블록체인
스마트컨트랙트

서강대학교
SOGANG UNIVERSITY

전자투표 Dapp

서강대학교 SOGANG UNIVERSITY

- **실행 과정**

1. 스마트 컨트랙트 작성(Ballot.sol 수정)
2. 가나쉬 실행: 테스트 블록체인 네트워크
3. 스마트 컨트랙트 컴파일&배포(truffle-config.js: 가나쉬 배포 설정)
   - cd ballot-contract
   - truffle migrate --reset
4. 웹서버(node.js) 실행
   - cd ballot-app
   - npm install
   - npm start

실행 과정

5. 웹브라우저(메타마스크) 실행
   - 브라우저에서 localhost:3000 접속
   - 메타마스크를 가나쉬 네트워크에 연결
   - 가나쉬 계정 가져오기(3개): 편의상 Account2, Account3, Account4 라고 함
     - Account2: chairperson
     - Account3: 투표자1
     - Account4: 투표자2
   - 가나쉬 계정(3개) localhost:3000 에 연결

실행 과정

6. 테스트

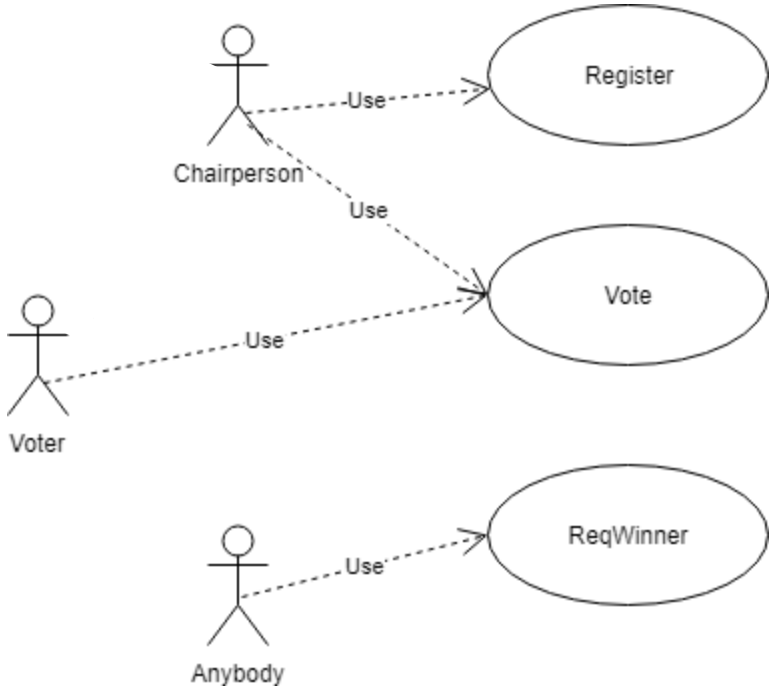
- Account2가 Account2, Account3, Account4 등록
- Account2가 4개의 제안 중 하나에 투표
- Account3가 4개의 제안 중 하나에 투표
- Account4가 4개의 제안 중 하나에 투표
- Declare Winner 버튼 클릭하여 우승자 확인(모든 계정 가능)

목차

- **스마트컨트랙트 예제**
  - 전자투표 스마트컨트랙트
    - 투표단계 추가
  - 호텔방 예약 스마트 컨트랙트
  - lottery 스마트 컨트랙트

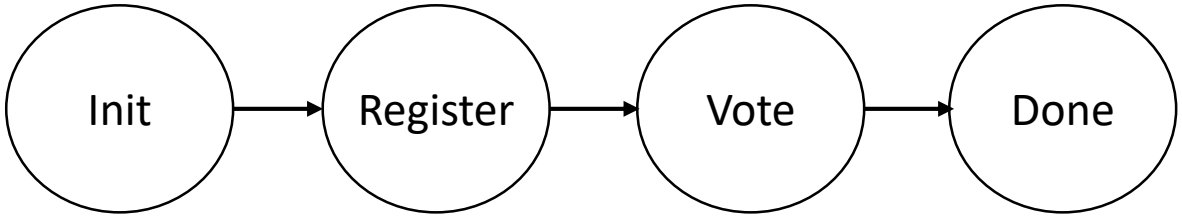
전자투표 스마트컨트랙트

- **Problem Statement**
  - 다수의 제안 가운데 하나에만 투표한다
  - 의장은 투표할 수 있는 사람을 등록하고(한 계정은 한번만 등록)
  - 등록된 사람만이 제안된 선택지 중의 하나에 오직 한번만 투표한다
  - 의장의 표는 가중치를 두어 두 표로 계산한다
  - 누구나 투표 결과를 확인할 수 있다



전자투표 스마트컨트랙트

- **Problem Statement**
  - 투표 단계 추가
    - 투표 과정을 위한 함수는 특정한 순서에 따라 실행한다
    - 등록은 투표전에 마쳐야 한다
    - 투표는 오직 정해진 기간만 가능하다
    - 투표가 종료되어야만 승자를 판단할 수 있다
    - 초기(Init), 등록(Register), 투표(Vote), 집계(Done) 단계로 구분



전자투표 스마트컨트랙트

- 데이터

```
struct Voter {
    uint weight;
    bool voted;
    uint vote;
}
struct Proposal {
    uint voteCount;
}

address chairperson;
mapping(address => Voter) public voters;
Proposal[] public proposals;

enum Phase {Init, Regs, Vote, Done}
Phase public currentPhase = Phase.Init;
```

투표자 상세 정보 (가중치, 투표 여부, 투표값)

제안 상세 정보(투표수)

투표자 주소를 투표자 상세 정보로 매핑

제안들을 담는 배열

현재 단계를 나타내는 변수

전자투표 스마트컨트랙트

서강대학교 SOGANG UNIVERSITY

- 함수
  - advancePhase()
    - 의장만 호출하도록 제한
    - 다음 단계로 변경

전자투표 스마트컨트랙트

- **함수**
  - 각 함수들에 단계가 맞는지 확인하는 코드 추가, 아니면 트랜잭션 실패

```
function register(address voter) public onlyChair {
    _____________________________________________
    …
}

function vote(uint toProposal) public validVoter {
    _____________________________________________
    …
}

function reqWinner() public view returns (uint winningProposal) {
    _____________________________________________
    …
}
```

- **단계가 맞는지 확인하는 modifier**
  - validPhase(Phase reqPhase)
    - 현재 단계에 reqPhase 일때만 호출하도록 제한

```
modifier validPhase(Phase reqPhase) {

    _____________________________________
    _;
}
```

- 함수
  - 각 함수들에 validPhase modifier 추가

```
function register(address voter) public ____________________ onlyChair {
    …
}

function vote(uint toProposal) public ____________________ validVoter {
    …
}

function reqWinner() public ____________________ view returns (uint winningProposal) {
    …
}
```

호텔방 예약 스마트 컨트랙트

- **호텔방 예약**
  - 암호화폐를 지불하고 호텔방 예약, 방은 하나
  - 예약함수: book()
    - 10 이더 이상으로 예약 가능: 미만일시 트랜잭션 실패
    - 방이 비어있는지에 대한 여부 확인: 비어 있지 않으면 트랜잭션 실패
    - 방의 상태 변경
    - 소유자에게 예약금 송금
    - 이벤트
      - 매개변수: 예약자와 금액
  - 빈방으로 초기화하는 함수: reset()
    - 소유자만 가능

호텔방 예약 스마트 컨트랙트

- **데이터 선언**

```
contract HotelRoom {
    enum Status {Vacant, Occupied}
    _____________________________
    _____________________________

    ...
}
```

- **이벤트 선언**

```

```

- **생성자**

```
    constructor() {
        _____________________________
        _____________________________
    }
```

호텔방 예약 스마트 컨트랙트

- modifier onlyWhileVacant: 빈방인지 확인

```
modifier onlyWhileVacant() {


}
```

- modifier costs(uint _amount): 일정금액 이상인지 확인

```
modifier costs(uint _amount) {


}
```

- book(): 예약 함수

```
function book()
{




}
```

호텔방 예약 스마트 컨트랙트

서강대학교 SOGANG UNIVERSITY

- modifier onlyOwner

```
modifier onlyOwner() {


}
```

- 빈방으로 초기화 reset()

```
function reset()
{


}
```

lottery 스마트 컨트랙트

- **lottery 게임**
  - 참여자들이 특정 금액을 베팅하고 추첨을 통해 승자를 결정
  - 1 이더로만 참여 가능: 초과나 미만일시 트랜잭션 실패
  - 같은 계정의 중복 참여 불가
    - 이미 참여했는지 확인하는 코드 필요
  - 베팅한 모든 금액을 승자에게 송금
  - 게임이 끝나면 다음 게임이 가능하도록 초기화

lottery 스마트 컨트랙트

- **데이터**
  - address public manager; // 관리자(배포자)
  - address[] public players; // 게임 참여자
- **당첨자를 위한 무작위수 구하기**
  - uint(keccak256(abi.encodePacked(block.number, block.timestamp, players.length)))
  - 이 값을 참여자수로 나누어 나머지값 구하기
  - 나머지값을 인덱스로 하는 참여자가 당첨자
- **이더리움의 무작위성(randomness) 시뮬레이션**
  - 일반적으로 현재 블록 타임스탬프, 블록 해시, 블록 번호 등과 같은 블록체인의 변수들을 활용

lottery 스마트 컨트랙트

- **함수**
  - enter
    - 사용자가 로또에 참여하는 함수
    - 베팅 금액을 확인
    - 이미 참여하였는지 확인한 후 아니면 players 배열에 참여자 주소를 추가
  - pickWinner
    - 당첨자를 무작위로 선택하고, 당첨자에게 컨트랙트 잔액을 송금
    - restricted 제한자(modifier)를 사용하여 관리자만 호출할 수 있음
    - players 배열 초기화
  - getPlayers
    - 참여자 목록을 반환하는 함수

lottery 스마트 컨트랙트

- **데이터 선언, 생성자, modifier, getPlayers()**

```
contract Lottery {
    address public manager;
    address[] public players;

    modifier restricted() {


    }

    constructor() {

    }

    function getPlayers()
    {

    }
```

lottery 스마트 컨트랙트

- enter()

lottery 스마트 컨트랙트

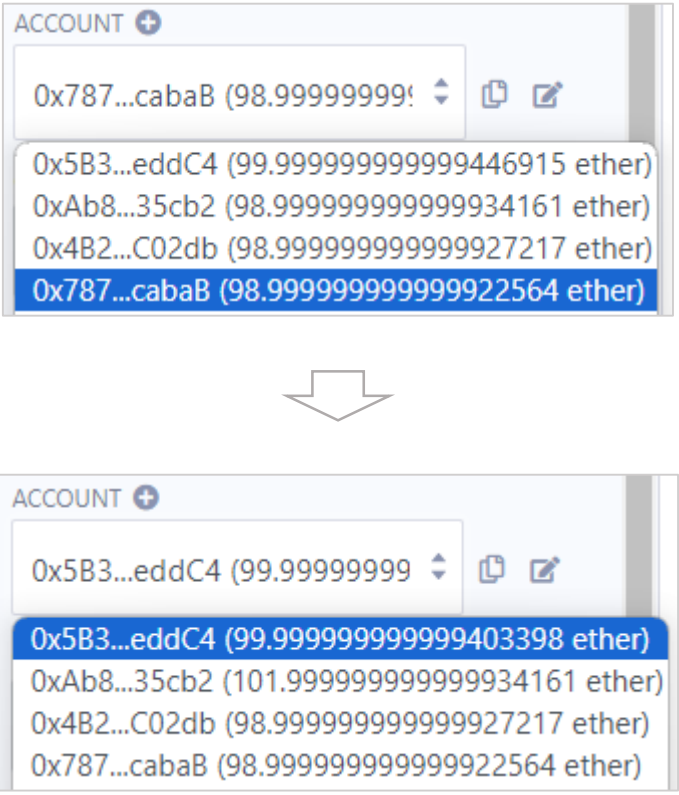
- **pickWinner()**

```
function random() private view returns (uint) {
    return uint(keccak256(abi.encodePacked(block.number, block.timestamp, players.length)));
}

function pickWinner()
```

lottery 스마트 컨트랙트

- 테스트
  - Account1 배포
  - Account2 1이더로 게임 참가
  - Account3 1이더로 게임 참가
  - Account4 1이더로 게임 참가
  - getPlayers
  - Account2가 pickWinner 호출
  - Account1이 pickWinner 호출



lottery 스마트 컨트랙트

- **이벤트 추가**
  - 참가자 정보: 베팅 참여시(enter 함수 실행시)
  - 우승자 정보와 금액: pickWinner 함수 호출시
- **배포자(소유자)는 베팅 참여 못하게 변경**
- **참여 가능(베팅 단계)에만 참여(베팅) 가능하게 변경**
  - 배포자(소유자)가 단계 변경
  - 베팅이 가능한 단계(enter 호출 가능)와 베팅이 불가능한 단계(enter 호출 불가)로 구분
  - 두가지 방법 가능
    - 단계별 처리 추가
    - 베팅 가능 여부 변수