assets Management (brunch/webpack boilerplate)
  - Generators (you should never use them 🤬)
  - And a lot of "glue" (some of it very nice) between the different components



Web Framework != App

since Phoenix 1.3 (with --umbrella)

Your app is no longer a part of the Framework

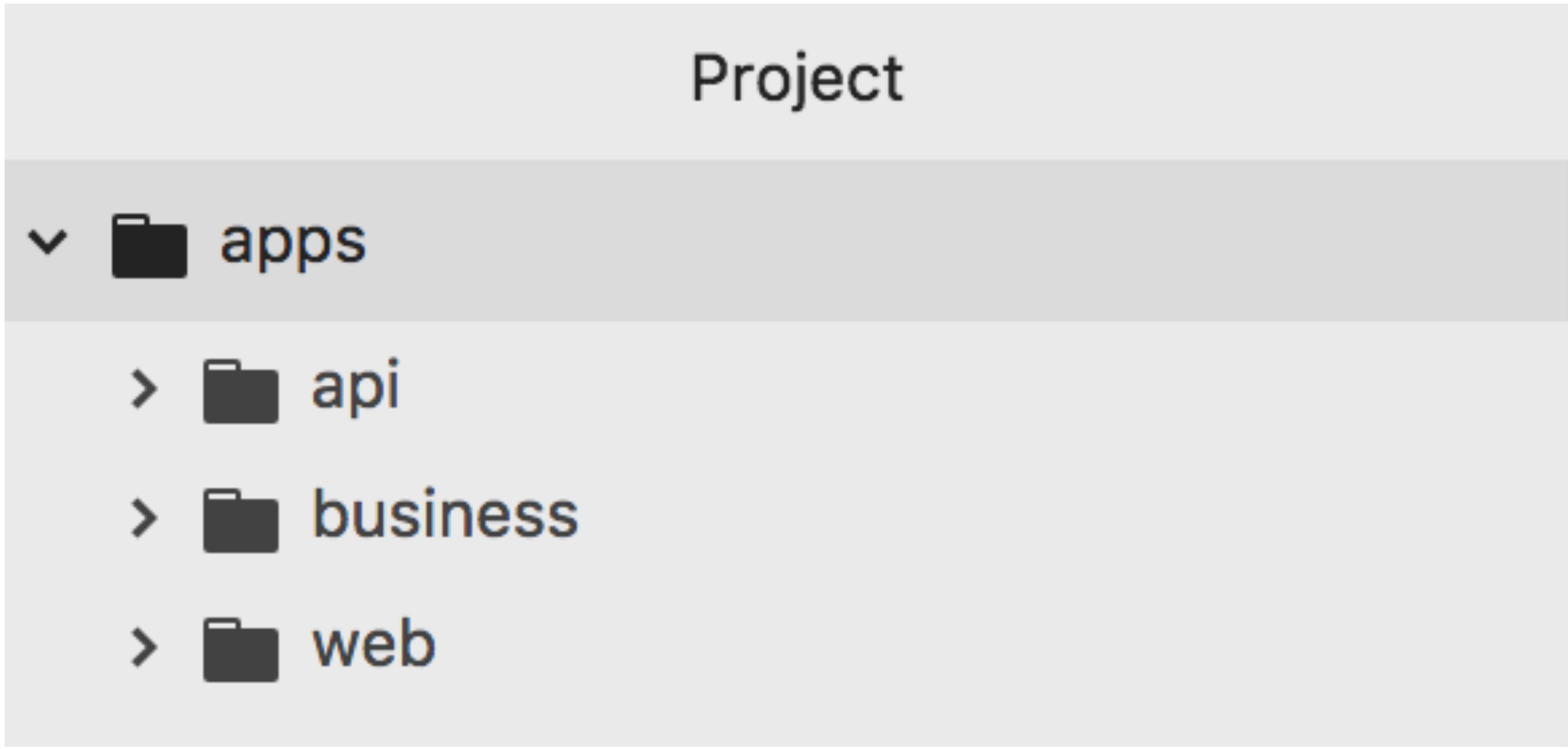
But what about your API?

JSON Restful API

Your API should not be part of a Web Framework

APIs just need JSON & HTTP

- JSON Serialization/Deserialization
- HTTP Request Composition - *Middlewares*
- Serving Requests



This is what we want

and we'll have it

So let's build a Todo

Yes, yet another one

Todo App

- Let's name it Odot
- It will be decoupled into 3 separate applications:

1. Business Logic
2. API exposing JSON
3. Web (Phoenix Framework) consuming the JSON API

Bonus Point 

Not only it is a good design pattern, we will learn a lot about Elixir and Phoenix by doing so

Live Code

Let the gods be with me 🙏

Part 0

Umbrella

Part I

Business Logic

Part I - Business Logic

- We start with an umbrella app that will be our main application and act as a container for our 3 components:
  - todo (business logic)
  - api (http/json)
  - web (http/html)

Todo

- Todo is an Elixir application with a Supervision Tree (more on this later)

- Its main modules are:

- List: the real todolist

- Task: a Struct for our List

- Server: a GenServer to hold our State

- Todo: the API (Interface) of our application

Supervision & Registry

- One of Elixir strenghs is the OTP Platform
- Supervisor is a Process whose job is to monitor other Processes (GenServers or other Supervisors)
- This enables the creation of a Supervision Tree where each Supervisor is responsible of its children
- For each Supervisor, we can select a pre defined strategy to apply when a child Process dies

We need 2 Supervisors

1. **Todo.Application**: launched with the application, it monitors its child: **Todo.Repository**

2. **Todo.Repository**: launched and monitored by **Todo.Application**, it has the following roles:

    1. Monitor **Todo.Supervisor** and Registry

    2. Find an already started GenServer and look for its pid in the Registry

    3. Ask **Todo.Supervisor** to create a new GenServer and register its pid in Registry

Application

```
defmodule Todo.Application do
  use Application

  def start(_type, _args) do
    import Supervisor.Spec, warn: false

    children = [
      supervisor(Todo.Repository, [])
    ]

    opts = [strategy: :one_for_one]
    Supervisor.start_link(children, opts)
  end
end
```

Repository

```elixir
def init(_) do
  children = [
    supervisor(Registry, [[keys: :unique, name: @registry]]),
    supervisor(Todo.Supervisor, [], id: :supervisor)
  ]

  supervise(children, strategy: :one_for_all)
end
```

Supervisor

```
def init(_) do
  children = [
    worker(Todo.Server, [], restart: :transient)
  ]

  supervise(children, strategy: :simple_one_for_one)
end
```

Todo.Server

- Just a GenServer
- Nothing too fancy
- It holds a List of %Task{} and acts as a proxy with the Todo.List module

Todo.List

- Nothing too fancy here either
- Just a very simple implementation of a task list as we can:
  - add a %Task{}
  - remove a %Task{}
  - mark a %Task{} as done

Todo.Task

- Uses Ecto but for Struct and Validation only

- Uses Ecto.UUID for the id key

- But the really nice feature is that we have Ecto.Changesets!

Todo

- This module is the *Public Interface* of our application
- As such, it only exposes the following functions:
  - add
  - done
  - list
  - remove

Demo Time 🤘🏾

Summary Part I

- We now have a fully functional application
- Our Business Logic has no coupling with any framework
- Cherry on the cake, we leverage OTP 🤓

Part II - API

- This is once again a new Elixir application, simply called API
- It also has a Supervisor whose only job will be to launch and monitor our web server
- It has the following roles
    - Expose JSON HTTP Endpoints
    - Parse those requests and call the functions on our top level Todo Public Interface
    - Get the result from those function calls, re-format it if necessary and send the result back as JSON

Dependencies

- This application will only have 3 dependencies:

1. Cowboy - A (very) web server

2. Poison - a very nice JSON serializer/deserializer

3. Plug - an HTTP Request Library with Middlewares & Routing

Application

```elixir
defmodule API.Application do
  use Application

  def start(_type, _args) do
    children = [
      Plug.Adapters.Cowboy.child_spec(:http, API.Router, [], [port: 4000])
    ]

    opts = [strategy: :one_for_one, name: API.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Router 😍

```
defmodule API.Router do
  use Plug.Router

  plug Plug.Logger
  plug Plug.Parsers, parsers: [:json], json_decoder: Poison
  plug :match
  plug :dispatch

  get "/lists/:name",    do: API.Controller.List.index(conn)
  post "/lists/:name",   do: API.Controller.List.create(conn)

  match _ do
    send_resp(conn, 404, "404 - Not Found")
  end
end
```

Controller

```elixir
defmodule API.Controller.List do
  use API.Controller
  alias Todo

  def index(conn) do
    {:ok, list} = Todo.list(conn.params["name"])

    render(conn, API.View.List, "index.json", 200, list: list)
  end
end
```

use API.Controller

I thought we said no magic! 

This is no magic

But this is Elixir best feature at work: Macros

Let's look at the code

`use API.Controller`

- This will import the code from our API.Controller module that we defined
- It will also execute the following

```
defmacro __using__(_params) do
  quote do
    import Plug.Conn
    import API.Controller
  end
end
```

So instead of having

```
defmodule API.Controller.List do
  import Plug.Conn
  alias Todo

  def index(conn) do
    {:ok, list} = Todo.list(conn.params["name"])

    send_resp(conn, 200, Poison.encode!(result))
  end
end
```

We have a much cleaner

```elixir
defmodule API.Controller.List do
  use API.Controller
  alias Todo

  def index(conn) do
    {:ok, list} = Todo.list(conn.params["name"])

    render(conn, API.View.List, "index.json", 200, list: list)
  end
end
```

```
render(conn, API.View.List, "index.json", 200, list: list)
```

This is nice 🤓

So with this custom render

- We can have our View rendering logic separated in its own custom Modules
- One for each Resource we want to render
- And a specific one to handle errors

render list of tasks

```
defmodule API.View.List do
  def render("index.json", %{list: list}) do
    %{list: list}
  end
end
```

render errors

```elixir
defmodule API.View.Error do
  def render("422.json", %{errors: errors}) do
    %{
      code: 422,
      message: "422 - Unprocessable Entity",
      errors: display_errors(errors)
    }
  end

  defp display_errors(errors) do
    errors
    |> Enum.map(fn {field, detail} -> {field, render_detail(detail)} end)
    |> Enum.into(%{})
  end

  defp render_detail({message, values}) do
    Enum.reduce values, message, fn {k, v}, acc ->
      String.replace(acc, "%{#{k}}", to_string(v))
    end
  end
end
```

Demo Time 🤘🏾

Summary Part II

- We now have a fully functional application
- AND a fully functional API to expose it with JSON
- Our Business Logic has no coupling with this API
- Cherry on the cake, we still leverage OTP 🚀

Part III - Web

Web

- This one is now very simple and basic
- We add Phoenix, yes, finally, the framework
- It doesn't care about
  - Business Logic
  - Persistance
  - Authorizations
  - Data Serialization / Deserialization on the API level (JSON)

Web

- Phoenix will just care about what it's really good at
  - Serving web content (assets)
  - Rendering webpages
  - Handling WebSockets with Channels
  - Compiling Assets
  - Internationalization
  - etc...

Demo Time 🤘🏾

Conclusion

- Phoenix is not your Application
- Phoenix is not your API
- Phoenix is your web application (and it's very good at it)
- Bundle your application in an umbrella app
- Create smaller, independent application, with supervision

Thank you 🤦🏽

https://github.com/tgautier/odot