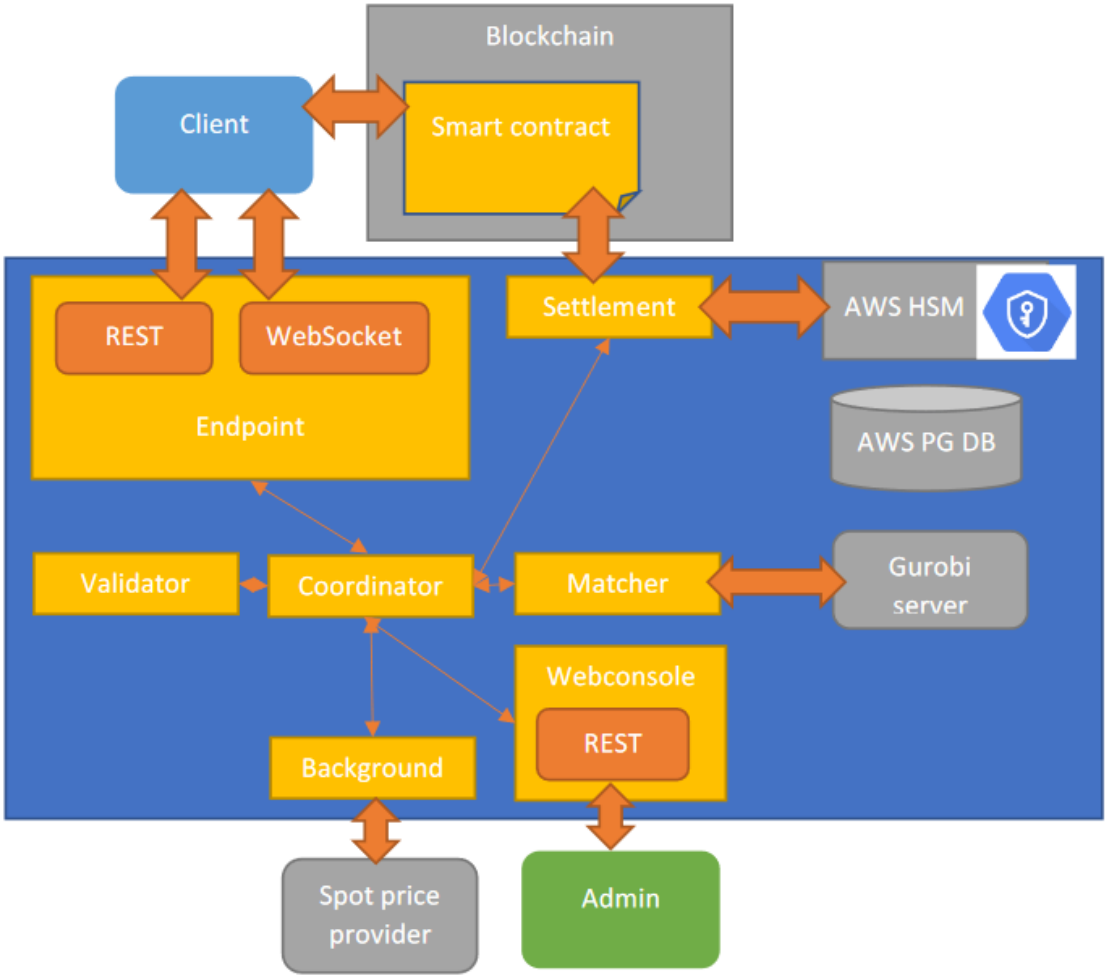
Backend



Notes

- Original project is https://github.com/NomismaTech/ccp-backend master branch
- Margin Loans are not implemented, this branch contains them: https://github.com/NomismaTech/ccp-backend/tree/trustful_with_redundancy

Testing

- You should have postgres 13 or above on localhost:5432 with user=postgres password=123456 and db=elektro (postgres should have privs for this db)

Components

Endpoint

- Provides REST interface to trade
- Provides WebSocket interface to observe orderbooks and order state

Background

- Digs spot prices from external resources
- Performs payoff calculations for expired contracts

Settlement

- Performs updates on the blockchain

Validator

- Calculates updates for new orders and order matches

Matcher

- Executes auctions (interacts with Gurobi)

Coordinator

- Gathers updates from Background, Settlement, Endpoints, Validators, Matchers
- Persists data into DB

Db tool

- Creates/Updates database schema (idempotent)

Webconsole

- REST interface to manage system object like currecncies, contracts, etc

DB objects

Reference prices

Matching prices. When contract is not matched yet initial price is there.

Spot prices

Spot prices loaded from external resource. Spot prices for settle calculation are also there

Fund locks

In the backend, for every currency client has 3 values:

- fundlock - amount that settlement monitor detected last time through the blockchain logs
- order value - amount that is locked with the orders submitted but not matched yet

- settle value - amount of the updates that are not yet performed on the blockchain side

Orders

Alive orders list - orderbook

Historical orders

Order state history

Contracts

Contracts to trade (products)

Positions

Client position in the contract. Negative - ASK. Positive - BID.

Lockups

We use collateral optimization. It also affects on payoff calculation. This object stores what amount of the clinent's funds was locked for specific expiry and currency pair. We use it when calculate payoff and re-calculate collateral.

Cash movements

Cashmovement object contains amount of funds to be moved and position update. Negative position value means ASK, positive - BID. Cash movement objects are are result of the matching calculations

Cash movement trackings

Objects to monitor status of the cash movement execution on the smart contract side

Edges

Stores datetime or number of the last performed and persisted operation

Logic

Spot prices

Background node continuously performs:

- Connect to the external resources like (Gemini, Coinbase, Binance) and loads spot prices for currency pairs in the system
- Saves spot prices into the database

Registration

- Client deposits some funds to the fundlock smart contract
- Settlement node monitors blockchain logs continuously
- Once deposit event is caught, monitor checks client registration. If client is not registered yet, it registers client (saves client data into DB)

New order

On the Endpoint:

- Client connects to endpoint
- Client authenticates through the nonce signing
- Client submits new order
- Endpoint validates order's structure and performs basic validations of the order
- Endpoint puts order into the queue

On the Background:

- Reads database (expired contracts, client positions, client lockups)
- Calculates updates to be performed for expired contracts
- Puts updates to the queue

On the Settlement:

- Reads database (cash movements to be settled, cash movement trackings)
- Submits updates to be performed to the smart contract
- Monitors blockchain events, prepares updates to be performed
- Puts updates to the queue

On the Coordinator, Validator, Matcher

- Coordinator gets updates from Background (prices at expiry and payoff cash movements)
- Coordinator gets updates from Settlement (analyzed range of the blocks from blockchain, ids of settled cash movements, balance updates for the clients)
- Coordinator gets updates from Endpoint[s] (new orders and order cancellations)
- Coordinator submits collected data to the Validator[s]
- Validator[s] validate orders, calculate updates to be performed and returns them to the Coordinator
- Coordinator submits new orders and orders to be cancelled to the Matcher[s]
- Matcher[s] update orderbook
- Coordinator persists updates into DB
- Coordinator sends updates to the Endpoint[s] (orders validaion results)
- Endpoint[s] notify subscribers with order state updates, orderbook updates

Auction

- Coordinator requests auction execution on the Matcher[s]
- Matcher[s] build model and solves it with Gurobi server
- Matcher[s] build matching results (executions)
- Matcher[s] return results to the Coordinator

- Coordinator submits matching results to the Validator[s]
- Validator[s] calculate updates to be performed and returns them to the Coordinator
- Coordinator persists updates into DB
- Coordinator sends updates to the Endpoint[s]
- Endpoint[s] notify subscribers with order state updates, orderbook updates, trade reports

Contract expiration

- All the contracts expire at 12:00 noon UTC.
- Contract becomes untradeable in 12 hours before expiration (safety period)
- At 12:10 approximately Background node requests spot price for the 12:00 noon UTC
- Background node calculates payoff for the clients
- Background node puts updates to the queue

Smartcontract interaction

- Payoff and matching processes on the backend generate cash movement objects.
- Cash movements are sent to the smartcontract.
- Smartcontract updates client's balance according cash movements and stores positions.

Local run

- Build sources with: `mvn clean install -DskipTests`
- Install Docker
- Go to `testing/docker-local-env`
- Put there Gurobi linux x64 client distributive and license file
- Run: `docker compose down`
- Run: `docker compose up`
- Http endpoint port: `8078`
- Ws endpoint port: `8079`
- Private key #1 with some funds depositted:
  `0x0000000000000000000000000000000000000000000000000000000000012001`
- Private key #2 with some funds depositted:
  `0x0000000000000000000000000000000000000000000000000000000000012002`
- Private key #3 with some funds not depositted:
  `0x0000000000000000000000000000000000000000000000000000000000012003`
- Private key #4 with some funds not depositted:
  `0x0000000000000000000000000000000000000000000000000000000000012004`

Matching engine

- There was a bug in Matcher's code, this is why sometimes Gurobi said the model is unfeasible. It seems that I fixed it, but have not checked my fix well, because Girobi server was turned off. See `old-way-matcher` branch to find matcher's code without changes.
- After the bug fix solution range became wider, so 10K orders solves slower than 30sec. It looks that something in range 30sec-2min.
- I wasn't able to await solution of 15K orders matching, it seems like something that takes too many time. But maybe if we buy more processors for Gurobi it will solve it. Also

there are some parameters that are tuning Gurobi solvers inside, unfortunately i had no chance play with these parameters.
- Once we have Gurobi, we need to update MatchingConfig in tests and in properties file
- Update CircleCI config to not skip MIP tests