



NFTicket

NFTicket

시스템 기술서

I.내용

II. 개요	2
1. 프로젝트 개요	2
2. 프로젝트 배경	2
3. 프로젝트 핵심 기술	2
III. 핵심 기술 소개	4
1. SOLIDITY 를 통한 DAPP 및 SMART CONTRACT 개발	5
2. IPFS 를 통한 파일 분산 저장	6
3. ERC-20 및 ERC-721 토큰 기반 티켓 예매 시스템	7
4. TRUFFLE 을 통한 SOLIDITY 컴파일, CONTRACT 테스트 및 배포	7
5. EXPRESS.JS 기반 REST API 서버 구축	8
6. DOCKER 를 통한 서버 이전 용이성 확보	9
7. PRISMA 기반 DBMS 비의존적 WAS 서버 구성	9
8. 암호화폐 지갑을 활용한 서명	11
9. MVC 모델에 기반한 REST API 서버 구축	11
10. UNITY - WebGL 을 통한 메타버스	12
11. 확장성을 고려한 UNITY 설계	13
12. PHOTON UNITY NETWORK	13
13. UNITY 커뮤니티	14

II. 개요

1. 프로젝트 개요

특화 프로젝트는 블록체인 기술을 이용하여 대체 불가능 토큰을 발행하고, 이를 이용한 서비스를 만드는 것이 목표입니다. 본 기술서에서는 이번 프로젝트에서 그 목표 달성을 위해서 어떤 상황과 조건이 주어졌고, 프로젝트를 진행하면서 이를 어떻게 달성해 나갔으며 무엇을 보고 느꼈는지에 대해 정리해보고자 합니다.

2. 프로젝트 배경

공연이 끝난 뒤, 누군가와 의 혹은 그 순간 나만의 소중한 추억이 담긴 티켓이 사라지는 것이 아쉽지 않으셨나요? 모아두고 싶어도 아름답지 않아 결국 사라지는 추억들.

수집가처럼 앨범을 준비하고 일상을 기록하기는 힘들지만 마음만큼은, 그 순간의 추억을 간직하고 싶은 당신을 위해 준비했습니다.

3. 프로젝트 핵심 기술

이번 프로젝트를 하면서 사용한 핵심 기술들입니다.

- **Solidity**

Dapp 이나 Smart Contract 를 작성 및 구현하기 위해 사용되는 계약 지향 프로그래밍 언어입니다. 솔리디티를 통해 EVM 에서 작동 가능한 바이트코드를 작성하고 이를 블록체인 네트워크 상에 배포함으로써 사용자에게 서비스 가능한 Dapp 을 배포하였습니다.

- **ERC-20, ERC-721 Token**

이번 블록체인 프로젝트의 핵심은 ERC-20 및 ERC-721 토큰입니다. NFTicket 은 ERC-721 토큰 기반 티켓을 발행하고, ERC-20 토큰을 통해 지불 거래를 제공하는 서비스입니다.

- **IPFS**

IPFS 는 탈중앙화, 분산화된 스토리지 환경을 추구하는 블록체인 스토리지 서비스입니다. IPFS 는 데이터의 내용을 해시 값으로 변환하여 해시데이틀을 생성하여 파일에 접근할 수 있으며, 이와 같은

이유로 업로드가 완료되기 전에도 파일에 접근할 수 있는 End-point 를 얻을 수 있다는 장점이 있습니다.

- **Truffle**

Solidity 로 작성된 Smart Contract 를 손쉽게 컴파일 및 테스트, 그리고 블록체인 네트워크 상에 배포할 목적으로 만들어진 블록체인 프레임워크입니다. Truffle 을 통해 Smart Contract 동작을 테스트하고 Ganache 등 로컬 네트워크와 배포 네트워크에 원활하게 배포할 수 있었습니다.

- **Docker**

위에서 설명한 IPFS 를 포함해 MariaDB, WAS 등 백엔드 서버 전반에 대하여 Dockerize 를 통해 컨테이너 화하였고, 이를 통해 별도의 환경 구성 없이 백엔드 서버를 배포할 수 있도록 하였습니다.

- **Express.js**

Node.js 의 대표적인 웹 애플리케이션 프레임워크인 Express.js 를 사용하였습니다. 이를 통해 필요한 CRUD API 를 제공하는 API 서버 구축을 진행하였습니다.

- **Prisma**

Node.js 의 대표적인 ORM 라이브러리인 Sequelize 를 대체할 데이터베이스 프레임워크로, DB 의존성이 매우 낮고, 높은 생산성을 가진 특징을 가지고 있어 가벼운 백엔드 서버 구축에 적합하다고 생각하여 채택하였습니다.

- **Web3.0**

분산된 웹으로서 Web3.0 을 활용하였습니다. 핵심적인 로직은 스마트 컨트랙트로 편의와 속도가 필요한 로직만을 백에 요청하는 식으로 탈중앙화를 구현하였습니다.

- **React**

프론트의 javascript 라이브러리로 react 를 활용하였습니다.

- **Redux**

전역 상태인 store 을 관리하기 위해 redux 를 사용하여 action 과 reducer 을 구축하였습니다.

- **MUI**

React 의 UI 라이브러리로 유명한 Material UI와 styled component 를 필요한 상황에 맞게 활용하여 디자인하였습니다.

- **TOAST UI Image editor**

NHN 에서 제공하는 Fabric.js 기반 이미지 에디터로 react 환경에서 이미지를 편집하고 File Saver 을 통해 저장할 수 있는 환경을 제공합니다.

- **Unity**

메타버스를 구축하기 위한 게임 엔진입니다. Unity 특유의 가벼움과 용량이 웹 환경에 적합하다고 생각하여 기술 스택으로써 Unity 를 채택하였습니다. 사용언어는 C# 입니다.

- **WebGL**

Web Graphic Library 로 캔버스에 3D 그래픽인 Unity 를 렌더링 하기 위한 javascript API 입니다. Send 와 JsLib 를 사용하여, react 와 Unity 간의 상호소통을 구축하였습니다.

- **Photon Network**

다양한 네트워크 환경을 제공하며, 유니티 네트워크에서 자주 사용되는 대중적인 네트워크 솔루션입니다. 동기화와 메치메이킹, 방 권한과 Websocket 등을 통한 채팅 기능 제공 등을 제공합니다.

- **Let's Encrypt**

SSL 무료 인증서를 받아 웹 서버에 https 를 적용하고, 통신에 보안을 적용하였습니다.

- **MySQL MariaDB**

관계형 DB 인 MariaDB 를 메인 DB 로 사용하였습니다.

- **AWS (EC2)**

EC2 인스턴스에서 프론트엔드 및 백엔드 서버를 구축하고, RDBMS 서버를 이용하여 관계형 DB 를 적재하였습니다.

III. 핵심 기술 소개

1. Solidity 를 통한 Dapp 및 Smart Contract 개발

Solidity 는 이더리움 프로젝트의 Solidity 팀에서 만든, 블록체인 네트워크를 통해 배포할 Smart Contract 및 Dapp 의 바이트코드를 손쉽게 생성하기 위해 고안된 블록체인 프레임 워크입니다. Solidity 를 통해 Contract 와 Interface 를 구현하고 상속을 통해 그림 1 과 같이 OOP 와 유사한 형태의 계약 지향 프로그래밍 형태의 개발을 수행할 수 있었습니다.

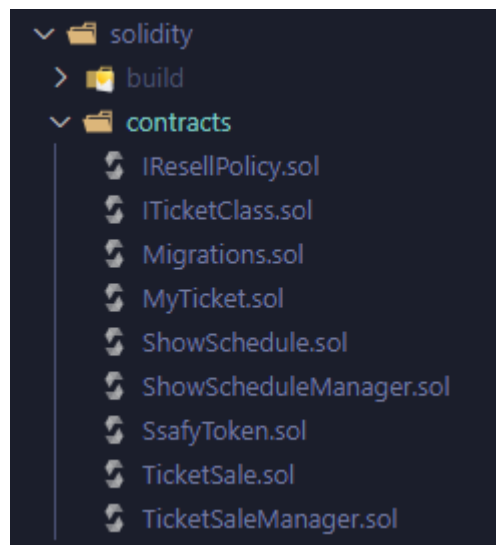


그림 1. 공연 관리, 티켓 생성 및 매매를 위한 계층적 계약 및 인터페이스 구성

그림 2 와 같이 메소드 실행 시, 메소드 호출 계정(지갑)에서 메소드가 성공적으로 수행되었는지 알 수 있도록 이벤트를 적극적으로 활용하였고, 이를 통해 트랜잭션 수행에 대한 결과를 원활하게 가져올 수 있었습니다.

또한, 인터페이스를 적극적으로 활용하여 계약 간 메소드 호출 시, 구조체 공유 및 최대 인수 전달 개수가 제한되는 문제를 해결하였습니다.

```
contract ShowSchedule is Ownable, IResellPolicy, ITicketClass {
    using Counters for Counters.Counter;

    event TicketRegistered(uint256 indexed ticketId, address from, uint256 classId, uint256 seatIndex);
    event TicketRevoked(uint256 indexed ticketId, address to, uint256 classId, uint256 seatIndex);
    event TicketRefunded(uint256 indexed ticketId, address to, uint256 classId, uint256 seatIndex);
    event Withdrawal(address indexed to, uint256 amount);
    event Cancelled();
}
```

그림 2. Contract 내 메소드의 동작 정보를 프론트엔드에 제공하도록 이벤트를 적극 활용

ERC-20 및 ERC-721 Contract 를 활용하여 그림 3 과 같이 공연 관리 및 NFT 기반 티켓 예매, 티켓 거래(구매, 판매) 등이 정상적으로 이루어 질 수 있도록 하였습니다. 가스비가 없는 프라이빗 네트워크의 특성을 최대한 활용하여, 계층적이고 체계적으로 모든 핵심 서비스가 Smart Contract 를 통해 Middle-man 없이 탈중앙화된 서비스를 제공하는 것을 프로젝트의 방향으로 잡고 프로젝트에 임했습니다.

```
// 어떤 ERC-20 Token도 화폐로 사용할 수 있도록 한 IERC20 Interface 객체 (가베포된 CA를 통해 생성)
IERC20 private _currencyContract;
// 발급된 티켓 정보가 저장된 NFT Contract 객체 (가베포된 CA를 통해 생성)
MyTicket private _ticketContract;
```

그림 3. ERC20 토큰 및 MyTicket(ERC-721 기반 Contract)과의 Contract 간 호출 구현

또한, 그림 4 와 같이 블록체인에 접근하는 프론트엔드 개발자와의 소통을 원활히 할 수 있도록 자세한 주석을 달아 불필요한 소통으로 발생하는 개발 시간 낭비를 줄여 단기간 최대의 성과를 낼 수 있도록 노력하였습니다.

```
/*
 * registerTicket
 * 임의의 classId A, 임의의 seatIndex B, 임의의 ticketId X에 대해
 * 요청자는 등록 비용을 지불하고 A등급 B번째 좌석에 티켓 ID X를 등록
 * Contract의 발급 티켓 수와 등급별 발급 티켓 수가 1씩 증가
 */
 * @ param uint256 classId 등급 ID
 * @ param uint256 seatIndex 좌석 Index
 * @ param uint256 ticketId 티켓 ID
 * @ return None
 * @ exception 티켓 발급 개수가 최대 발급 개수를 넘지 않아야함
 * @ exception 등급별 티켓 발급 개수가 등급별 최대 발급 개수를 넘지 않아야함
 * @ exception 공연이 취소 상태가 아니어야 함
 * @ exception 공연이 시작되기 전 이어야 함 (공연 시작 시간 > 현재 시간)
 * @ exception A등급 B번째 좌석이 비어 있어야 함
 * @ exception 티켓의 classId와 등록할 classId가 일치해야 함
 * @ exception msg.sender(요청자)에게 등록 비용 이상의 잔고가 있어야 함
 */
function registerTicket(uint256 classId, uint256 seatIndex, uint256 ticketId) public payable notFull notClassFull(ticketId) notCanceled notStarted {
```

그림 4. 메소드 및 호출 시 발생할 수 있는 상황에 대한 정보 제공

2. IPFS 를 통한 파일 분산 저장

IPFS 는 기존 인터넷 서비스보다 분산화 된 환경을 추구하며, 토렌트와 블록체인 같은 성격을 가진다는 특성이 있습니다. IPFS 는 데이터의 내용을 해시 값으로 변환하여 해시테이블을 생성합니다. 해시테이블은 정보를 키와 값의 쌍(key/value pairs)으로 저장하는데, 전 세계 수많은 분산화 된 노드들이 해당 정보를 저장함으로써 데이터를 단일 장애 지점없이 안전하게 저장할 수 있습니다. IPFS 의 해시 값을 이용하여 데이터의 변조를 방지하고 이를 블록체인에 등록함으로써, 높은 신뢰성을 갖는 탈중앙화 서비스를 제작할 수 있었습니다.

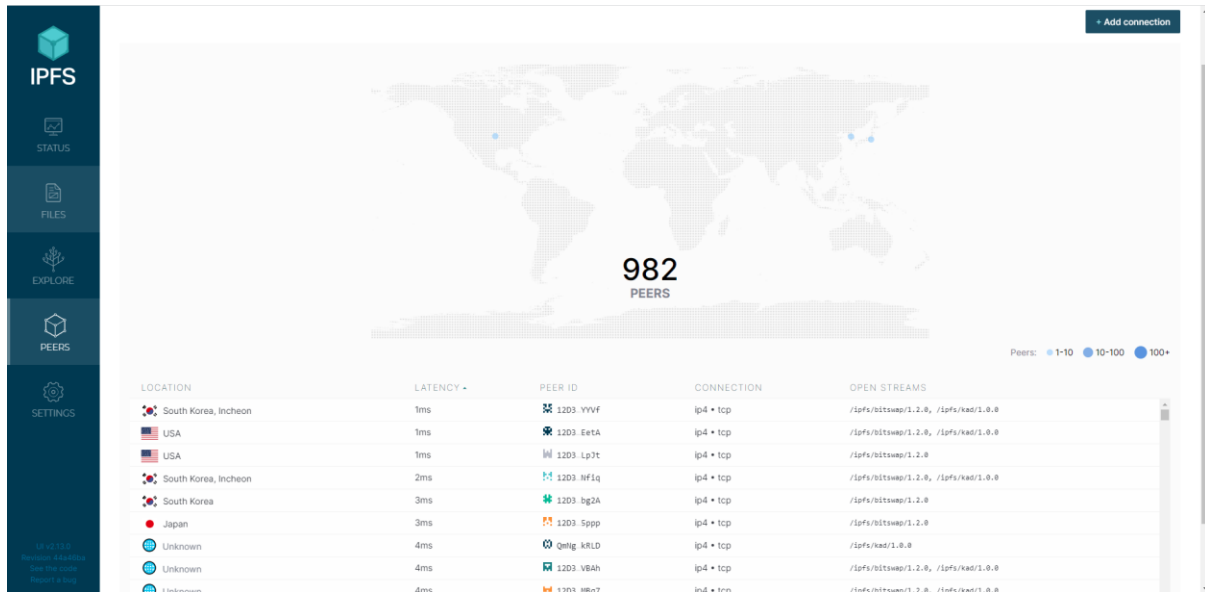


그림 5. IPFS 관리용 콘솔 Web UI 화면

3. ERC-20 및 ERC-721 토큰 기반 티켓 예매 시스템

NFTicket 프로젝트는 ERC-721 토큰 기반 티켓을 발행하고, ERC-20 토큰을 통해 지불 거래를 수행하는 Contract 를 만들어 이에 대한 CRUD 를 수행하는 것이 핵심 기술입니다. ERC-721 Contract 를 통해 생성된 ERC-721 토큰에는 각각 IPFS 상에 업로드 된 메타데이터의 해시 값이 기록되고 연결되어 AWS 등 클라우드 스토리지에도 기대지 않는 완전한 탈중앙화 NFT 를 생성하고 있습니다.

각각의 티켓은 대체 불가능한 고유성을 가지며, 이를 위해 NFT 기반 티켓 Contract(ERC-721) 뿐만 아니라 공연 스케줄 정보 관리를 위한 Contract 및 티켓 매매 전반을 관리하기 위한 Contract 등을 생성하여 서비스 내 모든 핵심 동작이 Smart Contract 를 통해 Middle-man(중개자) 없이 이루어질 수 있도록 Dapp 을 구현하고 배포하였습니다.

4. Truffle 을 통한 Solidity 컴파일, Contract 테스트 및 배포

Truffle 은 Solidity 로 작성된 Smart Contract 를 손쉽게 컴파일 및 테스트, 그리고 블록체인 네트워크 상에 배포할 목적으로 만들어진 블록체인 프레임워크입니다. Truffle 을 통해 Smart Contract 동작을 테스트하고 Ganache 등 로컬 네트워크와 Ropsten 등 이더리움 테스트 네트워크, 최종적으로 Hyperledger 기반 프라이빗 네트워크에 원활하게 배포할 수 있었습니다.

JSON-RPC 등을 통해 블록체인 네트워크 Endpoint 에 JSON 형태의 http request 를 보내어 블록체인 Contract 를 배포하거나 메소드를 실행할 수 있습니다. 하지만, Solidity 로 작성된 소스 코드를 Solc 로 컴파일 하고 바이트 코드와 ABI 를 만든 후, 소스 코드 형태가 아닌 EVM 바이트 코드를 data 인수로 Contract 를 배포해야 하며, 특히 메소드 실행시마다 원하는 메소드 이름과 인수를 바이트코드 형태로 바꾸는 것은 매우 어려운 일입니다.

Truffle 은 위와 같은 일련의 블록체인 개발 흐름을 자동화하여 개발자의 생산성을 비약적으로 개선하는데 많은 도움이 되었습니다. 또한, 배포할 로컬 혹은 원격 네트워크 간 전환이 쉽고 여러 개의 Contract 를 동시에 배포하여 그림 6 과 같이 배포된 Contract 간 연동을 통해 원하는 동작에 대한 테스트를 수행할 수 있었습니다.

```
it("purchase TicketSale", async () => {
  const [sender, receiver] = accounts;
  console.log('sender:', sender);
  console.log('receiver:', receiver);

  const ssafyTokenInstance = await SSafyToken.deployed();
  const myTicketInstance = await MyTicket.deployed();
  const ticketSaleManagerInstance = await TicketSaleManager.deployed();

  console.log('TicketSale Purchase');
  console.log(await myTicketInstance.getShowScheduleId(1, { from: receiver }));
  const saleAddr = await ticketSaleManagerInstance.getSale(1, { from: receiver });
  console.log('saleAddr:', saleAddr);

  console.log('getApproved:', await myTicketInstance.getApproved(1));

  const saleInstance = await TicketSale.at(saleAddr);
  console.log('owner:', await saleInstance.owner());
  console.log('ownerOfTicket:', await myTicketInstance.ownerOf(1));
  console.log('balanceOf(receiver):', await ssafyTokenInstance.balanceOf(receiver));
  console.log('balanceOf(saleAddr):', await ssafyTokenInstance.balanceOf(saleAddr));

  const recvBalance = await ssafyTokenInstance.balanceOf(receiver);
  await ssafyTokenInstance.approve(saleAddr, recvBalance, { from: receiver });
  console.log(await saleInstance.purchase({ from: receiver }));
  await ssafyTokenInstance.approve(saleAddr, 0, { from: receiver });

  console.log('ownerOfTicket:', await myTicketInstance.ownerOf(1));
  console.log('balanceOf(receiver):', await ssafyTokenInstance.balanceOf(receiver));
  console.log('balanceOf(saleAddr):', await ssafyTokenInstance.balanceOf(saleAddr));
});
```

그림 6. 티켓 매매 관련 블록체인 테스트 코드 중

5. Express.js 기반 REST API 서버 구축

Node.js 서버의 대표적인 웹 애플리케이션 프레임워크인 Express.js 를 선택했습니다. Node.js 는 단일 쓰레드 이벤트 루프 기반 비동기 방식으로 동작하기 메모리의 사용량이 낮고 경량화 된 서비스를 제작할 수 있습니다. 또한 Non-blocking I/O 방식을 사용해 빠른 Disk 읽기 쓰기가 가능합니다.

이번 프로젝트에서 최대한의 탈중앙화 서비스를 지향했기 때문에, 중앙화 된 서버의 역할을 최소화할 필요가 있었고, 위의 특성이 소규모 백엔드 제작에 적합하다고 생각하여 Node.js 와 Express.js 를 사용하게 되었습니다. 또한, 부차적으로 동일한 Javascript 를 사용하여 Node.js 기반 React 프론트엔드와 협력하고 소통하는데 많은 도움이 되었습니다.

6. Docker 를 통한 서버 이전 용이성 확보

Dockerfile 과 docker-compose 를 통해 WAS 서버를 컨테이너화 하여 쉽고 빠른 배포가 가능하도록 했습니다. 또한 IPFS 노드 및 MariaDB 또한 컨테이너를 통해 관리하여, 프로젝트 종료 후 서버 이전 시의 편의성을 높였습니다.

7. Prisma 기반 DBMS 비의존적 WAS 서버 구성

Node.js 의 대표적인 ORM 라이브러리인 *Sequelize* 를 대체할 데이터베이스 프레임워크로, DB 의존성이 매우 낮고 높은 생산성을 가진 특징을 가지고 있어 가벼운 백엔드 서버 구축에 적합하다고 생각하여 채택하였습니다. Prisma 는 MySQL Workbench 를 통해 MariaDB 에 정의한 DB 를 pull 하여 그림 7 과 같이 손쉽게 Schema 를 생성 가능하며, 반대로 생성한 Schema 를 다른 DBMS 로 push 하여 테이블을 생성하는 것 또한 가능합니다. 따라서 DB 의존성이 매우 낮고, 변수가 많고 기한이 촉박한 프로젝트에 있어 높은 생산성을 보여주어 원활한 프로젝트 진행에 많은 도움이 되었습니다.

테이블 생성 시 별도의 Entity 등을 정의하지 않고 콘솔 명령을 통해 자동 생성되며, DBMS 의존적인 SQL 기반의 명령이 아닌 ORM 형태의 CRUD 메소드를 제공하여 그림 8 과 같이 쉽고 빠르게 Select, Update 등 CRUD 및 Table Join, Group, Summarize 등 고급 SQL 기능을 제공합니다. 또한 그림 9 와 같이 Pagination 또한 제공하여 검색 결과 등을 전부 반환하지 않고 페이징하여 RESTAPI 성능을 높일 수 있습니다.

```

model Address {
  address      String @id @db.VarChar(100)
  show_schedule_id Int?
  show_id      Int?
  Show         Show? @relation(fields: [show_id], references: [show_id], onDelete: NoAction, onUpdate: NoAction, map: "show_id")

  @@index([show_id], map: "show_id_idx")
}

model Authorization {
  wallet_id      String @id @db.Char(20)
  nonce          Int
  nonce_expired_at DateTime @default(now()) @db.Timestamp(0)
  signature       String? @db.Char(64)
  jwt            String? @db.Text
}

model Profile {
  wallet_id      String @id @unique(map: "wallet_id_UNIQUE") @db.Char(42)
  nickname       String @db.VarChar(10)
  description     String? @db.Text
  created_at     DateTime @default(now()) @db.Timestamp(0)
  image_uri      String? @db.VarChar(255)
  gallery        String? @db.Char(10)
}

model RandomAdjective {
  id            Int @id @unique(map: "id_UNIQUE") @default(autoincrement()) @db.UnsignedInt
  adjective     String @unique(map: "adjective_UNIQUE") @db.Char(5)
}

model RandomNoun {
  id            Int @id @unique(map: "id_UNIQUE") @default(autoincrement()) @db.UnsignedInt
  noun          String @unique(map: "noun_UNIQUE") @db.Char(5)
}

model Role {
  staff_id      Int
  show_id       Int
  occupation     String @db.VarChar(45)
  Show          Show @relation(fields: [show_id], references: [show_id], onDelete: NoAction, onUpdate: NoAction, map: "Role_ibfk_2")
  Staff         Staff @relation(fields: [staff_id], references: [staff_id], onDelete: NoAction, onUpdate: NoAction, map: "Role_ibfk_1")

  @@id([staff_id, show_id])
  @@index([show_id], map: "show_id")
}

```

그림 7. Prisma 를 통해 자동 생성되는 Table schema

```

let include = {
  Role: { select: { staff_id: true } },
  Address: { select: { address: true } }
}

const result = await prisma.Show.findUnique({
  where: {
    show_id: Number(showId)
  },
  include
})

```

그림 8. Prisma 를 통한 쉽고 빠른 Table Select 및 Join

```
if (query[ 'offset' ]) skip = Number(query[ 'offset' ])
if (query[ 'limit' ]) take = Number(query[ 'limit' ])

const result = await prisma.Show.findMany({
  where,
  include,
  orderBy,
  skip,
  take,
})
```

그림 9. Prisma 를 통한 간단한 Paging 및 Sorting

8. 암호화폐 지갑을 활용한 서명

비공개키는 블록체인 분산원장에 기록되는 거래 트랜잭션 데이터를 생성하고 서명하는 일에 사용되며 암호화폐 지갑 내부에 저장됩니다. 그에 반해, 공개키는 지갑주소 그 자체, 혹은 지갑주소를 만드는 데 사용되며, 모두에게 공개되어집니다. 개인키로 서명한 메시지를 공개키로 검증함으로써 메시지 전송 주체가 확실하다는 사실을 검증할 수 있습니다. 서버에 저장된 개인 정보를 수정하는 경우, 개인이 소유한 web3.js 와 비공개 키를 사용해 메시지에 서명을 덧붙이고, 서버는 전달받은 메시지의 서명을 web3.js 와 지갑 주소를 사용해 검증할 수 있습니다.

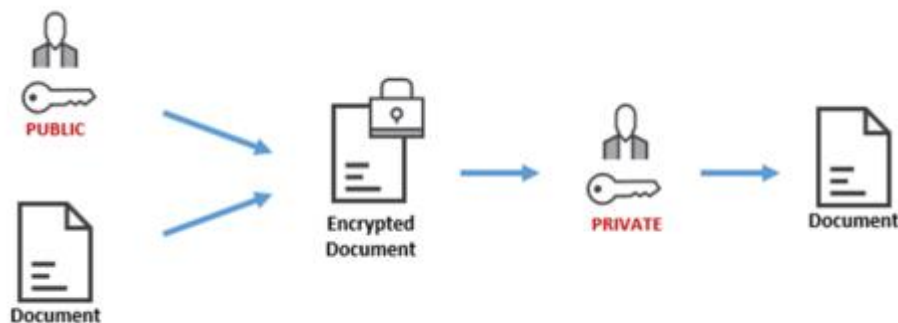


그림 10. 공개키-개인키 암호화 방식 도식화

9. MVC 모델에 기반한 REST API 서버 구축

백엔드 서버를 간소화하고 MVC 모델을 채택하였습니다. Model 은 Prisma 를 통해 Schema 를 생성하고 Controller 와 Service 로직을 나누어 필요한 Controller 에서 필요한 Service 로직을 쉽게 호출하여 사용자가 원하는 결과를 반환할 수 있도록 하였습니다.



그림 11. MVC 형태의 REST API 서버 구조

10. Unity – WebGL 을 통한 메타버스

유니티는 게임엔진으로써 가벼운 것이 특징이고 3 억명이 넘는 유저를 보유한 제페토 메타버스에서도 실제로 사용되는 엔진입니다. 이번 NFT 프로젝트에서는 어떻게 보여주는지도 사용자의 소유욕을 부추기고 거래를 활발히 하게 할 수 있는 중요한 요소라고 생각하여 메타버스로서 유니티를 기술스택에 추가하였습니다.



그림 12. 개인 NFT 를 자랑할 수 있는 전시공간

프로그램으로서의 유니티가 아닌 웹에서 제공하는 서비스의 일부로 활용해야 하기 때문에, 3D 그래픽인 Unity 를 2D 환경인 react Web 의 canvas 에 담아야 하였고, WebGL 을 활용하여 구현하였습니다.

11. 확장성을 고려한 Unity 설계

유니티를 단일 씬이 아닌 여러 씬이 유기적으로 연결되게 작성하여 방꾸미기나 여러 씬을 추가할 수 있게 확장성을 고려한 설계를 구현했습니다. 싱글톤 패턴으로 관리자 역할 오브젝트를 만들어서 Network Manager 를 통해서 각각의 씬은 네트워크 상황을 공유하고, Game Manager 을 통해서 환경변수를 포함한 변수와 설정을 공유하게 설정하였습니다.

또한, 플레이어 외에도 전시물, 배경, 액자 등의 복합 객체에도 Prefab 을 적극적으로 활용하여, 컴포넌트의 재사용성을 높였습니다.

12. Photon Unity Network

유니티에서 타 유저와 상호작용하거나 의사소통할 수 있게 하여, 공감대를 형성하기 쉬운 환경을 제공하였습니다. 그런 환경을 제공하기 위해서 유니티 네트워크에서 사용하는 대중적인 네트워크 솔루션인 PUN2.0 를 사용하여 매치메이킹부터 로비까지 일련의 흐름을

제어하고 관리하였으며, 타인의 방이나 커뮤니티에서 채팅이나 애니메이션과 같은 상호작용을 할 수 있게 설정하였습니다.



그림 13. 각종 상호작용과 사용 설명이 적힌 안내 가이드

Photon Network 는 CDN 을 통해 레이턴시를 줄이고, 가까운 네트워크 환경을 구축하는 것을 도와주지만 안타깝게도 Free 환경에서는 제한이 존재하고, 전용 클라우드 서버를 제공하지 않아 방장의 로컬서버를 사용해야 함으로 인원수에 제한이 있어 대규모의 인원을 수용할 수 있는 커뮤니티를 구현하지는 못하였습니다. (docs 권장 인원 16 명)

13. Unity 커뮤니티

나이키랜드와 아디버스 같이, 특정 NFT 를 보유한 사람만 커뮤니티의 일원으로서 활동할 수 있는 커뮤니티를 구현했습니다. 판매자를 인식할 수 있는 블록체인의 특징을 활용한 방식으로, 기존의 물건을 만들고 구매해주기를 바라는 기존의 비즈니스 모델을 뒤집어, 단순히 제품을 구입하는 것이 아니라 커뮤니티의 일원이 되게 하는 비즈니스 모델입니다.

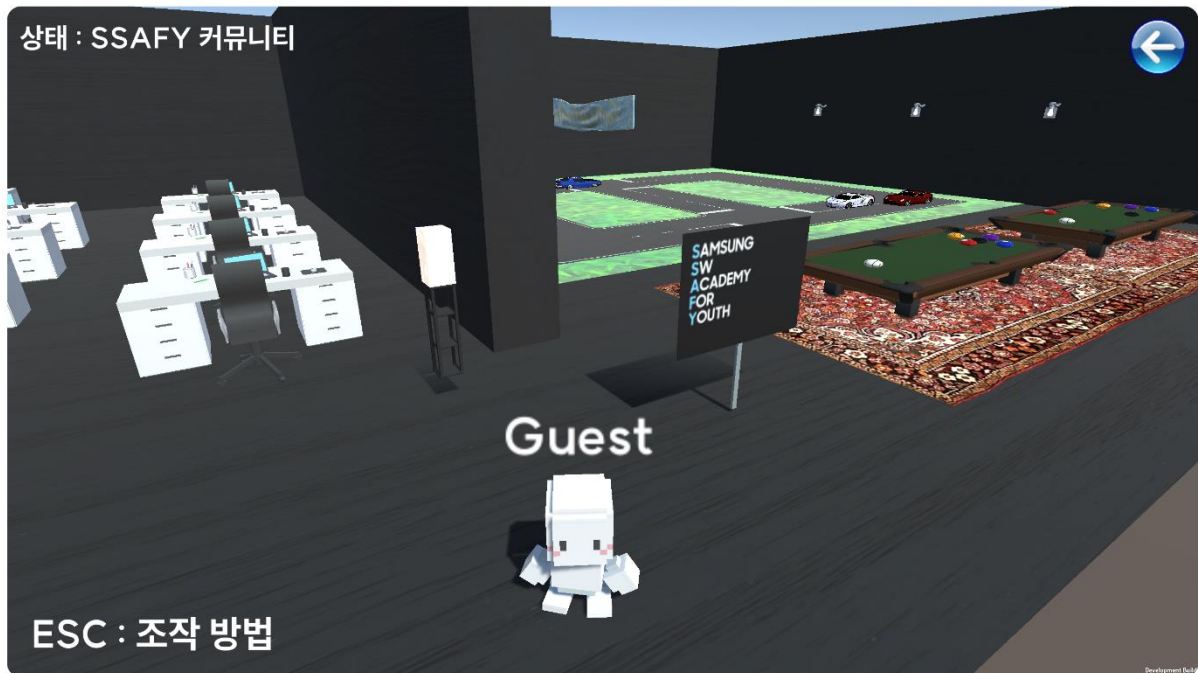


그림 14. 해당 NFT 를 보유한 사람만 들어올 수 있는 커뮤니티

추억을 공유하거나 강렬한 인상을 남긴 사물을 재현한 디오라마 등 해당 사용자의 추억을 자극함으로써 물건에 입장권으로서 더 강한 가치를 부여하는 새로운 비즈니스 모델을 구축할 수 있었습니다.