

<PROJECT 기밀보고서>

EOS Decentralized App 개발

: 음원 스트리밍 서비스

팀원 김신휘

유근국

이종서

교수 신영길

Neowiz 도종은

Table of Contents

1. Abstract	2
2. Introduction	2
3. Background Study	3
A. 관련 접근방법/기술 장단점 분석	
B. 프로젝트 개발환경	
4. Goal/Problem & Requirements	6
5. Approach	7
6. Project Architecture	7
A. Architecture Diagram	
B. Architecture Description	
7. Implementation Spec	8
A. Input/Output Interface	
B. Inter Module Communication Interface	
C. Modules	
8. Solution	10
A. Implementations Details	
B. Implementations Issues	
9. Results	15
A. Experiments	
B. Result Analysis and Discussion	
10. Division & Assignment of Work	18
11. Conclusion	18
◆ [Appendix] User Manual	19

1. Abstract

기존의 중앙화된 모델을 가지는 음원 스트리밍 서비스는 권력이 중앙에 집중되어 있고 그 내부가 불투명하다는 문제점을 가진다. 예컨대 최근의 음원조작 사태를 보면 그러한 문제점을 적나라하게 드러내고 있다고 할 수 있다. 또한 음원 제작자들에게 많은 수익이 돌아가지 않고 중앙 서비스사가 대부분의 수익을 취하는 불합리한 구조는 오래전부터 이야기 되어오던 문제이다.

블록체인 기술의 대두는 위에서 말한 중앙화된 모델의 문제점들을 해결할 수 있는 강력한 실마리가 될 수 있다. 블록체인 기술의 특징점이 탈중앙화와 투명한 기록 보관인 만큼 기존 음원 스트리밍 서비스의 문제점들에 정면으로 맞서는 해법이 될 수 있다.

다만 블록체인 기반의 서비스는 높은 투명성과 탈중앙화라는 장점과 함께 느린 서비스 속도라는 치명적인 단점을 가지고 있다. 2세대 블록체인인 이더리움이 약 20초의 블록 생성 속도를 보이기 때문에 사실상 일반 유저들에게 실용적인 서비스를 제공하기엔 무리가 있는 것이 사실이었다. 그러나 최근 3세대 블록체인인 EOS블록체인의 등장으로 이러한 단점은 크게 해소될 수 있을 것으로 보인다. EOS블록체인은 블록 생성 속도가 약 0.5초로, 기존의 중앙화 서비스에 비견될 수 있을 정도로 빠른 서비스 속도를 보여주고 있다.

따라서 우리는 기존 중앙화된 음원 스트리밍 서비스의 대안으로 EOS블록체인을 이용한 음원 스트리밍 서비스를 제안하고 중앙화된 음원 스트리밍 서비스가 가졌던 문제점들을 해소해 보고자 한다. 우리의 음원 스트리밍 서비스는 EOS블록체인의 스마트 컨트랙트를 통해 DAPP의 로직을 구현하고, IPFS와의 연동을 통해 음원 데이터를 저장하며, 웹페이지를 통해 유저 인터페이스를 제공할 것이다.

2. Introduction

BlockChain의 1세대인 Bit Coin은 2008년 10월 사토시 나카모토라는 가명을 쓰는 프로그래머가 개발하여, 2009년 1월 프로그램 소스를 배포했다. BitCoin은 중앙은행 없이 전세계적 범위에서 P2P방식으로 개인들 간에 자유롭게 송금 등의 거래를 할 수 있도록 설계되었다. 그리고 2015년 7월 30일 Vitalik Buterin에 의해 BlockChain 2세대인 ethereum이 개발되었다. 암호화폐로서의 기능만 하던 BitCoin에 비해 ethereum은 추가적인 정보, 계약서들을 담을 수 있게 됨으로써 다양한 서비스를 제공할 수 있는 플랫폼으로써의 기능을 제시하였다.

Ethereum이 이러한 플랫폼으로써의 기능을 제시하며 블록체인 기술은 미래의 기술로서 더욱 각광받게 되었다. BlockChain을 이용하면, 중앙화된 서버를 둔 서비스를 제공하는 것이 아니라 탈중앙화(decentralized)된 서비스를 사용자들에게 제공할 수 있게 되기 때문이다. 탈중앙화된 서비스의 가장 큰 장점으로 보안성과 투명성에 있다. 그러나 Ethereum을 이용한 서비스를 모든 중앙화된 서버를 대체하기에는 아직 치명적인 단점이 있다. 바로 속도가 중앙화된 서버에 비해 느리다는 것이다. 이것을 보완하기 위해 Daniel Larimer가 EOS 개발을 진행 중에 있다.

우리는 위에서 말한 EOS chain에서 Decentralized App을 개발해보고자 한다. 현재 EOS는 개발 중에 있어 Test-Network와 Main-Network도 출범되지 않은 상태에 있고, 안정화된 ethereum chain에 비해 Decentralized App 개발을 진행하는 것에 대한 정보도 부족하다.

그래서 우리는 먼저 BlockChain의 생태계에 대한 이해, Bit Coin과 Ethereum에 대한 이해를 먼저 하고자 하였다. 그 후에 Ethereum에서 Decentralized App을 개발해봄으로써 Decentralized App의 동작 원리와 구조를 파악하였다. 그리고 이제부터는 Test-Network와 Main-Network가 없는 상태인 EOS에서 개발을 진행하기 위하여 Local Test-Network를 구축하고, 구축한 Local Test-Network에서 다른 탈중앙화된 기술들, 그리고 웹 기술들과의 연동을 통하여 사용가능한 Decentralized App을 개발해보고자 한다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

- BlockChain 생태계

블록체인은 여러가지 문제점을 가지는 중앙화 서비스에 대한 대안적 개념으로 등장하였다. 블록체인의 가장 핵심적인 개념은 모든 정보가 투명하게 공유되고 탈중앙화 되어있다는 것이다. 모든 노드들이 동일한 거래장부의 복사본을 가지며 이는 외부에 공개된다. 기존의 중앙화 서비스가 모든것을 숨기는 방식을 통해 보안을 유지했다면 블록체인에서는 모든것을 공개하고 합의를 구하는 방식으로 보안을 유지하게 된다. 이러한 특성을 통해 블록체인은 기존의 중앙화 서비스가 가지고있던 권력집중문제, 데이터보관의 불안정성 문제, 불투명성 문제 등을 대부분 해소할 수 있다고 보여진다. 다만 모든 노드가 장부의 복사본을 가지고 합의 과정이 필요하다는 점은 동시에 느린 속도라는 치명적인 단점을 야기한다. 이는 3세대 블록체인을 비롯한 앞으로의 블록체인기술이 해결해야할 과제라고 할 수 있다.

블록체인 네트워크는 블록을 채굴하고 블록체인을 이어나가는 노드(마이너)들로 이루어진다. 노드들은 블록체인을 구성하고 합의된 공동의 거래장부를 각자가 가지게 된다. 블록체인의 한 유저가 일정한 정보를 담아 트랜잭션을 발생시키면 트랜잭션은 노드들 사이에 던져지게 된다. 노드들은 던져진 트랜잭션들을 블록에 주워담아 블록으로 포장하고자 노력하게 된다. 이 과정에서 어떤 노드가 포장한 블록이 전체 블록체인의 다음 블록이 될지를 결정해야 하는데, 이때 합의 알고리즘에 따라 다음 블록을 합의, 선택하여 블록체인에 이어붙이게 된다.

노드가 블록을 포장하는 과정은 해당 블록에 맞는 해쉬값을 찾는 과정이다. 여기서 해쉬값은 SHA-256 해쉬값이며, 이전 블록의 정보와 새롭게 연결할 블록의 트랜잭션 정보 등을 종합하여 구하게 된다. 각 블록의 해쉬값은 이전 블록의 해쉬값을 연쇄적으로 포함하게 되기때문에 이전의 모든 블록의 정보를 포함하게된다. 따라서 단 하나의 변조라도 발생하면 해쉬값의 변화를 일으키게 되어 합의받지 못해 블록체인에 연결될 수 없게된다. 블록체인이 위,변조로부터 안전하다는 것은 이와 같이 해쉬값의 연쇄적인 체인을 통해 위,변조를 찾아낼 수 있음을 말한다.

합의 알고리즘은 대표적으로 이더리움에서 채택하고있는 POW(작업 증명)과 EOS 등에서 채택하고 있는 POS(지분 증명)이 있다. 작업 증명의 경우에는 가장먼저 연산을 통해 올바른 해쉬값을 찾아낸 노드의 블록을 선택하게 된다. 찾아내기는 어렵지만 검증은 쉽다는 해쉬값의 특성을 이용하여 가장 빠르게 해쉬값을 찾아낸 노드가 broadcast 한 해쉬값에 대해 나머지 노드들이 검증을 거친 후 각자의 체인에 검증된 블록을 추가하는 방식으로 합의와 체인의 동기화가 이루어지게 된다. 이때, 가장먼저 해쉬값을 찾아낸 노드에게 블록마다 일정한 보상과 블록에 담은 트랜잭션에 할당된 수수료를 보상으로 제공하게된다. 이는 노드들이 블록체인에 참여하고 컴퓨팅파워를 제공하는 이유가된다.

POS(지분 증명)은 많은 지분을 오랫동안 가지고 있었던 노드의 블록을 선택, 합의할 가능성이 높도록 하는 합의알고리즘이다. 특히 EOS블록체인의 경우에는 DPOS(위임된 지분증명) 방식을 채택하고있다. DPOS는 유저가 투자한 지분만큼 투표력을 제공하고, 투표를 통해 블록프로듀서를 선출하여 선출된 블록프로듀서들만이 블록을 연결할 수 있도록 위임하는 합의 방식이다.

- Ethereum과 EOS의 특징 비교

1세대 블록체인인 비트코인은 오직 기축통화의 역할만을 수행하기 위해 만들어졌기 때문에 블록에 거래정보 등 아주 제한적인 데이터만을 담을 수 있었다. 그러나 2세대 블록체인인 이더리움이 등장하면서 블록에 거래정보 뿐만 아니라 실행 가능한 코드인 스마트 컨트랙트를 담을 수 있게 되었다. 블록체인이 응용프로그램의 플랫폼 역할까지 겸할 수 있게 된 것이다. 앞으로 등장할 3세대 블록체인인 EOS블록체인은 이러한 플랫폼 역할 뿐만 아니라 독자적인 네트워크를 구성할 수 있는 기능까지 가진다.

이더리움과 EOS블록체인의 가장 큰 차이점은 각자 채택한 합의 알고리즘에 있다. 이더리움은 POW 방식을 채택하고 있고, EOS는 DPOS방식을 채택하고 있다. 앞서 살펴보았듯 합의알고리즘은 블록체인에 어떤 블록을 연결 할지에 대한 합의를 어떻게 이끌어 낼 것인가에 대한 알고리즘이다. 이런 합의알고리즘의 차이는 결과적으로 블록체인의 속도와 보상방식, 수수료지불방식의 차이를 가져온다.

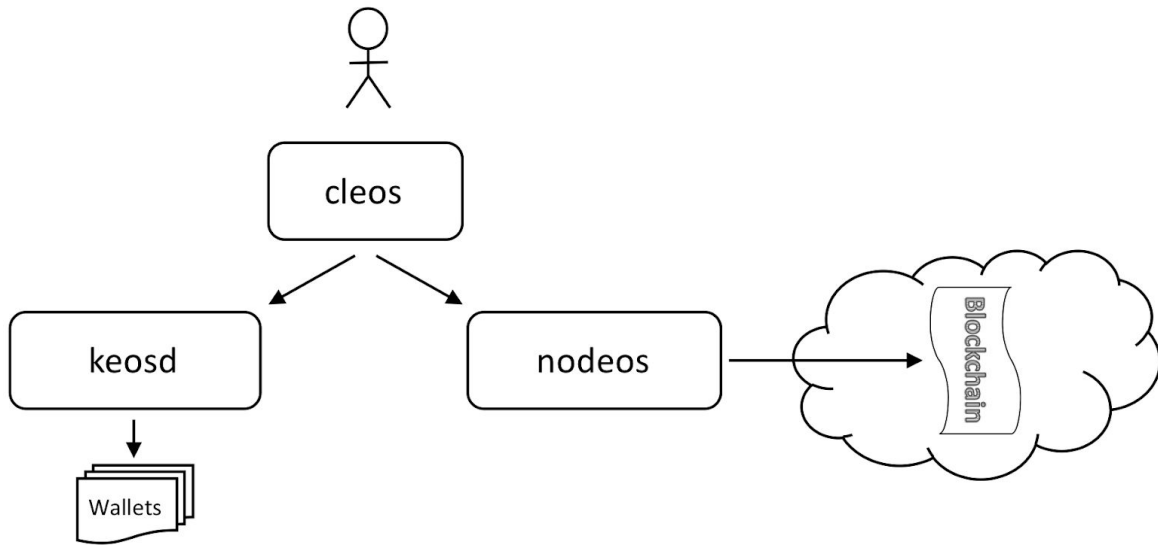
POW 방식을 채택하고 있는 이더리움은 누구든지 노드로서 블록체인에 참여할 수 있다. 무수히 많은 불특정의 노드들이 합의를 거치고, 장부를 동기화 하는 등의 과정이 필요하므로 필연적으로 속도가 느릴 수 밖에 없다. 사용자들은 더 빠른 처리 속도를 위해선 더 많은 수수료를 지불하여야 하고, 이러한 수수료는 더 많은 컴퓨팅파워를 유인하기 위한 방책으로 노드들에게 보상으로써 주어진다.

반면 DPOS 방식을 채택하고 있는 EOS는 일정한 수의 블록프로듀서를 선출하여 블록프로듀서들만 블록생성에 참여할 수 있도록 하고있다(이 때의 투표권은 네트워크 망에서의 token을 더욱 많이 걸어놓은 유저가 더 많은 힘을 가지게 된다). 적은 수의 믿음만한 노드들 만이 합의를 거치면 되므로 블록 생성 속도가 약 0.5초로, 20초에 달하는 생성 속도를 가진 이더리움에 비해 확연히 빠른 속도를 보여준다. 또한 EOS의 경우에는 유저에게 수수료를 받지않고 노드들에게 주어지는 보상 또한 일정 비율의 토큰으로 정해져 있다.

이더리움과 EOS의 또다른 차이점은 블록체인 네트워크의 구조에 있다. 이더리움은 하나의 메인 블록체인 네트워크를 가지고, 모든 노드들은 같은 하나의 네트워크 안에서 기능한다. 반면에 EOS에서는 통일된 하나의 네트워크만이 존재하는 것이 아니라 원한다면 얼마든지 독자적 네트워크를 구성할 수 있다. 또한 독자적인 네트워크간의 통신도 제공한다.

전반적으로 이더리움은 2세대 블록체인으로 출범한지 오래되었기때문에 비교적 안정화 되어있고 보편화 되어있는 반면 성능 및 기능에서 조금은 뒤떨어지는 모습을 보인다. 반대로 3세대 블록체인인 EOS는 성능 및 기능은 뛰어나지만 아직은 초기단계이기 때문에 안정성이 부족한 모습을 보인다고 볼 수 있다.

- EOS의 동작 방식



위의 그림에서 보드시피 EOS는 keosd 와 nodeos 이라는 두개의 데몬으로 이루어지고 cleos을 통해서 각각의 데몬을 컨트롤 하게된다.

Keosd는 월렛 관련 플러그인들을 로드하는 월렛 데몬이다. 월렛은 키들을 저장한다. 각 키들은 서로 다른 계정의 권한을 가질 수 있다. 지갑은 잠금 상태와 잠금이 해제된 상태를 가진다.

Nodeos는 EOS의 코어 데몬으로, 블록체인의 노드를 실행시키기 위한 플러그인들로 설정된다. 실질적으로 블록을 생성하고 전용의 API 엔드포인트를 제공하며 블록체인을 구성토록 한다.

Cleos는 nodeos 및 keosd와 통신하여 유저가 지갑 및 블록체인을 컨트롤 할 수 있도록 하는 커맨드라인 툴이다. Cleos를 이용하여 nodeos와 통신하기 위해선 nodeos 인스턴스의 엔드포인트를 필요로 한다. Cleos을 통해 월렛 및 키를 생성, 어카운트 생성, 월렛 락 및 언락, 컨트랙트 등록, 컨트랙트 사용 등을 할 수 있다.

- smart contract

EOS에서 smart contract는 C++로 작성한다. 그리고 EOS Blockchain에 smart contract가 저장될 때는 블록체인이 받아들일 수 있는 포맷인 WASM 포맷으로 저장되어야 하므로, cpp 파일을 이용하여 WASM의 텍스트 버전인 wast 파일로 컴파일하여 올라간다. 유저는 블록체인에 있는 wast파일을 쉽게 읽을 수 없기 때문에 abi 파일을 통하여 쉽게 블록체인에 있는 contract와 통신할 수 있다. 그래서 abi를 통하여 contract에 있는 action을 부르면 action handler에 의해 contract 안에서 알맞은 method가 작동하게 된다. 그리고 유저가 블록체인에 Application을 deploy할 때 자신이 걸어놓은 token의 갯수에 따라서(staking) Application이 사용할 수 있는 블록체인 망의 컴퓨팅 자원의 양이 결정된다.

B. 프로젝트 개발환경

Ubuntu 16.04

EOSIO dawn 4.2 build

eosjs 13.0.0

scatter 4.0.3

C++ (Smart Contract) / JavaScript (Web service)

Local testnet 구축

4. Goal/Problem & Requirements

우리의 목표는 블록체인 플랫폼에 맞는 탈 중앙화 앱을 개발하는 것이다. 현재 이더리움 플랫폼의 단점들이 많이 드러났고, 그것이 새롭게 등장한 EOS에서 빠르게 개선되고 있는 만큼, EOS 생태계에서 작동하는 탈 중앙화된 앱을 개발하고자 한다.

탈중앙화라는 특성을 어떤 앱으로 가장 잘 활용할 수 있을지 고민하다가, 음원 스트리밍 앱을 개발하기로 결정하게 되었다. 지금 현재의 음원시장은 음원의 저작권자보다 음원의 공급자(애플뮤직, 멜론 등)가 훨씬 더 많은 이윤을 취하고 있는 수익 분배 구조를 가지고 있다. 그럼에도 불구하고 음원 저작권자들이 음원시장을 떠나지 못하는 이유는 사용자들에게 원활하게 음원을 공급하고 저작권자에게 충분한 이윤이 돌아가는 시스템이 존재하지 않기 때문이다. 블록체인 플랫폼의 탈중앙화라는 특성은 중앙화 된 관리 없이도 음원의 공급, 수익의 분배를 할 수 있는 시스템을 구축하기에 최적의 방법이다.

음원 파일과 같은 비교적 큰 용량의 파일을 블록체인에 직접 저장하는 것은 매우 비효율적이므로, 그 대안으로 IPFS를 활용할 것이다. 음원을 저작권자의 저장소에 저장하고, 해당 음원 파일의 hash값을 chain에 등록하면 IPFS를 통해 음원 파일에 접근할 수 있다.

우리는 음원의 업로드, 검색, 재생 기능을 제공하는 음원 스트리밍 서비스를 EOS 블록체인 플랫폼 위에서 DAPP의 형태로 구현할 것이다.

5. Approach

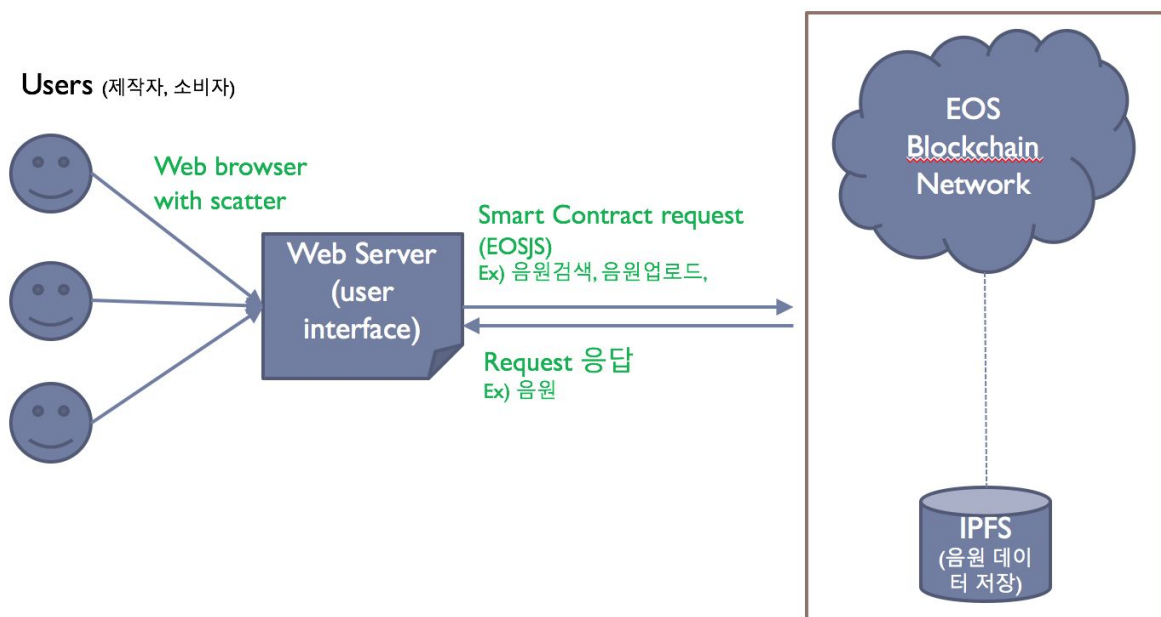
EOS DApp을 개발하기 위해서 먼저 Blockchain 생태계에 대한 전반적인 이해가 필요하였다. 그래서 오픈소스로 제공된 Ethereum의 소스코드와 white paper(백서)를 통해 공부가 진행되었다. 그 후 Ethereum git에서 제공되는 Decentralized App Tutorial을 통해 DApp의 기본적인 구조를 파악하였고, 간단한 Decentralized App을 Ethereum platform에서 개발해보았다.

그 후 EOS DApp 개발을 본격적으로 시작하였다. 그러나 EOS는 아직 개발이 완성되지 않은 상태에 있어, Main Network 뿐 아니라 Test Network도 존재하지 않는다. 그래서 우리는 먼저 EOS Local Test Network를 build하여 실행시켜 직접 Test Network를 구성하였고, 그 위에서 개발을 진행하였다.

결국 최종적으로는 자바 스크립트를 이용한 Web page를 제작하였다. Smart Contract는 C++을 이용하여 작성할 수 있었고, EOSJS, Scatter를 이용하여 웹과 블록체인 간의 통신이 가능하였다. 따라서 우리는 음원을 블록체인 망에 올리고 찾아올 수 있었다. 그리고 음원 저장소로 사용될 IPFS와 웹을 연결하였고, 결국 웹 브라우저를 통하여 사용자가 사용할 수 있는 블록체인 서비스를 만들었다.

6. Project Architecture

A. Architecture Diagram



B. Architecture Description

User : 유저 인터페이스로 제공되는 웹페이지를 통해 서비스를 요청한다. 제작자의 경우 음원의 업로드를 요청할 수 있으며 소비자의 경우 음원 검색 및 재생을 요청할 수 있다.

Scatter : Web browser extension 으로 제공된다. EOS network에 등록된 key pair와 account 정보를 identity 라는 개념으로 보관하는 wallet application이다. 또한 프론트엔드 로직을 통해 eosjs 와 연동하여 EOS network에 transaction을 등록할 수 있으며, identity를 통해 signature를 남길 수 있게 한다.

Web Server : 웹페이지를 제공하여 유저로부터 서비스 요청을 받아들이고 받아들이 요청에 따라 EOS 블록체인에 요청을 보낸다. 블록체인에 요청은 EOSJS API 및 IPFS API를 통해 보내게 된다. 블록체인으로부터 응답이 오면 일련의 처리과정을 거쳐 유저에게 보여주게 된다.

EOS blockchain : 서비스 DAPP의 smart contract 코드가 등록되어 있어 실질적인 application logic의 구동을 담당하게 된다. 이때 EOS 블록체인에서 제공하는 자체 data storage가 없으므로 EOS 블록체인에 음원 데이터들을 직접 저장하지 않고 IPFS를 통해 저장 한 후 그 hash address를 보관하는 방식으로 음원 데이터들을 저장한다.

IPFS : Blockchain 서비스를 위한 별도의 data storage 서비스로, 실질적인 음원 데이터를 저장하게 된다. 음원 데이터를 저장하면 고유한 hash value를 반환하는데, 이를 EOS blockchain에 저장한다.

7. Implementation Spec

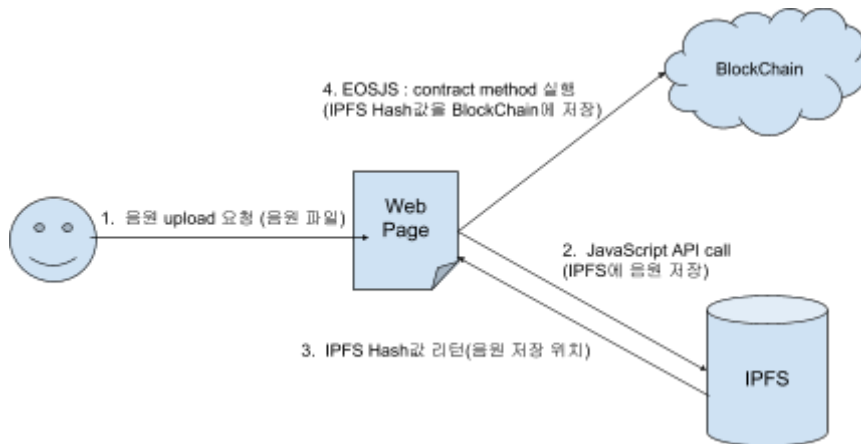
A. Input/Output Interface

웹페이지로 User Interface를 제공

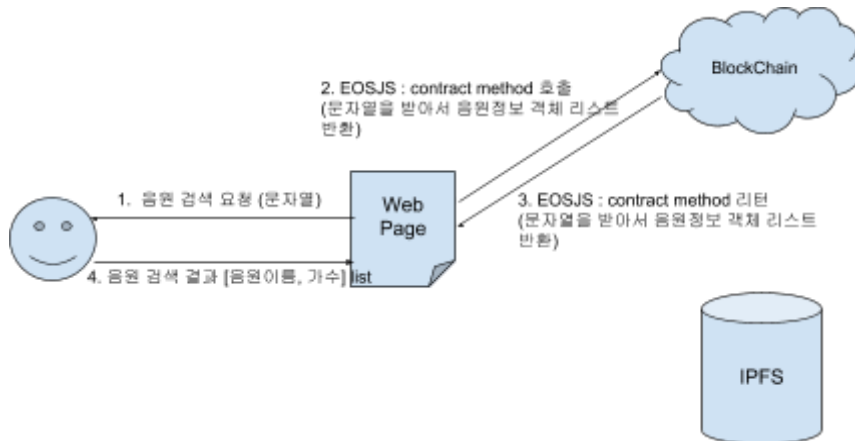
Feature	Input	Output
음원 업로드	음원 파일(10MB 이하, mp3)	결과 메시지(성공 or 실패)
음원 검색	음원 이름 부분문자열	[가수, 음원 이름, IPFS hash] list
음원 재생	[가수, 음원 이름, IPFS hash] 선택	음원 재생 (미디어 플레이어)

B. Inter Module Communication Interface

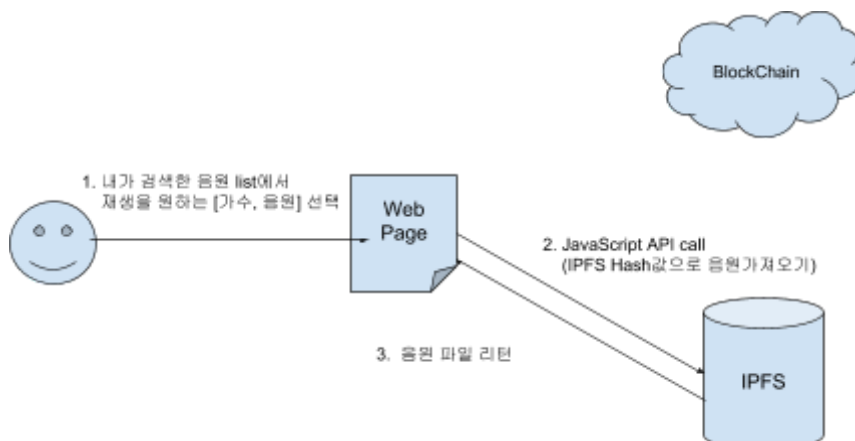
- 음원 업로드



- 음원 검색



- 음원 재생



C. Modules

BlockChain : EOSIO dawn 4.2 build Local Testnet

Data Storage : IPFS

8. Solution

A. Implementations Details

- Smart Contract

EOS의 smart contract는 C++ 언어로 구현된다. 우리는 전체 서비스 단계 중 Smart Contract level에서 수행되어야 하는 기능들을 C++ 로 코딩한 후 우리의 Local Testnet에 배포하였다.

전체 서비스를 제공하기 위해 Smart Contract level 에서 수행되어야 하는 기능들은 다음과 같다.

1) [IPFS hash, 음원정보] 테이블 유지

EOS 에서는 C++ boost 라이브러리의 멀티 인덱스 컨테이너를 제공한다. 개발자는 이를 이용하여 smart contract 마다 개별적인 table을 가지게 할 수 있다. 우리는 이를 활용하여 IPFS hash value를 key로 하고, 음원정보 객체를 value로 하는 멀티 인덱스 table을 구성하였다.

```
/**
 * Tables
 *****/

typedef multi_index<N(musics), musics> TABLE_Musics;
```

음원정보 객체는 C++ struct 로써 아래와 같이 IPFS hash, music name, singer, account name을 멤버로 가지도록 구현하였다.

```
// @abi table musics i64
struct musics {
    hash ipfs_hash; // use this as index
    string ipfs_hash_str;
    string music_name;
    string singer;
    account_name identity_account;

    hash primary_key()const { return ipfs_hash; }
```

또한 위 코드에서 알 수 있듯이 primary key의 uniqueness 를 보장할 수 있도록 IPFS hash 값을 table의 primary key로 지정해주었다.

2) 테이블에 데이터 삽입

우리는 테이블에 데이터를 삽입하기 위해서 addmusic 이라는 함수를 정의해주었다. addmusic은 void 타입 함수로, 음원정보 객체의 멤버가 되는 IPFS hash, music name, singer, account name을 인자로 받아 새로운 음원정보 객체를 구성한 후 smart contract에서 유지중인 테이블에 추가하도록 구현하였다.

```
// @abi action
void addmusic( string music_name, string singer, string ipfs_hash_str, account_name identity_account ) {
    require_auth(identity_account);
    TABLE_Musics _musics(_self, _self);

    //const char* contract_name_str = "music"; // our Dapp contract name
    //account_name contract_name = eosio::string_to_name(contract_name_str);

    hash ipfs_hash = stringToHash(ipfs_hash_str);
    auto record = _musics.find( ipfs_hash );
    if(record == _musics.end())
    {
        _musics.emplace( identity_account, [&]( auto& a ) {
            a.music_name = music_name;
            a.singer = singer;
            a.ipfs_hash = ipfs_hash;
            a.ipfs_hash_str = ipfs_hash_str;
            a.identity_account = identity_account;
        });
        eosio::print("===== music added! =====\n");
    }
}
```

위의 코드에서 볼 수 있듯이 직접 테이블에 음원정보 객체를 삽입하는 부분은 멀티인덱스 API인 emplace를 통해 이루어진다.

3) 테이블 정보 반환

테이블 정보 반환기능은 Smart contract에서 따로 함수를 정의해주지 않아도 EOSIO의 기본 API로써 제공된다. 다음과 같이 cleos에서 제공하는 명령어를 통해서도 테이블 정보를 확인할 수 있으며, eosjs의 method로도 제공된다.

```
cleos get table music music musics
```

```
{
  "rows": [{
    "ipfs_hash": 1350641643,
    "ipfs_hash_str": "QmRLn7gbBAp276887RbKNr7M8e7bHYZdcqJhMekTVnsX5B",
    "music_name": "red",
    "singer": "sasdfq",
    "identity_account": "user2"
  }, {
    "ipfs_hash": 3399406239,
    "ipfs_hash_str": "QmTWgg445ccorvTdwZhXs2yyHLqzcFWgJyXbn68iSg791S",
    "music_name": "red",
    "singer": "redred",
    "identity_account": "user1"
  }
],
  "more": false
}
```

- Web page

ReactJS 를 활용해 web page를 구현하였다. Network 등록, Account 생성, 음원 검색, 업로드, 재생 등을 각각 ReactJS의 Component로 구현했다.

1) Network 등록

Scatter에 우리의 local test network를 추가하기 위해서 간단히 버튼 으로 구성된 AddNetworkComponent 컴포넌트를 새롭게 정의하였으며 컴포넌트 내에서 scatter에서 제공하는 suggestNetwork 메소드를 호출하여 scatter에 local test network를 추가할 수 있도록 하였다.

```
this.props.statefunction.scatter.suggestNetwork(this.network).then(()=>
```

2) Account 생성

Local testnet에 새로운 account를 추가하기 위해서 account name과 public key를 입력받아 submit 하는 CreateAccountComponent 컴포넌트를 새롭게 정의하였으며 컴포넌트 내에서 eosjs에서 제공하는 newaccount 메소드를 호출하여 local testnet에 새로운 account를 추가할 수 있도록 하였다.

```
eos.newaccount({
  creator: stakerName,
  name: this.state.inputAccountName,
  owner: this.state.inputPublicKey,
  active: this.state.inputPublicKey,
})
```

3) 음원 업로드

우리의 Dapp service에 새로운 음원을 업로드 하기 위해서 음원 제목과 가수, 파일을 입력받아 submit 하는 AddMusicComponent 컴포넌트를 새롭게 정의하였다. 컴포넌트 내부에서는 우선 IPFS 라이브러리의 add method를 이용하여 IPFS에 파일을 업로드 하여 IPFS hash value를 받아온다.

```
// add file to ipfs, and call addMusicToEos with ipfs hash
await ipfs.add(this.state.buffer, (err, ipfsHash) => {
```

그 후 입력받은 음원제목, 가수와 반환받은 IPFS hash, scatter에서 가져온 account를 인자로 하여 scatter(eosjs)를 통해 등록해놓은 smart contract의 action을 호출하게 된다.

```
this.network = { blockchain: 'eos', host: LOCAL_NETWORK_HOST, port: LOCAL_NETWORK_PORT, };
this.props.statefunction.scatter.getIdentity({ accounts: [this.network] }).then(id => {
  console.log(id.accounts);
  const account = id.accounts.find(account => account.blockchain === 'eos');

  const options = {
    authorization: [
      `${account.name}@${account.authority}`,
    ],
  };
  console.log(this.props.statefunction.scatter.identity);

  this.props.statefunction.scatter_eos.contract(CONTRACT_NAME,).then(contract => {
    console.log(this.state.ipfsHash);
    console.log(contract);

    contract.addmusic(this.state.musicName, this.state.singer, this.state.ipfsHash, account.name, options).
```

4) 음원 검색

우리의 Dapp service에서 음원을 검색하기 위해서 음원 제목을 입력받아 submit 하는 SearchMusicComponent 컴포넌트를 새롭게 정의하였다. 컴포넌트 내부에서는 우선 eosjs의 getTableRows 메소드를 이용하여 smart contract에서 정의한 table을 통째로 가져오도록 하였다.

```
eos.getTableRows({
  "json": true,
  "scope": CONTRACT_NAME,
  "code": CONTRACT_NAME,
  "table": TABLE_NAME,
  "limit": 500
}).then(result => {
```

그 후 filtering을 통해 입력한 검색어에 맞게 검색 결과를 리스트로 보여주도록 하였다.

```
    filteredTable = result.rows.filter((val) => { return val.music_name.search(new RegExp(this.state.queryMusicName, "i")) !== -1; });
```

5) 음원 재생

음원을 검색하면 검색 결과가 리스트 형식으로 나타난다. 검색결과 리스트에서 재생 하고자하는 음원을 클릭하면 리스트 요소가 가지고있던 IPFS hash value를 가지고 IPFS라이브러리의 cat 메소드를 이용하여 음원파일을 받아오도록 하였다.

```
await ipfs.cat(data.value, (err, file) => {
```

그 후 기본 음원 재생 플레이어를 이용하여 음원이 자동으로 재생되도록 하였다.

B. Implementations Issues

- Version Issue (Signature format issue)

프로젝트를 진행함에 있어서 가장 큰 Issue는 프로젝트 기간동안 EOSIO의 개발이 완료되지 않은 상태라는 것이었다. EOSIO가 개발됨에 따라 버전별로 조금씩 차이가 발생하게 되었고 이는 또다시 Scatter 및 eosjs의 로직과의 차이를 발생 시켰다. 구동과 큰 관련이 없는 사소한 업데이트의 경우에는 크게 문제가 되지 않았지만 구동에 직접적인 영향을 주는 업데이트의 경우에는 얼마전까지만 해도 돌아가던 코드가 이유도 모르게 돌아가지 않는 등의 문제가 생겼다. Scatter와 eosjs는 EOSIO의 업데이트를 뒤따라가는 입장이기 때문에 해결방법을 찾기도 쉽지 않았다.

구현에 있어서 가장 문제가 되었던것은 version mismatch issue 중에서도 signature와 관련된 부분이었는데 EOSIO dawn 3.0 에서 EOSIO dawn 4.0으로 업데이트가 되면서 signature 양식이 바뀌게 되었지만 당시에는 직접 commit 내용을 살펴보지 않고는 이를 쉽게 알 수 없었다. Git hub issue등을 통해 해결방안을 찾아보려 했지만 그 또한 여의치 않았고 결국 잠시 개발을 중단하였다가 다시금 버전이 맞춰진 후에야 개발을 재개할 수 있었다.

- Data Storage

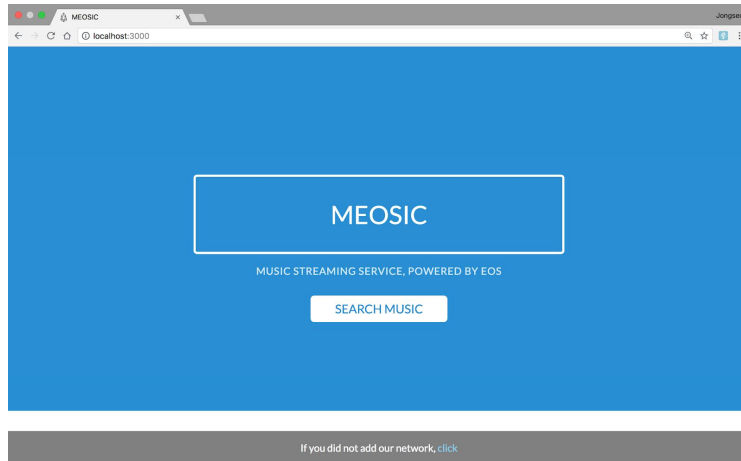
EOS blockchain은 별도로 파일을 저장할 수 있는 data storage를 제공하지 않는다. 그래서 IPFS를 이용하여 블록체인을 통해 음원파일에 접근하는 방법을 사용했다. IPFS를 통해 음원 파일은 유일한 hash value를 가지게 되고, 음원 파일은 IPFS 시스템에 연결된 storage에 저장된다. 그리고 hash value를 블록체인에 저장하여, 블록체인으로부터 hash value를 찾고, 다시 IPFS를 통해 음원을 찾는 방식으로 data storage를 사용하였다.

9. Results

아래는 우리가 구현한 DApp(이하 MEOSIC:가제)을 통해 음원을 등록하고, 실행하는 과정의 예시이다.

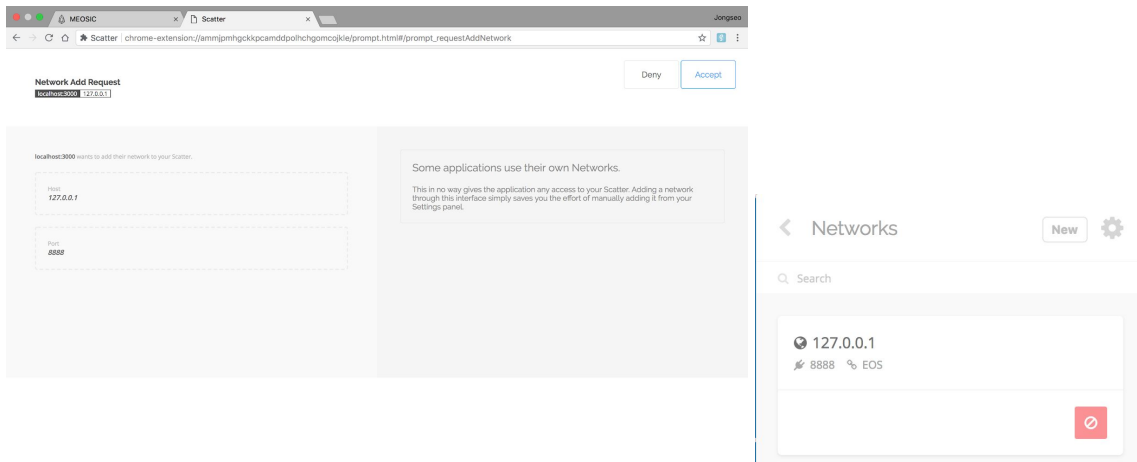
A. Experiments

MEOSIC을 실행하기 위해 웹사이트로 이동한다. 단, 현재 MEOSIC은 EOS blockchain의 wallet 관리 프로그램인 scatter를 기반으로 구현되었기 때문에, chrome에 scatter를 설치하여 가입한 뒤에 실행해야 한다.



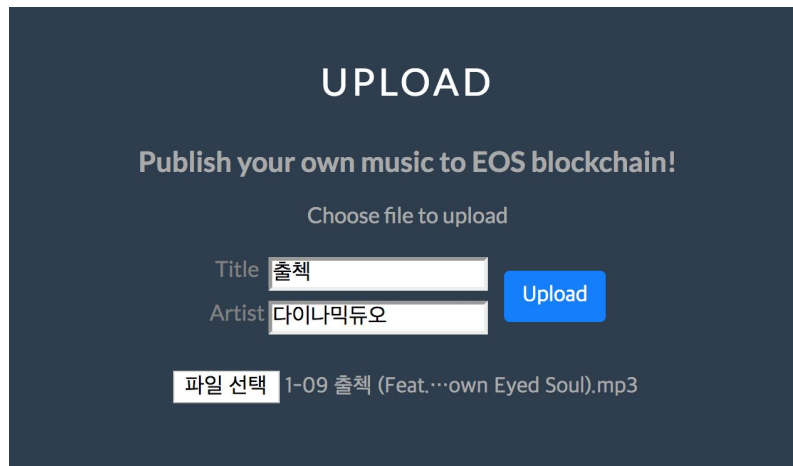
[MEOSIC의 홈 화면]

최초로 접속했을 경우, MEOSIC의 컨트랙트가 등록된 블록체인의 네트워크를 scatter에 추가해야 한다. 아래의 click버튼을 누르면, scatter에서 network를 추가하는 화면으로 이동하게 된다.

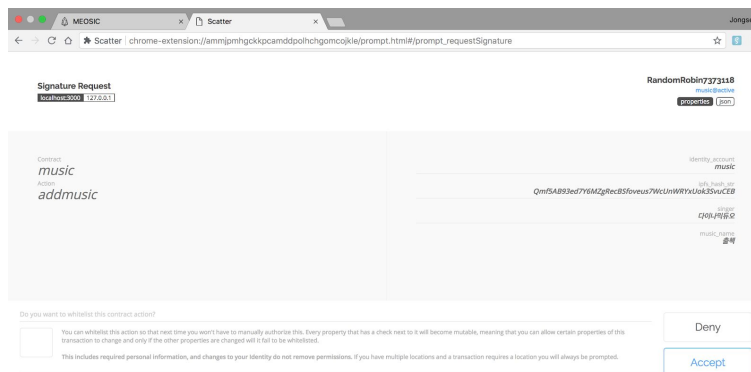


위 화면에서 Accept를 누르면 오른쪽 그림과 같이 scatter의 Networks 목록에 해당 Network가 추가되고, 해당 blockchain에 contract를 실행하고 chain에 transaction을 기록할 수 있는 상태가 된다.

이제 음원을 blockchain에 등록해보자.



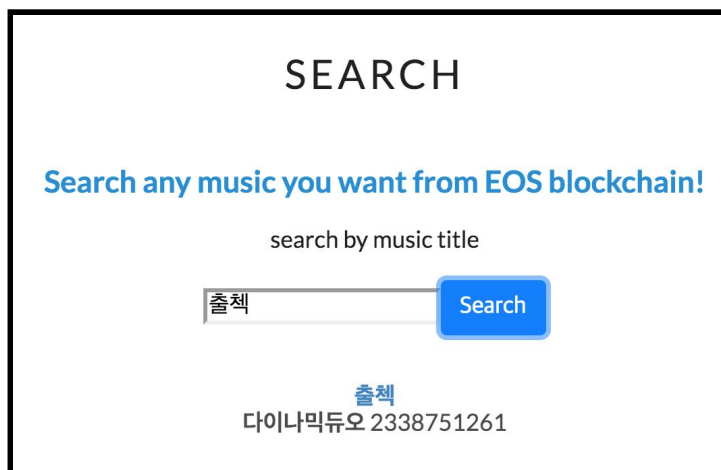
위와 같이 Upload 화면에서 음원 파일을 첨부하고 음원 정보를 기록한 뒤, Upload버튼을 누른다.



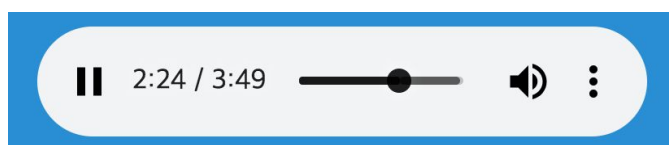
잠시 후 IPFS를 통해 음원파일의 hash value가 생성되면, hash value를 blockchain에 쓰기 위해 사용자의 권한을 묻는 scatter창이 나타난다. 이때 Accept버튼을 누르면 아래와 같이 성공했다는 메시지와 함께, hash value를 입력한 transaction의 id, 그리고 음원 파일의 hash value가 출력된다.

```
Upload success!
TXID = 35570e20f80624803f2b224b699f776ffe3364b8e27d9ee91623602dadbeeb43
IPFS hash = Qmf5AB93ed7Y6MZgRecBSfoveus7WcUnWRYxUok3SvuCEB
```

이제 등록된 음원을 검색하고 재생해보자. 아래와 같이 Search 화면에서 찾고자 하는 음원의 제목을 입력하고 버튼을 누른다. 단, 검색어와 음원을 등록할때 입력했던 음원의 제목이 정확히 일치해야만 검색결과가 나타난다.



검색은 blockchain에 기록하는 동작을 하지 않기 때문에, 사용자의 권한을 요구하지 않고, 따라서 scatter화면이 나타나지 않는다. 잠시 기다리면, 아래와 같이 음악 플레이어기가 나타나고, 음악을 재생할 수 있다.



B. Result Analysis and Discussion

본 Dapp 서비스는 EOSIO의 독자적인 Data Storage를 사용한것이 아니라 외부의 IPFS Data Storage를 사용한것이므로 IPFS에 파일을 업로드하고 다운로드 받기위해 긴 시간이 소요된다. 파일을 서비스에 업로드하고 다운로드하는데 긴 시간이 걸린다는것은 음원 스트리밍 서비스로선 큰 결점이 될 수 있다. 이번 프로젝트에서는 서비스의 상용화에는 초점을 두지 않았으나 만일 이를 상용 서비스로 제공하고자 한다면 업로드 및 다운로드 속도 개선이 반드시 고려되어야 할것으로 보인다.

만일 앞으로의 업데이트에서 EOSIO에 독자적인 Data Storage가 추가 되거나 IPFS를 대체할 수 있는 더욱 빠른 Data Storage를 사용할 수 있다면 느린 속도 문제를 해결할 수 있는 방안이 될 수 있을것이다.

또한 이번 프로젝트에서는 고려하지 않았으나 서비스의 상용화를 고려한다면 결제 시스템 역시 반드시 논의되어야 할 문제이다. 음원 등록자는 음원을 업로드할 때 서비스 제공자에게 일정량의 토큰을 지불하고, 일반 사용자는 음원을 스트리밍 할때 음원 등록자에게 일정량의 토큰을 지불하는 등의 수익구조가 가능할것이다. 내부적으로는 음원 등록 및 재생 시 transfer transaction을 발생시키도록 한다면 어렵지 않게 구현이 가능할것으로 보인다.

10. Division & Assignment of Work

항목	담당자
Blockchain study	김신휘, 유근국, 이종서
Ethereum DApp Study	김신휘, 유근국, 이종서
IPFS study	김신휘, 유근국, 이종서
EOS study	김신휘, 유근국, 이종서
Smart contract : Search	김신휘, 이종서
Smart contract : Upload	김신휘
Web page design	이종서, 유근국
EOSJS with smart contract	유근국, 김신휘
EOSJS with IPFS	유근국

11. Conclusion

차세대 blockchain 플랫폼으로 각광받는 EOS를 통해 직접 DApp을 개발했다. smart contract를 개발하기 위한 환경이 잘 갖춰져 있었다. eoscpp를 통해 작성한 contract를 컴파일하고, eosjs를 통해 web application에서 contract를 호출하는 과정이 매끄럽게 이루어졌다. BlockChain에 데이터를 입력 및 출력하는 과정도 일반적인 web application에 비해 큰 차이를 느낄 수 없을 정도로 빠른 속도로 실행됨을 확인할 수 있었다.

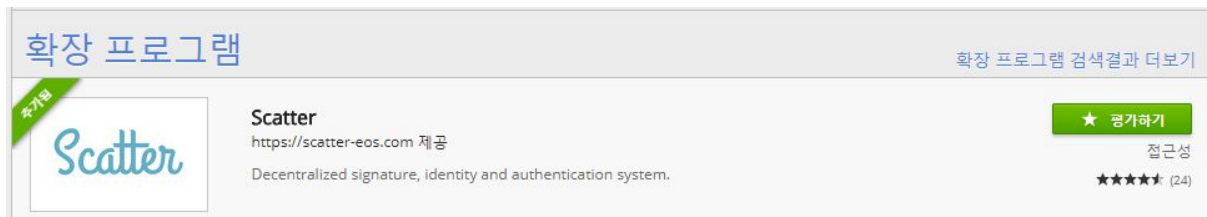
하지만 앞서 언급했듯 본 DApp의 경우 음원의 검색에 비해 음원의 등록과 실행의 속도가 아주 느리다. 이는 blockchain과 storage의 교두보 역할을 하는 IPFS의 파일 조회 속도가 느린 것에 의한 병목 현상이다. IPFS를 통한 데이터 전송의 속도가 개선된다면, 보다 다양한 환경에서 IPFS의 storage를 활용한 DApp이 개발, 활용될 수 있을 것이다.

그럼에도 불구하고 blockchain 플랫폼을 활용한 DApp이 현재의 중앙화된 App들을 모두 대체할 수 있을 것으로 보이지 않는다. 정보의 투명성과 안정성에 비해 느린 속도와 다소 번거로운 사용 환경 등이 해결해야 할 숙제로 남아있기 때문이다. 하지만 아직 연구가 시작된지 얼마 되지 않은 기술인 만큼, 아직도 발전할 수 있는 여지가 많이 남아있다. 본 프로젝트에서 DApp을 개발했던 경험을 통해 blockchain의 발전에 보탬이 될 수 있다면 더 없이 좋을 것이다.

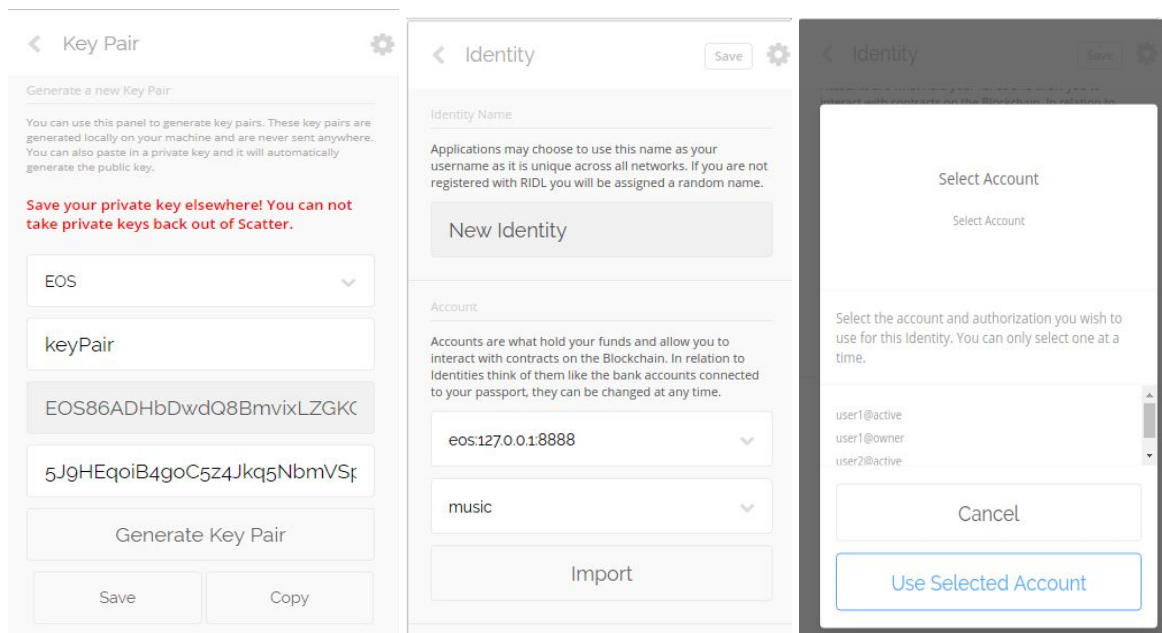
◆ [Appendix] User Manual

준비사항

1. Web browser extension Scatter 설치(chrome, firefox)



2. Scatter key pair, account, identity 등록



음원 업로드, 음원 검색, 음원 재생 사용법

[9-A 참고]