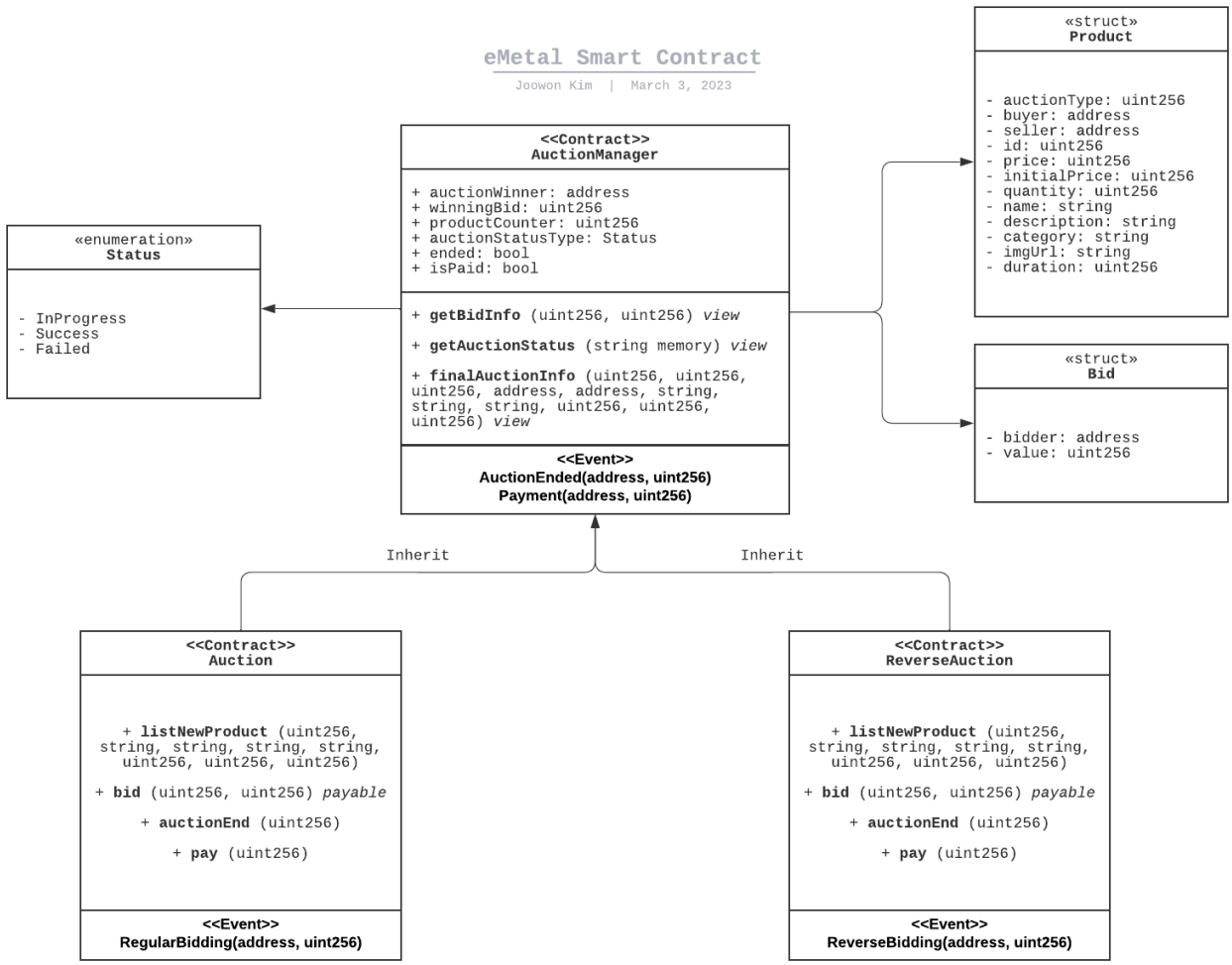
Trillions_eMetal_smartcontract

경매/역경매 Solidity Smart Contract

1. 개발환경

- **IDE** : VSCode
- **테스트환경**: Remix IDE
- `solidity 0.8.17`

2. Smart Contract 관계도



3. FLOW

- 경매 및 역경매를 만들면, 새로운 스마트 컨트랙트가 생성되며 경매 및 역경매 정보가 스마트 컨트랙트에 저장된다.
- 생성된 경매에 유저가 입찰을 하고, 이 입찰 정보가 저장된다.
- 모든 입찰이 완료되고 경매 시간이 종료되면, 최종 경매 정보를 조회할 수 있다.
- 최종 경매 정보에 판매자, 구매자 및 최종 지불해야할 가격이 명시되어 있고, 구매자가 이에 맞게 판매자에게 지불한다.

---

4. 스마트 컨트랙트 설명

1. **AuctionManager.sol**

- 경매 및 역경매에 관한 정보를 조회할 수 있는 스마트 컨트랙트

1. `getBidInfo(uint256, uint256) public view`
    - 입찰 정보를 확인할 수 있다.
    - 경매 id, 입찰자 인덱스 순서로 입력하여 해당 경매건에 대한 입찰자의 주소 및 입찰가를 확인할 수 있다.
2. `getAuctionStatus(string memory) public view`
    - 경매 상태를 확인할 수 있다.
    - 경매가 진행중이면 *In Progress*, 경매가 낙찰되면 *Success*, 경매가 실패하면 *Failed*로 표기된다.
3. `finalAuctionInfo(uint256 _id) public view`
    - 최종 경매 정보를 확인할 수 있다.
    - 경매 id로 조회하며 완료된 경매의 *id*, *경매종류(일반경매, 역경매)*, *시간*, *판매자*, *구매자*, *경매아이템 이름*, *카테고리*, *설명*, *경매시작가*, *낙찰가*, *수량*, *최종금액(낙찰가 x 수량)* 이 표시된다.

2. **Auction.sol**

- 일반경매를 생성, 입찰, 종료 및 지불을 할 수 있는 스마트 컨트랙트

1. `listNewProduct(uint256, string memory, string memory, string memory, string memory, uint256, uint256, uint256)`
    - 일반경매를 생성한다.
    - *경매종류*, *경매아이템 이름*, *설명*, *카테고리*, *이미지*, *경매시작가*, *수량*, *시간* - 을 입력하여 생성한다.
    - 일반 경매이므로 auctionType은 1로 설정한다.
    - 경매 시간은 미래로 설정해야한다. *(단위: 초)*
    - 경매가 생성되면 **auctionStatusType**은 *In Progress*로 바뀐다.
2. `bid(uint256 _id, uint256 _price) public payable`
    - 입찰자가 입찰을 한다.
    - 해당 경매 id와 입찰가를 입력하여 입찰을 시도한다.
    - 다른 입찰자의 입찰가가 기존 입찰가보다 높으면 **winningBid**가 더 높은 입찰가로 변경되며, **auctionWinner**도 더 높은 입찰가를 작성한 구매자로 변경된다.
    - 입찰을 하면, 해당 입찰자와 입찰가가 기록되며, 이 입찰 기록은 **getAuctionStatus**함수에 경매 id와 입찰자 index를 각각 입력하여 조회할 수 있다.
    - 입찰시, 입찰가는 경매 시작가보다 높아야 한다.

- 입찰시, 입찰가는 구매자가 가지고 있는 금액보다 낮은 금액이어야한다.
- 입찰시, 판매자(경매생성자)는 자신의 경매에 입찰을 할 수 없다.
- 입찰시, 해당경매가 종료되지 않은 상태여야 한다.

3. `auctionEnd(uint256 _id) public`

- 경매가 종료된다.
- 경매 생성시 설정했던 경매시간이 끝나면, 경매는 자동 종료된다.
- 경매시간이 다 끝나지 않았을지라도 판매자(경매생성자)만이 경매를 강제 종료시킬 수 있다.
- 이미 종료된 경매는 종료시킬 수 없다.
- 낙찰자가 존재할 경우, **auctionStatusType**은 *Success*로 표기되며, 낙찰자가 존재하지 않는 경우, **auctionStatusType**은 *Failed*로 표기된다.

4. `pay(uint256 _id) public payable`

- 구매자가 판매자에게 최종금액을 지불한다.
- 최종금액은 *낙찰가 x 수량*이다.
- 경매가 종료되지 않으면, 판매자에게 송금을 할 수 없다.
- 만약 판매자 주소가 *0*이면 판매자 주소가 설정되지 않은 경우이므로 송금을 할 수 없다.
- 송금시, 구매자의 소지금이 지불금액보다 많아야한다.
- 송금금액은 최종금액 (낙찰가 x 수량)으로 이 정확한 액수여야만 송금이 가능하다.

3. **ReverseAuction.sol**

- 역경매를 생성, 입찰, 종료 및 지불을 할 수 있는 스마트 컨트랙트

1. `listNewProduct(uint256, string memory, string memory, string memory, string memory, uint256, uint256, uint256)`

- 역경매를 생성한다.
- *경매종류*, *경매아이템 이름*, *설명*, *카테고리*, *이미지*, *경매시작가*, *수량*, *시간* - 을 입력하여 생성한다.
- 역경매이므로 auctionType은 2로 설정한다.
- 경매 시간은 미래로 설정해야한다. *(단위: 초)*
- 경매가 생성되면 **auctionStatusType**은 *In Progress*로 바뀐다.

2. `bid(uint256 _id, uint256 _price) public payable`

- 입찰자(판매자)가 입찰(오퍼)을 한다.
- 해당 경매 id와 입찰가를 입력하여 입찰을 시도한다.
- 다른 입찰자의 입찰가가 기존 입찰가보다 낮으면 **winningBid**가 더 낮은 입찰가로 변경되며, **auctionWinner**도 더 낮은 입찰가를 작성한 판매자로 변경된다.
- 입찰을 하면, 해당 입찰자와 입찰가가 기록되며, 이 입찰 기록은 **getAuctionStatus**함수에 경매 id와 입찰자 index를 각각 입력하여 조회할 수 있다.
- 입찰시, 입찰가는 경매 시작가보다 낮아야 한다.
- 입찰시, 구매자(경매생성자)는 자신의 경매에 입찰을 할 수 없다.
- 입찰시, 해당경매가 종료되지 않은 상태여야 한다.

3. `auctionEnd(uint256 _id) public`

- 경매가 종료된다.
- 경매 생성시 설정했던 경매시간이 끝나면, 경매는 자동 종료된다.

- 경매시간이 다 끝나지 않았을지라도 구매자(경매생성자)만이 경매를 강제 종료시킬 수 있다.
- 이미 종료된 경매는 종료시킬 수 없다.
- 낙찰자가 존재할 경우, **auctionStatusType**은 *Success*로 표기되며, 낙찰자가 존재하지 않는 경우, **auctionStatusType**은 *Failed*로 표기된다.

4. `pay(uint256 _id) public payable`

    - 구매자가 판매자에게 최종금액을 지불한다.
    - 최종금액은 *낙찰가 x 수량*이다.
    - 경매가 종료되지 않으면, 판매자에게 송금을 할 수 없다.
    - 만약 판매자 주소가 *0*이면 판매자 주소가 설정되지 않은 경우이므로 송금을 할 수 없다.
    - 송금시, 구매자의 소지금이 지불금액보다 많아야한다.
    - 송금금액은 최종금액 (낙찰가 x 수량)으로 이 정확한 액수여야만 송금이 가능하다.

---

5. 스마트 컨트랙트 실행

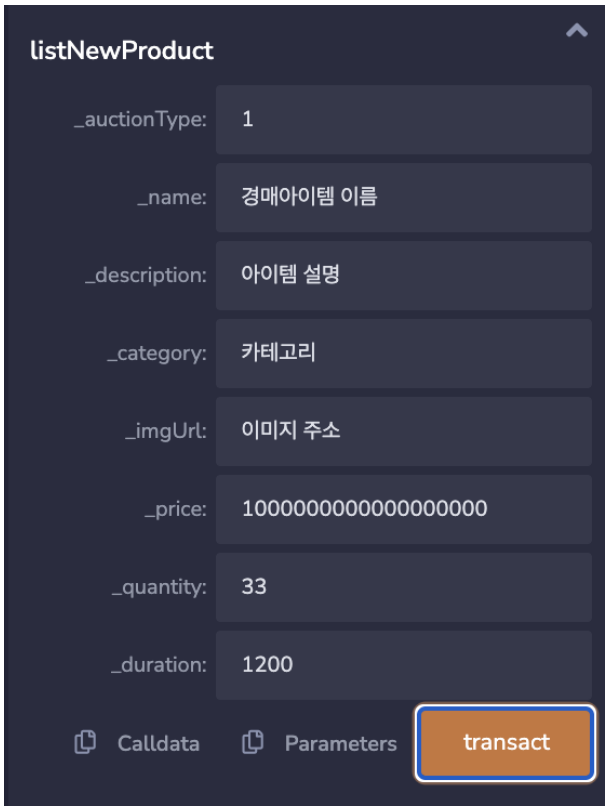
- 스마트 컨트랙트 실행시 Remix IDE에서 진행하였으며, 로컬에서 Ganache 사용하여 실행도 가능

1. `Remix IDE`에서 *Metamask* 혹은 *Remix VM(Merge, London, Berlin)* 중 선택한다.

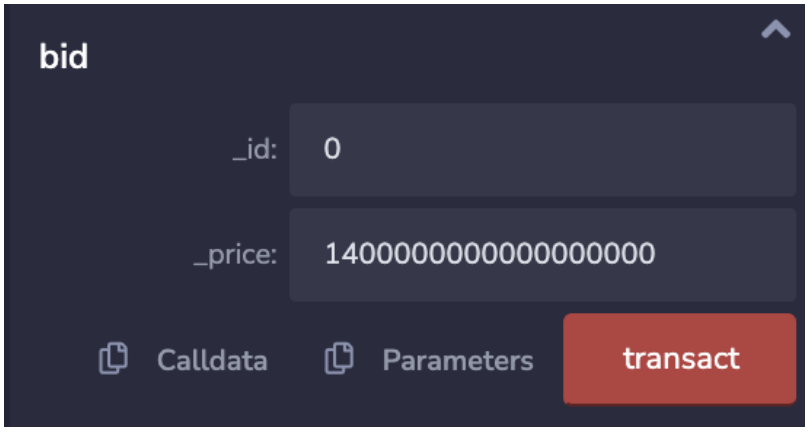
    - *Metamask*의 경우 메타마스크 계정이 있는경우에만 테스트 가능하며, Testnet을 사용하여 etherscan에서 모든 거래 정보 확인 가능
    - *Remix VM*을 사용할 경우, 샘플 주소 15개를 사용하여 테스트 가능하며, 아래 터미널에 모든 결과값이 표시된다.

2. **Auction.sol / ReverseAuction.sol**을 deploy 한다.
   배포된 스마트 컨트랙트는 아래 *Deployed Contract*에 표시되며 이를 확장하여 각각의 함수들을 실행시킬 수 있다.

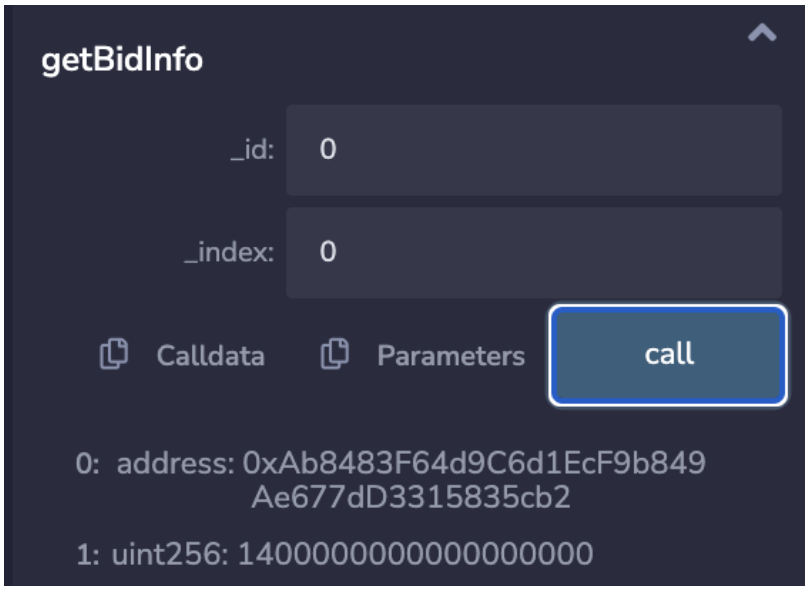
3. **listNewProduct** 함수를 확장시켜 해당 칸에 알맞는 값을 입력하여 경매를 생성한다.
   (*price* 입력시 단위는 **WEI**이며 **1 ETH = 1000000000000000000 WEI**)



4. 다른 Account를 선택하고 **bid** 함수를 확장시켜 *경매 id*, *입찰가*를 입력하여 입찰을 한다.
   경매생성을 한 Account를 제외한 모든 Account로 몇명이든 입찰할 수 있으며 Account 하나당 한번 입찰할 수 있다.

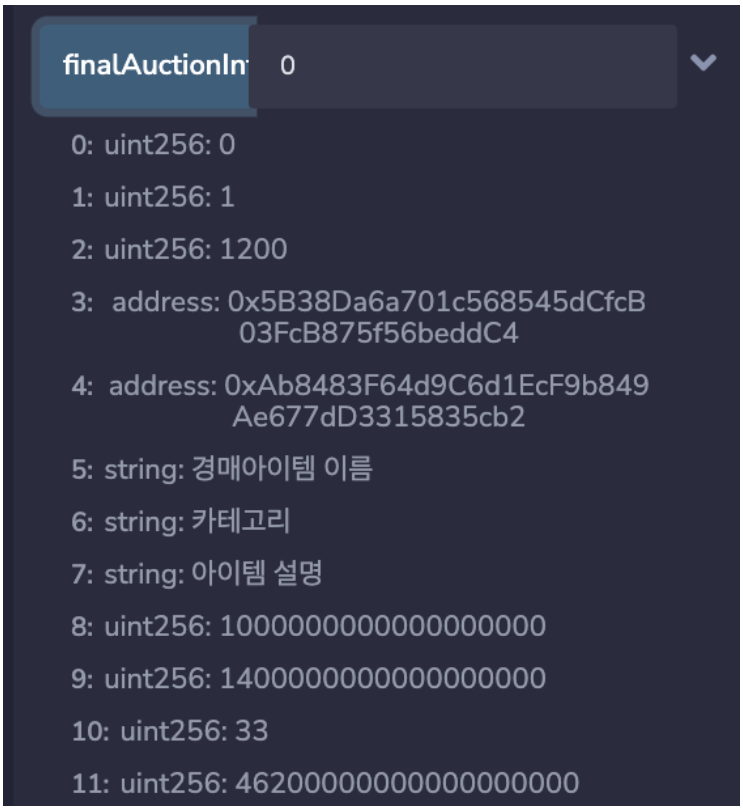


5. 위 **bid** 함수로 입찰한 모든 입찰 기록은 **getBidInfo**를 확장하여 *경매 id*, *입찰자 index* 를 순서대로 입력하여 조회할 수 있다.
   또한, 입찰할 때마다, **auctionWinner** 와 **winningBid** 버튼을 눌러 현재 기준 낙찰자와 낙찰금을 확인할 수 있다.



6. 판매자(입찰생성자) Account로 전환하고 **auctionEnd** 함수에 *경매 id*를 입력하여 실행시켜 경매를 종료시킨다.

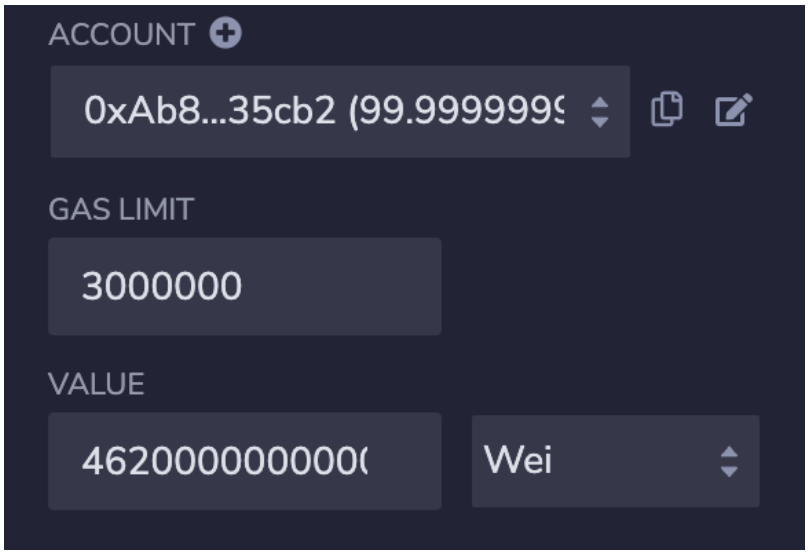
7. 경매가 종료되면 **finalAuctionInfo** 함수에 *경매 id*를 입력하여 종료된 경매의 최종 정보를 확인할 수 있다.



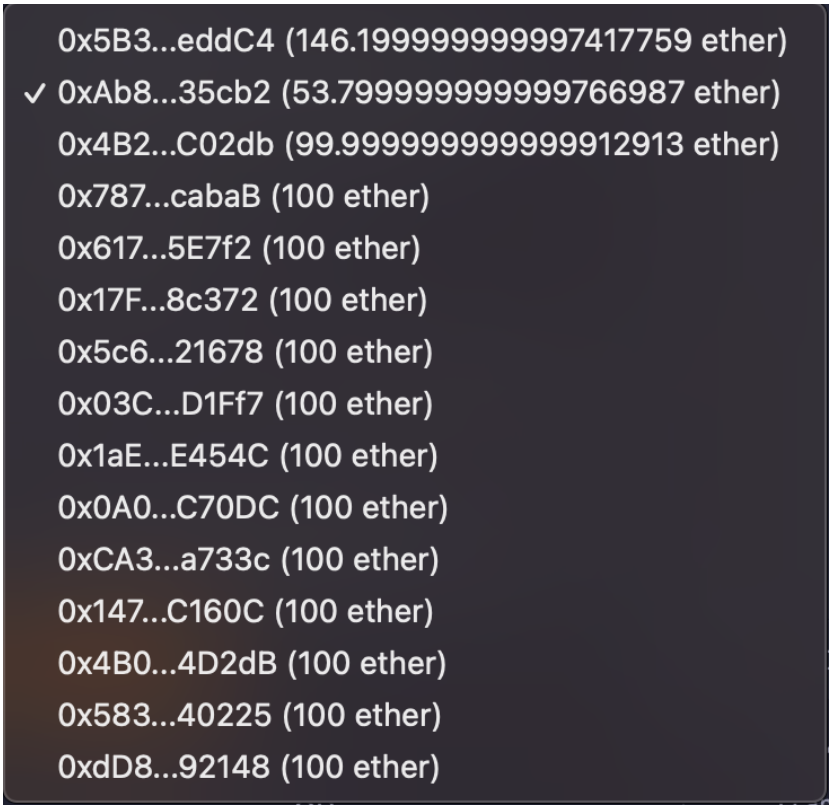
8. 위 예시의 4번에 해당하는 주소가 최종 낙찰자 주소이므로, 이 Account로 전환한다.

9. **pay**함수를 실행시키기 전, 최종금액(낙찰가 x 수량)의 금액을 **finalAuctionInfo) 맨 아래 항목에서 확인하고, 위로 스크롤 하여 VALUE 값에 입력한다. (단위는 그대로 wei)

- *VALUE* 값 입력후 **pay**함수에 *경매 id*를 입력하고 실행시키면 송금이 완료된다.
- **isPaid** 버튼을 눌러보면 *true* 값이 나온다. (송금전에는 *false*로 표기)



10. 위로 스크롤하여 Account를 눌러보면, 낙찰자 계정에서 판매자 계정으로 최종 금액만큼 송금이 된것을 확인할 수 있다.



11. 역경매의 경우, 전반적인 모든 과정이 같으나 구매자가 **listNewProduct** 함수로 경매를 생성하며, 입찰자들은 판매자이며, 가장 낮은 금액을 제시한 판매자가 최종 낙찰자로 선정된다.
auctionEnd 함수를 호출하는것도 구매자(경매생성자), 지불하는것도 구매자(경매생성자)이므로 **pay** 함수를 실행시킬때 계정을 반드시 확인하고 실행시켜주어야한다.

---

6. Truffle Test

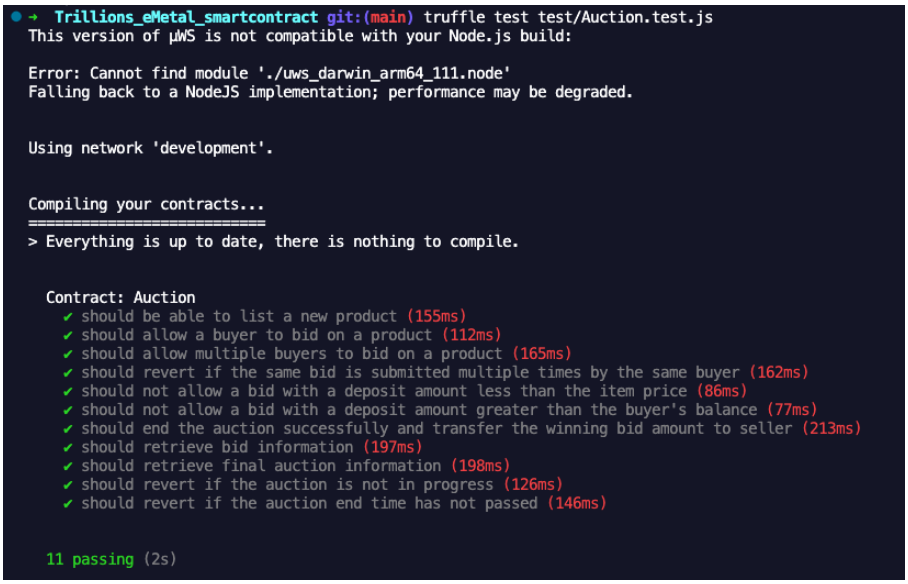
- 경매 및 역경매 스마트 컨트랙트에 대한 *Unit Test*로 **Mocha Framework**를 사용하여 실시

테스트 방법

- 프로젝트 *root* 폴더 진입
- **Ganache** 프로그램 작동
- `truffle test test/Auction.test.js` 명령어 입력하여 `Auction.sol`에 대한 Unit Test 실시
- `truffle test test/ReverseAuction.test.js` 명령어 입력하여 `ReverseAuction.sol`에 대한 Unit Test 실시

테스트 결과

1. **Auction.test.js**
   - *Auction.sol* 스마트 컨트랙트 파일에 총 `11개` 항목의 테스트 실시



2. **ReverseAuction.test.js**
   - *ReverseAuction.sol* 스마트 컨트랙트 파일에 총 `10개` 항목의 테스트 실시