Why I love

What's Elixir?

- Elixir is a dynamic, functional language designed for building scalable and maintainable applications.
- Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development and the embedded software domain.

Syntax

Ruby-like

```
defmodule Exchat.Time do
  epoch = {{1970, 1, 1}, {0, 0, 0}}
  @epoch :calendar.datetime_to_gregorian_seconds(epoch)

  def to_timestamp(datetime) do
    datetime
    |> Ecto.DateTime.to_erl
    |> :calendar.datetime_to_gregorian_seconds
    |> Kernel.-(@epoch)
    |> Kernel.+(datetime.usec / 1_000_000)
  end
end
```

Pipe operator

```
def to_timestamp(datetime) do
   datetime
   |> Ecto.DateTime.to_erl
   |> :calendar.datetime_to_gregorian_seconds
   |> Kernel.-(@epoch)
   |> Kernel.+(datetime.usec / 1_000_000)
 end
```

Pipe operator

```
 def to_timestamp(datetime) do
    erl_datetime = Ecto.DateTime.to_erl(datetime)
    gregorian_seconds
= :calendar.datetime_to_gregorian_seconds(erl_datetime)
    seconds = gregorian_seconds + @epoch
    seconds + datetime.usec / 1_000_000
  end
```

Keyword

```
[{:a, 1}, {:b, 2}] = [a: 1, b: 2]
```

Jiffy - JSON NIFs for Erlang

```
Erlang                              JSON
============================================
{[]}                             -> {}
{[{foo, bar}]}                   -> {"foo": "bar"}
{[{<<"foo">>, <<"bar">>}]}       -> {"foo": "bar"}
#{<<"foo">> => <<"bar">>}        -> {"foo": "bar"}
```

Keyword

Example 1:

```
import Ecto.Query, only: [from: 1, from: 2]
```

Example 2:

```
if false do
  1
else
  2
end

if false, do: 1, else: 2
```

Guard

```
def sum(a, b) when is_integer(a) and is_integer(b) do
  a + b
end
def sum(a, b) when is_list(a) and is_list(b) do
  a ++ b
end
def sum(a, b) when is_binary(a) and is_binary(b) do
  a <> b
end

> sum 1, 2
3
> sum [1], [2]
[1, 2]
> sum "a", "b"
"ab"
```

Pattern Match

= is Pattern Match

```
> [{1, 2}, point, point] = [{1, 2}, {3, 4}, {3, 4}]
> x
1
> y
2
> point
{3, 4}

> [{1, 3}, point, point] = [{1, 3}, {3, 4}, {5, 6}]
** (MatchError) no match of right hand side value:
[{1, 3}, {3, 4}, {5, 6}]
```

```
 test "index/2 returns list of users", %{conn: conn} do
    %{id: id1} = insert_user
    %{id: id2} = insert_user
    conn = get conn, user_path(conn, :index)
    assert [%{"id" => ^id1}, %{"id" => ^id2}] =
json_response(conn, 200)
  end
```

Function dispatch

```elixir
def join("event:general", _payload, socket) do
  send(self, :after_join_general)
  {:ok, socket}
end
def join("event:general:" <> user_id, _payload, socket) do
  if socket.assigns.user.id == String.to_integer(user_id) do
    {:ok, socket}
  else
    {:error, %{reason: "Unauthorized!"}}
  end
end
def join(_, _auth_msg, _socket) do
  {:error, %{reason: "Wrong topic!"}}
end
```

Verify MPEG header

MPEG header format(32 bits): AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM
(refer: http://www.mp3-tech.org/programmer/frame_header.html)

```
defmodule MPEG do
  def decode_header(<<0b11111111111::11, b::2, c::2,
    d::1, e::4, f::2, g::1, rest::9>>), do: :ok
  def decode_header(_), do: :error
end
```

Verify MPEG header

```
> header = <<0b11111111111::11, 0b01::2, 0b01::2, 1::1,
0b0101::4, 0b01::2, 1::1, 0b010101010::9>>
<<255, 235, 86, 170>>
> MPEG.decode_header(header)
:ok
> wrong_header = <<0b11111111110::11, 0b01::2, 0b01::2, 1::1,
0b0101::4, 0b01::2, 1::1, 0b010101010::9>>
<<255, 203, 86, 170>>
> MPEG.decode_header(wrong_header)
:error
```

Functional

First-class functions

```
channel.name
|> String.split(",")
|> Enum.map(&String.to_integer/1)

> Enum.reduce([1, 2, 3, 4], fn(x, acc) -> x * acc end)
24
```

Immutable

Ruby:

```ruby
def change_user(user)
  user.id += 1
end

> user = User.last
=> #<User id: 1184, ...>
> change_user(user)
> user
=> #<User id: 1185, ...>
```

Elixir:

```elixir
def change_user(user) do
  Map.put(user, :id, user.id + 1)
end

> user = %User{id: 1}
> UserUtil.change_user(user)
%User{id: 2}
> user
%User{id: 1}
```

Side effect

```ruby
# Rails
def create
  @work = Work.new(params)

  respond_to do |format|
    if @work.save
      format.html { redirect_to @work }
      format.json { render :show, status: :created,
        location: @work }
    else
      format.html { render :new }
      format.json { render json: @work.errors,
        status: :unprocessable_entity }
    end
  end
end
```

Side effect

```elixir
# Phoenix
def create(conn, %{"work" => work_params}) do
  changeset = Work.changeset(%Work{}, work_params)

  case Repo.insert(changeset) do
    {:ok, _work} ->
      conn
      |> put_flash(:info, "Work created successfully.")
      |> redirect(to: work_path(conn, :index))
    {:error, changeset} ->
      render(conn, "new.html", changeset: changeset)
  end
end
```

It’s really hard to avoid side effect,
but it can be reduced

- Time
- IO(Datebase)

```
def insert_channel_user(channel, user,
      joined_at \\ Exchat.Time.now_datetime) do
  Repo.insert!(%ChannelUser{channel_id: channel.id,
    user_id: user.id, joined_at: joined_at})
end
```

"a style of building the structure and elements of computer programs"

— https://en.wikipedia.org/wiki/Functional_programming

Macro

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

> quote do: changeset.valid?
{{:., [], [{:changeset, [], Elixir}, :valid?]}, [], []}
```

```
defmodule Exchat.User do
  use Exchat.Web, :model
end

defmodule Exchat.Web do
  defmacro __using__(which) when is_atom(which) do
    apply(__MODULE__, which, [])
  end

  def model do
    quote do
      use Ecto.Schema

      import Ecto.Query, only: [from: 1, from: 2]

      alias Exchat.Time, as: Extime
    end
  end
end
```

```
defmacro def(call, expr \\ nil) do
  define(:def, call, expr, __CALLER__)
end
```

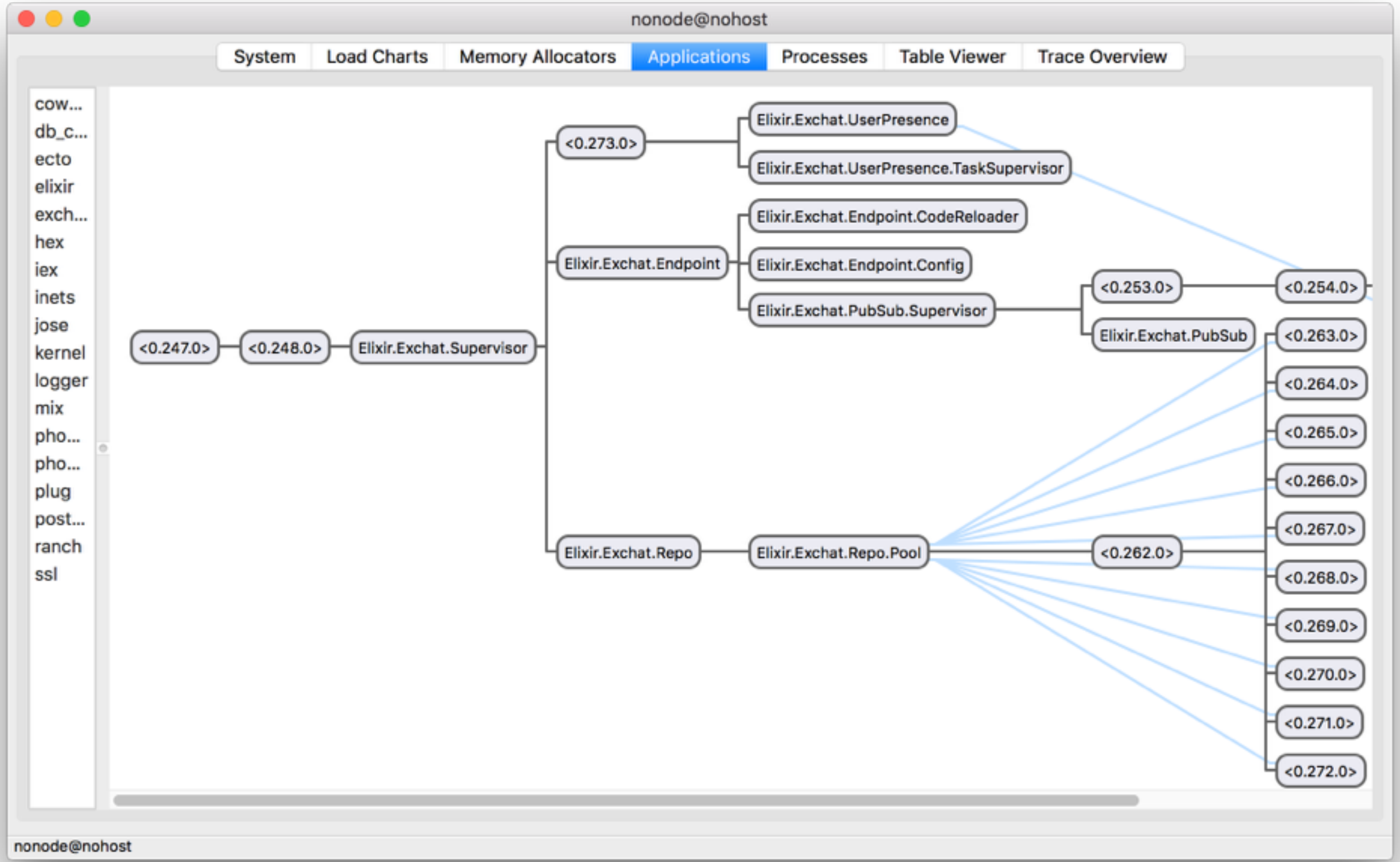
Doc

- first-class citizen
- Markdown
- Doctests

```elixir
@doc """
Examples

  iex> Exchat.Time.to_datetime(1446912799.000321)
  %Ecto.DateTime{year: 2015, month: 11, day: 7, hour: 16,
    min: 13, sec: 19, usec: 321}

  iex> Exchat.Time.to_datetime("1446912799.000321")
  %Ecto.DateTime{year: 2015, month: 11, day: 7, hour: 16,
    min: 13, sec: 19, usec: 321}
"""
def to_datetime(timestamp) when is_binary(timestamp) do

defmodule Exchat.TimeTest do
  use ExUnit.Case, async: true

  doctest Exchat.Time
end
```

Erlang VM & OTP

- Fault-tolerance
- Concurrency
- Distributed


nonode@nohost
System
Load Charts
Memory Allocators
Applications
Processes
Table Viewer
Trace Overview
cow...
db_c...
ecto
elixir
exch...
hex
iex
inets
jose
kernel
logger
mix
pho...
pho...
plug
post...
ranch
ssl
<0.247.0>
<0.248.0>
Elixir.Exchat.Supervisor
<0.273.0>
Elixir.Exchat.UserPresence
Elixir.Exchat.UserPresence.TaskSupervisor
Elixir.Exchat.Endpoint
Elixir.Exchat.Endpoint.CodeReloader
Elixir.Exchat.Endpoint.Config
Elixir.Exchat.PubSub.Supervisor
<0.253.0>
<0.254.0>
Elixir.Exchat.PubSub
Elixir.Exchat.Repo
Elixir.Exchat.Repo.Pool
<0.262.0>
<0.263.0>
<0.264.0>
<0.265.0>
<0.266.0>
<0.267.0>
<0.268.0>
<0.269.0>
<0.270.0>
<0.271.0>
<0.272.0>
nonode@nohost

Application scenarios

- Nerves — Embedded systems
- What’s App(Erlang) — Realtime Web applications
- Riak(Erlang) — Distributed NoSQL Database
- RabbitMQ(Erlang) — Messages Queue
- CouchDB(Erlang) — Document-oriented NoSQL
- Games
- … and more

Table 1.1 Comparison of technologies used in two real-life web servers

Technical requirement	Server A	Server B
HTTP server	Nginx and Phusion Passenger	Erlang
Request processing	Ruby on Rails	Erlang
Long-running requests	Go	Erlang
Server-wide state	Redis	Erlang
Persistable data	Redis and MongoDB	Erlang
Background jobs	Cron, Bash scripts, and Ruby	Erlang
Service crash recovery	Upstart	Erlang

Ecosystem

- iex
- mix
- ExUnit
- hex.pm
- ExDoc

- plug
- Echo
- Phoenix
- maru
- elixometer
- https://github.com/h4cc/awesome-elixir

- Pinterest — Notification system & API rate-limiting system
- Bleacher Report — over 100,000 requests per minute to mobile apps alone
- Puppet Labs — Internal Elixir and Phoenix apps in production. All new internal development targeted at Elixir
- UCloud, Peatio, Nashangban
- https://github.com/doomspork/elixir-companies

That’s why I love Elixir

Q&A