TP Elixir

Procesos Ping/Pong utilizando GenServer y Estructuras

Implementar dos procesos **Ping/Pong** que intercambien mensajes entre sí (por ej: :ball) con el siguiente API de cliente

Ping.start_link(:ok) - Lanza el proceso Ping y lo registra localmente con el nombre Ping

Pong.start_link(:ok) - Lanza el proceso localmente con el nombre Pong

Ping.play(n) - comienza el envío de mensajes entre Ping/Pong con 'n' mensajes.

Tanto Ping como Pong deben devolver el mensaje al otro proceso inmediatamente lo reciban. (Es recomendable utilizar la función *GenServer.cast(pid, msg)* para intercambiar los mensajes y evitar bloqueos de los procesos)

Ping debe contabilizar la cantidad de mensajes enviados y poseer una función **Ping.stats()** que devuelva una estructura con la cantidad de mensajes enviados junto a la cantidad de mensajes procesados por segundo.

Puede utilizar la función System.monotonic_time(:second) para medir el tiempo en segundos.

Traductor, utilizando GenServer y Supervisor

Reimplementar el proceso traductor realizado en clases utilizando GenServer.

- Registrarlo bajo el nombre local Translator
- Crear un módulo para supervisar el proceso de traducción (con Supervisor)
- Agregar soporte para traducir textos completos (no sólo palabras) y un diccionario recibido como parámetro.
- Implementar una estructura Translator.State que mantenga el estado del traductor incluyendo, la cantidad de documentos traducidos, la cantidad de palabras y la frecuencia de las palabras traducidas (mapa del tipo %{"hello" -> 4, "world" -> 2}). Puede utilizar Enum.frequencies y Map.merge(map1, map2, func)

Generador de JSON utilizando Protocolos

Implementar un módulo **Text.to_json/1** que reciba un valor y devuelva su representación en formato JSON. La función debe soportar los tipos predefinidos del lenguaje (por ej: enteros, strings, mapas y listas).

Cuando la función no reconozca un valor (algún ‘struct’ definido por el usuario), deberá buscar una implementación del protocolo **JsonEncoder** que provea una función **to_json/1**

Eventos de HackerNews

Crear un proceso con GenServer que recupere todas las noticias del sitio https://news.ycombinator.com/newest (puede utilizar HTTPoison y Floki para descargar y parsear la página html).

El proceso deberá buscar las noticias a un intervalo regular de tiempo (por ejemplo cada 5 minutos)

Cada vez que encuentre un nuevo link a una noticia, deberá enviar un evento a todos los procesos interesados.

Cualquier proceso debe poder suscribirse/desuscribirse para recibir mensajes con las nuevas noticias que se descubran en el sitio web.

Chat Distribuido

Implementar un sistema de chats con GenServer que pueda usarse en forma distribuida dentro de un cluster de Erlang.

El sistema debe estar formado por un actor principal ChatSystem que se registra globalmente y poseer un mapa con todos los 'rooms' que se hayan creado. Cada room está representado por un nombre y un actor que lo administra.

ChatSystem debe soportar el siguiente protocolo/API:

```
ChatSystem.create_room(name) -> room # room debe contener internamente el pid
ChatSystem.lookup_room(name) -> room
ChatSystem.list_rooms() -> List of Room Names
```

El actor ChatRoom debe soportar el siguiente protocolo/API:

ChatRoom.join(chat_room, user_name, client_pid) -> {:joined, Messages }
ChatRoom.talk(chat_room, user_name, message) -> Sin respuesta
ChatRoom.list_users(chat_room) -> List of users

Los clientes deben recibir los mensajes nuevos ({ :new_message, Message })

Ejemplo de cómo debe poder usarse desde el shell:

```
> ChatSystem.start_link(:ok)
> {:ok, elixir_room} = ChatSystem.create_room("Elixir").

> ChatRoom.join(elixir_room, "Juan", self()).
> flush().

> ChatRoom.talk(elixir_room, "Juan", "Hola").
> flush().

> ChatRoom:talk(elixir_room, "Juan", "Mundo!").
> flush().
```