CRP\_contract 설계

시나리오

➀ 사용자가 UI에서 요구하는 ‘입력 폼’ 작성

➁ 입력 폼 데이터를 기반으로 ‘token, fund, crowdsale contract’가 생성

➂ crowdsale ‘시작시간’(입력 폼 데이터)에 맞춰 crowdsale 시작

- hardcap 도달 시 sale 성공처리 및 종료

- hardcap 미 도달 시 sale 만료타임까지 sale 지속

- sale 종료까지 softcap 미 도달 시 모금액 refund

- crowsale은 여러 번에 걸쳐 생성할 수 있다. 중복 생성은 불가

➃ USER들이 CRP를 전송하면 crowdsale은 fund에 CRP보관 및 관리

➄ crowdsale 모인 CRP의 절반(협의 필요)은 ‘초기자금’으로 선 지급

- fund\_contract => owner\_wallet 전송

➅ 초기자금 지급 약 3개월 뒤부터 투표가능(withdraw, refund 둘 다)

- 두 개의 poll 모두 생성된 후 3개월의 재생성 기간을 가진다.

- withdraw는 owner, refund는 crowdsale 참가자가 개최 권한을 가진다.

➆ 투표결과에 따라 모금된 CRP를 전송한다.

- win : owner에 투자금 전송

- lose : 투자금을 투자자에게 refund

➇ 컨테츠 제작이 완료되고 플라블라에 서비스 등록 및 숨 거래소에 토큰이 상장되어

거래가 가능하고, 일정 기간에 대한 수익이 토큰 지분에 따라 CRP로 배분된다.

협의가 필요한 사항 Contract 설계

➀ 투표권 주체 범위(프로듀서도 투표에 참여 여부)

➁ 투표영향력 제한 범위(지갑 당 최대 행사 가능 표)

➂ 초기자금에 대한 비율

➃ 발행방식

➄ 전체 토큰에 대한 제작팀과 투자자팀의 비율

Contract 설계

**ㅇ Contract deploy 순서 및 과정**

1. 프로듀서가 mist의 입력란을 올바르게 전부 기입 후 프로젝트 생성
2. CrpProject.sol deploy
3. CrpToken.sol deploy – constructor(name, symbol)
4. CrpPoject.sol – setCrpTokenContract(address CrpToken)함수 실행
5. CrpFund.sol deploy
6. CrpPoject.sol – setCrpFundContract(address CrpFund)함수 실행
7. CrpCrowd.sol deploy

* constructor(address \_token, address \_fundContract, uint \_startSale
* , uint \_endSale, uint256 \_softCap, uint256 \_hardCap, uint \_foundRate

, address \_beneficiary)

1. CrpProject.sol – setCrpCrowdSaleContract(address CrpCrowdSale)함수 실행
2. Poll관련해서는 합의가 필요

**ㅇ CRPToken.contract(ERC20규격준수) – 토큰을 관리하는 contract**

ㅁ global variable

* string public name : 토큰 이름, 생성자 owner input
* string public symbol : 토큰 심볼, 생성자 owner input
* uint public decimals : 토큰 소수점 자리수, 18로 고정
* address public owner : 토큰 발행자의 주소, msg.sender
* bool public token\_enable : 토큰 활성화 플래그 변수
* uint256 public supply : 토큰 총 발행량
* bool public issuanceFinished : 토큰이슈 비활성화
* mapping(address => uint) balances

: 계정 별 balance를 매핑한 변수

* mapping(address => tokenInfo) pendings

: 계정 별 계정에 귀속되어 있으나 거래될 수 없는 토큰

* mapping(address => mapping(address => uint)) approvals

: 각 계정에 대해, 계정 별 허용 위임금액을 매핑한 변수

* mapping(address => bool) table\_lock

: 계정 별 lock 여부를 매핑한 변수

* mapping(address => bool) table\_admin

: 계정 별 관리자 권한 획득 여부를 매핑한 변수

ㅁ event

* event TokenBurned(uint \_amount, uint \_supply) 토큰 소각
* event TokenEnabled() 토큰 활성화
* event TokenDisabled() 토큰 비활성화
* event AdminAdded(address \_address) 관리자 권한 계정 추가
* event AdminDeleted(address \_address) 관리자 권한 계정 제거
* event AddressLocked(address \_address) 계정 락 설정
* event AddressUnlocked(address \_address) 계정 락 해제
* event Issue(address indexed \_to, uint256 \_value) 토큰 생성
* event Settled(address \_who, uint \_pendings, uint \_newbalance)

ㅁ modifier

* isTokenOwner(address) 토큰 오너인지 검사
* isTokenAdmin(address) 토큰 관리자인지 검사
* isTokenTransfer(address) 전송권한 여부 검사
* canIssue( ) 토큰생성 가능여부 검사

ㅁ 구매자 별 잠금토큰정보를 관리(pendings)\_struct

* dataType : mapping(address => struct[ ])pedings

struct { 잠긴토큰의 양, 잠기는 시간, 잠금 상태 }를 인자로 가짐

ㅁ constructor(string \_name, string \_symbol) public

* name, symbol을 인자로 받아 변수에 초기화
* decimals = 18, supply = 0 , owner = msg.sender로 고정
* table\_lock[owner] = fasle, token\_enable = false로 설정

ㅁ totalSupply( )\_public\_view - supply를 확인하는 기능

* return supply(공급량을 나타내는 글로벌 변수)

ㅁ balanceOf( address \_who)\_public\_view - 구매자 별 지분율을 관리

* param : CRP address, 정보를 조회하고자 하는 주소
* 기능 : 맵핑에 잔고 정보를 저장하여 필요할때마다 주소를 입력 받아 잔고 조회
* return : balances[\_who]에 저장된 토큰잔고량

ㅁ allowance(address \_owner, address \_spender) public view – 허가 잔액 조회

* Param : 오너주소 \_owner, 대상주소 \_spender
* Return : approvals[오너주소][대상주소]

ㅁ showAddrLockStat(address \_who)\_public\_view – 대상자 잠금상태 여부를 조회 기능

* Param : 대상 주소 \_who
* Return : table\_lock[\_who]

ㅁ showAdminStat(address \_who)\_public\_view - 대상자의 관리자 여부 조회 기능

* Param : 대상 주소 \_who
* Return : table\_admin[\_who]

ㅁ showPendings(address \_addr)\_public\_view – 대상자 잔고 팬딩상태 조회 기능

* param : 대상 주소 \_who
* return : amout 팬딩 양 정보

lockEndTime 락 종료 시간

releasd 락 여부

ㅁ transfer(address \_to, uint \_value)\_public – 전송기능

* 호출 : 컨트렉트 생성자(owner)가 송금할떄 사용 가능, safeMath 사용
* Modifier : isTokenTransfer(msg.sender)
* param : address \_to, uint \_value
* 기능 : 오너.safeSub(\_value), \_to.safeAdd(\_value)을 통해 잔고 밸런스 조정
* return : true
* Event : tranfer(msg.sender, \_to, \_value)

ㅁ transferForm(address \_from, address \_to, uint \_value)\_public

* 권한위임을 통한 대리송금기능
* Modifier : isTokenTransfer(msg.sender, \_form) 토큰전송 가능한가(수신,송신)
* Param : address \_from, address \_to, uint \_value
* 기능 : 보내고자 하는 토큰량에 대해 권한위임을 한 다음 (approv)

수신인과 송신인의 계좌를 \_value만큼 조정하여 전송

* return : true
* Event : tranfer(msg.sender, \_to, \_value)

ㅁ approve(address \_spender, uint \_value)\_public - 토큰 권한위임 기능

* Modifier : isTokenTransfer(msg.sender)
* Param : address \_spender, uint \_value
* DataType : mapping(address => mapping(address => uint)) approvals
* 기능 : 입력받은 계정 \_spender에 대한 \_valuea만큼의 허용 토큰량
* return true
* Event : Approval(msg.sender, \_spender, \_value)

ㅁ setTokenBurn(uint \_amount)\_public - 소각으로 인한 통화량 감소 기능

* Modifier : isTokenOwner(msg.sender)
* Param : 소각해야할 토큰의 양 uint\_amount
* 기능 : 오너가 가지고 있는 토큰의 양 및 총량을 param만큼의 양을 소각
* Event : TokenBurned(\_amount, suupply);
* Return true;

ㅁ issue(address \_to, uint256 –value, uint256 \_lockEndTime)

* Modifier : isTokenAdmin, canIssue
* Param : CRP구매자 주소, 지급받아야할 토큰 개수
* 기능 : 구매자 잔고 및 토탈 value에 정해진 비율따라 지급
* Event : issuer(\_to, \_value)

ㅁ 토큰을 사용 가능하게 허가하는 기능(setTokenEnable, setTokenDisable)\_external

* Modifier : isTokenOwner
* 기능 : 오너의 token\_enable을 true로 초기화, false로 초기화
* Event : TokenEnabled, TokenDisabled
* Return true

ㅁ 주소를 락/언락하는 기능(setAddressLock, setAddressUnlock)\_external

* Modifier : isTokenOwner
* Param : \_target 대상 주소
* 기능 : 락 변수인 table\_lock[\_target] true or false로 변경
* Event : AddressLocked, AddressUnlocked

ㅁ 세일 주소 잠금 기능(saleAddressLock, saleAddressUnlock)\_external

* Modifier : isTokenOwner
* Param : \_target 대상 주소
* 기능 : 락 변수인 table\_lock[\_target] true or false로 변경
* Event : AddressLocked(\_target), AddressUnlocked(\_target)

ㅁ 세일관리자를 위임 및 해제 하는 기능(saleAdminAdd, saleAdminDelete)\_external

* Modifier : isTokenOwner(tx.origin)
* Param : address \_admin
* 기능 : table\_admin[\_admin]의 변수를 true or false로 초기화한다.
* Event : AdminAdded((\_admin), AdminDeleted(\_admin)

ㅁ 대상에게 사용가능한 값을 허가해주는 기능(saleApprove)\_external

* Modifier : isTokenOwner(tx.origin)
* Param : address \_spender, uint \_value
* 기능 : approvals매핑에서 \_spender의 값에 \_value값 초기화

ㅁ 정산기능(전송된 토큰의 잠금을 푸는 기능(settle))

* Modifier : isTokenAdmin(msg.sender)
* Param : address \_who
* Balances[\_who]에 pendings[\_who].amount의 값 만큼 추가
* Pedings값 초기화

**ㅇ CrpCrowdSale.sol – 토큰 세일을 담당하는 Contract**

ㅁ global variable

* Address fundContract : 모금컨트렉트 주소
* CrpFund public funobj : 모금컨트렉트 객체
* uint public totalCRP : 받은 총 Crp 수
* uint public startSale : 판매시작시간
* uint public endSale : 판매종료시간
* CrpToken public tokenobj : 판매하는 토큰 객체
* uint public foundRate : 모금 비율
* uint256 public softCap : 소프트캡 금액
* uint public ceiling : 하드캡 금액
* uint256 public default\_ratio : 교환비율
* uint eth\_max & min : 최대 최소 전송 허용량
* address[ ] buyer : 구매자 명단
* Enum SaleStage public stage { 디플로이, 셋업, 시작, 종료 }
* SaleInfo\_구조체 : 구매여부, 계정, 송금한 CRP양, 토큰 양 기록
* Mapping(address => SaleInfo) sales 주소별 구매현황 기록

ㅁ modifier

* isValidPayload 이더리움 수취가능한지 검사

ㅁ event

* event SaleStarted(uint \_ceiling, uint \_duration); // 세일 시작 이벤트
* event SaleEnded(uint \_ceiling, uint \_elapsed); // 세일 만료 이벤트
* event SaleHalted(uint \_elapsed); // 세일 종료 이벤트
* event FundSaled(address \_purchaser, uint \_amount, uint \_rests, uint \_elapsed, uint \_ratio); // 펀드 전송 완료 이벤트

ㅁ constructor(address \_token, address \_fundContract, uint \_startSale, uint \_endSale

, uint256 \_softCap, uint256 \_hardCap, uint \_foundRate, address \_beneficiary

, uint \_max, uint \_min, uint \_ratio)\_public

* 위의 입력 폼들은 사용자 UI에서 입력 받아야할 사항
* 기능 : 입력받은 인자들을 알맞은 글로벌 변수에 초기화
* 토큰주소, 펀드컨트렉트 주소, 시작시간, 종료시간 초기화
* 토큰의 어드민으로 컨트랙트 주소 등록
* 각각의 토큰세일에 필요한 정보를 초기화

ㅁ CRP 모금 함수\_payable\_public

* 기능개요 : 입금된 금액 및 계좌정보를 처리하는 함수를 콜 및 직접 처리 기능
* 기능 1 : 각종 제한조건에 의한 환불기능(적절한 시간, 금액)
* 기능 2 : 입금정보(시간, 금액, 주소)를 구조체 및 맵핑에 기록하는 기능
* 기능 3 : 받은 CRP를 펀드 Contract에 전송하는 기능

ㅁ 세일정보를 업데이트 하는 기능(updateSaleInfo)\_private

* 컨트렉트 내부에서만 호출 가능
* Param : 주소, 전송CRP양, 토큰비율
* 기능 1 : 주소가 정확한지 판별(require)
* 기능 2 : 입금자의 주소, 보낸 양, 보낸 여부를 초기화

ㅁ sale종료 후 조건에 따른 동작을 하는 기능(exitCrowd)\_internal

* 기능 개요 : 종료상태 점검, 모금액에 따라 기능이 다르게 동작
* 기능 1 softCap 미달 : fund contract에 refund함수 호출 **or return false;**
* 기능 2-1 softCap 달성 : fund contract에 갱신 함수 호출 **or return true;**
* 기능 2-2 : 지정된 교환 비율에 따라 토큰 분배(tokenobj.settle 함수 콜)
* saleStart, saleHalt 함수로 시작 및 정산, **tokenObj에 대한 콜 주체 선별 필요**
* stampSaleStarted & stampSaleEnded로 크라우드 세일 상태변수 관리

ㅁ 각종 조회 기능을 가진 함수

* balanceof : 주소에 해당하는 수취한 토큰양
* amountof : 주소에 해당하는 전송한 CRP의 양
* numOfBuyer : 총 구매자 수

**ㅇ Fund Contract**

ㅁ global varable

* uint256 public totalFund : 총 모금량
* uint256 public currentFund : 현재 남은 모금량, 인출 가능한 C양
* address public owner : 생성 오너 주소

ㅁ event

* event deposit(address \_address, uint256 \_amount); //Crp입금시 알림
* event transfer(address \_address, uint256 \_amount); //Crp 출금시 알림

ㅁ construct( ) \_ public

* owner = msg.sender;

ㅁ 크라우드세일에서 보낸 CRP를 저장 및 정보처리 하는 함수 \_ payable \_public

* 기능 1 : totalFund에 갱신 totalFund+=\_value
* 기능 2 : currentFund에 갱신 currentFund+=\_value
* 기능 3 : event deposit을 활용한 입금 알림

ㅁ 투표 결과에 따른 지정된 값을 오너에게 전송(withdraw)

* Param : address \_to(수취자), uint \_amount(양)
* 전제 조건 : \_amount가 currentFund보다 적어야한다.
* 기능 1 : 현재 남은 금액(currentFund)에서 \_amount만큼 차감
* 기능 2 : \_to.tranfer(\_amount)를 통한 송금기능
* 기능 3 : event transfer을 활용한 송금 알림

ㅁ 각종 조회 기능

* showTotalFund( ) = 총 모금 된 절대 C게양을 보여 줌 return totalFund;
* showRemainFund( ) = 인출 가능한 Crp 양을 보여 줌 return currentFund;

**ㅇ poll interface**

ㅁ 종류 : roadMap, withdraw, refund, crowdsale 조절 예정

ㅁ 공통 변수

* uint agree, disagree : 찬반 표수
* uint totalPoll : 총 투표양
* uint startTime : 투표 시작시간
* uint endTime : 투표 종료시간
* maxWeight = 10000;
* bool resultPoll : 투표 결과

ㅁ 공통 함수

* 각 표수 확인 함수
* 투표 시간 확인 함수
* 개표 하는 함수, if(표 찬 ? 반, => resultPoll true ? fasle)

ㅁ roadMap Contract

* 기능 1 찬성 : 지정된 금액만큼 오너에게 전송 fund\_winTransfer call
* 기능 2 반대 : 크라우드세일 구매정보 인자를 활용한 fund\_refund call
* Return bool

ㅁ withDraw Contract

* 기능 1 찬성 : 지정된 금액만큼 오너에게 전송 fund\_winTransfer call
* 기능 2 반대 : null
* Return bool

ㅁ refund Contract

* 기능 1 찬성 : 크라우드세일 구매정보 인자를 활용한 fund\_refund call
* 기능 2 반대 : null
* Return bool

ㅁ CrowdSale Contract

* 기능 : 투표결과에 따른 프로젝트.ableCrowdPoll 플래그 변경

**ㅇ Project contract**

ㅁ global variable

* Address public owner : 프로젝트의 프로듀서 주소
* CrpToken tokenObj : 관리하는 토큰 객체
* Address public tokenAddr : 관리하는 토큰 주소
* CrpFund fundObj : 토큰 자금 관리하는 객체
* Address public fundAddr : fundContract 주소
* CrpCrowdSale public crowdobj [ ] : 회차별 크라우드세일 객체
* Address [ ] public crowdAddrs : 회차별 크라우드세일 주소
* uint crowdNum : 크라우드세일 회차번호
* CrpRoadMapPoll RMPobj [ ] : 회차별 로드맵 투표 객체
* Address[ ] roadMapAddrs : 회차별 로드맵 투표 주소
* uint RnpNum : 로드맵 투표 회차번호
* CrpWithdrawPoll WDPobj [ ] : 회차별 금액 전송 투표 객체
* Address[ ] WdpAddrs : 회차별 로드맵 투표 주소
* uint WdpNum : 전송 투표 회차번호
* CrpRefundPoll RFP [ ] : 회차별 리펀드 전송 투표 객체
* Address[ ] RfpAddrs : 회차별 리펀드 투표 주소
* uint RfpNum : 리펀드 투표 회차번호
* CrpCrowdSalePoll CSP [ ] : 회차별 크라우드세일 투표 객체
* Address[ ] CspAddrs : 회차별 크라우드세일 투표 주소
* uint CspNum : 크라우드세일 투표 회차번호
* bool ableCrowdPoll : 크라우드 세일 투표가능 여부
* constant uint WDPduring : 전송 투표 재생성 시간
* constant uint RFPduring : 환불 투표 재생성 기간
* constant uint CSPduring : 크라우드세일 투표 재생성 기간

ㅁ constructer

* param : 토큰객체, 자금객체
* 각종 회차번호 초기화

ㅁ 각 종 객체 배열 초기화 하는 함수

* Param : address \_addr
* setRMP, setWDP, setRFP, setCSP
* 종류obj[Num++] = \_addr

ㅁ 각종 투표를 관리하는 기능

* 투표 결과에 따라 fund의 함수를 콜, true false
* 각종 투표의 맴버변수 콜하는 기능
* 각종 투표의 재생성 시간 관리
* 투표 결과 및 재생성 시간에 따른 플레그 변수 관리