

Betunfair

Álvaro Plaza Carro

Agenda

Introducción

Requerimientos del sistema

Persistencia

Diseño del sistema

Escalabilidad

Tests

Conclusión





Introducción

El propósito de este proyecto final es la creación de un sistema de intercambio de apuestas escalable, desarrollado mediante el uso de Elixir, un lenguaje que se destaca por su tolerancia a fallos y sus capacidades de concurrencia. La funcionalidad principal de la plataforma estará encapsulada en un módulo de Elixir, que integrará las funciones esenciales necesarias para gestionar las apuestas, mercados, administrar las cuentas de los usuarios y emparejar apuestas y contrapuestas.

Requerimientos del sistema

Base de datos

- **PostgreSQL:** Se utilizará como nuestro sistema de gestión de bases de datos. PostgreSQL es robusto, escalable y maneja eficazmente transacciones concurrentes, lo que es fundamental para nuestro sistema de apuestas

Elixir

- **Mix:** Esta es la herramienta de construcción y gestión de proyectos de Elixir. Usaremos Mix para gestionar nuestras dependencias, correr pruebas y más.
- **Ecto:** Biblioteca de Elixir para interactuar con bases de datos. Nos ayudará a interactuar con nuestra base de datos PostgreSQL, proporcionando un API limpio y eficiente.
- **GenServer:** Tiene un comportamiento OTP (Open Telecom Platform) que implementa la funcionalidad de un servidor genérico en una arquitectura cliente-servidor. Esta será una parte fundamental de nuestra arquitectura, ya que GenServer nos permitirá manejar el estado y la concurrencia de una manera segura y eficiente.

Persistencia de Datos con Ecto y PostgreSQL

- **Abstracción de la Base de Datos:** Ecto proporciona una interfaz común para diferentes sistemas de bases de datos, permitiendo el uso de distintas bases de datos con cambios mínimos en el código.
- **Integración con Changesets:** Changesets en Ecto permiten realizar un seguimiento y verificación de los cambios en nuestros datos antes de aplicarlos a la base de datos. Esto nos proporciona una validación de datos eficiente y control sobre cuándo y cómo se aplican los cambios.
- **Consultas Componibles:** Ecto utiliza su propio lenguaje de consultas, que permite componer consultas de una manera legible y segura.

Diseño del Sistema

- **Usuarios (Users):** Cada usuario tiene un ID único y un nombre. Los usuarios mantienen un balance, que puede aumentar o disminuir a medida que se realizan apuestas. Los usuarios pueden tener múltiples apuestas asociadas a ellos.
- **Apuestas (Bets):** Cada apuesta pertenece a un usuario y a un mercado. Incluye información sobre la cantidad de la apuesta, las probabilidades, el tipo de apuesta (back o lay), y el estado de la apuesta.
- **Mercados (Markets):** Cada mercado tiene un nombre único y una descripción. Los mercados mantienen un estado que puede cambiar con el tiempo, y también tienen un resultado que se determina una vez que el mercado se resuelve. Las apuestas se realizan en relación a un mercado.
- **Emparejamientos (Matches):** Cada emparejamiento representa el hecho de que una apuesta back y una apuesta lay han sido emparejadas. Incluye la cantidad que ha sido emparejada entre estas dos apuestas

Escalabilidad y Confiabilidad

- 1. Concurrencia:** GenServer proporciona una abstracción para la creación y gestión de procesos concurrentes. Esto significa que puedes realizar múltiples tareas a la vez, lo que puede ayudar a aumentar la capacidad de tu aplicación para manejar cargas de trabajo más grandes.
- 2. Aislamiento y recuperación de errores:** Cada GenServer se ejecuta en su propio proceso, que está aislado de otros procesos. Si un proceso falla, no afectará a los demás. Además, OTP proporciona mecanismos para manejar y recuperarse de los errores. Por ejemplo, puedes utilizar supervisores para reiniciar automáticamente los procesos que fallan.
- 3. Mensajería asíncrona:** GenServer permite la comunicación asíncrona entre procesos a través de mensajes. Esto significa que un proceso puede continuar trabajando sin tener que esperar a que otro proceso termine su trabajo.
- 4. Estado mantenible:** Cada GenServer puede mantener su propio estado a lo largo del tiempo. Esto puede ser útil para tareas como mantener las conexiones a una base de datos, almacenar información sobre el estado actual de un usuario o cualquier otro dato que necesite persistir durante la vida del proceso.
- 5. Hot code swapping:** OTP, y por ende GenServer, permite la actualización de código en tiempo real sin detener el sistema. Esto puede ser especialmente útil para sistemas en producción que requieren alta disponibilidad.
- 6. Patrones de diseño de sistemas robustos:** GenServer es parte de las herramientas OTP que incluyen patrones de diseño predefinidos para construir aplicaciones robustas, tolerantes a fallos y con capacidad para recuperarse de los errores.

Escalabilidad y Confiabilidad

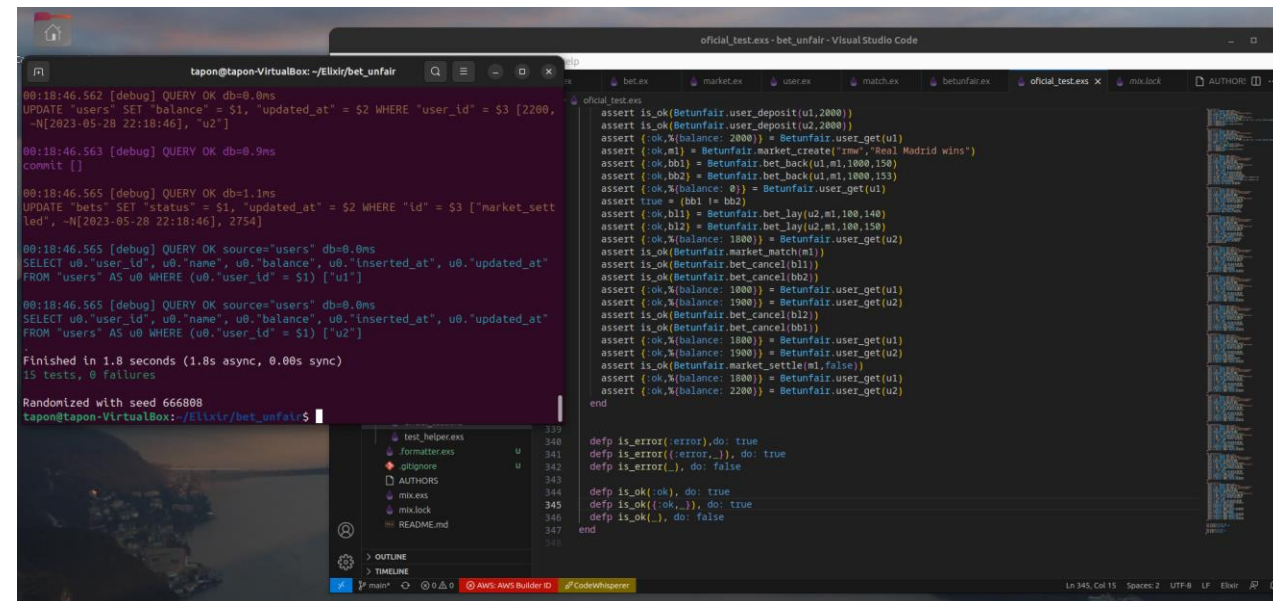
- **Ecto:** Ecto es un lenguaje de consultas integrado y un proyecto de adaptador de base de datos en Elixir. Nos permite manejar transacciones de manera muy eficiente y segura. En nuestro código, las transacciones se manejan mediante la función `Repo.transaction`, que nos asegura que todas las operaciones en la base de datos se realizan de forma atómica. Esto significa que si alguna parte de la transacción falla, todas las operaciones se revertirán, manteniendo así la integridad de nuestros datos.

Testing

Durante el desarrollo del programa, se han implementado una serie de pruebas unitarias usando ExUnit.

En un principio las pruebas se centraron en controlar las entradas y salidas en la base de datos. Comprobar los distintos algoritmos desarrollados. Hasta poco a poco cumplir todos los tests de betunfair_test_exs

Captura tests



The screenshot shows a development environment with a terminal window on the left and a code editor on the right. The terminal window, titled 'tapon@tapon-VirtualBox: ~/elixir/bet_unfair', displays the output of running tests. It shows several debug messages for SQL queries and updates, followed by a summary: 'Finished in 1.8 seconds (1.8s async, 0.00s sync) 15 tests, 0 failures'. The code editor, titled 'official_test_exs - bet_unfair - Visual Studio Code', shows the test code in Elixir. The code includes assertions for user deposits, market creation, bets, and cancellations, using functions like `assert_is_ok` and `assert`.

Conclusión

Un extenso proceso de desarrollo orientado a crear un proyecto que cumpliera con todos los requisitos y pruebas. El reto principal fue la comprensión e implementación de los algoritmos para emparejar y resolver apuestas, que requirió un considerable esfuerzo. Un área de mejora es la optimización del rendimiento al manejar un gran número de elementos. Por ejemplo, la función `market_bets` puede devolver todas las apuestas en un mercado, que podría ser un número considerable. En la versión actual del proyecto, hemos utilizado `Enum.map(&&1.id)` pero reconocemos que puede haber soluciones más eficientes disponibles, como las transmisiones perezosas (`lazystreams`) descritas en la documentación de Elixir. Implementar tal característica permitiría un mejor rendimiento al no necesitar inspeccionar todas las apuestas devueltas.

Muchas Gracias