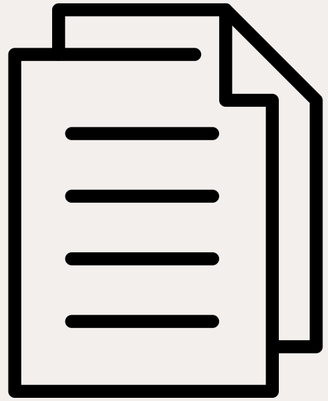
블록체인과 IPFS 문서 관리 시스템

분산 저장과 스마트 컨트랙트를 활용한 안전하고 효율적인 문서 관리 솔루션

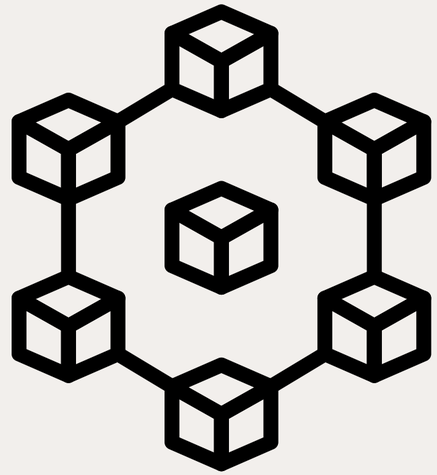
목차

프로젝트 배경



기존 문서 관리 시스템의 한계

-단일 실패 지점으로 인한 보안 위험
-데이터 무결성 보장의 어려움
-복잡한 접근 권한 관리
-높은 유지보수 비용
-조직의 효율성을 저하 중요한 정보 자산의 안전을 위협

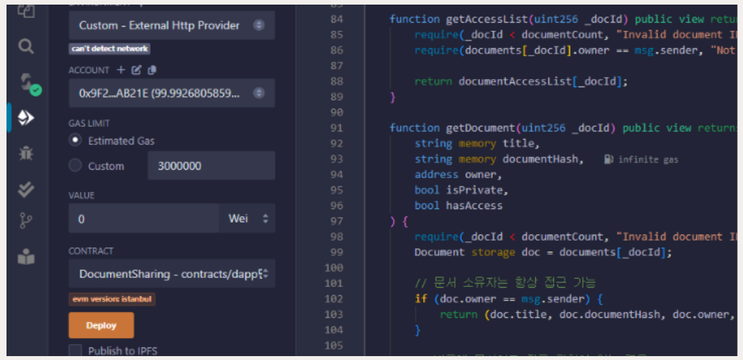


블록체인과 IPFS 도입의 필요성

- 탈중앙화로 단일 실패 지점 제거
- 스마트 컨트랙트로 자동화된 접근 제어
- IPFS로 효율적인 대용량 파일 관리
- 불변성을 통한 데이터 무결성 보장

프로젝트 목표

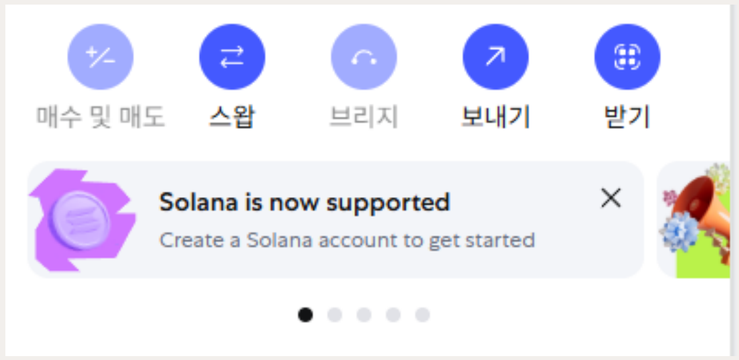
보안성과 무결성이 강화된 문서 공유 Dapp개발



Remix ide로 solidity기반 스마트 콘트랙트 작성

-Ethereum 스마트 컨트랙 활용해 문서 등록, 접근 권한 제어, 소유권 이전, 삭제 등을 온체인 프로세스로 처리

-기능 모듈화, 메타데이터 등록, 이벤트 기반 상태 추적 모니터링



Ganache와 metamask를 이용한 배포 및 적용 관리

-Ganache로컬 블록체인 사용하여 테스트 서버 구축 및 안전성 점검

-Remix에서 컴파일한 바이트 코드 및 ABI를 Web3.js 에 연결하여 배포

-RPC URL통해 메타마스크 연동 컨트랙트 매서드 호출 시 트랜잭션 흐름 및 이벤트 모니터링

IPFS를 통해 보안성 강화 및 NODE JS이용해 UI/UX 구성

-중앙 집중형 파일 저장소 대신 탈중앙화를 통한 무결성 확보를 위해 IPFS 사용

-IP기반 접근 배제, CID기반 문서조회

-HTML/CSS 기반 UI설정 및 Node.js 기반의 프록시 서버 통해 데이터 처리

-AES/PGP이용한 파일 암호화

주요 기능

01

파일 업로드/ 다운로드/삭제 등

IPFS 스토리지 레이어 기반 CID 생성 및 프로세스 관리

02

IPFS 해시 기반 파일 조회

Web3.js + Smart Contract 연동 파일 검색 기능 추가

03

파일 미리보기

IPFS에서 직접 fetch처리 방식 통한 이미지, PDF, TXT 미리보기 처리

04

권한 관리(공개 미공개 설정)

권한에 따라 UI 및 정보 접근 제어

05

소유권 이전 및 삭제 기능

transferOwnership 스마트 컨트랙트 함수 호출

SmartContract 기능별 함수 목록

DocumentSharing.sol

문서 생성 및 조회 관련

CreateDocument()

-사용자가 문서를 업로드하고 메타데이터를 스마트 컨트랙트에 등록

-/ipfs/add 호출로 IPFS에 파일 업로드(proxy서버)

getDocument(),

getDocumentCount()

-문서 목록 필터링 동작

-필터조건 !isPrivate, msg.sender ==owner, hasAccess() ==true

관리자 기능 관련

getUserOwnedDocuments(),

getUserAccessibleDocuments()

-사용자의 소유 문서 불러오기

-관리자의 경우 모든 문서에 access 가능

-

transferOwnership()

-문서 소유자 변경

-관리자 혹은 소유자 고유 기능

-Front구현위치

문서 상세정보 > 관리자 전용 메뉴

문서 다운로드 및 삭제 관련

deleteDocument()

-목록 삭제 및 이미 fetch된 문서 삭제

-관리자 권한 혹은 소유자 권한

-트리거이벤트: DocumentDeleted

getDownload(),

-IPFS에서 파일을 받아 사용자에 로컬로 전송

-해시 값 검증으로 권한 파악

-혹은 hasAccess로 소유자 확인

보안 및 권한 관련

grantAccess()

-문서 소유자나 관리자가 특정 사용자에게 접근 권한을 부여

-빈칸에 특정 이용자 지갑 주소를 적으면 됨.

revokeAccess(),

hasAccess()

-부여된 접근 권한 철회

-소유자와 관리자만 가능

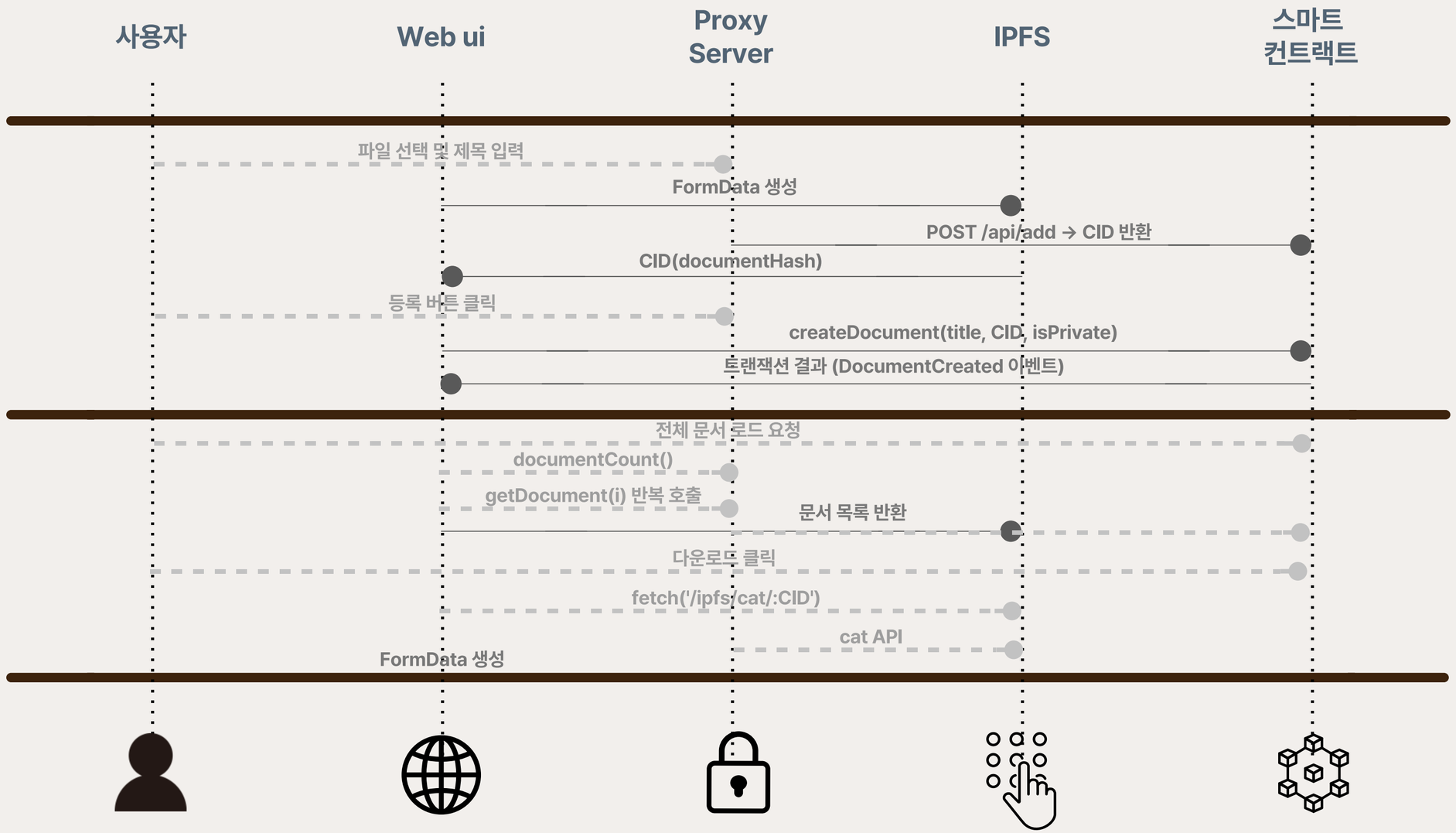
-트리거이벤트: AccessRevoked

-다운로드 버튼 클릭전 , UI 렌더 조건 판단 시 사용

시퀀스 다이어그램

문서 업로드~조회

조회 및 다운로드



Structure와 mapping 구조 형성

```
contract DocumentSharing {
    // 문서 구조체 정의
    struct Document {
        string title;              // 문서 제목
        string documentHash;       // IPFS 해시값
        address owner;             // 문서 소유자
        bool isPrivate;            // 비공개 여부
        string[] tags;             // 문서 태그
        uint256 viewCount;         // 조회수
        uint256 downloadCount;     // 다운로드 수
        bool isEncrypted;          // 암호화 여부
        string encryptionKey;      // 암호화 키 해시
    }
```

콘트랙트와 dapp의 기능 구성에 필요한 structure 정의

-스마트 컨트랙트 내에서 문서 개념을 온체인 오브젝트화

-블록체인에 저장되는 메타데이터의 단위

-문서 상태의 완결성 구축

-기능 의미 내포

```
// 문서 ID를 키로 하는 문서 매핑
mapping(uint256 => Document) public documents;

// 문서 접근 권한 매핑 (문서ID => (사용자주소 => 접근권한))
mapping(uint256 => mapping(address => bool)) public documentAccess;

// 문서별 접근 권한이 있는 사용자 목록
mapping(uint256 => address[]) public documentAccessList;

// 공유 링크 매핑
mapping(string => ShareLink) public shareLinks;

// 팀 매핑
mapping(uint256 => Team) public teams;
uint256 public teamCount;

// 사용자의 팀 목록
mapping(address => uint256[]) public userTeams;
```

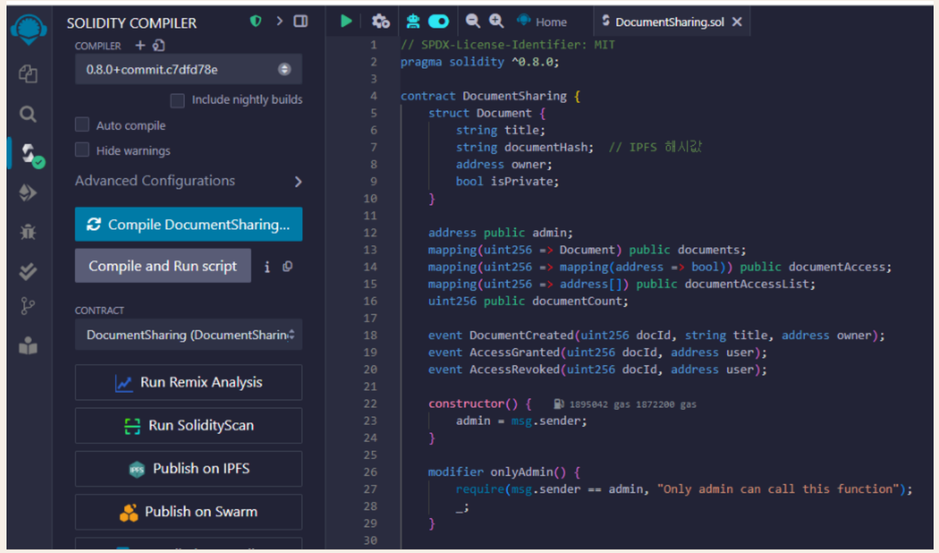
구조체와 mapping 이벤트 사전작업

-key-value 즉 해시를 저장하는 테이블로 사용

-상태 변수 저장 구조

-다중 매핑구조로 문서별 개별 사용자 권한 저장

스마트 콘트랙트 배포방법

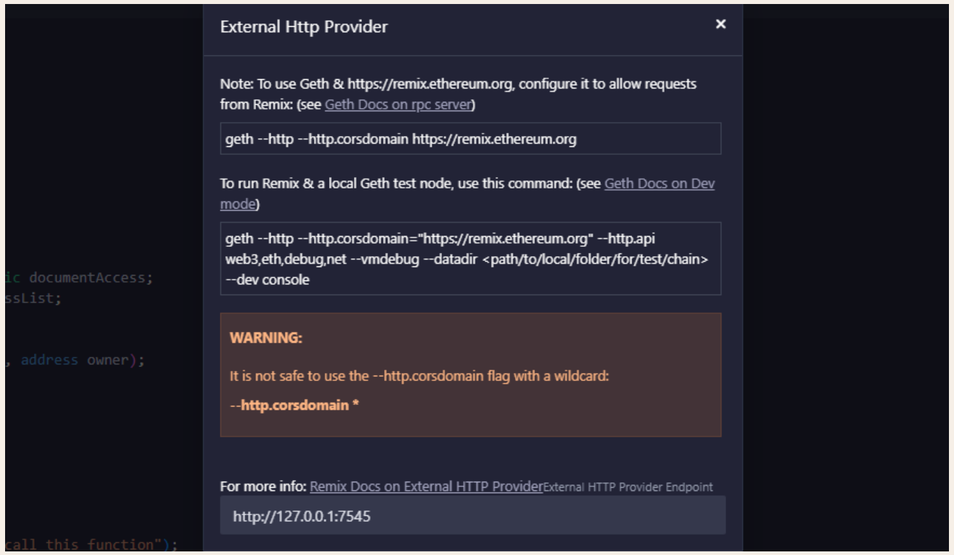


작성한 스마트 콘트랙트를 remix ide에서 컴파일

-컴파일 확인 후 ABI 코드 추출

-추출한 ABI 명세를 웹 구성을 위한 html.js에 import

-절차 완료후 deploy시도



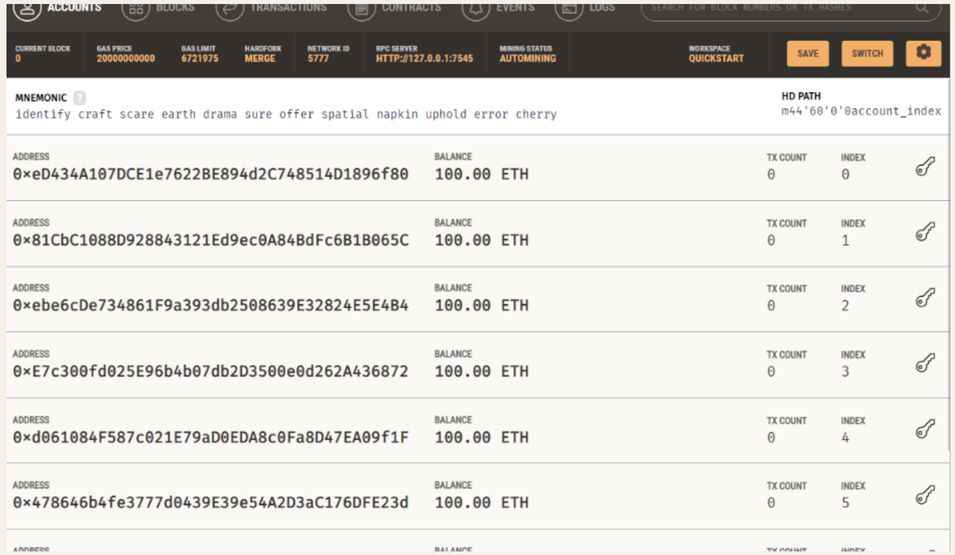
Environment를 External Http Provider로 변경

-연결 url http://127.0.0.1:7545

-Ganache로 구축 테스트 환경의 포트번호가 7545기 때문

-Deploy를 시작. Admin지갑주소 설정,

가나슈와 메타마스크 연결

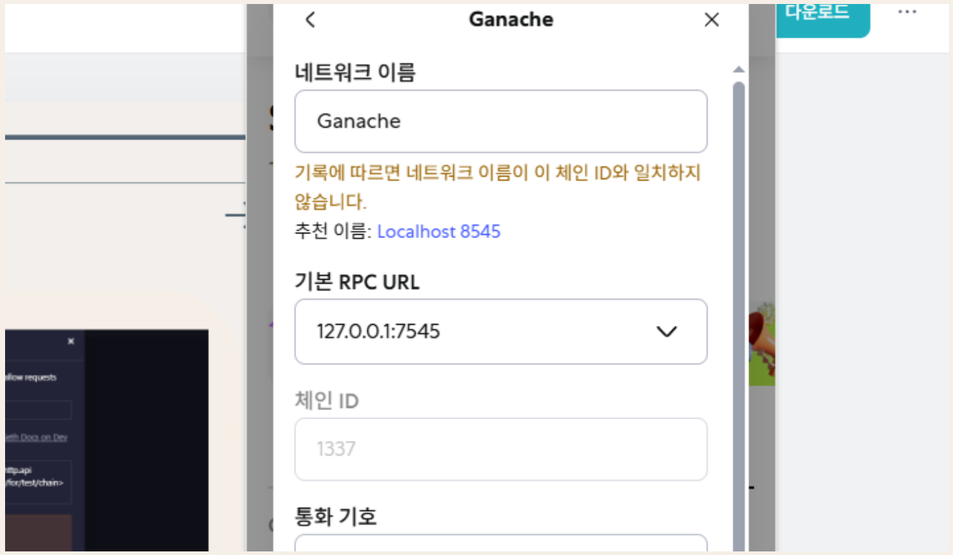


가나슈 quick start 후 로컬 테스트용 주소 확인

-테스트에 사용할 unlocked account 주소확인

-공개키와 개인키로 이루어짐

-개인키 누르면 복사할 수 있음. 이걸 복사해서 metamask 계정 설정에 넣으면 Ganache와 연결가능

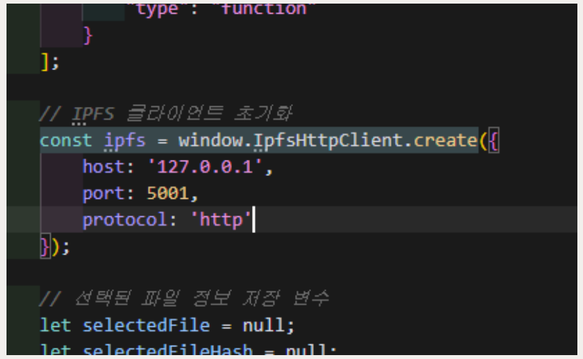


메타마스크 확장프로그램을 설치하고 Ganache네트워크 추가

-RPC URL을 가나슈의 주소와 동일하게 해서 네트워크 추가

-이후 네트워크 설정들어가 새로 생성한 Ganache네트워크로 바꿔줘야 각 주소별 이벤트를 모니터링 가능

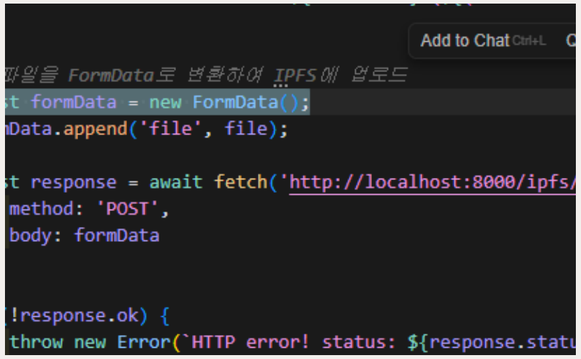
웹 서버 연결을 위한 html 및 json명세



IIPFS노드와 연결

-로컬에서 실행중이 ipfs 노드와 연결해서 파일을 등록하거나 가져올 수 있도록 설정

-분산파일 저장을 위해 ipfs사용



선택파일 백엔드 전송 로직

-구동하는 서버를 명시해주고 파일을 받아와 async ~await 로직을 사용하여 ipfs에 업로드

-IPFS는 직접 브라우저에서 post업로드를 받지않기때문에 post사용과 프록시 서버를 동시에 해야함.

사용자 경험 흐름도

-스마트 컨트랙트와 연결해 함수 호출 및 트랜잭션 실행.

-접근 실패에 대한 에러메시지 출력

-블록체인 기반 접근 제어/문서 등록 기능은 스마트 컨트랙트를 통해서만 가능

node.js 프로젝트 메타정보

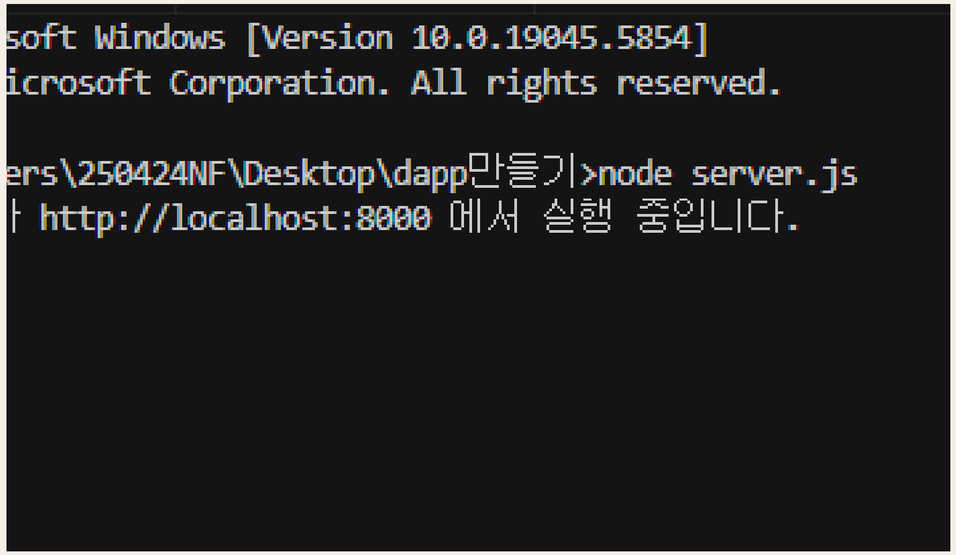
-프록시 서버에 대한 설정

-서버 시작 로직

-cors, express, multer, form data, node-fetch등 로직 정의

-IPFS HTTP API 호출

웹 연결 및 테스트



deploy후 주소와 abi를 html파일에 넣고 서버실행

-metamask 미리 만들어준 Ganache네트워크와 연결하는 것 잊지말것.

-remix에서 deploy한 계정과 같은 계정으로 메타마스크 연결

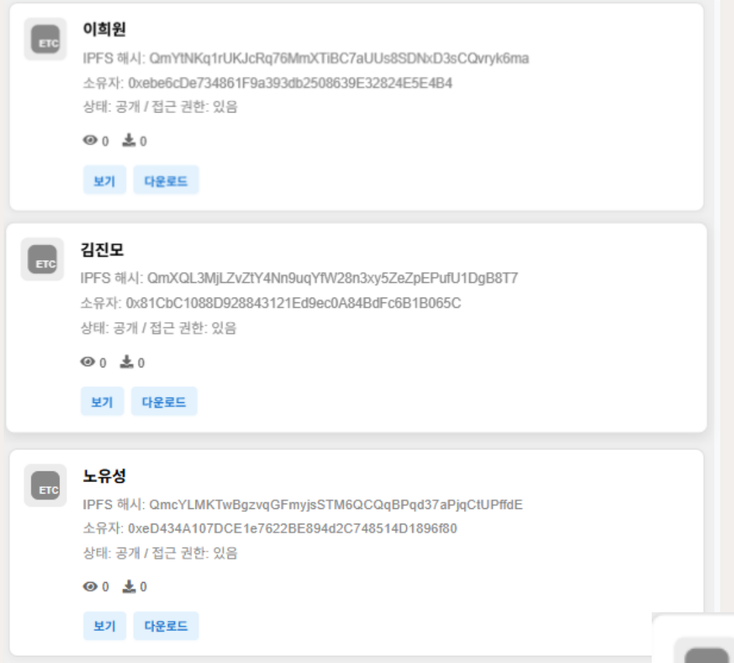
메타마스크에서 원하는 계정 설정 후 테스트

-Remix ide에서 주소 바꿀땐 admin설정 미리해주기

-계정 변환할때마다 권한 확인

-문서등록할때마다 메타마스크 컨펌

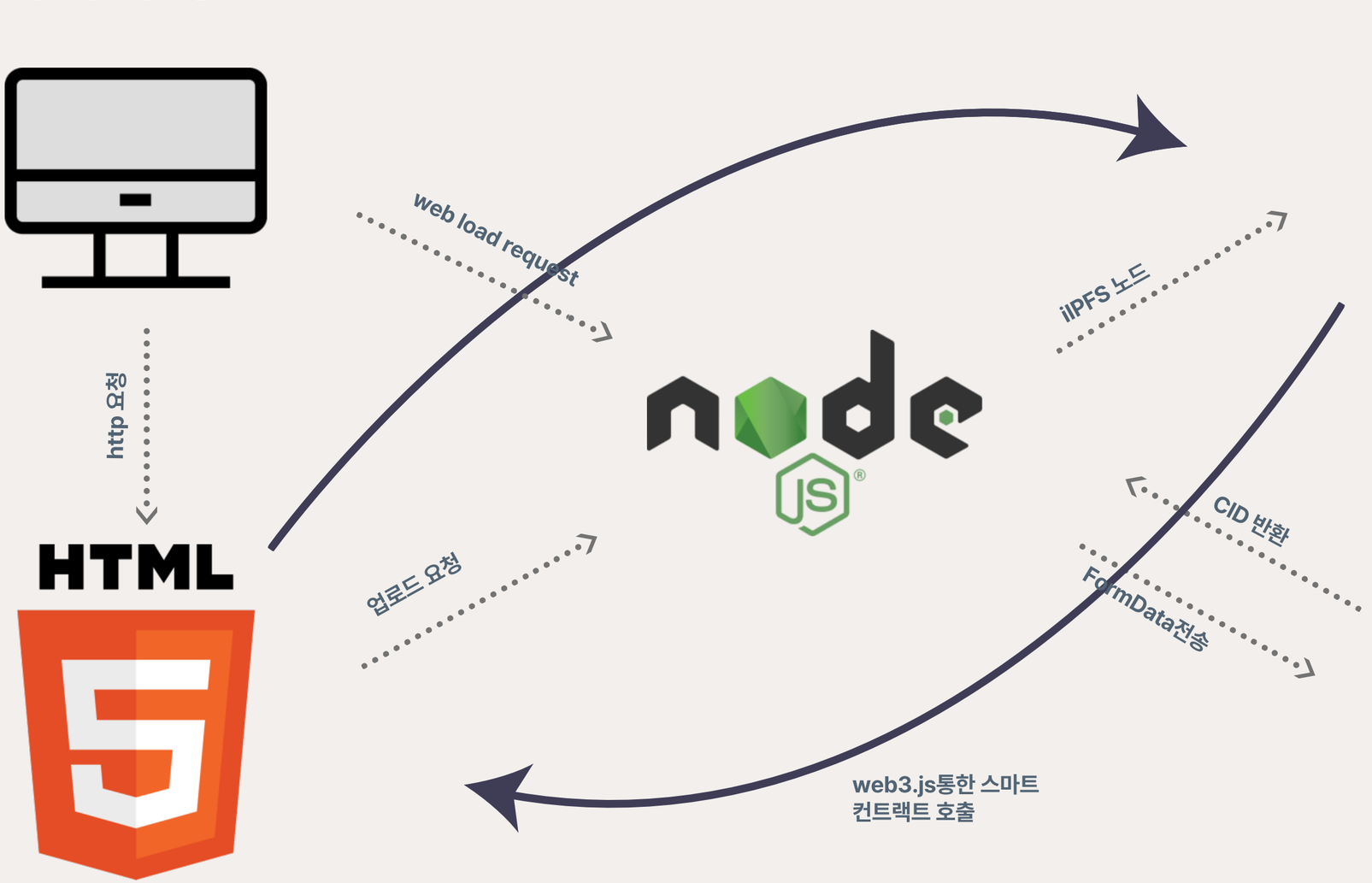
산출물 및 결과



웹 접속 후 디버깅 및 산출물 확인

-메타마스크로 계정 변경할 때마다 권한 변경되는 점 확인(권한부여, 수정, 삭제, 공유링크 생성)
-최종기능: 미리보기, 다운로드, 수정, 삭제, 소유권이전, 등등
-태그별 검색 기능 확인
-암호화 문서 다운로드시 비밀번호 입력

시스템 아키텍쳐

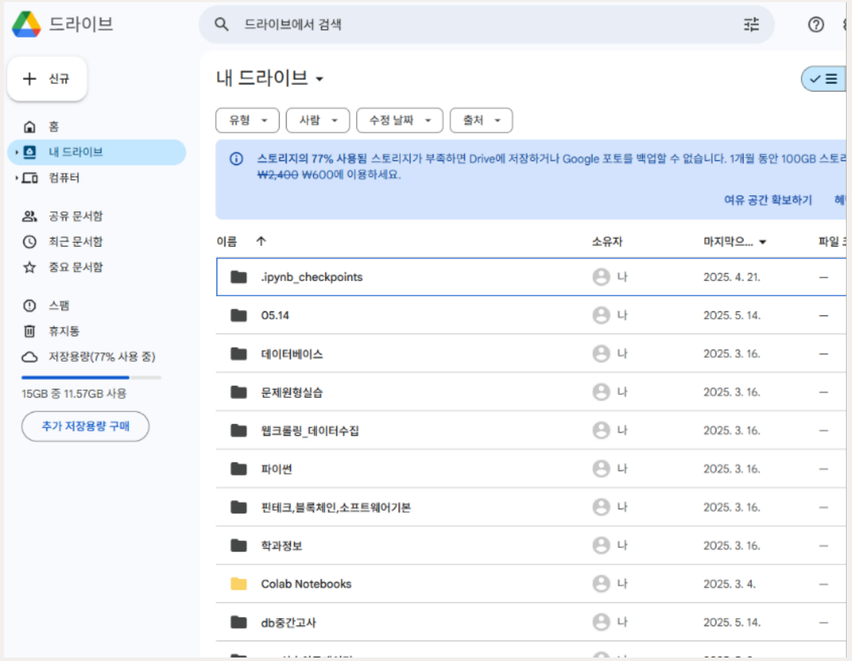

http 요청
HTML
web load request
업로드 요청
node
JS
ilPFS 노드
CID 반환
FormData전송
web3.js통한 스마트
컨트랙트 호출


remix
solidity


IPFS

한계점 및 개선 방안

파일의 암호화를 통한 무결성 확보와 다중 디바이스 동기화로 접근성 확보 최종적으로 웹사이트 연결형 Drive형태



IPFS 기반 공개형 데이터의 보안 한계

-문제점: CID는 암호화되자 않은 원본 파일의 해시값으로 생성되므로, CID를 알고있으면 누구나 파일에 접근가능.

-위험성: CID 자체는 메타데이터로 보호되지 않기에 암호화 필수

-개선방향: 업로드전 AES 또는 PGP 방식의 암호화

사용자 인증 체계 부재

-문제점: 현재 사용자 식별은 MetaMask 주소만으로 이루어지므로 보안 인증에 구멍이 발생할 가능성이 있음.

-한계점: 이메일/비밀번호 기반 인증, 다중 디바이스 간의 동기화 불가능

-개선방향: Web3Auth, Magic.link 등 Web2 + Web3 Hybrid 인증 방식 도입 및 ENS 기반 이름 부여 또는 Decentralized Identifier(DID) 연동

감사합니다