

Seguimiento de Programación
Bases de Erlang e Elixir

Juan David Cardozo Torres
Tomás Castaño Ortiz



Martínez Ceballos Jhan Carlos

Universidad del Quindío
Facultad de Ingeniería
Ingeniería de Sistemas y Computación
Programación III
Armenia, Quindío
2025

Preguntas

1. ¿Elixir se ejecuta sobre Erlang entonces que es Erlang y qué características tiene?
2. ¿Qué ventajas tiene usar elixir en lugar de Erlang?; Tener en cuenta hablar de distribuido, concurrente, resiliente, velocidad, fácil de usar, actualización, código en vivo, metaprogramación/DSL
3. ¿Por qué deberías aprender a usar Elixir?
4. ¿En qué tipo de proyectos es ideal usar Elixir?

Solución.

Pregunta 1.

1. Erlang es un lenguaje de programación que entra dentro de las características de ser funcional y concurrente, dentro de sí, incluye una máquina virtual denominada “BEAM” que dio a luz en los lejanos años 80 una de sus características principales es su amplia colección de bibliotecas que facilitan la creación de sistemas tolerantes a fallos y dadas estas características es más comúnmente usado para las telecomunicaciones o redes sociales y también en aplicaciones de alta disponibilidad.

Hoy en día el uso de Erlang está más centralizado a sistemas distribuidos críticos como WhatsApp, CouchDB, RabbitMQ, entre otros...

Como se mencionó anteriormente, Erlang es un lenguaje funcional y como tal posee un enfoque más basado en:

- Funciones y la recursión en lugar de los ciclos tradicionales (While, For)
- Un manejo de listas y patrones muy potente
- Un Sistema que está pensado para que en caso de error o falla, otro tenga la tarea de reiniciarlo, sin tumbar el sistema.

Pregunta 2

2. Para este punto se comparará Elixir con Erlang en pos de lograr evidenciar que ventajas tiene el usar Elixir por encima de Erlang, en primera instancia se podría decir que Elixir logra ser más eficaz y eficiente que Erlang dado que ofrece una sintaxis más liviana y entendible que Erlang, pero como se mencionó al principio se realizara una comparación entre ambos para ver esto más a fondo y mejor evidenciado

Aspecto	Erlang	Elixir
Distribuido	Es robusto, nativo de BEAM (máquina virtual de Erlang)	Igual de robusto, pero con sintaxis más clara y cómoda para manejar nodos (instancia de la máquina) y clusters (conjunto de nodos interconectados)
Concurrente	Basado en actores (unidad independiente de ejecución) y paso de mensajes	Igual, pero con APIs más limpias y soporte mejorado para flujos
Resiliente	Supervisores (reinicio a un estado limpio), Let it crash (El run time lo elimina de la memoria y lo marca como terminado)	Idéntico, pero con una sintaxis más accesible y cómoda y macros para configurar
Velocidad	Alta en tiempo en su respuesta	Igual ya que usa la misma máquina virtual
Facilidad de Uso	Sintaxis antigua, menos intuitiva	Sintaxis que está inspirada en Ruby, la cual es más legible y expresiva
Actualización de código en vivo	Soportado	También soportado, pero con mejoría de soporte en herramientas de despliegue
Metaprogramación/DSL (Capacidad de un lenguaje para escribir código que genera o modifica código)	Limitado	Muy potente gracias a Macros, permite crear que DSL expresivos para dominios específicos

Con esto visto, se puede evidenciar que Elixir logra emular de forma mucho más eficiente y eficaz las funciones de Erlang, en el siguiente punto se hablará del “Porque” deberíamos utilizar más elixir en pos de dejar más en claro lo expuesto en esta parte.

Pregunta 3.

- Aunque logra simular casi todas las capacidades de Erlang; Elixir te ofrece una versión que hace esto y más de manera mucho más eficiente y efectiva, lo que lo hace más recomendable a la hora de su uso.

Algunos aspectos a tener en cuenta:

- a. Combina lo mejor de dos mundos.
 - La solidez de Erlang
 - Legibilidad de Lenguajes modernos
- b. Productividad
 - Código conciso y sostenible
 - Legibilidad más amigable con el usuario
- c. Demanda Laboral / Uso organizacional
 - Su empleo se ha visto más evidenciado en plataformas que requieren el manejo de usuarios (solicitudes) a nivel de millones de estos simultáneamente
 - Ideal para sistemas distribuidos, chat en tiempo real, notificaciones, IoT.
 - Ejemplos de esto plataformas como Discord, Pinterest, Shopify
- d. Optimización
 - Está diseñado para el soporte de muchas conexiones simultáneas y más importante aún, siendo eficiente al utilizar muy poca cantidad de los recursos disponibles
 - Gracias a la VM de Erlang (BEAM), Elixir maneja millones de procesos concurrentes de manera más eficiente
- e. Enfoque
 - Programación funcional
 - Estimula un estilo de programación que es inmutable y declarativo, en otras palabras es un sistema que está preparado al ser más robusto para los errores

Pregunta 4.

4. ¿Y en dónde debo usar elixir?


Principalmente en proyectos que pueden tender a ser sistemas concurrentes, distribuidos, resilientes y escalables, en especial con una sintaxis más moderna y con

un ecosistema que promueve la producción, también en proyectos que requieran una gran cantidad de conexiones simultáneas, puesto que elixir busca usar la menor cantidad de recursos necesarios.

Algunos ejemplos de su empleo serían.

- Sistemas de comunicación y mensajería en tiempo real
- Procesamientos de datos en tiempo real
- Sistemas que puedan tender a fallar de concurrente, pero que aun así logre tolerar dichos fallos.
- Aplicaciones Web con muchas conexiones o solicitudes simultáneas
- Plataformas enfocadas en el Trading.

Actividad de Clase.



**** ACTIVIDAD - 4 - (3 de 3) ****



Observe los siguientes dos programas y conteste:
¿Qué hacen?, ¿Son equivalentes? y ¿Cuál es más legible?

5. Repita el proceso con un archivo llamado *termina_en-01.exs* y el código

```
IO.puts(String.ends_with?(String.upcase("Hola Mundo"), "UNDO"))
```

6. Repita el proceso con un archivo llamado *termina_en-02.exs* y el código

```
"Hola Mundo"  
|> String.upcase()  
|> String.ends_with?("UNDO")  
|> IO.puts()
```


en conexión territorial
28

The screenshot shows an IDE with several tabs: `sumaNumeros.exs`, `multiplicacion.exs`, `ejemplo_elixir.exs` (active), and `hello_world.ex`. The active file contains two code snippets:

```
1 #Caso1
2 IO.puts(String.ends_with?(String.upcase("Hola Mundo"), "UNDO"))
3
4 #Caso 2
5 "Hola Mundo"
6 |> String.upcase()
7 |> String.ends_with?("UNDO")
8 |> IO.puts()
9
```

Below the code editor, the output pane shows the execution results:

```
PROBLEMAS SALIDA ... Filtro Code
[Running] elixir "c:\Users\juand\Desktop\PracticaPIII\ejerciciosElixir\ejemplo_elixir.exs"
true
true
[Done] exited with code=0 in 2.03 seconds
```

- Ambos casos cumplen la misma función, solo que evidentemente de forma distinta, principalmente en el caso #1, convierte la cadena “Hola mundo” en mayúsculas gracias al `String.upcase`, y el `String.ends_with?(String.upcase(“Hola Mundo”), “UNDO”))` verifica que el “Hola mundo”, termina en “UNDO”, para posteriormente al verificar que esto es verídico confirmar imprimiendo `true`; El caso #2 cumple la misma función solo que este utiliza el operador *PIPE*, de igual manera convierte la cadena “Hola mundo” en mayúsculas con el `String.upcase()`, para comprobar que la cadena termina en “UNDO”, para de igual manera confirmar la veracidad de esto con un `true`.
- Visto el anterior punto, vemos que la mayor diferencia de ambos fragmentos de código, es el uso del operador *PIPE*, pero generalmente dado que ambos códigos cumplen la misma función, se puede concluir que son equivalentes.
- El caso #2 es más legible, puesto que, en menos código, permite cumplir la misma función del caso 1, además de que gasta menos memoria y que permite su edición de forma más familiar con el usuario.

Implementaciones

- Clase del 19/08/2025

Util.ex

```
ejercicios_elixir > util.ex > {} Util > pedir_informacion/0
1  defmodule Util do
2
3      def mostrar_mensaje(mensaje) do
4          mensaje
5          |> IO.puts()
6      end
7
8      def pedir_informacion() do
9          IO.gets("Ingresar su nombre: ")
10         |> String.trim()
11     end
12 end
13
```

- Definimos módulos y librerías permanentes para usar en cualquier otra parte del proyecto.

Secuencial.exs

```
Secuencial.exs X Problema1_publico.exs X Problema1_privado.exs util.ex
ejercicios_elixir > Secuencial.exs > ...
1  defmodule Secuencial do #-> UpperCammelCase
2      def mostrar_mensaje() do
3          "Hola a todos de prueba"
4          |> IO.puts()
5      end
6
7      def mostrar_mensaje_una_linea(), do: IO.puts("Hola a todos")
8
9
10     defp mostrar_mensaje_privado(mensaje) do
11         mensaje
12         |> IO.puts()
13     end
14
15     def invocar_privado() do
16         "Mensaje privado desde una función"
17         |> mostrar_mensaje_privado()
18     end
19
20
21 end
22
23 Secuencial.invocar_privado()
24 Secuencial.mostrar_mensaje_una_linea()
25 Secuencial.mostrar_mensaje()
26
27 #(condicion)? true : False //Ternario//
28
```

- Funciones para mostrar mensajes de diversas maneras

```
Sequencial.exs  Problema1_publico.exs x
ejercicios_elixir > VistoEnClase > Problema1_publico.exs > {} Mensaje_1_publico
1  defmodule Mensaje_1_publico do
2    def mostrar_mensaje_bienvenida() do
3      "Bienvenidos a la Empresa Once Ltda"
4      |> IO.puts()
5    end
6
7  end
8
9  defmodule Test do
10
11    def run_test do
12      Util.mostrar_mensaje("Bienvenido a la Empresa Once Ltda")
13    end
14
15  end
16
17  Mensaje_1_publico.mostrar_mensaje_bienvenida()
18  Test.run_test()
19
```

- Uso del [Util.ex](#) para la impresión de mensajes

```
Sequencial.exs  util.ex x
ejercicios_elixir > VistoEnClase > util.ex > {} Util
1  defmodule Util do
2    @moduledoc """
3    Modulo de funciones de mostrar mensaje y pedir información
4    """
5
6    @doc """
7    Funcione mostrar mensaje
8    """
9    def mostrar_mensaje(mensaje) do
10      mensaje
11      |> IO.puts()
12    end
13
14    @doc """
15    Funcion de pedir informacion
16    """
17    def pedir_informacion() do
18      IO.gets("Ingresar su nombre: ")
19      |> String.trim()
20    end
21
22    def show_message(message) do
23      System.cmd("Java", ["-cp", ".", "Mensaje", message])
24    end
25
26    def input_data(data) do
27      System.cmd("Java", ["-cp", ".", "Mensaje", "input", data])
28      |> elem(0)
29      |> String.trim()
30    end
31  end
```


- Implementación de Funciones en [util.ex](#) para la realización de una actividad seguimiento

Referencias

1. Sobre Erlang y su relación con Elixir

Elixir runs on the Erlang VM (BEAM):

- Documentación oficial de Elixir: <https://elixir-lang.org/> (Sección "Why Elixir?").

Libro "Programming Erlang" (Joe Armstrong, creador de Erlang) – Capítulo 1: "The Erlang View of the World".

Características de Erlang (conurrencia, distribución, OTP):

- Sitio oficial de Erlang: <https://www.erlang.org/> (Sección "Highlights").

Paper histórico: "Making reliable distributed systems in the presence of software errors" (Joe Armstrong, 2003).

Caso de éxito: WhatsApp (artículo de Wired, 2017: How WhatsApp Became the World's Most Popular Messaging App).

2. Ventajas de Elixir sobre Erlang

Sintaxis y productividad:

- Libro "Elixir in Action" (Saša Jurić) – Comparación en el prefacio.

Charla de José Valim (creador de Elixir): Elixir: The Power of Erlang, the Joy of Ruby.

Conurrencia y distribución:

Documentación de Elixir sobre procesos:

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

Meta-programación y DSLs:

Guía oficial de macros:

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

Actualización en vivo:

Documentación de OTP:

- <https://learnyoussomeerlang.com/relups>.

3. Razones para aprender Elixir

Demanda laboral y casos de éxito:

- *Blog de Discord: Why Discord is switching from Go to Rust (aunque usan Elixir para otros componentes).*
- *Shopify Engineering: How Shopify Uses Elixir to Scale.*

Ecosistema (Hex, Phoenix):

- *Sitio de Phoenix Framework: <https://www.phoenixframework.org/>.*

4. Proyectos ideales para Elixir

Aplicaciones en tiempo real:

Caso de estudio: Bleacher Report (Elixir en deportes).

IoT (Nerves):

- *Sitio oficial de Nerves: <https://www.nerves-project.org/>.*

Web y APIs (Phoenix):

- *Benchmark vs. otros frameworks: <https://github.com/mroth/phoenix-showdown>.*