BETUNFAIR

A betting exchange platform

STRUCTURE

Database

We store application data with a the external library **CubDB**.

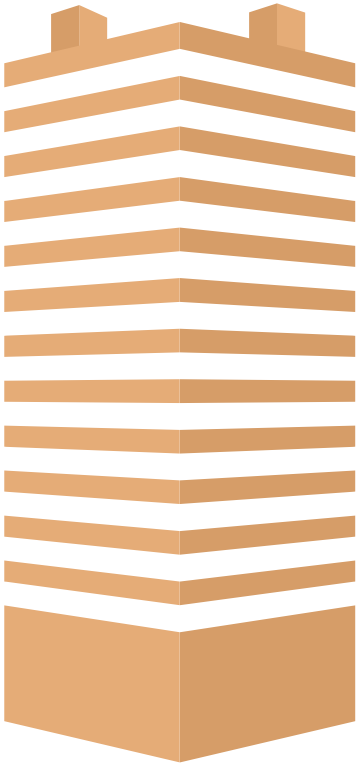
Supervisor

For ensuring application **fault tolerance**, CubDB process is supervised. Is located in **MySupervisor.ex**.

Logic

All business logic is located in **Betunfair.ex**.

DATABASE



CUBDB

As data storage system, we use **CubDB**. We prefer it because is **simpler** than Ecto, allows **concurrent** and **atomic** reads, and blocks concurrent writes.

Data structure

We store “users”, “markets” and “bets” in the **same table**, identifying them with the type and an unique id.

Data integrity

Using **transactions** and **snapshots**, database operations are atomic and consistent in concurrent queries.

DATABASE STRUCTURE

Identifier

Each value stored in database is identified as **{data_type,unique_id}**.

Ejs. {:user, "erktsc-1223-asd"} {:bet, "gaf-15341-ags"}

This allows to **filter** entries with **no additional cost** in a query. (Stream.filter())

DATABASE INTEGRITY

Transactions

Each function that requires a write in database is implemented with **CubDB.transaction()**, because some **gets** and **puts** must be done atomically.

Snapshots

List of pending backs or market bets are implemented with snapshots (**CubDB.select()**), reading multiple data atomically, and allowing other processes to make read queries concurrently.

```elixir
@spec market_freeze(binary) :: :ok | {:error, atom}
def market_freeze(id) do
  CubDB.transaction(Database, fn tx ->
    final_tx = update(tx, :market, id, :status, :frozen)
    {:commit, final_tx, :ok}
  end)
end

Updates an entry in database.
Returns new transaction with the operation.
"""
defp update(tx, type_entry, id, key, value) do
  case CubDB.Tx.fetch(tx, {type_entry, id}) do
    :error ->
      {:error, :not_found}

    {:ok, data} ->
      new_data = Map.put(data, key, value)
      CubDB.Tx.put(tx, {type_entry, id}, new_data)
  end
end
```

```elixir
# selects all active bets from market
list_active_bets =
  CubDB.select(Database)
  |> Stream.filter(fn {{type, _id}, _value} -> type == :bet end)
  |> Stream.filter(fn {{_type, _id}, value} -> value.market_id == id end)
  |> Stream.filter(fn {{_type, _id}, value} -> value.status == :active end)
  |> Stream.map(fn {{_type, _id}, value} -> value end)
  |> Enum.to_list()
```

Challenges faced

Problems

- **Concurrency problems**
- **Problems when deploying and stopping the system**
- **Emergence of defunct processes**

Solution

- Simplification of the system structure
- Removal of additional GenServers
- Clear :
  Stop -> Start ->Clear -> Stop

Testing

- Testing with **Mix**
- Number of tests: 52
- Test types : **Unit** and **functional** tests
- Tests per module: Bet_test, Market_test, User_test
- **Extra tests** for Fault tolerance

Benchmarking

Measurement value: test time

➔ **Betunfair.v1 (Additonal GenServers) -> low performance**

```
.....................................................
Finished in 2.9 seconds (0.00s async, 2.9s sync)
52 tests, 0 failures
```

➔ **Betunfair.v2 (Final implementation) -> high performance**

```
.....................................................
Finished in 2.1 seconds (0.00s async, 2.1s sync)
52 tests, 0 failures
```

Thanks!

Do you have any questions?