



elixir

A taste of elixir: Building concurrent and reactive web applications

Ruben Amortegui
[@ramortegui](https://github.com/ramortegui)
<https://github.com/ramortegui>



What's Elixir

- Functional Programming Language
- Runs on the Erlang Virtual Machine

Design Goals:

- Compatibility
- Productivity
- Extensibility

Updated Goals(2017)

- Productivity
- Maintainability
- Reliability

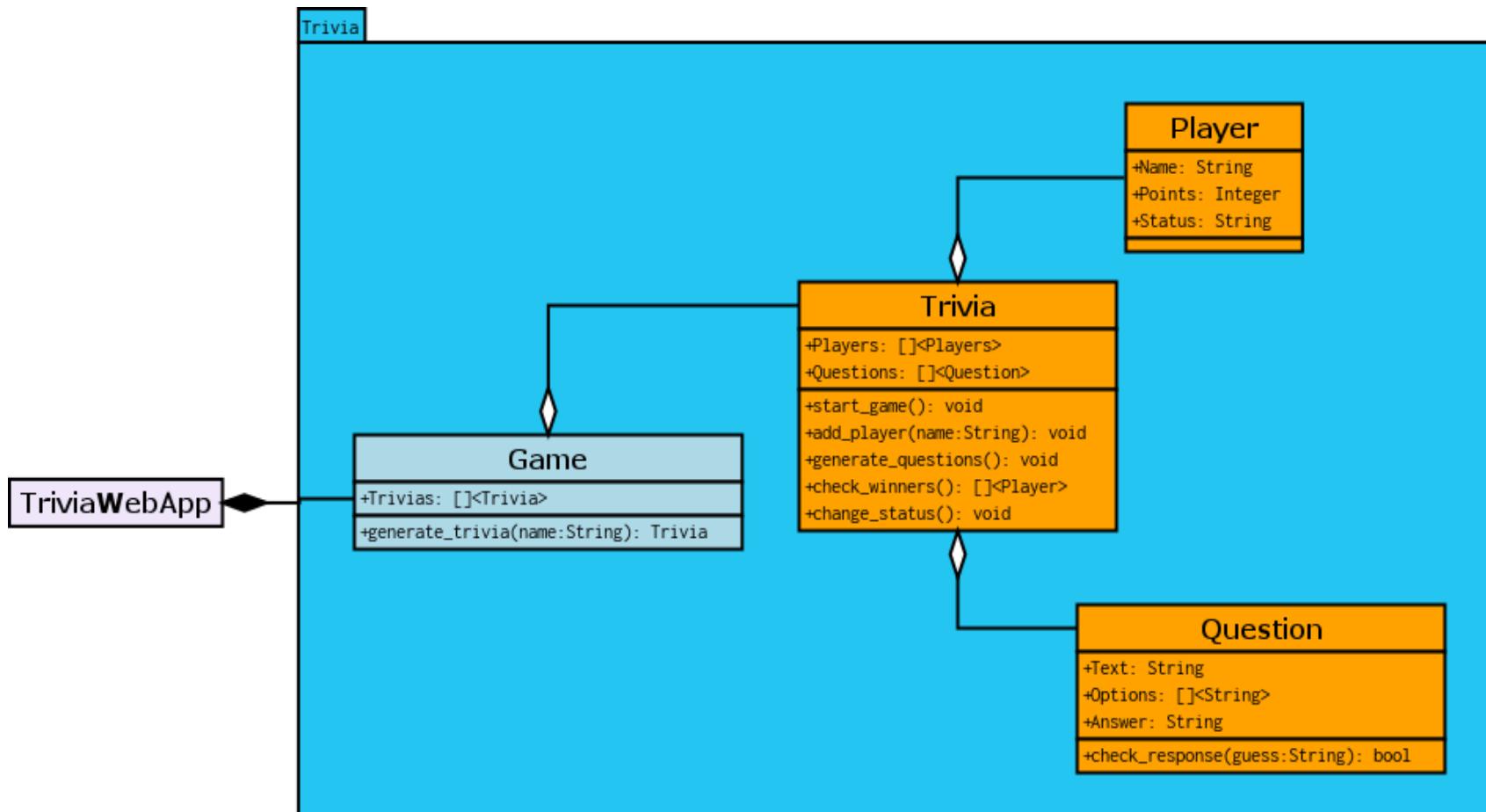
<https://elixir-lang.org/blog/2013/08/08/elixir-design-goals/>

Sample Web App

- Trivia Game
 - Multi-Player
 - Questions come from external service
 - Players can create a trivia game
 - Players can join a trivia game
 - Game timed
 - “Real-time” updates

<https://en.wikipedia.org/wiki/Trivia>

How to create a trivia app in OOP?



Concurrency Issues

- Race conditions
 - Lost updates, access to out-of-date data, etc.
- Workarounds:
 - Confinement.
 - Immutability.
 - Threadsafe data type.
 - Synchronization.

<http://web.mit.edu/6.005/www/fa15/classes/20-thread-safety/>

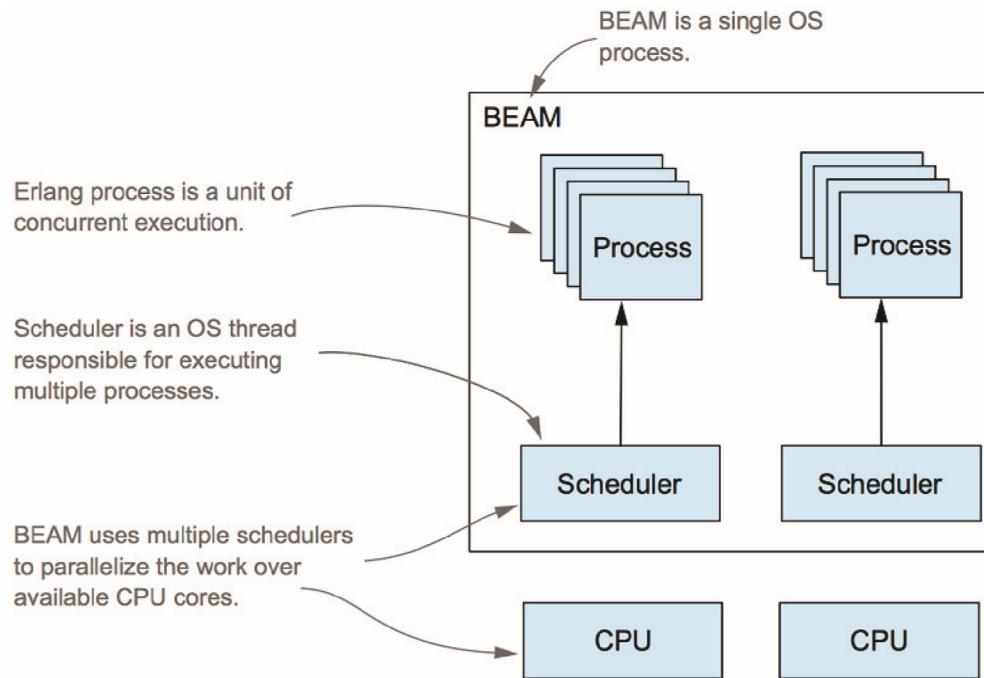
https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.admin.perf.doc/doc/c0005267.html



Erlang/OTP

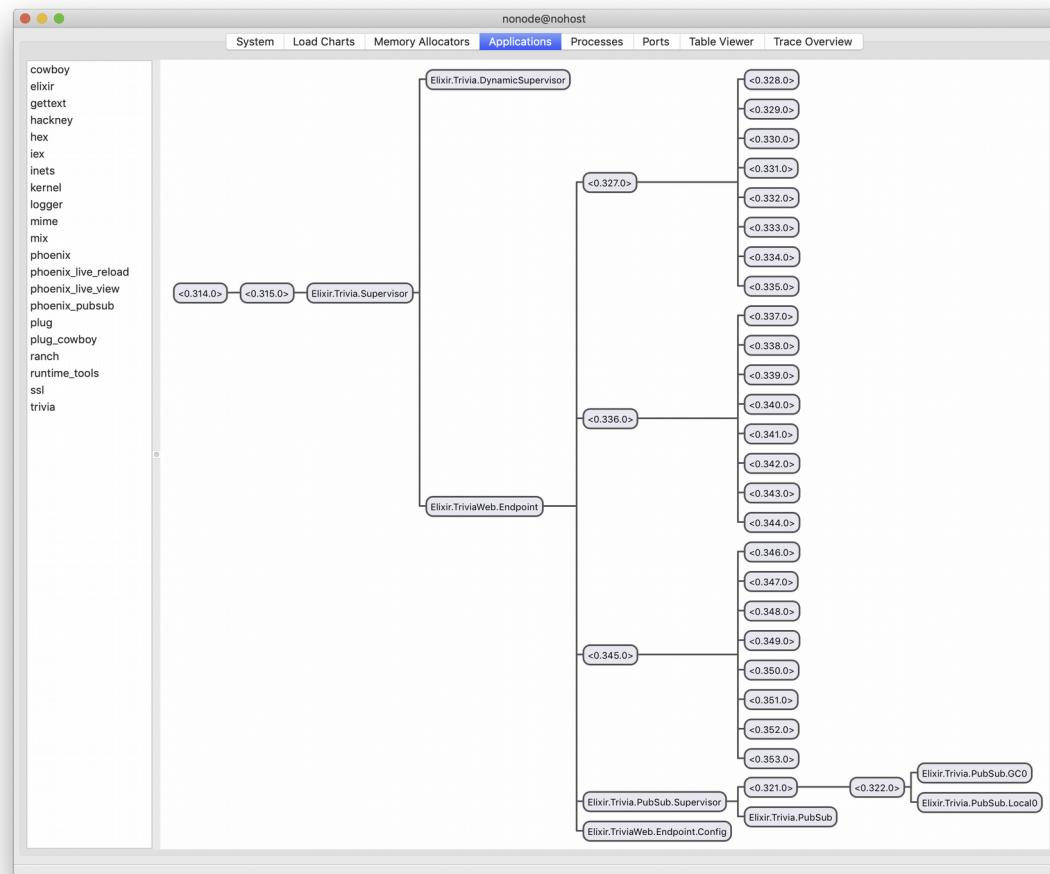
- Complete development environment for concurrent programming.
-  elixir takes advantage of all the ecosystem.

BEAM



<https://freecontent.manning.com/dipping-your-toes-into-elixir/>

Processes



<https://elixir-lang.org/getting-started/processes.html>



Phoenix Framework

- MVC
- Productive, Reliable, and Fast.

Creating an app

```
>mix phx.new trivia --no-ecto
```

Phoenix app Structure

```
› ramortegui trivia> ls -1
README.md
_build
assets
config
deps
lib
mix.exs
mix.lock
priv
test
› ramortegui trivia> █
```

Data Structures

- Trivia.Player
- Trivia.Question
- Trivia.Game
- Trivia.GameServer
- Trivia.DynamicSupervisor

Trivia.Player

```
0 defmodule Trivia.Player do
1   @moduledoc """
2   Module used to define a player for a trivia
3   """
4   defstruct name: "", points: 0, waiting_response: false
5
6   @doc """
7   Returns a `Trivia.Player` struct with the name populated.
8
9   ## Examples:
10
11   iex> Trivia.Player.new("Ruben")
12   %Trivia.Player{name: "Ruben", points: 0, waiting_response: false}
13
14   """
15   def new(name) when is_binary(name) do
16     %__MODULE__{name: name}
17   end
18 end
~
~
~
lib/trivia/player.ex          1,1          All
```

Player.Question

```
1. ramortegui@erudito.local (10.197.26.73) - byobu (vim)

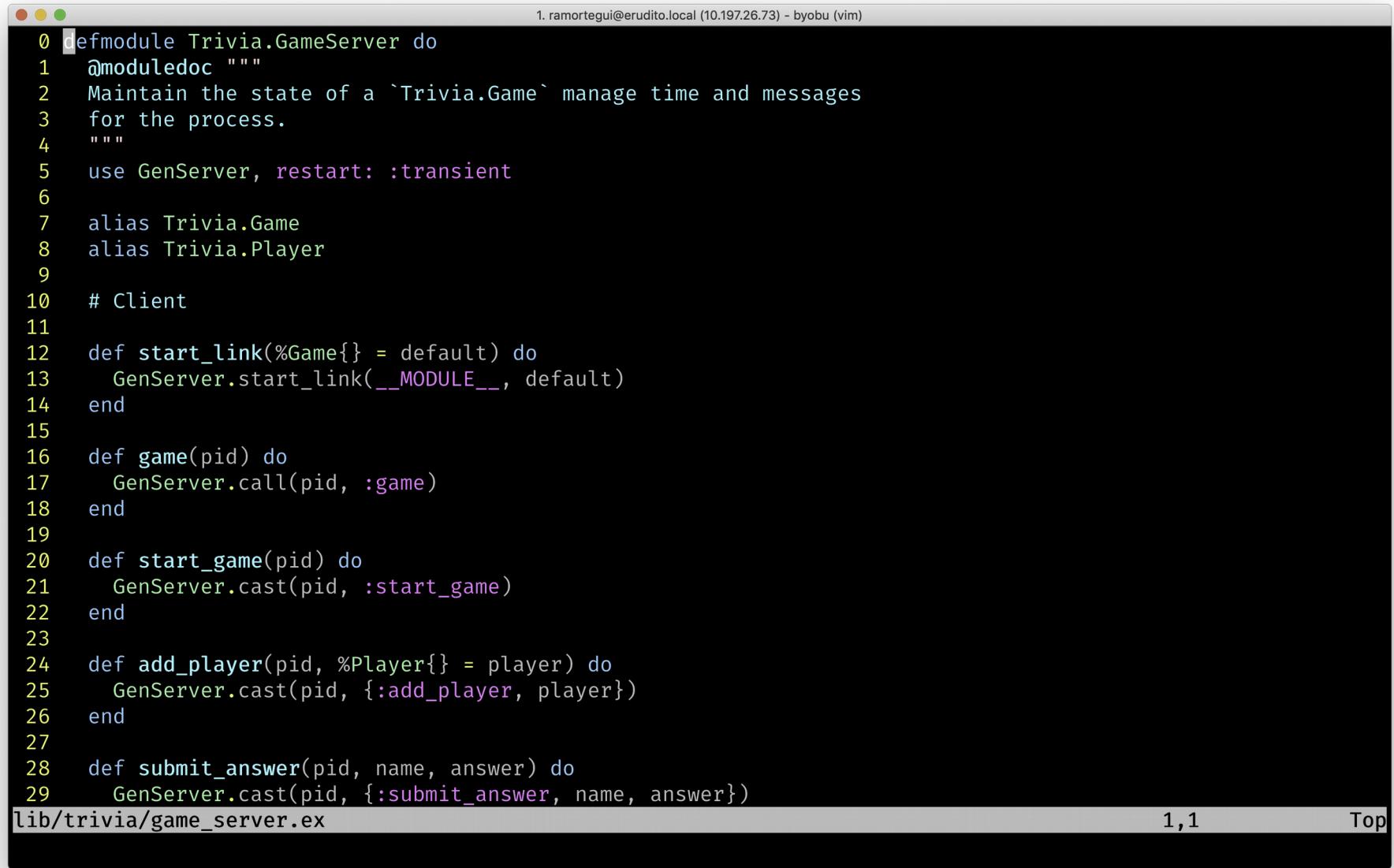
0 defmodule Trivia.Question do
1   @moduledoc """
2   Define question structure and validations.
3   """
4   defstruct text: "", options: [], answer: ""
5
6   @doc """
7   Returns a `Trivia.Question` Struct with shuffled options, the
8   answer is included as part of the options.
9   """
10  def new(%{text: text, options: options, answer: answer}) do
11    shuffled_options = Enum.shuffle([answer | options])
12    %__MODULE__{text: text, options: shuffled_options, answer: answer}
13  end
14
15  @doc """
16  Check the answer of a question
17  """
18  def valid_answer?(%__MODULE__{answer: answer}, guess)
19    when answer == guess,
20    do: true
21
22  def valid_answer?(_question, _guess), do: false
23 end

~
lib/trivia/question.ex                                     1,1          All
"lib/trivia/question.ex" 24L, 670C written
```

Trivia.Game

```
0 defmodule Trivia.Game do
1   @moduledoc """
2   Business logic to operate a trivia game.
3   """
4   defstruct name: '',
5           current_question: nil,
6           questions: [],
7           players: [],
8           status: '',
9           winners: [],
10          counter: 0,
11          counter_total: 0,
12          total_questions: 0,
13          used_questions: 0,
14          number_of_players: 0
15
16 alias Trivia.Game
17 alias Trivia.Player
18 alias Trivia.Question
19
20 @waiting_to_subscribe 10
21 @waiting_to_question 10
22 @waiting_to_end_game 15
23 @amount_of_questions 5
24 @category 18
25
26 @open_trivia_url "https://opentdb.com/api.php?amount=#{@amount_of_questions}&category=#{
27             @category
28             }&type=multiple"
29
lib/trivia/game.ex               1,1               Top
```

Trivia.GameServer



A screenshot of a terminal window displaying the source code for the `Trivia.GameServer` module. The window has a title bar showing the connection details: "1. ramortegui@eruditio.local (10.197.26.73) - byobu (vim)". The code is written in Elixir and defines a GenServer module. The code includes a module docstring, imports for `GenServer`, and aliases for `Trivia.Game` and `Trivia.Player`. It also contains functions for starting a link, getting a game, starting a game, adding a player, and submitting an answer.

```
0 defmodule Trivia.GameServer do
1   @moduledoc """
2     Maintain the state of a `Trivia.Game` manage time and messages
3     for the process.
4   """
5   use GenServer, restart: :transient
6
7   alias Trivia.Game
8   alias Trivia.Player
9
10  # Client
11
12  def start_link(%Game{} = default) do
13    GenServer.start_link(__MODULE__, default)
14  end
15
16  def game(pid) do
17    GenServer.call(pid, :game)
18  end
19
20  def start_game(pid) do
21    GenServer.cast(pid, :start_game)
22  end
23
24  def add_player(pid, %Player{} = player) do
25    GenServer.cast(pid, {:add_player, player})
26  end
27
28  def submit_answer(pid, name, answer) do
29    GenServer.cast(pid, {:submit_answer, name, answer})

```

lib/trivia/game_server.ex 1,1 Top

Trivia.GameServer 2

```
1. ramortegui@erudit0.local (10.197.26.73) - byobu (vim)
0  @impl true
1  def handle_cast(:start_game, %Game{} = game) do
2    check_timer()
3    {:noreply, Game.start_game(game)}
4  end
5
6  @impl true
7  def handle_cast({:add_player, %Player{} = player}, %Game{} = game) do
8    {:noreply, Game.add_player(game, player)}
9  end
10
11 @impl true
12 def handle_cast({:submit_answer, name, answer}, %Game{players: players} = game) do
13   case Enum.find(players, &(&1.name == name)) do
14     %Player{} = player ->
15       {:noreply, Game.check_answer(game, player, answer)}
16
17     nil ->
18       {:noreply, game}
19   end
20 end
21
22 @impl true
23 def handle_info(:change_status, %Game{} = game) do
24   {:noreply, Game.change_status(game)}
25 end
26
27 @impl true
28 def handle_info(:change_question, %Game{} = game) do
29   {:noreply, Game.change_question(game)}
lib/trivia/game_server.ex                                     45,1          57%
```

Trivia.GameServer 3

```
0 def handle_info(:check_timer, %Game{counter: counter} = game)
1   when counter > 0 do
2     check_timer()
3     {:noreply, Game.decrement_counter(game)}
4   end
5
6 @impl true
7 def handle_info(:check_timer, %Game{counter: counter, status: status} = game)
8   when counter <= 0 do
9     game =
10    case status do
11      "waiting" ->
12        Game.change_status(game)
13
14      "playing" ->
15        Game.change_question(game)
16
17      "finished" ->
18        Process.exit(self(), :shutdown)
19    end
20
21    check_timer()
22
23    {:noreply, game}
24  end
25
26  defp check_timer(time \\ 1000) do
27    Process.send_after(self(), :check_timer, time)
28  end
29 end
lib/trivia/game_server.ex
```

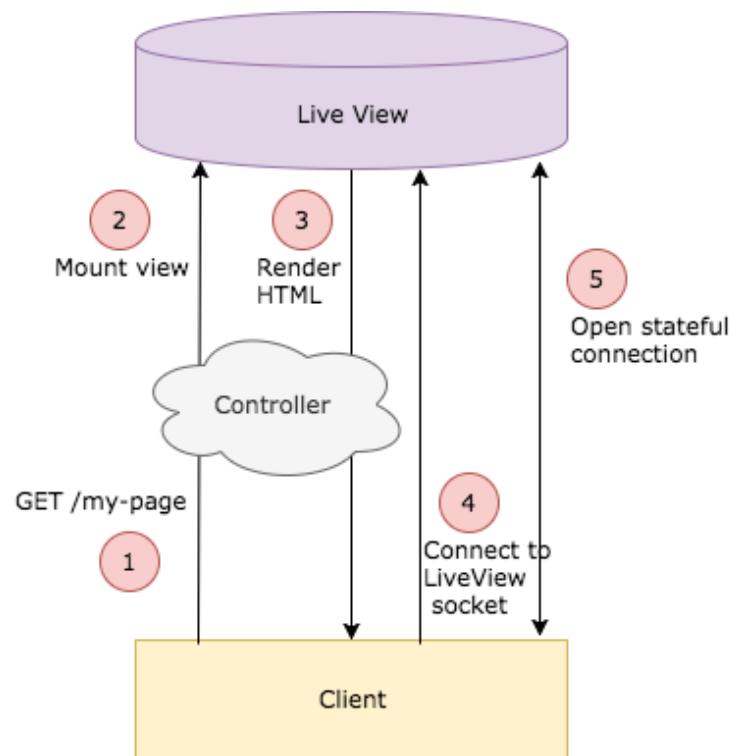
Trivia.DynamicSupervisor

A screenshot of a terminal window titled "1. ramortegui@erudito.local (10.197.26.73) - byobu (vim)". The window displays the source code for a module named "Trivia.DynamicSupervisor". The code defines a module with a module docstring, imports "DynamicSupervisor" and aliases "Game" and "GameServer", and implements start_link/1 and start_child/1 functions. It also includes an @impl true annotation and an init/1 function. The code ends with a final end. The terminal window has a dark background with light-colored text and vim status bars at the bottom.

```
0 defmodule Trivia.DynamicSupervisor do
1   @moduledoc """
2   Dynamic supervisor used to create trivia games on demand.
3   """
4   use DynamicSupervisor
5
6   alias Trivia.Game
7   alias Trivia.GameServer
8
9   def start_link(args) do
10    DynamicSupervisor.start_link(__MODULE__, args, name: __MODULE__)
11  end
12
13  def start_child(%Game{} = game) do
14    spec = {GameServer, game}
15    DynamicSupervisor.start_child(__MODULE__, spec)
16  end
17
18  @impl true
19  def init(_args) do
20    DynamicSupervisor.init(strategy: :one_for_one)
21  end
22 end
~
~
~
~
~
~
~
lib/trivia/dynamic_supervisor.ex
"lib/trivia/dynamic_supervisor.ex" 23L, 516C
```

Phoenix Live View

- Phoenix LiveView enables rich, real-time user experiences with server-rendered HTML.



<https://elixirschool.com/blog/phoenix-live-view/>

Config Live View

- Add LiveView dependency
- Setup configuration (add signing salt)
- Import functions into the web context
- Enable the socket to connect using the /live route.
- Add Phoenix Live View javascript package as dependency.

https://github.com/ramortegui/phoenix_trivia/commit/bc1bae7d32a24a51ca510e2ee3b2971d3c8b2735

TriviaWeb.Live.TriviaView

```
0 defmodule TriviaWeb.Live.TriviaView do
1   use Phoenix.LiveView
2
3   alias Trivia.Game
4   alias Trivia.GameServer
5   alias Trivia.Player
6
7   def render(assigns) do
8     TriviaWeb.PageView.render("trivia.html", assigns)
9   end
10
11  def mount(_session, socket) do
12    if(connected?(socket), do: :timer.send_interval(100, self(), :tick))
13
14    player_name = "p_#{:rand.uniform(10_000_000_000)}"
15
16    socket =
17      socket
18      |> assign(
19        trivia: nil,
20        process: nil,
21        number_of_trivias: 0,
22        number_of_available_trivias: 0,
23        available_trivias: [],
24        player_name: player_name,
25        player_info: nil
26      )
27
28    {:ok, socket}
29  end
30
lib/trivia_web/live/trivia_view.ex          1,1           Top
"lib/trivia_web/live/trivia_view.ex" 134L, 3475C
```

TriviaWeb.Live.TriviaView

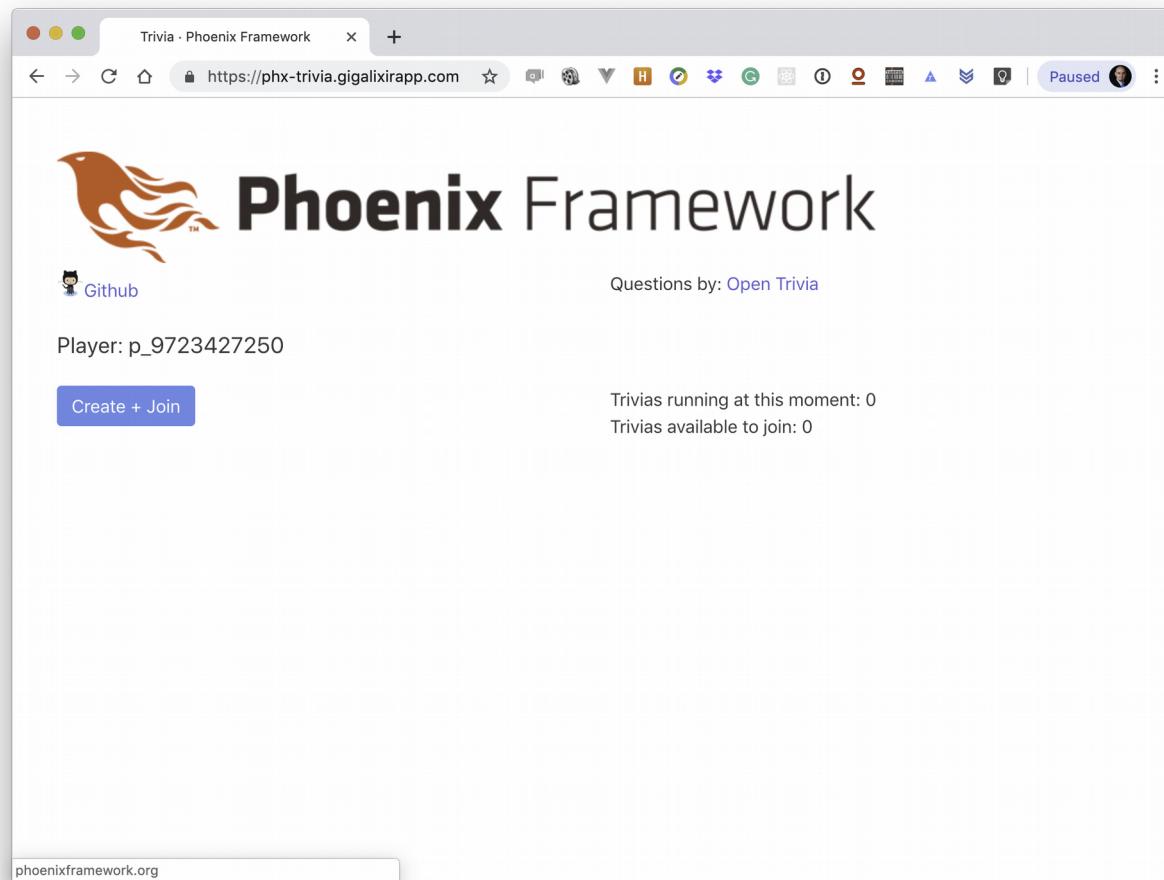
```
0 1 def handle_info(:tick, socket) do
1   {:noreply, update_game_info(socket)}
2 end
3
4 def update_game_info(socket) do
5   game_pid = socket.assigns.process
6   game = socket.assigns.trivia
7
8   if game == nil do
9     trivias(socket)
10  else
11    game = GameServer.game(game_pid)
12
13    if game.status == "finished" and game.counter == 0 do
14      clean_game(socket)
15    else
16      socket
17        |> trivias()
18        |> assign(:trivia, game)
19        |> player_info()
20    end
21  end
22 end
23
24 def handle_event("create_trivia", value, socket) do
25   player_name = socket.assigns.player_name
26   game = Game.new(%{name: value, player: Player.new(player_name)})
27   {:ok, game_pid} = Trivia.DynamicSupervisor.start_child(game)
28   GameServer.start_game(game_pid)
29   {:noreply, assign_game(socket, game_pid)}
30 end
lib/trivia_web/live/trivia_view.ex 32,1 30%
```

Trivia Live View Template

Trivia Live View Template

```
1. ramortegui@erudito.local (10.197.26.73) - byobu (vim)
0 <div class="columns">
1   <div class="column">
2     <button class="button is-info" phx-click="create_trivia" phx-value="<%=@player_name%>">
3       Create + Join
4     </button>
5   </div>
6   <div class="column">
7     <p>Trivias running at this moment: <%= @number_of_trivias %></p>
8     <p>Trivias available to join: <%= @number_of_available_trivias %></p>
9   </div>
10 </div>
11 <div class="columns is-multiline is-mobile">
12   <%= for trivia <- @available_trivias do %>
13     <div class="column is-one-fifth">
14       <button class="button is-info" phx-click="join_trivia" phx-value="<%=@elem(trivia,0)%>">
15         Join
16       </button>
17     </div>
18   <% end %>
19 </div>
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
lib/trivia_web/templates/page/lobby.html.leex [+]           1,1           All
:
```

<https://phx-trivia.gigalixirapp.com>



Summary

- Elixir and Phoenix provide an ecosystem to develop applications with concurrency in mind.
- You don't need to choose between fast and productive when you are going to develop a web application, you could have both.

Q & A?

<https://www.meetup.com/Elixir-Calgary/>

Thanks!

@ramortegui

<https://github.com/ramortegui>