네트워크최신 기술 프로젝트 ‹랜덤 뽑기›

20191581 김하연

랜덤 뽑기 프로젝트

이 프로젝트는 뽑기 기계에서 아이디어를 얻어 구현했습니다.

사용자는 해당 스마트 계약에서 일정한 금액을 지불함으로써 여러 상품들 중에서 랜덤한 상품을 뽑을 수 있습니다.

랜덤 뽑기 제약사항

ganache 이용

뽑기를 하기 위해 넣어야 하는 비용 : 10000000 Wei

상품의 최소 가격: 8000000 Wei

상품을 넣을 수 있는 건 오직 스마트 계약을 처음 배포한 사람이며 관리자로 명명한다.

관리자도 랜덤 뽑기를 할 수 있으며 관리자가 아닌 뽑기 사용자는 고객으로 명명한다.

사용자들이 투입한 돈(계약의 balance)는 오직 관리자만 알 수 있으며 뽑기가 끝날 시(kill) 관리자가 받을 수 있다.

관리자의 이익과 고객의 흥미 유발을 위해서 고객들에게는 상품의 총 종류와 가장 좋은 상품(최고가의 상품), 지금까지 사용자들의 이익만 노출한다.

관리자는 상품이 무제한으로 준비되어있다는 가정 하에 상품을 추가할 수 있다.

랜덤 뽑기 시 준비된 상품들 중에서 랜덤한 상품을 사용자에게 노출한다.

스마트 계약 구현(김하연.sol) - 변수 지정 및 초기화

```
    struct Product {
        string productName;
        uint price;
        string imgLink;
    }
    address public owner;        // 소유자(=총 판매자)
    int public totalParticipantInterest; // 참여자들의 총 이익
    uint public numParticipants; // 참여자 수
    uint public numProducts; // 총 상품 수
    uint public randomIndex; // 랜덤한 인덱스
    Product public highPriceProduct; // 상품 중 제일 비싼 가격
    mapping (uint => Product) public products; // 상품 리스트
```

```
    constructor () {
        owner = msg.sender;
        numParticipants = 0;
        numProducts = 0;
        totalParticipantInterest = 0;
        highPriceProduct.productName = "nothing";
        highPriceProduct.price = 0;
        highPriceProduct.imgLink = "https://dummyimage.com/600x400/ffffff/4c56e0&text=nothing";
    }
```

뽑기 안의 상품은 Product 구조체로 관리.

상품 목록은 매핑 타입을 이용.

스마트 계약 구현 - 관리자 여부 조회

```
    function checkOwner() view public
returns(bool) {
        if (msg.sender == owner) return
true;
        else return false;
    }
```

웹페이지 상에서 관리자와 고객의 분리를 위해서 public 접근 가능하고 bool 타입을 리턴하도록 함.

스마트 계약 구현 - 상품 추가

```
modifier onlyOwner () {
    require(msg.sender == owner);
    _;
}
```

```
function addProduct( string memory productName,
uint price,  string memory imgLink) public
onlyOwner returns (bool) {
        require(price >= 8000000);

        Product storage prd =
products[numProducts++];
        prd.productName = productName;
        prd.price = price;
        prd.imgLink = imgLink;
        if (highPriceProduct.price < price) {
            highPriceProduct.price = price;
            highPriceProduct.productName =
productName;
            highPriceProduct.imgLink = imgLink;
        }
        return true;
    }
```

관리자만 가능하도록 modifier 지정함.

상품의 최소가치를 유지하기 위해서 require 이용함.

상품 추가 시 상품 목록, 총 상품 수, 최고가 상품을 업데이트함.

스마트 계약 구현 - 랜덤 뽑기

```
 function applyRandom() public payable {
        // 랜덤한 상품 출력
        require(msg.value == 10000000);
        require(numProducts > 0);

        randomIndex =
uint(keccak256(abi.encodePacked(msg.sender,
block.timestamp))) % numProducts;
        numParticipants++;
        totalParticipantInterest +=
int(uint(products[randomIndex].price)) - 10000000;
    }
```

payable 지정

뽑기 시 드는 비용 유지와 상품이 있을 때만 뽑기가 가능하도록 require 이용함.

keccak256을 사용해서 랜덤한 수를 뽑고 총 상품수로 모듈러 연산을 해 랜덤한 상품 번호를 할당하고 참여자 수 증가와 참여자들의 비용 대비 상품 이익을 계산함

웹사이트 상에서 랜덤한 상품 번호를 이용해서 사용자에게 보여줄 예정

스마트 계약 구현 - 잔액 조회, 끝내기

```
modifier onlyOwner () {
    require(msg.sender == owner);
    _;
}
```

```
function getBalance() view public
onlyOwner returns(uint) {
    return address(this).balance;
}
```

```
function kill() public onlyOwner {
    selfdestruct(payable(owner));
}
```

관리자만 가능하도록 modifier 지정함.

웹 서버 구성(김하연.js)

```
"use strict";
var http = require("http");
var fs = require("fs");
var url = require("url");
async function gateWayPage(req, res) {
 var fname = "." + decodeURIComponent(url.parse(req.url).pathname);

 fs.readFile(fname, async function (err, data) {
   if (err) {
     res.writeHead(404, { "Content-Type": "text/html;charset=utf-8" });
     return res.end("404 Not Found");
   }
   res.writeHead(200, { "Content-Type": "text/html;charset=utf-8" });
   res.write(data);
   return res.end();
 });
}
http.createServer(gateWayPage).listen(8080);
```

node 이용

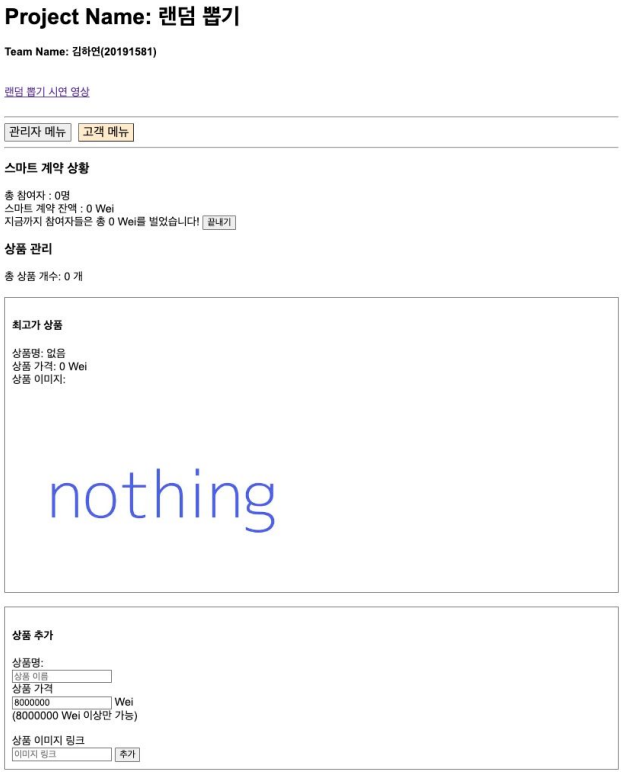
한글 파일명(김하연.html)을 제대로 읽어오기 위해서 decodeURIComponent 필수

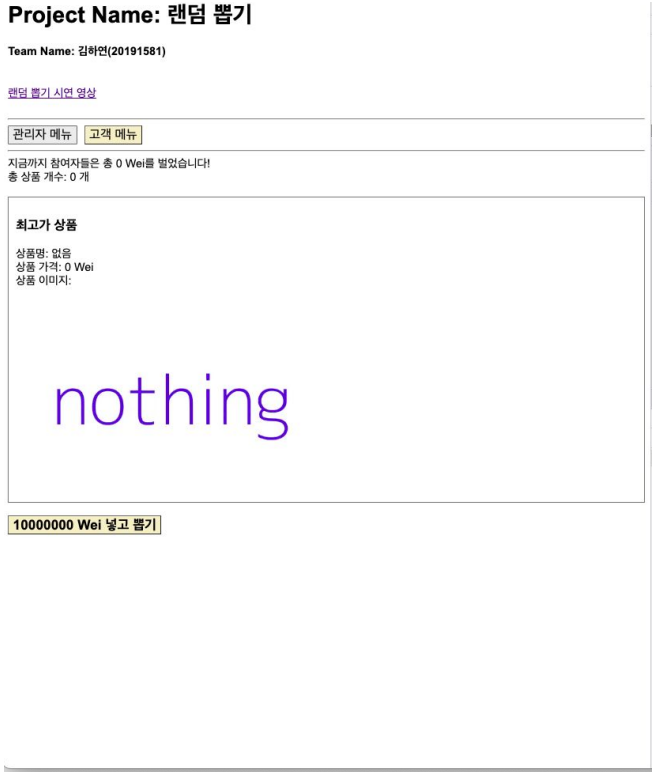
html 파일에 한글을 썼기 때문에 charset=utf-8 필수

node 명령어 사용 후 http://localhost:8080/김하연.html 로 접근 가능

웹 클라이언트 설명(김하연.html)

web3 이용



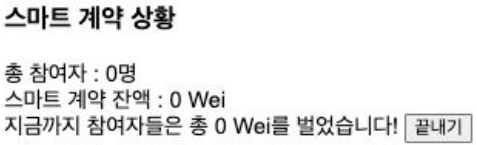


관리자 메뉴 화면과 고객 메뉴 화면으로 분리

checkOwner를 사용해 고객이 관리자 메뉴에 접근 불가 처리

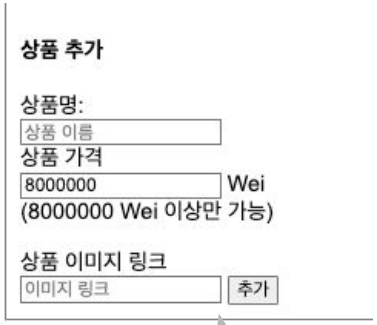
웹클라이언트 설명 - 관리자 화면

관리자 화면은 크게 스마트 계약 상황과 상품 관리로 분류되며
상품 관리에서는 기본적인 상품 조회와 상품 추가를 할 수
있다.



numParticipants, getBalance, totalParticipantInterest, kill 사용

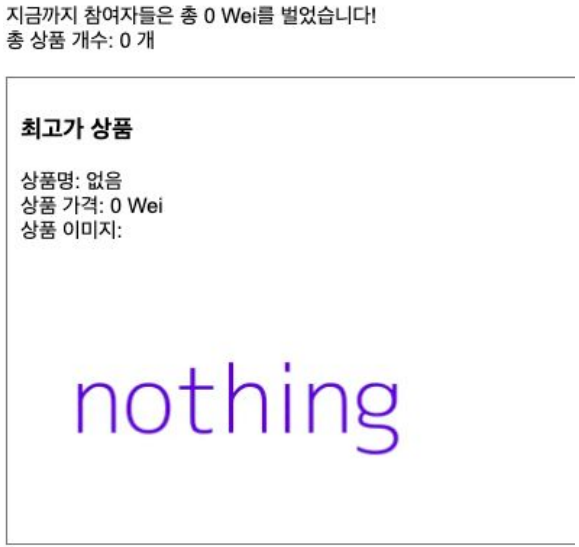
numProducts, highPriceProduct 조회



addProduct 사용

웹클라이언트 설명 - 고객 화면

고객 화면은 뽑기 욕구를 자극하기 위한 현재 상태와 뽑기 부분이 있다.



numParticipants, numProducts, totalParticipantInterest 조회

10000000 Wei 넣고 뽑기

applyRandom 사용

웹클라이언트 설명 - 상품 추가

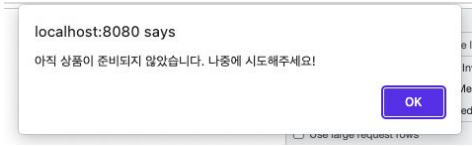
addProduct 성공 시 관리자, 사용자 메뉴의 상품과 관련된 정보(최고가 상품 등)가 화면에서 업데이트된다.


```
async function clickAddProduct() {
       // … 생략
        try {
          let data = await
randomDrawingContractInstance.methods
            .addProduct(
              productNameInput.value,
              parseInt(priceInput.value),
              imgLinkInput.value
            )
            .send({ from: accounts[0] });
          getContractInfo();
          alert("상품 추가했습니다.");
        } catch (err) {
          alert(err);
        }
      }
```

웹클라이언트 설명 - 랜덤 뽑기

numProducts를 조회해서 상품이 없을 때는 alert로 접근을 막고 상품이 있다면 applyRandom을 실행시킨다.

applyRandom 이 정상적으로 실행되면 뽑은 상품을 보여주고 관련된 모든 정보(참여자 수익 등)를 화면에서 업데이트한다.




```
async function clickApplyRandom() {
      try {
        let numProducts = await randomDrawingContractInstance.methods
          .numProducts()
          .call();
        if (!numProducts || parseInt(numProducts) === 0) {
          alert("아직 상품이 준비되지 않았습니다. 나중에 시도해주세요!");
          return;
        } else {
          // 생략
          await randomDrawingContractInstance.methods
            .applyRandom()
            .send({ from: accounts[0], value: 10000000 });

          let randomProductIdx = await randomDrawingContractInstance.methods
            .randomIndex()
            .call();
          let randomProduct = await randomDrawingContractInstance.methods
            .products(randomProductIdx)
            .call();
                // 생략
          getContractInfo();
          alert("뽑았습니다! 확인해보세요~!");
```

시연 영상 링크

https://youtu.be/fHzbyHtxT4E