Documentation

Team - SoBers
Aman Kumar Kashyap
Aman Rojjha
VJS Pranavasri

Code Structure

```
.
├── build                              - Build targets for our codebase
│   └── contracts
│       ├── Market.json
│       └── Migrations.json
├── constants.js                       - Generic constants
├── contracts                          - Solidity contracts
│   ├── Market.sol                       - Smart contract for auction (using
optimal resource allocation)
│   └── Migrations.sol
├── Documentation.md
├── migrations
│   ├── 1_initial_migration.js
│   └── 2_deploy_market.js
├── README.md                          - Instructions to run the codebase
├── test                               - Test cases for
│
- Optimal resource allocation;
│
- Profit optimization;
│
- Validity of product and ownership
└── truffle-config.js
```

Logical Flow for Auction

- **Market** is a solidity-contract responsible for implementing the required conditions and auction system as specified. Its main functionalities (and behavior) are:
    1. `isSupportedSupplier()`: checks if manufacturer can buy from supplier as specified.
    2. `assignSupplier()`: Assigns the *calling id* (provided in the environment) to the *supplier* provided in the parameter.
    3. `assignManufacturer()`: Assigns the *calling id* (provided in the environment) to the *manufacturer* provided in the parameter.
    4. `updateSupply()`: This updates the *supplier's* supply amount and price.
    5. `manufactureCars()`: Creates a new *car* (a virtual asset) made from a given unique *car-body* and *wheel* id.
    6. `getSupply()`: This returns *supply count* and *cost* for a provided supplier.\
    7. `secretBid()`:
        1. Only manufacturer can call this function.
        2. We send in a hashed bid which can be later verified in bid.
    8. `bid()`:
        1. Only *manufacturers* can call this function.
        2. A given *manufacturer* bids some *quantity* of resources for a given *supplier* on a specified *bidding amount*.
        3. We generate a hash of these values and compare it with previously stored hashed value to verify the bid.
    9. `auction()`:
        1. If there exists any valid *manufacturer* who didn't bid yet, it returns the control flow.
        2. When all the *manufacturers* have bid their respective bids, *auction* is held and the supplies are provided according to *resource optimal distribution* as specified in the next sections.
    10. `manufactureCars()`: Manufactures all the possible cars based on the supplies the calingf manufacturer has. It takes the price of the car as the input.
    11. `buyCar()`: Takes the price of the car, and transfers the car object from manufacturer to customer, also adds ownership to the car.

Flow of the supply chain

- All the *manufacturers* bid their respective amounts.
- *Auction* is invoked (with the corresponding checks).
- The *resource optimal allocation* algorithm runs and allocates *supplies* to respective *manufacturers*. If some *supplies* are still left, we consider a *profit optimal allocation*

in respect to supplier to allot the remaining supplies.

Validity of the product and its material

- To check the validity of *supplier* for a given supply, invoke `getSupplierFromSupply()`.
- To check the validity of manufacturer who owns the *supply*, invoke `getManufacturerFromSupply()`.

Optimal Supply

Optimal Resource Allocation

> A supply-chain follows optimal distribution of materials if the quanitity of material which is kept on the shelf is minimized, i.e. the amount of resource utilized is maximum.

Consider some supplier resources for *Vedanta*, *MRF* and *CEAT* respecitvely.
We first focus on *optimally alllocating resources* so as to maximize the total number of generated cars.
We do this by calculating the maximum possible cars that can be generated by each manufacturer based on the bid and available resources.
Then we send the supply required to generate the manufacturer with maximum possible cars.
Now we send the supply required to generate possible cars of the other manufacturer (as many as possible depending on supply available)
If there is still supply remaining, we look at the remaining bids and run auction based on maximising the profits for a supplier.

Test cases

No.	Goal	Test Case	Explanation
1	Flow of Supply Chain	1	The complete requested workflow from, update supply, secret bid, bid and auction followed by verification and purchase of car by customer
2	Validity of the product and material by buyer	1	Supplier/Manufacturer can can check the ownership of a given supply
3	Secrecy of bids until all bids are made	1	Bid is done in two steps, first a hashed bid and then a reveal bid to reveal the values
4	Optimal followed by Profit Optimised	1	Initially Optimal Resource allocation is done then the remaining supply is sold based on profit optimsation
5	Optimal Resource Allocation	2, 3	One of the two manufacturers bid will exhaust out the supplier, and the other manufacturer won't be able to get total supply bid for

Escrow Payments

- We consider the address of *owner* of the contract as the *address* for **Escrow payments**.
- At the end of reveal bid, the total price the manufacturer has bid for is sent to the escrow.
- At the end of all reveal bids, escrow holds the amount of each manufacturer.
- When auction completes, ecsrow transfers the amount to manufacturers/suppliers respectively based on the supply they got based on the result of the auction.